aws

ユーザーガイド

Amazon EKS



Copyright © 2026 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

Amazon EKS: ユーザーガイド

Copyright © 2026 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

Amazon の商標とトレードドレスは、Amazon 以外の製品またはサービスとの関連において、顧客に混乱を招いたり、Amazon の名誉または信用を毀損するような方法で使用することはできません。Amazon が所有していないその他のすべての商標は Amazon との提携、関連、支援関係の有無にかかわらず、それら該当する所有者の資産です。

Table of Contents

..... xxix
..... xxx
Amazon EKS とは 1
Amazon EKS: Kubernetes 管理の簡素化 1
Kubernetes を使用した構築とスケーリング: Amazon EKS 機能 2
Amazon EKS の機能 3
関連サービス 4
Amazon EKS の料金 5
一般的なユースケース 6
アーキテクチャ 8
コントロールプレーン 8
コンピューティング 9
EKS 機能 10
Kubernetes の概念 11
Kubernetes が選ばれる理由 12
クラスター 17
ワークロード 22
次のステップ 28
デプロイオプション 28
Amazon EKS デプロイオプションを理解する 28
クラウド内の Amazon EKS 29
データセンターまたはエッジ環境の Amazon EKS 30
エアギャップ環境用の Amazon EKS Anywhere 30
Amazon EKS ツール 31
セットアップ 32
次のステップ 32
AWS CLI の設定 32
アクセスキーを作成するには 33
AWS CLI を構成するには 33
セキュリティトークンを取得するには 34
ユーザー ID を確認するには 34
次のステップ 35
kubectl と eksctl のセットアップ 35
kubectl をインストールまたは更新する 35

ステップ 1: kubectl がインストールされているかどうかを確認する 36
ステップ 2: kubectl をインストールまたは更新する 36
eksctl をインストールする 49
次のステップ 49
クイックスタート 50
このチュートリアルでは 50
前提条件 50
クラスターを設定する 51
IngressClass を作成する 51
2048 ゲームサンプルアプリケーションをデプロイする 52
Amazon EKS 自動モードを使用してデータを永続化する 55
クリーンアップ 57
Amazon EKS の詳細 58
概要 58
Amazon EKS ワークショップ 58
Amazon EKS の実践的なクラスターセットアップチュートリアル 60
Amazon EKS のサンプル 61
AWS のチュートリアル 61
デベロッパーワークショップ 62
Terraform ワークショップ 62
AWS Amazon EKS トレーニング 62
はじめに 64
クラスターを作成する (EKS Auto Mode) 64
クラスターを作成する (eksctl) 65
前提条件 66
ステップ 1: Amazon EKS クラスターとノードを作成する 66
ステップ 2: Kubernetes リソースを表示する 67
ステップ 3: クラスターとノードを削除する 69
次のステップ 70
クラスターを作成する (コンソールと CLI) 70
前提条件 71
ステップ 1: Amazon EKS クラスターを作成する 72
ステップ 2: クラスターに接続する 74
ステップ 3: ノードを作成する 76
ステップ 4: リソースを表示する 78
ステップ 5: リソースを削除する 78

次のステップ 79
EKS Auto Mode 81
機能 81
自動コンポーネント 83
設定 84
責任共有モデル 85
クラスターを作成する 86
eksctl CLI 87
AWS CLI 89
マネジメントコンソール 98
既存のクラスターを有効にする 100
移行のリファレンス 101
EBS ボリュームの移行 102
ロードバランサーの移行 102
クラスターで有効にする 103
Karpenter からの移行 107
MNG から移行する 110
Fargate からの移行 111
ワークロードを実行する 119
拡張ワークロードのデプロイ 120
ロードバランサーをデプロイする 123
ステートフルワークロードをデプロイする 128
加速ワークロードのデプロイ 133
設定 140
コンピューティングとスケーリング 146
ネットワークとロードバランシング 201
Storage 237
セキュリティとコンプライアンス 249
ワークロードのスケジューリング 264
コスト最適化 269
アップグレードとライフサイクル 277
仕組み 280
管理されたインスタンス 280
ID とアクセス 286
ネットワーク 289
オブザーバビリティ 293

EKS 自動マネージドコンポーネントログ ... 293
トラブルシューティング ... 295
ノードモニタリングエージェント ... 297
AWS EC2 CLI を使用して EC2 マネージドインスタンスからコンソール出力を取得する ... 297
デバッグコンテナと kubectl CLI を使用してノードログを取得する ... 297
AWS コンソールで EKS Auto Mode に関連付けられたリソースを表示する ... 299
AWS アカウントの IAM エラーを表示します ... 300
Auto Mode ノードにスケジュールできない Pod のトラブルシューティング ... 300
クラスターに参加しないノードのトラブルシューティング ... 300
Pod 間でボリュームを共有する ... 303
コントロールプレーンログで Karpenter イベントを表示する ... 303
Auto Mode に含まれているコントローラーのトラブルシューティング ... 305
関連リソース ... 193
リリースノート ... 306
2026 年 10 月 1 日 ... 306
2026 年 9 月 14 日 ... 307
2026 年 8 月 17 日 ... 307
2026 年 8 月 12 日 ... 307
2026 年 8 月 5 日 ... 308
2026 年 7 月 27 日 ... 308
2026 年 7 月 21 日 ... 310
2026 年 6 月 30 日 ... 310
2026 年 6 月 9 日 ... 311
2026 年 6 月 4 日 ... 312
2026 年 6 月 3 日 ... 312
2026 年 5 月 5 日 ... 312
2026 年 4 月 10 日 ... 312
2026 年 4 月 2 日 ... 312
2026 年 2 月 2 日 ... 312
2025 年 12 月 19 日 ... 312
2025 年 11 月 19 日 ... 313
2025 年 11 月 19 日 ... 313
2025 年 10 月 23 日 ... 313
2025 年 10 月 1 日 ... 313
2025 年 9 月 30 日 ... 313
2025 年 9 月 29 日 ... 313

2025 年 9 月 10 日 313
2025 年 8 月 24 日 314
2025 年 8 月 15 日 314
2025 年 8 月 6 日 314
2025 年 6 月 30 日 314
2025 年 6 月 20 日 314
2025 年 6 月 13 日 314
2025 年 4 月 30 日 315
2025 年 4 月 18 日 315
2025 年 4 月 11 日 315
2025 年 4 月 4 日 315
2025 年 3 月 31 日 315
2025 年 3 月 21 日 316
2025 年 3 月 14 日 316
クラスターの設定 317
自動クラスターの作成 317
前提条件 50
クラスターの作成 - AWS コンソール 319
クラスターの作成 - AWS CLI 324
次のステップ 98
クラスターを作成する 330
前提条件 50
ステップ 1: クラスター IAM ロールを作成する 331
ステップ 2: クラスターを作成する 332
ステップ 3: kubeconfig を更新する 342
ステップ 4: クラスターのセットアップ 343
次のステップ 98
プロビジョンドコントロールプレーンの作成 345
概要 95
ユースケース 156
コントロールプレーンのスケーリング階層 348
考慮事項 102
プロビジョンドコントロールプレーンの開始方法 354
高度なコントロールプレーン設定 362
概要 95
サポートされている Kubernetes コントロールプレーンパラメータ 363

考慮事項 102
次のステップ 98
コントロールプレーンパラメータを設定する 374
クラスターのインサイト 384
クラスターインサイトタイプ 385
考慮事項 385
ユースケース 385
はじめに 388
クラスターインサイトを表示する 388
Kubernetes バージョンを更新する 397
Amazon EKS Auto Mode の考慮事項 221
概要 399
ステップ 1: アップグレードの準備をする 399
ステップ 2: アップグレードに関する考慮事項を確認する 400
ステップ 3: クラスターコントロールプレーンを更新する 401
ステップ 4: クラスターコンポーネントを更新する 404
Amazon EKS クラスターの Kubernetes バージョンをダウングレードする 405
Kubernetes バージョンのロールバック 405
ロールバックされるもの 406
ロールバックされないもの 406
前提条件 406
概要 408
ステップ 1: ロールバックの準備状況インサイトを確認する 408
ステップ 2: ワーカーノードを準備する 411
ステップ 3: クラスターコントロールプレーンをロールバックする 412
ステップ 4: ロールバックの進行状況をモニタリングする 414
考慮事項と警告 415
ロールバックとアドオン 416
関連リソース 416
EKS Auto Mode クラスターをロールバックする 417
クラスターを削除 425
考慮事項 102
前提となる手順 426
クラスターの削除 (eksctl) 426
クラスターを削除する (AWS コンソール) 427
クラスターを削除する (AWS CLI) 428

EKS クラスターを誤って削除しないように保護する 430
クラスターエンドポイントのアクセス 431
IPv6 クラスターエンドポイント形式 431
IPv4 クラスターエンドポイント形式 432
クラスタープライベートエンドポイント 433
クラスターエンドポイントのアクセスの変更 434
プライベート専用 API サーバーへのアクセス 438
エンドポイントアクセスの設定 439
Windows サポートを有効にする 443
考慮事項 102
前提条件 50
Windows サポートを有効にする 444
Windows ポッドをデプロイする 447
Windows ノードでより高い Pod 密度をサポートする 447
Windows サポートを無効にする 448
プライベートクラスター 448
クラスターアーキテクチャの要件 449
ノードの要件 449
ポッドの要件 451
コントロールプレーンエグレス 454
エグレスルーティングモード 455
前提条件 456
顧客ルーティングエグレスでクラスターを作成する 458
既存のクラスターの更新 459
IPv6 に関する考慮事項 460
考慮事項 461
IAM 条件キー 461
OIDC プロバイダー設定 462
接続を確認する 463
エグレスのトラブルシューティング 463
自動スケーリング 471
EKS Auto Mode 471
その他のソリューション 471
ゾーンシフトの詳細 472
ポッド間の東西ネットワークトラフィックフローを理解する 473
EKS ゾーンシフトの要件 477

よくある質問 485
その他のリソース 488
ゾーンシフトを有効にする 488
考慮事項 489
Amazon Application Recovery Controller とは 489
ゾーンシフトとは 489
ゾーンオートシフトとは 490
ゾーンシフト中に EKS は何をしますか? 490
EKS クラスターを Amazon Application Recovery Controller (ARC) に登録する (AWS コンソール) 491
次のステップ 98
アクセスを管理する 492
一般的なタスク 492
背景 235
EKS Auto Mode に関する考慮事項 426
Kubernetes API アクセス 494
IAM アイデンティティと Kubernetes のアクセス許可を関連付ける 495
クラスター認証モードを設定する 496
アクセスエントリ 497
aws-auth ConfigMap 555
OIDC プロバイダーをリンクする 567
OIDC プロバイダーのリンクを解除する 573
クラスターリソースにアクセスする 573
必要なアクセス許可 574
CloudTrail の可視性 578
クラスターリソースを変更する 579
必要なアクセス許可 579
CloudTrail の可視性 578
kubectl を使用してクラスターにアクセスする 580
EKS コンソールを使用して接続する (CloudShell) 581
AWS CLI を使用して kubeconfig ファイルを作成する 581
AWS へのワークロードアクセス 583
サービスアカウントトークン 583
クラスターアドオン 585
Amazon Elastic Kubernetes Service クラスター上のワークロードへの AWS Identity and Access Management 許可の付与 585

IRSA による認証情報 590
Pod Identity 616
コンピューティングを管理する 652
コンピューティングオプションの比較 652
マネージド型ノードグループ 658
マネージド型ノードグループの概念 659
マネージド型ノードグループのキャパシティータイプ 662
作成 666
更新 681
動作の詳細を更新する 686
起動テンプレート 691
削除 708
セルフマネージド型ノード 710
Amazon Linux 712
Bottlerocket 721
Windows 725
Ubuntu Linux 735
更新方式 738
AWS Fargate 751
AWS Fargate 考慮事項 752
Fargate 比較テーブル 755
はじめに 757
プロファイルを定義する 763
プロファイルを削除する 768
ポッド設定の詳細 769
OS パッチ適用イベント 772
メトリクスを収集する 775
ログ記録 777
Amazon EC2 インスタンスタイプ 789
maxPods の決定方法 791
EKS Auto Mode に関する考慮事項 426
事前構築済みの最適化された AMI 792
AL2 AMI の廃止 793
Amazon Linux 799
Bottlerocket 810
Ubuntu Linux 826

Windows ... 826
ノードのヘルスと修復 ... 919
ノードモニタリングエージェント ... 920
自動ノード修復 ... 921
ノードのヘルス検出 ... 922
自動ノード修復 ... 942
ノードのヘルスを表示する ... 947
ノードログの取得 ... 950
ネットワークトラフィックをキャプチャする ... 953
ハイブリッドノード ... 956
機能 ... 957
制限 ... 959
考慮事項 ... 959
その他のリソース ... 959
前提条件 ... 960
ハイブリッドノードを実行する ... 1031
構成する ... 1054
仕組み ... 1142
ハイブリッドノード用 nodeadm ... 1185
トラブルシューティング ... 1203
Hybrid Nodes ゲートウェイ ... 1225
アプリデータストレージ ... 1268
Amazon EBS ... 1268
考慮事項 ... 1269
前提条件 ... 1269
ステップ 1: IAM ロールを作成する ... 1270
ステップ 2: Amazon EBS CSI ドライバーを入手する ... 1278
ステップ 3: サンプルアプリケーションをデプロイする ... 1279
Amazon EC2 インスタンスストア ... 1279
考慮事項 ... 1279
前提条件 ... 1280
Amazon EC2 Instance Store CSI ドライバーをインストールする ... 1280
サンプルアプリケーションをデプロイする ... 1281
I/O スロットリング ... 1281
新たに起動したインスタンスでの NVMe 名前空間の変更 ... 1283
Amazon EFS ... 1284

考慮事項 1284
前提条件 1285
ステップ 1: IAM ロールを作成する 1286
ステップ 2: Amazon EFS CSI ドライバーを取得する 1292
ステップ 3:Amazon EFS ファイルシステムを作成する 1293
ステップ 4: サンプルアプリケーションをデプロイする 1293
Amazon S3 Files 1293
考慮事項 1293
前提条件 1294
ステップ 1: IAM ロールを作成する 1294
ステップ 2: Amazon EFS CSI ドライバーを取得する 1304
ステップ 3: Amazon S3 ファイルシステムを作成する 1304
ステップ 4: サンプルアプリケーションをデプロイする 1305
Amazon FSx for Lustre 1305
ドライバーをデプロイする 1305
最適化 (EFA) 1313
最適化 (EFA 以外) 1329
Amazon FSx for NetApp ONTAP 1337
Amazon FSx for OpenZFS 1338
Amazon File Cache 1338
Mountpoint for Amazon S3 1339
考慮事項 1339
ドライバーをデプロイする 1340
ドライバーを削除する 1350
CSI スナップショットコントローラー 1351
ネットワークを設定する 1353
管理コンソールから既存の VPC サブネットを Amazon EKS クラスターに追加する 1353
VPC およびサブネットの要件 1353
VPC の要件と考慮事項 1354
サブネットの要件と考慮事項 1356
共有サブネットの要件と考慮事項 1363
「VPC を作成する」 1364
前提条件 50
パブリックサブネットおよびプライベートサブネット 1364
パブリックサブネットのみ 1367
プライベートサブネットのみ 1369

セキュリティグループの要件 1371
デフォルトのクラスターセキュリティグループ 1371
クラスタートラフィックの制限 1372
共有セキュリティグループ 1374
ネットワーキングアドオンの管理 1375
組み込みアドオン 1375
オプションの AWS ネットワークアドオン 1376
Amazon VPC CNI 1376
代替 CNI プラグイン 1505
Multus 1507
AWS ロードバランサーコントローラー 1508
CoreDNS 1527
kube-proxy 1545
ワークロード 1551
サンプルデプロイ (Linux) 1551
前提条件 50
名前空間を作成します。 1552
Kubernetes デプロイメントを作成する 1552
サービスを作成する 1554
作成されたリソースを確認する 1555
ポッドでシェルを実行する 1558
次のステップ 1559
サンプルデプロイ (Windows) 1559
前提条件 50
名前空間を作成します。 1552
Kubernetes デプロイメントを作成する 1552
サービスを作成する 1554
作成されたリソースを確認する 1562
ポッドでシェルを実行する 1558
次のステップ 1566
Vertical Pod Autoscaler 1566
Vertical Pod Autoscaler をデプロイする 1567
Vertical Pod Autoscaler のインストールをテストします 1568
Horizontal Pod Autoscaler 1572
Horizontal Pod Autoscaler テストアプリケーションを実行する 1573
ネットワーク負荷分散 1576

前提条件 50
考慮事項 102
ネットワークロードバランサーを作成する 1580
(オプション) サンプルアプリケーションをデプロイする 1582
アプリケーション負荷分散 1586
前提条件 50
イングレスグループで ALB を再利用する 1589
(オプション) サンプルアプリケーションをデプロイする 1591
サービスの外部 IP を制限する 1594
リポジトリにイメージをコピーする 1596
Amazon イメージレジストリを表示する 1599
Amazon EKS アドオン 1603
考慮事項 1604
アドオン用のカスタム名前空間 1605
Amazon EKS 自動モードの考慮事項 1606
サポート 1607
AWS アドオン 1609
コミュニティアドオン 1633
マーケットプレイス アドオン 1639
アドオンを作成する 1656
アドオンを更新する 1669
互換性を確認する 1676
アドオンを削除する 1678
IAM ロール 1682
カスタマイズ可能なフィールド 1689
コンテナイメージを検証する 1692
EKS 機能 1694
利用可能な機能 1694
AWS Controllers for Kubernetes (ACK) 1694
Argo CD 1695
kro (Kube Resource Orchestrator) 1695
EKS 機能のメリット 1696
料金 295
EKS 機能の仕組み 1697
一般的なユースケース 1697
機能の使用 1699

EKS 機能リソース 1699
機能ステータスについて 1700
機能を作成する 1701
機能を一覧表示する 1701
機能の説明 1703
機能の設定を更新する 1704
機能を削除する 1705
次のステップ 98
機能コントローラーログ 1706
サポートされているログのタイプ 1706
ログ配信の設定 293
例: CloudWatch ログインサイトでのログのクエリ 1708
ログの表示 294
料金 295
Kubernetes resources 1709
Argo CD リソース 1709
kro リソース 1710
ACK リソース 1712
リソースの制限 1713
次のステップ 98
考慮事項 1714
機能 IAM ロールと Kubernetes RBAC 1714
マルチクラスターアーキテクチャパターン 1714
EKS 機能をセルフマネージドソリューションと比較する 1716
AWS Controllers for Kubernetes 1719
ACK の仕組み 1719
ACK のメリット 1720
サポートされる AWS サービス 1721
他の EKS マネージド機能との統合 1721
ACK の使用を開始する 1722
ACK 機能を作成する 1722
ACK の概念 1732
アクセス許可の設定 1740
考慮事項 1747
トラブルシューティング 1755
セルフマネージドとの比較 1759

Argo CD 1763
Argo CD の仕組み 1764
Argo CD のメリット 1765
AWS アイデンティティセンターとの統合 1766
他の EKS マネージド機能との統合 1721
Argo CD の使用を開始する 1766
Argo CD 機能を作成する 1767
Argo CD の概念 1783
アクセス許可の設定 1790
Argo CD の使用 1803
プライベートエンドポイントアクセス 1860
Argo CD に関する考慮事項 1865
トラブルシューティング 1874
セルフマネージドとの比較 1887
kro 1892
kro の仕組み 1892
kro のメリット 1893
他の EKS マネージド機能との統合 1721
kro の開始方法 1894
kro 機能の作成 1894
kro の概念 1906
アクセス許可の設定 1913
考慮事項 1918
トラブルシューティング 1923
セルフマネージドとの比較 1930
機能をトラブルシューティングする 1931
一般的なトラブルシューティングのアプローチ 1931
トラブルシューティングにコントローラーログを使用する 1932
機能のヘルスを確認する 1933
機能でよく見られるステータス 1934
Kubernetes リソースのステータスを確認する 1935
IAM アクセス許可とクラスターアクセスを確認する 1935
機能に固有のトラブルシューティング 1937
すべての機能に共通する問題 1938
次のステップ 98
クラスターの管理 1940

コストモニタリング 1940
Pod ごとにコストを表示する 1941
Kubecost のインストール 1942
Kubecost ダッシュボードにアクセスする 1944
Kubecost の詳細 1945
メトリクスサーバー 1957
考慮事項 102
Amazon EKS アドオンを使用してコミュニティアドオンとしてデプロイする 1958
マニフェストを使用してデプロイする 1959
Helm を使用してアプリケーションをデプロイする 1960
リソースのタグ付け 1961
タグの基本 1962
リソースのタグ付け 1962
タグの制限 1963
請求用のリソースにタグを付ける 1964
コンソールでのタグの処理 1965
CLI、API または `eksctl` でのタグの操作 1966
Service Quotas 1967
AWS マネジメントコンソール で EKS Service Quotas を表示する 1968
AWS CLI で EKS サービスクォータを表示する 1968
Amazon EKS Service Quotas 1969
AWS Fargate サービスクォータ 1969
セキュリティ 1971
ベストプラクティス 1972
脆弱性の分析 1972
Amazon EKS 用 Center for Internet Security (CIS) ベンチマーク 1973
Amazon EKS のプラットフォームバージョン 1973
オペレーティングシステムの脆弱性リスト 1974
Amazon Inspector によるノード検出 1974
Amazon GuardDuty によるクラスターとノードの検出 1974
コンプライアンスの検証 1974
EKSに関する考慮事項 1975
インフラストラクチャセキュリティ 1976
レジリエンス 1983
サービス間の混乱した代理の防止 1984
Kubernetes に関する考慮事項 1986

証明書への署名 1986
CA ローテーション 1989
デフォルトの ロールとユーザー 2027
シークレット暗号化を有効にする 2032
AWS 秘密マネジャー 2037
すべての Kubernetes API データを対象とするデフォルトのエンベロープ暗号化 2037
RBAC の強化 2046
EKS Auto に関する考慮事項 2048
API セキュリティと認証 2049
ネットワークセキュリティ 2049
EC2 マネージドインスタンスのセキュリティ 2049
自動リソース管理 2051
セキュリティのベストプラクティス 2051
EKS 機能に関する考慮事項 2052
機能 IAM ロール 2052
EKS アクセスエントリ 2053
その他の Kubernetes アクセス許可 2055
機能で必要な IAM アクセス許可 2057
セキュリティのベストプラクティス 2057
次のステップ 98
IAM リファレンス 2060
オーディエンス 2061
アイデンティティを使用した認証 2061
ポリシーを使用したアクセス権の管理 2065
IAM および Amazon EKS 2067
アイデンティティベースのポリシー 2072
サービスにリンクされた役割 2080
ポッド実行IAMロール 2098
コネクタ IAM ロール 2103
AWS マネージドポリシー 2107
トラブルシューティング 2146
クラスター IAM ロール 2148
ノード の IAM ロール 2152
Amazon EKS 自動モードI クラスター IAM ロール 2158
Auto Mode ノードの IAM ロール 2164
機能 IAM ロール 2167

クラスターをモニタリングする 2174
Amazon EKS でのモニタリングとログ記録 2174
Amazon EKS のモニタリングとログ記録のツール 2176
オブザーバビリティダッシュボード 2180
概要 2180
クラスターヘルス 2180
コントロールプレーンのモニタリング 2181
クラスターのインサイト 2183
ノードのヘルスの問題 2183
EKS 機能 2184
コンテナネットワークのオブザーバビリティ 2184
ユースケース 156
機能 81
はじめに 388
前提条件と重要な注意事項 2186
AWS CLI、EKS API、NFM API の使用 2188
Infrastructure as Code (IaC) の使用 2189
動作の仕組み 227
考慮事項と制限事項 2020
Prometheus メトリクス 2201
ステップ 1: Prometheus メトリクスを有効にする 2202
ステップ 2: Prometheus メトリクスを使用する 2204
ステップ 3: Prometheus スクレイパーを管理する 2204
Helm を使用してデプロイする 2205
コントロールプレーン 2208
Amazon CloudWatch 2217
Amazon CloudWatch の基本メトリクス 2217
Amazon CloudWatch オブザーバビリティオペレータ 2226
コントロールプレーンのログ 2227
コントロールプレーンログを有効または無効にする 2229
クラスターのコントロールプレーンログを表示する 2231
AWS CloudTrail 2233
リファレンス 2233
ログファイルエントリ 2234
Auto Scaling グループメトリクス 2237
ADOT Operator 2237

Amazon EKS ダッシュボード 2239
Amazon EKS ダッシュボードとは 2239
ダッシュボードは AWS Organizations をどのように使用しますか? 2240
AWS コンソールを使用して EKS ダッシュボードを有効にする 2241
EKS ダッシュボードを表示する 2242
ダッシュボードを設定する 2243
AWS コンソールを使用して EKS ダッシュボードを無効にする 2244
EKS ダッシュボードのトラブルシューティング 2244
他のサービスでの使用 2251
AWS バックアップ 2251
AWS CloudFormation 2252
Amazon EKS と AWS CloudFormation のテンプレート 2252
AWS CloudFormation の詳細はこちら 2253
AWS CodeConnections 2253
Argo CD で AWS CodeConnections を使用する 2253
Argo CD を用いた CodeConnections を使用する際の考慮事項 2254
Amazon Detective 2255
Amazon EKS との連携で Amazon Detective を使用する 2255
Amazon GuardDuty 2256
AWS ローカルゾーン 2257
AWS Resilience Hub 2258
AWS 秘密マネジャー 2258
Argo CD で AWS Secrets Manager を使用する 2259
Amazon Security Lake 2261
Security Lake を Amazon EKS で使用する利点 2261
Amazon EKS の Security Lake の有効化 2262
Security Lake の EKS ログの分析 2262
Amazon VPC Lattice 2262
Amazon EKS Connector 2263
Amazon EKS Connector の考慮事項 2263
Amazon EKS コネクターの必要な IAM ロール 2264
クラスターに接続する 2264
考慮事項 2265
前提条件 2265
ステップ 1: クラスターの登録 2265
ステップ 2: eks-connector エージェントをインストールする 2268

次のステップ 2270
クラスターに対するアクセスを許可する 2270
前提条件 2270
クラスターの登録解除 2272
Kubernetes クラスターの登録を解除する 2273
Kubernetes クラスターのリソースをクリーンアップする 2274
EKS Connector をトラブルシューティングする 2274
基本的なトラブルシューティング 2274
Helm の問題: 403 Forbidden 2276
コンソールのエラー: クラスターが Pending 状態でスタックしています 2277
コンソールのエラー: ユーザーシステム:serviceaccount:eks-connector:eks-connector can't impersonate resource users in API group at cluster scope (eks-connector はクラスタースコープで API グループのリソースユーザーを偽装できません) 2277
コンソールのエラー: […] is forbidden: User […] cannot list resource […] in API group at the cluster scope ([…] は禁止されています: ユーザー […] が、クラスタースコープで API グループのリソース […] をリストできません) 2278
コンソールのエラー: Amazon EKS can't communicate with your Kubernetes cluster API server. 正常に接続するには、クラスターが ACTIVE 状態である必要があります。数分後にもう一度お試しください。 2278
Amazon EKS Connector Pod がクラッシュループしている 2279
eks-connector の起動の失敗: InvalidActivation 2279
クラスターノードにアウトバウンド接続がありません 2280
Amazon EKS Connector Pod が ImagePullBackOff 状態になっている 2281
よくある質問 2281
セキュリティに関する考慮事項 2282
AWS の責任 2283
お客様の責任 2283
Amazon EKS on AWS Outposts 2284
各デプロイオプションを使用する時 2284
2 つのローカルクラスターの実装 2285
デプロイオプションの比較 2286
ローカルクラスタを実行する 2291
サポート対象の AWS リージョン 2291
ローカルクラスターをデプロイする 2291
EKS プラットフォームのバージョン 2302
VPC とサブネットを作成する 2323

切断の準備をする 2327
容量に関する考慮事項 2332
クラスターのトラブルシューティングを行う 2335
ローカルクラスターを実行する (EC2 インスタンスストア) 2347
EC2 インスタンスストアを設定した Outposts にある Amazon EKS ローカルクラスター . 2347
はじめに 2348
ローカルクラスターをデプロイする (EC2 インスタンスストア) 2348
VPC とサブネットを作成する (EC2 インスタンスストア) 2355
キャパシティに関する考慮事項 (EC2 インスタンスストア) 2357
切断への備え (EC2 インスタンスストア) 2360
Amazon EKS アドオン (EC2 インスタンスストア) 2366
クラスターをトラブルシューティングする (EC2 インスタンスストア) 2368
ノード 2370
eksctl 2372
AWS マネジメントコンソール 2374
EKS における AI/ML 2381
AI/ML に Amazon EKS を使用する理由 2381
主なユースケース 2382
ケーススタディ 2382
ガイド構造 2383
Amazon EKS で AI/ML の使用を開始する 2383
クラスターを設定する 2384
アーキテクチャの概要 2384
クラスターコンピューティングオプション 2385
設定内容 2385
はじめに 388
コマンドラインインターフェイス (CLI) 2386
Terraform 2438
推測 2463
推論に Amazon EKS を使用する理由 2463
用語集 2464
Amazon EKS での推論の仕組み 2464
推論パフォーマンスメトリクス 2465
コールドスタート時間の短縮 2466
デプロイするもの 2466
モデルのロードと提供 2467

自動スケーリング 2483
モデルのロードを高速化する 2505
クラスターの設定 2518
EKS Linux GPU AMI を使用する 2518
Windows GPU AMI のセットアップ 2524
EFA を使用したクラスターのトレーニングの設定 2530
Inferentia で推論クラスターを設定する 2542
容量管理 2548
EC2 インスタンスタイプから選択 2548
EC2 購入オプションを理解する 2549
購入オプションをワークロード要件に一致させる 2549
EC2 サービスクォータを確認する 2550
Amazon EKS で EC2 購入オプションを使用する 2550
混合戦略：購入オプションを組み合わせる 2551
EKS Auto Mode と Karpenter 2553
マネージドノードグループとセルフマネージドノード 2572
EKS で P6e-GB200 を使用する 2578
デバイスの管理 2592
動的リソース割り当てとデバイスプラグインの比較 2593
Amazon EKS 向け DRA ドライバー 2593
Amazon EKS 用のデバイスプラグイン 2594
考慮事項 102
DRA の ResourceClaim と ResourceClaimTemplate 2595
トピック 2597
EFA デバイス 2598
Neuron アクセラレーター 2615
NVIDIA GPU 2623
recipe 2672
特定のノードで Pod がスケジュールされないようにする 2673
リソース 2675
ワークショップ 2675
ベストプラクティス 2675
リファレンスアーキテクチャ 2677
チュートリアル 2678
ツール 2680
モデルコンテキストプロトコル (MCP) 2680

概要 95
統合の例 2682
はじめに 388
はじめに 2683
設定 2699
ツール 2702
Amazon Q 2712
仕組み 155
トラブルシューティングに Amazon Q を使用する 2713
考慮事項 102
詳細情報 2713
バージョニング 2714
Kubernetes バージョン 2714
標準サポートで入手可能なバージョン 2715
延長サポートで利用可能なバージョン 2715
バージョンのロールバック 2716
Amazon EKS Kubernetes リリースカレンダー 2716
AWS CLI を使用してバージョン情報を取得する 2717
Amazon EKS のバージョンのよくある質問 2718
Amazon EKS 延長サポートに関するよくある質問 2721
標準サポートバージョン 2723
延長サポートバージョン 2732
サポート期間を表示する 2736
アップグレードポリシーを表示する 2737
延長サポートを有効にする 2739
延長サポートを無効にする 2740
プラットフォームのバージョン 2741
Kubernetes バージョン 1.37 2743
Kubernetes バージョン 1.36 2743
Kubernetes バージョン 1.35 2746
Kubernetes バージョン 1.34 2749
Kubernetes バージョン 1.33 2755
Kubernetes バージョン 1.32 2762
Kubernetes バージョン 1.31 2771
最新のプラットフォームバージョンを取得する 2781
プラットフォームバージョンを変更する 2782

トラブルシューティング 2783
容量不足 2783
ノードがクラスターへの参加に失敗します 2783
許可されていないか、アクセスが拒否されました (kubectl) 2785
hostname doesn't match 2786
getsockopt: no route to host 2786
Instances failed to join the Kubernetes cluster 2787
マネージド型ノードグループのエラー 2787
Not authorized for images 2792
ノードが NotReady 状態です 2792
EKS ログコレクター 2793
コンテナランタイムネットワークの準備ができていません 2794
TLS ハンドシェイクタイムアウト 2795
無効なクライアントトークンID 2796
コントロールプレーンをアップグレードする前に、ノードグループが Kubernetes バージョンと一致する必要があります 2796
多数のノードを起動すると、Too Many Requests エラーが発生します。 2797
Kubernetes API サーバーリクエストに対する HTTP 401 Unauthorized エラーレスポンス 2797
Amazon EKS プラットフォームのバージョンは現在のプラットフォーム バージョンより 2 つ以上遅れています 2798
クラスターの正常性に関するよくある質問およびエラーコードと解決パス 2801
Amazon EKS に関連するプロジェクト 2809
EKS にデプロイされたソフトウェアのサポート 2809
管理ツール 2810
eksctl 2810
AWS Controllers for Kubernetes 2810
kro (Kube Resource Orchestrator) 2811
Argo CD 2811
Flux CD 2811
Kubernetes 用 CDK 2811
ネットワーク 2812
Amazon VPC CNI Plugin for Kubernetes 2812
AWS Load Balancer Controller for Kubernetes 2812
ExternalDNS 2812
機械学習 2812
Kubeflow 2813

Auto Scaling 2813
Cluster Autoscaler 2813
Karpenter 2813
Escalator 2813
モニタリング 2814
Prometheus 2814
継続的インテグレーション/継続的デプロイメント 2814
Jenkins X 2814
新機能とロードマップ 2815
ドキュメント履歴 2816
貢献する 2888
単一ページを編集 2888
前提条件 50
手順 105
プルリクエストの概要 2890
GitHub でファイルを編集する 2891
前提条件 50
手順 105
スタイルのフィードバックを表示する 2892
Vale をインストールする 2893
VS Code Vale 拡張機能をインストールする 2893
Vale を同期する 2894
VS Code でスタイルフィードバックを表示する 2894
ページを作成 2894
ページを作成 2894
ナビゲーションにページを追加 2895
リンクを挿入 2895
EKS User Guideのページまたはセクションへのリンク 2896
AWS ドキュメント内の別のガイドへのリンク 2896
外部ウェブページへのリンク 2896
Amazon Q で作成する 2897
VS Code で Amazon Q をインストールする 2897
Amazon Q にログインする 2898
Amazon Q を使用してコンテンツを作成する 2898
ヒント 2898
PR プレビューを見る 2899

プレビューを見る 2900
プレビューの制限事項 2900
AsciiDoc 構文 2901
新しいページ 2901
ヘッダー 2901
テキストのフォーマット 2901
Lists 2902
リンク 2902
コードの例 2902
イメージ 2903
テーブル 2903
コールアウト 2903
他のファイルを含める 2904
属性 (エンティティに類似) 2904
手順 2904

このページの改善にご協力ください

このユーザーガイドに貢献するには、すべてのページの右側のペインにある「GitHub でこのページを編集する」リンクを選択してください。

Amazon EKS とは

> **Tip**
>
> 今後開催予定の Amazon EKS ワークショップに登録してください。

Amazon EKS: Kubernetes 管理の簡素化

Amazon Elastic Kubernetes Service (EKS) は、Kubernetes クラスターの操作の複雑性を解消する、フルマネージドの Kubernetes サービスです。EKS を使用すると次の利点があります。

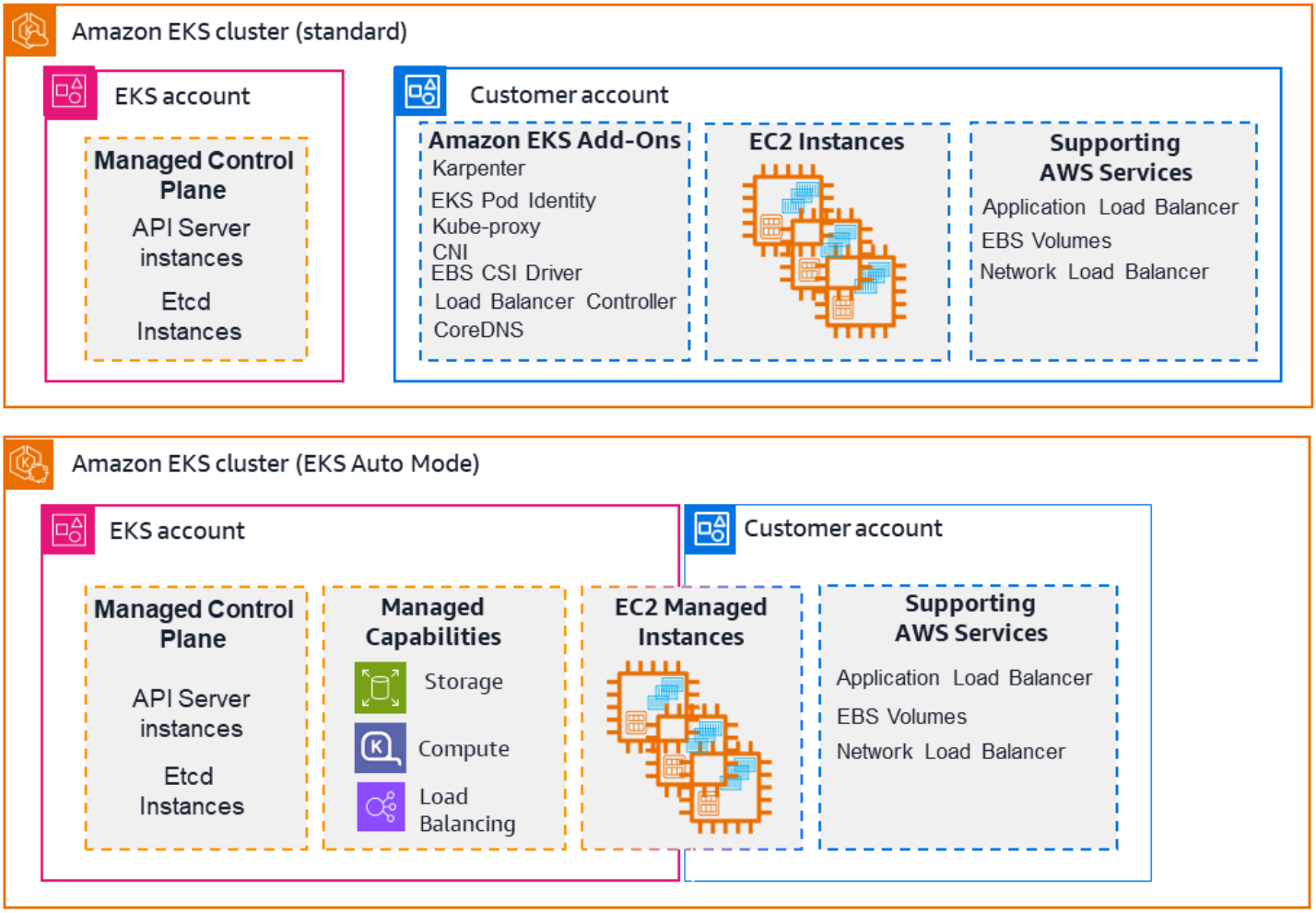
- 運用オーバーヘッドを減らしてアプリケーションをすばやくデプロイできる
- 絶えず変化するワークロードの需要に合わせてシームレスにスケールできる
- AWS 統合と自動更新によりセキュリティが向上する
- 標準 EKS または完全に自動化された EKS 自動モードから選択

Amazon Elastic Kubernetes Service (Amazon EKS) は、Amazon Web Services (AWS) クラウドと独自のデータセンター (EKS Anywhere と Amazon EKS Hybrid Nodes) のどちらでも、Kubernetes クラスターを実行するための要となるプラットフォームです。

Amazon EKS は、Kubernetes クラスターの構築、保護、および保守を簡素化します。独自のデータセンターを維持するよりも、ピーク需要を満たすのに十分なリソースを提供する方がコスト効率が良い場合があります。Amazon EKS を使用するためのアプローチには、主に次の 2 つがあります。

- EKS 標準: EKS でクラスターを作成すると、AWS が Kubernetes コントロールプレーンを管理します。ノードを管理する、ワークロードをスケジュールする、AWS クラウドと統合する、コントロールプレーン情報を保存しスケールしてクラスターを常時稼働させる、といったことを行うコンポーネントが自動的に処理されます。
- EKS Auto Mode: EKS Auto Mode 機能を使用すると、EKS の制御が Node (Kubernetes データプレーン) の管理にも及びます。インフラストラクチャの自動プロビジョニング、最適なコンピューティングインスタンスの選択、リソースの動的なスケーリング、継続的なコストの最適化、オペレーティングシステムへのパッチ適用、AWS セキュリティサービスとの統合により、Kubernetes の管理を簡素化します。

以下の図は、Amazon EKS が Kubernetes クラスターを AWS クラウドとどのように統合するかを示しています。これは、クラスターの作成にどの方法を選択したかによって異なります。



Amazon EKS では、摩擦を取り除いて本番稼働までの時間を短縮し、パフォーマンス、可用性、回復力を高め、システムセキュリティを強化できます。詳細については、「Amazon Elastic Kubernetes Service」を参照してください。

Kubernetes を使用した構築とスケーリング: Amazon EKS 機能

Amazon EKS はクラスターの構築と管理に役立つだけでなく、Kubernetes を使用したアプリケーションシステムの構築とスケーリングにも役立ちます。Amazon EKS 機能は、ハンズフリーの Kubernetes ネイティブツールによってクラスターの機能を拡張するフルマネージド型のクラスターサービスで、以下が含まれます。

- Argo CD: Argo CD は、ワークロード、AWS リソース、クラウドインフラストラクチャに対して宣言型の GitOps ベースの継続的デプロイを提供します。

- Kubernetes 用 AWS コントローラー (ACK): ACK は、AWS リソースの Kubernetes ネイティブな作成とライフサイクル管理を可能にし、ワークロードオーケストレーションと Infrastructure-as-code ワークフローを統合します。
- kro (Kube Resource Orchestrator): kro はネイティブ Kubernetes 機能を拡張して、カスタムリソースの作成、オーケストレーション、および構成を簡素化し、独自にカスタマイズしたクラウドの構成要素を作成するツールを提供します。

EKS 機能は、これらの基盤となるプラットフォームコンポーネントをクラスターにインストール、保守、スケーリングする運用上の負担を最小限に抑えるクラウドリソースであり、クラスタープラットフォームのオペレーションではなくソフトウェアの構築に集中できます。

詳細については、EKS 機能 を参照してください。

Amazon EKS の機能

Amazon EKS には次の高度な機能があります。

管理インターフェイス

EKS には、AWS マネジメントコンソール、Amazon EKS API/SDK、CDK、AWS CLI、eksctl CLI、AWS CloudFormation、Terraform など、クラスターをプロビジョニング、管理、保守するためのインターフェイスがいくつか用意されています。詳細については、はじめに および クラスターの設定 を参照してください。

アクセス制御ツール

EKS は、Kubernetes と AWS Identity and Access Management (AWS IAM) の両方の機能を利用して、ユーザーとワークロードからのアクセスを管理します。詳細については、the section called “Kubernetes API アクセス” および the section called “AWS へのワークロードアクセス ” を参照してください。

コンピューティングリソース

コンピューティングリソースの場合、幅広い Amazon EC2 インスタンスタイプと Nitro や Graviton などの AWS イノベーションを Amazon EKS で使用して、ワークロードのコンピューティングを最適化できます。詳細については、コンピューティングを管理する を参照してください。

[ストレージ]

EKS Auto Mode は、EBS ボリューム を使用して、ストレージクラスを自動的に作成します。コンテナストレージインターフェイス (CSI) ドライバーを使用すると、アプリケーションのストレージニーズに応じて Amazon S3、Amazon S3 Files、Amazon EFS、Amazon FSx、Amazon File Cache を使用することもできます。詳細については、アプリデータストレージ を参照してください。

セキュリティ

責任共有モデルは、Amazon EKS のセキュリティとの関連で採用されています。詳細については、「Security best practices」、「Infrastructure security」、「Kubernetes security」を参照してください。

モニタリングツール

オブザーバビリティダッシュボードを使用して Amazon EKS クラスターをモニタリングします。モニタリングツールには、Prometheus、CloudWatch、Cloudtrail、ADOT Operator などがあります。ダッシュボード、Metrics Server、およびその他のツールの詳細については、「Amazon EKS クラスターのコストをモニタリングして最適化する」および「Kubernetes Metrics Server でのリソース使用状況の表示」を参照してください。

クラスター機能

EKS は、オープンソースのイノベーションに基づいて、継続的なデプロイ、クラウドリソース管理、リソース構成のためのマネージドクラスター機能を提供します。EKS はクラスターに Kubernetes API をインストールしますが、コントローラーやその他のコンポーネントは EKS で実行され、フルマネージド型であるため、パッチ適用、スケーリング、モニタリングを自動化できます。詳細については、EKS 機能 を参照してください。

Kubernetes の互換性とサポート

Amazon EKS は Kubernetes 準拠の認定を受けており、Kubernetes と互換性のあるアプリケーションをリファクタリングなしでデプロイできるほか、Kubernetes コミュニティのツールとプラグインを使用できます。EKS では、Kubernetes について標準サポートと延長サポートの両方を提供しています。詳細については、the section called “Kubernetes バージョン” を参照してください。

関連サービス

Amazon EKS で利用できるサービス

Amazon EKS を使用してデプロイしたクラスターで他の AWS サービスを利用できます。

Amazon EC2

Amazon EC2 では、オンデマンドでスケーラブルなコンピューティング能力が得られます。

Amazon EBS

Amazon EBS では、スケーラブルで高性能のブロックストレージリソースをアタッチできます。

Amazon ECR

Amazon ECR では、コンテナイメージを安全に保存できます。

Amazon CloudWatch

Amazon CloudWatch では、AWS のリソースおよびアプリケーションをリアルタイムでモニタリングできます。

Amazon Prometheus

Amazon Managed Service for Prometheus では、コンテナ型アプリケーションのメトリクスを追跡できます。

Elastic Load Balancing

Elastic Load Balancing では、受信トラフィックを複数のターゲットに分散できます。

Amazon GuardDuty

Amazon GuardDuty で EKS クラスターへの脅威を検出します。

AWS Resilience Hub

AWS Resilience Hub では、EKS クラスターの耐障害性を評価できます。

Amazon EKS の料金

Amazon EKS には、Kubernetes クラスターバージョンのサポートに基づくクラスタごとの価格設定、Amazon EKS Auto Mode の価格設定、Amazon EKS Hybrid Nodes の vCPU ごとの価格設定があります。

Amazon EKS を使用する場合、Kubernetes ワーカーノードでアプリケーションを実行するために使用した AWS リソースに対して個別に料金を支払います。例えば、Amazon EBS ボリュームとパブリック IPv4 アドレスを持つ Amazon EC2 インスタンスとして Kubernetes ワーカーノードを実行し

ている場合、Amazon EC2 を介したインスタンス容量、Amazon EBS を介したボリューム容量、および Amazon VPC を介した IPv4 アドレスに対して課金されます。

Amazon EKS コントロールプレーンとクラスター内のワーカーノードの間の通信には、VPC のリクエスタマネージドネットワークインターフェイスを使用します。この接続のお客様側のトラフィックには AWS の標準のデータ転送料金がかかります。具体的には、コントロールプレーンからワーカーノードへのイングレスのトラフィックとワーカーノードからコントロールプレーンへのエグレスのトラフィックです。この接続のコントロールプレーン側のデータ転送コストは、Amazon EKS のコストに含まれています。データ転送料金の詳細については、「データ転送料金表」を参照してください。

詳細な料金情報については、Kubernetes アプリケーションで使用している AWS サービスのそれぞれの料金ページを参照してください。

- Amazon EKS クラスター、Amazon EKS Auto Mode、Amazon EKS 機能、Amazon EKS Hybrid Nodes の料金については、「Amazon EKS の料金」を参照してください。
- Amazon EC2 の料金については、「Amazon EC2 オンデマンド料金」および「Amazon EC2 スポット料金」を参照してください。
- AWS Fargate の料金については、「AWS Fargate の料金」を参照してください。
- Amazon EKS クラスターで使用されるコンピューティングに Savings Plans を使用できます。詳細については、「Pricing with Savings Plans」(貯蓄プランによる価格設定) を参照してください。

Amazon EKS の一般的なユースケース

Amazon EKS は、AWS 上で堅牢なマネージド型の Kubernetes サービスを提供しています。このサービスは、コンテナ化されたアプリケーションを最適化するように設計されています。以下は、Amazon EKS の強みを活用して特定のニーズに対応するのに役立つ、最も一般的ないくつかのユースケースを示しています。

高可用性アプリケーションのデプロイ

Elastic Load Balancing を使用することで、複数のアベイラビリティゾーンをまたいでアプリケーションの高可用性を確保できます。

マイクロサービスアーキテクチャの構築

Kubernetes サービス検出機能と AWS クラウドマップまたは Amazon VPC Lattice を使用して、回復力のあるシステムを構築します。

ソフトウェアリリースプロセスを自動化する

自動化されたアプリケーションの構築、テスト、デプロイのプロセスをシンプルにする、継続的インテグレーションおよび継続的デプロイ (CI/CD) パイプラインを管理します。宣言型の継続的デプロイについては、「Argo CD による継続的デプロイ」を参照してください。

サーバーレスアプリケーションの実行

AWS Fargate と Amazon EKS を使用して、サーバーレスアプリケーションを実行します。つまり、Amazon EKS と Fargate が基盤となるインフラストラクチャを処理している間に、お客様はアプリケーション開発のみに集中できます。

機械学習ワークロードの実行

Amazon EKS は、TensorFlow、MXNet、PyTorch などの一般的な機械学習フレームワークと互換性があります。GPU サポートにより、複雑な機械学習タスクでも効果的に処理できます。

オンプレミスとクラウドで一貫してデプロイする

オンプレミス環境での Kubernetes の実行を簡素化するために、同じ Amazon EKS クラスター、機能、ツールを使用して AWS Outposts でセルフマネージドノードを実行したり、独自のインフラストラクチャで Amazon EKS Hybrid Nodes を使用したりできます。エアギャップのある自己完結型の環境では、Amazon EKS Anywhere を使用して、独自のインフラストラクチャで Kubernetes クラスターのライフサイクル管理を自動化できます。

費用対効果の高いバッチ処理とビッグデータワークロードの実行

スポットインスタンスを使用すると、わずかなコストで Apache Hadoop や Spark などのバッチ処理およびビッグデータワークロードを実行できます。これにより、未使用の Amazon EC2 キャパシティを割引価格で活用できます。

Kubernetes から AWS リソースを管理する

Kubernetes 用 AWS コントローラー (ACK) を使用すると、ネイティブの Kubernetes API を使用して Kubernetes クラスターから直接 AWS リソースを作成および管理できます。

プラットフォームエンジニアリング抽象化を構築する

kro (Kube Resource Orchestrator) を使用すると、複数のリソースを高次の抽象化に構成するカスタム Kubernetes API を作成できます。

アプリケーションを保護し、コンプライアンスを確保する

AWS Identity and Access Management (IAM)、Amazon Virtual Private Cloud (Amazon VPC)、AWSKey Management Service (AWS KMS) などの AWS セキュリティサービスと統合さ

れている Amazon EKS を使用して、強力なセキュリティ対策を実施し、コンプライアンスを維持します。これにより、業界標準に準拠したデータのプライバシーと保護が保証されます。

Amazon EKS アーキテクチャ

Amazon EKS は、Kubernetes の一般的なクラスターアーキテクチャに準拠しています。詳細については、Kubernetes ドキュメントの「Kubernetes Components」を参照してください。以下のセクションでは、Amazon EKS のその他のアーキテクチャの詳細を要約しています。

コントロールプレーン

Amazon EKS では、すべてのクラスターで独自の Kubernetes コントロールプレーンが使用されます。この設計により、各クラスターのインフラストラクチャは常に分離され、クラスターや AWS アカウントの間で重複が生じることはありません。セットアップには以下が含まれます。

分散コンポーネント

コントロールプレーンは、AWS リージョン内の 3 つの AWS アベイラビリティーゾーンをまたいで、少なくとも 2 つの API サーバーインスタンスと 3 つの etcd インスタンスを配置します。

最適なパフォーマンス

Amazon EKS はコントロールプレーンインスタンスを積極的にモニタリングして調整し、ピークパフォーマンスを維持します。

耐障害性

コントロールプレーンインスタンスに問題が生じた場合、Amazon EKS は必要に応じて別のアベイラビリティーゾーンを使用して、迅速にそのインスタンスを置き換えます。

一貫したアップタイム

複数のアベイラビリティーゾーンにまたがってクラスターを稼働させることにより、信頼性の高い API サーバーエンドポイントの可用性のサービスレベルアグリーメント (SLA) を実現しています。

Amazon EKS は Amazon Virtual Private Cloud (Amazon VPC) を使用して、単一クラスター内のコントロールプレーンコンポーネント間のトラフィックを制限します。Kubernetes のロールベースのアクセスコントロール (RBAC) ポリシーで許可されている場合を除き、クラスターコンポーネントは、他のクラスターまたは AWS アカウントからの通信を表示したり受信したりすることはできません。

コンピューティング

Amazon EKS クラスターには、コントロールプレーンに加えて、ノードと呼ばれる一連のワーカーマシンがあります。特定の要件を満たし、リソース使用率を最適化するには、適切なタイプの Amazon EKS クラスターノードを選択することが重要です。Amazon EKS には、以下のタイプのプライマリノードが用意されています。

EKS Auto Mode

EKS Auto Mode は、AWS の管理範囲をコントロールプレーンを超えてデータプレーンにまで拡張し、クラスターインフラストラクチャ管理を自動化します。コンピューティングの自動スケーリング、ネットワーク、負荷分散、DNS、ストレージ、GPU サポートといった Kubernetes のコア機能を、組み込みコンポーネントとして統合します。EKS Auto Mode は、セキュリティ機能が強化されたイミュータブル AMI を動的に使用して、ワークロードの需要に基づきノードを管理します。ポッドの中断予算を考慮しながら更新とアップグレードを自動化し、通常であればアドオン管理を必要とするマネージドコンポーネントも管理対象に含みます。このオプションは、日々の運用に AWS の専門知識を活用し、運用オーバーヘッドを最小限に抑え、インフラストラクチャ管理ではなくアプリケーション開発に集中したいユーザーに最適です。

AWS Fargate

Fargate はコンテナ用のサーバーレスコンピューティングエンジンで、基盤となるインスタンスの管理が不要になります。Fargate でアプリケーションのリソースニーズを指定すると、AWS はインフラストラクチャーのプロビジョニング、スケーリング、メンテナンスを自動的に行います。このオプションは、使いやすさを優先し、インフラストラクチャの管理よりもアプリケーションの開発とデプロイに集中したいユーザーに最適です。

Karpenter

Karpenter は、アプリケーションの可用性とクラスターの効率の向上に役立つ柔軟で高性能な Kubernetes クラスターオートスケーラーです。Karpenter は、アプリケーションの負荷の変化に応じて適切なサイズのコンピューティングリソースを起動します。このオプションでは、ワークロードの要件を満たすジャストインタイムのコンピューティングリソースをプロビジョニングできます。

マネージドノードグループ

マネージド型ノードグループは、Amazon EKS クラスター内の Amazon EC2 インスタンスのコレクションを管理するための自動化とカスタマイズを組み合わせたものです。AWS がノードのパッチ適用、更新、スケーリングなどのタスクを処理することで、運用面を容易にします。それと同時に、カスタム `kubelet` 引数がサポートされていることで、高度な CPU およびメモリ

管理ポリシーの可能性が広がります。さらに、サービスアカウントの AWS Identity and Access Management (IAM) ロールによってセキュリティを強化しながら、クラスターごとに個別のアクセス許可を設定する必要性を抑えます。

セルフマネージド型ノード

セルフマネージド型ノードでは、Amazon EKS クラスター内の Amazon EC2 インスタンスを完全に制御できます。ユーザーは、ノードの管理、スケーリング、メンテナンスを担当することで、基盤となるインフラストラクチャを完全に制御できます。このオプションは、ノードをきめ細かく制御およびカスタマイズする必要があり、インフラストラクチャの管理と保守に時間を費やす準備ができているユーザーに適しています。

Amazon EKS Hybrid Nodes

Amazon EKS Hybrid Nodes を使用すると、オンプレミスおよびエッジインフラストラクチャを Amazon EKS クラスターのノードとして使用できます。Amazon EKS Hybrid Nodes は、Kubernetes 管理を環境間で統合し、Kubernetes コントロールプレーン管理をオンプレミスおよびエッジアプリケーションの AWS にオフロードします。

EKS 機能

Amazon EKS は、フルマネージド型のクラスター機能を提供し、AWS が所有するインフラストラクチャで、クラスターとは分離してコントローラーやその他のコンポーネントを運用しながら、クラスター内に Kubernetes API (Kubernetes Custom Resource Definitions を含む) をインストールおよび管理します。EKS は、これらの機能のパッチ適用、スケーリング、モニタリングを自動化し、それらのライフサイクルを完全に管理することで、ワークロードのオーケストレーション、AWS リソース管理などのクラスター内サービスを運用する負担を軽減します。

EKS には、以下の機能タイプがあります。

Kubernetes 用 AWS コントローラー (ACK)

AWS Controllers for Kubernetes (ACK) は、Kubernetes API を使用して AWS リソースを管理できるようにし、S3 バケット、RDS データベース、IAM ロール、およびその他の AWS リソースを Kubernetes カスタムリソースとして定義できるようにします。同じツールとワークフローを使用し、Kubernetes ワークロードと並行して AWS リソースを管理でき、S3、RDS、DynamoDB、Lambda を含む 50 以上の AWS サービスをサポートしています。

Argo CD

Argo CD は、Git リポジトリを信頼できるソースとして使用し、アプリケーションワークロード、AWS リソース、クラスター設定に対して GitOps ベースの継続的デプロイを実装します。Argo CD は、クラスターを Git リポジトリと自動的に同期してドリフトを検出し、継続的に状態を調整することで、デプロイされたアプリケーションとリソースが、バージョン管理で定義された目的の状態と一致するようにします。Argo CD を使用すると、特定のクラスター上のアプリケーションを管理することも、単一の Argo CD リソースから複数のクラスターにわたってアプリケーションをデプロイおよび管理することもでき、変更がコミットされるたびに Git リポジトリから自動的にデプロイされます。

kro (Kube Resource Orchestrator)

kro (Kube Resource Orchestrator) は、複数のリソースをより高位の抽象化に構成するカスタム Kubernetes API を作成できるようにし、プラットフォームチームが一般的なリソースの組み合わせに対して再利用可能なパターンを定義できるようにします。これにより、プラットフォームチームは適切なガードレールを備えたセルフサービス機能を提供でき、開発者は、組織の標準とベストプラクティスを維持しながら、シンプルで目的に特化した API を使用して複雑なインフラストラクチャをプロビジョニングできます。

Kubernetes の概念

Amazon Elastic Kubernetes Service (Amazon EKS) はオープンソースの Kubernetes プロジェクトをベースとした AWS マネージドサービスです。Amazon EKS サービスが AWS クラウドとどのように統合するのかについて理解しておくべき点がいくつかあります (特に Amazon EKS クラスターを初めて作成する場合) が、Amazon EKS クラスターを一度起動して実行した後の使用方法は他の Kubernetes クラスターを使う場合とほぼ変わりません。そのため、Kubernetes クラスターの管理とワークロードのデプロイを開始するには、少なくとも Kubernetes の概念の基本について理解しておく必要があります。

このページでは、Kubernetes の概念を the section called “Kubernetes が選ばれる理由”、the section called “クラスター”、the section called “ワークロード” の 3 つのセクションに分けています。1 つめのセクションでは、Kubernetes サービスを、特に Amazon EKS のようなマネージドサービスとして実行する場合の利点について説明します。ワークロードのセクションでは、Kubernetes アプリケーションの構築、保存、実行、管理の方法について説明します。クラスターのセクションでは、Kubernetes クラスターを構成しているさまざまなコンポーネントと、Kubernetes クラスターを作成して管理するときのユーザーの責務について説明します。

トピック

- Kubernetes が選ばれる理由
- クラスター
- ワークロード
- 次のステップ

このコンテンツに含まれているリンクをクリックすると、Amazon EKS と Kubernetes の両方のドキュメントに記載されている Kubernetes の概念について解説したページが開き、ここで取り上げたトピックをさらに深く掘り下げることができます。Amazon EKS がどのように Kubernetes コントロールプレーンとコンピューティング機能を実装するかの詳細については、「the section called "アーキテクチャ"」を参照してください。

Kubernetes が選ばれる理由

Kubernetes は、ミッションクリティカルで本番稼働品質のコンテナ化アプリケーションを実行するときの可用性とスケーラビリティを高めることを目的に設計されました。1 台のマシン上で Kubernetes を実行するのではなく (実行可能ではありますが)、需要に応じて拡張または縮小する複数のコンピュータを横断してアプリケーションを実行できるようにすれば、Kubernetes は上記の目標を達成できます。Kubernetes が備える機能により、次のことが簡単に行えます。

- アプリケーションを複数のマシンにデプロイする (ポッドにデプロイされたコンテナを使用)
- コンテナの状態をモニタリングし、障害が発生したコンテナを再起動する
- 負荷に基づいてコンテナをスケールアップ/スケールダウンする
- コンテナを新しいバージョンで更新する
- コンテナ間でリソースを割り当てる
- マシン間でトラフィックを分散する

Kubernetes でこのタイプの複雑なタスクを自動化すれば、アプリケーションデベロッパーはインフラストラクチャのことを気にすることなく、アプリケーションワークロードの構築と改善に専念できます。デベロッパーは通常、YAML 形式の設定ファイルを作成し、そこにアプリケーションの望ましい状態を記述します。これには実行するコンテナ、リソースの上限、ポッドレプリカの数、CPU/メモリの割り当て、アフィニティールールなどが含まれます。

Kubernetes の属性

上記の目標を達成するため、Kubernetes は次の属性を使用します。

- コンテナ化 – Kubernetes はコンテナオーケストレーションツールです。Kubernetes を使用するには、まずアプリケーションをコンテナ化する必要があります。アプリケーションの種類に応じて、これは マイクロサービスのセット、バッチジョブ、その他形式のいずれかになります。それにより、アプリケーションで、ツールの巨大なエコシステムを含む Kubernetes ワークフローを利用することができます。そこではコンテナを イメージとしてコンテナレジストリ に保存したり、Kubernetes クラスター にデプロイしたり、利用可能な ノード 上で実行したりすることができます。個々のコンテナを Kubernetes クラスターにデプロイする前に、Docker または別の コンテナランタイム を使用してローカルのコンピュータ上で構築しテストすることができます。
- スケーラブル – アプリケーションを実行しているインスタンスのキャパシティーを超えてアプリケーションの需要が拡大した場合は、Kubernetes をスケールアップすることができます。必要に応じて、Kubernetes はアプリケーションに追加の CPU やメモリが必要かどうかを判断し、自動的に利用可能な容量を拡張するか、既存の容量を追加で利用します。アプリケーションでより多くのインスタンスを実行するための十分なコンピューティングがある場合はポッドレベルでスケーリングを実行することができます (水平ポッド自動スケーリング)。あるいは容量の増加に対応するためにより多くのノードを起動する必要がある場合はノードレベルでスケーリングを実行することができます (クラスターオートスケーラー または Karpenter)。容量が不要になると、これらのサービスは不要なポッドを削除して、不要なノードをシャットダウンすることができます。
- 使用可能 – アプリケーションまたはノードが異常な状態になった場合や使用できなくなった場合、Kubernetes は実行中のワークロードを使用可能な別のノードに移動できます。この問題はワークロードを実行しているワークロードまたはノードの実行中のインスタンスを削除すれば、強制的に解決できます。ここでのポイントはワークロードを現在の場所で実行できなくなったときは他の場所で起動できるという点です。
- 宣言型 – Kubernetes は、アクティブな照合を使用して、クラスターに宣言した状態が実際の状態と一致していることを常にチェックします。Kubernetes オブジェクトをクラスターに適用すると、通常は YAML 形式の設定ファイルを通じて、例えばクラスター上で実行するワークロードの起動をリクエストすることができます。後で設定に変更を加えることができます。例えば、新しいバージョンのコンテナを使用する、メモリをさらに割り当てるといったことができます。Kubernetes は、目的の状態を確立するためにやるべきことをやります。例えば、ノードの起動や停止、ワークロードの停止および再起動、更新したコンテナの取得などを実行してください。
- コンポーザブル – アプリケーションは通常、複数のコンポーネントで構成されているため、これら一連のコンポーネント (通常は複数のコンテナで表されます) をまとめて管理できるようにしておくことをお勧めします。これを、Docker Compose では Docker を使用して直接実行するのに対して、Kubernetes では Kompose コマンドを利用して実行します。その方法の例については、「Docker Compose ファイルを Kubernetes リソースに変換する」を参照してください。

- 拡張可能 – 独自のソフトウェアとは異なり、オープンソースの Kubernetes プロジェクトではニーズに合わせて Kubernetes を自由に拡張できるようになっています。API と設定ファイルは直接変更することができます。サードパーティーは、インフラストラクチャとエンドユーザー Kubernetes 機能の両方を拡張する場合、独自の コントローラー を作成するよう奨励されています。Webhook を使用すると、クラスタールを設定して、ポリシーを適用し、状況の変化に対応することができます。Kubernetes クラスターを拡張する方法の詳細については、「Kubernetes を拡張する」を参照してください。
- ポータブル – 多くの組織では、アプリケーションのニーズをすべて同じ方法で管理できることから、自社での Kubernetes の運用を標準化しています。デベロッパーはコンテナ化されたアプリケーションを同じパイプラインを使って構築し、保存することができます。これらのアプリケーションは、オンプレミス、クラウド、レストランの POS 端末、企業のリモートサイト分散されている IoT デバイスなどで実行している Kubernetes クラスターにデプロイできます。オープンソースであるため、このような特殊な Kubernetes ディストリビューションをその管理に必要なツールと併せて開発することができます。

Kubernetes の管理

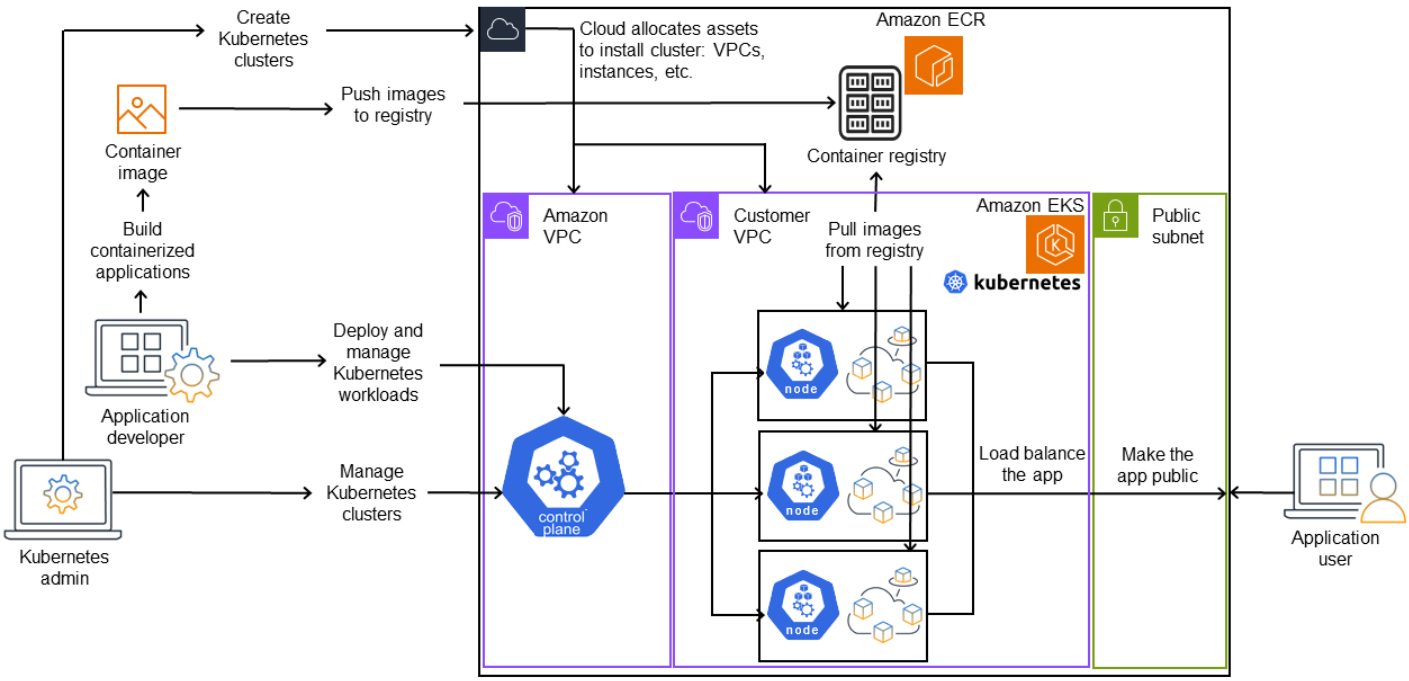
Kubernetes ソースコードは無料で入手できるため、手元にある機器を使用して Kubernetes をインストールし、管理することができます。ただし、Kubernetes を自己管理するには運用に関する専門知識が必要であり、維持していくには時間と労力がかかります。そのため、本番環境のワークロードをデプロイする人の大半が、独自のテスト済み Kubernetes ディストリビューションと、Kubernetes の専門家のサポートを備えた、クラウドプロバイダー (Amazon EKS など) またはオンプレミスプロバイダー (Amazon EKS Anywhere など) を利用しています。これらを利用することで、以下のようなクラスターのメンテナンスに必要な未分化で手間のかかる作業の多くを軽減しています。

- ハードウェア – 要件に応じて Kubernetes を実行できるハードウェアがない場合、AWS Amazon EKS などのクラウドプロバイダーを利用すると初期費用を節約できます。Amazon EKS では、コンピューティングインスタンス (Amazon Elastic Compute Cloud)、独自のプライベート環境 (Amazon VPC)、アイデンティティとアクセス許可の一元管理 (IAM)、ストレージ (Amazon EBS) など、AWS で提供されている最善のクラウドリソースを利用できます。AWS はコンピュータ、ネットワーク、データセンター、および Kubernetes の実行に必要な他のすべての物理コンポーネントを管理します。同様に、需要が最も高い日に最大容量を処理できるようにデータセンターを計画しておく必要はありません。Amazon EKS Anywhere やその他のオンプレミスの Kubernetes クラスターの場合、Kubernetes デプロイに使用されるインフラストラクチャの管理は利用する側の責任となりますが、引き続き AWS を利用して Kubernetes を最新の状態に保つことができます。

- コントロールプレーン管理 – Amazon EKS は AWS がホストしている Kubernetes コントロールプレーンのセキュリティと可用性を管理します。コンテナのスケジュール、アプリケーションの可用性の管理、その他の重要なタスクはこのコントロールプレーンが行うため、ユーザーはアプリケーションのワークロードに専念できます。クラスターが故障した場合、AWS には実行状態に復元するための方法がいくつかあります。Amazon EKS Anywhere の場合、コントロールプレーンを自分で管理することになります。
- テスト済みのアップグレード – クラスターをアップグレードするときは、Amazon EKS または Amazon EKS Anywhere を利用して、Kubernetes ディストリビューションのテスト済みバージョンを用意できます。
- アドオン – 拡張して Kubernetes と連携するように構築された数百ものプロジェクトがあり、クラスターのインフラストラクチャに追加したり、ワークロードの実行を支援するために使用したりできます。これらのアドオンを自分で構築して管理する代わりに、AWS にはクラスターで使用できる the section called “Amazon EKS アドオン”Amazon EKS アドオンが用意されています。Amazon EKS Anywhere には多くの一般的なオープンソースプロジェクトのビルドを含む、キュレーションパッケージ があります。そのため、ユーザーはソフトウェアを自分で構築したり、重要なセキュリティパッチ、バグ修正、アップグレードを管理したりする必要はありません。同様に、デフォルトの設定がニーズを満たしていれば、通常、それらのアドオンの設定はほとんど必要ありません。アドオンを使用してクラスターを拡張する方法については、「the section called “拡張クラスター”」を参照してください。

Kubernetes の動作

以下の図は、Kubernetes 管理者またはアプリケーションデベロッパーが Kubernetes クラスターを作成して使用する際に実行する主要なアクティビティを示したものです。そのプロセスにおいて、基盤となるクラウドプロバイダーの例として AWS クラウドを使用し、Kubernetes のコンポーネントがどのように相互作用するかを説明しています。



Kubernetes 管理者は、クラスターが構築されるプロバイダーのタイプに固有のツールを使用して、Kubernetes クラスターを作成します。この例では、Amazon EKS という Kubernetes マネージドサービスを提供する AWS クラウドをプロバイダーとして使用します。マネージドサービスは、クラスター用に Virtual Private Clouds (Amazon VPC) を新たに 2 つ作成する、ネットワークを設定する、クラウドアセット管理用に新しい VPC に対して Kubernetes アクセス許可を直接マッピングするなど、クラスターの作成に必要なリソースを自動的に割り当てます。マネージドサービスは、コントロールプレーンサービスの実行先を確認し、ワークロードを実行するための Kubernetes ノードとして 0 個以上の Amazon EC2 インスタンスを割り当てます。AWS はコントロールプレーン用に 1 つの Amazon VPC 自体を管理し、もう 1 つの Amazon VPC にはワークロードを実行するカスタマーノードが含まれます。

Kubernetes 管理者が今後行うタスクの多くは、`kubectl` などの Kubernetes ツールを使って行われます。このツールはクラスターのコントロールプレーンに直接サービスをリクエストします。その場合、クラスターに対してクエリや変更を行う方法は任意の Kubernetes クラスターに対して行う場合の方法と非常によく似ています。

このクラスターにワークロードをデプロイする場合、アプリケーションデベロッパーはいくつかのタスクを実行することができます。デベロッパーはアプリケーションを 1 つ以上のコンテナイメージに構築し、そのイメージを Kubernetes クラスターにアクセスできるコンテナレジストリにプッシュ

する必要があります。AWS には、それを行うための Amazon Elastic Container Registry (Amazon ECR) というレジストリがあります。

このアプリケーションを実行するにはデベロッパーはレジストリからどのコンテナを取得するか、それらのコンテナをポッドにどのようにラップするかなど、アプリケーションの実行方法をクラスターに指示する YAML 形式の設定ファイルを作成します。コントロールプレーン (スケジューラー) はコンテナを 1 つ以上のノードにスケジュールし、各ノードのコンテナランタイムが必要なコンテナーを実際にプルして実行してください。また、デベロッパーは Application Load Balancer を設定することで、各ノードで実行されている利用可能なコンテナにトラフィックを分散し、パブリックネットワーク上で外部から利用できるようにアプリケーションを公開することもできます。これで、アプリケーションを使用したいユーザーが、アプリケーションエンドポイントに接続してアクセスできるようになります。

以下のセクションでは、これらの各機能について、Kubernetes のクラスターとワークロードの観点から詳しく説明します。

クラスター

ジョブに Kubernetes クラスターの起動および管理が含まれる場合は、Kubernetes クラスターの作成、強化、管理、削除の方法を理解しておく必要があります。また、クラスターを構成しているコンポーネントと、それらのコンポーネントを維持するために行うべきことについても、知っておく必要があります。

クラスターを管理するツールは、Kubernetes のサービスとその基盤となるハードウェアプロバイダーとの重複を処理します。そのため、これらのタスクの自動化は Kubernetes プロバイダー (Amazon EKS や Amazon EKS Anywhere など) が、そのプロバイダーに固有のツールを使って行うのが一般的です。例えば、Amazon EKS クラスターを起動する場合は `eksctl create cluster` を使用できますが、Amazon EKS Anywhere には `eksctl anywhere create cluster` を使用できます。これらのコマンドは、Kubernetes クラスターを作成しますが、プロバイダー固有のものであるため、Kubernetes プロジェクト自体には含まれていないことに注意してください。

クラスターの作成および管理ツール

Kubernetes プロジェクトには、Kubernetes クラスターを手動で作成するためのツールが用意されています。したがって、Kubernetes を単一のマシンにインストールしたり、マシン上でコントロールプレーンを実行したり、ノードを手動で追加したりする場合は、Kubernetes インストールツールに一覧表示されている kind、minikube、kubeadm などの CLI ツールを使用できます。クラスターの作成と管理のライフサイクル全体を簡素化して自動化するには、Amazon EKS や Amazon EKS

Anywhere といった定評ある Kubernetes プロバイダーがサポートしているツールを使用してください。作業がはるかに簡単になります。

AWS クラウドではAmazon EKS クラスターを、eksctl などの CLI ツールや、Terraform などのより宣言的なツールを使用して作成することができます (「Amazon EKS Blueprints for Terraform」を参照)。クラスターは AWS マネジメントコンソール を使用して作成することもできます。Amazon EKS で何ができるかについては、「Amazon EKS の機能」を参照してください。Kubernetes に関して Amazon EKS 側で果たす責務は次のとおりです。

- マネージドコントロールプレーン – AWS はコントロールプレーンをユーザーに代わって管理し、AWS アベイラビリティーゾーン全体で利用可能にするため、Amazon EKS クラスターが利用可能かつスケーラブルになります。
- ノード管理 – ノードを手動で追加する代わりに、the section called “マネージド型ノードグループ” マネージドノードグループまたは Karpenter を使用して、必要に応じて Amazon EKS でノードを自動的に作成できます。マネージドノードグループは、Kubernetes クラスター自動スケーリング と統合されています。ノード管理ツールを使用すると、スポットインスタンス やノード統合などによりコストを節約したり、ワークロードのデプロイ方法やノードの選択方法を設定する スケジューリング 機能を使用して可用性を実現したりすることができます。
- クラスターネットワーク – `eksctl` は、CloudFormation テンプレートを使用して、Kubernetes クラスター内のコントロールプレーンとデータプレーン (ノード) コンポーネントとの間にネットワークを設定します。また、内部と外部の通信を行うためのエンドポイントも設定します。詳細については、「ツールのインストール Amazon EKS ワーカーノードのクラスターネットワーキングの謎を解く」を参照してください。Amazon EKS 内のポッド間の通信は Amazon EKS Pod Identity を使用して行われます (the section called “Pod Identity” を参照)。これによりポッドは認証情報やアクセス許可を管理する AWS クラウドの方法を利用できるようになります。
- アドオン – Amazon EKS を使用すると、Kubernetes クラスターをサポートするためによく使用されるソフトウェアコンポーネントを構築して追加する必要がなくなります。例えば、AWS マネジメントコンソール から Amazon EKS クラスターを作成すると、Amazon EKS kube-proxy (the section called “kube-proxy”)、Amazon VPC CNI plugin for Kubernetes (the section called “Amazon VPC CNI”)、CoreDNS (the section called “CoreDNS”) アドオンが自動的に追加されます。利用可能なアドオンも含む、これらのアドオンの詳細については、「the section called “Amazon EKS アドオン”」を参照してください。

クラスターをオンプレミスのコンピュータとネットワーク上で実行できるように、Amazon は Amazon EKS Anywhere を提供しています。AWS クラウドをプロバイダーにする代わりに、Amazon EKS Anywhere を VMware vSphere、ベアメタル (Tinkerbell プロバイ

ダー)、Snow、CloudStack、Nutanix の各プラットフォームで、独自の機器を使用して実行することを選択できます。

Amazon EKS Anywhere は Amazon EKS で使用されているものと同じ Amazon EKS Distro ソフトウェアをベースにしています。ただし、Amazon EKS Anywhere は Amazon EKS Anywhere クラスター内のマシンのライフサイクル全体を管理するために、Kubernetes クラスター API (CAPI) インターフェイスのさまざまな実装に依存しています (vSphere の場合は CAPV、CloudStack の場合は CAPC など)。クラスター全体がユーザーの機器上で実行されるため、ユーザーはコントロールプレーンの管理とデータのバックアップの責任を追加で負うことになります (本ドキ `etcd` ュメントで後述する etcd を参照)。

クラスターコンポーネント

Kubernetes クラスターコンポーネントは、コントロールプレーンとワーカーノードという 2 つの主要な領域に分かれています。コントロールプレーンコンポーネントはクラスターを管理し、その API へのアクセスを提供します。ワーカーノード (単にノードとも呼ばれます) は実際のワークロードを実行する場所です。ノードコンポーネントは各ノード上で実行されるサービスで構成され、コントロールプレーンと通信してコンテナを実行してください。クラスターのワーカーノードのセットはデータプレーンと呼ばれます。

コントロールプレーン

コントロールプレーンはクラスターを管理する一連のサービスで構成されています。これらのサービスはすべて 1 台のコンピュータで実行される場合もあれば、複数のコンピュータに分散される場合もあります。内部的にはこれらはコントロールプレーンインスタンス (CPI) と呼ばれます。CPI の実行方法はクラスターの規模と、高可用性の要件に応じて異なります。クラスターの需要が増えると、コントロールプレーンサービスはスケールしてそのサービスのインスタンスを増やし、リクエストはインスタンス間で負荷分散されます。

Kubernetes コントロールプレーンのコンポーネントが実行するタスクには、以下が含まれます。

- クラスターコンポーネント (API サーバー) との通信 – API サーバー (kube-apiserver) はクラスターへのリクエストをクラスターの内部と外部の両方から行えるようにするために Kubernetes API を公開します。つまり、クラスターのオブジェクト (ポッド、サービス、ノードなど) を追加または変更するリクエストはポッドを実行する `kubectl` のリクエストなどの外部コマンドから送信することができます。同様に、ポッドのステータスを `kubelet` サービスにクエリするときのように、API サーバーからクラスター内のコンポーネントに対してリクエストを実行することができます。

- クラスターに関するデータの保存 (**etcd** キーバリューストア)* – サービスはクラスターの現在の状態を追跡するという重要なロールを担っています。etcdetcd サービスにアクセスできなくなると、ワークロードはしばらく実行し続けますが、クラスターの状態を更新したりクエリしたりすることはできなくなります。そのため、重要なクラスターは通常、複数の負荷分散された etcd サービスのインスタンスを同時に実行し、データの損失や破損に備えて、etcd キーバリューストアのバックアップを定期的に行っています。Amazon EKS では、これはすべてデフォルトで自動的に処理されます。Amazon EKS Anywhere は etcd のバックアップと復元 の手順を公開しています。etcd によるデータ管理の方法については、「etcd データモデル」を参照してください。
- ノードへの Pod のスケジュール (スケジューラー) – Kubernetes での Pod の起動または停止のリクエストは、Kubernetes スケジューラー (kube-scheduler) に送信されます。クラスターにはポッドを実行できるノードが複数含まれている場合があるため、どのノード (レプリカの場合は複数のノード) でポッドを実行するかはスケジューラーが決定します。リクエストされたポッドを既存のノードで実行するための十分な容量がない場合、他のポッドをプロビジョニングしていなければそのリクエストは失敗します。プロビジョニングには新しいノードを自動的に起動してワークロードを処理する the section called “マネージド型ノードグループ”マネージド型ノードグループや Karpenter などのサービスを有効化する作業が含まれます。
- コンポーネントを目的の状態に維持する (コントローラーマネージャー) – Kubernetes コントローラーマネージャーは、クラスターの状態をモニタリングしたり、クラスターに変更を加えて期待される状態に戻したりするデーモンプロセス (kube-controller-manager) として実行されます。特に、statefulset-controller、endpoint-controller、cronjob-controller、node-controller など、さまざまな Kubernetes オブジェクトを監視するコントローラーがいくつかあります。
- クラウドリソースの管理 (Cloud Controller Manager) – Kubernetes とその基盤となるデータセンターリソースのリクエストを実行するクラウドプロバイダーとのやり取りは、Cloud Controller Manager (cloud-controller-manager) が処理します。Cloud Controller Manager が管理するコントローラーにはルートコントローラー (クラウドネットワークルートの設定用)、サービスコントローラー (クラウドのロードバランシングサービスを使用するためのもの)、ノードライフサイクルコントローラー (クラウド API を使用してライフサイクル全体でノードを Kubernetes と同期させるためのもの) があります。

ワーカーノード (データプレーン)

単一ノード Kubernetes クラスターの場合、ワークロードはコントロールプレーンと同じマシンで実行してください。ただし、より標準的な構成では Kubernetes ワークロードを実行するための専用の個別のコンピュータシステム (ノード) を 1 つ以上用意します。

Kubernetes クラスターを初めて作成する場合、作成ツールによっては (既存のコンピュータシステムを識別するか、プロバイダーに新しいコンピュータシステムを作成させることによって) 特定の数のノードをクラスターに追加するように設定できるものがあります。これらのシステムにワークロードを追加する前に、以下の機能を実装するためのサービスを各ノードに追加します。

- 各ノードの管理 (**`kubelet`**)* – API サーバーは各ノードで実行されている kubelet サービスと通信し、ノードが正しく登録され、スケジューラーからリクエストされたポッドが実行されていることを確認します。kubelet はポッドマニフェストを読み取り、ローカルシステム上で Pod が必要とするストレージボリュームやその他の機能を設定できます。また、ローカルで実行されているコンテナの状態もチェックすることができます。
- ノードでコンテナを実行する (コンテナランタイム) – 各ノードの コンテナランタイム はノードに割り当てられた各ポッドに対してリクエストされたコンテナを管理します。つまり、適切なレジストリからコンテナイメージをプルし、そのコンテナを実行して停止し、コンテナに関するクエリに応答することができます。デフォルトのコンテナランタイムは containerd です。Kubernetes 1.24 以降、コンテナランタイムとして使用できる Docker (`dockershim`) 固有の統合は、Kubernetes から削除されました。ローカルシステムでコンテナをテストおよび実行する際は、引き続き Docker を使用することができますが、Kubernetes で Docker を使用するには、各ノードに Docker Engine をインストールして Kubernetes で使用する必要があります。
- コンテナ間のネットワークを管理する (**`kube-proxy`**) – Pod 間の通信をサポートできるようにするために、Kubernetes は Service と呼ばれる機能を使用して、それらの Pod に関連付けられた IP アドレスとポートを追跡する Pod ネットワークを設定します。kube-proxy サービスは各ノードで実行し、ポッド間でその通信を可能にします。

拡張クラスター

クラスターをサポートするために Kubernetes に追加できるサービスがいくつかありますが、そのいずれもコントロールプレーンでは実行されません。それらのサービスは多くの場合、kube-system 名前空間のノードで直接実行されるか、独自の名前空間で実行されます (サードパーティーのサービスプロバイダーではよく行われています)。一般的な例が、クラスターに DNS サービスを提供する CoreDNS サービスです。クラスターの kube-system でどのクラスターサービスが実行されているのかを確認する方法については、「組み込みサービスの発見」を参照してください。

クラスターに追加できるアドオンにはさまざまな種類があります。クラスターを健全な状態に保つには ログ記録、監査、メトリクスなどの作業を実行できる、クラスターをモニタリングするオブザーバビリティ機能を追加します。この情報があれば、多くの場合、同じオブザーバビリティインターフェイスを使用して、発生した問題をトラブルシューティングすることができます。こうしたタイプのサービスには、Amazon GuardDuty、CloudWatch (the section called “Amazon CloudWatch” を

参照)、AWS Distro for OpenTelemetry、Amazon VPC CNI plugin for Kubernetes (the section called “Amazon VPC CNI” を参照)、Grafana Kubernetes Monitoring などがあります。アプリデータストレージストレージでは Amazon EKS のアドオンには the section called “Amazon EBS”Amazon エラスティック・ブロック・ストア CSI ドライバー (ブロックストレージデバイスの追加用)、the section called “Amazon EFS”Amazon エラスティック・ファイル・システム CSI ドライバー (ファイルシステムストレージの追加用)、サードパーティー製ストレージのアドオン (the section called “Amazon FSx for NetApp ONTAP”Amazon FSx for NetApp ONTAP CSI ドライバーなど) などがあります。

使用可能な Amazon EKS アドオンの完全なリストについては、the section called “Amazon EKS アドオン” を参照してください。

ワークロード

Kubernetes では、ワークロードを「Kubernetes で動作するアプリケーション」と定義しています。そのアプリケーションは ポッド 内の コンテナ として実行される、一連のマイクロサービスで構成されていますが、バッチジョブやその他の種類のアプリケーションとして実行される場合もあります。Kubernetes は、それらのオブジェクトを設定またはデプロイするために行われたリクエストを確実に実行するロールを担います。アプリケーションをデプロイするユーザーはコンテナの構築方法、ポッドの定義方法、デプロイに使用する方法を学ぶ必要があります。

コンテナ

Kubernetes にデプロイして管理するアプリケーションワークロードの最も基本的な要素が Pod です。ポッドはアプリケーションのコンポーネントを格納する方法であるだけでなく、ポッドの属性を記述する仕様を定義する方法でもあります。RPM や Deb パッケージは Linux システム用のソフトウェアをまとめてパッケージ化しますが、それ自体はエンティティとしては動作しません。

ポッドはデプロイ可能な最小のユニットであるため、通常は 1 つのコンテナしか格納できません。しかし、コンテナが密結合している場合は複数のコンテナを 1 つのポッドに含めることができます。例えばウェブサーバーコンテナはログ記録、モニタリング、またはウェブサーバーコンテナと密結合したその他のサービスを提供できる サイドカー タイプのコンテナとともに、ポッドにパッケージ化できます。この場合、同じポッド内にあることで、そのポッドが実行している各インスタンスでは両方のコンテナが常に同じノードで実行されることになります。同様に、ポッド内のすべてのコンテナは同じ環境を共有し、ポッド内のコンテナは隔離された同じホストにあるかのように実行されます。これにより、各コンテナはポッドへのアクセスを可能にする 1 つの IP アドレスを共有し、コンテナはあたかも独自のローカルホスト上で実行しているかのように相互に通信することができます。

ポッドの仕様 (PodSpec) はポッドの望ましい状態を定義します。ワークロードリソースを使用して ポッドテンプレート を管理することで、個別のポッドまたは複数のポッドをデプロイできます。

ワークロードリソースには 展開 (複数のポッドレプリカを管理する)、StatefulSets (データベースポッドなどの一意にする必要があるポッドをデプロイする)、DaemonSets (ポッドをすべてのノードで継続的に実行する必要がある場合) が含まれます。詳細は後ほど説明します。

ポッドはデプロイできる最小のユニットですが、コンテナは構築して管理できる最小のユニットです。

コンテナの構築

ポッドは実際には 1 つ以上のコンテナを取り囲む構造に過ぎず、各コンテナ自体にファイルシステム、実行ファイル、設定ファイル、ライブラリ、その他アプリケーションを実際に実行するためのコンポーネントが格納されています。コンテナを普及させた会社の名前が Docker Inc. であったため、Docker コンテナと呼ばれることもあります。しかし、コンテナのランタイム、イメージ、ディストリビューション方法を業界向けに定義してきたのは オープン・コンテナ構想 です。また、コンテナは Linux に既に存在していた多くの機能から作成されていることから、OCI コンテナ、Linux コンテナ、あるいはただ単にコンテナとも呼ばれます。

コンテナを構築するときは通常、Dockerfile (会社名から命名) から始めます。Dockerfile 内では以下を識別します。

- ベースイメージ – ベースのコンテナイメージは通常はオペレーティングシステムのファイルシステム (レッドハット・エンタープライズ Linux や Ubuntu など) の最小バージョン、または特定の種類のアプリケーション (nodejs や python アプリケーションなど) を実行するソフトウェアを提供するように拡張された最小システムのいずれかから構築されるコンテナです。
- アプリケーションソフトウェア – Linux システムに追加するときとほぼ同じ方法で、アプリケーションソフトウェアをコンテナに追加できます。例えば、Dockerfile では `npm` と `yarn` を実行して JavaScript アプリケーションをインストールしたり、`yum` と `dnf` を実行して RPM パッケージをインストールしたりできます。つまり、Dockerfile で RUN コマンドを使用すると、ベースイメージのファイルシステムで使用可能な任意のコマンドを実行して、生成されたコンテナイメージの内部でソフトウェアをインストールしたり、設定したりできます。
- インストラクション – Dockerfile リファレンスには Dockerfile の設定時に追加できるインストラクションが記載されています。これにはコンテナ自体の中にあるもの (ローカルシステムの `ADD` または `COPY` ファイル) の構築、コンテナを実行するときに実行するコマンドの特定 (`CMD` または `ENTRYPOINT`)、コンテナを実行するシステムへのコンテナの接続 (実行する `USER`、マウントするローカル `VOLUME`、`EXPOSE` するポート) などに使用する指示が含まれています。

`docker` コマンドとサービスは従来、コンテナ (`docker build`) の構築に使用されてきましたが、コンテナイメージの構築に使用できるその他ツールには podman や nerdctl などがあります。コンテ

ナの構築方法については、「より良いコンテナ・イメージの構築」または「Dockerビルドの概要」参照してください。

コンテナの保存

コンテナイメージを作成したら、ワークステーションのコンテナディストリビューションレジストリ、またはパブリックコンテナレジストリに保存できます。ワークステーションでプライベートコンテナレジストリを実行すると、コンテナイメージをローカルに保存して、すぐに利用できるようにすることができます。

コンテナイメージを、よりパブリックな方法で保存するときはイメージをパブリックコンテナレジストリーにプッシュします。パブリックコンテナレジストリはコンテナイメージの保存と配布を一元的に行える場所です。代表的なパブリックコンテナレジストリには Amazon Elastic Container Registry、Red Hat Quay レジストリ、Docker Hub レジストリなどがあります。

Amazon Elastic Kubernetes Service (Amazon EKS) でコンテナ化されたワークロードを実行するときは、Amazon Elastic Container Registry に保存されている Docker 公式イメージのコピーをプルすることをお勧めします。Amazon ECR は2021 年からこれらのイメージを保存しています。人気の高いコンテナイメージは Amazon ECR Public Gallery で検索でき、中でも Docker Hub イメージは Amazon ECR Docker Gallery で検索できます。

コンテナの実行

コンテナは標準フォーマットで構築されているため、コンテナランタイム (Docker など) を実行でき、コンテンツがローカルマシンのアーキテクチャ (`x86_64` や `arm` など) に一致するマシンであれば、どのマシンでもコンテナを実行できます。コンテナをテストしたり、ローカルのデスクトップで実行したりするには `docker run` または `podman run` コマンドを使用して、ローカルホストでコンテナを起動します。ただし Kubernetes では、各ワーカーノードにコンテナランタイムがデプロイされており、そのノードにコンテナの実行をリクエストするかどうかは Kubernetes が決定します。

コンテナがノード上で実行されるように割り当てられると、そのノードはリクエストされたバージョンのコンテナイメージがノード上にすでに存在するかどうかを確認します。存在しなければ、Kubernetes がコンテナランタイムに対して、適切なコンテナレジストリからそのコンテナをプルしてローカルで実行するように指示します。コンテナイメージとは、ノートパソコン、コンテナレジストリ、Kubernetes ノード間を移動するソフトウェアパッケージのことです。コンテナとはそのイメージの実行中のインスタンスを指します。

ポッド

コンテナの準備が整ったら、ポッドの操作として、ポッドの設定、デプロイ、ポッドをアクセス可能にするなどの作業を行います。

ポッドの設定

ポッドを定義するときは一連の属性をポッドに割り当てます。これらの属性には少なくともポッド名と実行するコンテナイメージが含まれている必要があります。ただし、ポッドの定義を使用して設定する必要のある項目は他にも多数あります (ポッドに追加できる内容の詳細については、「PodSpec」のページを参照してください)。具体的には次のとおりです。

- ストレージ – 実行中のコンテナを停止して削除すると、より永続的なストレージを設定しない限り、そのコンテナ内のデータストレージは削除されます。Kubernetes は、多種多様なストレージタイプをサポートしており、Volumes の制御の下でタイプを抽出します。ストレージタイプにはCephFS、NFS、iSCSI などがあります。ローカルのコンピュータからローカルブロックデバイスを使用することもできます。クラスターからこれらのストレージタイプのいずれかを使用すれば、ストレージボリュームをコンテナのファイルシステム内の選択したマウントポイントにマウントできます。永続ボリュームはポッドが削除された後も残り、エフェメラルボリュームはポッドが削除されると、削除されます。クラスター管理者がクラスター用に異なる ストレージクラス を作成した場合、使用後にボリュームを削除するか、再利用するか、スペースがさらに必要になった場合に拡張するか、また、特定のパフォーマンス要件を満たすようにするかなど、使用するストレージの属性を選択できることがあります。
- シークレット – ポッド仕様のコンテナでシークレットを使用できるようにすると、ファイルシステム、データベース、その他保護されているアセットにアクセスするために必要な許可をそれらのコンテナに付与できます。キー、パスワード、トークンはシークレットとして保存できるアイテムです。シークレットを使用すると、この情報をコンテナイメージに保存する必要はなく、実行中のコンテナでシークレットを利用できるようにすることができます。シークレットに似ているのが ConfigMaps です。ConfigMap には一般に、サービスを設定するためのキーと値のペアなど、重要度が低い情報が格納されます。
- コンテナリソース – コンテナを詳細に設定するためのオブジェクトはリソース設定の形式をとることができます。コンテナごとに、そのコンテナが使用できるメモリと CPU の量をリクエストできるほか、コンテナが使用できるリソースの合計量の上限を設定できます。例については、「ポッドとコンテナ向けのリソース管理」を参照してください。
- 中断 – ポッドは意図せずに (ノードがダウンしたとき)、または意図的に (アップグレードが必要なとき) 中断することがあります。ポッド中断の予算を設定することで、中断が起きた場合のアプリ

ケーションの可用性をある程度制御することができます。アプリケーションの例については、「中断予算の指定」を参照してください。

- 名前空間 – Kubernetes には、Kubernetes コンポーネントとワークロードを互いに分離するさまざまな方法が用意されています。特定のアプリケーションのすべてのポッドを、同じ 名前空間 で実行することはそれらのポッドをまとめて保護し管理する一般的な方法です。独自の名前空間を作成して使用することも、名前空間を指定しない (Kubernetes が `default` の名前空間を使用する) ように選択することもできます。Kubernetes コントロールプレーンコンポーネントは通常、kube-system 名前空間で動作します。

これらの設定は通常、YAML ファイルにまとめて一緒に Kubernetes クラスターに適用します。パーソナルな Kubernetes クラスターの場合は、これらの YAML ファイルをローカルシステムに保存します。ただし、よりクリティカルなクラスターやワークロードの場合は、ストレージを自動化し、ワークロードと Kubernetes インフラストラクチャリソースの両方を更新する方法として、GitOps がよく使用されます。

ポッド情報の収集とデプロイに使用されるオブジェクトは以下のデプロイ方法のいずれかによって定義されます。

ポッドのデプロイ

ポッドをデプロイする際に選択すべき方法はそのポッドで実行する予定のアプリケーションのタイプに応じて異なります。この方法には以下のようなものがあります。

- ステートレスアプリケーション – ステートレスアプリケーションはクライアントのセッションデータを保存しないため、その次のセッションは前のセッションで起きたことを参照する必要がありません。これにより、ポッドが異常な状態になった場合は新しいポッドと交換するか、状態を保存せずにポッドを移動させるだけで済みます。ステートレスアプリケーション (ウェブサーバーなど) を実行している場合は Deployment を使用して Pods と ReplicaSets をデプロイできます。ReplicaSet は同時に実行したいポッドのインスタンス数を定義します。ReplicaSet は直接実行することもできますが、1 回に実行するポッドのレプリカの数を定義するため、デプロイ内でレプリカを直接実行するのが一般的です。
- ステートフルアプリケーション – ステートフルアプリケーションとはポッドの ID とポッドの起動順序が重要であるアプリケーションのことです。これらのアプリケーションには安定した永続的ストレージが必要であり、一貫した方法でデプロイしスケールインする必要があります。Kubernetes にステートフルアプリケーションをデプロイするには、StatefulSets を使用します。通常、StatefulSet として実行されるアプリケーションの一例が、データベースです。StatefulSet 内ではレプリカ、ポッドとそのコンテナ、マウントするストレージボリューム、

データを保存するコンテナ内の場所を定義できます。データベースを StatefulSet としてデプロイする例については、レプリケートされたステートフルアプリケーションの実行を参照してください。

- ノードあたりのアプリケーション – アプリケーションを Kubernetes クラスター内のすべてのノードで実行しなければならない場合があります。例えば、データセンターで、すべてのコンピュータでモニタリングアプリケーションまたは特定のリモートアクセスサービスを実行する必要がある場合です。Kubernetes では、DaemonSet を使用することで、選択したアプリケーションがクラスター内のすべてのノードで実行されるようにすることができます。
- 完了まで実行するアプリケーション – 特定のタスクを完了するために、複数のアプリケーションを実行する必要がある場合があります。例えば、毎月のステータスレポートを実行したり、古いデータを消去したりする場合です。Job オブジェクトを使用すると、アプリケーションを起動して実行し、タスクが完了したら、終了するように設定することができます。CronJob オブジェクトを使用すると、Linux の crontab 形式で定義された構造を使用して、特定の時間、分、日、月、曜日に、アプリケーションを実行するように設定できます。

ネットワークからアプリケーションにアクセスできるようにする

アプリケーションはさまざまな場所を移動するマイクロサービスのセットとしてデプロイされることが多いため、Kubernetes にはそれらのマイクロサービスが相互に検索できる方法が必要でした。また、Kubernetes クラスターの外にあるアプリケーションにアクセスするユーザーのために、Kubernetes にはそのアプリケーションを外部のアドレスとポートで公開する方法が必要でした。これらのネットワーク関連の機能はそれぞれ サービス オブジェクトと イングレス オブジェクトを使って実行されます。

- サービス – ポッドは異なるノードやアドレスに移動できるため、最初のポッドと通信する必要のある別のポッドがそのポッドの場所を特定することが困難になる場合があります。この問題を解決するため、Kubernetes ではアプリケーションを Service として表すことができます。サービス を使用すると、特定の名前を持つポッドまたはポッドのセットを識別し、そのアプリケーションのサービスをポッドから公開するポートと、他のアプリケーションがそのサービスにアクセスする際に使用できるポートを指定することができます。クラスター内の別の Pod は名前で Service をリクエストでき、Kubernetes はそのリクエストをその Service を実行している Pod のインスタンスの適切なポートに転送します。
- イングレス – イングレスは、Kubernetes Service によって表されたアプリケーションをクラスター外のクライアントから利用できるようにします。イングレス の基本機能にはロードバランサー (イングレス が管理)、イングレス コントローラー、コントローラーから サービス にリクエ

ストを送信するルールなどがあります。Kubernetes で選択できるイングレスコントローラーがいくつかあります。

次のステップ

Kubernetes の基本的な概念と、それらが Amazon EKS とどのように関連するのかを理解することで、Amazon EKS ドキュメントと Kubernetes のドキュメントの両方を参照しながら、Amazon EKS クラスターを管理したりワークロードをクラスターにデプロイしたりする際に必要になる情報をすぐに見つけられるようになります。Amazon EKS の使用を開始するには以下から選択してください。

- the section called “クラスターを作成する (eksctl)”
- the section called “クラスターを作成する”
- the section called “サンプルデプロイ (Linux)”
- クラスターの管理

クラウド環境とオンプレミス環境に Amazon EKS クラスターをデプロイする

Amazon EKS デプロイオプションを理解する

Amazon Elastic Kubernetes Service (Amazon EKS) は、クラウドおよびオンプレミス環境で Kubernetes をシームレスに実行できるフルマネージド型の Kubernetes サービスです。

クラウドでは、Amazon EKS は Kubernetes コントロールプレーンとノードの Kubernetes クラスターインフラストラクチャ管理を自動化します。これは、コンテナのスケジューリング、アプリケーションの可用性の管理、リソースの動的なスケーリング、コンピューティングの最適化、クラスターデータの保存、その他の重要な機能の実行に不可欠です。Amazon EKS を使用すると、AWS インフラストラクチャの堅牢なパフォーマンス、スケーラビリティ、信頼性、可用性とともに、AWS のネットワーキング、セキュリティ、ストレージ、オブザーバビリティサービスとのネイティブな統合を実現できます。

オンプレミス環境での Kubernetes の実行を簡素化するには、同じ Amazon EKS クラスター、機能、およびツールを使用して、「the section called “ノード”」を実行するか、独自のインフラストラクチャで Amazon EKS Hybrid Nodes を作成する、あるいは自己完結型の エアギャップ環境に対して Amazon EKS Anywhere を使用できます。

クラウド内の Amazon EKS

Amazon EKS は、AWS リージョン、AWS ローカルゾーン、および AWS Wavelength Zones のコンピューティングで使用できます。クラウド上の Amazon EKS では、Kubernetes コントロールプレーンのセキュリティ、スケーラビリティ、可用性は、AWS リージョンの AWS によって完全に管理されます。AWS リージョンでコンピューティングを使用してアプリケーションを実行すると、Amazon EKS 自動モードなど、AWS および Amazon EKS の全機能を利用できます。これにより、AWS のコンピューティング、ストレージ、ネットワークに関する Kubernetes クラスターインフラストラクチャの管理がワンクリックで完全に自動化されます。AWS Local Zones と AWS Wavelength Zones でコンピューティングを使用してアプリケーションを実行する場合、Amazon EKS セルフマネージド型ノードを使用してクラスターコンピューティング用の Amazon EC2 インスタンスを接続すると、AWS Local Zones と AWS Wavelength Zones で利用可能な他の AWS サービスを使用できます。詳細については、「AWS Local Zones の機能」と「AWS Wavelength Zones の機能」を参照してください。

	AWS リージョンの Amazon EKS	Local/Wavelength Zones の Amazon EKS
Kuberenetes コントロールプレーンの管理	AWS マネージド	AWS マネージド
Kubernetes コントロールプレーンの場所	AWS リージョン	AWS リージョン
Kubernetes データプレーン	• Amazon EKS 自動モード • Amazon EKS マネージド型ノードグループ • Amazon EC2 セルフマネージド型ノード • AWS Fargate	• Amazon EKS マネージド型ノードグループ (Local Zones のみ) • Amazon EC2 セルフマネージド型ノード
Kubernetes データプレーンの場所	AWS リージョン	AWS Local Zones または Wavelength Zones

データセンターまたはエッジ環境の Amazon EKS

独自のデータセンターまたはエッジ環境でアプリケーションを実行する必要がある場合は、Amazon EKS on AWS Outposts または Amazon EKS Hybrid Nodes を使用できます。クラスターコンピューティングには AWS Outposts の Amazon EC2 インスタンスでセルフマネージド型ノードを使用するか、クラスターコンピューティングには独自のオンプレミスまたはエッジインフラストラクチャで Amazon EKS Hybrid Nodes を使用できます。AWSOutposts は、データセンターまたはコロケーション施設で実行する AWS マネージドインフラストラクチャです。一方、Amazon EKS Hybrid Nodes は、オンプレミス環境またはエッジ環境で管理する物理マシンまたは仮想マシンで実行されます。Amazon EKS on AWS Outposts と Amazon EKS Hybrid Nodes には、オンプレミス環境から AWS リージョンへの信頼性の高い接続が必要です。また、クラウドでアプリケーションを実行するために使用するのと同じ Amazon EKS クラスター、機能、ツールを使用できます。AWS Outposts で実行する場合、代わりに AWS Outposts 上の Amazon EKS ローカルクラスターを使用して、Kubernetes クラスター全体を AWS Outposts にデプロイすることができます。

	Amazon EKS Hybrid Nodes	Amazon EKS on AWS Outposts
Kuberenetes コントロールプレーンの管理	AWS マネージド	AWS マネージド
Kubernetes コントロールプレーンの場所	AWS リージョン	AWS リージョンまたは AWS Outposts
Kubernetes データプレーン	カスタマー管理の物理マシンまたは仮想マシン	Amazon EC2 セルフマネージド型ノード
Kubernetes データプレーンの場所	カスタマーデータセンターまたはエッジ環境	カスタマーデータセンターまたはエッジ環境

エアギャップ環境用の Amazon EKS Anywhere

Amazon EKS Anywhere は、オンプレミス環境とエッジ環境でのインフラストラクチャのセットアップや Kubernetes クラスターのライフサイクルオペレーションなど、差別化につながらない手間のかかる作業を自動化することで、Kubernetes クラスターの管理を簡素化します。Amazon EKS とは異なり、Amazon EKS Anywhere はカスタマーマネージド製品であり、お客様は Amazon EKS Anywhere クラスターのクラスターのライフサイクルオペレーションとメンテナンスを担当しま

す。Amazon EKS Anywhere は Kubernetes サブプロジェクトクラスター API (CAPI) 上に構築されており、VMware vSphere、ベアメタル、Nutanix、Apache CloudStack、AWS Snow などのさまざまなインフラストラクチャをサポートしています。Amazon EKS Anywhere はエアギャップ環境で実行でき、オブザーバビリティと ID 管理のためのリージョナル AWS サービスとの統合を選択的に利用できます。Amazon EKS Anywhere のサポートと AWS が提供する Kubernetes アドオンへのアクセスを受けるには、Amazon EKS Anywhere エンタープライズサブスクリプションを購入できます。

	Amazon EKS Anywhere
Kuberenetes コントロールプレーンの管理	カスタマーマネージド
Kubernetes コントロールプレーンの場所	カスタマーデータセンターまたはエッジ環境
Kubernetes データプレーン	カスタマー管理の物理マシンまたは仮想マシン
Kubernetes データプレーンの場所	カスタマーデータセンターまたはエッジ環境

Amazon EKS ツール

Amazon EKS Connector を使用して、準拠する Kubernetes クラスターを AWS に登録および接続し、Amazon EKS コンソールに表示できます。クラスターが接続されると、Amazon EKS コンソールでクラスターのステータス、設定、およびワークロードを確認できます。この機能を使用して Amazon EKS コンソールで接続されたクラスターを表示できますが、Amazon EKS Connector では、Amazon EKS コンソールを介して接続されたクラスターに対する管理操作や変更操作は有効になりません。

Amazon EKS Distro は、すべての Amazon EKS サービスを強化する基盤となる Kubernetes コンポーネントの AWS ディストリビューションです。これには、Kubernetes コントロールプレーンコンポーネント (etcd、kube-apiserver、kube-scheduler、kube-controller-manager) やネットワークコンポーネント (CoreDNS、kube-proxy、CNI プラグイン) など、機能する Kubernetes クラスターに必要なコアコンポーネントが含まれています。Amazon EKS Distro を使用して、選択したツールで Kubernetes クラスターを自己管理できます。Amazon EKS Distro デプロイは、AWS サポートプランの対象外です。

Amazon EKS を使用するようにセットアップする

Amazon EKS クラスターのコマンドライン管理を準備するには、いくつかのツールをインストールする必要があります。以下を使用して、認証情報の設定、クラスターの作成と変更、クラスターの実行後の操作を行います。

- AWS CLI のセットアップ – AWS CLI を入手して、Amazon EKS クラスターを操作するために必要なサービスをセットアップおよび管理します。特に、AWS CLI で認証情報を設定する必要がありますが、これは他の AWS サービスでも必要になります。
- kubectl と eksctl のセットアップ – `eksctl` CLI は AWS とやり取りして、Amazon EKS クラスターを作成、変更、削除します。クラスターが起動したら、オープンソース `kubectl` コマンドを使用して Amazon EKS クラスター内の Kubernetes オブジェクトを管理します。
- 開発環境をセットアップする (オプション) – 以下のツールを追加することを検討してください:
  - ローカルデプロイツール – Kubernetes を初めて使用する場合は、minikube や kind などのローカルデプロイツールをインストールすることを検討してください。これらのツールを使用すると、アプリケーションをテストするためにローカルマシン上に Amazon EKS クラスターを配置できます。
  - パッケージマネージャー – Helm は複雑なパッケージのインストールと管理を簡素化する Kubernetes の一般的なパッケージマネージャーです。Helm を使用すると、Amazon EKS クラスターの AWS Load Balancer Controller などのパッケージを簡単にインストールして管理できます。

次のステップ

- AWS CLI のセットアップ
- kubectl と eksctl のセットアップ
- クイックスタート: ウェブアプリケーションをデプロイしてデータを保存する

AWS CLI の設定

AWS CLI は、Amazon EKS など AWS のサービスを操作するためのコマンドラインツールです。また、ローカルマシンから Amazon EKS クラスターやその他の AWS リソースにアクセスするための IAM ユーザーまたはロールの認証にも使用されます。コマンドラインから AWS のリソースをプロビジョニングするには、コマンドラインで使用する AWS アクセスキー ID とシークレットキーを取

得する必要があります。次に、これらの認証情報を AWS CLI で設定する必要があります。AWS CLI をまだインストールしていない場合は、AWS コマンドラインインターフェイスユーザーガイドの「AWS CLI の最新バージョンをインストールまたは更新する」を参照してください。

アクセスキーを作成するには

1. にサインインします。。AWS マネジメントコンソール
2. シングルユーザーアカウントまたはマルチユーザーアカウントの場合:
   - シングルユーザーアカウント –:: 右上の AWS ユーザー名を選択し、ナビゲーションメニューを開きます。例えば、[**webadmin**] を選択します。
   - マルチユーザーアカウント –:: サービスのリストから IAM を選択します。IAM ダッシュボードから [ユーザー] を選択し、ユーザーの名前を選択します。
3. [Security credentials] (セキュリティ認証情報) を選択します。
4. [アクセスキー] の下で、 [アクセスキーの作成] を選択します。
5. [コマンドラインインターフェイス (CLI)] を選択し、[次へ] を選択します。
6. [アクセスキーを作成] を選択します。
7. [.csv ファイルをダウンロード] を選択します。

AWS CLI を構成するには

AWS CLI をインストールしたら、以下の手順に従って設定を行います。詳細については、AWS コマンドラインインターフェイスユーザーガイドの「AWS CLI を設定する」を参照してください。

1. ターミナルウィンドウで、以下のコマンドを入力します。

   ```
   aws configure
   ```

   オプションで、`--profile cluster-admin` などの名前付きプロファイルを設定できます。AWS CLI で名前付きプロファイルを設定する場合、以降のコマンドでは必ずこのフラグを渡す必要があります。

2. AWS 認証情報を入力します。例えば、次のようになります。

   ```
   Access Key ID [None]: AKIAIOSFODNN7EXAMPLE
   Secret Access Key [None]: wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY
   Default region name [None]: region-code
   ```

```
Default output format [None]: json
```

セキュリティトークンを取得するには

必要に応じて、次のコマンドを実行して AWS CLI の新しいセキュリティトークンを取得します。詳細については、AWS CLI コマンドリファレンスの「get-session-token」を参照してください。

デフォルトでは、トークンは 15 分間有効です。デフォルトのセッションタイムアウトを変更するには、--duration-seconds フラグを渡します。例えば、次のようになります。

```
aws sts get-session-token --duration-seconds 3600
```

このコマンドは、AWS CLI セッションの一時的なセキュリティ認証情報を返します。以下のようなレスポンス出力が表示されます。

```
{
    "Credentials": {
        "AccessKeyId": "ASIA5FTRU3LOEXAMPLE",
        "SecretAccessKey": "JnKgvwfqUD9mNsPoi9IbxAYEXAMPLE",
        "SessionToken": "VERYLONGSESSIONTOKENSTRING",
        "Expiration": "2023-02-17T03:14:24+00:00"
    }
}
```

ユーザー ID を確認するには

必要に応じて、次のコマンドを実行して、ターミナルセッションの IAM ユーザー ID (*ClusterAdmin* など) の AWS 認証情報を確認します。

```
aws sts get-caller-identity
```

このコマンドは、AWS CLI に設定されている IAM エンティティの Amazon リソースネーム (ARN) を返します。以下のようなレスポンス出力例が表示されます。

```
{
    "UserId": "AKIAIOSFODNN7EXAMPLE",
    "Account": "01234567890",
    "Arn": "arn:aws:iam::01234567890:user/ClusterAdmin"
}
```

次のステップ

- kubectl と eksctl のセットアップ
- クイックスタート: ウェブアプリケーションをデプロイしてデータを保存する

kubectl および eksctl のセットアップ

> **Tip**
>
> 今後開催予定の Amazon EKS ワークショップに 登録 してください。

AWS CLI をインストールしたら、Kubernetes クラスターを作成し管理するためのツールをあと 2 つインストールします。

- `kubectl`: `kubectl` コマンドラインツールは、Kubernetes クラスター内のリソースを管理するときに使用する主要ツールです。このページでは、Kubernetes クラスターのバージョンに一致する `kubectl` バイナリを、ダウンロードしてセットアップする方法について説明します。kubectl をインストールまたは更新する を参照してください。
- `eksctl`: `eksctl` コマンドラインツールは、AWS クラウドまたはオンプレミス (EKS Anywhere を使用) で EKS クラスターを作成するため、ならびにそれらのクラスターを変更および削除するために作られています。「eksctl をインストールする」を参照してください。

kubectl をインストールまたは更新する

このトピックは、デバイス上の `kubectl` バイナリのダウンロードおよびインストール、または更新に役立ちます。バイナリは アップストリームコミュニティのバージョン と同様です。バイナリは Amazon EKS または AWS に固有のものではありません。多くのビルダーはインストールのために `brew install kubectl` を実行するだけですが、必要な `kubectl` の特定のバージョンを取得するには、以下のステップを使用します。

> **Note**
>
> Amazon EKS クラスターコントロールプレーンとのマイナーバージョンの相違が 1 つ以内である `kubectl` バージョンを使用する必要があります。例えば、`1.36` `kubectl` クライアントは Kubernetes `1.35`、`1.36`、および `1.37` クラスターで動作します。

> **Tip**
>
> AWS CloudShell 経由で EKS コンソールから kubectl を直接使用することもできます。任意のクラスターの詳細ページで [接続] を選択して、事前設定されたセッションを取得します。詳細については、「the section called “kubectl を使用してクラスターにアクセスする”」を参照してください。

ステップ 1: kubectl がインストールされているかどうかを確認する

デバイスに kubectl が既にインストールされているかどうかを判断します。

```
kubectl version --client
```

kubectl をデバイスのパスにインストールした場合、出力例には以下のような情報が含まれます。現在インストールされているバージョンを新しいバージョンで更新する場合は、次のステップを完了し、新しいバージョンを現在のバージョンと同じ場所にインストールするようにします。

```
Client Version: v1.31.X-eks-1234567
```

出力が表示されない場合は、kubectl がインストールされていないか、デバイスのパス内の場所にインストールされていません。

ステップ 2: kubectl をインストールまたは更新する

次のオペレーティングシステムのいずれかで kubectl をインストールまたは更新します。

- the section called “macOS”
- the section called “Linux (amd64)”
- the section called “Linux (arm64)”
- the section called “Windows”

> **Note**
>
> このセクションで使用されている AWS リージョンからご使用の AWS リージョンへのダウンロードに時間がかかる場合は、コンテンツを前面に表示するように CloudFront を設

定することを検討してください。詳細については、「Get started with a basic CloudFront distribution」を参照してください。

macOS

kubectl を macOS にインストールするには、以下の手順に従います。ステップには以下が含まれます。

- 必要な Kubernetes バージョンのバイナリを選択してダウンロードします。
- 必要に応じて、バイナリのチェックサムを確認します。
- バイナリのアクセス許可に実行を追加します。
- バイナリを PATH のフォルダにコピーします。
- 必要に応じて、バイナリのディレクトリを PATH に追加します。

手順:

1. クラスターの Kubernetes バージョンのバイナリを Amazon S3 からダウンロードします。
   - Kubernetes 1.36

     ```
     curl -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.36.2/2026-07-05/bin/darwin/amd64/kubectl
     ```

   - Kubernetes 1.35

     ```
     curl -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.35.6/2026-07-05/bin/darwin/amd64/kubectl
     ```

   - Kubernetes 1.34

     ```
     curl -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.34.9/2026-07-05/bin/darwin/amd64/kubectl
     ```

   - Kubernetes 1.33

     ```
     curl -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.33.13/2026-07-05/bin/darwin/amd64/kubectl
     ```

   - Kubernetes 1.32

```
curl -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.32.13/2026-07-05/bin/
darwin/amd64/kubectl
```

- Kubernetes `1.31`

```
curl -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.31.14/2026-07-05/bin/
darwin/amd64/kubectl
```

- Kubernetes `1.30`

```
curl -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.30.14/2026-07-05/bin/
darwin/amd64/kubectl
```

2. (オプション) ダウンロードしたバイナリを、バイナリの SHA-256 チェックサムで検証します。

   a. クラスターの Kubernetes バージョンの SHA-256 チェックサムをダウンロードします。

   - Kubernetes `1.36`

```
curl -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.36.2/2026-07-05/bin/
darwin/amd64/kubectl.sha256
```

   - Kubernetes `1.35`

```
curl -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.35.6/2026-07-05/bin/
darwin/amd64/kubectl.sha256
```

   - Kubernetes `1.34`

```
curl -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.34.9/2026-07-05/bin/
darwin/amd64/kubectl.sha256
```

   - Kubernetes `1.33`

```
curl -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.33.13/2026-07-05/bin/
darwin/amd64/kubectl.sha256
```

   - Kubernetes `1.32`

```
curl -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.32.13/2026-07-05/bin/
darwin/amd64/kubectl.sha256
```

- Kubernetes 1.31

```
curl -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.31.14/2026-07-05/bin/darwin/amd64/kubectl.sha256
```

- Kubernetes 1.30

```
curl -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.30.14/2026-07-05/bin/darwin/amd64/kubectl.sha256
```

b. ダウンロードしたバイナリの SHA-256 チェックサムを確認します。

```
openssl sha1 -sha256 kubectl
```

c. 出力で生成されたチェックサムが、ダウンロードした kubectl.sha256 ファイルのチェックサムと一致することを確認してください。

3. バイナリへの実行アクセス許可を適用します。

```
chmod +x ./kubectl
```

4. バイナリを PATH のフォルダにコピーします。既に kubectl のバージョンが既にインストールされている場合、$HOME/bin/kubectl を作成し、$HOME/bin が $PATH の最初にあることを確認することをお勧めします。

```
mkdir -p $HOME/bin && cp ./kubectl $HOME/bin/kubectl && export PATH=$HOME/bin:$PATH
```

5. (オプション) シェルを開いたときに設定されるように、シェルの初期化ファイルに $HOME/bin パスを追加します。

```
echo 'export PATH=$HOME/bin:$PATH' >> ~/.bash_profile
```

Linux (amd64)

Linux (amd64) に kubectl をインストールするには、以下の手順に従います。ステップには以下が含まれます。

- 必要な Kubernetes バージョンのバイナリを選択してダウンロードします。
- 必要に応じて、バイナリのチェックサムを確認します。

- バイナリのアクセス許可に実行を追加します。
- バイナリを PATH のフォルダにコピーします。
- 必要に応じて、バイナリのディレクトリを PATH に追加します。

手順:

1. クラスターの Kubernetes バージョンの `kubectl` バイナリを Amazon S3 からダウンロードします。
   - Kubernetes `1.36`

     ```
     curl -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.36.2/2026-07-05/bin/linux/amd64/kubectl
     ```

   - Kubernetes `1.35`

     ```
     curl -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.35.6/2026-07-05/bin/linux/amd64/kubectl
     ```

   - Kubernetes `1.34`

     ```
     curl -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.34.9/2026-07-05/bin/linux/amd64/kubectl
     ```

   - Kubernetes `1.33`

     ```
     curl -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.33.13/2026-07-05/bin/linux/amd64/kubectl
     ```

   - Kubernetes `1.32`

     ```
     curl -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.32.13/2026-07-05/bin/linux/amd64/kubectl
     ```

   - Kubernetes `1.31`

     ```
     curl -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.31.14/2026-07-05/bin/linux/amd64/kubectl
     ```

   - Kubernetes `1.30`

```
curl -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.30.14/2026-07-05/bin/linux/amd64/kubectl
```

2. (オプション) ダウンロードしたバイナリを、バイナリの SHA-256 チェックサムで検証します。

   a. デバイスのハードウェアプラットフォームのコマンドを使用して、クラスターの Kubernetes バージョンの SHA-256 チェックサムを Amazon S3 からダウンロードします。

   - Kubernetes 1.36

     ```
     curl -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.36.2/2026-07-05/bin/linux/amd64/kubectl.sha256
     ```

   - Kubernetes 1.35

     ```
     curl -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.35.6/2026-07-05/bin/linux/amd64/kubectl.sha256
     ```

   - Kubernetes 1.34

     ```
     curl -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.34.9/2026-07-05/bin/linux/amd64/kubectl.sha256
     ```

   - Kubernetes 1.33

     ```
     curl -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.33.13/2026-07-05/bin/linux/amd64/kubectl.sha256
     ```

   - Kubernetes 1.32

     ```
     curl -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.32.13/2026-07-05/bin/linux/amd64/kubectl.sha256
     ```

   - Kubernetes 1.31

     ```
     curl -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.31.14/2026-07-05/bin/linux/amd64/kubectl.sha256
     ```

   - Kubernetes 1.30

```
curl -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.30.14/2026-07-05/bin/
linux/amd64/kubectl.sha256
```

b. 次のいずれかのコマンドを使用して、ダウンロードしたバイナリの SHA-256 チェックサムを確認します。

```
sha256sum -c kubectl.sha256
```

または

```
openssl sha1 -sha256 kubectl
```

c. まず kubectl: OK と表示されることを確認します。その後、出力されたチェックサムが、ダウンロードした kubectl.sha256 ファイルのチェックサムと一致することを確認できます。

3. バイナリへの実行アクセス許可を適用します。

```
chmod +x ./kubectl
```

4. バイナリを PATH のフォルダにコピーします。既に kubectl のバージョンが既にインストールされている場合、$HOME/bin/kubectl を作成し、$HOME/bin が $PATH の最初にあることを確認することをお勧めします。

```
mkdir -p $HOME/bin && cp ./kubectl $HOME/bin/kubectl && export PATH=$HOME/bin:$PATH
```

5. (オプション) シェルを開いたときに設定されるように、シェルの初期化ファイルに $HOME/bin パスを追加します。

> **Note**
>
> このステップでは、Bash シェルを使用していることを前提としています。別のシェルを使用している場合は、特定のシェル初期化ファイルを使用するよう、コマンドを変更します。

```
echo 'export PATH=$HOME/bin:$PATH' >> ~/.bashrc
```

Linux (arm64)

Linux (arm64) に kubectl をインストールするには、以下の手順に従います。ステップには以下が含まれます。

- 必要な Kubernetes バージョンのバイナリを選択してダウンロードします。
- 必要に応じて、バイナリのチェックサムを確認します。
- バイナリのアクセス許可に実行を追加します。
- バイナリを PATH のフォルダにコピーします。
- 必要に応じて、バイナリのディレクトリを PATH に追加します。

手順:

1. クラスターの Kubernetes バージョンの kubectl バイナリを Amazon S3 からダウンロードします。
   - Kubernetes 1.36

     ```
     curl -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.36.2/2026-07-05/bin/linux/arm64/kubectl
     ```

   - Kubernetes 1.35

     ```
     curl -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.35.6/2026-07-05/bin/linux/arm64/kubectl
     ```

   - Kubernetes 1.34

     ```
     curl -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.34.9/2026-07-05/bin/linux/arm64/kubectl
     ```

   - Kubernetes 1.33

     ```
     curl -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.33.13/2026-07-05/bin/linux/arm64/kubectl
     ```

   - Kubernetes 1.32

     ```
     curl -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.32.13/2026-07-05/bin/linux/arm64/kubectl
     ```

- Kubernetes 1.31

```
curl -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.31.14/2026-07-05/bin/linux/arm64/kubectl
```

- Kubernetes 1.30

```
curl -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.30.14/2026-07-05/bin/linux/arm64/kubectl
```

2. (オプション) ダウンロードしたバイナリを、バイナリの SHA-256 チェックサムで検証します。
   a. デバイスのハードウェアプラットフォームのコマンドを使用して、クラスターの Kubernetes バージョンの SHA-256 チェックサムを Amazon S3 からダウンロードします。
      - Kubernetes 1.36

```
curl -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.36.2/2026-07-05/bin/linux/arm64/kubectl.sha256
```

      - Kubernetes 1.35

```
curl -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.35.6/2026-07-05/bin/linux/arm64/kubectl.sha256
```

      - Kubernetes 1.34

```
curl -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.34.9/2026-07-05/bin/linux/arm64/kubectl.sha256
```

      - Kubernetes 1.33

```
curl -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.33.13/2026-07-05/bin/linux/arm64/kubectl.sha256
```

      - Kubernetes 1.32

```
curl -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.32.13/2026-07-05/bin/linux/arm64/kubectl.sha256
```

      - Kubernetes 1.31

```
curl -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.31.14/2026-07-05/bin/
linux/arm64/kubectl.sha256
```

- Kubernetes `1.30`

```
curl -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.30.14/2026-07-05/bin/
linux/arm64/kubectl.sha256
```

b. 次のいずれかのコマンドを使用して、ダウンロードしたバイナリの SHA-256 チェックサムを確認します。

```
sha256sum -c kubectl.sha256
```

または

```
openssl sha1 -sha256 kubectl
```

c. まず `kubectl: OK` と表示されることを確認します。その後、出力されたチェックサムが、ダウンロードした `kubectl.sha256` ファイルのチェックサムと一致することを確認できます。

3. バイナリへの実行アクセス許可を適用します。

```
chmod +x ./kubectl
```

4. バイナリを PATH のフォルダにコピーします。既に `kubectl` のバージョンが既にインストールされている場合、`$HOME/bin/kubectl` を作成し、`$HOME/bin` が `$PATH` の最初にあることを確認することをお勧めします。

```
mkdir -p $HOME/bin && cp ./kubectl $HOME/bin/kubectl && export PATH=$HOME/bin:$PATH
```

5. (オプション) シェルを開いたときに設定されるように、シェルの初期化ファイルに `$HOME/bin` パスを追加します。

> **Note**
>
> このステップでは、Bash シェルを使用していることを前提としています。別のシェルを使用している場合は、特定のシェル初期化ファイルを使用するよう、コマンドを変更します。

```
echo 'export PATH=$HOME/bin:$PATH' >> ~/.bashrc
```

Windows

kubectl を Windows にインストールするには、以下の手順に従います。ステップには以下が含まれます。

- 必要な Kubernetes バージョンのバイナリを選択してダウンロードします。
- 必要に応じて、バイナリのチェックサムを確認します。
- バイナリを PATH のフォルダにコピーします。
- 必要に応じて、バイナリのディレクトリを PATH に追加します。

手順:

1. PowerShell ターミナルを開きます。
2. クラスターの Kubernetes バージョンの kubectl バイナリを Amazon S3 からダウンロードします。
   - Kubernetes 1.36

     ```
     curl.exe -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.36.2/2026-07-05/bin/windows/amd64/kubectl.exe
     ```

   - Kubernetes 1.35

     ```
     curl.exe -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.35.6/2026-07-05/bin/windows/amd64/kubectl.exe
     ```

   - Kubernetes 1.34

     ```
     curl.exe -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.34.9/2026-07-05/bin/windows/amd64/kubectl.exe
     ```

   - Kubernetes 1.33

```
curl.exe -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.33.13/2026-07-05/bin/windows/amd64/kubectl.exe
```

- Kubernetes `1.32`

```
curl.exe -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.32.13/2026-07-05/bin/windows/amd64/kubectl.exe
```

- Kubernetes `1.31`

```
curl.exe -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.31.14/2026-07-05/bin/windows/amd64/kubectl.exe
```

- Kubernetes `1.30`

```
curl.exe -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.30.14/2026-07-05/bin/windows/amd64/kubectl.exe
```

3. (オプション) ダウンロードしたバイナリを、バイナリの SHA-256 チェックサムで検証します。
   a. クラスターの Kubernetes バージョンに応じて、SHA-256 チェックサム (Windows 版) をダウンロードします。
      - Kubernetes `1.36`

```
curl.exe -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.36.2/2026-07-05/bin/windows/amd64/kubectl.exe.sha256
```

      - Kubernetes `1.35`

```
curl.exe -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.35.6/2026-07-05/bin/windows/amd64/kubectl.exe.sha256
```

      - Kubernetes `1.34`

```
curl.exe -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.34.9/2026-07-05/bin/windows/amd64/kubectl.exe.sha256
```

      - Kubernetes `1.33`

```
curl.exe -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.33.13/2026-07-05/
bin/windows/amd64/kubectl.exe.sha256
```

- Kubernetes `1.32`

```
curl.exe -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.32.13/2026-07-05/
bin/windows/amd64/kubectl.exe.sha256
```

- Kubernetes `1.31`

```
curl.exe -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.31.14/2026-07-05/
bin/windows/amd64/kubectl.exe.sha256
```

- Kubernetes `1.30`

```
curl.exe -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.30.14/2026-07-05/
bin/windows/amd64/kubectl.exe.sha256
```

b. ダウンロードしたバイナリの SHA-256 チェックサムを確認します。

```
Get-FileHash kubectl.exe
```

c. 出力で生成されたチェックサムが、ダウンロードした `kubectl.exe.sha256` ファイルのチェックサムと一致することを確認してください。PowerShell の出力は、大文字の同じ文字列である必要があります。

4. バイナリを PATH のフォルダにコピーします。コマンドラインユーティリティで使用する PATH 内に既存のディレクトリがある場合、そのディレクトリにバイナリをコピーします。それ以外の場合は、以下の手順を完了します。

a. コマンドラインのバイナリ用に新しいディレクトリ (`C:\bin` など) を作成します。

b. `kubectl.exe` バイナリを新しいディレクトリにコピーします。

c. ユーザーまたはシステムの PATH 環境変数を編集し、新しいディレクトリを PATH に追加します。

d. 新しい PATH 変数を取得するには、PowerShell ターミナルを閉じ、新しいターミナルを開きます。

5. `kubectl` をインストールしたら、そのバージョンを確認できます。

```
kubectl version --client
```

6. kubectl を初めてインストールする場合、まだどのサーバーとも通信するように設定されていません。この設定については、必要に応じて他の手順で説明します。特定のクラスターと通信するために設定を更新する必要が生じた場合は、以下のコマンドを実行できます。*region-code* を、クラスターのある AWS リージョンに置き換えます。*my-cluster* の部分は自分のクラスター名に置き換えます。

```
aws eks update-kubeconfig --region region-code --name my-cluster
```

7. 最初の数文字を入力した後タブキーを使用することで kubectl サブコマンドを完全入力できる、オートコンプリートの設定を検討してください。詳細については、Kubernetes ドキュメントの「Kubectl autocomplete」を参照してください。

eksctl をインストールする

eksctl CLI は EKS クラスターの操作に使用されます。個々のタスクの多くを自動化します。eksctl のインストール手順については、eksctl ドキュメントの「インストール」を参照してください。Linux の場合は、UNIX の手順を使用します。

eksctl を使用する場合、使用している IAM セキュリティプリンシパルは、Amazon EKS の IAM ロール、サービスにリンクされたロール、AWS CloudFormation、VPC、関連リソースを使用するために許可が必要です。詳細については、IAM ユーザーガイドの「アクション」および「サービスにリンクされたロールの使用」を参照してください。このガイドのすべてのステップは1 つのユーザーとして実行する必要があります。現在のユーザーを確認するには次のコマンドを実行してください。

```
aws sts get-caller-identity
```

次のステップ

- クイックスタート: ウェブアプリケーションをデプロイしてデータを保存する

クイックスタート: ウェブアプリケーションをデプロイしてデータを保存する

このクイックスタートチュートリアルでは、eksctl を使用して 2048 ゲームサンプルアプリケーションをデプロイし、そのデータを Amazon EKS Auto Mode クラスターに保持する手順について説明します。

Amazon EKS Auto Mode は、ブロックストレージ、ネットワーク、ロードバランシング、コンピューティングの自動スケーリングなどのルーチンタスクを自動化することで、クラスター管理を簡素化します。セットアップ中、EC2 マネージドインスタンス、Application Load Balancer、EBS ボリュームを使用したノードの作成を処理します。

要約すると、AWS サービスとのシームレスな統合に必要なカスタム注釈を含むサンプルワークロードをデプロイします。

このチュートリアルでは

次の `eksctl` クラスターテンプレートを使用して、自動ノードプロビジョニング用の EKS 自動モードでクラスターを構築します。

- VPC 設定: 以下の eksctl クラスターテンプレートを使用すると、eksctl はクラスターの IPv4 仮想プライベートクラウド (VPC) を自動的に作成します。デフォルトでは、eksctl はパブリックエンドポイントとプライベートエンドポイントの両方を作成するだけでなく、すべてのネットワーク要件に対応する VPC を設定します。
- インスタンス管理: EKS Auto Mode は、Kubernetes アプリケーションの需要に基づいて、EKS クラスター内のノードを動的に追加または削除します。
- データ永続性: EKS Auto Mode のブロックストレージ機能を使用して、ポッドの再起動や障害が発生するシナリオでも、アプリケーションデータの永続性を確保します。
- 外部アプリケーションアクセス: EKS Auto Mode の負荷分散機能を使用して、Application Load Balancer (ALB) を動的にプロビジョニングします。

前提条件

始める前に、次のタスクを実行していることを確認してください。

- Amazon EKS の環境をセットアップする

- eksctl の最新バージョンをインストールする

クラスターを設定する

このセクションでは、動的ノードプロビジョニングに EKS Auto Mode を使用してクラスターを作成します。

`cluster-config.yaml` ファイルを作成して、次の内容を貼り付けます。`region-code` を有効なリージョン (例: `us-east-1`) に置き換えます。

```
apiVersion: eksctl.io/v1alpha5
kind: ClusterConfig

metadata:
  name: web-quickstart
  region: <region-code>

autoModeConfig:
  enabled: true
```

これで、クラスターを作成する準備が整いました。

`cluster-config.yaml` を使用して EKS クラスターを作成します。

```
eksctl create cluster -f cluster-config.yaml
```

> **Important**
>
> eksctl を使用してクラスターを作成しない場合は、VPC サブネットに手動でタグ付けする必要があります。

IngressClass を作成する

EKS Auto Mode の Kubernetes `IngressClass` を作成します。IngressClass は、EKS Auto Mode が Ingress リソースを処理する方法を定義します。このステップでは、EKS Auto Mode の負荷分散機能を設定します。アプリケーションの Ingress リソースを作成すると、EKS Auto Mode はこの IngressClass を使用してロードバランサーを自動的にプロビジョニングおよび管理し、Kubernetes アプリケーションを AWS ロードバランシングサービスと統合します。

次の yaml ファイルを ingressclass.yaml として保存します。

```
apiVersion: networking.k8s.io/v1
kind: IngressClass
metadata:
  name: alb
  annotations:
    ingressclass.kubernetes.io/is-default-class: "true"
spec:
  controller: eks.amazonaws.com/alb
```

IngressClass をクラスターに適用します。

```
kubectl apply -f ingressclass.yaml
```

2048 ゲームサンプルアプリケーションをデプロイする

このセクションでは、人気のある「2048 ゲーム」をクラスター内のサンプルアプリケーションとしてデプロイする手順について説明します。提供されたマニフェストには、Application Load Balancer (ALB) のカスタム注釈が含まれています。これらの注釈は EKS と統合され、受信 HTTP トラフィックを「internet-facing」として処理し、ターゲットタイプ「ip」を使用して game-2048 名前空間の適切なサービスにルーティングするように指示します。

> **Note**
>
> この例の docker-2048 イメージは x86_64 コンテナイメージであり、他のアーキテクチャでは実行できません。

1. --save-config フラグを付けて game-2048 という Kubernetes 名前空間を作成します。

   ```
   kubectl create namespace game-2048 --save-config
   ```

   以下のようなレスポンス出力が表示されます。

   ```
   namespace/game-2048 created
   ```

2. 2048 ゲームサンプルアプリケーションをデプロイします。

```
kubectl apply -n game-2048 -f https://raw.githubusercontent.com/kubernetes-sigs/aws-
load-balancer-controller/v2.8.0/docs/examples/2048/2048_full.yaml
```

このマニフェストは、game-2048 名前空間の Kubernetes デプロイ、サービス、およびイングレスをセットアップし、クラスター内で game-2048 アプリケーションをデプロイして公開するために必要なリソースを作成します。これには、ポート 80 でのデプロイを公開する service-2048 という名前のサービスの作成と、受信 HTTP トラフィックのルーティングルールと internet-facing Application Load Balancer (ALB) の注釈を定義する ingress-2048 という名前のイングレスリソースの作成が含まれます。以下のようなレスポンス出力が表示されます。

```
namespace/game-2048 configured
deployment.apps/deployment-2048 created
service/service-2048 created
ingress.networking.k8s.io/ingress-2048 created
```

3. 次のコマンドを実行して、game-2048 名前空間のイングレスリソースを取得します。

```
kubectl get ingress -n game-2048
```

以下のようなレスポンス出力が表示されます。

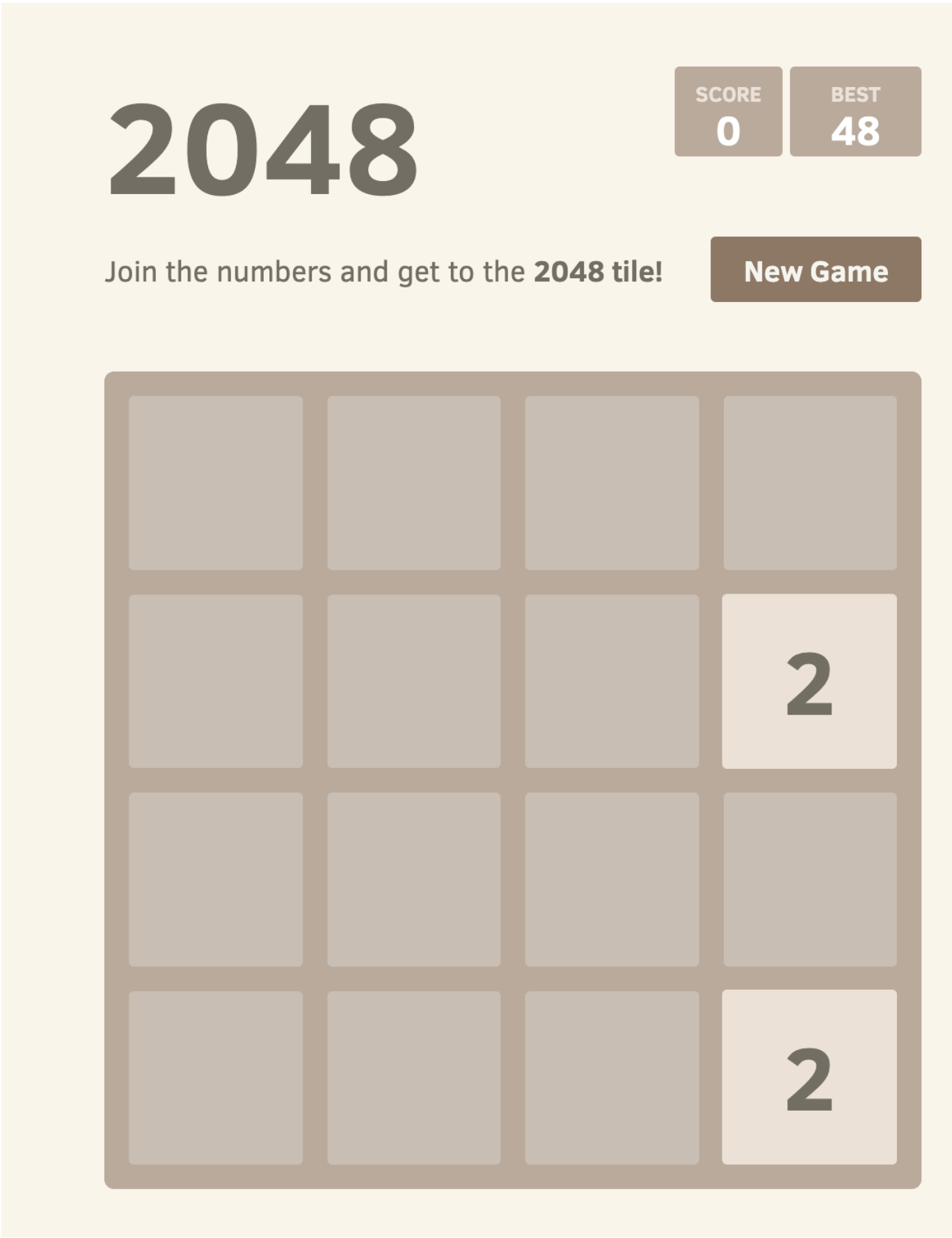
```
NAME           CLASS   HOSTS   ADDRESS
                      PORTS   AGE
ingress-2048   alb     *       k8s-game2048-ingress2-eb379a0f83-378466616.region-
code.elb.amazonaws.com   80      31s
```

Application Load Balancer (ALB) がプロビジョニングされるまで数分待ってから、次のステップを開始する必要があります。

4. ウェブブラウザを開き、前のステップの ADDRESS を入力してウェブアプリケーションにアクセスします。例えば、次のようになります。

```
k8s-game2048-ingress2-eb379a0f83-378466616.region-code.elb.amazonaws.com
```

ブラウザに 2048 ゲームが表示されるはずです。プレイしてください!


2048
SCORE
0
BEST
48
Join the numbers and get to the 2048 tile!
New Game
2
2

Amazon EKS 自動モードを使用してデータを永続化する

2048 ゲームが Amazon EKS クラスターで稼働しているので、Amazon EKS Auto Mode のブロックストレージ機能を使用してゲームデータが安全に保持されるようにします。

1. storage-class.yaml という名前のファイルを作成します：

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: auto-ebs-sc
  annotations:
    storageclass.kubernetes.io/is-default-class: "true"
provisioner: ebs.csi.eks.amazonaws.com
volumeBindingMode: WaitForFirstConsumer
parameters:
  type: gp3
  encrypted: "true"
```

2. StorageClass を適用します。

```
kubectl apply -f storage-class.yaml
```

3. ゲームデータのストレージをリクエストするには、永続ボリュームのクレーム (PVC) を作成します。ebs-pvc.yaml という名前のファイルを作成し、そのファイルに以下の内容を追加します。

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: game-data-pvc
  namespace: game-2048
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 10Gi
  storageClassName: auto-ebs-sc
```

4. PVC をクラスターに適用します。

```
kubectl apply -f ebs-pvc.yaml
```

以下のようなレスポンス出力が表示されます。

```
persistentvolumeclaim/game-data-pvc created
```

5. 次に、データの保存にこの PVC を使用するように 2048 ゲームデプロイを更新する必要があります。次のデプロイは、ゲームデータの保存に PVC を使用するように設定されています。ebs-deployment.yaml という名前のファイルを作成し、そのファイルに以下の内容を追加します。

```
apiVersion: apps/v1
kind: Deployment
metadata:
  namespace: game-2048
  name: deployment-2048
spec:
  replicas: 3  # Adjust the number of replicas as needed
  selector:
    matchLabels:
      app.kubernetes.io/name: app-2048
  template:
    metadata:
      labels:
        app.kubernetes.io/name: app-2048
    spec:
      containers:
        - name: app-2048
          image: public.ecr.aws/l6m2t8p7/docker-2048:latest
          imagePullPolicy: Always
          ports:
            - containerPort: 80
          volumeMounts:
            - name: game-data
              mountPath: /var/lib/2048
      volumes:
        - name: game-data
          persistentVolumeClaim:
            claimName: game-data-pvc
```

6. 更新されたデプロイを適用します。

```
kubectl apply -f ebs-deployment.yaml
```

以下のようなレスポンス出力が表示されます。

```
deployment.apps/deployment-2048 configured
```

これらのステップでは、クラスター上の 2048 ゲームが、Amazon EKS Auto Mode のブロックストレージ機能を使用してデータを保持するように設定されました。これにより、ポッドやノードに障害が発生した場合でも、ゲームの進行状況とデータは安全になります。

このチュートリアルが気に入った場合は、フィードバックを提供してお知らせください。これにより、このようなユースケース固有のクイックスタートチュートリアルをより多く提供できるようになります。

クリーンアップ

今後の料金が発生しないようにするには、関連する CloudFormation スタックを手動で削除して、VPC ネットワークを含め、このガイドで作成したすべてのリソースを削除する必要があります。

CloudFormation スタックを削除します。

```
eksctl delete cluster -f ./cluster-config.yaml
```

Amazon EKS を例で学ぶ

概要

このAmazon EKS ユーザーガイドには、コマンドラインまたは AWS マネジメントコンソール から最初の EKS クラスターを作成する汎用手順と、すべての主要な Amazon EKS コンポーネントに関する強固なリファレンスが含まれています。ただし、Amazon EKS クラスターの管理者またはデベロッパーは、このガイド以外のサイトに存在するラーニングパスに従うことで、Amazon EKS をより深く理解できます。これらのサイトは以下に役立ちます。

- 特定のタイプのクラスターをセットアップします。特定のクラスタータイプは、ワークロードタイプまたはセキュリティ要件に基づいて設定できます。例えば、クラスターを調整して、バッチ、機械学習、または計算負荷の高いワークロードを実行することができます。
- クラスターを強化します。クラスターに高度な機能を追加して、オブザーバビリティ、柔軟なストレージ、自動スケーリング、特殊なクラスターネットワークなどを提供できます。
- 更新を自動化します。GitOps などの機能を使用すると、Git リポジトリ内のコンポーネントに発生した変更に基づいて、クラスターインフラストラクチャとワークロードを自動的にプロビジョニングするようにセットアップできます。
- 高度なクラスターセットアップツールを使用します。`eksctl` ではクラスターをすばやく作成できますが、より複雑なクラスターの設定とアップグレードを容易にするツールが他にもあります。これには、Terraform や CloudFormation などのツールが含まれます。

Amazon EKS の学習パスを開始するには、このページで説明されているいくつかのサイトにアクセスすることをお勧めします。途中で問題が発生した場合は、その対処に役立つリソースもあります。例えば、Re:post ナレッジセンターでは、Amazon EKS 関連のサポートの問題についてサポートデータベースを検索できます。また、Amazon EKS ベストプラクティスガイドには、本番稼働レベルのクラスターをセットアップするための最適な方法に関するヒントが記載されています。

Amazon EKS ワークショップ

Amazon EKS ワークショップは、Kubernetes とコンテナの基本的な理解から始めて、Amazon EKS の重要な機能をクラスター管理者に説明する学習プラットフォームです。Amazon EKS ワークショップに参加する方法は次のとおりです。

- Amazon EKS の基本: Amazon EKS が AWS クラウドに Kubernetes 機能を実装する方法については、「Introduction」ページの動画をご覧ください。Kubernetes に関するより基本的な理解が必要な場合は、「What is Kubernetes」の動画をご覧ください。
- Amazon EKS のセットアップ: AWS アカウントをお持ちの場合、「Setup」セクションは、クラスターの作成に使用するための CloudShell 環境をセットアップするのに役立ちます。Amazon EKS クラスターを作成する際は、eksctl (シンプルなクラスター作成コマンドライン) と Terraform (クラスターを作成するためのより Infrastructure as Code のアプローチ) を選択できます。
- Amazon EKS の開始:「Sample application」セクションからシンプルなウェブストアを試してください。これは、他の演習全体で使用できます。このセクションでは、コンテナイメージのパッケージ化と、Kubernetes ポッド、デプロイ、サービス、StatefulSet、名前空間を使用したマイクロサービスの管理方法についても説明します。そして、Kustomize を使用して Kubernetes マニフェストに変更をデプロイします。
- Amazon EKS の基礎: このワークショップでは、AWS ロードバランサーコントローラー などの AWS 機能を使用して、アプリケーションを外部に公開する方法を説明します。ストレージについては、このワークショップでは、ブロックストレージに Amazon EBS、ファイルシステムストレージに Amazon EFS、および AWS で ONTAP ファイルシステムを管理するために Amazon FSx for NetApp ONTAP を使用する方法を紹介しています。ノード管理の場合、このワークショップはマネージドノードグループのセットアップに役立ちます。
- Amazon EKS の高度な機能: Amazon EKS ワークショップを通じて提供されるより高度な機能には、以下をセットアップするためのラボが含まれます。
  - 自動スケーリング: これには、ノードの自動スケーリング (Cluster Autoscaler または Karpenter を使用) とワークロードの自動スケーリング (Horizontal Pod Autoscaler と Cluster Proportional Autoscaler を使用) が含まれます。
  - オブザーバビリティ: 一連のオブザーバビリティラボでの、ログ記録、OpenSearch、Amazon EKS の Container Insights、Kubecost によるコストの可視化について説明します。
  - セキュリティ: この一連のセキュリティラボでは、シークレット管理、Amazon GuardDuty、ポッドセキュリティ標準、および Kyverno ポリシー管理について調べることができます。
  - ネットワーク: Amazon VPC CNI (ネットワークプラグインをサポート) と Amazon VPC Lattice (VPC とユーザーアカウント間でクラスターを設定するため) を含むネットワークラボから Amazon EKS のネットワーク機能について説明します。
  - オートメーション: オートメーションに関するラボでは、AWS Controllers for Kubernetes や Amazon EKS コントロールプレーンを管理する Crossplane など、クラスターやプロジェクトを管理する GitOps メソッドについて説明します。

Amazon EKS の実践的なクラスターセットアップチュートリアル

AWS コミュニティサイトにある一連の Amazon EKS クラスターセットアップチュートリアルは、専用 Amazon EKS クラスターを作成し、さまざまな方法でそのクラスターを強化するのに役立ちます。チュートリアルは 3 つの異なるタイプに分かれています。

クラスターの構築

これらのチュートリアルは、特別な目的に使用できるクラスターを構築するのに役立ちます。これらの特別な目的には、以下を実行する機能が含まれます。

- IPv6 に基づくグローバルにスケーラブルなアプリケーション
- 非同期バッチタスク
- トラフィック量の多いマイクロサービス
- Fargate での Karpenter による自動スケーリング
- 財務ワークロード
- Windows マネージドノードグループ

クラスターの強化

既存のクラスターがある場合は、特殊なワークロードを実行したり、複数のクラスターを強化したりできるように、そのクラスターを拡張および強化することができます。これらのチュートリアルには、以下の方法が含まれています。

- EFS CSI によるストレージソリューションを提供する
- EBS CSI による動的データベースストレージを提供する
- AWS ロードバランサーコントローラーを使用して IPv4 クラスターでアプリケーションを公開する
- AWS ロードバランサーコントローラーを使用して IPv6 クラスターでアプリケーションを公開する

AWS サービスの最適化

これらのチュートリアルを使用すると、クラスターを AWS サービスとより適切に統合できます。これらのチュートリアルには、以下に役立つチュートリアルが含まれています。

- ExternalDNS を使用してマイクロサービスの DNS レコードを管理する

- CloudWatch でアプリケーションをモニタリングする
- SQS および EFS ストレージを使用して非同期タスクを管理する
- ワークロードから AWS Secrets Manager シークレットを使用する
- Fargate、NGINX、および ACM PCA で mTLS をセットアップする

Amazon EKS のサンプル

Amazon EKS Samples リポジトリには、Amazon EKS で使用するマニフェストが保存されます。これらのマニフェストを使用すると、Amazon EKS でさまざまな種類のアプリケーションを試したり、特定のタイプの Amazon EKS クラスターを作成したりできます。サンプルには、以下のマニフェストが含まれます。

- AWS Amazon EKS Fargate クラスターを作成する
- 既存の IAM ロールを使用してクラスターを作成する
- クラスターに Ubuntu マネージドノードグループを追加する
- ボリュームスナップショットを使用してポッドストレージをバックアップし、復元する
- 複数のアカウントを持つ PVC としてマウントされた EBS ボリュームを復旧する
- Classic Load Balancer で NGINX Ingress Controller のプロキシプロトコルを有効にする
- Fargate でのログ記録を AWS OpenSearch に設定する
- ウェブフェデレーティッド ID プロバイダーで Python SDK を実行する
- NFS CSI コントローラーにサンプルアプリケーションをデプロイする
- StatefulSet のボリュームスナップショットを使用する
- 異なるアベイラビリティーゾーンのノード間でポッドをデプロイする

これらのサンプルは学習およびテストのみを目的としており、本番環境での使用は意図されていませんのでご注意ください。

AWS のチュートリアル

AWS Tutorials サイトでは、いくつかの Amazon EKS チュートリアルが公開されていますが、AWS サイト (AWS コミュニティサイトなど) で公開されている他のチュートリアルを検索するための検索ツールも用意されています。このサイトで直接公開されている Amazon EKS チュートリアルには以下が含まれます。

- 「Deploy a Container Web App on Amazon EKS」
- 「Run Kubernetes clusters for less (Amazon EKS and Spot instances)」
- How to cost optimize Jenkins jobs on Kubernetes

デベロッパーワークショップ

ソフトウェアデベロッパーが Amazon EKS で実行するアプリケーションを作成またはリファクタリングする場合は、「Amazon EKS Developers workshop」から始めることをお勧めします。このワークショップは、コンテナ化されたアプリケーションの構築に役立つだけでなく、これらのコンテナのコンテナレジストリ (ECR) へのデプロイや、そこから Amazon EKS クラスターへのデプロイにも役立ちます。

「Amazon EKS Python Workshop」から始めて、Python アプリケーションのリファクタリングプロセスを進め、アプリケーションのデプロイを準備するための開発環境をセットアップします。コンテナ、Kubernetes、Amazon EKS のセクションを順に実行して、コンテナ化されたアプリケーションをこれらの環境で実行する準備をします。

Terraform ワークショップ

`eksctl` はクラスターを作成するためのシンプルなツールですが、より複雑な Infrastructure as Code タイプの Amazon EKS デプロイの場合は、Terraform が一般的な Amazon EKS クラスター作成および管理ツールとなります。「Terraform Amazon EKS Workshop」では、Terraform を使用して AWS VPC を構築し、Amazon EKS クラスターを作成し、クラスターにオプションの拡張機能を追加する方法について説明します。特に、プライベート Amazon EKS クラスターを作成するためのセクションがあります。

AWS Amazon EKS トレーニング

AWS では、Amazon EKS について学習するための正式なトレーニングを提供しています。「Running Containers on Amazon Elastic Kubernetes Service」という 3 日間のトレーニングコースでは、次のことを学習します。

- Kubernetes と Amazon EKS の基礎
- Amazon EKS クラスターを構築する方法
- AWS IAM および Kubernetes RBAC 認可による Amazon EKS の保護
- GitOps 自動化ツール

- モニタリングツール
- コスト、効率、回復性を改善するためのテクニック

Amazon EKS の使用を開始する

入門ガイドを読む前に、Amazon EKS を使用するように設定されていることを確認してください。詳細については、「セットアップ」を参照してください。

Amazon EKS で、ノード付の新しい Kubernetes クラスターを作成して使用を開始する方法については、以下の 2 つの入門ガイドで解説しています。

- Amazon EKS – eksctl の使用開始 – この入門ガイドは、`eksctl` (Amazon EKS 上に Kubernetes クラスターを作成および管理するためのシンプルなコマンドラインユーティリティ) を使用して、Amazon EKS の開始に必要なすべてのリソースをインストールする方法を解説しています。チュートリアルを完了すると、アプリケーションのデプロイが可能な Amazon EKS クラスターが実行状態になります。このガイドの利用が、Amazon EKS の使用を開始するための最も迅速かつシンプルな手段です。
- Amazon EKS の使用を開始する – AWS マネジメントコンソール と AWS CLI – この入門ガイドは、AWS マネジメントコンソール と AWS CLI を使用して Amazon EKS を開始するために必要なすべてのリソースの作成に役立ちます。チュートリアルを完了すると、アプリケーションのデプロイが可能な Amazon EKS クラスターが実行状態になります。このガイドでは、Amazon EKS クラスターに必要なリソースを手動で個別に作成していきます。ここでの手順に従うことで、各リソースがどのように作成されたか、および、リソース間のやり取りがどのように行われているかなどを、把握できるようになります。

また、以下のリファレンスも提供しています。

- 実践的なチュートリアルのコレクションについては、AWS コミュニティにある EKS クラスターのセットアップに関する記事を参照してください。
- コード例については、「Code examples for Amazon EKS using AWS SDKs」を参照してください。

Amazon EKS – EKS Auto Mode の使用開始

他の EKS の開始方法エクスペリエンスと同様に、EKS Auto Mode で最初のクラスターを作成すると、クラスター自体の管理は AWS に委任されます。ただし、EKS Auto Mode は、ワークロードインフラストラクチャ (ノード、ネットワーク、およびさまざまなサービス) の設定に必要な多くの重要なサービスに責任を持たせることで EKS オートメーションを拡張するため、ノードの管理とワークロードの需要に合わせたスケールアップが容易になります。

EKS Auto Mode でクラスターを作成するには、次のいずれかの方法を選択します。

- the section called “ AWS CLI”: aws コマンドラインインターフェイスを使用してクラスターを作成します。
- the section called “マネジメントコンソール”:AWS マネジメントコンソール を使用してクラスターを作成します。
- the section called “eksctl CLI”: eksctl コマンドラインインターフェイスを使用してクラスターを作成します。

最初の EKS クラスターを作成するために異なるアプローチを比較する場合、EKS Auto Mode は、コンポーネントのセットアップを含むクラスター管理の責任を AWS に引き継がせる必要があることを知っておく必要があります。

- ワークロードの需要の増減に応じて、ノードを起動およびスケーリングすること。
- クラスター自体 (コントロールプレーン)、ノードのオペレーティングシステム、ノードで実行されているサービスを定期的にアップグレードすること。
- ノードのストレージのサイズや速度、ポッドネットワーク設定などを決定するデフォルト設定を選択すること。

EKS Auto Mode クラスターが提供する機能の詳細については、「EKS Auto Mode」を参照してください。

Amazon EKS – **eksctl** の使用を開始する

> **Note**
>
> このトピックでは、EKS Auto Mode を使用せずに開始する方法について説明します。EKS Auto Mode は、クラスターコンピューティング、ストレージ、ネットワーキングのルーチンタスクを自動化します。Amazon EKS Auto Mode の使用を開始する方法について説明します。

この入門ガイドでは、eksctl (Amazon EKS 上に Kubernetes クラスターを作成し管理するためのシンプルなコマンドラインユーティリティ) を使用して、Amazon Elastic Kubernetes Service (Amazon EKS) の開始に必要なすべてのリソースをインストールする方法を解説していきます。この

チュートリアルの終わりには、アプリケーションのデプロイが可能な、実行状態の Amazon EKS クラスターが完成します。

AWS マネジメントコンソール からクラスターを作成する場合には手動で作成する必要のあるリソースのいくつかが、このガイドの手順に従うことで自動的に作成されます。リソース間での連携方法について良く理解するために、大半のリソースを手動で作成する場合には、クラスターと計算機能の作成に AWS マネジメントコンソール を使用します。詳細については、「the section called “クラスターを作成する (コンソールと CLI)”」を参照してください。

前提条件

このチュートリアルを開始する前に、「Amazon EKS を使用するようにセットアップする」の説明に従って、AWS CLI、kubectl、および eksctl ツールをインストールして設定する必要があります。

ステップ 1: Amazon EKS クラスターとノードを作成する

> **Important**
>
> 可能な限りシンプルかつ迅速に使用を開始するため、このトピックでは、クラスターとノードをデフォルト設定で作成する手順について説明します。実稼働で使用するクラスターとノードを作成する際には、すべての設定内容に習熟した上で、ご自身の要件を満たす設定でクラスターとノードをデプロイし直すことをお勧めします。詳細については、「the section called “クラスターを作成する”」および「コンピューティングを管理する」を参照してください。一部の設定は、クラスターとノードの作成時にのみ有効にできます。

クラスターの作成には、次のいずれかのノードタイプが使用できます。各タイプの詳細については、「コンピューティングを管理する」を参照してください。クラスターをデプロイした後に、他のノードタイプを追加できます。

- Fargate – Linux – the section called “ AWS Fargate” で Linux アプリケーションを実行する場合は、このタイプのノードを選択します。Fargate は、Amazon EC2 インスタンスを管理せずに Kubernetes ポッドをデプロイできるサーバーレスコンピューティングエンジンです。
- マネージド型ノード – Linux – Amazon EC2 インスタンスで Amazon Linux アプリケーションを実行する場合は、このタイプのノードを選択します。このガイドでは説明していませんが、Windows セルフマネージド型および Bottlerocket ノードを、クラスターに追加することもできます。

次のコマンドを使用して、Amazon EKS クラスターを作成します。*my-cluster* を独自の値に置き換えることができます。この名前には英数字 (大文字と小文字が区別されます) とハイフンのみを使用できます。先頭の文字は英数字である必要があります。また、100 文字より長くすることはできません。名前は、クラスターを作成する AWS リージョンおよび AWS アカウント内で一意である必要があります。*region-code* は、Amazon EKS がサポートする任意の AWS リージョンに置き換えます。AWS リージョンの一覧についてはAWS の全般的なリファレンスガイドの「Amazon EKS エンドポイントとクォータ」を参照してください。

Example

Fargate - Linux

```
eksctl create cluster --name my-cluster --region region-code --fargate
```

Managed nodes - Linux

```
eksctl create cluster --name my-cluster --region region-code
```

クラスターの作成には数分かかります。作成中に、数行の出力が表示されます。出力の最後の行は次のサンプル行のようになります。

```
[...]
[#]  EKS cluster "my-cluster" in "region-code" region is ready
```

`eksctl` により、`~/.kube/config` で `kubectl` config ファイルが作成されるか、コンピュータ上の `~/.kube/config` にある既存の config ファイル内に新規クラスター設定が追加されました。

クラスターの作成が完了したら、AWS CloudFormation コンソールで、`eksctl-`*my-cluster*`-cluster` という名前の AWS CloudFormation スタックを表示して、作成されたすべてのリソースを確認します。

ステップ 2: Kubernetes リソースを表示する

1. クラスターノードを表示します。

   ```
   kubectl get nodes -o wide
   ```

   出力例は次のとおりです。

Example

Fargate - Linux

```
NAME                                                STATUS   ROLES    AGE
 VERSION              INTERNAL-IP   EXTERNAL-IP   OS-IMAGE         KERNEL-VERSION
                CONTAINER-RUNTIME
fargate-ip-192-0-2-0.region-code.compute.internal   Ready    <none>
 8m3s    v1.2.3-eks-1234567   192.0.2.0     <none>        Amazon Linux 2
 1.23.456-789.012.amzn2.x86_64   containerd://1.2.3
fargate-ip-192-0-2-1.region-code.compute.internal   Ready    <none>
 7m30s   v1.2.3-eks-1234567   192.0.2.1     <none>        Amazon Linux 2
 1.23.456-789.012.amzn2.x86_64   containerd://1.2.3
```

Managed nodes - Linux

```
NAME                                        STATUS   ROLES    AGE    VERSION
        INTERNAL-IP   EXTERNAL-IP   OS-IMAGE         KERNEL-VERSION
   CONTAINER-RUNTIME
ip-192-0-2-0.region-code.compute.internal   Ready    <none>   6m7s
  v1.2.3-eks-1234567   192.0.2.0     192.0.2.2     Amazon Linux 2
 1.23.456-789.012.amzn2.x86_64   containerd://1.2.3
ip-192-0-2-1.region-code.compute.internal   Ready    <none>   6m4s
  v1.2.3-eks-1234567   192.0.2.1     192.0.2.3     Amazon Linux 2
 1.23.456-789.012.amzn2.x86_64   containerd://1.2.3
```

出力に表示される内容の詳細については、「the section called “クラスターリソースにアクセスする”」を参照してください。

2. クラスターで実行されているワークロードを表示します。

```
kubectl get pods -A -o wide
```

出力例は次のとおりです。

Example

Fargate - Linux

```
NAMESPACE     NAME                          READY   STATUS    RESTARTS   AGE   IP
          NODE                                                NOMINATED NODE
 READINESS GATES
kube-system   coredns-1234567890-abcde   1/1     Running   0          18m
 192.0.2.0   fargate-ip-192-0-2-0.region-code.compute.internal   <none>
 <none>
kube-system   coredns-1234567890-12345   1/1     Running   0          18m
 192.0.2.1   fargate-ip-192-0-2-1.region-code.compute.internal   <none>
 <none>
```

Managed nodes - Linux

```
NAMESPACE     NAME                          READY   STATUS    RESTARTS   AGE     IP
          NODE                                          NOMINATED NODE   READINESS
 GATES
kube-system   aws-node-12345             1/1     Running   0          7m43s
 192.0.2.1   ip-192-0-2-1.region-code.compute.internal   <none>           <none>
kube-system   aws-node-67890             1/1     Running   0          7m46s
 192.0.2.0   ip-192-0-2-0.region-code.compute.internal   <none>           <none>
kube-system   coredns-1234567890-abcde   1/1     Running   0          14m
 192.0.2.3   ip-192-0-2-3.region-code.compute.internal   <none>           <none>
kube-system   coredns-1234567890-12345   1/1     Running   0          14m
 192.0.2.4   ip-192-0-2-4.region-code.compute.internal   <none>           <none>
kube-system   kube-proxy-12345           1/1     Running   0          7m46s
 192.0.2.0   ip-192-0-2-0.region-code.compute.internal   <none>           <none>
kube-system   kube-proxy-67890           1/1     Running   0          7m43s
 192.0.2.1   ip-192-0-2-1.region-code.compute.internal   <none>           <none>
```

出力に表示される内容の詳細については、「the section called “クラスターリソースにアクセスする”」を参照してください。

ステップ 3: クラスターとノードを削除する

このチュートリアルのために作成したクラスターとノードの使用が終了したら、クリーンアップのために、次のコマンドを使用してそれらのクラスターとノードを削除する必要があります。クリーン

アップせずに、他の目的でこのクラスターを使用する場合は、「the section called “次のステップ”」を参照してください。

```
eksctl delete cluster --name my-cluster --region region-code
```

次のステップ

以下のトピックはクラスターの機能を拡張するのに役立ちます。

- サンプルアプリケーションをクラスターにデプロイします。
- クラスターを作成した IAM プリンシパルは、kubectl または AWS マネジメントコンソール を使用して Kubernetes API サーバーを呼び出すことができる唯一のプリンシパルです。他の IAM プリンシパルがクラスターにアクセスできるようにする場合はそれらを追加する必要があります。詳細については、「the section called “Kubernetes API アクセス”」および「the section called “必要なアクセス許可”」を参照してください。
- 本番用にクラスターをデプロイする前に、クラスターとノードのすべての設定を理解しておくことをお勧めします。Amazon EC2 ノードへの SSH アクセスの有効化など一部の設定はクラスターの作成時に行う必要があります。
- クラスターのセキュリティを強化するにはサービスアカウントの IAM ロールを使用する Amazon VPC コンテナネットワークインターフェイスプラグインの設定を行ってください。

Amazon EKS の使用を開始する – AWS マネジメントコンソールと AWS CLI

> **Note**
>
> このトピックではEKS Auto Mode を使用せずに開始する方法について説明します。管理されたノードグループ を使用してノードをデプロイします。
> EKS Auto Mode はクラスターコンピューティング、ストレージ、ネットワーキングのルーチンタスクを自動化します。Amazon EKS Auto Mode の使用を開始する方法について説明します。ノードをデプロイするにはEKS Auto Mode が推奨されます。

このガイドではAWS マネジメントコンソール と AWS CLI を使用して、Amazon Elastic Kubernetes Service (Amazon EKS) を使用開始するために必要なすべてのリソースを作成する方法を解説しま

す。ここでは各リソースを手動で作成します。このチュートリアルの終わりにはアプリケーションのデプロイが可能な、実行状態の Amazon EKS クラスターが完成します。

このガイドの手順に従うと、各リソースがどのように作成され、リソース間でどのようにやり取りするかを全面的に把握できます。大半のリソースを自動的に作成させたい場合にはクラスターとノードの作成に `eksctl` CLI を使用します。詳細については、「the section called “クラスターを作成する (eksctl)”」を参照してください。

前提条件

このチュートリアルを開始する前に、Amazon EKS クラスターの作成と管理に必要な次のツールとリソースを、インストールおよび設定しておく必要があります。

- AWS CLI – Amazon EKS など AWS のサービスを操作するためのコマンドラインツールです。詳細についてはAWS コマンドラインインターフェイスユーザーガイドの「インストール」を参照してください。AWS CLI のインストール後は設定も行っておくことをお勧めします。詳細についてはAWS コマンドラインインターフェイスユーザーガイドの「aws configure を使用したクイック設定」を参照してください。なお、このページにある update-kubeconfig オプションを使用するには AWS CLI v2 が必要です。
- **`kubectl`** – Kubernetes クラスターを操作するためのコマンドラインツール。詳細については、「the section called “kubectl と eksctl のセットアップ”」を参照してください。
- 必要な IAM アクセス許可 – 使用している IAM セキュリティプリンシパルには、Amazon EKS IAM ロール、サービスにリンクされたロール、AWS CloudFormation、VPC、および関連リソースを操作するためのアクセス許可が必要です。詳細については、IAM ユーザーガイドの「アクション」および「サービスにリンクされたロールの使用」を参照してください。このガイドのすべてのステップは1 つのユーザーとして実行する必要があります。現在のユーザーを確認するには次のコマンドを実行してください:

```
aws sts get-caller-identity
```

このトピック内のステップはバッシュ シェル内で実行することが推奨されます。Bash シェルを使用していない場合、行継続文字や、変数の設定と使用に関する方法など、一部のスクリプトコマンドのためにシェルの調整が必要となります。さらに、シェルの引用規則とエスケープ規則は異なる場合があります。詳細については、AWS コマンドラインインターフェイスのユーザーガイドの「AWS CLI での文字列への引用符の使用」を参照してください。

ステップ 1: Amazon EKS クラスターを作成する

> **Important**
>
> 可能な限りシンプルかつ迅速に使用を開始するため、このトピックではクラスターをデフォルトの設定で作成するステップについて説明します。本番で使用するクラスターを作成する前に、すべての設定内容に習熟した上で、要件を満たす設定でクラスターをデプロイすることをお勧めします。詳細については、「the section called “クラスターを作成する”」を参照してください。一部の設定はクラスターの作成時にのみ有効にできます。

1. Amazon EKS の要件を満たすように、パブリックサブネットとプライベートサブネットを持つ Amazon VPC を作成します。*region-code* を、Amazon EKS がサポートする任意の AWS リージョンに置き換えます。AWS リージョンの一覧については、AWS の全般的なリファレンスガイドの「Amazon EKS エンドポイントとクォータ」を参照してください。*my-eks-vpc-stack* は任意の名前に置き換えることができます。

   ```
   aws cloudformation create-stack \
     --region region-code \
     --stack-name my-eks-vpc-stack \
     --template-url https://s3.us-west-2.amazonaws.com/amazon-eks/
   cloudformation/2020-10-29/amazon-eks-vpc-private-subnets.yaml
   ```

   > **Tip**
   >
   > 前のコマンドで作成されたすべてのリソースの一覧については、https://console.aws.amazon.com/cloudformation/ で AWS CloudFormation コンソールを開きます。*my-eks-vpc-stack* スタックを選択し、[リソース] タブを選択してください。

2. クラスターの IAM ロールを作成して、そのロールに必要な Amazon EKS IAM 管理ポリシーをアタッチします。Amazon EKS によって管理される Kubernetes クラスターは、そのサービスで使用するリソースを管理するために、ユーザーに代わって他の AWS のサービスを呼び出します。

   a. 次の内容を *eks-cluster-role-trust-policy.json* という名前のファイルにコピーします。

      ```
      {
        "Version":"2012-10-17",
        "Statement": [
      ```

```
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "eks.amazonaws.com"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
```

b. ロールを作成します。

```
aws iam create-role \
  --role-name myAmazonEKSClusterRole \
  --assume-role-policy-document file://"eks-cluster-role-trust-policy.json"
```

c. このロールに、必要な Amazon EKS 管理の IAM ポリシーをアタッチします。

```
aws iam attach-role-policy \
  --policy-arn arn:aws:iam::aws:policy/AmazonEKSClusterPolicy \
  --role-name myAmazonEKSClusterRole
```

3. Amazon EKS コンソール のhttps://console.aws.amazon.com/eks/home#/clusters 開きます:

   コンソールの右上に表示されている AWS リージョンが、クラスターを作成する AWS リージョンであることを確認します。異なる場合はAWS リージョン名の横にあるドロップダウンを展開し、使用する AWS リージョンを選択してください。

4. [クラスターを作成] を選択してください。このオプションが表示されない場合はまず左のナビゲーションペインの [クラスター] を選択してください。

5. [クラスターの設定] ページで、次の手順を実行してください:

   a. カスタム設定を選択し、Use EKS Auto Modeを無効にします。(EKS Auto Mode クラスターを使用する場合は代わりに「the section called “マネジメントコンソール”」を参照してください)。

   b. [名前] に、クラスターの名前 (*my-cluster* など) を入力します。この名前には英数字 (大文字と小文字が区別されます) とハイフンのみを使用できます。先頭の文字は英数字である必要があります。また、100 文字より長くすることはできません。名前はクラスターを作成する AWS リージョンおよび AWS アカウント内で一意である必要があります。

   c. [クラスターサービスロール] で、*[myAmazonEKSClusterRole]* を選択してください。

   d. その他の設定はデフォルト値のままにし、[次へ] をクリックします。

6. [ネットワーキングを指定] ページで、以下の作業を行います:
   a. 前のステップで [VPC] ドロップダウンリストから作成した VPC の ID を選択してください。*`* | my-eks-vpc-stack-VPC`* に類似した ID です。
   b. 前のステップで作成したサブネットを [サブネット] のドロップダウンリストの中から選択します。このサブネットは *`* | my-eks-vpc-stack-*`* のように表示されます。
   c. 前のステップで作成したセキュリティグループを [追加のセキュリティグループ] のドロップダウンリストの中から選択します。このグループは *`* | my-eks-vpc-stack-ControlPlaneSecurityGroup-*`* のように表示されます。
   d. その他の設定はデフォルト値のままにし、[次へ] をクリックします。
7. [オブザーバビリティの設定]ページで、[次へ] を選択してください。
8. [アドオンの選択] ページで、[次へ] を選択してください。

   アドオンの詳細については、「the section called “Amazon EKS アドオン”」を参照してください。
9. [選択したアドオンセッティングの設定] ページで、[次へ] を選択してください。
10. [確認して作成] ページで、[作成] をクリックします。

    クラスターのプロビジョニングプロセスが完了するまで、数分の間、クラスター名の右側でステータスが [作成中] と表示されます。ステータスが [アクティブ] になるまで、次のステップに進まないでください。

> **Note**
>
> リクエストで指定したアベイラビリティーゾーンのいずれかに、Amazon EKS クラスターの作成に十分な容量がない場合にはエラーが表示されることがあります。このエラー出力には新しいクラスターをサポートできるアベイラビリティーゾーンが表示されます。アカウント向けにサポートされているアベイラビリティーゾーンにある 2 つ以上のサブネットを使用して、クラスターを作成します。詳細については、「the section called “容量不足”」を参照してください。

ステップ 2: クラスターに接続する

このセクションでは、Kubernetes リソースを管理できるように `kubectl` をクラスターに接続します。

先に進む前に、ステップ 1 で必ずクラスターの作成を正常に完了してください。

オプション 1: AWS CLI を使用して接続する

1. クラスター用の kubeconfig ファイルを作成もしくは更新します。*region-code* を、クラスターを作成した AWS リージョンに置き換えます。*my-cluster* の部分は自分のクラスター名に置き換えます。

   ```
   aws eks update-kubeconfig --region region-code --name my-cluster
   ```

   デフォルトではconfig ファイルが ~/.kube に作成されるか、config ファイルが既に ~/.kube に存在する場合にはその中に新しいクラスター設定が追加されます。
2. 設定をテストします。

   ```
   kubectl get svc
   ```

   > **Note**
   >
   > 認可またはリソースタイプのエラーが発生した場合はトラブルシューティングトピックの「the section called “許可されていないか、アクセスが拒否されました (kubectl)”」を参照してください。

   出力例は次のとおりです。

   ```
   NAME             TYPE        CLUSTER-IP   EXTERNAL-IP   PORT(S)   AGE
   svc/kubernetes   ClusterIP   10.100.0.1   <none>        443/TCP   1m
   ```

オプション 2: EKS コンソールを使用して接続する

別の方法として、ローカルセットアップを使用せずにコンソールから直接接続することもできます。

1. ステップ 1 で作成したクラスターの詳細ページで、[接続] を選択します。
2. AWS CloudShell は、クラスター用に事前設定された kubectl で開きます。
3. 接続をテストします。

```
kubectl get svc
```

詳細については、「the section called “kubectl を使用してクラスターにアクセスする”」を参照してください。

ステップ 3: ノードを作成する

> **Important**
>
> 可能な限りシンプルかつ迅速に使用を開始するため、このトピックではノードをデフォルトの設定で作成するステップについて説明します。本番で使用するノードを作成する前に、すべての設定内容に習熟した上で、要件を満たす設定でノードをデプロイすることをお勧めします。詳細については、「コンピューティングを管理する」を参照してください。一部の設定はノードの作成時にのみ有効にできます。

このステップではマネージド型ノードグループを使用してノードを作成するようにクラスターを設定し、前のステップで作成したサブネットとノード IAM ロールを指定します。これにより、Amazon EC2 インスタンスで Amazon Linux アプリケーションを実行できます。

EKS でノードを設定するさまざまな方法については、「コンピューティングを管理する」を参照してください。クラスターをデプロイした後に、他のノードタイプを追加できます。このガイドでは説明していませんが、Windows セルフマネージド型および Bottlerocket ノードを、クラスターに追加することもできます。

EC2 Linux マネージド型ノードグループを作成するには

1. ノードの IAM ロールを作成して、必要な Amazon EKS IAM 管理ポリシーをアタッチします。Amazon EKS ノード `kubelet` デーモンが、ユーザーに代わって AWS API への呼び出しを実行してください。ノードはIAM インスタンスプロファイルおよび関連ポリシーを通じて、これらの API コールのアクセス許可を受け取ります。
   a. 次の内容を `node-role-trust-policy.json` という名前のファイルにコピーします。

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
```

```
      "Effect": "Allow",
      "Principal": {
        "Service": "ec2.amazonaws.com"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
```

b. ノードの IAM ロールを作成します。

```
aws iam create-role \
  --role-name myAmazonEKSNodeRole \
  --assume-role-policy-document file://"node-role-trust-policy.json"
```

c. ロールに、必要なマネージド IAM ポリシーをアタッチします。

```
aws iam attach-role-policy \
  --policy-arn arn:aws:iam::aws:policy/AmazonEKSWorkerNodePolicy \
  --role-name myAmazonEKSNodeRole
aws iam attach-role-policy \
  --policy-arn arn:aws:iam::aws:policy/AmazonEC2ContainerRegistryReadOnly \
  --role-name myAmazonEKSNodeRole
aws iam attach-role-policy \
  --policy-arn arn:aws:iam::aws:policy/AmazonEKS_CNI_Policy \
  --role-name myAmazonEKSNodeRole
```

d. Amazon EKS コンソール のhttps://console.aws.amazon.com/eks/home#/clusters 開きます:

e. ステップ 1: Amazon EKS クラスターを作成する で作成したクラスターの名前 (*my-cluster* など 選択してください:

f. ***my-cluster*** ページで、次の操作を行います:

g. [コンピューティング] タブを開きます。

h. [ノードグループを追加] をクリックします。

2. [ノードグループの設定] ページで以下を実行してください:

a. [名前] に、マネージド型ノードグループの一意の名前 (*#########* など 入力してください。ノードグループ名は 63 文字以下である必要があります。先頭は文字または数字でなければなりませんが、残りの文字にはハイフンおよびアンダースコアを含めることもできます。

b. [ノード IAM ロール名] で、前のステップで作成した *myAmazonEKSNodeRole* ロールを選択してください。各ノードグループには独自の一意の IAM ロールを使用することをお勧めします。

c. [次へ] を選択してください。

3. [コンピューティングとスケーリングの設定] ページではデフォルトの値を受け入れ、[次へ] をクリックします。
4. [ネットワーキングの指定] ページではデフォルトの値を受け入れ、[次へ] をクリックします。
5. [確認と作成] ページで、マネージド型ノードグループの設定を確認し、[作成] を選択してください。
6. 数分後、[ノードグループの設定] セクションにある [ステータス] の表示が、[作成中] から [アクティブ] に変わります。ステータスが [アクティブ] になるまで、次のステップに進まないでください。

ステップ 4: リソースを表示する

ノードと Kubernetes のワークロードを表示することができます。

1. 左のナビゲーションペインで [クラスター] を選択します。[クラスター] のリストで *my-cluster* など、作成したクラスターの名前を選択してください。
2. ***my-cluster*** ページで、次の項目を選択してください:
   a. [コンピューティング] タブ – クラスターにデプロイした [ノード] が一覧表示されます。ノードの名前を選択すると、そのノードに関するより詳細な情報が表示されます。
   b. [リソース] タブ – Amazon EKS クラスターにデフォルトでデプロイされたすべての Kubernetes リソースが表示されます。コンソールでリソースタイプを選択すると、その詳細を確認できます。

ステップ 5: リソースを削除する

このチュートリアルのために作成したクラスターとノードの使用が終了したら、作成したリソースを削除する必要があります。リソースを削除する前に、他の目的でこのクラスターを使用する場合はthe section called “次のステップ” を参照してください。

1. ここで作成した、いずれかのノードグループもしくは Fargate プロファイルを削除します。
   a. Amazon EKS コンソール のhttps://console.aws.amazon.com/eks/home#/clusters 開きます:
   b. 左のナビゲーションペインで [クラスター] を選択します。クラスターのリストから *my-cluster* を選択します。
   c. [コンピューティング] タブを開きます。

d. ノードグループを作成している場合はノードグループ「*#########*」を選択した上で、[削除]を クリックします。*#########* と入力し、[削除] を選択します。

e. ノード グループ プロファイルが削除されるまで続行しないでください。

2. クラスターを削除します。

a. 左のナビゲーションペインで [クラスター] を選択します。クラスターのリストから *my-cluster* を選択します。

b. [クラスターの削除] を選択します。

c. *my-cluster* と入力し、[削除] を選択します。クラスターが削除されるまで続行しないでください。

3. 作成した VPC AWS CloudFormation スタックを削除します。

a. CloudFormation コンソール (https://console.aws.amazon.com/cloudformation/) を開きます。

b. 「*my-eks-vpc-stack*」スタックを選択した上で、[削除] クリックします。

c. [***my-eks-vpc-stack*** の削除] 確認ダイアログボックスで、[スタックを削除] を選択します。

4. 作成した IAM ロールを削除します。

a. https://console.aws.amazon.com/iam/ で IAM コンソールを開きます。

b. 左のナビゲーションペインで、[ロール] を選択します。

c. リスト (***[myAmazonEKSClusterRole]*** または *[myAmazonEKSNodeRole]* から、作成した各ロールを選択します。[削除] を選択し、要求された確認テキストを入力した後で、[削除] をクリックします。

次のステップ

以下のトピックはクラスターの機能を拡張するのに役立ちます。

- クラスターを作成した IAM プリンシパルは、kubectl または AWS マネジメントコンソール を使用して Kubernetes API サーバーを呼び出すことができる唯一のプリンシパルです。他の IAM プリンシパルがクラスターにアクセスできるようにする場合はそれらを追加する必要があります。詳細については、「the section called “Kubernetes API アクセス”」および「the section called “必要なアクセス許可”」を参照してください。
- サンプルアプリケーションをクラスターにデプロイします。
- 本番用にクラスターをデプロイする前に、クラスターとノードのすべての設定を理解しておくことをお勧めします。Amazon EC2 ノードへの SSH アクセスの有効化など一部の設定はクラスターの作成時に行う必要があります。

- クラスターのセキュリティを強化するには[サービスアカウントの IAM ロールを使用する Amazon VPC コンテナネットワークインターフェイスプラグインの設定]を行ってください。

EKS Auto Mode を使用してクラスターインフラストラクチャを自動化する

> **Tip**
>
> 今後開催予定の Amazon EKS Auto Mode ワークショップに登録してください。

EKS Auto Mode は、Kubernetes クラスターの AWS 管理をクラスター自体を超えて拡張し、ワークロードのスムーズな運用を可能にするインフラストラクチャのセットアップおよび管理も AWS でできるようにします。インフラストラクチャに関する重要な意思決定を委任し、AWS の専門知識を日常業務に使用できます。AWS によって管理されるクラスターインフラストラクチャには、コンピューティングの自動スケーリング、ポッドとサービスのネットワーキング、アプリケーションのロードバランシング、クラスター DNS、ブロックストレージ、GPU サポートなど、アドオンとは対照的に、コアコンポーネントとして多くの Kubernetes 機能が備わっています。

開始するには、新しい EKS Auto Mode クラスターをデプロイするか、既存のクラスターで EKS Auto Mode を有効にすることができます。eksctl、AWS CLI、AWS マネジメントコンソール、EKS API、または任意の Infrastructure as Code ツールを使用して、EKS Auto Mode クラスターをデプロイ、アップグレード、または変更できます。

EKS Auto Mode を使用すると、任意の Kubernetes 互換ツールを引き続き使用できます。EKS Auto Mode は、Amazon EC2、Amazon EBS、ELB などの AWS サービスと統合され、ベストプラクティスに従う AWS クラウドリソースを使用します。これらのリソースは自動的にスケールされ、コストが最適化され、定期的に更新されるため、運用コストおよびオーバーヘッドを最小限に抑えることができます。

機能

EKS Auto Mode には次の高度な機能があります。

Kubernetes クラスター管理の合理化: EKS Auto Mode は、運用オーバーヘッドを最小限に抑えながら本番環境に対応したクラスターを提供することで、EKS 管理を合理化します。EKS Auto Mode を使用すると、EKS の深い専門知識を必要とせずに、要求の厳しい動的なワークロードを自信を持って実行できます。

アプリケーションの可用性: EKS Auto Mode は、Kubernetes アプリケーションの需要に基づいて EKS クラスター内のノードを動的に追加または削除します。この結果、手動キャパシティープランニングの必要性が最小限に抑えられ、アプリケーションの可用性が確保されます。EKS Auto Mode は Amazon Application Recovery Controller (ARC) ゾーンシフトとも統合され、障害のあるアベイラビリティーゾーン内のノードのプロビジョニングを自動的に停止し、ゾーンシフトイベント中の自発的な中断を一時停止します。

効率: EKS Auto Mode は、NodePool とワークロードの要件で定義された柔軟性を維持しながら、計算コストを最適化するように設計されています。また、未使用のインスタンスを終了し、ワークロードを他のノードに統合して、コスト効率を向上させます。コスト効率の高いクラスターの設定と、統合を妨げる設定の回避に関するガイダンスについては、「EKS Auto Mode でのコスト最適化」を参照してください。

セキュリティ: EKS Auto Mode は、ノードに対してイミュータブルとして扱われる AMI を使用します。これらの AMI は、ロックダウンされたソフトウェアを適用し、SELinux の必須アクセスコントロールを有効にして、読み取り専用のルートファイルシステムを提供します。さらに、EKS Auto Mode によって起動されたノードの最大存続期間は 21 日 (これは短縮可能) で、その後は自動的に新しいノードに置き換えられます。このアプローチは、多くのお客様が既に採用しているベストプラクティスに沿って、ノードを定期的にサイクルすることで、セキュリティ体制を強化します。

自動アップグレード: EKS Auto Mode は、設定されたポッド中断の予算 (PDB) や NodePool 中断の予算 (NDB) を維持しながら、Kubernetes クラスター、ノード、関連コンポーネントを最新のパッチにより最新版に保ちます。PDB やその他の設定をブロックすることで更新が妨げられる場合、最大 21 日間の最大存続期間まで介入が必要になる場合があります。バージョンロールバックでは、EKS Auto Mode はコントロールプレーンが元に戻る前にワーカーノードを自動的にロールバックし、設定された中断コントロールに従います。詳細については、「the section called “EKS Auto Mode クラスターをロールバックする”」を参照してください。

マネージドコンポーネント: EKS Auto Mode には、Kubernetes と AWS クラウドの機能がコアコンポーネントとして備わっていますが、それ以外の場合は、アドオンとして管理が必要になります。これには、ポッド IP アドレスの割り当て、ポッドネットワークポリシー、ローカル DNS サービス、GPU プラグイン、ヘルスチェッカー、EBS CSI ストレージのサポートが組み込まれています。

カスタマイズ可能な NodePools と NodeClasses: ワークロードでストレージ、コンピューティング、またはネットワーキングの設定を変更する必要がある場合は、EKS Auto Mode を使用してカスタム NodePools と NodeClasses を作成できます。ユーザーはデフォルトの NodePools と NodeClasses は編集できませんが、デフォルトの設定で新しいカスタムの NodePools または NodeClasses を追加して、特定の要件を満たすことができます。

自動コンポーネント

EKS Auto Mode は、主要なインフラストラクチャコンポーネントを自動化することで、Amazon EKS クラスターのオペレーションを合理化します。EKS Auto Mode を有効にすると、EKS クラスターを管理するタスクがさらに減ります。

以下は、自動化されているデータプレーンコンポーネントのリストです。

- コンピューティング: 多くのワークロードでは、EKS Auto Mode を使用すると、EKS クラスターのコンピューティングの多くの側面を気にする必要がなくなります。具体的には次のとおりです。
  - ノード: EKS Auto Mode ノードは、アプライアンスのように扱われるように設計されています。EKS Auto Mode は以下を実行します。
    - ワークロードを介入なしで実行するのに必要な多くのサービスを備えて設定された適切な AMI を選択します。
    - SELinux 強制モードと読み取り専用のルートファイルシステムを使用して、AMI 上のファイルへのアクセスをロックダウンします。
    - SSH または SSM アクセスを禁止することで、ノードへの直接アクセスを防止します。
    - GPU サポートがあり、NVIDIA GPU および Neuron GPU 用のカーネルドライバーとプラグインが分離されているため、高性能ワークロードが実現されます。
    - EC2 スポットインスタンスの中断通知、EC2 インスタンスのヘルスイベント/スケジュールされたメンテナンスイベント、EC2 インスタンスのステータスチェックの失敗が自動的に処理されます
  - 自動スケーリング: Karpenter の自動スケーリングに依拠して、EKS Auto Mode はスケジュールできないポッドを監視し、それらのポッドを実行するための新しいノードをデプロイできるようにします。ワークロードが終了すると、不要になったノードは EKS Auto Mode によって動的に中断および終了され、リソースの使用状況が最適化されます。
  - アップグレード: ノードを制御することで、EKS Auto Mode が必要に応じてセキュリティパッチやオペレーティングシステム、コンポーネントのアップグレードを提供する機能が効率化されます。これらのアップグレードは、ワークロードの中断を最小限に抑えるように設計されています。EKS Auto Mode は、ソフトウェアと API を最新にするために、ノードの最大存続期間を強制的に 21 日に設定します。
- ロードバランシング: EKS Auto Mode は、Amazon の Elastic Load Balancing サービスと統合することでロードバランシングを合理化し、Kubernetes サービスおよびイングレスリソースに対するロードバランサーのプロビジョニングと設定を自動化します。また、Application Load Balancer と Network Load Balancer の両方の高度な機能をサポートし、それらのライフサイクルを管理し、そ

れらをクラスターの需要に合わせてスケールします。この統合により、AWS ベストプラクティスに準拠した本番環境対応のロードバランシングソリューションが提供されるため、インフラストラクチャ管理ではなくアプリケーションに集中できます。

- ストレージ: EKS Auto Mode は、ノード終了時にボリュームタイプ、ボリュームサイズ、暗号化ポリシー、削除ポリシーを設定することで、エフェメラルストレージを設定します。
- ネットワーキング: EKS Auto Mode は、ポッドとサービスの接続に関する重要なネットワーキングタスクを自動化します。これには、IPv4/IPv6 サポートと、IP アドレススペースを拡張するためのセカンダリ CIDR ブロックの使用が含まれます。
- Identity and Access Management: EKS Auto Mode クラスターに EKS Pod Identity Agent をインストールする必要はありません。

これらのコンポーネントの詳細については、「the section called “仕組み”」を参照してください。

設定

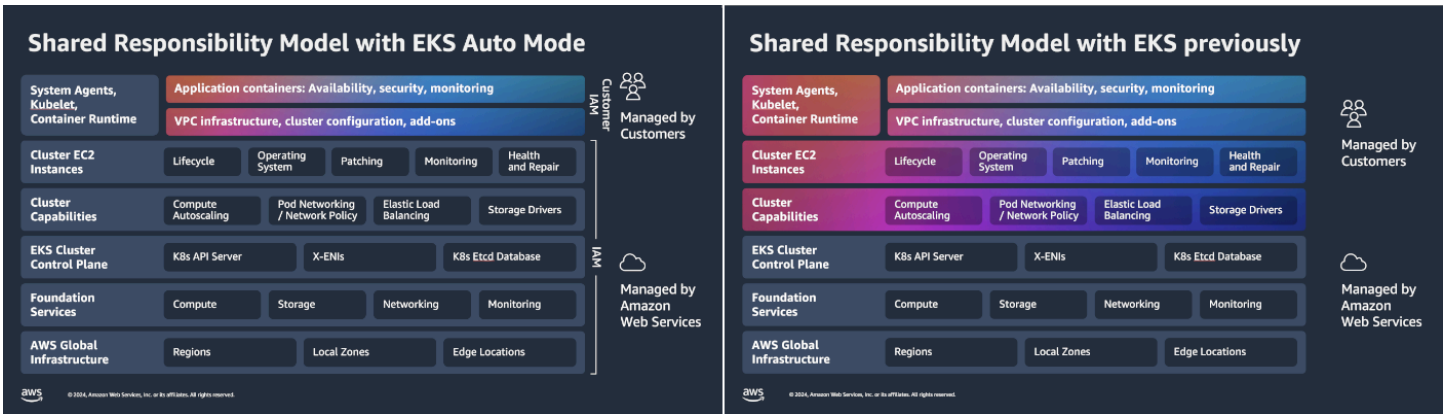
EKS Auto Mode は、ユーザーの介入なしにほとんどのデータプレーンサービスを効果的に管理しますが、これらのサービスの一部の動作を変更したい場合もあるでしょう。EKS Auto Mode クラスターの設定は、次の方法で変更できます。

- Kubernetes DaemonSets: ノードにインストールされているサービスを変更する代わりに、Kubernetes DaemonSets を使用できます。Kubernetes は、クラスター内のすべてのノードで実行される DaemonSets を管理します。このようにして、ノードをモニタリングしたりその他の方法で監視したりするための特別なサービスを追加できます。
- カスタムの NodePools および NodeClasses: デフォルトの NodePools と NodeClasses は EKS Auto Mode によって設定されており、ユーザーはこれを編集できません。ノードの動作をカスタマイズするには、次のようなユースケース用に追加の NodePools または NodeClasses を作成できます。
  - 特定のインスタンスタイプ (高速プロセッサや EC2 スポットインスタンスなど) の選択。
  - セキュリティまたはコスト追跡の目的でのワークロード分離。
  - IOPS、サイズ、スループットなどのエフェメラルストレージの設定。
- ロードバランシング: EKS Auto Mode が Kubernetes オブジェクトとして実行するロードバランシングなどの一部のサービスは、EKS Auto Mode クラスターで直接設定できます。

EKS Auto Mode を設定するためのオプションの詳細については、「the section called “設定”」を参照してください。

責任共有モデル

AWS 責任共有モデルは、AWS とお客様の間のセキュリティとコンプライアンスの責任を定義します。以下の画像とテキストは、EKS Auto Mode と EKS 標準モードの間でお客様と AWS の責任がどのように異なるかを比較および対照しています。



EKS Auto Mode は、Kubernetes インフラストラクチャの共有責任の大部分をお客様から AWS に移行します。EKS Auto Mode では、AWS がクラウドセキュリティに対してより多くの責任を負います。これは、以前はお客様の責任であり、現在は共有されています。お客様はアプリケーションに集中できるようになり、基盤となるインフラストラクチャを AWS が管理します。

顧客の責任

EKS Auto Mode では、お客様は可用性、セキュリティ、モニタリングなど、アプリケーションコンテナに対する責任を引き続き維持します。また、VPC インフラストラクチャと EKS クラスター設定の制御も維持します。このモデルにより、お客様はクラスターインフラストラクチャ管理を AWS に委任し、アプリケーション固有の懸念事項に集中できます。オプションのノードごとの機能は、AWS アドオンを介してクラスターに含めることができます。

AWS の責任

EKS Auto Mode では、AWS は、Auto Mode を使用しない EKS クラスターで既に管理されているものと比較して、いくつかの追加の重要なコンポーネントの管理を含む責任を拡張します。特に、EKS Auto Mode は、起動された EC2 インスタンスの設定、管理、セキュリティ、スケーリングに加えて、ロードバランシング、IP アドレス管理、ネットワークポリシー、ブロックストレージのクラスター機能も引き受けます。次のコンポーネントは、AWS によって EKS Auto Mode で管理されます。

- Auto Mode 起動 EC2 インスタンス: AWS は、Amazon EC2 マネージドインスタンスを使用してノードの完全なライフサイクルを処理します。EC2 マネージドインスタンスは、オペレーティン

グシステムの設定、パッチ適用、モニタリング、ヘルスメンテナンスを担当します。このモデルでは、インスタンス自体とそこで実行されているゲストオペレーティングシステムの両方の責任を AWS が担います。ノードは、コンテナを実行するように最適化された Bottlerocket AMI のバリアントを使用します。Bottlerocket AMI には、ロックダウンされたソフトウェア、イミュータブルなルートファイルシステム、安全なネットワークアクセス (SSH または SSM を介した直接通信を防ぐため) があります。2026 年 4 月 22 日以降、新しいマネージドインスタンスと関連リソースは、EC2 コンソールビューと API リストオペレーションからデフォルトで非表示になります。この動作は、マネージドリソースの可視性設定で調整できます。詳細については、「the section called “管理されたインスタンス”」を参照してください。

- クラスター機能: AWS は、コンピューティングの自動スケーリング、ネットワークポリシーの適用による Pod ネットワーキング、Elastic Load Balancing の統合、ストレージドライバーの設定を管理します。
- クラスターコントロールプレーン: AWS は、標準の EKS と同様に、Kubernetes API サーバー、クロスアカウント ENI、および etcd データベースを引き続き管理します。
- Foundation Services とグローバルインフラストラクチャ: AWS は、基盤となるコンピューティング、ストレージ、ネットワーキング、およびモニタリングサービス、ならびにリージョン、ローカルゾーン、エッジロケーションのグローバルインフラストラクチャに対する責任を維持します。

Amazon EKS 自動モードI クラスターを作成する

この章ではさまざまなツールとインターフェイスを使用して、自動モードI を有効にした Amazon EKS クラスターを作成する方法について説明します。自動モードI はクラスターのコンピューティング、ネットワーク、およびストレージインフラストラクチャを自動的に設定および管理することで、クラスターの作成を簡素化します。AWS CLI、AWS マネジメントコンソール、または eksctl コマンドラインツールを使用して、Auto Mode クラスターを作成する方法について説明します。

> **Note**
>
> EKS 自動モード にはKubernetes バージョン 1.29 以降が必要です。

ニーズに応じて任意のツールを選択する: AWS マネジメントコンソールには、EKS Auto Mode の機能について学習し、個々のクラスターを作成するのに最適なビジュアルインターフェイスが用意されています。AWS CLI は特にクラスター作成を既存のワークフローまたは CI/CD パイプラインに統合する場合、スクリプト作成および自動化タスクに最適です。eksctl CLI は Kubernetes ネイティブの

エクスペリエンスを提供し、実用的なデフォルトにより簡素化されたコマンドラインオペレーションを求める、Kubernetes ツールに精通しているユーザーにお勧めです。

開始する前に、EKS クラスターを作成するための適切な IAM アクセス許可など、必要な前提条件がインストールおよび設定されている状態にします。`kubectl`、`aws`、`eksctl` などの CLI ツールをインストールする方法については「セットアップ」を参照してください。

AWS CLI、AWS マネジメントコンソール、または eksctl CLI を使用して、Amazon EKS Auto Mode でクラスターを作成できます。

トピック

- eksctl CLI を使用して EKS Auto Mode クラスターを作成する
- AWS CLI を使用して EKS Auto Mode クラスターを作成する
- AWS マネジメントコンソール を使用して EKS Auto Mode クラスターを作成する

eksctl CLI を使用して EKS Auto Mode クラスターを作成する

このトピックでは、eksctl コマンドラインインターフェイス (CLI) を使用して Amazon EKS Auto Mode クラスターを作成する方法について説明します。Auto Mode クラスターは、単一の CLI コマンドを実行するか、YAML 設定ファイルを適用することで作成できます。どちらの方法でも同じ機能が得られ、YAML アプローチでクラスター設定をより細かく制御できます。

eksctl CLI は、基盤となる AWS リソースの作成と設定を処理することで、EKS Auto Mode クラスターの作成と管理のプロセスを簡素化します。先に進む前に、必要な AWS 認証情報とアクセス許可をローカルマシンに設定してください。このガイドでは、Amazon EKS の基本的な概念に精通しており、必要な CLI ツールが既にインストールされていることを前提としています。

> **Note**
>
> eksctl のバージョン `0.195.0` 以降をインストールする必要があります。詳細については、GitHub の「eksctl releases」を参照してください。

CLI コマンドを使用して EKS Auto Mode クラスターを作成する

`aws` および `eksctl` ツールがインストールされている必要があります。EC2 インスタンス、EC2 ネットワーキング、EKS クラスター、IAM ロールなどの AWS リソースを管理するには、十分なア

クセス許可が付与された状態で AWS CLI にログインする必要があります。詳細については、「セットアップ」を参照してください。

次のコマンドを実行して、eksctl を使用して新しい EKS オートモードクラスターを作成します。

```
eksctl create cluster --name=<cluster-name> --enable-auto-mode
```

YAML ファイルを使用して EKS Auto Mode クラスターを作成する

aws および eksctl ツールがインストールされている必要があります。EC2 インスタンス、EC2 ネットワーキング、EKS クラスター、IAM ロールなどの AWS リソースを管理するには、十分なアクセス許可が付与された状態で AWS CLI にログインする必要があります。詳細については、セットアップ を参照してください。

以下のサンプル ClusterConfig リソースの EKS Auto Mode 設定オプションを確認してください。ClusterConfig の完全な仕様については、「eksctl ドキュメント」を参照してください。

AWS では、EKS Auto Mode を有効にすることをお勧めします。EKS Auto Mode クラスターを初めて作成する場合は、nodeRoleARN を指定せずに EKS Auto Mode のノード IAM ロールを作成します。AWS アカウントにノード IAM ロールが既にある場合は、AWS ではそのロールの再利用をお勧めします。

AWS では、nodePools には値を指定しないことをお勧めします。EKS Auto Mode では、デフォルトのノードプールが作成されます。Kubernetes API を使用して、追加のノードプールを作成できます。

```
# cluster.yaml
apiVersion: eksctl.io/v1alpha5
kind: ClusterConfig

metadata:
  name: <cluster-name>
  region: <aws-region>

iam:
  # ARN of the Cluster IAM Role
  # optional, eksctl creates a new role if not supplied
  # suggested to use one Cluster IAM Role per account
  serviceRoleARN: <arn-cluster-iam-role>

autoModeConfig:
```

```
  # defaults to false
  enabled: boolean
  # optional, defaults to [general-purpose, system].
  # suggested to leave unspecified
  # To disable creation of nodePools, set it to the empty array ([]).
  nodePools: []string
  # optional, eksctl creates a new role if this is not supplied
  # and nodePools are present.
  nodeRoleARN: string
```

`ClusterConfig` ファイルを `cluster.yaml` として保存し、次のコマンドを使用してクラスターを作成します。

```
eksctl create cluster -f cluster.yaml
```

AWS CLI を使用して EKS Auto Mode クラスターを作成する

EKS Auto Mode クラスターは、コンピューティング、ストレージ、ネットワーキングのルーティン化されたクラスター管理タスクを自動化します。例えば、EKS Auto Mode クラスターは、追加のノードが必要なタイミングを自動的に検出し、ワークロードの需要に合わせて新しい EC2 インスタンスをプロビジョニングします。

このトピックでは、AWS CLI を使用して新しい EKS Auto Mode クラスターを作成し、必要に応じてサンプルワークロードをデプロイする方法について説明します。

前提条件

作業を開始する前に、次の項目があることを確認します。

- AWS コマンドラインインターフェイス (AWS CLI) の最新バージョンがデバイスにインストールおよび設定されている。現在のバージョンを確認するには、`aws --version` を実行してください。AWS CLI をインストールまたは更新するには、「AWS コマンドラインインターフェイスユーザーガイド」の「インストール」および「クイック設定」を参照してください。
- IAM ポリシー、IAM ロール、EKS クラスターなどの AWS リソースを作成するための十分な IAM アクセス許可がある。
- kubectl コマンドラインツールがデバイスにインストールされている。AWS では、EKS クラスターの Kubernetes バージョンと同じ kubectl バージョンを使用することをお勧めします。kubectl をインストールまたはアップグレードするには、the section called “kubectl と eksctl のセットアップ” を参照してください。

VPC サブネットを指定する

Amazon EKS Auto Mode は、VPC サブネットにノードをデプロイします。EKS クラスターの作成時に、ノードをデプロイする VPC サブネットを指定する必要があります。AWS アカウントでデフォルトの VPC サブネットを使用するか、重要なワークロード専用の VPC を作成することができます。

- AWS では、クラスター専用の VPC を作成することをお勧めします。the section called “「VPC を作成する」” では、その方法を説明しています。
- EKS コンソールは、新しい VPC の作成に役立ちます。the section called “マネジメントコンソール” では、その方法を説明しています。
- また、AWS アカウントのデフォルト VPC を使用することができます。サブネット ID を検出するには、以下の手順を使用します。

デフォルト VPC のサブネット ID を検出するには

AWS CLI の使用:

1. 次のコマンドを実行して、デフォルト VPC とそのサブネットを一覧表示します。

```
aws ec2 describe-subnets --filters "Name=vpc-id,Values=$(aws ec2 describe-vpcs
 --query 'Vpcs[?IsDefault==`true`].VpcId' --output text)" --query 'Subnets[*].
{ID:SubnetId,AZ:AvailabilityZone}' --output table
```

2. 出力を保存し、サブネット ID をメモします。

   サンプル出力:

```
----------------------------------------
|             DescribeSubnets          |
----------------------------------------
|   SubnetId        |   AvailabilityZone  |
|-------------------|---------------------|
|   subnet-012345678 |   us-west-2a        |
|   subnet-234567890 |   us-west-2b        |
|   subnet-345678901 |   us-west-2c        |
----------------------------------------
```

EKS Auto Mode クラスターの IAM ロール

クラスター IAM ロール

EKS Auto Mode では、新しい EC2 インスタンスのプロビジョニングなど、AWS アカウントでアクションを実行するのにクラスター IAM ロールが必要です。必要なアクセス許可を EKS に付与するには、このロールを作成する必要があります。AWS では、クラスター IAM ロールに次の AWS マネージドポリシーをアタッチすることをお勧めします。

- AmazonEKSComputePolicy
- AmazonEKSBlockStoragePolicyV2
- Amazon EKSロードバランシングポリシー
- Amazon EKSネットワーキングポリシー
- AmazonEKSクラスターポリシー

ノード IAM ロール

EKS Auto Mode クラスターを作成する際に、ノード IAM ロールを指定します。EKS Auto Mode が保留中のワークロードを処理するノードを作成すると、新しい EC2 インスタンスノードごとにノード IAM ロールが割り当てられます。このロールにより、ノードは EKS と通信できますが、通常、ノードで実行されているワークロードからはアクセスされません。

ノードで実行されているワークロードにアクセス許可を付与する場合は、EKS Pod Identity を使用します。詳細については、the section called “Pod Identity” を参照してください。

このロールを作成し、次の AWS マネージドポリシーをアタッチする必要があります。

- AmazonEKSWorkerNodeMinimalPolicy
- AmazonEC2ContainerRegistryPullOnly

サービスにリンクされたロール

EKS Auto Mode には、AWS によって自動的に作成および設定されるサービスリンクロールも必要です。詳細については、「AWSServiceRoleForAmazonEKS」 に関するページを参照してください。

EKS Auto Mode クラスター IAM ロールを作成する

ステップ 1: 信頼ポリシーを作成する

Amazon EKS サービスがロールを引き受けることを許可する信頼ポリシーを作成します。ポリシーを `trust-policy.json` として保存します。

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "eks.amazonaws.com"
      },
      "Action": [
        "sts:AssumeRole",
        "sts:TagSession"
      ]
    }
  ]
}
```

ステップ 2: IAM ロールを作成する

信頼ポリシーを使用して、クラスター IAM ロールを作成します。

```
aws iam create-role \
    --role-name AmazonEKSAutoClusterRole \
    --assume-role-policy-document file://trust-policy.json
```

ステップ 3: ロール ARN をメモする

以降のステップで使用する新しいロールの ARN を取得して保存します。

```
aws iam get-role --role-name AmazonEKSAutoClusterRole --query "Role.Arn" --output text
```

ステップ 4: 必要なポリシーをアタッチする

次の AWS マネージドポリシーをクラスター IAM ロールにアタッチして、必要なアクセス許可を付与します。

AmazonEKSClusterPolicy:

```
aws iam attach-role-policy \
    --role-name AmazonEKSAutoClusterRole \
    --policy-arn arn:aws:iam::aws:policy/AmazonEKSClusterPolicy
```

AmazonEKSコンピュートポリシー:

```
aws iam attach-role-policy \
    --role-name AmazonEKSAutoClusterRole \
    --policy-arn arn:aws:iam::aws:policy/AmazonEKSComputePolicy
```

AmazonEKSBlockStoragePolicyV2:

```
aws iam attach-role-policy \
    --role-name AmazonEKSAutoClusterRole \
    --policy-arn arn:aws:iam::aws:policy/AmazonEKSBlockStoragePolicyV2
```

AmazonEKSロードバランシングポリシー:

```
aws iam attach-role-policy \
    --role-name AmazonEKSAutoClusterRole \
    --policy-arn arn:aws:iam::aws:policy/AmazonEKSLoadBalancingPolicy
```

AmazonEKSNetworkingPolicy:

```
aws iam attach-role-policy \
    --role-name AmazonEKSAutoClusterRole \
    --policy-arn arn:aws:iam::aws:policy/AmazonEKSNetworkingPolicy
```

EKS Auto Mode ノード IAM ロールを作成する

ステップ 1: 信頼ポリシーを作成する

Amazon EKS サービスがロールを引き受けることを許可する信頼ポリシーを作成します。ポリシーを `node-trust-policy.json` として保存します。

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
```

```
      "Effect": "Allow",
      "Principal": {
        "Service": "ec2.amazonaws.com"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
```

ステップ 2: ノード IAM ロールを作成する

前のステップの node-trust-policy.json ファイルを使用して、ロールを引き受けることができるエンティティを定義します。次のコマンドを実行してノード IAM ロールを作成します。

```
aws iam create-role \
    --role-name AmazonEKSAutoNodeRole \
    --assume-role-policy-document file://node-trust-policy.json
```

ステップ 3: ロール ARN をメモする

ロールを作成したら、ノード IAM ロールの ARN を取得して保存します。以降のステップでこの ARN が必要になります。次のコマンドを使用して ARN を取得します。

```
aws iam get-role --role-name AmazonEKSAutoNodeRole --query "Role.Arn" --output text
```

ステップ 4: 必要なポリシーをアタッチする

次の AWS マネージドポリシーをノード IAM ロールにアタッチして、必要なアクセス許可を付与します。

AmazonEKSWorkerNodeMinimalPolicy:

```
aws iam attach-role-policy \
    --role-name AmazonEKSAutoNodeRole \
    --policy-arn arn:aws:iam::aws:policy/AmazonEKSWorkerNodeMinimalPolicy
```

AmazonEC2ContainerRegistryPullOnly:

```
aws iam attach-role-policy \
    --role-name AmazonEKSAutoNodeRole \
    --policy-arn arn:aws:iam::aws:policy/AmazonEC2ContainerRegistryPullOnly
```

EKS Auto Mode クラスターを作成する

概要

AWS CLI を使用して EKS Auto Mode クラスターを作成するには、次のパラメータが必要になります。

- `cluster-name`: クラスターの名前。
- `k8s-version`: Kubernetes バージョン (1.31 など)。
- `subnet-ids`: 前のステップで特定したサブネット ID。
- `cluster-role-arn`: クラスター IAM ロールの ARN。
- `node-role-arn`: ノード IAM ロールの ARN。

デフォルトのクラスター設定

クラスターを作成する前に、以下のデフォルト値および機能を確認してください。

- `nodePools`: EKS Auto Mode には、汎用ノードプールとシステムデフォルトノードプールが含まれています。「ノードプール」の詳細を確認してください。

注: EKS Auto Mode のノードプールは Amazon EKS マネージドノードグループとは異なりますが、同じクラスター内で共存できます。

- `computeConfig.enabled`: EC2 インスタンスの作成や削除などのルーティン化されたコンピューティングタスクを自動化します。
- `kubernetesNetworkConfig.elasticLoadBalancing.enabled`: Elastic Load Balancer の作成や削除などのロードバランシングタスクを自動化します。
- `storageConfig.blockStorage.enabled`: Amazon EBS ボリュームの作成や削除などのストレージタスクを自動化します。
- `accessConfig.authenticationMode`: EKS アクセスエントリを要求します。EKS 認証モードの詳細を確認してください。

環境変数の設定

`create-cluster` コマンドを実行する前に、以下の環境変数を設定します。プレースホルダーの値を、独自の値に置き換えます:

```
export AWS_REGION=us-east-1
export CLUSTER_NAME=my-auto-cluster
export K8S_VERSION=1.35
export CLUSTER_ROLE_ARN=arn:aws:iam::111122223333:role/AmazonEKSAutoClusterRole
export NODE_ROLE_ARN=arn:aws:iam::111122223333:role/AmazonEKSAutoNodeRole
export SUBNETS_JSON='["subnet-ExampleID1","subnet-ExampleID2","subnet-ExampleID3"]'
```

> **Note**
>
> 前のステップのサブネット ID を使用して SUBNETS_JSON 変数を設定します。シェルが角括弧と引用符を特殊文字として解釈しないように、値を一重引用符で囲む必要があります。

変数の検証 (オプション)

変数が正しく設定されていることを確認します:

```
echo "Region: ${AWS_REGION}"
echo "Cluster Name: ${CLUSTER_NAME}"
echo "K8s Version: ${K8S_VERSION}"
echo "Cluster Role ARN: ${CLUSTER_ROLE_ARN}"
echo "Node Role ARN: ${NODE_ROLE_ARN}"
echo "Subnets JSON: ${SUBNETS_JSON}"
```

コマンドを実行する

次のコマンドを使用してクラスターを作成します。

```
aws eks create-cluster \
  --region ${AWS_REGION} \
  --cli-input-json \
  "{
      \"name\": \"${CLUSTER_NAME}\",
      \"version\": \"${K8S_VERSION}\",
      \"roleArn\": \"${CLUSTER_ROLE_ARN}\",
      \"resourcesVpcConfig\": {
        \"subnetIds\": ${SUBNETS_JSON},
        \"endpointPublicAccess\": true,
        \"endpointPrivateAccess\": true
      },
      \"computeConfig\": {
```

```
        \"enabled\": true,
        \"nodeRoleArn\":\"${NODE_ROLE_ARN}\",
        \"nodePools\": [\"general-purpose\", \"system\"]
      },
      \"kubernetesNetworkConfig\": {
        \"elasticLoadBalancing\": {
          \"enabled\": true
        }
      },
      \"storageConfig\": {
        \"blockStorage\": {
          \"enabled\": true
        }
      },
      \"accessConfig\": {
        \"authenticationMode\": \"API\"
      }
    }"
```

クラスターのステータスを確認する

ステップ 1: クラスターの作成を確認する

次のコマンドを実行して、クラスターのステータスを確認します。通常、クラスターの作成には約 15 分かかります。

```
aws eks describe-cluster --name "${CLUSTER_NAME}" --output json
```

ステップ 2: kubeconfig を更新する

クラスターの準備ができたら、ローカルの kubeconfig ファイルを更新して、kubectl がクラスターと通信できるようにします。この設定では、認証に AWS CLI を使用します。

```
aws eks update-kubeconfig --name "${CLUSTER_NAME}"
```

ステップ 3: ノードプールを検証する

次のコマンドを使用して、クラスター内のノードプールを一覧表示します。

```
kubectl get nodepools
```

次のステップ

- 新しい EKS Auto Mode クラスターに サンプルワークロードをデプロイ する方法について説明します。

AWS マネジメントコンソール を使用して EKS Auto Mode クラスターを作成する

AWS マネジメントコンソールで EKS Auto Mode クラスターを作成すると、他のオプションよりも必要な設定が少なくなります。EKS は AWS IAM および VPC ネットワーキングと統合されて、EKS クラスターに関連付けられたリソースの作成に役立ちます。

コンソールでクラスターを作成するオプションは 2 つあります。

- クイック設定 (EKS Auto Mode を使用)
- カスタム設定

このトピックでは、クイック設定オプションを使用して EKS Auto Mode クラスターを作成する方法について説明します。

クイック設定オプションを使用して EKS Auto Mode クラスターを作成する

EC2 インスタンス、EC2 ネットワーキング、EKS クラスター、IAM ロールなどの AWS リソースを管理するには、十分なアクセス許可が付与された状態で AWS マネジメントコンソール にログインする必要があります。

1. EKS コンソールに移動します
2. [クラスターを作成] を選択します
3. [クイック設定] オプションが選択されていることを確認します
4. 次の値を決定するか、テストクラスター用のデフォルトを使用します。
   - クラスター名
   - Kubernetes バージョン
5. クラスター IAM ロールを選択します。EKS Auto Mode クラスターを初めて作成する場合は、[推奨ロールを作成] オプションを使用します。
   - 必要に応じて、すべての EKS Auto Mode クラスターに対して、AWS アカウントで 1 つのクラスター IAM ロールを再利用できます。

- クラスター IAM ロールには、EC2 インスタンス、EBS ボリューム、EC2 ロードバランサーなどのリソースを管理するのに必要な EKS Auto Mode のアクセス許可が含まれています。
- [推奨ロールを作成] オプションでは、すべてのフィールドに推奨値が事前入力されています。[次へ]、[作成] の順に選択します。ロールは推奨される `AmazonEKSAutoClusterRole` という名前を使用します。
- 最近新しいロールを作成した場合は、[更新] アイコンを使用してロール選択ドロップダウンを再ロードします。

6. ノード IAM ロールを選択します。EKS Auto Mode クラスターを初めて作成する場合は、[推奨ロールを作成] オプションを使用します。
   - 必要に応じて、すべての EKS Auto Mode クラスターに対して、AWS アカウントで 1 つのノード IAM ロールを再利用できます。
   - ノード IAM ロールには、Auto Mode ノードがクラスターに接続するのに必要なアクセス許可が含まれています。ノード IAM ロールには、コンテナの ECR イメージを取得するためのアクセス許可が付与されている必要があります。
   - [推奨ロールを作成] オプションでは、すべてのフィールドに推奨値が事前入力されています。[次へ]、[作成] の順に選択します。ロールは推奨される `AmazonEKSAutoNodeRole` という名前を使用します。
   - 最近新しいロールを作成した場合は、[更新] アイコンを使用してロール選択ドロップダウンを再ロードします。
7. EKS Auto Mode クラスターの VPC を選択します。[VPC を作成] を選択して EKS 用の新しい VPC を作成するか、以前に EKS 用に作成した VPC を選択します。
   - VPC コンソールを使用して新しい VPC を作成する場合は、AWS ではアベイラビリティーゾーンごとに少なくとも 1 つの NAT ゲートウェイを作成することをお勧めします。それ以外の場合は、他のすべてのデフォルトを使用できます。
   - さらなる情報と IPv6 クラスターの要件の詳細については、「the section called “「VPC を作成する」”」を参照してください。
8. (任意) EKS Auto Mode は、選択した VPC のプライベートサブネットを自動的に入力します。不要なサブネットは削除できます。
   - EKS では、ベストプラクティスに従って VPC からプライベートサブネットが自動的に選択されます。必要に応じて、パブリックサブネットなど、VPC から追加のサブネットを選択できます。
9. (任意) [クイック設定のデフォルトを表示] を選択して、新しいクラスターのすべての設定値を確認します。この表は、クラスターの作成後には一部の値を編集できないことを示しています。

10[Create cluster (クラスターの作成)] を選択します。クラスターの作成が完了するまでに 15 分かかる場合があります。

次のステップ

- サンプルワークロードを EKS Auto Mode クラスターにデプロイする 方法について説明します。

既存の EKS クラスターで EKS 自動モードを有効にする

既存の EKS クラスターで EKS 自動モード を有効にできます。

AWS では次の移行がサポートされています。

- Karpenter から EKS 自動モードノードへの移行。詳細については、the section called “Karpenter からの移行” を参照してください。
- EKS マネージドノードグループから EKS 自動モードノードへの移行。詳細については、the section called “MNG から移行する” を参照してください。
- EKS Fargate から EKS 自動モードへの移行。詳細については、the section called “Fargate からの移行” を参照してください。

AWS では以下の移行はサポートされていません。

- EBS CSI コントローラー (Amazon EKS アドオンを使用) から EKS 自動モード EBS CSI コントローラー (EKS 自動モードで管理) へのボリュームの移行。これらは 2 つの異なる Kubernetes ボリュームプロビジョナーを使用するため、一方で作成された PVC をもう一方でマウントすることはできません。
  - `eks-auto-mode-ebs-migration-tool` (AWS Labs プロジェクト) により、標準の EBS CSI StorageClass (`ebs.csi.aws.com`) と EKS Auto EBS CSI StorageClass (`ebs.csi.eks.amazonaws.com`) 間の移行が可能になります。移行では、既存の PersistentVolumeClaim/PersistentVolume リソースを削除して再作成する必要があるため、実装前に非本番環境での検証が不可欠です。
- AWS ロードバランサーコントローラー から EKS 自動モードへのロードバランサーの移行

  Amazon EKS 自動モードクラスターに AWS ロードバランサーコントローラーをインストールできます。`IngressClass` または `loadBalancerClass` のオプションを使用して、サービスリソースとイングレスリソースをロードバランサーコントローラー、または EKS 自動モー

ドのいずれかに関連付けます。規範的なガイダンスについては、Migrate NGINX Ingress Controllers when enabling Amazon EKS Auto Mode を参照してください。

- 代替 CNI またはその他のサポートされていないネットワーク設定を使用した EKS クラスターの移行

移行のリファレンス

次の移行のリファレンスを使用して、Kubernetes リソースをセルフマネージドコントローラーまたは EKS 自動モードが所有するように設定します。

機能	[リソース]	フィールド	セルフマネージド型	EKS 自動モード
ブロックストレージ	StorageClass	provisioner	ebs.csi.aws.com	ebs.csi.eks.amazonaws.com
負荷分散	Service	loadBalancerClass	service.k8s.aws/nlb	eks.amazonaws.com/nlb
負荷分散	IngressClass	controller	ingress.k8s.aws/alb	eks.amazonaws.com/alb
負荷分散	IngressClassParams	apiversion	elbv2.k8s.aws/v1beta1	eks.amazonaws.com/v1
負荷分散	TargetGroupBinding	apiversion	elbv2.k8s.aws/v1beta1	eks.amazonaws.com/v1
コンピューティング	NodeClass	apiVersion	karpenter.sh/v1	eks.amazonaws.com/v1

EBS ボリュームの移行

ワークロードを EKS 自動モードに移行する場合、CSI ドライバープロビジョナーが異なるため、EBS ボリュームの移行を処理する必要があります。

- EKS 自動モードのプロビジョナー: `ebs.csi.eks.amazonaws.com`
- オープンソース EBS CSI のプロビジョナー: `ebs.csi.aws.com`

永続ボリュームを移行するには、次の手順を実行します。

1. ボリューム保持ポリシーの変更: 既存の PersistentVolume (PV) の `persistentVolumeReclaimPolicy` を `Retain` に変更して、基盤となる EBS ボリュームが削除されないようにします。
2. Kubernetes からの PV の削除: 実際の EBS ボリュームはそのまま保持しながら、古い PV リソースを削除します。
3. 静的プロビジョニングを使用した新規 PV の作成: 同じ EBS ボリュームを参照するが、ターゲット CSI ドライバーで動作する新しい PV を作成します。
4. 新規 PVC へのバインド: `volumeName` フィールドを使用して、特定の PV を明示的に参照する新しい PVC を作成します。

考慮事項

- この移行を開始する前に、アプリケーションが停止していることを確認します。
- 移行プロセスを開始する前に、データをバックアップします。
- このプロセスは、永続ボリュームごとに実行する必要があります。
- 新しい PVC を使用するようにワークロードを更新する必要があります。

ロードバランサーの移行

AWS セルフマネージドロードバランサーコントローラーから EKS 自動モード に既存のロードバランサーを直接転送することはできません。代わりに、ブルー/グリーンデプロイ戦略を実装する必要があります。これはマネージドコントローラーで新しいロードバランサーを作成しながら、既存のロードバランサー設定を維持することになります。

サービスの中断を最小限に抑えるにはDNS ベースのトラフィックシフトアプローチをお勧めします。まず、既存の設定を稼働させたまま、EKS 自動モードを使用して新しいロードバランサーを作

成します。次に、DNS ルーティング (Route 53 など) を使用して、古いロードバランサーから新しいロードバランサーにトラフィックを徐々に移行します。トラフィックが正常に移行され、新しい設定を確認したら、古いロードバランサーとセルフマネージドコントローラーを廃止できます。

既存のクラスターで EKS Auto Mode を有効にする

このトピックでは、既存の Amazon EKS クラスターで Amazon EKS Auto Mode を有効にする方法について説明します。既存のクラスターで Auto Mode を有効にするには、IAM アクセス許可を更新し、コア EKS Auto Mode 設定を構成する必要があります。有効にすると、既存のコンピューティングワークロードの移行を開始して、Auto Mode のシンプルなオペレーションと自動化されたインフラストラクチャ管理を活用できます。

> **Important**
>
> EKS Auto Mode を有効にする前に、特定の Amazon EKS アドオンの求められるバージョン以降がインストールされていることを確認します。詳細については、the section called “必要なアドオンバージョン” を参照してください。

開始する前に、Amazon EKS クラスターへの管理者アクセスと、IAM ロールを変更するアクセス許可があることを確認してください。このトピックのステップでは、AWS マネジメントコンソールまたは AWS CLI を使用して Auto Mode を有効にする方法について説明します。

AWS マネジメントコンソール

IAM、EKS、および EC2 リソースを管理するアクセス許可が付与された状態で AWS コンソールにログインする必要があります。

> **Note**
>
> EKS クラスターのクラスター IAM ロールは、クラスターの作成後に変更することはできません。EKS Auto Mode には、このロールに対する追加のアクセス許可が必要です。追加のポリシーを現在のロールにアタッチする必要があります。

クラスター IAM ロールを更新する

1. AWS マネジメントコンソールでクラスターの概要ページを開きます。

2. [クラスター IAM ロール ARN] で、[IAM で表示] を選択します。
3. [アクセス許可を追加] ドロップダウンから [ポリシーをアタッチ] を選択します。
4. [検索] ボックスより、次のポリシーを検索して選択します。
   - AmazonEKSComputePolicy
   - AmazonEKSBlockStoragePolicyV2
   - AmazonEKSLoadBalancingPolicy
   - AmazonEKSNetworkingPolicy
   - AmazonEKSClusterPolicy
5. [アクセス許可を追加] を選択します。
6. [信頼関係] タブで、[信頼ポリシーを編集] を選択します。
7. 次のクラスター IAM ロール信頼ポリシーを挿入し、[ポリシーを更新] を選択します。

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "eks.amazonaws.com"
      },
      "Action": [
        "sts:AssumeRole",
        "sts:TagSession"
      ]
    }
  ]
}
```

EKS Auto Mode を有効にする

1. AWS マネジメントコンソールでクラスターの概要ページを開きます。
2. [EKS Auto Mode] で [管理] を選択します
3. EKS Auto Mode をオンに切り替えます。
4. [EKS ノードプール] ドロップダウンから、作成するデフォルトのノードプールを選択します。
   - EKS Auto Mode のノードプールの詳細を確認してください。詳細については、「the section called “ノードプールの作成”」を参照してください。

5. この AWS アカウントで EKS オートモードノード IAM ロールを以前に作成している場合は、[ノード IAM ロール] ドロップダウンでそのロールを選択します。以前にこのロールを作成したことがない場合は、[推奨ロールを作成] を選択し、手順に従います。

AWS CLI

前提条件

- 既存の EKS クラスターのクラスター IAM ロールには、以下のポリシーなど、EKS Auto Mode に対する十分なアクセス許可が付与されている必要があります。
  - `AmazonEKSComputePolicy`
  - `AmazonEKSBlockStoragePolicyV2`
  - `AmazonEKSLoadBalancingPolicy`
  - `AmazonEKSNetworkingPolicy`
  - `AmazonEKSClusterPolicy`
- クラスター IAM ロールには、`sts:TagSession` アクションを含む更新された信頼ポリシーが必要です。クラスター IAM ロール作成の詳細については、the section called “ AWS CLI” を参照してください。
- `aws` CLI がインストール済み、ログイン済み、および十分なバージョンであること。IAM、EKS、および EC2 リソースを管理するためのアクセス許可が必要です。詳細については、セットアップ を参照してください。

手順

次のコマンドを使用して、既存のクラスターで EKS Auto Mode を有効にします。

> **Note**
>
> コンピューティング、ブロックストレージ、およびロードバランシング機能はすべて、同じリクエストで有効または無効にする必要があります。

```
aws eks update-cluster-config \
 --name $CLUSTER_NAME \
 --compute-config enabled=true \
 --kubernetes-network-config '{"elasticLoadBalancing":{"enabled": true}}' \
```

```
  --storage-config '{"blockStorage":{"enabled": true}}'
```

必要なアドオンバージョン

既存のクラスターで EKS Auto Mode を有効にする場合は、特定のアドオンを更新することが必要になる場合があります。次のことにご注意ください。

- これは、既存のクラスターを EKS Auto Mode に移行する場合にのみ該当します。
- EKS Auto Mode を有効にして作成された新しいクラスターでは、この更新は必要ありません。

次のいずれかのアドオンがインストールされている場合は、指定された最小バージョン以上であることを確認してください。

アドオン名	求められる最小バージョン
Amazon VPC CNI Kubernetes 用プラグイン	v1.19.0-eksbuild.1
Kube-proxy	• v1.26.15-eksbuild.19 • v1.27.16-eksbuild.14 • v1.28.15-eksbuild.4 • v1.29.10-eksbuild.3 • v1.30.6-eksbuild.3 • v1.31.2-eksbuild.3
Amazon EBS CSI ドライバー	v1.37.0-eksbuild.1
CSI スナップショットコントローラー	v8.1.0-eksbuild.2
EKS Pod Identity Agent	v1.3.4-eksbuild.1

詳細については、the section called “アドオンを更新する” を参照してください。

次のステップ

- マネージドノードグループワークロードを移行するには、「the section called “MNG から移行する”」を参照してください。

- セルフマネージド Karpenter から移行するには、the section called “Karpenter からの移行” を参照してください。

kubectl を使用して Karpenter から EKS Auto Mode に移行する

このトピックではkubectl を使用してワークロードを Karpenter から Amazon EKS Auto Mode に移行するプロセスについて説明します。移行は段階的に実行できるため、移行を通じてクラスターの安定性とアプリケーションの可用性を維持しながら、ワークロードを自分のペースで移動できます。

以下に説明するステップバイステップのアプローチにより、移行期間中に Karpenter と EKS Auto Mode をサイドバイサイドで実行できます。このデュアルオペレーション戦略により、Karpenter を完全に廃止する前に EKS Auto Mode でワークロードの動作を検証できるため、スムーズな移行が可能になります。アプリケーションは個別またはグループで移行できるため、特定の運用要件とリスク許容度に柔軟に対応できます。

前提条件

移行を開始する前に、以下のことを確認してください:

- Karpenter v1.1 以降がクラスターにインストールされている。詳細についてはKarpenter ドキュメントの「Upgrading to 1.1.0+」を参照してください。
- `kubectl` がインストールされ、クラスターに接続されている。詳細については、「セットアップ」を参照してください。

このトピックでは Karpenter と NodePools に精通していることを前提としています。詳細についてはKarpenter ドキュメントを参照してください。

> **Note**
>
> 一部の CRD (例: `nodepools.karpenter.sh`、`nodeclaims.karpenter.sh`) は、EKS Auto Mode と Karpenter の間で共有されています。移行中にこれらの CRD を更新または変更しないでください。

ステップ 1: クラスターで EKS Auto Mode を有効にする

AWS CLI またはマネジメントコンソールを使用して、既存のクラスターで EKS Auto Mode を有効にします。詳細については、「the section called “クラスターで有効にする”」を参照してください。

> **Note**
>
> EKS Auto Mode を有効にする際には移行中のこの段階で `general purpose` ノードプールを有効にしないでください。このノードプールは選択的ではありません。
> 詳細については、「the section called “組み込みノードプールを確認する”」を参照してください。

ステップ 2: テイントが適用された EKS Auto Mode NodePool を作成する

テイントを使用して、EKS Auto Mode 用の新しい NodePool を作成します。これを行うことで、既存のポッドが新しい EKS Auto Mode ノードで自動的にスケジュールされることはありません。このノードプールはEKS Auto Mode に組み込まれた `default` `NodeClass` を使用します。詳細については、「the section called “ノードクラスの作成”」を参照してください。

テイントが適用されたノードプールの例:

```
apiVersion: karpenter.sh/v1
kind: NodePool
metadata:
  name: eks-auto-mode
spec:
  template:
    spec:
      requirements:
        - key: "eks.amazonaws.com/instance-category"
          operator: In
          values: ["c", "m", "r"]
      nodeClassRef:
        group: eks.amazonaws.com
        kind: NodeClass
        name: default
      taints:
        - key: "eks-auto-mode"
          effect: "NoSchedule"
```

移行元の Karpenter 設定と一致するように、ノード プールの要件を更新します。少なくとも 1 つの要件が必要です。

ステップ 3: 移行するワークロードを更新する

EKS Auto Mode に移行するワークロードを特定して更新します。これらのワークロードに許容範囲とノードセレクターの両方を追加します:

```
apiVersion: apps/v1
kind: Deployment
spec:
  template:
    spec:
      tolerations:
      - key: "eks-auto-mode"
        effect: "NoSchedule"
      nodeSelector:
        eks.amazonaws.com/compute-type: auto
```

この変更により、新しい EKS Auto Mode ノードでワークロードをスケジュールできます。

EKS Auto Mode は Karpenter とは異なるラベルを使用します。EC2 マネージドインスタンスに関連するラベルは、`eks.amazonaws.com` で始まります。詳細については、「the section called “ノードプールの作成”」を参照してください。

ステップ 4: ワークロードを段階的に移行する

移行するワークロードごとにステップ 3 を繰り返します。この結果、要件とリスク許容度に基づいて、ワークロードを個別またはグループで移動できます。

ステップ 5: 元の Karpenter NodePool を削除する

すべてのワークロードを移行したら、元の Karpenter NodePool を削除できます:

```
kubectl delete nodepool <original-nodepool-name>
```

ステップ 6: EKS Auto Mode NodePool からテイントを削除する (任意)

EKS Auto Mode を新しいワークロードのデフォルトにする場合は EKS Auto Mode NodePool からテイントを削除できます:

```
apiVersion: karpenter.sh/v1
kind: NodePool
metadata:
  name: eks-auto-mode
```

```
spec:
  template:
    spec:
      nodeClassRef:
        group: eks.amazonaws.com
        kind: NodeClass
        name: default
      # Remove the taints section
```

ステップ 7: ワークロードからノードセレクターを削除する (任意)

EKS Auto Mode NodePool からテイントを削除した場合はEKS Auto Mode がデフォルトになったため、必要に応じてワークロードからノードセレクターを削除できます:

```
apiVersion: apps/v1
kind: Deployment
spec:
  template:
    spec:
      # Remove the nodeSelector section
      tolerations:
      - key: "eks-auto-mode"
        effect: "NoSchedule"
```

ステップ 8: クラスターから Karpenter をアンインストールする

Karpenter を削除するステップはインストール方法によって異なります。詳細については、「Karpenter インストールの手順」を参照してください。

EKS マネージドノードグループから EKS Auto Mode に移行する

EKS Auto Mode を使用するように Amazon EKS クラスターを移行する場合、eksctl CLI ツールを使用してマネージドノードグループ (MNG) から既存のワークロードをスムーズに移行できます。このプロセスにより、EKS Auto Mode がコンピューティングリソースを最適化しながら、アプリケーションの継続的な可用性が確保されます。移行は、実行中のアプリケーションの中断を最小限に抑えて実行できます。

このトピックでは、既存のマネージドノードグループからポッドを安全にドレインし、EKS Auto Mode が新しくプロビジョニングされたインスタンスでポッドを再スケジュールできるようにする手順について説明します。この手順に従うことで、移行を通じてアプリケーションの可用性を維持しながら、EKS Auto Mode のインテリジェントなワークロード統合を活用できます。

前提条件

- EKS Auto Mode が有効になっているクラスター
- `eksctl` CLI がインストールされ、クラスターに接続されている。詳細については、「セットアップ」を参照してください。
- Karpenter がクラスターにインストールされていない。

手順

次の `eksctl` CLI コマンドを使用して、既存のマネージドノードグループインスタンスからポッドのドレインを開始します。EKS Auto Mode は、置き換えられたポッドをバックアップする新しいノードを作成します。

```
eksctl delete nodegroup --cluster=<clusterName> --name=<nodegroupName>
```

クラスター内のマネージドノードグループごとに、このコマンドを実行する必要があります。

このコマンドの詳細については、eksctl のドキュメントの「Deleting and draining nodegroups」を参照してください。

EKS Fargate から EKS Auto Mode に移行する

このトピックでは、`kubectl` を使用してワークロードを EKS Fargate から Amazon EKS Auto Mode に移行するプロセスについて説明します。移行は段階的に実行できるため、移行を通じてクラスターの安定性とアプリケーションの可用性を維持しながら、ワークロードを自分のペースで移動できます。

以下に説明するステップバイステップのアプローチを取ると、移行期間中に EKS Fargate と EKS Auto Mode を並行して実行できます。このデュアルオペレーション戦略では、EKS Fargate を完全に廃止する前に EKS Auto Mode でワークロードの動作を検証できるため、スムーズな移行が可能です。アプリケーションは個別またはグループで移行できるため、特定の運用要件とリスク許容度に柔軟に対応できます。

Amazon EKS Auto Mode と、AWS Fargate を使用した EKS の比較

AWS Fargate を使用する Amazon EKS は、EKS を実行するお客様に引き続きオプションとしてご利用いただけますが、今後のアプローチとしては Amazon EKS Auto Mode がお勧めです。EKS Auto Mode は、Kubernetes に完全に準拠しており、Fargate では対応していない Istio などのアップス

トリーム Kubernetes プリミティブとプラットフォームツールをすべてサポートしています。EKS Auto Mode は、GPU インスタンスやスポットインスタンスを含め、すべての EC2 ランタイム購入オプションにも完全に対応しているため、EC2 の割引の交渉といった仕組みを活用して節約を行えます。Fargate で EKS を使用する場合は、そのような機能が用意されていません。

さらに、EKS Auto Mode なら、Fargate と同じ分離モデルを実現できます。これを行うには、標準の Kubernetes スケジューリング機能を使用して、各 EC2 インスタンスで 1 つのアプリケーションコンテナを実行します。Amazon EKS Auto Mode を導入すると、Kubernetes を AWS で実行するメリットを最大限に活用できます。これはKubernetes に完全準拠したプラットフォームであり、EC2 と購入の広範なオプションをすべて利用できます。また、Fargate が実現するインフラストラクチャ管理の利便性や抽象化も維持されています。

EKS Auto Mode で Fargate のような分離を実現する

各ポッドがそれぞれの専用インスタンスで実行される Fargate のポッド分離モデルをレプリケートするには、Kubernetes トポロジの分散制約を使用できます。これは、ノード間でポッド分散を制御するための推奨アプローチです。

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: isolated-app
spec:
  replicas: 3
  selector:
    matchLabels:
      app: isolated-app
  template:
    metadata:
      labels:
        app: isolated-app
      annotations:
        eks.amazonaws.com/compute-type: ec2
    spec:
      topologySpreadConstraints:
      - maxSkew: 1
        topologyKey: kubernetes.io/hostname
        whenUnsatisfiable: DoNotSchedule
        labelSelector:
          matchLabels:
            app: isolated-app
        minDomains: 1
```

```
      containers:
      - name: app
        image: nginx
        ports:
        - containerPort: 80
```

この設定では、次のようになります。

- maxSkew: 1 は、任意の 2 つのノード間のポッド数の差が最大 1 であることを保証し、ノードごとに 1 つのポッドを効果的に分散する
- topologyKey: kubernetes.io/hostname はノードをトポロジドメインとして定義する
- whenUnsatisfiable: DoNotSchedule は、制約が満たされない場合、スケジューリングを禁止する
- minDomains: 1 は、スケジュールする前に、少なくとも 1 つのドメイン (ノード) が存在することを確認する

EKS Auto Mode は、この制約を満たすために必要に応じて新しい EC2 インスタンスを自動的にプロビジョニングし、Fargate と同じ分離モデルを提供しながら、すべての EC2 インスタンスタイプと購入オプションへのアクセス権を付与します。

または、ポッドアンチアフィニティルールを使用して、より厳密な分離を行うこともできます。

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: isolated-app
spec:
  replicas: 3
  selector:
    matchLabels:
      app: isolated-app
  template:
    metadata:
      labels:
        app: isolated-app
      annotations:
        eks.amazonaws.com/compute-type: ec2
    spec:
      affinity:
        podAntiAffinity:
```

```
          requiredDuringSchedulingIgnoredDuringExecution:
          - labelSelector:
              matchExpressions:
              - key: app
                operator: In
                values:
                - isolated-app
            topologyKey: kubernetes.io/hostname
      containers:
      - name: app
        image: nginx
        ports:
        - containerPort: 80
```

requiredDuringSchedulingIgnoredDuringExecution を含む podAntiAffinity ルールにより、ラベル app: isolated-app を持つ 2 つのポッドを同じノードでスケジュールすることを禁止します。このアプローチにより、Fargate と同様の強固な分離を保証します。

前提条件

移行を開始する前に、以下のことを確認してください:

- Fargate を使用してクラスターをセットアップします。詳細については、「the section called “はじめに”」を参照してください。
- kubectl がインストールされ、クラスターに接続されている。詳細については、「セットアップ」を参照してください。

ステップ 1: Fargate クラスターを確認する

1. Fargate を使用する EKS クラスターが実行中かどうかを確認します。

```
kubectl get node
```

```
NAME STATUS ROLES AGE VERSION
fargate-ip-192-168-92-52.ec2.internal Ready <none> 25m v1.30.8-eks-2d5f260
fargate-ip-192-168-98-196.ec2.internal Ready <none> 24m v1.30.8-eks-2d5f260
```

2. 実行中のポッドを確認します。

```
kubectl get pod -A
```

```
NAMESPACE NAME READY STATUS RESTARTS AGE
kube-system coredns-6659cb98f6-gxpjz 1/1 Running 0 26m
kube-system coredns-6659cb98f6-gzzsx 1/1 Running 0 26m
```

3. deployment_fargate.yaml という名前のファイルにデプロイを記述します。

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
  labels:
    app: nginx
spec:
  replicas: 3
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
      annotations:
        eks.amazonaws.com/compute-type: fargate
    spec:
      containers:
      - name: nginx
        image: nginx
        ports:
        - containerPort: 80
```

4. デプロイを適用します:

```
kubectl apply -f deployment_fargate.yaml
```

```
deployment.apps/nginx-deployment created
```

5. ポッドとデプロイを確認します。

```
kubectl get pod,deploy
```

```
NAME                                    READY   STATUS    RESTARTS   AGE
pod/nginx-deployment-5c7479459b-6trtm   1/1     Running   0          61s
pod/nginx-deployment-5c7479459b-g8ssb   1/1     Running   0          61s
pod/nginx-deployment-5c7479459b-mq4mf   1/1     Running   0          61s

NAME                                READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/nginx-deployment    3/3     3            3           61s
```

6. ノードを確認します。

```
kubectl get node -owide
```

```
NAME                                      STATUS  ROLES  AGE VERSION
   INTERNAL-IP     EXTERNAL-IP OS-IMAGE       KERNEL-VERSION
 CONTAINER-RUNTIME
fargate-ip-192-168-111-43.ec2.internal  Ready   <none> 31s v1.30.8-eks-2d5f260
 192.168.111.43  <none>      Amazon Linux 2 5.10.234-225.910.amzn2.x86_64
 containerd://1.7.25
fargate-ip-192-168-117-130.ec2.internal Ready   <none> 36s v1.30.8-eks-2d5f260
 192.168.117.130 <none>      Amazon Linux 2 5.10.234-225.910.amzn2.x86_64
 containerd://1.7.25
fargate-ip-192-168-74-140.ec2.internal  Ready   <none> 36s v1.30.8-eks-2d5f260
 192.168.74.140  <none>      Amazon Linux 2 5.10.234-225.910.amzn2.x86_64
 containerd://1.7.25
```

ステップ 2: クラスターで EKS Auto Mode を有効にする

1. AWS CLI またはマネジメントコンソールを使用して、既存のクラスターで EKS Auto Mode を有効にします。詳細については、「the section called “クラスターで有効にする”」を参照してください。
2. ノードプールを確認します。

```
kubectl get nodepool
```

```
NAME              NODECLASS   NODES   READY   AGE
general-purpose   default     1       True    6m58s
system            default     0       True    3d14h
```

ステップ 3: 移行するワークロードを更新する

EKS Auto Mode に移行するワークロードを特定して更新します。

Fargate から EKS Auto Mode にワークロードを移行するには、注釈 `eks.amazonaws.com/compute-type: ec2` を適用します。こうすることで、Fargate プロファイルに関係なく、Fargate によるワークロードスケジュールは行われず、そのロールは EKS Auto Mode NodePool に引き継がれます。詳細については、「the section called “ノードプールの作成”」を参照してください。

1. デプロイ (deployment_fargate.yaml ファイルなど) を変更して、コンピューティングタイプを ec2 に変更します。

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
  labels:
    app: nginx
spec:
  replicas: 3
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
      annotations:
        eks.amazonaws.com/compute-type: ec2
    spec:
      containers:
      - name: nginx
        image: nginx
        ports:
        - containerPort: 80
```

2. デプロイを適用します。この変更により、新しい EKS Auto Mode ノードでワークロードをスケジュールできます。

```
kubectl apply -f deployment_fargate.yaml
```

3. デプロイが EKS Auto Mode クラスターで実行されていることを確認します。

```
kubectl get pod -o wide
```

```
NAME                                READY   STATUS    RESTARTS   AGE     IP
    NODE                    NOMINATED NODE   READINESS GATES
nginx-deployment-97967b68d-ffxxh   1/1     Running   0          3m31s
 192.168.43.240   i-0845aafcb51630ffb   <none>           <none>
nginx-deployment-97967b68d-mbcgj   1/1     Running   0          2m37s
 192.168.43.241   i-0845aafcb51630ffb   <none>           <none>
nginx-deployment-97967b68d-qpd8x   1/1     Running   0          2m35s
 192.168.43.242   i-0845aafcb51630ffb   <none>           <none>
```

4. Fargate ノードが実行されておらず、デプロイが EKS Auto Mode のマネージドノードで実行中であることを確認します。

```
kubectl get node -owide
```

```
NAME                  STATUS ROLES  AGE   VERSION              INTERNAL-IP     EXTERNAL-
IP OS-IMAGE                                         KERNEL-VERSION CONTAINER-RUNTIME
i-0845aafcb51630ffb Ready  <none> 3m30s v1.30.8-eks-3c20087 192.168.41.125
 3.81.118.95 Bottlerocket (EKS Auto) 2025.3.14 (aws-k8s-1.30) 6.1.129
 containerd://1.7.25+bottlerocket
```

ステップ 4: ワークロードを段階的に移行する

移行するワークロードごとにステップ 3 を繰り返します。この結果、要件とリスク許容度に基づいて、ワークロードを個別またはグループで移動できます。

ステップ 5: 元の fargate プロファイルを削除する

すべてのワークロードを移行したら、元の `fargate` プロファイルを削除できます。*<fargate profile name>* を Fargate プロファイルの名前に置き換えます。

```
aws eks delete-fargate-profile --cluster-name eks-fargate-demo-cluster --fargate-
profile-name <fargate profile name>
```

ステップ 6: CoreDNS をスケールダウンする

EKS Auto Mode で CoreDNS が処理されるため、`coredns` デプロイを 0 にスケールダウンします。

```
kubectl scale deployment coredns -n kube-system --replicas=0
```

EKS 自動モードI クラスターでワークロードを実行する

この章では自動モードI で実行されている アマゾン EKS クラスターにさまざまなタイプのワークロードをデプロイする方法の例を示します。この例ではサンプルアプリケーション、ロードバランシングされたウェブアプリケーション、永続的ストレージを使用するステートフルワークロード、特定のノード配置要件があるワークロードなどの主要なワークロードパターンを示しています。各例には独自のアプリケーションのテンプレートとして使用できる、完全なマニフェストとステップバイステップのデプロイステップが含まれています。

例に進む前に、EKS クラスターが 自動モードI で実行され、AWS CLI と kubectl がインストールされた状態にしてください。詳細については「セットアップ」を参照してください。この例では Kubernetes の概念と kubectl コマンドに関する基本的な知識があることを前提としています。

これらのユースケースベースのサンプルを使用して、EKS 自動モードI クラスターでワークロードを実行できます。

the section called “拡張ワークロードのデプロイ”

kubectl コマンドを使用してサンプルワークロードを EKS 自動モードI クラスターにデプロイする方法を示しています。

the section called “ロードバランサーをデプロイする”

アマゾン EKS で 2048 ゲームのコンテナ化されたバージョンをデプロイする方法を示しています。

the section called “ステートフルワークロードをデプロイする”

サンプルステートフルアプリケーションを EKS 自動モードI クラスターにデプロイする方法を示しています。

the section called “加速ワークロードのデプロイ”

EKS 自動モードで管理されるノードにハードウェアアクセラレーションワークロードをデプロイする方法を示します。

the section called “デプロイを制御する”

注釈を使用して、EKS 自動モードI によって管理されるノードにワークロードがデプロイされるかどうかを制御する方法を示しています。

Amazon EKS Auto Mode クラスターにサンプルの拡張ワークロードをデプロイする

このチュートリアルでは、サンプルワークロードを EKS Auto Mode クラスターにデプロイする方法と、必要なコンピューティングリソースがどのように自動的にプロビジョニングされるか説明します。kubectl コマンドを使用してクラスターの動作を監視し、Auto Mode により AWS での Kubernetes オペレーションがどのように簡素化されるかを直接確認します。このチュートリアルを終了すると、基盤となるコンピューティングリソースを自動的に管理することで、ノードグループを手動で設定する必要なく、EKS Auto Mode がワークロードのデプロイにどのように対応するかがわかります。

前提条件

- Amazon EKS Auto Mode クラスター。クラスターの名前と AWS リージョンをメモします。
- ネットワーキング、コンピューティング、および EKS リソースを管理するのに十分なアクセス許可を持つ、ユーザーやロールなどの IAM プリンシパル。
  - 詳細については、「IAM ユーザーガイド」の「Creating roles and attaching policies in the IAM User Guide」を参照してください。
- IAM ID で aws CLI がインストールおよび設定されている。
- kubectl CLI がインストールされ、クラスターに接続されている。
  - 詳細については、セットアップ を参照してください。

ステップ 1: 既存のコンピューティングリソースを確認する (任意)

まず、kubectl を使用してクラスターのノードプールを一覧表示します。

```
kubectl get nodepools
```

サンプル出力:

```
general-purpose
```

このチュートリアルでは、general-purpose ノードプールを使用するように設定されたワークロードをデプロイします。このノードプールは EKS Auto Mode に組み込まれ、マイクロサービスやウェブアプリケーションなどの一般的なワークロードに適したデフォルトが含まれます。独自のノー

ドプールを作成できます。詳細については、the section called “ノードプールの作成” を参照してください。

次に、kubectl を使用してクラスターに接続されているノードを一覧表示します。

```
kubectl get nodes
```

EKS Auto Mode クラスターを作成したばかりの場合は、ノードはありません。

このチュートリアルでは、サンプルワークロードをデプロイします。ノードがない場合、またはワークロードが既存のノードに収まらない場合、EKS Auto Mode は新しいノードをプロビジョニングします。

ステップ 2: サンプルアプリケーションをクラスターにデプロイする

次の Kubernetes デプロイを確認し、inflate.yaml として保存します

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: inflate
spec:
  replicas: 1
  selector:
    matchLabels:
      app: inflate
  template:
    metadata:
      labels:
        app: inflate
    spec:
      terminationGracePeriodSeconds: 0
      nodeSelector:
        eks.amazonaws.com/compute-type: auto
      securityContext:
        runAsUser: 1000
        runAsGroup: 3000
        fsGroup: 2000
      containers:
        - name: inflate
          image: public.ecr.aws/eks-distro/kubernetes/pause:3.7
          resources:
            requests:
```

```
            cpu: 1
        securityContext:
          allowPrivilegeEscalation: false
```

`eks.amazonaws.com/compute-type: auto` セレクターでは、ワークロードを Amazon EKS Auto Mode ノードにデプロイする必要があることに注意してください。

デプロイをクラスターに適用します。

```
kubectl apply -f inflate.yaml
```

ステップ 3: Kubernetes イベントを監視する

次のコマンドを使用して、新しいノードの作成などの Kubernetes イベントを監視します。`ctrl+c` を使用してイベントの監視を停止します。

```
kubectl get events -w --sort-by '.lastTimestamp'
```

`kubectl` を使用して、クラスターに接続されているノードを再度一覧表示します。新しく作成されたノードをメモします。

```
kubectl get nodes
```

ステップ 4: AWS コンソールでノードとインスタンスを表示する

EKS Auto Mode ノードは EKS コンソールで表示でき、関連付けられた EC2 インスタンスは EC2 コンソールで表示できます。

EKS Auto Mode によってデプロイされた EC2 インスタンスは制限されています。EKS Auto Mode ノードで任意のコマンドを実行することはできません。

ステップ 5: デプロイを削除する

`kubectl` を使用してサンプルデプロイを削除します

```
kubectl delete -f inflate.yaml
```

クラスターに他のワークロードがデプロイされていない場合、EKS Auto Mode によって作成されたノードは空になります。

デフォルト設定では、EKS オートモードは 30 秒間空のノードを検出し、そのノードを終了します。

kubectl または EC2 コンソールを使用して、関連付けられたインスタンスが削除されたことを確認します。

サンプルロードバランサーワークロードを EKS Auto Mode にデプロイする

このガイドではロードバランシングとインターネットアクセシビリティを備えて、コンテナ化されたバージョンの 2048 ゲームを Amazon EKS にデプロイする方法について説明します。

前提条件

- EKS Auto Mode クラスター
- クラスターとやり取りするように設定された kubectl
- ALB リソースを作成するための適切な IAM アクセス許可

ステップ 1: 名前空間を作成する

まず、2048 ゲームアプリケーション専用の名前空間を作成します。

01-namespace.yaml という名前のファイルを作成します:

```
apiVersion: v1
kind: Namespace
metadata:
  name: game-2048
```

この名前空間設定を適用します:

```
kubectl apply -f 01-namespace.yaml
```

ステップ 2: アプリケーションをデプロイする

アプリケーションは 2048 ゲームコンテナの複数のレプリカを実行してください。

02-deployment.yaml という名前のファイルを作成します:

```
apiVersion: apps/v1
kind: Deployment
metadata:
  namespace: game-2048
  name: deployment-2048
```

```
spec:
  selector:
    matchLabels:
      app.kubernetes.io/name: app-2048
  replicas: 5
  template:
    metadata:
      labels:
        app.kubernetes.io/name: app-2048
    spec:
      containers:
        - image: public.ecr.aws/l6m2t8p7/docker-2048:latest
          imagePullPolicy: Always
          name: app-2048
          ports:
            - containerPort: 80
          resources:
            requests:
              cpu: "0.5"
```

> **Note**
>
> イメージ public.ecr.aws/l6m2t8p7/docker-2048:latest のロード中にエラーが表示された場合は、ノード IAM ロールに ECR からイメージをプルするための十分なアクセス許可があることを確認します。詳細については、「the section called “ノード の IAM ロール”」を参照してください。また、この例の docker-2048 イメージは x86_64 イメージであり、他のアーキテクチャでは実行できません。

主要コンポーネント:

- アプリケーションのレプリカ 5 つをデプロイ
- パブリック ECR イメージを使用
- ポッドあたり 0.5 CPU コアをリクエスト
- HTTP トラフィック用にポート 80 を公開

デプロイを適用します:

```
kubectl apply -f 02-deployment.yaml
```

ステップ 3: サービスを作成する

サービスによりデプロイがクラスターネットワークに公開されます。

`03-service.yaml` という名前のファイルを作成します:

```
apiVersion: v1
kind: Service
metadata:
  namespace: game-2048
  name: service-2048
spec:
  ports:
    - port: 80
      targetPort: 80
      protocol: TCP
  selector:
    app.kubernetes.io/name: app-2048
```

主要コンポーネント:

- ClusterIP サービスを作成
- ポート 80 をコンテナのポート 80 にマップ
- ラベルセレクターを使用してポッドを検索

サービスを適用します:

```
kubectl apply -f 03-service.yaml
```

ステップ 4: ロードバランシングを設定する

アプリケーションがインターネットに公開されるようにイングレスを設定します。

まず `IngressClass` を作成します。`04-ingressclass.yaml` という名前のファイルを作成します:

```
apiVersion: networking.k8s.io/v1
kind: IngressClass
metadata:
  labels:
    app.kubernetes.io/name: LoadBalancerController
```

```
  name: alb
spec:
  controller: eks.amazonaws.com/alb
```

> **Note**
>
> EKS Auto Mode では、パブリックサブネットとプライベートサブネットを識別するためにサブネットタグが必要です。
> eksctl を使用してクラスターを作成した場合は、これらのタグは既にあります。
> the section called “サブネットにタグを付ける” では、その方法を説明しています。

次に、イングレスリソースを作成します。05-ingress.yaml という名前のファイルを作成します:

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  namespace: game-2048
  name: ingress-2048
  annotations:
    alb.ingress.kubernetes.io/scheme: internet-facing
    alb.ingress.kubernetes.io/target-type: ip
spec:
  ingressClassName: alb
  rules:
    - http:
        paths:
          - path: /
            pathType: Prefix
            backend:
              service:
                name: service-2048
                port:
                  number: 80
```

主要コンポーネント:

- インターネット向け ALB を作成
- 直接ポッドルーティングに IP ターゲットタイプを使用
- すべてのトラフィックをルーティングまたはゲームサービスにルーティング

イングレス設定を適用します:

```
kubectl apply -f 04-ingressclass.yaml
kubectl apply -f 05-ingress.yaml
```

ステップ 5: デプロイを検証する

1. すべてのポッドが実行されていることを確認します:

   ```
   kubectl get pods -n game-2048
   ```

2. サービスが作成されていることを確認します:

   ```
   kubectl get svc -n game-2048
   ```

3. ALB エンドポイントを取得します:

   ```
   kubectl get ingress -n game-2048
   ```

イングレス出力の ADDRESS フィールドに ALB エンドポイントが表示されます。ALB がすべてのターゲットをプロビジョニングして登録するまで 2~3 分待ちます。

ステップ 6: ゲームにアクセスする

ウェブブラウザを開き、前のステップの ALB エンドポイント URL にアクセスします。2048 ゲームインターフェイスが表示されるはずです。

ステップ 7: クリーンアップ

このチュートリアルで作成したリソースをすべて削除するには:

```
kubectl delete namespace game-2048
```

これにより、デプロイ、サービス、イングレスリソースなど、名前空間内のすべてのリソースが削除されます。

シーンの背後で起こっていること

1. デプロイでは2048 ゲームを実行するポッドが 5 つ作成されます
2. このサービスではこれらのポッドに安定したネットワークアクセスを提供します

3. EKS Auto Mode:
   - AWS で Application Load Balancer を作成します
   - ポッドのターゲットグループを設定します
   - サービスにトラフィックを誘導するルーティングルールを設定します

トラブルシューティング

ゲームがロードされない場合:

- すべてのポッドが実行されていることを確認: `kubectl get pods -n game-2048`
- イングレスステータスの確認: `kubectl describe ingress -n game-2048`
- ALB ヘルスチェックを確認する: AWS コンソールでターゲットグループの状態を確認します

ステートフルワークロードのサンプルを EKS Auto Mode にデプロイする

このチュートリアルでは、EKS Auto Mode クラスターにステートフルアプリケーションのサンプルをデプロイする方法について説明します。このアプリケーションは永続ボリュームにタイムスタンプを書き込み、EBS Auto Mode の自動 EBS ボリュームプロビジョニングと永続化機能を示します。

前提条件

- EKS Auto Mode クラスター
- 適切なアクセス許可で設定された AWS CLI
- インストールおよび設定済みの `kubectl`
  - 詳細については、「セットアップ」を参照してください。

ステップ 1: 環境を設定する

1. 環境変数を設定する:

```
export CLUSTER_NAME=my-auto-cluster
export AWS_REGION="us-west-2"
```

2. kubeconfig を更新します:

```
aws eks update-kubeconfig --name "${CLUSTER_NAME}"
```

ステップ 2: ストレージクラスを作成する

StorageClassは、EBS ボリュームを EKS Auto Mode でプロビジョニングする方法を定義します。

EKS Auto Mode では、お客様用の StorageClass は作成されません。EKS Auto Mode のストレージ機能を使用するには、ebs.csi.eks.amazonaws.com を参照する StorageClass を作成する必要があります。

1. storage-class.yaml という名前のファイルを作成します :

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: auto-ebs-sc
  annotations:
    storageclass.kubernetes.io/is-default-class: "true"
provisioner: ebs.csi.eks.amazonaws.com
volumeBindingMode: WaitForFirstConsumer
parameters:
  type: gp3
  encrypted: "true"
```

2. StorageClass を適用します。

```
kubectl apply -f storage-class.yaml
```

主要コンポーネント:

- provisioner: ebs.csi.eks.amazonaws.com - EKS Auto Mode を使用する
- volumeBindingMode: WaitForFirstConsumer - ポッドが必要とするまでボリュームの作成を遅らせる
- type: gp3 - EBS ボリュームタイプを指定する
- encrypted: "true" - EBS はデフォルトの aws/ebs キーを使用して、このクラスで作成されたボリュームを暗号化します。これはオプションですが推奨されます。
- storageclass.kubernetes.io/is-default-class: "true" - 永続ボリュームのクレームで別のボリュームクラスを指定しない限り、Kubernetes はデフォルトでこのストレージクラスを使用します。別のストレージコントローラーから移行する場合は、この値を設定するときに注意してください 。(オプション)

ステップ3: 永続的なボリューム要求を作成する

PVC はStorageClassからのストレージをリクエストします。

1. pvc.yaml という名前のファイルを作成します :

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: auto-ebs-claim
spec:
  accessModes:
    - ReadWriteOnce
  storageClassName: auto-ebs-sc
  resources:
    requests:
      storage: 8Gi
```

2. SCP を適用します:

```
kubectl apply -f pvc.yaml
```

主要コンポーネント:

- accessModes: ReadWriteOnce - ボリュームは一度に 1 つのノードからマウントすることができます
- storage: 8Gi - 8 GiB ボリュームをリクエストします
- storageClassName: auto-ebs-sc - 作成したStorageClassを参照します

ステップ 4: アプリケーションのデプロイ

Deployment は、永続的ボリュームにタイムスタンプを書き込むコンテナを実行します。

1. deployment.yaml という名前のファイルを作成します :

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: inflate-stateful
spec:
```

```
  replicas: 1
  selector:
    matchLabels:
      app: inflate-stateful
  template:
    metadata:
      labels:
        app: inflate-stateful
    spec:
      terminationGracePeriodSeconds: 0
      nodeSelector:
        eks.amazonaws.com/compute-type: auto
      containers:
        - name: bash
          image: public.ecr.aws/docker/library/bash:4.4
          command: ["/usr/local/bin/bash"]
          args: ["-c", "while true; do echo $(date -u) >> /data/out.txt; sleep 60;
done"]
          resources:
            requests:
              cpu: "1"
          volumeMounts:
            - name: persistent-storage
              mountPath: /data
      volumes:
        - name: persistent-storage
          persistentVolumeClaim:
            claimName: auto-ebs-claim
```

2. デプロイを適用します。

```
kubectl apply -f deployment.yaml
```

主要コンポーネント:

- ファイルにタイムスタンプを書き込むシンプルな bash コンテナ
- PVC を /data にマウントします
- 1 CPU コアをリクエストします
- EKS マネージドノードのノードセレクタを使用します

ステップ 5: セットアップを検証する

1. ポッドが稼働していることを確認します:

```
kubectl get pods -l app=inflate-stateful
```

2. PVC がバインドされていることを確認します:

```
kubectl get pvc auto-ebs-claim
```

3. EBS ボリュームを確認します:

```
# Get the PV name
PV_NAME=$(kubectl get pvc auto-ebs-claim -o jsonpath='{.spec.volumeName}')
# Describe the EBS volume
aws ec2 describe-volumes \
  --filters Name=tag:CSIVolumeName,Values=${PV_NAME}
```

4. データが書き込まれていることを確認します:

```
kubectl exec "$(kubectl get pods -l app=inflate-stateful \
  -o=jsonpath='{.items[0].metadata.name}')" -- \
  cat /data/out.txt
```

ステップ 6: クリーンアップ

次のコマンドを実行して、このチュートリアルで作成したすべてのリソースを削除します。

```
# Delete all resources in one command
kubectl delete deployment/inflate-stateful pvc/auto-ebs-claim storageclass/auto-ebs-sc
```

シーンの背後で起こっていること

1. PVC はStorageClassからのストレージをリクエストします
2. ポッドがスケジュールされている場合:
   a. EKS Auto Mode が EBS ボリュームをプロビジョニングします
   b. PersistentVolume を作成します
   c. ボリュームをノードにアタッチします。

3. ポッドはボリュームをマウントし、タイムスタンプの書き込みを開始します

Snapshot Controller

EKS Auto Mode は、スナップショットコントローラーとも呼ばれる Kubernetes CSI Snapshotter と互換性があります。ただし、EKS Auto Mode にはスナップショットコントローラーがありません。スナップショットコントローラーのインストールと設定はお客様の責任となります。詳細については、「the section called “CSI スナップショットコントローラー”」を参照してください。

EKS Auto Mode のストレージ機能を参照する以下の `VolumeSnapshotClass` を確認してください。

```
apiVersion: snapshot.storage.k8s.io/v1
kind: VolumeSnapshotClass
metadata:
  name: auto-ebs-vsclass
driver: ebs.csi.eks.amazonaws.com
deletionPolicy: Delete
```

Kubernetes CSI Snapshotter の詳細をご覧ください。

加速ワークロードをデプロイする

このチュートリアルでは、Amazon EKS Auto Mode がハードウェアアクセラレーションワークロードの開始をどのようにシンプルにしているかを説明します。Amazon EKS Auto Mode は、コンピューティング、ネットワーキング、ロードバランシング、ストレージ、Identity and Access Management の各機能を追加の設定なしで実現する主要なインフラストラクチャコンポーネントを自動化することで、クラスター自体にとどまらずさまざまな運用を簡素化します。

Amazon EKS Auto Mode には、NVIDIA ドライバーや AWS Neuron ドライバーなど、特定のインスタンスタイプに必要なドライバーとデバイスプラグインが含まれています。これらのコンポーネントをお客様側でインストールおよび更新する必要はありません。

EKS Auto Mode が次のアクセラレーター用のドライバーを自動的に管理します。

- AWS Trainium
- AWS Inferentia
- NVIDIA GPUs on Amazon EC2 加速インスタンス

> **Note**
>
> EKS Auto Mode には、Kubernetes 用の NVIDIA デバイスプラグインが含まれています。このプラグインは、自動的に動作し、クラスターにデーモンセットとして表示されません。

ネットワーキングに関するその他のサポート:

- Elastic Fabric Adapter (EFA) – EKS クラスター内の EFA デバイスの管理については、「the section called “EFA デバイス”」を参照してください。

Amazon EKS Auto Mode により、アクセラレーター用のドライバーとデバイスプラグインを管理するという手間が不要になります。

また、クラスターがゼロにスケーリングされるので、コスト削減というメリットも得られます。実行中のワークロードがなくなったらインスタンスを終了するように、EKS Auto Mode を設定できます。これはバッチベースの推論ワークロードに便利です。

以下に、Amazon EKS Auto Mode で加速ワークロードを開始する方法の例を示します。

前提条件

- Amazon EKS Auto Mode で Kubernetes クラスターが設定されていること。
- `general-purpose` または `system` のいずれかのマネージドノードプールが有効であれば、`default` EKS ノードクラスが作成されること。

ステップ 1: GPU ワークロードをデプロイする

この例では、45 GB の GPU メモリを必要とする NVIDIA ベースのワークロード用に NodePool を作成します。EKS Auto Mode では、Kubernetes スケジュール制約を使用して、インスタンス要件を定義します。

Amazon EKS Auto Mode `NodePool` とサンプルの `workload` をデプロイするには、NodePool とポッドに関する以下の定義を確認し、`nodepool-gpu.yaml` および `pod.yaml` として保存します。

nodepool-gpu.yaml

```
apiVersion: karpenter.sh/v1
kind: NodePool
```

```
metadata:
  name: gpu
spec:
  disruption:
    budgets:
    - nodes: 10%
    consolidateAfter: 1h
    consolidationPolicy: WhenEmpty
  template:
    metadata: {}
    spec:
      nodeClassRef:
        group: eks.amazonaws.com
        kind: NodeClass
        name: default
      requirements:
        - key: "karpenter.sh/capacity-type"
          operator: In
          values: ["on-demand"]
        - key: "kubernetes.io/arch"
          operator: In
          values: ["amd64"]
        - key: "eks.amazonaws.com/instance-family"
          operator: In
          values:
          - g6e
          - g6
      taints:
        - key: nvidia.com/gpu
          effect: NoSchedule
      terminationGracePeriod: 24h0m0s
```

pod.yaml

```
apiVersion: v1
kind: Pod
metadata:
  name: nvidia-smi
spec:
  nodeSelector:
    eks.amazonaws.com/compute-type: auto
  restartPolicy: OnFailure
  containers:
```

```
  - name: nvidia-smi
    image: public.ecr.aws/amazonlinux/amazonlinux:2023-minimal
    args:
    - "nvidia-smi"
    resources:
      requests:
        memory: "30Gi"
        cpu: "3500m"
        nvidia.com/gpu: 1
      limits:
        memory: "30Gi"
        nvidia.com/gpu: 1
  tolerations:
  - key: nvidia.com/gpu
    effect: NoSchedule
    operator: Exists
```

eks.amazonaws.com/compute-type: auto セレクターでは、ワークロードを Amazon EKS Auto Mode ノードにデプロイする必要があることに注意してください。また、NodePool は NVIDIA GPU に対する許容範囲が設定されたポッドに限りスケジュールできるというテイントを設定します。

NodePool とワークロードをクラスターに適用します。

```
kubectl apply -f nodepool-gpu.yaml
kubectl apply -f pod.yaml
```

以下の出力が表示されます。

```
nodepool.karpenter.sh/gpu created
pod/nvidia-smi created
```

数秒待ってからクラスター内のノードを確認します。これで、新しいノードが Amazon EKS Auto Mode クラスターにプロビジョニングされるはずです。

```
> kubectl get nodes

NAME        TYPE           CAPACITY    ZONE         NODE                   READY   AGE
gpu-dnknr   g6e.2xlarge    on-demand   us-west-2b   i-02315c7d7643cdee6   True    76s
```

ステップ 2: 検証する

Amazon EKS Auto Mode が以下の Kubernetes スケジュール制約に従って、L40S GPU 搭載のインスタンスを必要とするワークロードとして g6.2xlarge ではなく g6e.2xlarge が起動されていることを確認できます。

```
...
  nodeSelector:
    eks.amazonaws.com/instance-gpu-name: l40s
...
    requests:
        memory: "30Gi"
        cpu: "3500m"
        nvidia.com/gpu: 1
      limits:
        memory: "30Gi"
        nvidia.com/gpu: 1
```

次のコマンドを実行して、コンテナのログを参照します。

```
kubectl logs nvidia-smi
```

サンプル出力:

```
+-----------------------------------------------------------------------------------------
+
| NVIDIA-SMI 535.230.02             Driver Version: 535.230.02   CUDA Version: 12.2
 |
|-----------------------------------------+----------------------
+----------------------+
| GPU  Name                 Persistence-M | Bus-Id        Disp.A | Volatile Uncorr. ECC
 |
| Fan  Temp   Perf          Pwr:Usage/Cap |         Memory-Usage | GPU-Util  Compute M.
 |
|                                         |                      |               MIG M.
 |
|=========================================+======================
+======================|
|   0  NVIDIA L40S                    On  | 00000000:30:00.0 Off |                    0
 |
| N/A   27C    P8              23W / 350W |      0MiB / 46068MiB |      0%      Default
 |
```

```
|                                         |                      |                  N/A
 |
+-----------------------------------------+----------------------
+----------------------+

+-----------------------------------------------------------------------------------------
+
| Processes:
 |
|  GPU   GI   CI        PID   Type   Process name                             GPU Memory
 |
|        ID   ID                                                              Usage
 |
|
=============================================================================================|
|  No running processes found
 |
+-----------------------------------------------------------------------------------------
+
```

出力を見ると、コンテナは NVIDIA GPU 搭載のインスタンスで実行されていること、デバイスドライバーは Amazon EKS Auto Mode によって管理されるためインストールする必要がなかったことを確認できます。

ステップ 3: クリーンアップする

作成されたすべてのオブジェクトを削除するには、kubectl を使用してサンプルのデプロイと NodePool を削除します。これでノードが終了します。

```
kubectl delete -f nodepool-gpu.yaml
kubectl delete -f pod.yaml
```

NodePool 参照の例

NVIDIA NodePool を作成する

NodePool で定義する内容は次のとおりです。

- g6e と g6 ファミリーのインスタンスを開始するのみ
- 1 時間が過ぎても空のままであればノードを統合する

- consolidateAfter が 1 時間というのは、ワークロードの急増に対応し、ノードのチャーンを抑制するのに十分な値です。consolidateAfter はワークロード要件に基づいて調整できます。

NodePool を GPU インスタンスファミリーおよび統合で使用する例

```
apiVersion: karpenter.sh/v1
kind: NodePool
metadata:
  name: gpu
spec:
  disruption:
    budgets:
    - nodes: 10%
    consolidateAfter: 1h
    consolidationPolicy: WhenEmpty
  template:
    metadata: {}
    spec:
      nodeClassRef:
        group: eks.amazonaws.com
        kind: NodeClass
        name: default
      requirements:
        - key: "karpenter.sh/capacity-type"
          operator: In
          values: ["on-demand"]
        - key: "kubernetes.io/arch"
          operator: In
          values: ["amd64"]
        - key: "eks.amazonaws.com/instance-family"
          operator: In
          values:
          - g6e
          - g6
      terminationGracePeriod: 24h0m0s
```

eks.amazonaws.com/instance-gpu-name を設定するのではなく、eks.amazonaws.com/instance-family を使用してインスタンスファミリーを指定することもできます。この他にスケジュールのレビューに影響を与えるラベルとしてよく知られたものについては、「the section called “EKS Auto Mode でサポートされているラベル”」を参照してください。

固有のストレージ要件がある場合は、NodePool で参照する NodeClass を独自に作成することで、ノードのエフェメラルストレージの iops、size、throughput を調整できます。設定可能な NodeClass オプションについて確認してください。

NodeClass 用にストレージを設定する例

```
apiVersion: eks.amazonaws.com/v1
kind: NodeClass
metadata:
  name: gpu
spec:
  ephemeralStorage:
    iops: 3000
    size: 80Gi
    throughput: 125
```

AWS Trainium と AWS Inferentia の NodePool を定義する

以下の NodePool では、Inferentia と Trainium ファミリーのインスタンスを開始するだけという eks.amazonaws.com/instance-category 設定を定義しています。

```
        - key: "eks.amazonaws.com/instance-category"
          operator: In
          values:
            - inf
            - trn
```

EKS Auto Mode 設定を構成する

この章ではAmazon エラスティックKubernetesサービス (EKS) Auto Mode クラスターの特定の特徴を設定する方法について説明します。EKS Auto Mode はほとんどのインフラストラクチャコンポーネントを自動的に管理しますが、ワークロードの要件を満たすように特定の機能をカスタマイズできます。

このトピックで説明されている設定オプションを使用して、自動インフラストラクチャ管理の利点を維持しながら、ネットワーク設定、コンピューティングリソース、ロードバランシングの動作を変更できます。設定を変更する前に、以下のセクションで使用可能なオプションを確認して、ニーズに最適なアプローチを決定してください。

どの機能を設定したいですか?	設定オプション
ノードネットワークとストレージ • パブリックサブネットとプライベートサブネットにまたがるノードの配置を設定する • ノードアクセスコントロールのカスタムセキュリティグループを定義する • ネットワークアドレス変換 (SNAT) ポリシーをカスタマイズする • Kubernetes ネットワークポリシー (詳細なネットワークポリシーのログ記録とモニタリング) を有効にする • エフェメラルストレージパラメータ (サイズ、IOPS、スループット) を設定する • カスタム KMS キーを使用して暗号化されたエフェメラルストレージを設定する • ポッドトラフィックをノードから別々のサブネットに分離する	the section called “ノードクラスの作成”
ノードコンピューティングリソース • 特定の EC2 インスタンスタイプとファミリーを選択する • CPU アーキテクチャを定義する (x86_64、ARM64) • キャパシティタイプを設定する (オンデマンド、スポット) • アベイラビリティーゾーンを指定する • ノードのテイントとラベルを設定する • CPU とメモリの使用量にリソースの制限を設定する	the section called “ノードプールの作成”
静的キャパシティノードプール	the section called “静的ノードプールを作成する”

どの機能を設定したいですか?	設定オプション
• ポッドの需要に関係なく、一定数のノードを維持する • リザーブドインスタンスの予測可能な容量を設定する • インフラストラクチャのフットプリントを一定にするために特定のレプリカ数を設定する • 静的ノードプールを手動でスケールする • 静的キャパシティノードプールのステータスをモニタリングする	
Application Load Balancer の設定 • 内部ロードバランサーまたはインターネット向けロードバランサーをデプロイする • クロスゾーン負荷分散の設定 • アイドルタイムアウト期間を設定する • HTTP/2 および WebSocket サポートを有効にする • ヘルスチェックパラメータを設定する • TLS 証明書設定を指定する • ターゲットグループの属性を定義する • IP アドレスタイプを設定する (IPv4、デュアルスタック)	the section called “ingress class の作成”

どの機能を設定したいですか?	設定オプション
Network Load Balancer の設定 • 直接ポッド IP ルーティングを設定する • クロスゾーン負荷分散の有効化 • 接続のアイドルタイムアウトを設定する • ヘルスチェックパラメータを設定する • サブネットの配置を指定する • IP アドレスタイプを設定する (IPv4、デュアルスタック) • クライアントソース IP の保存を設定する • ターゲットグループの属性を定義する	the section called “サービスの作成”
マルチクラスターターゲットグループ • ロードバランサーコントローラーがターゲットグループメンバーシップを管理する方法を理解する • マルチクラスターサポートを有効にして、複数の Kubernetes クラスター間でターゲットグループを共有する • 共有ターゲットグループ内の他のソースからの任意のターゲットをサポートする	the section called “マルチクラスターターゲットグループ”

どの機能を設定したいですか?	設定オプション
マネージドコントローラー向けに RBAC を拡張する • EKS Auto Mode のマネージドコントローラーがクラスターに対して認証する仕組みを理解する • `eks:managed` グループを介してマネージドコントローラーに追加の Kubernetes RBAC を付与する • 最小特権のために `Role` / `RoleBinding` および `resourceNames` で付与するスコープを設定する	the section called “マネージドコントローラーに RBAC を付与する”
ストレージクラスの設定 • EBS ボリュームタイプ (gp3、io1、io2 など) を定義する • ボリューム暗号化と KMS キーの使用を設定する • IOPS とスループットパラメータを設定する • デフォルトのストレージクラスとして設定する • プロビジョニングされたボリュームのカスタムタグを定義する	the section called “StorageClass を作成する”

どの機能を設定したいですか?	設定オプション
ODCR 使用の制御 • EC2 オンデマンドのキャパシティ予約にワークロードデプロイを設定する • ターゲット容量の使用には、ID で特定の ODCR を明示的に選択する • タグベースの選択を使用して ODCR のグループをターゲットする • クロスアカウントシナリオの AWS アカウントを所有して ODCR をフィルタリングする • ワークロードがオープン ODCR を自動的に使用するかどうかを管理する	the section called “キャパシティ予約を制御する”
ノードの高度なセキュリティ • FIPS 140-2 検証済み暗号化モジュールを有効にする • カーネルロックダウンセキュリティモードを設定する	the section called “アドバンスドセキュリティ”
コスト最適化 • 統合、ビンパッキング、適切なサイズ設定によってコンピューティングコストを削減する方法を理解する • 統合をブロックする設定を特定する (do-not-disrupt 、制限付き PDB) • NodePool limits をコスト上限として設定する • インスタンスファミリーとサイズを制限してオールインコストを制御する • バースト CI/CD ワークロードに推奨パターンを適用する	EKS Auto Mode でのコスト最適化

コンピューティングとスケーリングの設定

Amazon EKS Auto Mode がワークロードのコンピューティングをどのようにプロビジョニングおよびスケーリングするかを設定します。以下のトピックを参照して、ノードクラス、ノードプール、キャパシティ、および関連するコンピューティングコントロールをカスタマイズします。

トピック

- Amazon EKS のノードクラスを作成する
- EKS 自動モードI 用のノードプールを作成する
- EKS Auto Mode での静的キャパシティノードプール
- 組み込み NodePool を有効または無効にする
- EKS 自動モードでキャパシティ予約へのワークロードのデプロイを制御する
- Local Zone に EKS 自動モードノードをデプロイする
- EKS Auto Mode の組織コントロールを更新する

Amazon EKS のノードクラスを作成する

Amazon EKS ノードクラスは、EKS Auto Mode マネージドノードの設定をきめ細かく制御できるテンプレートです。ノードクラスは、ネットワーク設定、ストレージ設定、リソースタグ付けなど、EKS クラスター内のノードのグループに適用されるインフラストラクチャレベルの設定を定義します。このトピックでは、特定の運用要件を満たすようにノードクラスを作成および設定する方法について説明します。

EKS Auto Mode がデフォルト設定を超えて EC2 インスタンスをプロビジョニングおよび設定する方法をカスタマイズする必要がある場合は、ノードクラスを作成すると、重要なインフラストラクチャパラメータを正確に制御できます。例えば、セキュリティを強化するためにプライベートサブネットの配置を指定したり、パフォーマンス重視のワークロード用にインスタンスエフェメラルストレージを設定したり、コスト配分のためにカスタムタグ付けを適用したりできます。

ポッドスパースまたは大規模ワークロードに関する考慮事項

EKS Auto Mode は、デフォルトでプレフィックス委任 (/28 プレフィックス) を使用します。各ノードは 16 個のポッド IP を予約します。これは、それらのスロットを使い切るポッド密度の高いワークロードに対して効率的です。アベイラビリティーゾーンあたり数百個を超えるノードをターゲットとするポッドスパースワークロード (ML または GPU ワークロードで一般的なノードあたり数個のポッド、またはアンチアフィニティが多いワークロード) の場合、セカンダリ IP モード ("32") はよ

り最適化された設定です。ノードごとに 16 個を予約するのではなく、ポッドごとに 1 つの IP を割り当てることで、/20 ポッドサブネットの実効容量を拡張します。

詳細については、「the section called “ポッドのセカンダリ IP モード”」を参照してください。ポッドをノードとは別のサブネットに保持するには (割り当てモードとは無関係)、「the section called “ポッドのサブネットとセキュリティグループを分離する”」を参照してください。

ノードクラスを作成する

NodeClass を作成するには、次の手順に従います。

1. 自分のノードクラス設定で YAML ファイル (nodeclass.yaml など) を作成します
2. kubectl を使用してこの設定をクラスターに適用します
3. ノードプール設定でノードクラスを参照します。詳細については、「the section called “ノードプールの作成”」を参照してください。

kubectl がインストールおよび設定されている必要があります。詳細については、「セットアップ」を参照してください。

基本的なノードクラスの例

ノードクラスの例を次に示します。

```
apiVersion: eks.amazonaws.com/v1
kind: NodeClass
metadata:
  name: private-compute
spec:
  subnetSelectorTerms:
    - tags:
        Name: "private-subnet"
        kubernetes.io/role/internal-elb: "1"
  securityGroupSelectorTerms:
    - tags:
        Name: "eks-cluster-sg"
  ephemeralStorage:
    size: "160Gi"
```

この NodeClass により、ノード上のエフェメラルストレージの量が増加します。

次を使用してこの設定を適用します。

```
kubectl apply -f nodeclass.yaml
```

次に、ノードプール設定でノードクラスを参照します。詳細については、「the section called “ノードプールの作成”」を参照してください。

ノードクラスのアクセスエントリを作成する

カスタムノードクラスを作成する場合は、ノードがクラスターに参加できるように EKS アクセスエントリを作成する必要があります。組み込みのノードクラスとノードプールを使用すると、EKS によってアクセスエントリが自動的に作成されます。

アクセスエントリがどのように機能するかについては、「the section called “アクセスエントリ”」を参照してください。

EKS Auto Mode ノードクラスのアクセスエントリを作成するときは、EC2 のアクセスエントリタイプを使用する必要があります。

CLI でアクセスエントリを作成する

EC2 ノードのアクセスエントリを作成して EKS Auto Node ポリシーを関連付けるには:

次の CLI コマンドを、使用するクラスター名とノードロール ARN で更新します。ノードロール ARN は、ノードクラス YAML で指定されます。

```
# Create the access entry for EC2 nodes
aws eks create-access-entry \
  --cluster-name <cluster-name> \
  --principal-arn <node-role-arn> \
  --type EC2

# Associate the auto node policy
aws eks associate-access-policy \
  --cluster-name <cluster-name> \
  --principal-arn <node-role-arn> \
  --policy-arn arn:aws:eks::aws:cluster-access-policy/AmazonEKSAutoNodePolicy \
  --access-scope type=cluster
```

CloudFormation でアクセスエントリを作成する

EC2 ノードのアクセスエントリを作成して EKS Auto Node ポリシーを関連付けるには:

次の CloudFormation を、使用するクラスター名とノードロール ARN で更新します。ノードロール ARN は、ノードクラス YAML で指定されます。

```
EKSAutoNodeRoleAccessEntry:
  Type: AWS::EKS::AccessEntry
  Properties:
    ClusterName: <cluster-name>
    PrincipalArn: <node-role-arn>
    Type: "EC2"
    AccessPolicies:
      - AccessScope:
          Type: cluster
        PolicyArn: arn:aws:eks::aws:cluster-access-policy/AmazonEKSAutoNodePolicy
  DependsOn: [ <cluster-name> ] # previously defined in CloudFormation
```

CloudFormation スタックのデプロイの詳細については、「Getting started with CloudFormation」を参照してください

ノードクラスの指定

```
apiVersion: eks.amazonaws.com/v1
kind: NodeClass
metadata:
  name: my-node-class
spec:
  # Required fields

  # role and instanceProfile are mutually exclusive fields.
  role: MyNodeRole  # IAM role for EC2 instances
  # instanceProfile: eks-MyNodeInstanceProfile  # IAM instance-profile for EC2
 instances

  subnetSelectorTerms:
    - tags:
        Name: "private-subnet"
        kubernetes.io/role/internal-elb: "1"
    # Alternative using direct subnet ID
    # - id: "subnet-0123456789abcdef0"

  securityGroupSelectorTerms:
    - tags:
        Name: "eks-cluster-sg"
    # Alternative approaches:
```

```
    # - id: "sg-0123456789abcdef0"
    # - name: "eks-cluster-security-group"

  # Optional: Pod subnet selector for advanced networking
  podSubnetSelectorTerms:
    - tags:
        Name: "pod-subnet"
        kubernetes.io/role/pod: "1"
    # Alternative using direct subnet ID
    # - id: "subnet-0987654321fedcba0"
  # must include Pod security group selector also
  podSecurityGroupSelectorTerms:
    - tags:
        Name: "eks-pod-sg"
    # Alternative using direct security group ID
    # - id: "sg-0123456789abcdef0"

  # Optional: Selects on-demand capacity reservations and capacity blocks
  # for EKS Auto Mode to prioritize.
  capacityReservationSelectorTerms:
    - id: cr-56fac701cc1951b03
    # Alternative Approaches
    - tags:
        Name: "targeted-odcr"
      # Optional owning account ID filter
      owner: "012345678901"

  # Optional fields
  snatPolicy: Random  # or Disabled

  networkPolicy: DefaultAllow  # or DefaultDeny
  networkPolicyEventLogs: Disabled  # or Enabled

  ephemeralStorage:
    size: "80Gi"    # Range: 1-59000Gi or 1-64000G or 1-58Ti or 1-64T
    iops: 3000      # Range: 3000-16000
    throughput: 125 # Range: 125-1000
    # Optional KMS key for encryption
    kmsKeyID: "arn:aws:kms:region:account:key/key-id"
    # Accepted formats:
    # KMS Key ID
    # KMS Key ARN
    # Key Alias Name
    # Key Alias ARN
```

```
  advancedNetworking:
    # Optional: Controls whether public IP addresses are assigned to instances that are
 launched with the nodeclass.
    # If not set, defaults to the MapPublicIpOnLaunch setting on the subnet.
    associatePublicIPAddress: false

    # Optional: Forward proxy, commonly requires certificateBundles as well
    # for EC2, see https://repost.aws/knowledge-center/eks-http-proxy-containerd-
automation
    httpsProxy: http://192.0.2.4:3128 #commonly port 3128 (Squid) or 8080 (NGINX) #Max
 255 characters
    #httpsProxy: http://[2001:db8::4]:3128 # IPv6 address with port, use []
    noProxy: #Max 50 entries
        - localhost #Max 255 characters each
        - 127.0.0.1
        #- ::1 # IPv6 localhost
        #- 0:0:0:0:0:0:0:1 # IPv6 localhost
        - 169.254.169.254 # EC2 Instance Metadata Service
        #- [fd00:ec2::254] # IPv6 EC2 Instance Metadata Service
        # Domains to exclude, put all VPC endpoints here
        - .internal
        - .eks.amazonaws.com
    # ipv4PrefixSize is default to Auto which is prefix and fallback to secondary IP.
 "32" is the secondary IP mode.
    ipv4PrefixSize: Auto # or "32"

    # enableV4Egress is default to true. Setting it to false when using network policy
 or blocking IPv4 traffic in IPv6 clusters
    enableV4Egress: false

    # Optional: Static network interface configuration for EFA workloads
    networkInterfaces:
      # The primary ENI (networkCardIndex 0, deviceIndex 0) must use interfaceType:
 interface
      # secondaryIPv4Count and secondaryIPv4PrefixCount are only supported on
 networkCardIndex 0
    - deviceIndex: 0
      interfaceType: interface
      networkCardIndex: 0
      secondaryIPv4Count: 10
    - deviceIndex: 1
      interfaceType: interface
      networkCardIndex: 0
```

```
      secondaryIPv4PrefixCount: 2 # Only one of secondaryIPv4Count or
secondaryIPv4PrefixCount per ENI
    - deviceIndex: 0
      interfaceType: efa-only
      networkCardIndex: 1

  advancedSecurity:
    # Optional, US regions only: Specifying `fips: true` will cause nodes in the
nodeclass to run FIPS compatible AMIs.
    fips: false

  # Optional: Custom certificate bundles.
  certificateBundles:
    - name: "custom-cert"
      data: "base64-encoded-cert-data"

  # Optional: EC2 Placement Group
  placementGroupSelector:
    name: "targeted-pg"
    # Alternative use direct placement group ID
    # id: "pg-02465754522cda020"

  # Optional: Additional EC2 tags (with restrictions)
  tags:
    Environment: "production"
    Team: "platform"
    # Note: Cannot use restricted tags like:
    # - kubernetes.io/cluster/*
    # - karpenter.sh/provisioner-name
    # - karpenter.sh/nodepool
    # - karpenter.sh/nodeclaim
    # - karpenter.sh/managed-by
    # - eks.amazonaws.com/nodeclass
```

考慮事項

- インスタンスのローカルストレージの容量を確認したい場合は、ノードを記述してエフェメラルストレージリソースを表示できます。
- ボリュームの暗号化 – インスタンスの読み取り専用ルートボリュームおよび読み取り/書き込みデータボリュームを暗号化するため、EKS は設定されたカスタム KMS キーを使用します。
- ノード IAM ロールの置換 – NodeClass に関連付けられているノード IAM ロールを変更する場合は、新しいアクセスエントリを作成する必要があります。EKS は、クラスターの作成

時にノード IAM ロールのアクセスエントリを自動的に作成します。ノード IAM ロールには AmazonEKSAutoNodePolicy EKS アクセスポリシーが必要です。詳細については、「the section called “アクセスエントリ”」を参照してください。

- 最大 Pod 密度 – EKS では、ノード上の Pod の最大数が 110 に制限されています。この制限は、既存の最大 Pod を計算した後に適用されます。詳細については、「the section called “Amazon EC2 インスタンスタイプ”」を参照してください。
- タグ - Kubernetes から EC2 にタグを伝播させる場合は、追加の IAM アクセス許可を設定する必要があります。詳細については、「the section called “ID とアクセス”」を参照してください。
- デフォルトのノードクラス – カスタムノードクラスに `default` という名前を付けないでください。これは、EKS Auto Mode には、少なくとも 1 つの組み込み `NodePool` を有効にすると自動的にプロビジョニングされる `default` と呼ばれる `NodeClass` が含まれるためです。組み込み `NodePools` を有効にする方法については、「the section called “組み込みノードプールを確認する”」を参照してください。
- 複数のサブネットを使用した **`subnetSelectorTerms`** 動作 – `subnetSelectorTerms` 条件に一致するサブネットまたは ID で指定したサブネットが複数ある場合、EKS Auto Mode はサブネット全体に分散されたノードを作成します。
  - サブネットが異なるアベイラビリティーゾーン (AZ) にある場合、Pod トポロジスプレッド制約や Topology Aware Routing といった Kubernetes 機能を使用して、Pod とトラフィックをそれぞれゾーン間に分散できます。
  - 同じ AZ 内に `subnetSelectorTerms` に一致するサブネットが複数ある場合、EKS Auto Mode はその AZ のサブネット全体に分散した各ノードに Pod を作成します。EKS Auto Mode は、同じ AZ 内の他のサブネットの各ノードにセカンダリネットワークインターフェイスを作成します。各サブネットで使用可能な IP アドレスの数に基づいて選択し、サブネットをより効率的に使用します。ただし、EKS Auto Mode が各 Pod に使用するサブネットを指定することはできません。特定のサブネットで Pod を実行する必要がある場合は、代わりに the section called “ポッドのサブネットとセキュリティグループを分離する” を使用してください。
- プレイスメントグループ戦略の制限 – 各プレイスメントグループ戦略 (クラスター、パーティション、スプレッド) には、インスタンスタイプ、AZ、容量に特定の制限があります。詳細については、Amazon EC2 ユーザーガイドの「プレイスメントグループのプレイスメント戦略」を参照してください。
- スプレッドプレイスメントグループ – 7 インスタンスの制限 – ラックレベルのスプレッドプレイスメントグループでは、1 グループあたり、アベイラビリティーゾーンごとに最大 7 つの実行中のインスタンスを使用できます。これにより、以下のエッジケースが作成されます。

  - 容量不足によるドリフト置き換えのブロック – EKS Auto Mode は、古いノードを終了する前に代替ノードを起動します。¢ スプレッド PG が AZ に 7 つのインスタンスを持つ場合、置き換えの起動は失敗し、スロットが解放されるまでドリフトしたノードは実行されたままになります。
  - すべての AZ が容量上限に達している場合 – スプレッド PG 内のすべての AZ が 7 インスタンスの制限に達している場合、置き換えをスケジュールすることはできません。ドリフトノードまたは統合候補ノードは無期限に実行され続けます。
  - プレイスメントグループ外へのフォールバックなし – EKS Auto Mode はプレイスメントグループ外での代替インスタンスの起動を試みません。
  - 回避策 – `WhenEmpty` 統合ポリシー (`consolidationPolicy: WhenEmpty`) を使用します。ノードは、デーモンセット以外のポッドがすべてドレインオフされた後にのみ削除されるため、先に置き換えを起動することなく、PG スロットを解放できます。ドリフトは常に統合ポリシーに関係なく replace-then-delete を使用するため、ドリフトは容量制限に達したままブロックされることに注意してください。
- クラスタープレイスメントグループ AZ 固定 – 最初のインスタンスがクラスタープレイスメントグループで起動されると、PG はその AZ に固定されます。NodePool で複数の AZ が許可されている場合、初期スケールアップ中の並列起動が競合する可能性があります。1 つは成功し、AZ を固定します。残りは容量不足エラーで失敗します。NodePool 要件で AZ を固定して、一時的な障害を回避します。
- パーティションプレイスメントグループ – パーティションプレイスメントグループは、標準の EC2 制限を超える追加の制約なしでサポートされています。
- 統合によってポッドがプレイスメントグループから移動されることがあります – ポッドにプレイスメントグループのスケジューリング制約 (`eks.amazonaws.com/placement-group-id` の `nodeSelector` など) がない場合、統合によってポッドが PG 外のノードに移動されることがあります。プレイスメントグループメンバーシップを必要とするアプリケーションは、これをポッドレベルの制約で表現する必要があります。
- 存在しない、または削除されたプレイスメントグループ – `NodeClass` が存在しない、または削除されたプレイスメントグループを参照する場合、インスタンスは起動されません。プレイスメントグループ ID 形式は登録時に検証されますが、存在は起動時にのみチェックされます。ノードの実行中にプレイスメントグループが削除された場合、既存のノードはドリフト済みとしてマークされますが、ドリフト置き換えの起動もブロックされるため、無期限に実行されたままになります。
- Hugepages – ノードで hugepages を設定する方法については、「the section called “ノードで hugepages を設定する”」を参照してください。
- Kubelet 設定 – ポッド密度、エビクションしきい値、ログローテーションなどの kubelet 設定の上書きについては、「the section called “ノードでの kubelet 設定の構成”」を参照してください。

- カーネル sysctls – ノードでカーネル sysctls を設定する方法については、「the section called "ノードでカーネル sysctl を設定する"」を参照してください。

ポッドのサブネットとセキュリティグループを分離する

podSubnetSelectorTerms と podSecurityGroupSelectorTerms フィールドにより、ポッドがノードとは異なるサブネットとセキュリティグループを使用できるようになるため、高度なネットワーク設定が可能になります。両方のフィールドを一緒に指定する必要があります。この分離により、ネットワークトラフィックのルーティングとセキュリティポリシーの制御が強化されます。

> **Note**
>
> この機能は、EKS Auto Mode 以外のコンピューティング用の VPC CNI で使用されるポッドのセキュリティグループ (SGPP) 機能とは異なります。SGPP は EKS Auto Mode ではサポートされていません。代わりに、NodeClass で podSecurityGroupSelectorTerms を使用して、ポッドトラフィックに個別のセキュリティグループを適用します。セキュリティグループは NodeClass レベルで適用されます。つまり、その NodeClass を使用するノード上のすべてのポッドは、同じのポッドセキュリティグループを共有します。

仕組み

podSubnetSelectorTerms と podSecurityGroupSelectorTerms を設定する場合:

1. ノードのプライマリ ENI は、subnetSelectorTerms および securityGroupSelectorTerms のサブネットとセキュリティグループを使用します。このインターフェイスには、ノード自身の IP アドレスのみが割り当てられます。
2. EKS Auto Mode では、podSubnetSelectorTerms に一致するサブネット内にセカンダリ ENI が作成され、podSecurityGroupSelectorTerms のセキュリティグループがそれらにアタッチされます。ポッド IP アドレスは、デフォルトで /28 プレフィックスを使用してこれらのセカンダリ ENI から割り当てられます。連続するプレフィックスブロックが使用できない場合、セカンダリ IP (/32) への自動フォールバックが行われます。advancedNetworking で ipv4PrefixSize が "32" に設定されている場合、セカンダリ IP のみが使用されます。
3. podSecurityGroupSelectorTerms で指定されたセキュリティグループは、VPC 内のポッドトラフィックに適用されます。VPC 外を送信先とするトラフィックの場合、ソースネットワークアドレス変換 (SNAT) はポッド IP をノード IP に変換するため、ポッドはノードのプライマリ ENI (およびそのセキュリティグループ) を使用します。この動作は、NodeClass の snatPolicy フィールドで変更できます。

ユースケース

以下の場合に podSubnetSelectorTerms および podSecurityGroupSelectorTerms を使用します。

- 異なるセキュリティグループを適用して、ノードとポッドのトラフィックを個別に制御する必要がある場合。
- インフラストラクチャトラフィック (ノード間の通信) とアプリケーショントラフィック (ポッド間の通信) を分離する必要がある場合。
- Pod サブネットとは異なるネットワーク設定をノードサブネットに適用する。
- Pod トラフィックに影響を与えずに、ノードトラフィック専用のリバースプロキシまたはネットワークフィルタリングを設定する。advancedNetworking と certificateBundles を使用して、リバースプロキシと、プロキシの自己署名証明書またはプライベート証明書を定義する。

設定例

```
apiVersion: eks.amazonaws.com/v1
kind: NodeClass
metadata:
  name: advanced-networking
spec:
  role: MyNodeRole

  # Subnets and security groups for EC2 instances (nodes)
  subnetSelectorTerms:
    - tags:
        Name: "node-subnet"
        kubernetes.io/role/internal-elb: "1"

  securityGroupSelectorTerms:
    - tags:
        Name: "eks-cluster-sg"

  # Separate subnets and security groups for Pods
  podSubnetSelectorTerms:
    - tags:
        Name: "pod-subnet"
        kubernetes.io/role/pod: "1"

  podSecurityGroupSelectorTerms:
  - tags:
```

```
      Name: "eks-pod-sg"
```

個別のポッドサブネットとセキュリティグループに関する考慮事項

- セキュリティグループの範囲: podSecurityGroupSelectorTerms のセキュリティグループはセカンダリ ENI にアタッチされ、VPC 内のポッドトラフィックに適用されます。SNAT が有効になっている場合 (デフォルトの snatPolicy: Random)、VPC から出るトラフィックはノードのプライマリ ENI IP アドレスに変換されるため、securityGroupSelectorTerms からのノードのセキュリティグループは代わりにそのトラフィックに適用されます。snatPolicy: Disabled を設定すると、ポッドはすべてのトラフィックに独自の IP アドレスを使用するため、ルーティンググループとセキュリティグループが適切に設定されていることを確認する必要があります。
- NodeClass レベルの粒度: ポッドセキュリティグループは、NodeClass を使用してノードでスケジュールされたすべてのポッドに適用されます。異なるセキュリティグループを異なるワークロードに適用するには、個別の NodeClass リソースと NodePool リソースを作成し、テイント、許容範囲、またはノードセレクターを使用してワークロードを適切なノードにスケジュールします。
- ポッド密度の低下: ノードのプライマリネットワークインターフェイスはノード IP 用に予約されており、ポッドに使用できないため、各ノードで実行できるポッドの数が少なくなります。
- サブネットセレクターの制限: 標準 subnetSelectorTerms および securityGroupSelectorTerms 設定は、ポッドサブネットまたはセキュリティグループの選択には適用されません。
- ネットワーク計画: ワークロード要件をサポートするために、ノードサブネットと Pod サブネットの両方に十分な IP アドレススペースを確保します。
- ルーティング設定: ノードサブネットとポッドサブネット間の通信用に、ポッドサブネットのルートテーブルとネットワークアクセスコントロールリスト (ACL) が適切に設定されていることを確認します。
- アベイラビリティーゾーン: 複数の AZ にわたって Pod サブネットを作成したことを確認します。特定のポッドサブネットを使用している場合、そのサブネットはノードサブネット AZ と同じ AZ に存在している必要があります。

ポッドのセカンダリ IP モード

ipv4PrefixSize フィールドは、ノードへのセカンダリ IP アドレスのみを割り当てることで、高度なネットワーク設定を有効にします。この機能はノードにプレフィックス (/28) を割り当てず、1 つのセカンダリ IP のみを MinimalIPTarget として維持します。

advancedNetworking.ipv4PrefixSize は、次の 2 つの値を受け入れます。Auto (デフォルト、プレフィックス委任、ポッド作成速度の向上、ノードごとに最低 16 個の IP を予約) および "32" (セカンダリ IP モード、ポッドごとに 1 個の IP、ポッドごとの AssignPrivateIpAddresses 呼び出しを追加、インスタンスタイプの ENI IP 制限によってノードあたりのポッド数が制限される)。

Auto モードでは、EKS Auto Mode は各ノードに /28 プレフィックス (16 個の IP) を事前に割り当て、ポッドの需要が現在のブロックの容量を超えたときに別の /28 を追加するため、16 はノードごとの上限ではなく最小値です。

計画には次の数値を使用しますが、実際のクラスターは異なります。ポッドサブネットがフラグメント化され、連続した /28 が解放されない場合、Auto モードはそのノードの IP ごとの割り当てにフォールバックします。ノードごとの IP 数は、以下に示すプレフィックスごとに 16 個のパターンを下回ります。

例 A — ノードに 1 つのポッド

- セカンダリ IP モード: ポッドの IP 1 個 + ノード の IP 1 個 = 2 個の IP を使用。
- プレフィックスモード: ノード の IP 1 個 + 最初の /28 プレフィックスからの IP 16 個 (うち 1 つはポッドで使用、15 個は将来のポッド用に確保) = 17 個の IP を割り当て、2 個を使用。

例 B — ノードに 18 個のポッド

- セカンダリ IP モード: ポッド の IP 18 個+ ノードの IP 1 個 = 19 個の IP を使用。
- プレフィックスモード: ノードの IP 1 個 + 2 × /28 プレフィックス (合計 32 個の IP、最初の /28 は 16 ポッドに充当し、2 番目の /28 はポッド 17 ~ 18 を保持し、将来のポッド用に 14 個を確保) = 33 個の IP を割り当て、19 個を使用。

ポッドスパースワークロード向けにセカンダリ IP モード (ポッドごとに 1 IP、プレフィックス予約なし) に切り替えるには、advancedNetworking.ipv4PrefixSize: "32" を使用してカスタム NodeClass を作成し、カスタム NodePool でそれを参照します。組み込みの general-purpose および system ノードプールは、自動的にプロビジョニングされた default NodeClass を使用します。ipv4PrefixSize を上書きするには、カスタム NodePool とペアリングされたカスタム NodeClass が必要です。

ユースケース

- サブネット IP アドレスの使用率の向上

- プレフィックスの断片化なし

設定例

```
apiVersion: eks.amazonaws.com/v1
kind: NodeClass
metadata:
  name: advanced-networking
spec:
  role: MyNodeRole

  advancedNetworking:
    ipv4PrefixSize: "32"
```

セカンダリ IP モードに関する考慮事項

- ポッド作成速度の低下: 1 つのセカンダリ IP のみがウォームアップされるため、より多くのポッドが作成された場合、IPAM サービスによる IP のプロビジョニングにより多くの時間がかかります。

IPv6 クラスター内の IPv6 ポッドからの IPv4 egress を無効にする。

デフォルトでは、`enableV4Egress` フィールドは `true` です。Auto Mode IPv6 クラスターの場合、この機能を無効にすることで、Auto Mode が IPv6 ポッドに対して egress 専用 IPv4 インターフェイスを作成しないようにすることができます。IPv4 egress インターフェイスはネットワークポリシーの適用対象外であるため、これは重要です。ネットワークポリシーは、ポッドのプライマリインターフェイス (eth0) にのみ適用されます。

ユースケース

次の場合に `enableV4Egress` を使用します:

- IPv6 クラスターを使用する: IPv4 egress トラフィックはデフォルトで許可されています。
- ネットワークポリシーを使用する: 現在、EKS ネットワークポリシーはデュアルスタックをサポートしていません。`enableV4Egress` を無効にすると、ポッドトラフィックが IPv4 経由で予期せず出力されるのを防ぐことができます。

設定例

```
apiVersion: eks.amazonaws.com/v1
kind: NodeClass
metadata:
  name: advanced-networking
spec:
  role: MyNodeRole

  advancedNetworking:
    enableV4Egress: false
```

enableV4Egress を無効にする際の考慮事項

- IPv6 クラスターのネットワークポリシー: IPv6 クラスターでは、デフォルトで IPv4 トラフィックが許可されています。enableV4Egress: false に設定すると、IPv4 egress トラフィックがブロックされ、特にネットワークポリシーで使用する場合にセキュリティが強化されます。

静的ネットワークインターフェイスの設定

advancedNetworking の networkInterfaces フィールドを使用すると、起動時にインスタンスにアタッチされたネットワークインターフェイスを静的に定義できます。これは主に、分散トレーニングおよび推論ワークロードでの高性能なノード間通信用に Elastic Fabric Adapter (EFA) デバイスを設定するために使用されます。この機能は、静的キャパシティノードプールと組み合わせて、事前ウォーミングされた EFA 対応ノードを維持できます。EKS での EFA デバイス管理の詳細については、the section called “EFA デバイス” を参照してください。特定のインスタンスタイプの推奨される EFA 設定については、「Amazon EC2 ユーザーガイド」の「EFA 対応インスタンスタイプのネットワーク帯域幅を最大化する」を参照してください。

仕組み

networkInterfaces の各エントリは、起動時にインスタンスにアタッチされるネットワークインターフェイスを定義します。各エントリは、networkCardIndex (ネットワークカード、ここで 0 はプライマリ)、deviceIndex (そのカード上のデバイス位置)、および interfaceType (標準 IP ベースのトラフィックの場合は interface、または IP アドレスのない RDMA トラフィック専用の EFA インターフェイスの場合は efa-only) を指定します。IP ベースのノード通信をサポートするには、プライマリ ENI (networkCardIndex: 0、deviceIndex: 0) が interfaceType: interface を使用する必要があります。efa-only インターフェイスは RDMA の EFA デバイス機能のみをサポートし、IP アドレスで設定することはできません。

secondaryIPv4Count (各ユニットは 1 つの IP アドレスを提供する) または secondaryIPv4PrefixCount (各ユニットは 16 個の IP アドレスを持つ /28 プレフィックスを提供する) を使用して、プライマリネットワークカード (networkCardIndex: 0) のインターフェイスに IP キャパシティを割り当てることができます。インターフェイスごとに使用できるのはこれらのうちの 1 つだけで、networkCardIndex: 0 でのみサポートされています。efa-only インターフェイスでは使用できません。

> **Important**
>
> networkInterfaces が設定されている場合、EKS Auto Mode はインスタンスの起動後に追加の IP、プレフィックス、または ENI をアタッチしません。起動時に設定されたインターフェイスと IP アドレスのみが Pod で使用できます。設定された IP の数に基づいてポッド密度を計画する必要があります。

考慮事項

IPv6 は、静的に定義されたネットワークインターフェイスではサポートされていません。associatePublicIPAddress は、複数のネットワークインターフェイスが定義されている場合は互換性がないため、複数のインターフェイスを使用するノードにパブリック IP (パブリック専用クラスターまたはパブリック/プライベートクラスターの混在) を含めることはできません。

静的ネットワークインターフェイスで podSubnetSelectorTerms と podSecurityGroupSelectorTerms を使用する場合は、プライマリ ENI を 0 個のセカンダリ IP で設定します。プライマリ ENI はノードサブネットとセキュリティグループを使用し、Pod トラフィックはセカンダリ ENI でポッドサブネット設定を使用します。

例: GPU トレーニング用の EFA 専用インターフェイス

次の例は、分散トレーニングで使用される GPU インスタンスの EFA 専用インターフェイスで設定された NodeClass を示しています。プライマリインターフェイスには /28 プレフィックス (16 個のポッド IP) があり、4 個の追加の EFA 専用インターフェイスは RDMA 接続を提供します。この設定は、静的キャパシティノードプールと組み合わせて、事前ウォーミングされた EFA 対応ノードを維持できます。

```
apiVersion: eks.amazonaws.com/v1
kind: NodeClass
metadata:
  name: efa-training
spec:
```

```
  role: MyNodeRole
  subnetSelectorTerms:
    - tags:
        Name: "private-subnet"
  securityGroupSelectorTerms:
    - tags:
        Name: "efa-security-group"
  placementGroupSelector:
    name: "ml-training-pg"
  advancedNetworking:
    networkInterfaces:
    - deviceIndex: 0
      interfaceType: interface
      networkCardIndex: 0
      secondaryIPv4PrefixCount: 1
    - deviceIndex: 1
      interfaceType: efa-only
      networkCardIndex: 0
    - deviceIndex: 0
      interfaceType: efa-only
      networkCardIndex: 1
    - deviceIndex: 0
      interfaceType: efa-only
      networkCardIndex: 2
    - deviceIndex: 0
      interfaceType: efa-only
      networkCardIndex: 3
```

ノードで hugepages を設定する

`advancedCompute.hugepages` フィールドを使用すると、`NodeClass` のノードに hugepages を設定できます。hugepages は、ハイパフォーマンスコンピューティング (HPC)、データベース、ネットワーク負荷の高いアプリケーションなど、レイテンシーに敏感なワークロードのメモリアクセス性能を向上させます。2 つの独立した hugepages 設定を個別に、または組み合わせて使用できます。

- 静的な hugepages – `pages` フィールドを使用して、各ノードで固定数の `2Mi` または `1Gi` hugepages、あるいは両方のサイズの組み合わせを事前に割り当てます。静的な hugepages は、スケジューリング可能なノードリソースです。使用するには、`hugepages-2Mi` や `hugepages-1Gi` などの hugepage リソースを明示的にリクエストするようにポッドを設定します。詳細については、Kubernetes ドキュメントの「HugePages を管理する」を参照してください。

- Transparent hugepages (THP) – `transparent` フィールドを使用してカーネルの THP ポリシーを設定します。THP は、割り当て可能なリソースではなく、カーネル全体のポリシーです。スケジューリングやノード容量には影響せず、ポッドからリクエストすることもありません。詳細については、Linux カーネルドキュメントの「Transparent Hugepage Support」を参照してください。

設定例

```
apiVersion: eks.amazonaws.com/v1
kind: NodeClass
metadata:
  name: hugepages-compute
spec:
  role: MyNodeRole

  subnetSelectorTerms:
    - tags:
        Name: "private-subnet"

  securityGroupSelectorTerms:
    - tags:
        Name: "eks-cluster-sg"

  advancedCompute:
    hugepages:
      # Static hugepage reservations: up to one entry for each size
      pages:
        - size: "2Mi"
          count: 512
        - size: "1Gi"
          count: 2
      # Transparent hugepages (THP) kernel policy
      transparent:
        enabled: always  # or madvise, never
        defrag: defer    # or always, defer+madvise, madvise, never
```

Transparent hugepage の設定

どちらの `transparent` 設定もオプションです。設定を省略すると、ノードはカーネルのデフォルト (`madvise`) を使用します。

- **enabled** – enabled を使用して THP 割り当てポリシーを制御します。THP をシステム全体に適用するには、これを always に設定します。madvise(MADV_HUGEPAGE) でオプトインするメモリリージョンにのみ THP を適用するには、madvise に設定します。THP を無効にするには、never に設定します。
- **defrag** – defrag を使用して、THP 割り当てを満たすためにカーネルがメモリをどの程度積極的に圧縮するかを制御します。有効な値は、always、defer、defer +madvise、madvise、never です。

THP を無効にすることを推奨するデータベースワークロードなどで THP を無効にするには、enabled: never を設定します。enabled が never の場合、defrag を未設定のままにするか、never に設定します。NodeClass 検証は他の組み合わせを拒否します。

hugepages に関する考慮事項

NodeClass で hugepages を設定する際は、次の点を考慮してください。

- 静的な hugepage サイズ – pages フィールドを使用して、hugepage サイズ (2Mi および 1Gi) ごとに最大 1 つのエントリを指定します。同じノードで両方のサイズを予約できます。
- メモリ予約 – 静的 hugepage はインスタンスの総メモリに対して不利になります。すべてのページサイズにわたる hugepage の予約の合計が総メモリの 80% を超える場合、Amazon EKS はそのインスタンスタイプを NodeClass と互換性がないものとしてマークします。
- EFA の相互排他 – Elastic Fabric Adapter (EFA) と静的 hugepage は同じ NodeClass で共存できません。interfaceType: efa-only と pages を同時に設定すると、NodeClass の検証が失敗します。Amazon EKS は、静的 hugepage が設定された NodeClass から vpc.amazonaws.com/efa をリクエストするポッドをプロビジョニングしません。EFA ワークロードには別の NodeClass を使用してください。Transparent hugepages は EFA と競合しません。EFA インターフェイスを使用する NodeClass で transparent を設定できます。

ノードでの kubelet 設定の構成

advancedCompute.kubelet フィールドを使用して、NodeClass 内のノードの kubelet 設定を上書きします。これらの上書きを使用すると、ポッド密度、リソースプレッシャーによるエビクション、コンテナログのローテーション、out-of-memory (OOM) の動作を調整できます。Amazon EKS はデフォルトの kubelet 設定を Auto Mode ノードに適用するため、ワークロードで異なる動作が必要な場合にのみこのフィールドを設定してください。基盤となる設定の詳細については、Kubernetes ドキュメントの「Kubelet 設定」を参照してください。

以下の kubelet 設定を使用できます。

- `maxPods`: 1 つのノードで実行できるポッドの最大数。
- `podPidsLimit`: ポッドあたりのプロセス ID (PID) の最大数。
- `idsPerPod`: kubelet が各ポッドのユーザー名前空間に割り当てる UID と GID の数。
- `singleProcessOOMKill`: kubelet OOM がグループとしてではなく、コンテナ内のプロセスを個別に強制終了するかどうか。
- `eviction`: リソースプレッシャーシグナルのハードエビクションしきい値とソフトエビクションしきい値。
- `logging`: コンテナログのローテーションサイズとファイル数。
- `allowedUnsafeSysctls`: ノードでスケジュールされたポッドが設定できる、安全でない sysctl のリスト、または `*` で終わる名前空間パターンのリスト。

設定例

```
apiVersion: eks.amazonaws.com/v1
kind: NodeClass
metadata:
  name: kubelet-compute
spec:
  role: MyNodeRole

  subnetSelectorTerms:
    - tags:
        Name: "private-subnet"

  securityGroupSelectorTerms:
    - tags:
        Name: "eks-cluster-sg"

  advancedCompute:
    kubelet:
      # Range: 1-110
      maxPods: 60
      # -1 disables the limit, or specify 100 or greater
      podPidsLimit: 4096
      # Multiple of 65536, range: 65536-4294967295. Requires
      # advancedCompute.kernel.sysctl.user.max_user_namespaces > 0.
      idsPerPod: 65536
```

```
    singleProcessOOMKill: false
    eviction:
      # Valid signals: memory.available, nodefs.available, nodefs.inodesFree,
      # imagefs.available, imagefs.inodesFree, pid.available
      hard:
        memory.available: "100Mi"
        nodefs.available: "10%"
      soft:
        memory.available: "500Mi"
        nodefs.available: "15%"
      # Required when soft thresholds are set
      softGracePeriod:
        memory.available: "1m30s"
        nodefs.available: "2m"
      # 0 uses each Pod's terminationGracePeriodSeconds
      maxPodGracePeriod: 60
    logging:
      containerLogMaxSize: "10Mi"
      # Minimum: 2
      containerLogMaxFiles: 5
    # Up to 64 entries. Allowed patterns: kernel.shm*, kernel.msg*,
    # kernel.sem, fs.mqueue.*, net.*
    allowedUnsafeSysctls:
      - "net.core.somaxconn"
      - "kernel.shm*"
```

Kubelet 設定に関する考慮事項

NodeClass で kubelet 設定を行う際は、次の点を考慮してください。

- ノード置換: 既存の NodeClass で kubelet 設定を変更すると、それを使用するノードがドリフトとしてマークされます。Amazon EKS は、これらのノードを、更新された設定を使用する新しいノードに置き換えます。Amazon EKS は、実行中のノードに kubelet 設定をインプレースで適用しません。
- **maxPods** クランプ: 設定した値は上限であり、保証ではありません。実行時に、Amazon EKS は 3 つの値のうち最も低い値を適用します。それは、maxPods の値、インスタンスタイプで使用できる IP アドレス数、ノードあたり 110 ポッドの Auto Mode 制限です。詳細については、「the section called “Amazon EC2 インスタンスタイプ”」を参照してください。
- **podPidsLimit** 値: -1 を指定して制限を無効にするか、100 以上の値を指定します。NodeClass 検証は 0 ~ 99 の値を拒否します。

- **idsPerPod** 要件: 値は 65536 ~ 4294967295 で 65536 の倍数である必要があります。idsPerPod を設定するには、同じ NodeClass 上で advancedCompute.kernel.sysctl.user.max_user_namespaces を正の値に設定する必要があります。NodeClass アドミッションは、user.max_user_namespaces を 0 より大きい値に設定せずに idsPerPod を設定する構成を拒否します。user.max_user_namespaces の詳細については、「the section called “サポートされている sysctl”」を参照してください。
- ソフトエビクションしきい値: ソフトしきい値は、同じシグナルのハードしきい値よりも緩やかである必要があります。つまり、より早くトリガーされます。ソフトしきい値を設定する場合は、シグナルごとに softGracePeriod も設定する必要があります。ソフトしきい値とハードしきい値では、パーセンテージと絶対数量など、異なる単位を使用できます。この場合、Amazon EKS はインスタンスタイプごとに両方の値を解決します。その後、Amazon EKS は比較に違反するインスタンスタイプを NodeClass と互換性がないものとしてマークします。
- **singleProcessOOMKill** バージョン要件: この設定には Kubernetes バージョン 1.32 以降が必要です。以前のバージョンでは、Amazon EKS は NodeClass に警告イベントを発行し、設定は効果がありません。
- **allowedUnsafeSysctls** 制限: 最大 64 個のエントリを指定できます。各エントリは、kernel.shm*、kernel.msg*、kernel.sem、fs.mqueue.*、または net.* の許可されたパターンのいずれかと一致する必要があります。ノードで安全でない sysctl を許可しても、ポッドがそれをリクエストできるようになるだけです。各ポッドは引き続き sysctl を securityContext.sysctls にリストする必要があります。詳細については、Kubernetes ドキュメントの「Kubernetes クラスターで sysctl を使用する」を参照してください。

ノードでカーネル sysctl を設定する

advancedCompute.kernel.sysctl フィールドを使用して、NodeClass のノードに Linux カーネル sysctl を設定します。sysctl は、より高いネットワーク制限、異なるメモリ管理ポリシー、または特定の TCP 設定を必要とするワークロードのランタイムカーネル動作を調整します。sysctl フィールドのプロパティ名は、基盤となるカーネル sysctl 名と完全に一致します (例: vm.swappiness および net.core.somaxconn)。Amazon EKS はノードの起動時に sysctl 値を適用します。既存の NodeClass で sysctl を変更すると、Amazon EKS は新しい値を有効にするためにノードの置き換えを開始します。

設定例

```
apiVersion: eks.amazonaws.com/v1
kind: NodeClass
metadata:
```

```
  name: tuned-compute
spec:
  role: MyNodeRole

  subnetSelectorTerms:
    - tags:
        Name: "private-subnet"

  securityGroupSelectorTerms:
    - tags:
        Name: "eks-cluster-sg"

  advancedCompute:
    kernel:
      sysctl:
        vm.swappiness: 30
        net.core.somaxconn: 4096
        net.ipv4.tcp_congestion_control: "cubic"
        vm.dirty_background_ratio: 10
        vm.dirty_ratio: 20
```

サポートされている sysctl

Amazon EKS は、次の表の sysctl のみをサポートしています。各数値 sysctl は、示された最小値と最大値を適用し、NodeClass アドミッションはその範囲外の値を拒否します。文字列 sysctl は、リストされた値のみを受け入れます。

sysctl	説明	許可される値
fs.aio-max-nr	非同期 I/O リクエストの最大数。	65536–4194304
fs.inotify.max_user_instances	実際のユーザー ID あたりの inotify インスタンスの最大数。	8192–1048576
fs.inotify.max_user_watches	実際のユーザー ID あたりの inotify ウォッチの最大数。	8192–1048576
fs.nr_open	プロセスが割り当てることができるファイルハンドルの最大数。	1048576–2147483584

sysctl	説明	許可される値
kernel.perf_event_paranoid	パフォーマンスイベントシステムへの非特権アクセスを制御します。	-1–2
net.core.netdev_max_backlog	CPU あたりの INPUT 側でキューに入れられるパケットの最大数。	1000–3240000
net.core.rmem_max	受信ソケットバッファの最大サイズ (バイト単位)。	212992–134217728
net.core.somaxconn	listen syscall のバックログ引数の最大値。	4096–3240000
net.core.wmem_max	送信ソケットバッファの最大サイズ (バイト単位)。	212992–134217728
net.ipv4.neigh.default.gc_thresh1	ガベージコレクションを実行できるようになるまでの最小 ARP キャッシュエントリ数。	0–262144
net.ipv4.neigh.default.gc_thresh2	ARP キャッシュエントリのソフト最大数。	512–524288
net.ipv4.neigh.default.gc_thresh3	ARP キャッシュエントリのハード最大数。	1024–1048576
net.ipv4.tcp_congestion_control	新しい接続に対する TCP 輻輳制御アルゴリズム。	cubic, reno
net.ipv4.tcp_fin_timeout	孤立した接続がローカルエンドで中止されるまで FIN_WAIT_2 状態に留まる時間 (秒単位)。	5–120
net.ipv4.tcp_keepalive_time	キープアライブが有効になっている場合に、TCP がキープアライブメッセージを送信する頻度 (秒単位)。	30–432000
net.ipv4.tcp_max_syn_backlog	許可される未処理の SYN リクエストの最大数。この値を超えるリクエストはカーネルによってドロップされます。	128–3240000

sysctl	説明	許可される値
net.ipv4.tcp_tw_reuse	プロトコルの観点から安全であれば、新しい接続に対して TIME-WAIT ソケットの再利用を有効にします。0 は無効、1 はグローバルに有効、2 はループバックのみ有効にします。	0–2
user.max_user_namespaces	ユーザーが作成できるユーザー名前空間の最大数。0 の値を指定すると、ユーザー名前空間が無効になります。	0–2147483647
vm.dirty_background_ratio	カーネルフラッシャースレッドがダーティページのディスクへの書き戻しを開始する、使用可能なメモリの割合。	1–20
vm.dirty_expire_centisecs	ダーティページが書き戻しの対象になるまでの 100 分の 1 秒単位の時間。	0–6000
vm.dirty_ratio	書き込みプログラムが書き込み時に強制的にブロックされる、使用可能なメモリの割合。	5–40
vm.dirty_writeback_centisecs	フラッシャースレッド呼び出し間の 100 分の 1 秒単位の時間。	0–1000
vm.max_map_count	1 つのプロセスが保持できるメモリマップ領域の最大数。	65530–2147483647
vm.swappiness	スワップとファイルシステムのページングの相対的な IO コスト。	0–200
vm.watermark_scale_factor	メモリ再利用のウォーターマーク比率 (10,000 分の 1 単位)。	10–3000

sysctl に関する考慮事項

NodeClass でカーネル sysctl を設定する際は、次の点を考慮してください。

- サポートされている sysctl – the section called “サポートされている sysctl” にリストされている sysctl のみがサポートされています。各フィールドはドキュメント化された最小値と最大値を適用し、NodeClass アドミッションはその範囲外の値を拒否します。
- 起動時に適用 – sysctl はノード起動時に Bottlerocket の `[settings.kernel.sysctl]` に書き込まれます。`spec.advancedCompute.kernel.sysctl` へのランタイムの変更により、既存のノードはドリフトとしてマークされます。新しい値は置き換えられたノードでのみ有効になります。
- **`vm.dirty_background_ratio`** と **`vm.dirty_ratio`** – この 2 つのフィールドは両方とも設定するか、両方とも省略する必要があり、`vm.dirty_background_ratio` は `vm.dirty_ratio` より必ず小さくなければなりません。NodeClass アドミッションは、いずれかのルールに違反する設定を拒否します。
- **`net.ipv4.neigh.default.gc_thresh1`**、**`gc_thresh2`**、および **`gc_thresh3`** – 3 つの ARP キャッシュしきい値はすべて一緒に設定するか、すべて省略する必要があり、その値は `gc_thresh1 < gc_thresh2 < gc_thresh3` を満たす必要があります。NodeClass アドミッションは、いずれかのルールに違反する設定を拒否します。
- **`net.ipv4.tcp_congestion_control`** – `cubic` と `reno` のみが受け入れられます。

EKS 自動モードI 用のノードプールを作成する

Amazon EKS ノードプールにより、Kubernetes クラスター内のコンピューティングリソースを柔軟に管理できます。このトピックでは、クラスターのスケーリングとリソース使用率を最適化するノードプロビジョニングツールである Karpenter を使用してノードプールを作成および設定する方法を示します。Karpenter の ノードプール リソースを使用すると、インスタンスタイプ、アベイラビリティーゾーン、アーキテクチャ、キャパシティータイプなど、コンピューティングリソースの特定の要件を定義できます。

組み込みの `system` および `general-purpose` ノードプールは変更できません。これらは、有効または無効にすることのみができます。詳細については、「the section called “組み込みノードプールを確認する”」を参照してください。

ノードプール の指定により、サポートされているさまざまなラベルと要件を通じて、EKS クラスターのコンピューティングリソースをきめ細かく制御できます。これには EC2 インスタンスカテゴリ、CPU 設定、アベイラビリティーゾーン、アーキテクチャ (ARM64/AMD64)、キャパシティタイプ (スポットまたはオンデマンド) を指定するためのオプションが含まれます。CPU とメモリの使用量のリソース制限を設定して、クラスターが必要な運用境界内に収まるようにすることもできます。

EKS 自動モードI は広く知られている Kubernetes ラベルを活用して、ノードの特性を識別する一貫性のある標準化された方法を提供します。アベイラビリティーゾーンの `topology.kubernetes.io/zone` や CPU アーキテクチャの `kubernetes.io/arch` といったこのラベルは確立された Kubernetes 規則に従います。さらに、EKS 固有のラベル (プレフィックスが `eks.amazonaws.com/`) により、インスタンスタイプ、CPU メーカー、GPU 機能、ネットワーク仕様などの AWS 固有の属性でこの機能が拡張されます。この標準化されたラベル付けシステムにより、既存の Kubernetes ツールとシームレスに統合できると同時に、緊密な AWS インフラストラクチャ統合が提供されます。

ノードプール を作成する

Amazon EKS クラスターの ノードプール を作成するには次のステップに従います：

1. 必要な NodePool 設定で `nodepool.yaml` という名前の YAML ファイルを作成します。以下のサンプル設定を使用できます。
2. ノードプール をクラスターに適用します：

```
kubectl apply -f nodepool.yaml
```

3. ノードプール が正常に作成されたことを確認します：

```
kubectl get nodepools
```

4. (任意 ノードプール のステータスをモニタリングします：

```
kubectl describe nodepool default
```

ノードプール がクラスターに存在する有効な ノードクラス を参照していることを確認します。ノードクラス はコンピューティングリソースの AWS 固有の設定を定義します。詳細については「the section called “ノードクラスの作成”」を参照してください。

サンプル ノードプール

```
apiVersion: karpenter.sh/v1
kind: NodePool
metadata:
  name: my-node-pool
spec:
  template:
```

```
    metadata:
      labels:
        billing-team: my-team
    spec:
      nodeClassRef:
        group: eks.amazonaws.com
        kind: NodeClass
        name: default

      requirements:
        - key: "eks.amazonaws.com/instance-category"
          operator: In
          values: ["c", "m", "r"]
        - key: "eks.amazonaws.com/instance-cpu"
          operator: In
          values: ["4", "8", "16", "32"]
        - key: "topology.kubernetes.io/zone"
          operator: In
          values: ["us-west-2a", "us-west-2b"]
        - key: "kubernetes.io/arch"
          operator: In
          values: ["arm64", "amd64"]

  disruption:
    consolidationPolicy: Balanced
    consolidateAfter: 30s

  limits:
    cpu: "1000"
    memory: 1000Gi
```

EKS Auto Mode でサポートされているラベル

EKS 自動モードI は以下の広く知られてるラベルをサポートしています。

> **Note**
>
> EKS 自動モードI は Karpenter とは異なるラベルを使用します。EC2 マネージドインスタンスに関連するラベルは`eks.amazonaws.com` で始まります。

ラベル	例	説明
topology.kubernetes.io/zone	us-east-2a	AWS リージョン
node.kubernetes.io/インスタンス型	g4dn.8xlarge	AWS インスタンスタイプ
kubernetes.io/arch	amd64	アーキテクチャはインスタンスの GOARCH 値によって定義されます
karpenter.sh/capacity-type	spot	キャパシティタイプには spot、on-demand 、reserved などがあります。キャパシティタイプを指定しない場合、Karpenter は reserved を優先し、次に spot 、次に on-demand を優先します。
eks.amazonaws.com/instance-hypervisor	nitro	特定のハイパーバイザーを使用するインスタンスタイプ
eks.amazonaws.com/compute-type	自動	EKS 自動モードI マネージドノードを識別します
eks.amazonaws.com/instance-encryption-in-transit-supported	true	転送中の暗号化をサポートする (またはサポートしない) インスタンスタイプ
eks.amazonaws.com/instance-category	g	同じカテゴリのインスタンスタイプ、通常は生成番号の前の文字列
eks.amazonaws.com/instance-generation	4	インスタンスカテゴリ内のインスタンスタイプ生成番号
eks.amazonaws.com/instance-family	g4dn	プロパティが類似していてリソース量が異なるインスタンスタイプ
eks.amazonaws.com/instance-size	8xlarge	リソース量が類似していてプロパティが異なるインスタンスタイプ
eks.amazonaws.com/instance-cpu	32	インスタンスにおける CPU の数

ラベル	例	説明
eks.amazonaws.com/instance-cpu-manufacturer	aws	CPU メーカーの名前
eks.amazonaws.com/instance-memory	131072	インスタンスのメモリのメビバイト数
eks.amazonaws.com/instance-ebs-bandwidth	9500	インスタンスで使用できる EBS の最大メガビット数
eks.amazonaws.com/instance-network-bandwidth	131072	インスタンスで使用可能なベースラインメガビット数
eks.amazonaws.com/instance-gpu-name	t4	インスタンス上の GPU の名前 (使用可能な場合)
eks.amazonaws.com/instance-gpu-manufacturer	nvidia	GPU メーカーの名前
eks.amazonaws.com/instance-gpu-count	1	インスタンスにおける GPU の数
eks.amazonaws.com/instance-gpu-memory	16384	GPU のメモリのメビバイト数
eks.amazonaws.com/instance-local-nvme	900	インスタンス上のローカル nvme ストレージのギビバイト数
eks.amazonaws.com/capacity-reservation-interruptible	true	キャパシティ予約が中断可能かどうか。`karpenter.sh/capacity-type: reserved` を持つノード上でのみ存在します。

> **Note**
>
> EKS Auto Mode は特定のインスタンスのみをサポートし、最小サイズ要件があります。詳細については、「the section called “EKS Auto Mode でサポートされているインスタンスリファレンス”」を参照してください。

EKS Auto Mode でサポートされていないラベル

EKS 自動モードI は次のラベルをサポートしていません。

- EKS 自動モードは Linux のみをサポートしています
  - `node.kubernetes.io/windows-build`
  - `kubernetes.io/os`

組み込みノードプールを無効にする

カスタムノードプールを作成する場合は、組み込みノードプールを無効にすることができます。詳細については、「the section called “組み込みノードプールを確認する”」を参照してください。

組み込みノードプールがないクラスター

組み込みノードプールがないクラスターを作成できます。これは、組織がカスタマイズしたノードプールを作成している場合に役立ちます。

> **Note**
>
> 組み込みノードプールなしでクラスターを作成すると、`default` NodeClass は自動的にプロビジョニングされません。カスタム NodeClass を作成する必要があります。詳細については、「the section called “ノードクラスの作成”」を参照してください。

概要：

1. `nodePools` と `nodeRoleArn` の両方の値を空にして、EKS クラスターを作成します。
   - eksctl `autoModeConfig` の例:

     ```
     autoModeConfig:
       enabled: true
       nodePools: []
       # Do not set a nodeRoleARN
     ```

     詳細については、the section called “eksctl CLI”を参照してください。
2. ノードロール ARN を使用してカスタムノードクラスを作成します
   - 詳細については、the section called “ノードクラスの作成”を参照してください。

3. カスタムノードクラスのアクセスエントリを作成します
   - 詳細については、the section called “ノードクラスのアクセスエントリを作成する”を参照してください。
4. 前述のように、カスタムノードプールを作成します。

中断

NodePool を介してノードを中断するように、EKS Auto Mode を複数の方法で設定できます。`spec.disruption.consolidationPolicy`、`spec.disruption.consolidateAfter`、または `spec.template.spec.expireAfter` を使用できます。NodePool の `spec.disruption.budgets` を使用して、EKS Auto Mode の中断をレート制限することもできます。中断される時間枠と同時ノード数を制御することもできます。この動作を設定する手順については、Karpenter ドキュメントの「Disruption」を参照してください。

`consolidationPolicy` フィールドは 3 個の値を受け入れます:

- `WhenEmpty` – 空のノードのみを統合するため、実行中のワークロードが中断されることはありません。
- `WhenEmptyOrUnderutilized` – コストに合わせて積極的に統合し、ノードのポッドをより安価に再パックできるたびにワークロードを中断します。
- `Balanced` – 中断コストとコスト削減を比較することで、各統合アクションを採点します。中断が削減を上回るアクションをスキップします。

各値の詳細については、Karpenter ドキュメントの「中断」を参照してください。

ノードプールの中断は、次の目的で設定できます。

- インスタンスの使用率が低いタイミングを特定し、ワークロードを統合する。
- ノードプールの中断予算を作成して、ドリフト、空、統合が理由でのノードの終了をレート制限する。

デフォルトでは、EKS Auto Mode では以下が行われます。

- 使用率が低いインスタンスを統合します。
- 336 時間後にインスタンスを終了します。
- ノードの 10% の単一の中断予算を設定します。

- 新しい Auto Mode AMI がリリースされたときに、ドリフトを理由にノードを置き換えることを許可します。これは、およそ週に 1 回行われます。

終了猶予期間

EKS Auto Mode NodePool で `terminationGracePeriod` を明示的に定義していない場合、関連付けた NodeClaim に 24 時間 (デフォルト) の終了猶予期間が自動的に適用されます。EKS Auto Mode のカスタム NodePool 設定では `terminationGracePeriod` がデフォルトであることを確認できませんが、NodeClaim には、このデフォルト値が表示されます。猶予期間を NodePool で明示的に設定済みか、NodeClaim でデフォルト設定を行ったかに関係なく、この機能には一貫性があり、予測可能なノード終了動作がクラスター全体で維持されます。

EKS Auto Mode での静的キャパシティノードプール

Amazon EKS Auto Mode は、静的キャパシティノードプールをサポートしており、ポッドの需要に関係なく決まった数のノードを維持します。静的キャパシティノードプールは、予測可能なキャパシティ、リザーブドインスタンス、または一貫したインフラストラクチャフットプリントの維持が求められる特定のコンプライアンス要件を必要とするワークロードに有用です。

ポッドスケジューリングの需要に基づいてスケールする動的ノードプールとは異なり、静的キャパシティノードプールでは設定したノードの数が維持されます。

静的キャパシティノードプールを設定する

静的キャパシティノードプールを作成するには、NodePool 仕様に `replicas` フィールドを設定します。`replicas` フィールドでは、ノードプールが維持するノードの正確な数を定義します。`replicas` の設定方法については、「the section called “例”」を参照してください。

静的キャパシティノードプールに関する考慮事項

静的キャパシティノードプールには、重要な制約と動作がいくつかあります。

設定に関する制約:

- モードを切り替えることができない: ノードプールで `replicas` を設定したら、以後削除することはできません。ノードプールでは、静的モードと動的モードとを切り替えることができません。
- リソースが制限されている: 制限セクションでは、`limits.nodes` フィールドのみがサポートされています。CPU とメモリに関する制限は適用されません。
- 重みフィールドがない: ノードの選択が優先度に基づいていないため、静的キャパシティノードプールに `weight` フィールドを設定することができません。

オペレーション動作:

- 統合対象外: 静的キャパシティプールのノードは、統合対象と見なされません。
- スケールオペレーション: スケールオペレーションでは、ノード中断の予算はバイパスされますが、PodDisruptionBudgets は引き続き考慮されます。
- ノードの置き換え: 設定に基づいてドリフト (AMI アップデートなど) と有効期限が発生した場合は、ノードが置き換えられます。

ベストプラクティス

容量プランニング:

- `limits.nodes` を `replicas` よりも高い値に設定すると、ノードの置き換えオペレーション中に一時的にスケールできるようになります。
- 制限を設定するときは、ノードドリフトや AMI アップデートに必要な最大キャパシティを考慮してください。

インスタンス選択:

- リザーブドインスタンスがある場合や特定のハードウェア要件がある場合は、特定のインスタンスタイプを使用します。
- 制限を過度に厳しくしてスケール中にインスタンスの可用性が制限されるような要件は避けてください。

中断の管理:

- 可用性とメンテナンスオペレーションとのバランスが取れるように、適切な中断の予算を設定します。
- 予算の割合を設定するときは、アプリケーションでノードの置き換えをどの程度許容するかを考慮してください。

モニタリング:

- `status.nodes` フィールドを定期的にモニタリングして、必要なキャパシティが維持されていることを確認します。
- 実際のノード数が目的のレプリカから逸脱した場合に備えてアラートを設定します。

ゾーンの分散:

- 高可用性を実現するには、静的キャパシティをいくつかのアベイラビリティーゾーンに分散させます。
- 複数のアベイラビリティーゾーンにまたがる静的キャパシティノードプールを作成すると、EKS Auto Mode により、指定されたゾーンにノードが分散しますが、その分散が均等であるとは限りません。
- アベイラビリティーゾーン間に想定の範囲内で均等に分散させるには、静的キャパシティノードプールを個別にいくつか作成し、`topology.kubernetes.io/zone` 要件を使用してそれぞれを特定のアベイラビリティーゾーンに固定します。
- 3 つのゾーンに 12 個のノードを均等に分散させる必要がある場合は、1 つのノードプールを作成して 3 つのゾーンに 12 個のレプリカを分散させるのではなく、3 つのノードプールを作成してそれぞれに 4 個のレプリカを分散させます。

静的キャパシティノードプールをスケールする

`kubectl scale` コマンドを使用すると、静的キャパシティノードプール内のレプリカの数を変更できます。

```
# Scale down to 5 nodes
kubectl scale nodepool static-nodepool --replicas=5
```

スケールダウンすると、EKS 自動モードはノードを適切に終了し、PodDisruptionBudgets を考慮のうえ、実行中のポッドを残りのノードに再スケジュールできるようにします。

静的キャパシティノードプールをモニタリングする

以下のコマンドを使用すると、静的キャパシティノードプールをモニタリングできます。

```
# View node pool status
kubectl get nodepool static-nodepool

# Get detailed information including current node count
kubectl describe nodepool static-nodepool

# Check the current number of nodes
kubectl get nodepool static-nodepool -o jsonpath='{.status.nodes}'
```

status.nodes フィールドは、ノードプールで管理されているノードの現在の数を示し、通常の条件下で必要になる replicas の数と一致する必要があります。

トラブルシューティング

ノードが目的のレプリカに到達しない:

- limits.nodes の値が十分かどうかを確認します。
- 要件がインスタンス選択を過度に制限していないことを確認します。
- 使用しているインスタンスタイプとリージョンの AWS サービスクォータを確認します。

ノードの置き換えに時間がかかりすぎる:

- 同時により多くの置き換えができるように中断予算を調整します。
- PodDisruptionBudgets がノードの終了を妨げていないかどうかを確認します。

ノードが予期せずに終了する:

- expireAfter と terminationGracePeriod の設定を確認します。
- 手動によるノードの終了や AWS メンテナンスイベントを確認します。

例

基本的な静的キャパシティノードプール

```
apiVersion: karpenter.sh/v1
kind: NodePool
metadata:
  name: basic-static
spec:
  replicas: 5

  template:
    spec:
      nodeClassRef:
        group: eks.amazonaws.com
        kind: NodeClass
        name: default
```

```
      requirements:
        - key: "eks.amazonaws.com/instance-category"
          operator: In
          values: ["m"]
        - key: "topology.kubernetes.io/zone"
          operator: In
          values: ["us-west-2a"]

  limits:
    nodes: 8  # Allow scaling up to 8 during operations
```

特定のインスタンスタイプの静的キャパシティ

```
apiVersion: karpenter.sh/v1
kind: NodePool
metadata:
  name: reserved-instances
spec:
  replicas: 20

  template:
    metadata:
      labels:
        instance-type: reserved
        cost-center: production
    spec:
      nodeClassRef:
        group: eks.amazonaws.com
        kind: NodeClass
        name: default

      requirements:
        - key: "node.kubernetes.io/instance-type"
          operator: In
          values: ["m5.2xlarge"]  # Specific instance type
        - key: "karpenter.sh/capacity-type"
          operator: In
          values: ["on-demand"]
        - key: "topology.kubernetes.io/zone"
          operator: In
          values: ["us-west-2a", "us-west-2b", "us-west-2c"]

  limits:
```

```
    nodes: 25

  disruption:
    # Conservative disruption for production workloads
    budgets:
      - nodes: 10%
```

マルチゾーンの静的キャパシティノードプール

```
apiVersion: karpenter.sh/v1
kind: NodePool
metadata:
  name: multi-zone-static
spec:
  replicas: 12  # Will be distributed across specified zones

  template:
    metadata:
      labels:
        availability: high
    spec:
      nodeClassRef:
        group: eks.amazonaws.com
        kind: NodeClass
        name: default

      requirements:
        - key: "eks.amazonaws.com/instance-category"
          operator: In
          values: ["c", "m"]
        - key: "eks.amazonaws.com/instance-cpu"
          operator: In
          values: ["8", "16"]
        - key: "topology.kubernetes.io/zone"
          operator: In
          values: ["us-west-2a", "us-west-2b", "us-west-2c"]
        - key: "karpenter.sh/capacity-type"
          operator: In
          values: ["on-demand"]

  limits:
    nodes: 15
```

```
  disruption:
    budgets:
      - nodes: 25%
```

キャパシティ予約による静的キャパシティ

以下の例は、EC2 キャパシティ予約で静的キャパシティノードプールを使用する方法を示しています。EKS Auto Mode での EC2 キャパシティ予約の使用の詳細については、「the section called “キャパシティ予約を制御する”」を参照してください。

`capacityReservationSelectorTerms` を定義する NodeClass

```
apiVersion: eks.amazonaws.com/v1
kind: NodeClass
metadata:
  name: capacity-reservation-nodeclass
spec:
  role: AmazonEKSNodeRole
  securityGroupSelectorTerms:
  - id: sg-0123456789abcdef0
  subnetSelectorTerms:
  - id: subnet-0123456789abcdef0
  capacityReservationSelectorTerms:
  - id: cr-0123456789abcdef0
```

上記の NodeClass を参照し、`karpenter.sh/capacity-type: reserved` を使用する NodePool

```
apiVersion: karpenter.sh/v1
kind: NodePool
metadata:
  name: static-capacity-reservation-nodepool
spec:
  replicas: 5
  limits:
    nodes: 8  # Allow scaling up to 8 during operations
  template:
    metadata: {}
    spec:
      nodeClassRef:
        group: eks.amazonaws.com
        kind: NodeClass
```

```
        name: capacity-reservation-nodeclass
      requirements:
      - key: karpenter.sh/capacity-type
        operator: In
        values: ['reserved']
```

組み込み NodePool を有効または無効にする

EKS Auto Mode には 2 つの組み込み NodePool があります。これらの NodePool を有効または無効にするには、AWS コンソール、CLI、または API を使用します。

組み込み NodePool リファレンス

- `system`
  - この NodePool には `CriticalAddonsOnly` テイントがあります。CoreDNS など多くの EKS アドオンで、このテイントが許容されています。このシステムノードプールを使用して、クラスターに不可欠なアプリケーションを分離します。
  - `amd64` と `arm64` アーキテクチャの両方をサポートします。
- `general-purpose`
  - この NodePool は、クラスター内の汎用ワークロード用のノードの起動をサポートします。
  - `amd64` アーキテクチャのみを使用します。

両方の組み込み NodePool:

- デフォルトの EKS NodeClass を使用する
- オンデマンド EC2 容量のみを使用する
- C、M、および R EC2 インスタンスファミリーを使用する
- 世代 5 以降の EC2 インスタンスが必要になる

> **Note**
>
> EKS が「デフォルト」の NodeClass をプロビジョニングするには、少なくとも 1 つの組み込み NodePool を有効にする必要があります。すべての組み込み NodePool を無効にする場合は、カスタム NodeClass を作成し、それを使用するように NodePool を設定する必要があります。NodeClass の詳細については、the section called “ノードクラスの作成” を参照してください。

> **Important**
>
> `computeConfig.nodePools` から組み込み NodePool 名を削除すると、対応する NodePool Kubernetes リソースがクラスターから削除されます。その NodePool によって管理されるノードはすべてドレインされ、終了します。組み込み NodePool 名を `computeConfig.nodePools` に戻すと、対応する NodePool リソースがクラスターに再作成されます。Karpenter は Kubernetes API を介して NodePool リソースを検出し、削除された NodePools のノードのスケジュールを停止するため、この動作はキャパシティの変更を計画するときに重要です。

手順

前提条件

- AWS コマンドラインインターフェイス (AWS CLI) の最新バージョンがデバイスにインストールおよび設定されていること。現在のバージョンを確認するには、`aws --version` を参照してください。最新バージョンをインストールするには、「AWS コマンドラインインターフェイスユーザーガイド」の「インストール」および「aws configure を使用したクイック設定」を参照してください。
  - 十分な IAM アクセス許可を使用して CLI にログインし、IAM ポリシー、IAM ロール、EKS クラスターなどの AWS リソースを作成します。

AWS CLI で有効にする

次のコマンドを使用して、両方の組み込み NodePool を有効にします。

```
aws eks update-cluster-config \
  --name <cluster-name> \
  --compute-config '{
    "nodeRoleArn": "<node-role-arn>",
    "nodePools": ["general-purpose", "system"],
    "enabled": true
  }' \
  --kubernetes-network-config '{
  "elasticLoadBalancing":{"enabled": true}
  }' \
  --storage-config '{
  "blockStorage":{"enabled": true}
```

```
  }'
```

コマンドを変更して、NodePool を選択的に有効にすることができます。

AWS CLI で無効にする

次のコマンドを使用して、両方の組み込み NodePool を無効にします。

```
aws eks update-cluster-config \
  --name <cluster-name> \
  --compute-config '{
  "enabled": true,
  "nodePools": []
  }' \
  --kubernetes-network-config '{
  "elasticLoadBalancing":{"enabled": true}}' \
  --storage-config '{
  "blockStorage":{"enabled": true}
  }'
```

EKS 自動モードでキャパシティ予約へのワークロードのデプロイを制御する

キャパシティ予約へのワークロードのデプロイを制御できます。EKS 自動モードは、EC2 オンデマンドキャパシティ予約 (ODCR)、EC2 Capacity Blocks for ML、および中断可能なキャパシティ予約 (IODCR) をサポートしています。

> **Tip**
>
> デフォルトでは、EKS Auto Mode はオープンマッチングを通じてオープン ODCR に起動できますが、優先順位は付けられません。オープンマッチングによって起動されたインスタンスには、`reserved` ではなく `karpenter.sh/capacity-type: on-demand` というラベルが付けられます。ODCR の使用を優先し、インスタンスに `karpenter.sh/capacity-type: reserved` というラベルを付けるには、NodeClass 定義で `capacityReservationSelectorTerms` を設定します。Capacity Blocks for ML には、常に `capacityReservationSelectorTerms` が必要であり、自動的には使用されません。

EC2 オンデマンドキャパシティ予約 (ODCR)

EC2 オンデマンドキャパシティ予約 (ODCR) を使用すると、任意の期間中に特定のアベイラビリティーゾーンで Amazon EC2 インスタンスのコンピューティングキャパシティを予約できます。EKS Auto Mode を使用するとき、Kubernetes ワークロードをこれらの予約されたインスタンスにデプロイするかどうかを制御し、事前購入された容量の使用率を最大化したり、重要なワークロードが保証されたリソースにアクセスできるようにしたりすることができます。

デフォルトでは、EKS Auto Mode は自動的にオープン ODCR として起動します。ただし、NodeClass で `capacityReservationSelectorTerms` を設定することで、ワークロードが使用する ODCR を明示的に制御できます。設定された ODCR を使用してプロビジョニングされたノードには `karpenter.sh/capacity-type: reserved` があり、オンデマンドおよびスポットよりも優先されます。この機能を有効にすると、EKS Auto Mode はオープン ODCR を自動的に使用しなくなります。NodeClass で明示的に選択する必要があり、クラスター全体のキャパシティ予約の使用状況を正確に制御できます。

> **⚠ Warning**
>
> クラスター内の NodeClass で `capacityReservationSelectorTerms` を設定した場合、EKS Auto Mode はクラスター内のすべての NodeClass に対してオープン ODCR を自動的に使用しなくなります。

NodeClass の例

```
apiVersion: eks.amazonaws.com/v1
kind: NodeClass
spec:
  # Optional: Selects upon on-demand capacity reservations and capacity blocks
  # for EKS Auto Mode to prioritize.
  capacityReservationSelectorTerms:
    - id: cr-56fac701cc1951b03
    # Alternative Approaches
    - tags:
        app: "my-app"
      # Optional owning account ID filter
      owner: "012345678901"
```

この NodeClass の例では、ODCR を選択するための 2 つのアプローチが示されています。1 番目の方法では、特定の ODCR が ID (`cr-56fac701cc1951b03`) で直接参照されます。2 番目の方法では

タグベースの選択を使用し、タグ `Name: "targeted-odcr"` で ODCR がターゲットされます。オプションとして、予約を所有する AWS アカウントでフィルタリングすることもできます。アカウント間シナリオや共有キャパシティ予約を使用する場合に特に便利です。

キャパシティ予約のスケジューリングラベル

NodeClass で `capacityReservationSelectorTerms` を設定すると、キャパシティ予約からプロビジョニングされたノードには追加のスケジューリングラベルが付けられます。これらのラベルを NodePool 要件またはポッドスケジューリング制約 (ノードアフィニティなど) として使用して、ワークロードの配置を制御できます。

ラベル	例	説明
`eks.amazonaws.com/capacity-reservation-id`	`cr-56fac701cc1951b03`	キャパシティ予約の ID
`eks.amazonaws.com/capacity-reservation-type`	`default`、または `capacity-block`	キャパシティ予約のタイプ
`eks.amazonaws.com/capacity-reservation-interruptible`	`true`、または `false`	キャパシティ予約が中断可能かどうか

これらのラベルは、`karpenter.sh/capacity-type: reserved` を持つノードにのみ存在します。

中断可能なキャパシティ予約 (IODCR)

中断可能なキャパシティ予約を使用すると、未使用のオンデマンドキャパシティ予約キャパシティを組織内の他のワークロードと共有できます。未使用のキャパシティを中断可能な ODCR として再利用することで、バッチ処理やデータ分析など、柔軟で耐障害性のあるオペレーションに適したワーク

ロードは、一時的に利用可能なキャパシティのメリットを得ることができます。予約所有者はいつでもキャパシティを再利用できますが、中断可能な ODCR のコンシューマーは、ノードが終了する前に正常なシャットダウンまたはチェックポイントを可能にするために、終了前に中断通知を受け取ります。

中断可能なキャパシティ予約を、ID またはタグで、標準の ODCR と同じ方法で、`capacityReservationSelectorTerms` を使用して選択します。ワークロードが中断可能なリザーブドキャパシティと中断不可能なリザーブドキャパシティのどちらでスケジュールされるかを制御するには、`eks.amazonaws.com/capacity-reservation-interruptible` スケジューリングラベルを使用します。

キャパシティが再利用されると、実行中のインスタンスは EventBridge イベントを通じて 2 分間の中断警告を受け取ります。通知期間後、再利用されたキャパシティで実行中のインスタンスはシャットダウン状態になり、終了します。EKS Auto Mode は、2 分間の中断警告を受け取ると、中断可能なキャパシティ予約から起動されたノードのドレインを自動的に開始し、ノードが再利用される前にワークロードを適切に終了できるようにします。

EC2 Capacity Blocks for ML

Capacity Blocks for ML は、短期間の機械学習ワークロードをサポートするため、GPU ベースの高速コンピューティングインスタンスを将来の日付で予約します。キャパシティブロック内で実行されるインスタンスは、Amazon EC2 UltraClusters 内に自動的に互いに近く配置され、低レイテンシーでペタビットスケールのノンブロッキングネットワーキングを実現します。

サポートされているプラットフォームとインスタンスタイプの詳細については、「EC2 ユーザーガイド」の「Capacity Blocks for ML」を参照してください。

前述の ODCR と同じく、EKS 自動モードの NodeClass を作成し、その中で Capacity Block for ML を使用できます。

次のサンプル定義では、3 つのリソースを作成しています。

1. キャパシティブロック予約を参照する NodeClass
2. NodeClass を使用し、テイントを適用する NodePool
3. テイントを許容し、GPU リソースをリクエストするポッド仕様

NodeClass の例

この NodeClass は、特定の Capacity Block for ML をその予約 ID で参照します。この ID は、EC2 コンソールから取得できます。

```
apiVersion: eks.amazonaws.com/v1
kind: NodeClass
metadata:
  name: gpu
spec:
  # Specify your Capacity Block reservation ID
  capacityReservationSelectorTerms:
    - id: cr-56fac701cc1951b03
```

詳細については、「the section called “ノードクラスの作成”」を参照してください。

NodePool の例

この NodePool は、gpu NodeClass を参照し、以下の重要な設定を指定します。

- karpenter.sh/capacity-type: reserved を設定してリザーブドキャパシティのみを使用します。
- ML ワークロードに適した特定の GPU インスタンスファミリーをリクエストします。
- nvidia.com/gpu テイントを適用して、これらのノードで GPU ワークロードのみがスケジュールされるようにします。

```
apiVersion: karpenter.sh/v1
kind: NodePool
metadata:
  name: gpu
spec:
  template:
    spec:
      nodeClassRef:
        group: eks.amazonaws.com
        kind: NodeClass
        name: gpu
      requirements:
        - key: eks.amazonaws.com/instance-family
          operator: In
          values:
            - g6
            - p4d
            - p4de
            - p5
            - p5e
```

```
            - p5en
            - p6
            - p6-b200
        - key: karpenter.sh/capacity-type
          operator: In
          values:
            - reserved
            # Enable other capacity types
            # - on-demand
            # - spot
      taints:
        - effect: NoSchedule
          key: nvidia.com/gpu
```

詳細については、「the section called “ノードプールの作成”」を参照してください。

ポッドの例

この例に示したポッドは、キャパシティブロックノードで動作するようにワークロードを設定する方法を示しています。

- nodeSelector を使用して、特定の GPU タイプ (この例では H200 GPU) をターゲットとしています。
- NodePool で適用される nvidia.com/gpu テイントの許容値が含まれています。
- nvidia.com/gpu リソースタイプを使用して明示的に GPU リソースをリクエストします。

```
apiVersion: v1
kind: Pod
metadata:
  name: nvidia-smi
spec:
  nodeSelector:
    # Select specific GPU type - uncomment as needed
    # eks.amazonaws.com/instance-gpu-name: l4
    # eks.amazonaws.com/instance-gpu-name: a100
    eks.amazonaws.com/instance-gpu-name: h200
    # eks.amazonaws.com/instance-gpu-name: b200
    eks.amazonaws.com/compute-type: auto
  restartPolicy: OnFailure
  containers:
  - name: nvidia-smi
```

```
    image: public.ecr.aws/amazonlinux/amazonlinux:2023-minimal
    args:
    - "nvidia-smi"
    resources:
      requests:
        # Uncomment if needed
        # memory: "30Gi"
        # cpu: "3500m"
        nvidia.com/gpu: 1
      limits:
        # Uncomment if needed
        # memory: "30Gi"
        nvidia.com/gpu: 1
  tolerations:
  - key: nvidia.com/gpu
    effect: NoSchedule
    operator: Exists
```

詳細については、Kubernetes ドキュメントの「ポッド」を参照してください。

関連リソース

- 「Amazon EC2 ユーザーガイド」の「Capacity Blocks for ML」
- 「Amazon EC2 ユーザーガイド」の「キャパシティブロックの検索と購入」
- 「Amazon EC2 ユーザーガイド」の「中断可能なキャパシティ予約」
- Amazon EKS で AI/ML ワークロードのコンピューティングリソースを管理する
- 「EKS ベストプラクティスガイド」の「GPU リソースの最適化とコスト管理」

Local Zone に EKS 自動モードノードをデプロイする

EKS 自動モードでは、自動ノードプロビジョニングでクラスター管理をシンプルにできます。AWSLocal Zone は、エンドユーザーに地理的に近い場所まで AWS インフラストラクチャを拡張して、レイテンシーの影響を受けやすいアプリケーションのレイテンシーを軽減します。このガイドでは、EKS 自動モードノードを AWS Local Zone にデプロイするプロセスを見ていきます。ガイドに従うと、特定の地理的エリアのユーザーを対象に、コンテナ化されたアプリケーションを低レイテンシーで実行できます。

また、Kubernetes テイントおよび許容を使用して、特定のワークロードのみが Local Zone ノードで実行されるようにする方法も示します。これにより、コストを抑え、リソースの使用を最適化できます。

前提条件

EKS 自動モードノードを Local Zone にデプロイする前に、次の前提条件が満たされていることを確認してください。

- EKS 自動モードのクラスターが既に存在している
- AWS アカウントで Local Zone にオプトインされている

ステップ 1: Local Zone サブネットを作成する

EKS 自動モードノードを Local Zone にデプロイする最初のステップは、その Local Zone にサブネットを作成することです。このサブネットは、ノードのネットワークインフラストラクチャとなるもので、VPC の他の要素と通信できるようになります。「Local Zone サブネットを作成する」の手順 (「AWS Local Zone ユーザーガイド」) に従って、選択した Local Zone にサブネットを作成します。

> **Tip**
>
> Local Zone サブネットの名前をメモします。

ステップ 2: Local Zone サブネット用に NodeClass を作成する

Local Zone サブネットを作成したら、このサブネットを参照する NodeClass を定義する必要があります。NodeClass は、ノードのインフラストラクチャ属性を指定する Kubernetes カスタムリソースで、例えば使用するサブネット、セキュリティグループ、ストレージ設定を指定します。以下の例では、Local Zone サブネットをその名前に基づいてターゲットとする NodeClass を「local-zone」という名前で作成しています。また、サブネット ID を使用することもできます。Local Zone サブネットをターゲットとするように、この設定を調整する必要があります。

詳細については、「the section called “ノードクラスの作成”」を参照してください。

```
apiVersion: eks.amazonaws.com/v1
kind: NodeClass
metadata:
  name: local-zone
spec:
  subnetSelectorTerms:
    - id: <local-subnet-id>
```

ステップ 3: NodeClass とテイントで NodePool を作成する

NodeClass を設定したら、この NodeClass を使用する NodePool を作成する必要があります。NodePool では、インスタンスタイプなど、ノードのコンピューティング特性を定義します。NodePool は、インスタンスの起動場所を決定する際に NodeClass をリファレンスとして使用します。

以下の例では、「local-zone」NodeClass を参照する NodePool を作成しています。また、ノードにテイントを追加して、これらの Local Zone ノードでは許容範囲が一致するポッドのみをスケジュールできるようにしています。これは、Local Zone ノードにとって特に重要です。このノードは、通常コストが高く、レイテンシーを短縮すると特にメリットが得られるワークロードでのみ使用する必要があるからです。

詳細については、「the section called “ノードプールの作成”」を参照してください。

```
apiVersion: karpenter.sh/v1
kind: NodePool
metadata:
  name: my-node-pool
spec:
  template:
    metadata:
      labels:
        node-type: local-zone
    spec:
      nodeClassRef:
        group: eks.amazonaws.com
        kind: NodeClass
        name: local-zone
      taints:
        - key: "aws.amazon.com/local-zone"
          value: "true"
          effect: NoSchedule

      requirements:
        - key: "eks.amazonaws.com/instance-category"
          operator: In
          values: ["c", "m", "r"]
        - key: "eks.amazonaws.com/instance-cpu"
          operator: In
          values: ["4", "8", "16", "32"]
```

key が aws.amazon.com/local-zone で effect が NoSchedule のテイントにより、許容範囲が一致しないポッドがこれらのノードでスケジュールされることはありません。これにより、通常のワークロードが誤って Local Zone で実行されて、想定外のコストが発生するのを防ぐことができます。

ステップ 4: 許容範囲とノードアフィニティでワークロードをデプロイする

Local Zone ノードへのワークロードの配置を最適に制御するには、テイント/許容範囲とノードアフィニティの両方を一緒に使用します。このように組み合わせたアプローチには以下のメリットがあります。

1. コストの抑制: テイントにより、許容範囲が明示的に指定されたポッドのみが、コストが高くつく可能性がある Local Zone リソースを使用できるようになります。
2. 保証された配置: ノードアフィニティにより、レイテンシーの影響を受けやすいアプリケーションは、通常のクラスターノードではなく、Local Zone でのみ実行されます。

以下に、Local Zone ノードでのみ実行されるように設定された Deployment の例を示します。

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: low-latency-app
  namespace: default
spec:
  replicas: 2
  selector:
    matchLabels:
      app: low-latency-app
  template:
    metadata:
      labels:
        app: low-latency-app
    spec:
      tolerations:
      - key: "aws.amazon.com/local-zone"
        operator: "Equal"
        value: "true"
        effect: "NoSchedule"
      affinity:
        nodeAffinity:
          requiredDuringSchedulingIgnoredDuringExecution:
            nodeSelectorTerms:
```

```
            - matchExpressions:
              - key: "node-type"
                operator: "In"
                values: ["local-zone"]
      containers:
      - name: application
        image: my-low-latency-app:latest
        resources:
          limits:
            cpu: "1"
            memory: "1Gi"
          requests:
            cpu: "500m"
            memory: "512Mi"
```

この Deployment には、重要なスケジューリング設定が 2 つあります。

1. 許容範囲により、テイントが aws.amazon.com/local-zone のノードにポッドをスケジュールできます。
2. ノードアフィニティ要件により、こうしたポッドはラベルが node-type: local-zone のノードでのみ実行されます。

これにより、レイテンシーの影響を受けやすいアプリケーションは Local Zone ノードでのみ実行され、通常のアプリケーションは明示的に設定されていない限り Local Zone リソースを消費しません。

ステップ 5: AWS コンソールで検証する

NodeClass、NodePool、Deployment をセットアップしたら、ノードが想定どおりに Local Zone にプロビジョニングされていることと、ワークロードがノードで実行されていることを確認する必要があります。AWS マネジメントコンソールを使用すると、EC2 インスタンスが適切な Local Zone サブネットで起動されていることを確認できます。

また、kubectl get nodes -o wide を使用すると、Kubernetes ノードリストをチェックして、ノードが適切なラベルとテイントでクラスターに参加していることを確認できます。

```
kubectl get nodes -o wide
kubectl describe node <node-name> | grep -A 5 Taints
```

さらに、ワークロードポッドが Local Zone ノードにスケジュールされていることを確認することもできます。

```
kubectl get pods -o wide
```

このアプローチにより、指定された範囲で Local Zone テイントを許容するワークロードのみがこれらのノードでスケジュールされるため、コストを抑え、Local Zone リソースを最も効率的に使用できます。

EKS Auto Mode の組織コントロールを更新する

一部の組織コントロールは、EKS Auto Mode の正常な機能を妨げる場合があります。その場合、これらのコントロールを更新し、EKS Auto Mode がユーザーに代わって EC2 インスタンスを管理するために必要なアクセス許可を持てるようにする必要があります。

EKS Auto Mode には、EKS Auto Mode ノードをバックアップする EC2 インスタンスを起動するためのサービスロールが使用されます。サービスロールはユーザーの名義で作成された IAM ロールであり、サービスがユーザーに代わってアクションを実行するために引き受けます。サービスコントロールポリシー (SCP) はサービスロールで実行されるアクションに常に適用されます。SCP が Auto Mode のオペレーションを禁止できるようにします。最も一般的なケースは、SCP を使用して起動できる Amazon マシンイメージ (AMI) を制限する場合です。EKS Auto Mode を機能させるには、SCP を変更して EKS Auto Mode アカウントから AMI の起動を許可します。

EC2 が許可された AMI 機能を使用し、他のアカウントで AMI の可視性を制限することもできます。この機能を使用する場合、対象リージョンに EKS Auto Mode AMI アカウントも含めるため、イメージ条件を拡張する必要があります。

EKS Auto Mode AMI を除くすべての AMI をブロックする SCP の例

以下の SCP では、AMI が us-west-2 または us-east-1 の EKS Auto Mode AMI アカウントに属していない限り、`ec2:RunInstances` の呼び出しが防止されます。

> **Note**
>
> `ec2:Owner` コンテキストキーを使用しないことが重要です。Amazon は EKS Auto Mode AMI アカウントを所有しており、このキーの値は常に `amazon` になります。`ec2:Owner` が `amazon` の場合に AMI の起動を許可する SCP を構築すると、EKS Auto Mode 用の AMI だけでなく、Amazon が所有する AMI の起動も許可されます。

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Sid": "DenyAMI",
      "Effect": "Deny",
      "Action": "ec2:RunInstances",
      "Resource": "arn:*:ec2:*::image/ami-*",
      "Condition": {
        "StringNotEquals": {
          "aws:ResourceAccount": [
            "767397842682",
            "992382739861"
          ]
        }
      }
    }
  ]
}
```

EKS Auto Mode AMI アカウント

AWSリージョンによって異なる アカウントは、EKS Auto Mode パブリック AMI をホストします。

AWS リージョン	アカウント
af-south-1	471112993317
ap-east-1	590183728416
ap-east-2	381492200852
ap-northeast-1	851725346105
ap-northeast-2	992382805010
ap-northeast-3	891377407544
ap-south-1	975049899075
ap-south-2	590183737426

ap-southeast-1	339712723301
ap-southeast-2	058264376476
ap-southeast-3	471112941769
ap-southeast-4	590183863144
ap-southeast-5	654654202513
ap-southeast-6	905418310314
ap-southeast-7	533267217478
ca-central-1	992382439851
ca-west-1	767397959864
eu-central-1	891376953411
eu-central-2	381492036002
eu-north-1	339712696471
eu-south-1	975049955519
eu-south-2	471112620929
eu-west-1	381492008532
eu-west-2	590184142468
eu-west-3	891376969258
il-central-1	590183797093
me-central-1	637423494195
me-south-1	905418070398
mx-central-1	211125506622

sa-east-1	339712709251
us-east-1	992382739861
us-east-2	975050179949
us-west-1	975050035094
us-west-2	767397842682
us-gov-east-1	446077414359
us-gov-west-1	446098668741

パブリック IP アドレスを関連付ける

`ec2:RunInstances` が呼び出されると、インスタンス起動の `AssociatePublicIpAddress` フィールドは、インスタンスが起動されるサブネットのタイプによって自動的に決定されます。SCP を使用すると、起動するサブネットのタイプに関係なく、この値を明示的に false に設定することができます。この場合、SCP の要件を満たすために NodeClass フィールド `spec.advancedNetworking.associatePublicIPAddress` を false に設定することもできます。

```
{
      "Sid": "DenyPublicEC2IPAddresses",
      "Effect": "Deny",
      "Action": "ec2:RunInstances",
      "Resource": "arn:aws:ec2:*:*:network-interface/*",
      "Condition": {
          "BoolIfExists": {
              "ec2:AssociatePublicIpAddress": "true"
          }
      }
  }
```

ネットワークとロードバランシングの設定

Amazon EKS Auto Mode クラスターのネットワークとロードバランシングの動作を設定します。以下のトピックを参照して、Application Load Balancer、Network Load Balancer、ターゲットグループ、ネットワークポリシー、およびサブネットのタグ付けを設定してください。

トピック

- IngressClass を作成して Application Load Balancer を設定する
- サービス注釈を使用して Network Load Balancer を設定する
- マルチクラスターターゲットグループを設定する
- 2 つのクラスター間でターゲットグループを共有する
- EKS Auto Mode でネットワークポリシーを使用する
- EKS Auto Mode のサブネットにタグを付ける

IngressClass を作成して Application Load Balancer を設定する

EKS Auto Mode は、インターネットへのクラスターアプリケーションの公開など、ロードバランシングのルーチンタスクを自動化します。

AWS では、Application Load Balancer (ALB) を使用して HTTP および HTTPS トラフィックを処理することを推奨しています。Application Load Balancer は、リクエストの内容に基づいてリクエストをルーティングできます。Application Load Balancer の詳細については、「What is Elastic Load Balancing?」を参照してください。

EKS Auto Mode は、Application Load Balancer (ALB) の作成と設定を行います。例えば、`Ingress` Kubernetes オブジェクトを作成すると EKS Auto Mode によってロードバランサーが作成され、クラスターワークロードにトラフィックをルーティングするように設定されます。

概要:

1. インターネットに公開するワークロードを作成します。
2. SSL/TLS や VPC サブネットに使用する証明書などの AWS 固有の設定値を指定して、`IngressClassParams` リソースを作成します。
3. `IngressClass` リソースを作成し、EKS Auto Mode がリソースのコントローラーになるように指定します。
4. HTTP パスおよびポートをクラスターワークロードに関連付ける `Ingress` リソースを作成します。

EKS Auto Mode は、`IngressClassParams` リソースで指定されたロードバランサー設定を使用して、`Ingress` リソースで指定されたワークロードをポイントする Application Load Balancer を作成します。

前提条件

- EKS Auto Mode が Amazon EKS クラスターで有効になっている
- kubectl がクラスターに接続するように設定されている
  - `kubectl apply -f <filename>` を使用して、以下のサンプル設定 YAML ファイルをクラスターに適用できます。

> **Note**
>
> EKS Auto Mode では、パブリックサブネットとプライベートサブネットを識別するためにサブネットタグが必要です。
> `eksctl` を使用してクラスターを作成した場合は、これらのタグは既にあります。
> the section called “サブネットにタグを付ける” では、その方法を説明しています。

ステップ 1: ワークロードを作成する

まず、インターネットに公開するワークロードを作成します。これは、デプロイやサービスなど、HTTP トラフィックを処理する任意の Kubernetes リソースです。

この例では、`service-2048` というシンプルな HTTP サービスを使用して、ポート `80` をリッスンします。次の `2048-deployment-service.yaml` というマニフェストを適用して、このサービスとそのデプロイを作成します。

```
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: deployment-2048
spec:
  selector:
    matchLabels:
      app.kubernetes.io/name: app-2048
  replicas: 2
  template:
    metadata:
      labels:
        app.kubernetes.io/name: app-2048
    spec:
      containers:
```

```
        - image: public.ecr.aws/l6m2t8p7/docker-2048:latest
          imagePullPolicy: Always
          name: app-2048
          ports:
            - containerPort: 80
---
apiVersion: v1
kind: Service
metadata:
  name: service-2048
spec:
  ports:
    - port: 80
      targetPort: 80
      protocol: TCP
  type: NodePort
  selector:
    app.kubernetes.io/name: app-2048
```

設定をクラスターに適用します。

```
kubectl apply -f 2048-deployment-service.yaml
```

前記のリソースは、デフォルトの名前空間に作成されます。次のコマンドを実行して、これを検証できます。

```
kubectl get all -n default
```

ステップ 2: IngressClassParams を作成する

IngressClassParams オブジェクトを作成して、Application Load Balancer に対する AWS 固有の設定オプションを指定します。この例では、alb という名前の IngressClassParams リソース (次のステップで使用する) を作成し、ロードバランサースキームを internet-facing として alb-ingressclassparams.yaml というファイルで指定します。

```
apiVersion: eks.amazonaws.com/v1
kind: IngressClassParams
metadata:
  name: alb
spec:
  scheme: internet-facing
```

設定をクラスターに適用します。

```
kubectl apply -f alb-ingressclassparams.yaml
```

ステップ 3: IngressClass を作成する

alb-ingressclass.yaml という名前のファイル内に、IngressClassParams リソースで設定された AWS 固有の設定値を参照する IngressClass を作成します。IngressClass の名前をメモしてください。この例では、IngressClass と IngressClassParams の両方が alb という名前になっています。

is-default-class 注釈を使用して、Ingress リソースがこのクラスをデフォルトで使用するかどうかを制御します。

```
apiVersion: networking.k8s.io/v1
kind: IngressClass
metadata:
  name: alb
  annotations:
    # Use this annotation to set an IngressClass as Default
    # If an Ingress doesn't specify a class, it will use the Default
    ingressclass.kubernetes.io/is-default-class: "true"
spec:
  # Configures the IngressClass to use EKS Auto Mode
  controller: eks.amazonaws.com/alb
  parameters:
    apiGroup: eks.amazonaws.com
    kind: IngressClassParams
    # Use the name of the IngressClassParams set in the previous step
    name: alb
```

設定オプションの詳細については、「the section called “IngressClassParams リファレンス”」を参照してください。

設定をクラスターに適用します。

```
kubectl apply -f alb-ingressclass.yaml
```

ステップ 4: Ingress を作成する

alb-ingress.yaml という名前のファイルに Ingress リソースを作成します。このリソースの目的は、Application Load Balancer のパスとポートをクラスター内のワークロードに関連付けることで

す。この例では、ポート 80 で service-2048 という名前のサービスにトラフィックをルーティングする 2048-ingress という名前の Ingress リソースを作成します。

このリソースの構成方法の詳細については、Kubernetes ドキュメントの「Ingress」を参照してください。

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: 2048-ingress
spec:
  # this matches the name of IngressClass.
  # this can be omitted if you have a default ingressClass in cluster: the one with
 ingressclass.kubernetes.io/is-default-class: "true"  annotation
  ingressClassName: alb
  rules:
    - http:
        paths:
          - path: /*
            pathType: ImplementationSpecific
            backend:
              service:
                name: service-2048
                port:
                  number: 80
```

設定をクラスターに適用します。

```
kubectl apply -f alb-ingress.yaml
```

ステップ 5: ステータスを確認する

kubectl を使用して、Ingress のステータスを取得します。ロードバランサーが利用できるようになるまで数分かかる場合があります。

前のステップで設定した Ingress リソースの名前を使用します。例えば、次のようになります。

```
kubectl get ingress 2048-ingress
```

リソースの準備ができたら、ロードバランサーのドメイン名を取得します。

```
kubectl get ingress 2048-ingress -o
 jsonpath='{.status.loadBalancer.ingress[0].hostname}'
```

ウェブブラウザでサービスを表示するには、Ingress リソースで指定したポートとパスを確認します。

ステップ 6: クリーンアップ

ロードバランサーをクリーンアップするには、次のコマンドを使用します。

```
kubectl delete ingress 2048-ingress
kubectl delete ingressclass alb
kubectl delete ingressclassparams alb
```

EKS Auto Mode が、AWS アカウント内の関連するロードバランサーを自動的に削除します。

IngressClassParams リファレンス

IngressClassParams リソースで設定できるフィールドを次の表に示します。設定は、IngressClass を使用するすべての Ingress に適用されます。

フィールド	説明	値の例
scheme	ALB が内部向けかインターネット向けかを定義します	internet-facing
loadBalancerName	ALB の名前を設定します。名前は最大 32 文字です。alb.ingress.kubernetes.io/load-balancer-name 注釈を上書きします。	my-alb
namespaceSelector	この IngressClass を使用できる名前空間を制限します	environment: prod
group.name	1 つの ALB を共有する複数の Ingress をグループ化します	retail-apps

フィールド	説明	値の例
ipAddressType	ALB の IP アドレスタイプを設定します。有効な値は、ipv4、dualstack 、dualstack-without-public-ipv4 です。	dualstack
subnets.ids	ALB デプロイにおけるサブネット ID のリスト	subnet-xxxx, subnet-yyyy
subnets.matchTags	ALB のサブネットを選択するタグフィルター。各フィルターには、key と values のリストが含まれています。	key: Environment, values: [prod]
certificateARNs	使用する SSL 証明書の ARN	arn:aws:acm:region:account:certificate/id
sslPolicy	HTTPS リスナーのセキュリティポリシーを設定します。alb.ingress.kubernetes.io/ssl-policy 注釈を上書きします。	ELBSecurityPolicy-TLS13-1-2-2021-06
inboundCIDRs	ALB へのアクセスが許可されている CIDR ブロック。alb.ingress.kubernetes.io/inbound-cidrs 注釈を上書きします。	10.0.0.0/16, 192.168.0.0/24

フィールド	説明	値の例
prefixListsIDs	ALB へのアクセスが許可されているマネージドプレフィックスリストの ID。alb.ingress.kubernetes.io/security-group-prefix-lists 注釈を上書きします。	pl-00000000, pl-11111111
targetType	ターゲットグループのターゲットタイプを設定します。有効な値は、instance および ip です。デフォルト: ip。	instance
tags	AWS リソースのカスタムタグ	Environment: prod, Team: platform
loadBalancerAttributes	ロードバランサー固有の属性	idle_timeout.timeout_seconds: 60
listeners	Ingress で定義されたリスナーに対して、リスナー属性を設定します。各エントリは、port と protocol (HTTP または HTTPS) に基づいてリスナーと照合され、attributes を key と value のペアとして一覧表示します。このフィールドではリスナーは作成されません。同じ属性キーの alb.ingress.kubernetes.io/listener-attributes.${Protocol}-${Port} 注釈を上書きします。	port: 443, protocol: HTTPS, attributes: routing.http.response.server.enabled: "false"

フィールド	説明	値の例
minimumLoadBalancerCapacity.capacityUnits	ALB の最小容量を、ロードバランサーキャパシティーユニット (LCU) 単位で予約します。予約を削除する場合は、0 に設定します。alb.ingress.kubernetes.io/minimum-load-balancer-capacity 注釈を上書きします。	1000
ipamConfiguration.ipv4IPAMPoolId	インターネット向け ALB がパブリック IPv4 アドレスに使用する Amazon VPC IP Address Manager (IPAM) プールの ID。alb.ingress.kubernetes.io/ipam-ipv4-pool-id 注釈を上書きします。	ipam-pool-0123456789abcdef0

考慮事項

- EKS Auto Mode でロードバランサーを設定するのに、IngressClass で注釈を使用することはできません。IngressClass の設定は、IngressClassParams を介して行う必要があります。ただし、個々の Ingress リソースで注釈を使用すると、ロードバランサーの動作を設定できます (alb.ingress.kubernetes.io/security-group-prefix-lists や alb.ingress.kubernetes.io/conditions.* など)。
- IngressClass 内のすべての Ingress に ListenerAttribute 値を設定するには、IngressClassParams の listeners フィールドを使用します。詳細については、「the section called “IngressClassParams リファレンス”」を参照してください。
- クラスター IAM ロールを更新して、Kubernetes から AWS ロードバランサーリソースへのタグの伝播を有効にする必要があります。詳細については、「the section called “EKS Auto リソースのカスタム AWS タグ”」を参照してください。
- リソースを EKS Auto Mode またはセルフマネージド AWS Load Balancer Controller に関連付ける方法については、「the section called “移行のリファレンス”」を参照してください。

- ロードバランサーの問題の修正方法については、「the section called “トラブルシューティング”」を参照してください。
- EKS Auto Mode のロードバランシング機能の使用に関する考慮事項については、「the section called “負荷分散”」を参照してください。

次の表は、EKS Auto Mode の IngressClassParams、Ingress 注釈、および TargetGroupBinding 設定の変更の詳細な比較を示しています。これらの表は、API バージョンの変更、廃止された機能、更新されたパラメータ名など、EKS Auto Mode のロードバランシング機能とオープンソースのロードバランサーコントローラーの主な違いを強調しています。

IngressClassParams

旧	新	説明
`elbv2.k8s.aws/v1beta1`	`eks.amazonaws.com/v1`	API バージョンの変更
`spec.certificateArn`	`spec.certificateARNs`	複数の証明書 ARN のサポート
`spec.subnets.tags`	`spec.subnets.matchTags`	サブネット一致スキーマの変更
`spec.listeners.listenerAttributes`	`spec.listeners.attributes`	リスナー属性のフィールド名の変更
`spec.PrefixListsIDs`	`spec.prefixListsIDs`	`prefixListsIDs` という表記のみがサポートされています
`spec.sslRedirectPort`	サポートされていません	Ingress オブジェクトで `alb.ingress.kubernetes.io/ssl-redirect` 注釈を使用
`spec.wafv2AclArn`	サポートされていません	Ingress オブジェクトで `alb.ingress.kubernetes.io/wafv2-acl-arn` 注釈を使用

旧	新	説明
spec.wafv2AclName	サポートされていません	Ingress オブジェクトで alb.ingress.kubernetes.io/wafv2-acl-name 注釈を使用

Ingress 注釈

旧	新	説明
kubernetes.io/ingress.class	サポートされていません	Ingress オブジェクトで spec.ingressClassName を使用
alb.ingress.kubernetes.io/group.name	サポートされていません	IngressClass でのみグループを指定
alb.ingress.kubernetes.io/waf-acl-id	サポートされていません	代わりに WAF v2 を使用
alb.ingress.kubernetes.io/web-acl-id	サポートされていません	代わりに WAF v2 を使用
alb.ingress.kubernetes.io/dry-run-plan	サポートされていません	ドライランプランは現在サポートされていません
alb.ingress.kubernetes.io/create-acm-cert	サポートされていません	ACM 証明書の作成は現在サポートされていません
alb.ingress.kubernetes.io/acm-pca-arn	サポートされていません	ACM PCA ARN は現在サポートされていません
alb.ingress.kubernetes.io/auth-type: oidc	追加の RBAC でサポートされています	ロードバランサーコントローラーに OIDC Secret への get アクセスを付与しま

旧	新	説明
		す。詳細については、「the section called “LBC にシークレットへのアクセスを許可する”」を参照してください。

TargetGroupBinding

旧	新	説明
`elbv2.k8s.aws/v1beta1`	`eks.amazonaws.com/v1`	API バージョンの変更
`spec.targetType` が任意	`spec.targetType` が必須	明示的なターゲットタイプの指定
`spec.networking.ingress.from`	サポートされていません	セキュリティグループなしでは NLB がサポートされなくなりました

カスタム TargetGroupBinding 機能を使用するには、ターゲットグループにクラスター名と共に eks:eks-cluster-name タグを付けて、必要な IAM アクセス許可をコントローラーに付与する必要があります。TargetGroupBinding リソースまたはクラスターを削除すると、コントローラーがターゲットグループを削除することに注意してください。

サービス注釈を使用して Network Load Balancer を設定する

Kubernetes サービス注釈を使用して Amazon EKS で Network Load Balancer (NLB) を設定する方法について説明します。このトピックでは、インターネットアクセシビリティ、ヘルスチェック、SSL/TLS ターミネーション、IP ターゲティングモードなど、NLB の動作をカスタマイズするための EKS Auto Mode でサポートされている注釈について説明します。

EKS Auto Mode でタイプ `LoadBalancer` の Kubernetes サービスを作成すると、EKS は指定した注釈に基づいて AWS Network Load Balancer を自動的にプロビジョニングして設定します。この宣言型アプローチにより、Kubernetes マニフェストを通じてロードバランサーの設定を直接管理し、Infrastructure as Code プラクティスを維持できます。

EKS Auto Mode は、デフォルトでタイプ LoadBalancer のサービスすべてに対する Network Load Balancer プロビジョニングを処理します。追加のコントローラーのインストールや設定は必要ありません。`loadBalancerClass: eks.amazonaws.com/nlb` の仕様は、クラスターのデフォルトとして自動的に設定され、既存の Kubernetes ワークロードとの互換性を維持しながらデプロイプロセスが合理化されます。

> **Note**
>
> EKS Auto Mode では、パブリックサブネットとプライベートサブネットを識別するためにサブネットタグが必要です。
> `eksctl` を使用してクラスターを作成した場合は、これらのタグは既にあります。
> the section called “サブネットにタグを付ける” では、その方法を説明しています。

サンプルサービス

Kubernetes `Service` リソースの詳細については、「Kubernetes ドキュメント」を参照してください。

以下のサンプル `Service` リソースを確認してください。

```
apiVersion: v1
kind: Service
metadata:
  name: echoserver
  annotations:
    # Specify the load balancer scheme as internet-facing to create a public-facing
 Network Load Balancer (NLB)
    service.beta.kubernetes.io/aws-load-balancer-scheme: internet-facing
spec:
  selector:
    app: echoserver
  ports:
    - port: 80
      targetPort: 8080
      protocol: TCP
  type: LoadBalancer
  # Specify the new load balancer class for NLB as part of EKS Auto Mode feature
  # For clusters with Auto Mode enabled, this field can be omitted as it's the default
  loadBalancerClass: eks.amazonaws.com/nlb
```

一般的に使用される注釈

次の表に、Amazon EKS Auto Mode でサポートされているアノテーションを示します。Amazon EKS Auto Mode は、AWS Load Balancer Controller で利用可能なすべてのサービスアノテーションをサポートしているわけではありません。

> **Tip**
>
> 以下の注釈はすべて、`service.beta.kubernetes.io/` というプレフィックスを付ける必要があります

フィールド	説明	例
`aws-load-balancer-type`	ロードバランサーのタイプを指定します。新しいデプロイには `external` を使用します。	`external`
`aws-load-balancer-nlb-target-type`	トラフィックをノードインスタンスにルーティングするか、ポッド IP に直接ルーティングするかを指定します。標準デプロイでは `instance` を、直接ポッドルーティングでは `ip` を使用します。	`instance`
`aws-load-balancer-scheme`	ロードバランサーが内部向けかインターネット向けかを指定します。	`internet-facing`
`aws-load-balancer-healthcheck-protocol`	ターゲットグループのヘルスチェックプロトコル。一般的なオプションは TCP (デフォルト) または HTTP です。	`HTTP`

フィールド	説明	例
aws-load-balancer-healthcheck-path	HTTP/HTTPS プロトコルを使用する場合のヘルスチェックの HTTP パス。	/healthz
aws-load-balancer-healthcheck-port	ヘルスチェックに使用されるポート。特定のポート番号または traffic-port を指定できます。	traffic-port
aws-load-balancer-subnets	ロードバランサーを作成するサブネットを指定します。サブネット ID またはサブネット名を使用できます。	subnet-xxxx, subnet-yyyy
aws-load-balancer-ssl-cert	HTTPS/TLS 用の AWS Certificate Manager からの SSL 証明書の ARN。	arn:aws:acm:region:account:certificate/cert-id
aws-load-balancer-ssl-ports	SSL/TLS を使用するポートを指定します。	443, 8443
load-balancer-source-ranges	ロードバランサーへのアクセスが許可されている CIDR 範囲。	10.0.0.0/24, 192.168.1.0/24
aws-load-balancer-additional-resource-tags	ロードバランサーおよび関連リソースに適用する追加の AWS タグ。	Environment=prod,Team=platform
aws-load-balancer-ip-address-type	ロードバランサーが使用するのが IPv4 かデュアルスタック (IPv4 + IPv6) かを指定します。	ipv4-または-dualstack

フィールド	説明	例
`aws-load-balancer-listener-attributes.${Protocol}-${Port}`	リスナーに適用するリスナー属性を指定します。	`tcp.idle_timeout.seconds=350`

考慮事項

- クラスター IAM ロールを更新して、Kubernetes から AWS ロードバランサーリソースへのタグの伝播を有効にする必要があります。詳細については、「the section called “EKS Auto リソースのカスタム AWS タグ”」を参照してください。
- リソースを EKS Auto Mode またはセルフマネージド AWS Load Balancer Controller に関連付ける方法については、「the section called “移行のリファレンス”」を参照してください。
- ロードバランサーの問題の修正方法については、「the section called “トラブルシューティング”」を参照してください。
- EKS Auto Mode のロードバランシング機能の使用に関する考慮事項については、「the section called “負荷分散”」を参照してください。

ロードバランシングのために EKS Auto Mode に移行する場合は、サービス注釈とリソース設定にいくつかの変更が必要です。次の表は、サポートされていないオプションや推奨される代替案など、新旧の実装の主な違いをまとめたものです。

サービス注釈

旧	新	説明
`service.beta.kubernetes.io/load-balancer-source-ranges`	サポートされていません	Service で `spec.loadBalancerSourceRanges` を使用
`service.beta.kubernetes.io/aws-load-balancer-type`	サポートされていません	Service で `spec.loadBalancerClass` を使用

旧	新	説明
service.beta.kubernetes.io/aws-load-balancer-internal	サポートされていません	service.beta.kubernetes.io/aws-load-balancer-scheme を使用する
service.beta.kubernetes.io/aws-load-balancer-proxy-protocol	サポートされていません	代わりに service.beta.kubernetes.io/aws-load-balancer-target-group-attributes を使用
各種ロードバランサー属性	サポートされていません	service.beta.kubernetes.io/aws-load-balancer-attributes を使用する
service.beta.kubernetes.io/aws-load-balancer-access-log-enabled	サポートされていません	代わりに service.beta.kubernetes.io/aws-load-balancer-attributes を使用
service.beta.kubernetes.io/aws-load-balancer-access-log-s3-bucket-name	サポートされていません	代わりに service.beta.kubernetes.io/aws-load-balancer-attributes を使用
service.beta.kubernetes.io/aws-load-balancer-access-log-s3-bucket-prefix	サポートされていません	代わりに service.beta.kubernetes.io/aws-load-balancer-attributes を使用
service.beta.kubernetes.io/aws-load-balancer-cross-zone-load-balancing-enabled	サポートされていません	代わりに service.beta.kubernetes.io/aws-load-balancer-attributes を使用

非推奨のロードバランサー属性注釈から移行するには、これらの設定を `service.beta.kubernetes.io/aws-load-balancer-attributes` 注釈に統合します。この注釈は、各種ロードバランサー属性のキーと値のペアのカンマ区切りリストを受け入れます。例えば、アクセスログ記録とクロスゾーンロードバランシングを指定するには、次の形式を使用します。

```
service.beta.kubernetes.io/aws-load-balancer-attributes:
 access_logs.s3.enabled=true,access_logs.s3.bucket=my-bucket,access_logs.s3.prefix=my-
prefix,load_balancing.cross_zone.enabled=true
```

この統合形式により、より優れた一貫性と柔軟性で、必要な個々の注釈の数を減らしつつロードバランサー属性を設定することができます。既存の Service 設定を確認し、この統合形式を使用するように更新してください。

TargetGroupBinding

旧	新	説明
`elbv2.k8s.aws/v1beta1`	`eks.amazonaws.com/v1`	API バージョンの変更
`spec.targetType` が任意	`spec.targetType` が必須	明示的なターゲットタイプの指定
`spec.networking.ingress.from`	サポートされていません	セキュリティグループなしでは NLB がサポートされなくなりました

注意: カスタム TargetGroupBinding 機能を使用するには、ターゲットグループにクラスター名と共に `eks:eks-cluster-name` タグを付けて、必要な IAM アクセス許可をコントローラーに付与する必要があります。TargetGroupBinding リソースまたはクラスターを削除すると、コントローラーがターゲットグループを削除することに注意してください。

マルチクラスターターゲットグループを設定する

デフォルトでは、Amazon EKS Auto Mode ロードバランサーコントローラーは、設定したターゲットグループの完全な制御を引き受けます。ターゲットグループがコントローラーに登録されると、現在クラスターにないターゲットが登録解除されます。これにより、ターゲットグループには、そのクラスターによって管理されるターゲットの正確なセットが反映されます。

MultiCluster サポートを有効にすると、この動作が変更されます。MultiCluster サポートが有効になっているターゲットグループは、複数の Kubernetes クラスターに関連付けることも、他のソースからの任意のターゲットをサポートすることもできます。各クラスターは、他のクラスターまたはソースに属するターゲットを登録解除する代わりに、所有するターゲットのみを登録および登録解除します。

仕組み

MultiCluster サポートが有効になっている場合、各クラスターは管理するターゲットのセットを追跡し、共有ターゲットグループ内のそれらのターゲットのみを調整します。Amazon EKS Auto Mode は、各クラスターの追跡されたターゲットを、ロードバランサーリソースと同じ名前空間の ConfigMap という名前の Kubernetes `eks-lbc-targets-<targetgroupbinding-name>` に保存します。

> **Note**
>
> アップストリームの AWS Load Balancer Controller は、この状態を `aws-lbc-targets-` プレフィックスを持つ ConfigMap に保存します。Amazon EKS Auto Mode では、プレフィックスは `eks-lbc-targets-` になるため、アップストリームガイダンスに従うときは、`eks-lbc-targets-` ConfigMap を検査します。

ロードバランサーは、登録されたすべてのターゲット全体にトラフィックを均等に分散します。コントローラーは、クラスター間の加重分散をサポートしていません。MultiCluster サポートは、`ip` と `instance` の両方のターゲットタイプで動作します。

MultiCluster サポートを有効にする

リソースタイプに応じて、次のいずれかの方法を使用して MultiCluster サポートを有効にします。

- TargetGroupBinding – `spec.multiClusterTargetGroup: true` を設定します。
- Ingress (ALB) – 注釈 `alb.ingress.kubernetes.io/multi-cluster-target-group: "true"` を追加します。
- Service (NLB) – 注釈 `service.beta.kubernetes.io/aws-load-balancer-multi-cluster-target-group: "true"` を追加します。

Amazon EKS 自動モードの考慮事項

AWS Load Balancer Controller GitHub ウェブサイトのアップストリーム MultiCluster Target Groups ガイドでは、2 つのクラスター間で 1 つのターゲットグループを共有する方法について説明します。マルチクラスターアーキテクチャの詳細については、AWS ブログの「Amazon EKS を使用した回復力のあるマルチクラスターアプリケーションの構築」を参照してください。Amazon EKS Auto Mode でも同じフローが動作しますが、次の違いがあります:

- `TargetGroupBinding` カスタムリソースは、`elbv2.k8s.aws/v1beta1` ではなく `eks.amazonaws.com/v1` API グループにあります。
- `IngressClass` コントローラーは `eks.amazonaws.com/alb` です。
- `alb.ingress.kubernetes.io/tags` 注釈を設定しないでください。`AmazonEKSLoadBalancingPolicy` マネージドポリシーは `aws:TagKeys` を許可リストに制限するので、カスタムタグはアクセス拒否エラーを引き起こし、コントローラーはそれらを調整できません。これにより、Ingress ファイナライザーが削除時にスタックする可能性があります。
- ターゲットグループは 1 つの VPC にスコープされているため、すべての参加クラスターは同じ VPC 内にある必要があります。

例

2 つのクラスター間で 1 つのターゲットグループを共有するステップバイステップの説明については、the section called “ターゲットグループを共有する” を参照してください。ここでは、1 つのクラスターは Ingress を介してロードバランサーを所有し、もう 1 つのクラスターは `TargetGroupBinding` を介して結合します。

2 つのクラスター間でターゲットグループを共有する

次の例では、2 つのクラスター間で 1 つのターゲットグループを共有します。最初のクラスターは Ingress を介して Application Load Balancer を所有し、2 番目のクラスターは手動で作成された `TargetGroupBinding` を介して共有ターゲットグループを結合します。開始する前に、the section called “マルチクラスターターゲットグループ” をレビューして、MultiCluster サポートの仕組みと有効にする方法を理解してください。

クラスター 1 – Ingress を介してロードバランサーを所有する

アプリケーション (`NodePort` Service を使用したデプロイなど) をデプロイし、マルチクラスター注釈を持つ `IngressClass` と `Ingress` を作成します。

```
apiVersion: networking.k8s.io/v1
kind: IngressClass
metadata:
  name: alb-mc
spec:
  controller: eks.amazonaws.com/alb
---
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: echoserver
  namespace: echoserver
  annotations:
    alb.ingress.kubernetes.io/scheme: internet-facing
    alb.ingress.kubernetes.io/target-type: instance
    alb.ingress.kubernetes.io/multi-cluster-target-group: "true"
spec:
  ingressClassName: alb-mc
  rules:
    - http:
        paths:
          - path: /
            pathType: Exact
            backend:
              service:
                name: echoserver
                port:
                  number: 80
```

コントローラーはロードバランサーをプロビジョニングし、spec.multiClusterTargetGroup: true を使用して TargetGroupBinding を生成します。生成された TargetGroupBinding と Service の次の値に注意してください。2 番目のクラスターには 3 つすべてが必要です:

- targetGroupARN。
- Service nodePort。
- spec.networking.ingress[].from[].securityGroup.groupID からのロードバランサーのセキュリティグループ ID。

クラスター 2 – TargetGroupBinding を介して共有ターゲットグループを結合する

同じアプリケーションをデプロイし、クラスター 1 の nodePort に Service をピン留めしてから、クラスター 1 のターゲットグループ ARN とセキュリティグループ ID を参照する TargetGroupBinding を作成します。

```
apiVersion: eks.amazonaws.com/v1
kind: TargetGroupBinding
metadata:
  name: mc-echoserver
  namespace: echoserver
spec:
  multiClusterTargetGroup: true
  serviceRef:
    name: echoserver
    port: 80
  targetGroupARN: <cluster-1-target-group-arn>
  targetType: instance
  networking:
    ingress:
      - from:
          - securityGroup:
              groupID: <cluster-1-alb-security-group-id>
        ports:
          - port: <cluster-2-node-port>
            protocol: TCP
```

両方のバインドで MultiCluster サポートを有効にすると、各クラスターは自身のターゲットのみを登録および登録解除します。両方のクラスターのノードは単一の共有ターゲットグループで正常な状態を維持し、ロードバランサーはそれらのすべてにトラフィックを分散します。

EKS Auto Mode でネットワークポリシーを使用する

概要

お客様が EKS を使用してアプリケーション環境をスケールするにつれて、クラスター内外のリソースへの不正アクセスを防ぐ必要があり、そのためにはネットワークトラフィックの分離がますます欠かせないものになります。これは、クラスター内で互いに関連性のない複数のワークロードが並行して動作するマルチテナント環境で特に重要です。Kubernetes ネットワークポリシーを使用すると、Kubernetes ワークロードのネットワークセキュリティ体制を強化し、それをクラスター外のエンドポイントと強固に統合できます。EKS Auto Mode は、さまざまなタイプのネットワークポリシーをサポートしています。

レイヤー 3 と 4 の分離

標準の Kubernetes ネットワークポリシーは OSI ネットワークモデルのレイヤー 3 と 4 で動作し、Amazon EKS クラスター内の IP アドレスまたはポートレベルでトラフィックフローを制御できます。

ユースケース

- ワークロード間でネットワークトラフィックをセグメント化すると、関連するアプリケーションのみが相互に通信できるようになります。
- ポリシーを使用して名前空間レベルでテナントを分離すると、ネットワークを分離できます。

DNS ベースの適用

お客様が EKS にデプロイするワークロードはより広範な分散環境に属しているのが一般的で、その中にはクラスター外のシステムやサービスとの通信を必要とするものもあります (インバウンドトラフィック)。こうしたシステムやサービスは、AWS クラウド内にあっても AWS 外にあってもかまいません。ドメインネームシステム (DNS) ベースのポリシーを使用すると、予測精度の高い安定したアプローチに沿って、ポッドからクラスター外のリソースやエンドポイントへの不正アクセスを防ぎ、セキュリティ体制を強化できます。このメカニズムにより、特定の IP アドレスを手動で追跡して許可リストに登録する必要がなくなります。DNS ベースのアプローチに沿ってリソースを保護することで、アップストリームサーバーやホストを変更する際にセキュリティ体制を緩和することも、ネットワークポリシーを変更することもなく、柔軟に外部インフラストラクチャを更新できます。完全修飾ドメイン名 (FQDN)、または DNS ドメイン名に一致するパターンを使用すると、外部エンドポイントへの出力トラフィックをフィルタリングできます。これにより、クラスター外の特定のエンドポイントに複数のサブドメインが関連付けられている場合に、各サブドメインへのアクセスを柔軟に拡張できます。

ユースケース

- Kubernetes 環境からクラスター外のエンドポイントへのアクセスをフィルタリングする場合には、DNS ベースのアプローチを標準とします。
- マルチテナント環境で AWS サービスへのアクセスを保護します。
- ハイブリッドクラウド環境でポッドからオンプレミスワークロードへのネットワークアクセスを管理します。

管理者 (またはクラスター範囲の) ルール

マルチテナントシナリオのように、ネットワークセキュリティ標準をクラスター全体に適用しなければならないという要件が求められる場合があります。名前空間ごとにポリシーを繰り返し定義して維持するのではなく、単一のポリシーを使用して、名前空間に関係なく、クラスター内のさまざまなワークロードに対するネットワークアクセスコントロールを一元管理できます。こうしたタイプのポリシーでは、レイヤー 3、レイヤー 4、および DNS ルールの使用時に適用されるネットワークフィルタリングルールの適用範囲を拡張できます。

ユースケース

- EKS クラスター内のすべての (または一部の) ワークロードに対するネットワークアクセスコントロールを一元管理します。
- クラスター全体にわたるデフォルトのネットワークセキュリティ体制を定義します。
- 組織のセキュリティ標準を運用効率の高い方法でクラスターの範囲へと拡張します。

開始方法

前提条件

- EKS Auto Mode が有効になっている Amazon EKS クラスター
- kubectl がクラスターに接続するように設定されている

ステップ 1: ネットワークポリシーコントローラーを有効にする

EKS Auto Mode でネットワークポリシーを使用するには、まずクラスターに ConfigMap を適用してネットワークポリシーコントローラーを有効にする必要があります。

1. 以下の内容で `enable-network-policy.yaml` という名前のファイルを作成します。

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: amazon-vpc-cni
  namespace: kube-system
data:
  enable-network-policy-controller: "true"
```

2. ConfigMap をクラスターに適用します。

```
kubectl apply -f enable-network-policy.yaml
```

ステップ 2: ネットワークポリシーを作成してテストする

これで、EKS Auto Mode クラスターが Kubernetes ネットワークポリシーをサポートするように設定されました。これは the section called “Stars Policy デモ” でテストできます。

ステップ 3: ノードクラスでネットワークポリシーエージェント設定を調整する (オプション)

オプションで新しいノードクラスを作成して、ノード上のネットワークポリシーエージェントのデフォルトの動作を変更したり、ネットワークポリシーイベントのログ記録を有効にしたりできます。これを実行するには、以下の手順を実行します。

1. 次の内容でノードクラス YAML ファイル (例: `nodeclass-network-policy.yaml`) を作成または編集します。

```
apiVersion: eks.amazonaws.com/v1
kind: NodeClass
metadata:
  name: network-policy-config
spec:
  # Optional: Changes default network policy behavior
  networkPolicy: DefaultAllow
  # Optional: Enables logging for network policy events
  networkPolicyEventLogs: Enabled
  # Include other Node Class configurations as needed
```

2. ノードクラス設定をクラスターに適用します。

```
kubectl apply -f nodeclass-network-policy.yaml
```

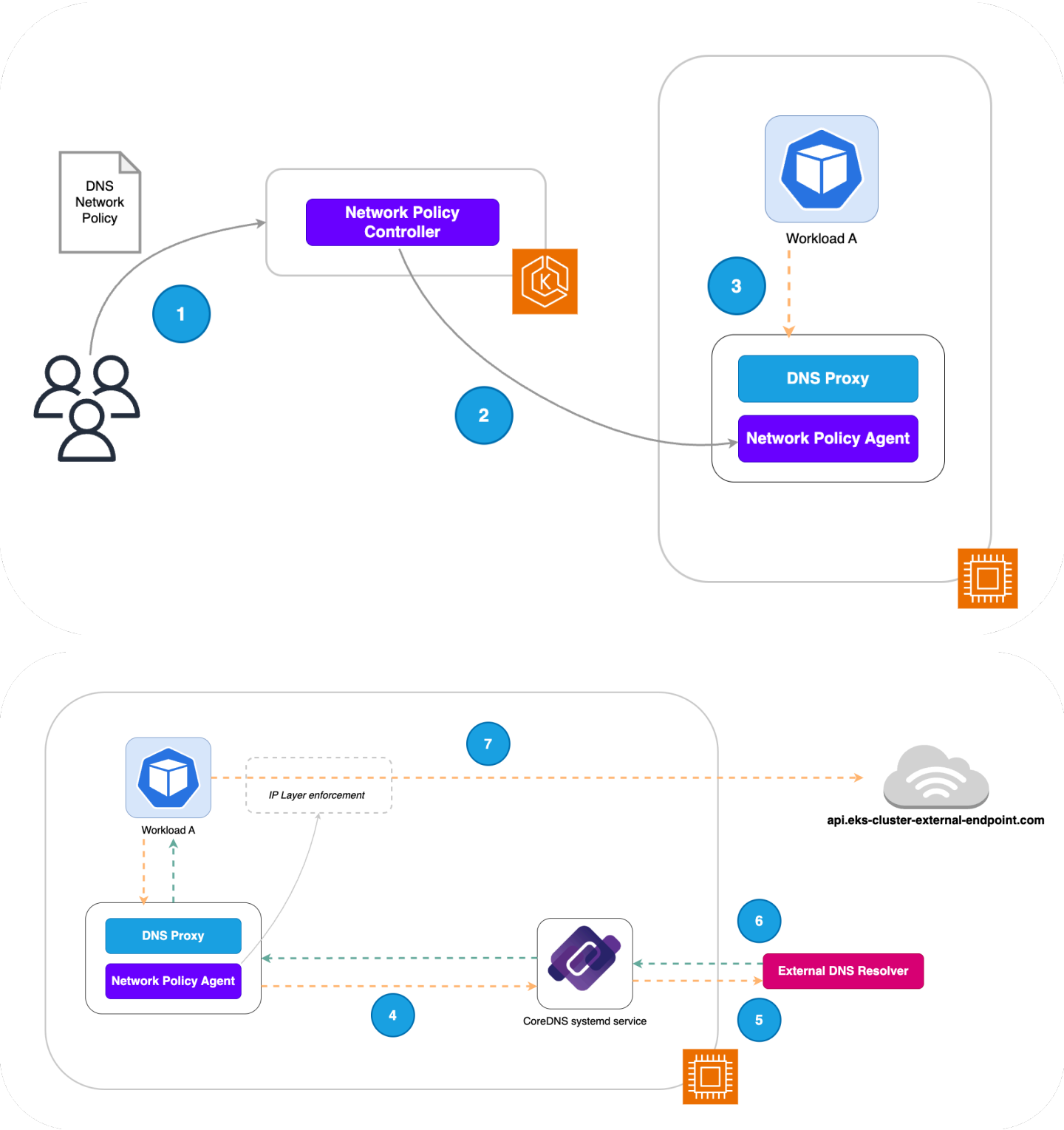
3. ノードクラスが作成されていることを確認します。

```
kubectl get nodeclass network-policy-config
```

4. このノードクラスを使用するようにノードプールを更新します。詳細については、the section called “ノードプールの作成” を参照してください。

動作の仕組み

DNS ベースのネットワークポリシー



1. プラットフォームチームが、DNS ベースのポリシーを EKS クラスターに適用します。

2. ネットワークポリシーコントローラーが、クラスター内でのポリシーの作成をモニタリングし、ポリシーエンドポイントを調整します。このユースケースの場合、ネットワークポリシーコントローラーは、作成されたポリシーの許可リストに登録されたドメインに基づいて DNS リクエストをフィルタリングするようにノードエージェントに指示します。ドメイン名を許可リストに登録する場合は、FQDN を使用するか、Kubernetes リソース設定に定義されたパターンと一致するドメイン名を使用します。
3. ワークロード A が、クラスター外のエンドポイントの IP を解決しようとします。DNS リクエストはまずプロキシを通過します。その際に、ネットワークポリシーで適用された許可リストに基づいて該当するリクエストがフィルタリングされます。
4. DNS リクエストが DNS フィルターの許可リストを通過すると、プロキシはそれを CoreDNS に転送します。
5. CoreDNS は、受け取ったリクエストを外部 DNS リゾルバー (Amazon Route 53 Resolver) に送信して、ドメイン名の背後にある IP アドレスのリストを取得します。
6. 解決した IP が TTL と共に、DNS リクエストへのレスポンスで返されます。こうして解決した IP が eBPF マップに書き込まれて、次のステップで IP レイヤーを適用する際に使用されます。
7. ポッドの veth インターフェイスにアタッチされた eBPF プローブが、所定のルールに基づいて、ワークロード A からクラスター外のエンドポイントへのエグレストラフィックをフィルタリングします。これにより、ポッドは許可リストに登録されたドメインの IP にのみクラスター外のトラフィックを送信できるようになります。こうした IP の有効性は、外部 DNS リゾルバー (Amazon Route 53 Resolver) から取得した TTL に基づいています。

ApplicationNetworkPolicy を使用する

`ApplicationNetworkPolicy` は、単一のカスタムリソース定義 (CRD) を使用して、標準の Kubernetes ネットワークポリシーの機能と DNS ベースのフィルタリングを名前空間レベルで結合します。そのため、`ApplicationNetworkPolicy` は以下の用途に使用できます。

1. IP ブロックとポート番号を使用して、ネットワークスタックのレイヤー 3 と 4 に制限内容を定義します。
2. ネットワークスタックのレイヤー 7 で機能するルールを定義し、FQDN に基づいてトラフィックをフィルタリングします。

> **Important**
>
> ApplicationNetworkPolicy で定義した DNS ベースのルールは、EKS Auto Mode で起動した EC2 インスタンスで動作するワークロードにのみ適用できます。ApplicationNetworkPolicy は、標準の Kubernetes NetworkPolicy のすべてのフィールドをサポートするほか、Egress ルール用の FQDN フィルターにも対応しています。

> **Warning**
>
> 同じ名前空間内で ApplicationNetworkPolicy と NetworkPolicy に同じ名前を使用しないでください。名前が競合すると、生成された PolicyEndpoints オブジェクトにどちらのポリシーも正しく反映されない可能性があります。エラーになることなくどちらのリソースも受け入れられるため、この問題を診断するのは容易ではありません。
> 名前の競合を解決するには、その名前空間内で一意になるように ApplicationNetworkPolicy または NetworkPolicy の名前を変更し、対応する PolicyEndpoints オブジェクトが正しく更新されることを確認します。

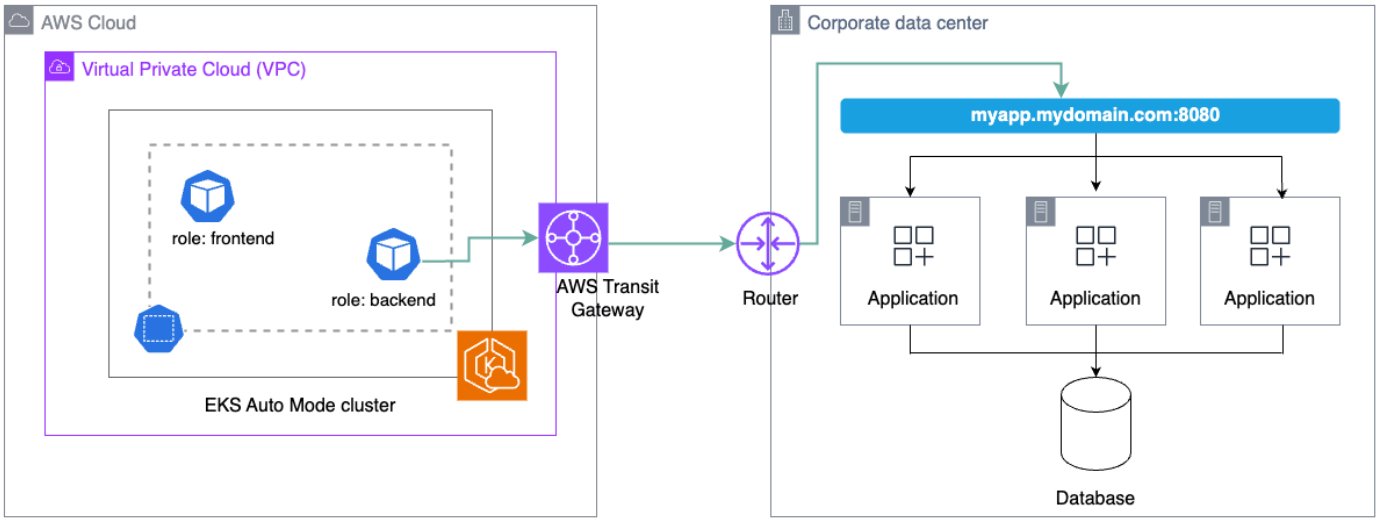
例

EKS Auto Mode クラスターにワークロードがあり、DNS 名が付与されたロードバランサーの背後にあるオンプレミスのアプリケーションと通信する必要があります。そのために、次のネットワークポリシーを使用します。

```
apiVersion: networking.k8s.aws/v1alpha1
kind: ApplicationNetworkPolicy
metadata:
  name: my-onprem-app-egress
  namespace: galaxy
spec:
  podSelector:
    matchLabels:
      role: backend
  policyTypes:
  - Egress
  egress:
  - to:
    - ipBlock: { cidr: 10.100.0.10/32 }
```

```
    ports:
    - protocol: TCP
      port: 53
    - protocol: UDP
      port: 53
  - to:
    - domainNames:
      - "myapp.mydomain.com"
    ports:
    - protocol: TCP
      port: 8080
```

Kubernetes ネットワークレベルでは、`role: backend` というラベルが付いた「galaxy」名前空間内のポッドからのエグレスが、TCP ポート 8080 で myapp.mydomain.com というドメイン名に接続し、ポート 53 で CoreDNS に接続できるようになります。また、VPC から社内のデータセンターへのエグレストラフィック用にネットワーク接続をセットアップする必要があります。



CoreDNS IP アドレスの決定

アプリケーションが DNS を解決するには、各 EKS Auto Mode インスタンスでローカルに実行される CoreDNS との通信が許可される必要があります。CoreDNS IP アドレスは、クラスター用に設定されたサービス CIDR 範囲から派生し、クラスターの存続期間中固定されたままになります。したがって、それを一度決定し、ネットワークポリシーで直接参照できます。

1. クラスターの Service CIDR を取得します。

   IPv4 クラスターの場合:

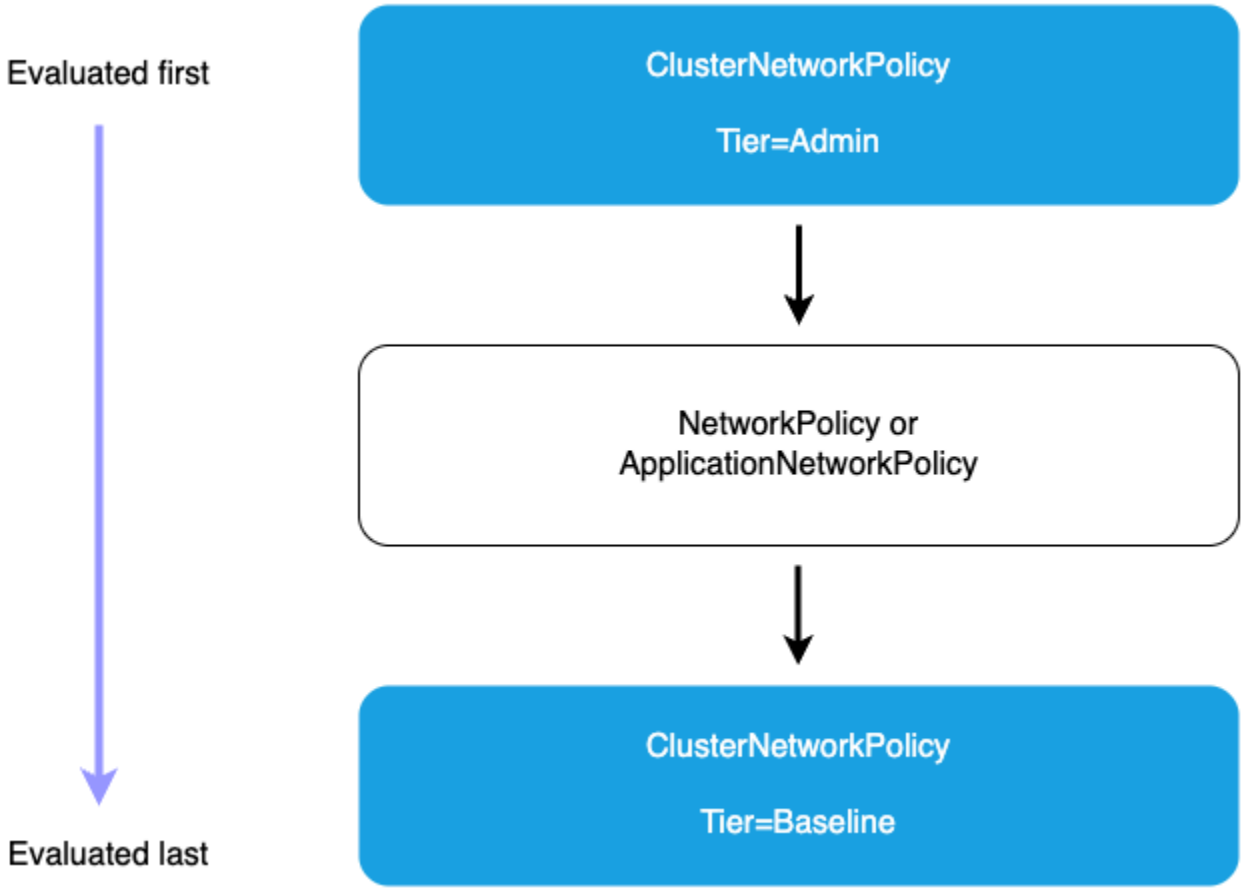
```
aws eks describe-cluster --name my-cluster --query
 'cluster.kubernetesNetworkConfig.serviceIpv4Cidr' --output text
```

IPv6 クラスターの場合:

```
aws eks describe-cluster --name my-cluster --query
 'cluster.kubernetesNetworkConfig.serviceIpv6Cidr' --output text
```

2. Service CIDR から CoreDNS IP アドレスを取得します。
   - IPv4 – CIDR の `.10` アドレスを使用します。例えば、`10.100.0.0/16` は `10.100.0.10/32` になります。
   - IPv6 – ネットワークアドレスに `a` を追加します。例えば、`fd12:3456:789a::/108` は `fd12:3456:789a::a/128` になります。

管理者 (あるいはクラスター) ネットワークポリシー



ClusterNetworkPolicy を使用する

ClusterNetworkPolicy を使用する場合、管理者階層ポリシーが最初に評価され、このポリシーを上書きすることはできません。管理者階層ポリシーが評価されたら、名前空間範囲の標準ポリシーに従って、適用したネットワークセグメンテーションルールが実行されます。そのためには、ApplicationNetworkPolicy または NetworkPolicy を使用します。最後に、ベースライン階層ルールが適用されます。ここには、クラスターワークロードに対するデフォルトのネットワーク制限が定義されています。こうしたベースライン階層ルールは、必要に応じて名前空間範囲のポリシーで上書きできます。

例

クラスター内にアプリケーションがあり、これを他のテナントワークロードから分離する必要があります。他の名前空間からクラスタートラフィックを明示的にブロックすると、機密性の高いワークロード名前空間へのネットワークアクセスを防ぐことができます。

```
apiVersion: networking.k8s.aws/v1alpha1
kind: ClusterNetworkPolicy
metadata:
  name: protect-sensitive-workload
spec:
  tier: Admin
  priority: 10
  subject:
    namespaces:
      matchLabels:
        kubernetes.io/metadata.name: earth
  ingress:
    - action: Deny
      from:
      - namespaces:
          matchLabels: {} # Match all namespaces.
      name: select-all-deny-all
```

考慮事項

ポリシー評価順序を理解する

EKS でサポートされているネットワークポリシー機能は、トラフィックを予測して安全に管理できるように、特定の順序で評価されます。そのため、評価の流れを理解したうえで、ご使用の環境に適した効果的なネットワークセキュリティ体制を設計することが重要です。

1. 管理者層ポリシー (最初に評価): 管理者階層のすべての ClusterNetworkPolicy が他のポリシーよりも先に評価されます。管理者階層内では、優先順位に従って (優先順位番号の最も小さいものから) ポリシーが処理されます。次にどうなるかは、アクションタイプによって決まります。
   - 拒否アクション (最も高い優先順位): 拒否アクションを定義した管理者ポリシーがトラフィックと一致すると、そのトラフィックは他のポリシーとは関係なくすぐにブロックされます。ClusterNetworkPolicy や NetworkPolicy ルールはそれ以上処理されません。これにより、組織全体のセキュリティコントロールを名前空間レベルのポリシーで上書きできないようにしています。
   - 許可アクション: 拒否ルールが評価されると、許可アクションを定義した管理者ポリシーが優先順位に従って (優先順位番号の最も小さいものから) 処理されます。許可アクションに一致すると、トラフィックは受け入れられ、ポリシーはそれ以上評価されません。これらのポリシーは、ラベルセレクターに基づいて複数の名前空間に対するアクセスを許可して、特定のリソースにどのワークロードがアクセスできるかを一元的に制御できます。
   - パスアクション: 管理階層ポリシーにパスアクションを定義すると、意思決定が下位の階層に委任されます。トラフィックがパスルールに一致すると、そのトラフィックの残りの管理者階層ルールの評価がスキップされ、NetworkPolicy 階層に直接進みます。これにより、管理者は特定のトラフィックパターンに対する制御をアプリケーションチームに明示的に委任できます。例えば、パスルールを使用すると、外部アクセスを厳密に制御しつつ、名前空間内のトラフィック管理を名前空間管理者に委任できます。
2. ネットワークポリシー階層: 拒否または許可に一致する管理者階層ポリシーがない場合や、パスアクションが一致した場合、次は名前空間範囲の ApplicationNetworkPolicy と従来の NetworkPolicy リソースが評価されます。これらは個々の名前空間内できめ細かく制御できるポリシーで、アプリケーションチームによって管理されます。名前空間範囲のポリシーでは、管理者ポリシーよりも厳しく制限を課すことができます。管理者ポリシーによる拒否の判断を上書きすることはできませんが、管理者ポリシーによって許可またはパスされたトラフィックをさらに制限できます。
3. ベースライン階層管理ポリシー: トラフィックに一致する管理者ポリシーや名前空間範囲のポリシーがない場合、ベースライン階層の ClusterNetworkPolicy が評価されます。これにより、デフォルトのセキュリティ体制を名前空間範囲のポリシーで上書きできるため、管理者が組織全体のデフォルトを設定する一方で、チームが必要に応じて柔軟にカスタマイズできます。ベースラインポリシーは、優先順位に従って (優先順位番号の最も小さいものから) 評価されます。
4. デフォルト拒否 (一致するポリシーがない場合): この deny-by-default 動作により、明示的に認められた接続のみが許可されるため、強力なセキュリティ体制を維持できます。

最小特権の原則を適用する

- 制限ポリシーから開始し、必要に応じて徐々にアクセス許可を追加する - まずはクラスターレベルで deny-by-default ポリシーを実装し、次に正当な接続要件の検証に合わせて許可ルールを段階的に追加していきます。このアプローチにより、チームは外部接続ごとにその正当性を明示的に評価して、より安全で監査可能な環境を構築できるようになります。
- 未使用のポリシールールを定期的に監査して削除する - ネットワークポリシーは、アプリケーションの進化に伴い時間が経つにつれて増えていきます。そのために、古いルールが残ってアタックサーフェスが不必要に拡大する可能性があります。ポリシールールを定期的に見直すプロセスを実装して、不要になったものを特定して削除するようにすると、厳格で管理しやすいセキュリティ体制を維持できます。
- 可能であれば、広範なパターンではなく特定のドメイン名を使用する - `*.amazonaws.com` のようなワイルドカードパターンは便利ですが、同時に幅広いサービスへのアクセスも許可することになります。可能な限り、`s3.us-west-2.amazonaws.com` のような厳密なドメイン名を指定して、アプリケーションが必要とする特定のサービスにのみアクセスを制限して、ワークロードが侵害された場合に横展開されるリスクを軽減します。

EKS での DNS ベースのポリシーを使用する

- `ApplicationNetworkPolicy` を使用して定義した DNS ベースのルールは、EKS Auto Mode で起動された EC2 インスタンスで動作するワークロードにのみ適用できます。混合モードクラスター (EKS Auto ワーカーノードと非 EKS Auto ワーカーノードの両方で設定) を実行している場合、DNS ベースのルールは EKS Auto Mode のワーカーノード (EC2 マネージドインスタンス) でのみ有効です。

DNS ポリシーを検証する

- ステージングクラスターを使用してテスト用に本番稼働のネットワークトポロジをミラーリングする - ステージング環境では、ポリシーを正確にテストできるように、本番稼働のネットワークアーキテクチャ、外部依存関係、および接続パターンをレプリケートする必要があります。例えば、VPC 設定、DNS 解決動作、本番稼働のワークロードで必要になるのと同じ外部サービスへのアクセスを一致させます。
- 重要なネットワークパスを自動的にテストする機能を実装する - CI/CD パイプラインの一部として、必要不可欠な外部サービスへの接続を検証する自動テストを構築します。自動テストでは、正当なトラフィックフローは許可され、不正な接続はブロックされることを検証します。インフラス

トラクチャが進化しても、ネットワークポリシーにより、正しいセキュリティ体制が維持されていることを継続的に検証する必要があります。

- ポリシーが変更されたらアプリケーションの動作をモニタリングする - 新規または変更後のネットワークポリシーを本番稼働にデプロイしたら、アプリケーションログ、エラー率、パフォーマンスメトリクスを注意深くモニタリングして、接続の問題があればすばやく特定できるようにします。明確なロールバック手順を確立して、予期しないアプリケーションの動作やサービスの中断が発生したら、ポリシーへの変更をすばやく元に戻せるようにします。

Amazon Route 53 DNS ファイアウォールとのインタラクション

EKS 管理者ポリシーとネットワークポリシーは、トラフィックの開始時にポッドレベルで最初に評価されます。EKS ネットワークポリシーで特定のドメインへのエグレスを許可している場合、ポッドは Route 53 Resolver に到達する DNS クエリを実行します。この時点で、Route 53 DNS ファイアウォールルールが評価されます。DNS ファイアウォールがドメインクエリをブロックしている場合、EKS ネットワークポリシーで許可していたとしても、DNS 解決は失敗し、接続を確立できません。そのため、セキュリティレイヤーをいくつか作成して補完します。EKS DNS ベースのネットワークポリシーではアプリケーションに固有のアクセス要件とマルチテナントセキュリティ境界に応じてポッドレベルでエグレスを制御し、DNS ファイアウォールでは既知の悪意のあるドメインから VPC 全体を保護して組織全体のブロックリストを適用します。

EKS Auto Mode のサブネットにタグを付ける

EKS Auto Mode の負荷分散機能を使用する場合は、VPC サブネットに AWS タグを追加する必要があります。

背景

これらのタグは、サブネットがクラスターに関連付けられているものと特定し、さらに重要な点として、サブネットがパブリックかプライベートかを識別します。

パブリックサブネットは、インターネットゲートウェイを介してインターネットに直接アクセスできます。ロードバランサーなど、パブリックにアクセスできるようにする必要があるリソースに使用されます。

プライベートサブネットには直接インターネットにアクセスできる機能がなく、アウトバウンドトラフィックに NAT ゲートウェイを使用します。プライベートサブネットは、パブリック IP を必要としない EKS ノードなどの内部リソースに使用されます。

NAT ゲートウェイとインターネットゲートウェイの詳細については、「Amazon Virtual Private Cloud (VPC) ユーザーガイド」の「VPC を他のネットワークに接続する」を参照してください。

要件

現時点で、EKS Auto Mode で負荷分散に使用するサブネットには、次のいずれかのタグが必要です。

パブリックサブネット

パブリックサブネットは、インターネット向けロードバランサーに使用されます。これらのサブネットには、次のタグが必要です。

キー	値
kubernetes.io/role/elb	1 または ``

プライベートサブネット

プライベートサブネットは、内部ロードバランサーに使用されます。これらのサブネットには、次のタグが必要です。

キー	値
kubernetes.io/role/internal-elb	1 または ``

手順

開始する前に、パブリックサブネット (インターネットゲートウェイアクセスを使用) とプライベートサブネット (NAT ゲートウェイを使用) を特定します。VPC リソースを変更するには、アクセス許可が必要です。

AWS マネジメントコンソール

1. Amazon VPC コンソールを開き、[サブネット] に移動します。
2. タグ付けするサブネットを選択します。
3. [タグ] タブを選択してから、[タグを追加] を選択します。
4. 適切なタグを追加します。
   - パブリックサブネットの場合: Key=kubernetes.io/role/elb

- プライベートサブネットの場合: Key=kubernetes.io/role/internal-elb

5. [値] を 1 に設定するか、空白のままにします。
6. 保存し、残りのサブネットについても同じ操作を繰り返します。

AWS CLI

パブリックサブネットの場合:

```
aws ec2 create-tags \
    --resources subnet-ID \
    --tags Key=kubernetes.io/role/elb,Value=1
```

プライベートサブネットの場合:

```
aws ec2 create-tags \
    --resources subnet-ID \
    --tags Key=kubernetes.io/role/internal-elb,Value=1
```

subnet-ID を実際のサブネット ID に置き換えます。

ストレージの設定

Amazon EKS Auto Mode がワークロードのストレージをどのようにプロビジョニングするかを設定します。以下のトピックを参照して、ストレージクラスとボリューム設定を定義します。

トピック

- ストレージクラスを作成する

ストレージクラスを作成する

Amazon EKS Auto Mode の StorageClass は、アプリケーションが永続的ストレージをリクエストしたときの Amazon EBS ボリュームを自動的にプロビジョニングする方法を定義します。このページでは、Amazon EKS Auto Mode で動作する StorageClass を作成して設定し、EBS ボリュームをプロビジョニングする方法について説明します。

StorageClass を設定することで、ボリュームタイプ、暗号化、IOPS、その他のストレージパラメータなど、EBS ボリュームのデフォルト設定を指定できます。暗号化管理に AWS KMS キーを使用するように StorageClass を設定することもできます。

EKS Auto Mode では、お客様用の StorageClass は作成されません。EKS Auto Mode のストレージ機能を使用するには、ebs.csi.eks.amazonaws.com を参照する StorageClass を作成する必要があります。

まず、storage-class.yaml という名前のファイルを作成します。

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: auto-ebs-sc
  annotations:
    storageclass.kubernetes.io/is-default-class: "true"
allowedTopologies:
- matchLabelExpressions:
  - key: eks.amazonaws.com/compute-type
    values:
    - auto
provisioner: ebs.csi.eks.amazonaws.com
volumeBindingMode: WaitForFirstConsumer
parameters:
  type: gp3
  encrypted: "true"
```

次に、ストレージクラスをクラスターに適用します。

```
kubectl apply -f storage-class.yaml
```

主要コンポーネント:

- provisioner: ebs.csi.eks.amazonaws.com - EKS Auto Mode を使用する
- allowedTopologies - matchLabelExpressions を eks.amazonaws.com/compute-type:auto に一致するように指定した場合、自動モードを使用してポッドに必要なボリュームを自動的にプロビジョニングするようにすれば、そのポッドは自動以外のノードにスケジュールされなくなります。
- volumeBindingMode: WaitForFirstConsumer - ポッドが必要とするまでボリュームの作成を遅らせる
- type: gp3 - EBS ボリュームタイプを指定する
- encrypted: "true" – EBS は、StorageClass を使用して作成されたすべてのボリュームを暗号化します。EBS はデフォルトの aws/ebs キーエイリアスを使用します。詳細については、

「Amazon EBS ユーザーガイド」の「How Amazon EBS encryption works」を参照してください。この値は省略可能ですが、推奨です。

- `storageclass.kubernetes.io/is-default-class: "true"` – 永続ボリュームのクレームで別のボリュームクラスを指定しない限り、Kubernetes はデフォルトでこのストレージクラスを使用します。この値はオプションです。別のストレージコントローラーから移行する場合は、この値を設定するときに注意してください。

セルフマネージド KMS キーを使用して EBS ボリュームを暗号化する

セルフマネージド KMS キーを使用して EKS Auto Mode によって自動化される EBS ボリュームを暗号化するには、以下を実行する必要があります。

1. セルフマネージド KMS キーを作成します。
   - 詳細については、「KMS ユーザーガイド」の「Create a symmetric encryption KMS key」または「How Amazon Elastic Block Store (Amazon EBS) uses KMS」を参照してください。
2. KMS キーへのアクセスを許可する新しいポリシーを作成します。
   - 以下のサンプル IAM ポリシーを使用してポリシーを作成します。新しいセルフマネージド KMS キーの ARN を挿入します。詳細については、「AWS IAM ユーザーガイド」の「ロールの作成とポリシーのアタッチ (コンソール)」を参照してください。
3. ポリシーを EKS クラスターロールにアタッチします。
   - AWS コンソールを使用して、EKS クラスターロールの ARN を見つけます。ロール情報は [概要] セクションに表示されます。詳細については、「the section called “クラスター IAM ロール”」を参照してください。
4. `parameters.kmsKeyId` フィールドの KMS キー ID を参照するように `StorageClass` を更新します。

サンプルセルフマネージド KMS IAM ポリシー

以下のポリシーで次の値を更新します。

- `<account-id>` – AWS アカウント ID (111122223333 など)
- `<aws-region>` – クラスターの AWS リージョン (us-west-2 など)

```
{
  "Version":"2012-10-17",
```

```
  "Id": "key-auto-policy-3",
  "Statement": [
      {
          "Sid": "Enable IAM User Permissions",
          "Effect": "Allow",
          "Principal": {
              "AWS": "arn:aws:iam::123456789012:root"
          },
          "Action": "kms:*",
          "Resource": "*"
      },
      {
        "Sid": "Allow access through EBS for all principals in the account that are
 authorized to use EBS",
        "Effect": "Allow",
        "Principal": {
            "AWS": "*"
        },
        "Action": [
            "kms:Encrypt",
            "kms:Decrypt",
            "kms:ReEncrypt*",
            "kms:GenerateDataKey*",
            "kms:CreateGrant",
            "kms:DescribeKey"
        ],
        "Resource": "*",
        "Condition": {
            "StringEquals": {
                "kms:CallerAccount": "123456789012",
                "kms:ViaService": "ec2.us-east-1.amazonaws.com"
            }
        }
    }
  ]
}
```

サンプルセルフマネージド KMS **StorageClass**

```
parameters:
  type: gp3
  encrypted: "true"
  kmsKeyId: <custom-key-arn>
```

`StorageClass` パラメータリファレンス

Kubernetes `StorageClass` リソースの一般的な情報については、Kubernetes ドキュメントの「ストレージクラス」を参照してください。

`StorageClass` リソースの `parameters` セクションは AWS に固有です。次の表を使用して使用可能なオプションを確認してください。

パラメータ	値	デフォルト	説明
「csi.storage.k8s.io/fstype」	xfs、ext2、ext3、ext4	ext4	ボリュームの作成時にフォーマットされるファイルシステムタイプ。このパラメータは大文字と小文字が区別されます。
「type」	io1、io2、gp2、gp3、sc1、st1、standard、sbp1、sbg1	gp3	EBS ボリュームタイプ。
「iopsPerGB」			GiB ごとの 1 秒あたりの I/O オペレーション回数秒。IO1、IO2、GP3 ボリュームで指定できます。
「allowAutoIOPSPerGBIncrease」	true、false	false	`"true"` であると、`iopsPerGB * <volume size>` が小さすぎて AWS でサポートされている IOPS 範囲に収まらない場合に、CSI ドライバーによりボリュームの IOPS が増加します。この結

パラメータ	値	デフォルト	説明
			果、ユーザーが指定する PVC キャパシティーや iopsPerGB 値が小さすぎる場合でも、動的プロビジョニングが常に成功します。一方、このようなボリュームの IOPS は iopsPerGB でリクエストされた値よりも高くなるため、追加コストが発生する可能性があります。
「iops」			1 秒あたりの I/O オペレーション回数。IO1、IO2、GP3 ボリュームで指定できます。
「throughput」		125	単位 MiB/秒でのスループット。gp3 ボリュームタイプが指定されている場合にのみ有効です。
「encrypted」	true、false	false	ボリュームを暗号化するかどうか。有効な値は「true」または「false」です。
「blockExpress」	true、false	false	io2 Block Express ボリュームの作成を有効にします。

パラメータ	値	デフォルト	説明
「kmsKeyId」			ボリュームの暗号化時に使用するキーの完全な ARN。指定されていない場合、AWS ではボリュームが存在するリージョンのデフォルトの KMS キーが使用されます。これは、変更されていない場合、/aws/ebs という名前の自動生成キーになります。
「blockSize」			基盤となるファイルシステムをフォーマットするときに使用するブロックサイズ。Linux ノードと fstype ext2、ext3、ext4 または xfs でのみサポートされます。
「inodeSize」			基盤となるファイルシステムをフォーマットするときに使用する inode サイズ。Linux ノードと fstype ext2、ext3、ext4 または xfs でのみサポートされます。

パラメータ	値	デフォルト	説明
「bytesPerInode」			基盤となるファイルシステムをフォーマットするときに使用する `bytes-per-inode` 。Linux ノードと fstype `ext2`、`ext3`、`ext4` でのみサポートされます。
「numberOfInodes」			基盤となるファイルシステムをフォーマットするときに使用する `number-of-inodes` 。Linux ノードと fstype `ext2`、`ext3`、`ext4` でのみサポートされます。
「ext4BigAlloc」	true、false	false	`bigalloc` フォーマットオプションを有効にして、クラスター化されたブロック割り当てを使用するように `ext4` ファイルシステムを変更します。警告: ノードの Linux カーネルでは `bigalloc` が完全にはサポートされない場合があります。

パラメータ	値	デフォルト	説明
「ext4ClusterSize」			bigalloc 機能が有効になっている場合に、ext4 ファイルシステムをフォーマットするときに使用するクラスターサイズ。注: ext4BigAlloc パラメータを true に設定する必要があります。

詳細については、GitHub の「AWS EBS CSI Driver」を参照してください。

考慮事項

> **Note**
>
> EKS オートモードノードの EKS オートモード StorageClasses に依存するワークロードのみをデプロイできます。ノードのタイプが混在するクラスターがある場合は、ワークロードを EKS オートモードノードでのみ実行するように設定する必要があります。詳細については、「the section called “デプロイを制御する”」を参照してください。

EKS Auto Mode のブロックストレージ機能は、EBS CSI ドライバーとは異なります。

- 静的プロビジョニング
  - EKS Auto Mode で外部作成の EBS ボリュームを使用する場合は、キー eks:eks-cluster-name とクラスター名の値を含む AWS タグを手動で追加する必要があります。
- ノード起動テイント
  - ノード起動テイント機能を使用して、ストレージ機能の準備が整う前にポッドのスケジューリングを防ぐことはできません
- 動的にプロビジョニングされたボリュームのカスタムタグ
  - 追加タグ CLI フラグを使用して、動的にプロビジョニングされた EBS ボリュームにカスタムタグを設定することはできません

  - `StorageClass` タグ付けを使用してカスタムタグを追加できます。EKS Auto Mode は、関連付けられた AWS リソースにタグを追加します。カスタムタグのクラスター IAM ロールを更新する必要があります。詳細については、「the section called “EKS Auto リソースのカスタム AWS タグ”」を参照してください。
- EBS 詳細パフォーマンスメトリクス
  - EBS 詳細パフォーマンスの Prometheus メトリクスにアクセスできません

> ⚠ 注釈ベースの EBS ボリューム変更は非推奨になりました
>
> `volume-modifier-for-k8s` プロジェクトは、ネイティブ Kubernetes `VolumeAttributesClass` API に移行するため非推奨になりました。このプロジェクトは、EBS ボリュームを変更するために PersistentVolumeClaim に注釈を設定する目的で使用されていました。2026 年 10 月 31 日以降、Amazon EKS Auto Mode ではこれらの注釈はサポートされなくなります。その日付より後に EBS ボリューム (`type`、`iops`、`throughput` など) を変更するには、代わりに Kubernetes `VolumeAttributesClass` API を使用します。詳細については、Kubernetes ドキュメントの「VolumeAttributesClass」を参照してください。

volume-modifier-for-k8s から VolumeAttributesClass に移行する

Kubernetes 1.34 以降を実行している EKS Auto Mode クラスターの場合、`VolumeAttributesClass` (VAC) を使用して EBS ボリュームのタイプ、IOPS、またはスループットを変更します。

既存の PVC は、VAC を作成し、既存の PVC に `spec.volumeAttributesClassName` を設定することでインプレースで移行できます。PVC と PV は再作成されず、このプロセス中にワークロードが中断されることはありません。

1. VAC でボリューム設定を定義する

   VMK 注釈は、次のように VAC パラメータにマッピングされます。

既存の VMK 注釈	VAC パラメータ
`ebs.csi.eks.amazonaws.com/volumeType`	型

既存の VMK 注釈	VAC パラメータ
ebs.csi.eks.amazonaws.com/iops	iops
ebs.csi.eks.amazonaws.com/throughput	スループット

VAC では volumeType ではなく type を使用します。値は文字列でなければならず、VAC には目的の EBS 設定に関連するパラメータのみが必要です。例:

```
apiVersion: storage.k8s.io/v1
kind: VolumeAttributesClass
metadata:
  name: ebs-auto-gp3-6000-250
driverName: ebs.csi.eks.amazonaws.com
parameters:
  type: gp3
  iops: "6000"
  throughput: "250"
```

VAC はクラスタースコープであるため、マニフェストには名前空間がありません。次のように通常どおり適用します。

```
kubectl apply -f vac.yaml
```

2. VAC を PVC に割り当てる

```
VAC_NAME=ebs-auto-gp3-6000-250

kubectl patch pvc <pvc-name> -n <namespace> --type=merge \
  -p "{\"spec\":{\"volumeAttributesClassName\":\"${VAC_NAME}\"}}"
```

3. 変更が完了したことを確認する

```
kubectl get pvc <pvc-name> -n <namespace> \
  -o custom-
columns='NAME:.metadata.name,REQUESTED:.spec.volumeAttributesClassName,CURRENT:.status.curren
```

移行は、REQUESTED と CURRENT に同じ VAC 名が含まれ、STATUS が空になった時点で完了します。

完了しない場合は、PVC イベントを次のように確認します。

```
kubectl describe pvc <pvc-name> -n <namespace>
```

Pending は通常、参照される VAC が使用できないことを意味します。Infeasible は、ドライバーがリクエストされた EBS パラメータを拒否したことを意味します。代わりに有効な VAC を作成して割り当てます。

オプション: VMK 注釈を削除する

VAC が機能するために、古い注釈を削除する必要はありません。PVC または PV が VAC を参照すると、VMK は注釈ベースの変更を拒否し、VAC が信頼できるパスになります。

注釈は、後で設定のクリーンアップとして削除できます。削除する場合は、Helm、GitOps、またはそれらを追加するその他のオートメーションも更新してください。PV に残された VMK ステータス注釈は無害であり、削除する必要はありません。

設定の変更または復元

VAC パラメータは変更できません。EBS 設定を変更するには、別の VAC を作成し、その名前を PVC に割り当てます。

完了していない変更をキャンセルするには、以前に割り当てられた VAC を復元します。PVC が以前に使用していない場合は、フィールドをクリアします。

```
kubectl patch pvc <pvc-name> -n <namespace> --type=merge \
  -p '{"spec":{"volumeAttributesClassName":null}}'
```

フィールドをクリアしても、すでに完了した EBS の変更は元に戻されません。以前のボリューム設定を復元するには、それらの設定を含む VAC を作成し、PVC に割り当てます。通常の EBS パラメータの制約と変更のクールダウンが適用されます。

CSI スナップショットコントローラーアドオンをインストールする

EKS Auto Mode は、CSI スナップショットコントローラー Amazon EKS アドオンと互換性があります。

AWS では、組み込み system ノードプールで実行するようにこのアドオンを設定することをお勧めします。

詳細については、以下を参照してください。

- the section called “重要なアドオンを実行する”
- the section called “組み込みノードプールを確認する”
- the section called “CSI スナップショットコントローラー”

スナップショットコントローラーをシステムノードプールにインストールするには

1. AWS コンソールで EKS クラスターを開きます
2. [アドオン] タブから、[アドオンをさらに追加] を選択します
3. [CSI スナップショットコントローラー]、[次へ] の順に選択します
4. [選択したアドオン設定を構成する] ページで、[オプションの設定] を選択して [アドオン設定スキーマ] を表示します
   a. 次の yaml を挿入して、スナップショットコントローラーを system ノードプールに関連付けます。スナップショットコントローラーには、CriticalAddonsOnly テイントの許容が含まれています。

```
{
        "nodeSelector": {
            "karpenter.sh/nodepool": "system"
        }
}
```

   b. [次へ] を選択します
5. アドオン設定を確認し、[作成] を選択します

セキュリティとコンプライアンスを設定

Amazon EKS Auto Mode クラスターのセキュリティとコンプライアンス体制を設定します。以下のトピックを参照して、マネージドコントローラーへの RBAC の付与、コンプライアンスレポートの生成、ボリュームの暗号化、および高度なセキュリティ機能の有効化を行います。

トピック

- EKS Auto Mode のマネージドコントローラーに追加の Kubernetes RBAC を付与する

- EKS Auto Mode のロードバランサーコントローラーに特定のシークレットへのアクセスを許可する
- kubectl debug を使用して Kubernetes ノードの CIS コンプライアンスレポートを生成する
- EKS オートモードでカスタマーマネージド KMS キー使用した EBS ボリュームの暗号化を有効にする
- ノードの高度なセキュリティ設定を構成する

EKS Auto Mode のマネージドコントローラーに追加の Kubernetes RBAC を付与する

Amazon EKS Auto Mode は、ロードバランシング、ネットワーキング、コンピューティング、ブロックストレージ、クラスターインサイトの各コントローラーをお客様に代わって管理します。これらのコントローラーは EKS コントロールプレーン上で実行されます。これらは `AWSServiceRoleForAmazonEKS` サービスにリンクされたロールとしてお客様のクラスターに対して認証を行い、クラスター内でのアクセス許可は、そのアクセスエントリにアタッチされた AWS マネージドアクセスポリシーから付与されます。

マネージドコントローラーの Kubernetes RBAC は、お客様自身で拡張できます。これは、マネージドポリシーが必要なアクセス許可をまだ付与していない場合に役立ちます。例えば、特定の `Secret` に対する `get` などです。

Amazon EKS Auto Mode のクラスターでは、`AWSServiceRoleForAmazonEKS` に対して自動作成されたアクセスエントリが、Kubernetes グループ `eks:managed` も宣言します。Kubernetes RBAC は付加的です。`subjects:` に `eks:managed` グループを指定した `RoleBinding` (または `ClusterRoleBinding`) は、マネージドアクセスポリシーに加えて、マネージドコントローラーに追加のアクセス許可を付与します。

仕組み

- EKS Auto Mode のマネージドコントローラーは、サービスにリンクされたロール `AWSServiceRoleForAmazonEKS` としてお客様のクラスターに接続します。
- そのロールに対して自動作成されたアクセスエントリは、`kubernetesGroups: ["eks:managed"]` を宣言します。
- 追加のアクセス許可を記述する `Role` (名前空間スコープ) または `ClusterRole` (クラスター全体) と、`subjects:` に `eks:managed` グループを指定した `RoleBinding` / `ClusterRoleBinding` を作成します。
- Kubernetes RBAC はお客様のバインディングとマネージドアクセスポリシーを組み合わせるため、マネージドコントローラーは両方のアクセス許可を受け取ります。

グループが存在することを確認するには、以下を実行します。

```
CLUSTER=<your-cluster>
REGION=<your-region>
ACCT=$(aws sts get-caller-identity --query Account --output text)

aws eks describe-access-entry \
  --cluster-name "$CLUSTER" --region "$REGION" \
  --principal-arn "arn:aws:iam::${ACCT}:role/aws-service-role/eks.amazonaws.com/
AWSServiceRoleForAmazonEKS" \
  --query 'accessEntry.kubernetesGroups'
```

正常な出力:

```
[
    "eks:managed"
]
```

付与のスコープを厳密にする

eks:managed グループは、すべての EKS Auto Mode のマネージドコントローラーに適用されます。ユースケースで許容される範囲で追加付与するスコープをできるだけ狭め、マネージドコントローラーに必要なものだけを公開するようにします。

- Role+RoleBinding (名前空間スコープ) を、ClusterRole+ClusterRoleBinding (クラスター全体) よりも優先します。
- resources、resourceNames、verbs は、ユースケースを妨げない最小限のセットに制限します。例えば、すべての Secret オブジェクトに対する list, watch ではなく、名前を指定した単一の Secret に対する get を付与します。
- 不要になったら、Role と RoleBinding を削除します。

Amazon EKS Auto Mode の考慮事項

- この仕組みは、EKS Auto Mode のマネージドコントローラーの Kubernetes RBAC を拡張するだけです。マネージドアクセスポリシーによって付与される AWS IAM アクセス許可は変更しません。
- eks:managed グループは、サービスにリンクされたロールが AWSServiceRoleForAmazonEKS である EKS Auto Mode のクラスターに存在します。

- eks:managed へのバインディングは、すべての EKS Auto Mode のマネージドコントローラーに適用されます。実際の影響範囲を小さく抑えるには、resourceNames と名前空間の Role オブジェクトを使用します。これにより、名前を指定したリソースのみが、付与した動詞を通じてのみ公開されます。
- これは、マネージドアクセスポリシーが必要なアクセス許可をまだカバーしていない場合を対象としたブリッジパターンです。AWS マネージドポリシーが既にそのアクセス許可をカバーしている場合は、代わりにマネージドポリシーを使用します。

例

EKS Auto Mode のロードバランサーコントローラーに、名前を指定した OIDC Secret に対する get を付与する (ALB Ingress の alb.ingress.kubernetes.io/auth-type: oidc 注釈のブロックを解除します) 手順を順を追って説明するチュートリアルについては、「the section called “LBC にシークレットへのアクセスを許可する”」を参照してください。

EKS Auto Mode のロードバランサーコントローラーに特定のシークレットへのアクセスを許可する

このチュートリアルでは、Amazon EKS Auto Mode のロードバランサーコントローラーに、特定の Kubernetes Secret への読み取りアクセス権を付与します。これにより、ALB Ingress での OIDC 認証が可能になります。eks:managed グループを対象にした、名前空間スコープの Role と RoleBinding を使用します。

このチュートリアルの完了には約 15 分かかります。作成する Kubernetes RBAC オブジェクトには AWS 料金は発生しません。

Amazon EKS Auto Mode のロードバランサーコントローラーは、ALB Ingress オブジェクトでの OIDC 認証をサポートしています。この認証を有効にするには、alb.ingress.kubernetes.io/auth-type: oidc アノテーションを使用します。OIDC 設定を解決するために、コントローラーは alb.ingress.kubernetes.io/auth-idp-oidc 注釈で指定された Kubernetes Secret を読み取ります。現在、そのマネージドアクセスポリシーは Secret オブジェクトに対する get を付与していないため、リコンサイラーは失敗し、次のようなメッセージを受け取ります。

```
Failed build model due to ingress: <ns>/<name>:
  secrets "<secret>" is forbidden:
  User "arn:aws:sts::<acct>:assumed-role/AWSServiceRoleForAmazonEKS/<session>"
  cannot get resource "secrets" in API group "" in the namespace "<ns>"
```

the section called “マネージドコントローラーに RBAC を付与する” で説明されているメカニズムを使用すると、マネージドコントローラーに、必要な OIDC Secret に対する get のみを付与できます。

前提条件

開始する前に、以下の準備が整っていることを確認します。

- 自動的に作成された AWSServiceRoleForAmazonEKS アクセスエントリを持つ Amazon EKS Auto Mode クラスター。
- クラスターアクセスが設定された kubectl。
- aws sts get-caller-identity と aws eks describe-access-entry のアクセス許可でインストールおよび設定された AWS CLI。
- alb.ingress.kubernetes.io/auth-type: oidc 注釈を使用する ALB Ingress、およびそれが参照する Kubernetes Secret。

ステップ 1: 許可リストに追加するシークレットを特定する

ALB Ingress 上の OIDC 設定は次のようになります。Secret の名前と Ingress 名前空間を記録します – RBAC オブジェクトは Secret と同じ名前空間に存在する必要があります。

```
annotations:
  alb.ingress.kubernetes.io/auth-type: oidc
  alb.ingress.kubernetes.io/auth-idp-oidc: >
    {
      "issuer": "...",
      "authorizationEndpoint": "...",
      "tokenEndpoint": "...",
      "userInfoEndpoint": "...",
      "secretName": "oidc"
    }
```

この例の残りの部分では、名前空間 monitoring と Secret 名 oidc を使用します。

ステップ 2: 名前空間スコープのロールとロールバインディングを適用する

この形式では、1 つの名前空間内の指定された Secret に対して get のみを付与します。次を eks-managed-oidc-secret-reader.yaml として保存します。

```
apiVersion: rbac.authorization.k8s.io/v1
```

```
kind: Role
metadata:
  name: eks-managed-oidc-secret-reader
  namespace: monitoring
rules:
- apiGroups: [""]
  resources: ["secrets"]
  resourceNames: ["oidc"]
  verbs: ["get"]
---
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: eks-managed-oidc-secret-reader
  namespace: monitoring
subjects:
- kind: Group
  name: eks:managed
  apiGroup: rbac.authorization.k8s.io
roleRef:
  kind: Role
  name: eks-managed-oidc-secret-reader
  apiGroup: rbac.authorization.k8s.io
```

適用:

```
kubectl apply -f eks-managed-oidc-secret-reader.yaml
```

後で許可リストにさらに Secret オブジェクトを追加するには、Role's `resourceNames リストに名前を追加して再適用します。

ステップ 3: Ingress が復旧したことを確認する

ロードバランサーコントローラーは、次のパス (通常は数分以内) で Ingress を再調整します。付与が機能すると、以前の secrets "oidc" is forbidden エラーが Ingress に表示されなくなり、ロードバランサーの処理が進みます。エラーが解消されない場合は、Role's `namespace、resourceNames、および RoleBinding サブジェクトを再確認します。

付与を削除する

> **Important**
>
> ALB Ingress が `alb.ingress.kubernetes.io/auth-type: oidc` を使用している限り、Role と RoleBinding をそのまま保持します。これらを削除すると、Ingress での OIDC 解決が再び機能しなくなります。

Ingress で OIDC の使用を停止する場合は、付与を削除します:

```
kubectl delete rolebinding eks-managed-oidc-secret-reader -n monitoring
kubectl delete role        eks-managed-oidc-secret-reader -n monitoring
```

関連リソース

- the section called “マネージドコントローラーに RBAC を付与する” – `eks:managed` グループの概念的な概要と、付与のスコープ設定方法。
- the section called “アクセスエントリ” – Amazon EKS アクセスエントリ。
- Kubernetes ドキュメントの「RBAC 認可を使用する」。

kubectl debug を使用して Kubernetes ノードの CIS コンプライアンスレポートを生成する

このトピックでは、`kubectl debug` コマンドを使用して Amazon EKS ノードの CIS (Center for Internet Security) コンプライアンスレポートを生成する方法について説明します。このコマンドを使用すると、Kubernetes ノードにデバッグコンテナを一時的に作成し、`apiclient` ツールを使用して CIS コンプライアンスチェックを実行できます。この `apiclient` ツールは、Bottlerocket OS (EKS Auto Mode ノードで使用する OS) に組み込まれています。

前提条件

開始する前に、以下を確認してください。

- `kubectl` が設定された Amazon EKS クラスターへのアクセス (バージョンは v1.32.0 以上である必要があります。確認するには `kubectl version` と入力します)。
- ノードをデバッグするための適切な IAM アクセス権限。

- デバッグオペレーションが許可される有効なプロファイル (例: sysadmin)。

kubectl とデバッグプロファイルの併用方法については、Kubernetes ドキュメントの「Debugging a Pod or Node while applying a profile」で詳しく確認できます。

手順

1. レポートを実行するノードの AWS インスタンス ID を決定します。クラスター内のノードを一覧表示するには、以下のコマンドを使用します。インスタンス ID は名前の列にあり、i- で始まります。

```
kubectl get nodes
```

```
NAME                  STATUS   ROLES    AGE   VERSION
i-0ea0ba0f8ef9ad609   Ready    <none>   62s   v1.30.10-eks-1a9dacd
```

2. <instance-id> を、クエリするノードのインスタンス ID に置き換えて、次のコマンドを実行します。

```
kubectl debug node/<instance-id> -it --profile=sysadmin --image=public.ecr.aws/
amazonlinux/amazonlinux:2023 -- bash -c "yum install -q -y util-linux-core; nsenter -
t 1 -m apiclient report cis --level 1 --format text"
```

このコマンドのコンポーネントは次のとおりです。

- kubectl debug node/<instance-id> — 指定した EC2 インスタンス ID でデバッグセッションを作成します。
- -it — TTY (コマンドラインシェル) を割り当てて、stdin を開いたままにし、インタラクティブに使用できるようにします。
- --profile=sysadmin — 指定した kubectl プロファイル (適切なアクセス権限が付与されている) を使用します。
- --image=public.ecr.aws/amazonlinux/amazonlinux:2023 — amazonlinux:2023 を、デバッグ用のコンテナイメージに使用します。
- bash -c "…" — bash シェルで次のコマンドを実行します。
  - yum install -q -y util-linux-core — 必要なユーティリティパッケージをサイレントインストールします。

- `nsenter -t 1 -m` — nsenter を実行して、ホストプロセス (PID 1) の名前空間に入ります。
- `apiclient report cis --level 1 --format text` — テキスト出力を使用して、レベル 1 で CIS コンプライアンスレポートを実行します。

3. レポートのテキスト出力を確認します。

出力の解釈

このコマンドにより、テキストベースのレポートを生成して、さまざまな CIS コントロールのコンプライアンスステータスを確認できます。以下のような情報が出力されます。

- 個々の CIS コントロール ID
- 各コントロールの説明
- 各チェックの合格、失敗、スキップのステータス
- コンプライアンス上の問題の詳細説明

Bottlerocket インスタンスで実行したレポートの出力例を次に示します。

```
Benchmark name:  CIS Bottlerocket Benchmark
Version:         v1.0.0
Reference:       https://www.cisecurity.org/benchmark/bottlerocket
Benchmark level: 1
Start time:      2025-04-11T01:40:39.055623436Z

[SKIP] 1.2.1     Ensure software update repositories are configured (Manual)
[PASS] 1.3.1     Ensure dm-verity is configured (Automatic)[PASS] 1.4.1     Ensure
 setuid programs do not create core dumps (Automatic)
[PASS] 1.4.2     Ensure address space layout randomization (ASLR) is enabled
 (Automatic)
[PASS] 1.4.3     Ensure unprivileged eBPF is disabled (Automatic)
[PASS] 1.5.1     Ensure SELinux is configured (Automatic)
[SKIP] 1.6       Ensure updates, patches, and additional security software are
 installed (Manual)
[PASS] 2.1.1.1   Ensure chrony is configured (Automatic)
[PASS] 3.2.5     Ensure broadcast ICMP requests are ignored (Automatic)
[PASS] 3.2.6     Ensure bogus ICMP responses are ignored (Automatic)
[PASS] 3.2.7     Ensure TCP SYN Cookies is enabled (Automatic)
[SKIP] 3.4.1.3   Ensure IPv4 outbound and established connections are configured
 (Manual)
```

```
[SKIP] 3.4.2.3   Ensure IPv6 outbound and established connections are configured
 (Manual)
[PASS] 4.1.1.1   Ensure journald is configured to write logs to persistent disk
 (Automatic)
[PASS] 4.1.2     Ensure permissions on journal files are configured (Automatic)

Passed:          11
Failed:          0
Skipped:         4
Total checks:    15
```

ベンチマークの詳細については、Center for Internet Security (CIS) の「Kubernetes Benchmark」を参照してください。

関連リソース

- 「Bottlerocket CIS Benchmark」(Bottlerocket OS ドキュメント)
- 「実行中のポッドのデバッグ」(Kubernetes ドキュメント)
- 「Kubernetes Benchmark」(Center for Internet Security (CIS))

EKS オートモードでカスタマーマネージド KMS キー使用した EBS ボリュームの暗号化を有効にする

カスタマーマネージド KMS キーを使用すると、EKS オートモードのインスタンスのエフェメラルルートボリュームを暗号化することができます。

Amazon EKS オートモードは、Kubernetes クラスターの暗号化された EBS ボリュームを管理するときに、サービスリンクロールを使用して他の AWS サービスにアクセス許可を委任します。このトピックでは、EKS オートモードの Amazon EBS 暗号化のカスタマーマネージドキーを指定するときに必要になる、キーポリシーの設定方法について解説します。

考慮事項:

- EKS オートモードでは、デフォルトの AWS マネージドキーを使用してアカウントにある暗号化されたボリュームを保護する場合、追加の承認は不要です。
- このトピックでは、EC2 インスタンスのルートボリュームである、エフェメラルボリュームを暗号化する方法について取り上げます。ワークロードに使用するデータボリュームの暗号化の方法については、「the section called “StorageClass を作成する”」を参照してください。

概要

以下の AWS KMS キーは、EKS オートモードでインスタンスを起動するときに、Amazon EBS ルートボリュームの暗号化に使用できます。

- AWS マネージドキー – Amazon EBS が作成、所有、管理するアカウント内の暗号化キーです。これは、新しいアカウントのデフォルトの暗号化キーです。
- カスタマーマネージド型キー – お客様が作成、所有、および管理するカスタム暗号化キー。

> **Note**
>
> KMS キーは対称である必要があります。Amazon EBS は非対称カスタマー管理キーをサポートしていません。

ステップ 1: キーポリシーを設定する

KMS キーには、カスタマーマネージドキーで暗号化された Amazon EBS ボリュームを使用してインスタンスを起動することを EKS オートモードに許可する、キーポリシーが必要です。

キーポリシーは次の構造で設定します。

> **Note**
>
> このポリシーには EKS オートモードのアクセス許可しか含まれていません。他の ID でキーを使用したり許可を管理したりする必要がある場合、このキーポリシーに追加のアクセス許可が必要になる可能性があります。

```
{
    "Version":"2012-10-17",
    "Id": "MyKeyPolicy",
    "Statement": [
        {
            "Sid": "Allow use of the key",
            "Effect": "Allow",
            "Principal": {
                "AWS": [
                    "arn:aws:iam::123456789012:role/ClusterServiceRole"
```

```
                ]
            },
            "Action": [
                "kms:Encrypt",
                "kms:Decrypt",
                "kms:ReEncrypt*",
                "kms:GenerateDataKey*",
                "kms:DescribeKey"
            ],
            "Resource": "*"
        },
        {
            "Sid": "Allow attachment of persistent resources",
            "Effect": "Allow",
            "Principal": {
                "AWS": [
                    "arn:aws:iam::123456789012:role/ClusterServiceRole"
                ]
            },
            "Action": [
                "kms:CreateGrant",
                "kms:ListGrants",
                "kms:RevokeGrant"
            ],
            "Resource": "*",
            "Condition": {
                "Bool": {
                    "kms:GrantIsForAWSResource": "true"
                }
            }
        }
    ]
}
```

<account-id> を、現在お使いの AWS アカウント ID に置き換えます。

キーポリシーを設定する場合:

- ClusterServiceRole には、暗号化のオペレーションに KMS キーを使用するために必要な、IAM アクセス許可が含まれている必要があります。
- kms:GrantIsForAWSResource 条件によって、AWS サービスに対してのみ許可を作成することができます。

ステップ 2: カスタマーマネージドキーを使用して NodeClass を設定する

キーポリシーを設定したら、EKS オートモード NodeClass 設定で KMS キーを参照します。

```
apiVersion: eks.amazonaws.com/v1
kind: NodeClass
metadata:
  name: my-node-class
spec:
  # Insert existing configuration

  ephemeralStorage:
    size: "80Gi"  # Range: 1-59000Gi or 1-64000G or 1-58Ti or 1-64T
    iops: 3000    # Range: 3000-16000
    throughput: 125  # Range: 125-1000

    # KMS key for encryption
    kmsKeyID: "arn:aws:kms:<region>:<account-id>:key/<key-id>"
```

プレースホルダーの値を実際の値にそれぞれ置き換えます。

- <region> は現在の AWS リージョンに
- <account-id> はご自身の AWS アカウント ID に
- <key-id> はご自身の KMS キー ID に

KMS キーは以下のいずれかの形式を使用して指定することができます。

- KMS キー ID: 1a2b3c4d-5e6f-1a2b-3c4d-5e6f1a2b3c4d
- KMS キー ARN: arn:aws:kms:us-west-2:111122223333:key/1a2b3c4d-5e6f-1a2b-3c4d-5e6f1a2b3c4d
- キーエイリアス名: alias/eks-auto-mode-key
- キーエイリアス ARN: arn:aws:kms:us-west-2:111122223333:alias/eks-auto-mode-key

kubectl を使用して NodeClass 設定を適用します。

```
kubectl apply -f nodeclass.yaml
```

関連リソース

- Amazon EKS のノードクラスを作成する
- 詳細については、「AWS Key Management Service デベロッパーガイド」を参照してください。
  - キーポリシーにおける AWS サービスのアクセス許可
  - キーポリシーを変更する
  - Grants in AWS KMS

ノードの高度なセキュリティ設定を構成する

このトピックでは、ノードクラスの advancedSecurity 仕様を使用して Amazon EKS Auto Mode ノードの高度なセキュリティ設定を構成する方法について説明します。

前提条件

開始する前に、以下を確認してください。

- Amazon EKS Auto Mode クラスター。詳細については、「the section called “クラスターを作成する”」を参照してください。
- kubectl がインストールされ、設定済みであること。詳細については、「セットアップ」を参照してください。
- ノードクラス設定について。詳細については、「the section called “ノードクラスの作成”」を参照してください。

高度なセキュリティ設定を構成する

ノードの高度なセキュリティ設定を構成するには、ノードクラス仕様の advancedSecurity フィールドを設定します。

```
apiVersion: eks.amazonaws.com/v1
kind: NodeClass
metadata:
  name: security-hardened
spec:
  role: MyNodeRole

  subnetSelectorTerms:
    - tags:
        Name: "private-subnet"
```

```
  securityGroupSelectorTerms:
    - tags:
        Name: "eks-cluster-sg"

  advancedSecurity:
    # Enable FIPS-compliant AMIs (US regions only)
    fips: true

    # Configure kernel lockdown mode
    kernelLockdown: "integrity"
```

この設定を適用します。

```
kubectl apply -f nodeclass.yaml
```

ノードプール設定でこのノードクラスを参照します。詳細については、「the section called “ノードプールの作成”」を参照してください。

フィールドの説明

- `fips` (ブール値、オプション): `true` に設定すると、FIPS 140-2 検証済み暗号化モジュールを含む AMI を使用してノードをプロビジョニングします。この設定では、FIPS 準拠の AMI が選択されます。お客様はコンプライアンス要件を管理する責任があります。詳細については、「AWS FIPS Compliance」を参照してください。デフォルト: `false`。
- `kernelLockdown` (文字列、オプション): カーネルロックダウンセキュリティモジュールモードを制御します。使用できる値:
  - `integrity`: カーネルメモリを上書きしたり、カーネルコードを変更するメソッドをブロックします。署名なしカーネルモジュールがロードされないようにします。
  - none: カーネルロックダウン保護を無効にします。

    詳細については、「Linux kernel lockdown」ドキュメントを参照してください。

考慮事項

- FIPS 準拠の AMI は、AWS 米国東部/西部、AWS GovCloud (米国)、AWS カナダ (中部/西部) の各リージョンで利用できます。詳細については、「AWS FIPS Compliance」を参照してください。
- `kernelLockdown: "integrity"` を使用する場合は、ワークロードが署名なしカーネルモジュールのロードやカーネルメモリの変更を必要としないことを確認してください。

関連リソース

- the section called “ノードクラスの作成” – 完全なノードクラス設定ガイド
- the section called “ノードプールの作成” – ノードプールの設定

EKS Auto Mode でのワークロードのスケジューリング

Amazon EKS Auto Mode ノードへのワークロードのスケジューリングを制御します。以下のトピックを参照して、EKS Auto Mode へのデプロイを指示し、重要なアドオンを実行してください。

トピック

- ワークロードが EKS Auto Mode ノードにデプロイされるかどうかを制御する
- ハードウェア専有インスタンスで重要なアドオンを実行する

ワークロードが EKS Auto Mode ノードにデプロイされるかどうかを制御する

EKS Auto Mode を使用して EKS クラスターでワークロードを実行する場合、特定のワークロードが EKS Auto Mode ノードで実行されるか、他のコンピューティングタイプで実行されるかを制御することが必要になる場合があります。このトピックでは、ノードセレクターとアフィニティルールを使用して、ワークロードを目的のコンピューティングインフラストラクチャでスケジュールする方法について説明します。

このトピックの例では、`eks.amazonaws.com/compute-type` ラベルを使用して EKS Auto Mode ノードへのワークロードのデプロイを要求または禁止する方法を示します。これは、EKS Auto Mode と、セルフマネージド Karpenter プロビジョナーや EKS マネージドノードグループなどの他のコンピューティングタイプの両方を実行している混合モードクラスターで特に便利です。

EKS Auto Mode ノードでは、ラベル `eks.amazonaws.com/compute-type` の値が `auto` に設定されています。このラベルを使用して、ワークロードが EKS Auto Mode によって管理されるノードにデプロイされるかどうかを制御できます。

ワークロードを EKS Auto Mode ノードにデプロイすることを要求する

> **Note**
>
> この `nodeSelector` 値は EKS Auto Mode では必須ではありません。この `nodeSelector` 値は、クラスターを混合モードで実行している場合にのみ関連し、ノードタイプは EKS

Auto Mode で管理されません。例えば、EKS マネージドノードグループを使用してクラスターに静的コンピューティング容量をデプロイし、EKS Auto Mode で動的計算能力を管理しているとします。

この nodeSelector をデプロイまたは他のワークロードに追加して、Kubernetes が EKS Auto Mode ノードにスケジュールするように要求できます。

```
apiVersion: apps/v1
kind: Deployment
spec:
  template:
    spec:
      nodeSelector:
        eks.amazonaws.com/compute-type: auto
```

ワークロードを EKS Auto Mode ノードにデプロイしないことを要求する

この nodeAffinity をデプロイまたは他のワークロードに追加して、Kubernetes が EKS Auto Mode ノードにスケジュールしないように要求できます。

```
  affinity:
    nodeAffinity:
      requiredDuringSchedulingIgnoredDuringExecution:
        nodeSelectorTerms:
        - matchExpressions:
          - key: eks.amazonaws.com/compute-type
            operator: NotIn
            values:
            - auto
```

特定の NodePool をターゲットにする

Amazon EKS Auto Mode クラスターには、複数の NodePool を含めることができます。ワークロードを特定の NodePool のノードでのみ実行したい場合があります。これを実現するには、ワークロードを次のいずれかのラベルに一致させます。

- NodePool で定義するカスタムラベル。NodePool の spec.template.metadata.labels の下にラベルを追加すると、Amazon EKS Auto Mode はそれらのラベルを、その NodePool がプロビジョニングするすべてのノードに適用します。カスタムラベルが推奨されるアプローチです。

カスタムラベルは、ワークロードを特定の NodePool 名ではなく、`workload-class: gpu-inference` などの機能に結び付けます。

- 既知の **`karpenter.sh/nodepool`** ラベル。Amazon EKS Auto Mode は、プロビジョニングするすべてのノードにこのラベルを適用し、NodePool の名前を値として使用します。カスタムラベルを定義しておらず、名前で NodePool をターゲットにしたい場合は、このラベルを使用します。

カスタム NodePool ラベルをターゲットにする

まず、NodePool でラベルを定義します。

```
apiVersion: karpenter.sh/v1
kind: NodePool
metadata:
  name: gpu
spec:
  template:
    metadata:
      labels:
        workload-class: gpu-inference
```

NodePool の設定の詳細については、「the section called “ノードプールの作成”」を参照してください。

次に、ワークロードからそのラベルに一致させます。

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: inference
spec:
  template:
    spec:
      nodeSelector:
        workload-class: gpu-inference
```

このデプロイのポッドは、gpu NodePool がプロビジョニングするノードにのみスケジュールされます。この `nodeSelector` を持たないワークロードは、引き続きデフォルトの NodePool で実行されます。

名前で NodePool をターゲットにする

> **NodePool 名の変更はスケジューリングに影響する**
>
> 名前でワークロードを固定すると、ターゲットの NodePool の名前を変更または削除した場合に、ポッドが Pending 状態のままになる可能性があります。単一の NodePool 定義よりも長く存続すると予想されるワークロードには、カスタム機能ラベルを使用してください。

```
apiVersion: apps/v1
kind: Deployment
spec:
  template:
    spec:
      nodeSelector:
        karpenter.sh/nodepool: <your-nodepool-name>
```

Karpenter がポッドを NodePool に照合する方法の詳細については、Karpenter ドキュメントの「スケジューリング」を参照してください。

ハードウェア専有インスタンスで重要なアドオンを実行する

このトピックでは、EKS Auto Mode が `system` ノードプールにワークロードをスケジュールできるように、`CriticalAddonsOnly` の容認を持つワークロードをデプロイする方法について説明します。

EKS Auto Mode の組み込みの `system` ノードプールは、専用インスタンス上で重要なアドオンを実行するために設計されています。この分離により、重要なコンポーネントに専用のリソースが備わり、一般的なワークロードから分離されることが保証され、クラスター全体の安定性とパフォーマンスが向上します。

このガイドでは、`CriticalAddonsOnly` の容認と適切なノードセレクターを利用して、`system` ノードプールにアドオンをデプロイする方法を説明します。これらのステップに従うことで、EKS Auto Mode の特殊なノードプール構造が提供する分離とリソース割り当ての利点を活用して、重要なアプリケーションを専用の `system` ノードにスケジュールできるようになります。

EKS Auto Mode には、`general-purpose` と `system` の 2 つのノードプールが組み込まれています。詳細については、「the section called “組み込みノードプールを確認する”」を参照してください。

system ノードプールの目的は、重要なアドオンを異なるノードに分離することです。system ノードプールによってプロビジョニングされたノードには、CriticalAddonsOnly という Kubernetes のテイントがあります。Kubernetes は、対応する容認がある場合のみ、これらのノードにポッドをスケジュールします。詳細については、Kubernetes ドキュメントの「テイントと容認」を参照してください。

前提条件

- EKS Auto Mode クラスターで組み込みの system ノードプールが有効になっていること。詳細については、「the section called “組み込みノードプールを確認する”」を参照してください。
- kubectl がインストールされ、設定済みであること。詳細については、「セットアップ」を参照してください。

手順

以下の yaml の例を確認してください。以下の設定に留意してください。

- nodeSelector — これにより、ワークロードが組み込みの system ノードプールに関連付けられます。このノードプールは AWS API で有効にする必要があります。詳細については、「the section called “組み込みノードプールを確認する”」を参照してください。
- tolerations — この容認は、system ノードプール内のノードの CriticalAddonsOnly テイントを克服します。

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: sample-app
spec:
  replicas: 3
  selector:
    matchLabels:
      app: sample-app
  template:
    metadata:
      labels:
        app: sample-app
    spec:
      nodeSelector:
        karpenter.sh/nodepool: system
```

```
      tolerations:
      - key: "CriticalAddonsOnly"
        operator: "Exists"
      containers:
      - name: app
        image: nginx:latest
        resources:
          requests:
            cpu: "500m"
            memory: "512Mi"
```

system ノードプールで実行するようにワークロードを更新するには、以下の操作を行う必要があります。

1. 既存のワークロードを更新して、上記で説明した以下の設定を追加します。
   - nodeSelector
   - tolerations
2. kubectl apply を使用して、更新されたワークロードをクラスターにデプロイする

ワークロードを更新すると、専用ノードで実行されるようになります。

EKS Auto Mode のコスト最適化

Amazon EKS Auto Mode クラスターのコンピューティングコストを最適化します。以下のトピックを参照して、統合について理解し、コスト管理パターンを適用してください。

トピック

- EKS オートモードでのコスト最適化

EKS オートモードでのコスト最適化

EKS オートモードでは、統合、ビンパッキング、適切なサイジングを通じてクラスターのコンピューティングコストが継続的に最適化されます。ただし、特定のワークロード設定では、これらの最適化を妨げられる場合があります。このトピックでは、コスト最適化の仕組み、コスト最適化を妨げるもの、コスト効率を維持するためにクラスターを設定する方法について説明します。

EKS オートモードによるコストの最適化方法

EKS オートモードでは、次のメカニズムを通じてコンピューティングコストが削減されます。

- ビンパッキング — ポッドをノードにスケジューリングする場合、EKS オートモードによって集約リソースリクエストに厳密に一致するインスタンスタイプが選択され、未使用のキャパシティが最小限に抑えられます。
- 統合 — EKS オートモードによって実行中のノードが定期的に評価され、ワークロードをより少ないインスタンスまたはより低コストなインスタンスで実行できる場合、ノードが置き換えられるか削除されます。
- 適切なサイジング — ワークロードがスケールダウンすると、EKS オートモードによってポッドが小さなノードに統合され、使用率の低いインスタンスが終了されます。

これらの最適化は、手動操作なしで継続的に実行されます。ただし、特定のポッドアノテーションと NodePool 設定により、統合の有効化が妨げられる場合があります。

組み込みノードプールとコストガードレール

組み込みの `general-purpose` ノードプールと `system` ノードプールでは、既にいくつかのコスト保護のためのデフォルトが適用されています。

- C、M、R に制限されたインスタンスファミリー — 高速 (P、G、Inf、Trn) またはエキゾチックのインスタンスタイプは許可されません。
- オンデマンドキャパシティのみ — スポットインスタンスは使用されません。これにより、中断によるチャーンは回避されますが、スポット節約も行われません。
- 世代 5 以降 — 古い、コスト効率の低いインスタンス世代は除外されます。

組み込みノードプールのみを使用している場合は、既にこれらのガードレールによるメリットを得ています。インスタンスファミリーの除外とインスタンスサイズの制約に関するこのトピックのガイダンスは、これらの制限を継承しないカスタム NodePool を作成するときに最も関連します。

ただし、組み込みノードプールでも、以下のセクションは適用されます。

- the section called “統合を防ぐもの” — `do-not-disrupt` 注釈と制限付き PDB により、ノードをプロビジョニングした NodePool に関係なく、統合がブロックされます。
- the section called “NodePool の制限をコスト上限として使用する” — 組み込みノードプールにはリソースの `limits` が設定されていません。ワークロードを大幅にスケールできる場合は、無制限の組み込みプールに依存するのではなく、制限付きのカスタム NodePool を作成することを検討してください。
- the section called “ノードのライフサイクルとコスト” — ノード置換の重複は、組み込みプールによってプロビジョニングされるノードを含むすべてのノードに適用されます。

ガードレール	組み込みノードプール	カスタム NodePool
高速インスタンスの除外	強制	設定が必要
インスタンスサイズの制限	未設定	設定が必要
リソースの `limits` (CPU/メモリの上限)	未設定	設定が必要
オンデマンドのみ	強制	(スポット/オンデマンド) を選択
統合防止 (`do-not-disrupt` /PDB)	お客様の責任	お客様の責任

統合を防ぐもの

ノードの中断がワークロードの可用性要件に違反すると EKS オートモードで判断されると、統合が防がれます。次の設定では、統合が妨げられます。

`do-not-disrupt` 注釈

`karpenter.sh/do-not-disrupt` 注釈により、ノードで注釈付きポッドが実行されている限り、ノードを保持するように EKS オートモードに指示されます。これにより、ノードが十分に活用されていなくても、ノードが統合、置換、または終了されるのを防ぎきます。

```
metadata:
  annotations:
    karpenter.sh/do-not-disrupt: "true"
```

> **Important**
>
> コストへの影響: ポッドに `do-not-disrupt` 注釈がある場合、そのポッドが実行されるノードは統合から除外されます。つまり、次のようになります。
>
> - ノードは、実際の使用率に関係なく、現在のインスタンスサイズで引き続き実行されます。
> - ワークロードの需要が減少しても、そのノードの vCPU とメモリの使用量が高止まりすることがあります。

- 多くのノードにまたがる複数のポッドにこの注釈がある場合、クラスター全体の統合が大幅に削減され、コストが持続的に高くなります。

do-not-disrupt 注釈は可用性メカニズムです。コストは考慮されません。チェックポイントなしで長時間実行されるバッチジョブやステートフルプロセスなど、実行中の中断によってデータ損失や大幅な再作業が発生するワークロードにのみ使用してください。

考慮すべき代替案:

- ポッド中断の予算 (PDB) — 完全に防がずに中断率を制御するには、PDB を使用します。PDB を使用すると、最小限の数のレプリカを引き続き使用しながら、統合を続行できます。
- 存続期間の短いワークロード — CI/CD ランナーとビルドエージェントの場合、中断を許可し、do-not-disrupt を使用せずに CI システムの組み込み再試行ロジックを利用します。
- 時間制限付き注釈 — 重要な操作の期間中のみ do-not-disrupt を適用し、操作の完了時にプログラムによって解除します。

ポッド中断の予算 (PDB)

maxUnavailable: 0 または minAvailable を現在のレプリカ数以上に設定する PDB により、影響を受けるポッドのすべての統合が効果的に防がれます。PDB を確認して、一度に少なくとも 1 つのポッドを中断できることを確認します。

NodePool の制限をコスト上限として使用する

NodePool の limits により、NodePool がプロビジョニングできるコンピューティングリソースの合計に絶対的な上限が設定されます。この制限に達すると、EKS オートモードによってその NodePool の新しいノードの起動が停止されます。これは、ポッドが保留中であっても行われます。

特に、無制限のスケーリングが適切でない非本番稼働、テスト、またはバーストのワークロードを処理する NodePool に対して、コストガードレールとして limits を使用します。

```
apiVersion: karpenter.sh/v1
kind: NodePool
metadata:
  name: ci-runners
spec:
  template:
```

```
    spec:
      nodeClassRef:
        group: eks.amazonaws.com
        kind: NodeClass
        name: default
      requirements:
        - key: "eks.amazonaws.com/instance-category"
          operator: In
          values: ["c", "m"]
  limits:
    cpu: "500"
    memory: 1000Gi
```

この例では、ci-runners NodePool はプロビジョニングするすべてのノードで合計 500 vCPU または 1000 GiB のメモリを超えることはできません。この制限を超えるポッドは、キャパシティが解放されるまで Pending 状態のままになります。

> **Tip**
>
> 予想される最大バーストサイズにノード置換のバッファを加えた値に基づいて limits を設定します。NodePool の使用率を定期的に確認し、ワークロードパターンの変化に応じて制限を調整します。

コスト管理のためにインスタンスファミリーを除外する

デフォルトの EKS オートモードでは、幅広いインスタンスタイプから選択することでスケジューリングの柔軟性が最大化されます。特殊なハードウェアを必要としないワークロードの場合は、コストの高いインスタンスタイプが起動されないようにインスタンスファミリーを制限します。

高速インスタンスを除外する

GPU またはアクセラレーターリソースをリクエストしないワークロードの場合は、高速インスタンスファミリーを NodePool から除外します。これにより、キャパシティの制約中に高速インスタンスが選択されるシナリオを回避できます。

```
spec:
  template:
    spec:
      requirements:
        - key: "eks.amazonaws.com/instance-category"
```

```
        operator: In
        values: ["c", "m", "r"]
```

コンピューティング最適化、汎用、メモリ最適化のカテゴリのみを指定することで、高速 (P、G、Inf、Trn) やその他の特殊なインスタンスファミリーを選択から除外できます。

インスタンスの選択がキャパシティの制約とどのように相互作用するか

EKS オートモードでは、通常のインスタンスの選択中は高速およびエキゾチックのインスタンスタイプの優先順位が下がります。ただし、持続的な起動障害が発生すると、EKS オートモードでは残りの使用可能なインスタンスタイプから起動され、ワークロードの可用性が優先されます。これは例えば、すべての優先インスタンスタイプで EC2 のサービスクォータが一時的に使い果たされた場合に発生します。

このフォールバック動作を防ぐには、ワークロードに必要なインスタンスカテゴリのみに NodePool 要件を明示的に制限します。優先タイプが使用できず、NodePool 設定で他のタイプが許可されていない場合、ポッドはコストの高いインスタンスにスケジュールされずに、Pending 状態のままになります。

インスタンスサイズを制約する

インスタンスファミリーの制限に加えて、NodePool 内の最大インスタンスサイズを制限できます。インスタンスサイズの制約により、統合できない単一ノードからのコストエクスポージャーが制限されます。例えば、do-not-disrupt 注釈によってブロックされたノードは、ワークロードが小さい場合でも縮小できません。

eks.amazonaws.com/instance-cpu ラベルを使用して、NodePool 要件の最大インスタンスサイズを制限します。

```
requirements:
  - key: "eks.amazonaws.com/instance-cpu"
    operator: Lte
    values: ["32"]
```

この設定により、EKS オートモードではこの NodePool で 32 vCPU を超えるインスタンスを起動できなくなります。

既存のクラスターで最適化の機会を特定するには、実行中の最大のインスタンスを確認します。大きなノードの統合が一貫して防がれている場合、そのアイドルキャパシティのノードあたりのコストは比例して高くなります。

バーストワークロードに推奨されるパターン

CI/CD パイプライン、バッチジョブ、エフェメラルランナーでは、コスト効率を維持するために特定の設定を必要とするバーストとアイドルのパターンが作成されます。

バーストワークロードに推奨されるデフォルト

設定	推奨事項
`do-not-disrupt`	CI/CD ランナーには使用しないでください。代わりに、CI システムの再試行とキューのメカニズムを利用します。
NodePool の `limits`	予想される最大同時実行数にノード置換の重複のバッファを加えた値に基づいて CPU/メモリの上限を設定します。
インスタンスカテゴリ	c および m ファミリーに制限します。GPU 以外のワークロードの高速インスタンスファミリー (P、G、Inf、Trn) を除外します。
インスタンスサイズ	中程度のサイズ (4 ~ 32 個の vCPU など) に制約して、統合が防がれる単一ノードからのコストエクスポージャーを制限することを検討してください。
統合タイミング	デフォルトの `consolidateAfter` 設定を使用します。ランナーの完了後に、バーストキャパシティをオンラインに保つ長い遅延を設定しないでください。
キャパシティータイプ	耐障害性ランナーにはスポットインスタンスを使用します。実行中に状態を保持するビルドエージェントでは、オンデマンドと組み合わせます。

例: CI ランナー NodePool

```
apiVersion: karpenter.sh/v1
kind: NodePool
metadata:
  name: ci-runners
```

```
spec:
  template:
    spec:
      nodeClassRef:
        group: eks.amazonaws.com
        kind: NodeClass
        name: default
      requirements:
        - key: "eks.amazonaws.com/instance-category"
          operator: In
          values: ["c", "m"]
        - key: "eks.amazonaws.com/instance-cpu"
          operator: Lte
          values: ["32"]
        - key: "karpenter.sh/capacity-type"
          operator: In
          values: ["spot", "on-demand"]
  limits:
    cpu: "500"
    memory: 1000Gi
  disruption:
    consolidationPolicy: WhenEmptyOrUnderutilized
    consolidateAfter: 30s
```

この設定では、次のようになります。

- コスト効率の高いインスタンスファミリーに制限
- NodePool の合計キャパシティは 500 vCPU が上限
- 積極的な統合を許可 (ポッドが削除されてから 30 秒後)
- スポットとオンデマンド両方のキャパシティを許可

ノードのライフサイクルとコスト

EKS オートモードでは、ノードが目的の仕様から逸脱した場合 (新しいオートモード AMI のリリース後など)、またはノードの有効期限が近づいた場合、グレースフルな中断によってノードが置き換えられます。グレースフルな置き換え中に、以下が行われます。

- 新しい代替ノードが起動し、準備が整います。
- ポッド中断の予算を考慮して、ポッドが古いノードからドレインされます。
- 短期間、古いノードと代替ノードの両方が同時に実行されます。

大規模なノードまたは多数のノードがあるクラスターの場合、この重複により定期的にコストが増加する可能性があります。影響を最小限に抑えるには、以下が必要です。

- 中断の予算の確認 — 中断の予算でタイムリーなドレインが許可されていることを確認します。制限付きの予算では、古いノードと新しいノードの両方が実行されている重複期間が長くなります。
- 適切なサイズのインスタンス — インスタンスが小さいほど、重複期間の絶対コストが削減されます。
- ノードの最大有効期間の短縮 — 有効期限の値を短くすると (7 日間など)、置き換えイベントの頻度は高くなりますが、置き換えイベントは小規模になります。これにより、コストが集中せずに、時間的に均等に分散されます。

ノードのライフサイクルの詳細については、「Amazon EKS Auto Mode マネージドインスタンスについての説明」を参照してください。

アップグレードとライフサイクルの管理

Amazon EKS Auto Mode クラスターの継続的なライフサイクルを管理します。以下のトピックを参照して、Kubernetes バージョンを更新し、EKS Auto Mode を無効にします。

トピック

- EKS Auto Mode クラスターの Kubernetes バージョンを更新する
- EKS Auto Mode を無効にする

EKS Auto Mode クラスターの Kubernetes バージョンを更新する

このトピックでは、Auto Mode クラスターの Kubernetes バージョンを更新する方法について説明します。Auto Mode は、ポッド中断の予算を通じてワークロードの可用性を維持しながら、ノード交換によるコントロールプレーンの更新を調整することで、バージョン更新プロセスを簡素化します。

Auto Mode クラスターのアップグレード時に、従来手動更新が必要だった多くのコンポーネントがサービスの一部として管理されるようになりました。アップグレードプロセスの自動化された側面とユーザー担当部分を理解することで、クラスターのバージョンをスムーズに移行できます。

EKS Auto Mode による更新についての説明

コントロールプレーンのアップグレードを開始したら、EKS Auto Mode によってクラスター内のノードがアップグレードされます。ノードの有効期限が切れると、EKS Auto Mode によって新しい

ノードに置き換えられます。新しいノードには、対応する新しい Kubernetes バージョンがあります。EKS Auto Mode は、ノードのアップグレード時にポッド中断の予算を監視します。

さらに、次のようなコンポーネントは更新する必要がありません。

- Amazon VPC CNI
- AWS ロードバランサーコントローラー
- CoreDNS
- `kube-proxy`
- Karpenter
- AWS EBS CSI ドライバー

EKS Auto Mode は、これらのコンポーネントをサービス機能に置き換えます。

以下の更新は引き続きユーザーが行います。

- クラスターにデプロイされたアプリケーションとワークロード
- セルフマネージドアドオンおよびコントローラー
- Amazon EKS アドオン
  - 「the section called “アドオンを更新する”」ではその方法を説明しています。

「クラスターのアップグレードのベストプラクティス」を参照してください

クラスターの更新を開始する

クラスターの更新を開始するには、「the section called “Kubernetes バージョンを更新する”」を参照してください。

EKS Auto Mode を無効にする

既存の EKS クラスターで EKS Auto Mode を無効にすることができます。これは破壊的な操作です。

- EKS はEKS Auto Mode によって運用されているすべての EC2 インスタンスを終了します。
- EKS はEKS Auto Mode によって運用されているすべてのロードバランサーを削除してください。
- EKS はEKS Auto Mode によってプロビジョニングされた EBS ボリュームを削除しません。

EKS Auto Mode は作成するリソースを完全に管理するように設計されています。手動で操作すると、EKS Auto Mode が無効になっているときにそれらのリソースを完全にクリーンアップできなくなる可能性があります。例えば、外部のセキュリティグループルールからマネージドセキュリティグループを参照し、クラスターの EKS Auto Mode を無効にする前にその参照を削除するのを忘れた場合、マネージドセキュリティグループはリークします (削除されません)。以下のステップではリークされたセキュリティグループが発生した場合に削除する方法について説明します。

EKS Auto Mode を無効にする (AWS コンソール)

1. AWS マネジメントコンソールでクラスターの概要ページを開きます。
2. [EKS Auto Mode] で [管理] を選択します
3. [EKS Auto Mode] を [off] に切り替えます。

このプロセスの終了時にマネージドセキュリティグループが削除されていない場合は、セキュリティグループを削除するの手順に従って手動で削除できます。

EKS Auto Mode を無効にする (AWS CLI)

既存のクラスターで EKS Auto Mode を無効にするには次のコマンドを使用します。

aws CLI をインストールし、EKS クラスターを管理するのに十分なアクセス許可が付与された状態でログインする必要があります。詳細については、セットアップ を参照してください。

> **Note**
>
> コンピューティング、ブロックストレージ、およびロードバランシング機能はすべて、同じリクエストで有効または無効にする必要があります。

```
aws eks update-cluster-config \
 --name $CLUSTER_NAME \
 --compute-config enabled=false \
 --kubernetes-network-config '{"elasticLoadBalancing":{"enabled": false}}' \
 --storage-config '{"blockStorage":{"enabled": false}}'
```

EKS Auto Mode を無効にした後、リークした EKS Auto Mode セキュリティグループ の削除に失敗したかどうかは次のように確認できます。

```
aws ec2 describe-security-groups \
```

```
    --filters Name=tag:eks:eks-cluster-name,Values=<cluster-Name> Name=tag-
key,Values=ingress.eks.amazonaws.com/resource,service.eks.amazonaws.com/resource --
query "SecurityGroups[*].[GroupName]"
```

セキュリティグループを削除するには:

```
aws ec2 delete-security-group --group-name=<sg-name>
```

EKS Auto Mode の仕組みの説明

この章では、Amazon EKS Auto Mode クラスターのコンポーネントの仕組みについて説明します。

トピック

- Amazon EKS Auto Mode マネージドインスタンスについての説明
- EKS Auto Mode での ID とアクセスについての説明
- EKS 自動モードI での VPC ネットワーキングとロードバランシングの説明

Amazon EKS Auto Mode マネージドインスタンスについての説明

このトピックでは Amazon EKS Auto Mode が EKS クラスター内の Amazon EC2 インスタンスを管理する方法について説明します。EKS Auto Mode を有効にすると、クラスターのコンピューティングリソースが EKS によって自動的にプロビジョニングおよび管理され、クラスター内のノードとして機能する EC2 インスタンスの操作方法が変わります。

Amazon EKS Auto Mode がインスタンスを管理する方法を理解することはワークロードのデプロイ戦略と運用手順を計画するのに不可欠です。従来の EC2 インスタンスやマネージドノードグループとは異なり、これらのインスタンスは特定のタイプのアクセスとカスタマイズを制限しながら、EKS が多くの運用面に対して責任を負う別のライフサイクルモデルに従います。

Amazon EKS Auto Mode は新しい EC2 インスタンスを作成するためのルーチンタスクを自動化し、それらをノードとして EKS クラスターにアタッチします。EKS Auto Mode はワークロードが既存のノードに収まらない状況を検出し、新しい EC2 インスタンスを作成します。

Amazon EKS Auto Mode はEC2 インスタンスの作成、削除、パッチ適用を担います。インスタンスにデプロイされたコンテナとポッドについてはユーザーの責任となります。

EKS Auto Mode によって作成された EC2 インスタンスは他の EC2 インスタンスとは異なり、マネージドインスタンスです。このマネージドインスタンスは EKS によって所有され、制限が厳しく

なっています。EKS Auto Mode によって管理されているインスタンスに直接アクセスしたりソフトウェアをインストールしたりすることはできません。

AWS では EKS Auto Mode またはセルフマネージド Karpenter の実行をお勧めします。移行中でも高度な設定においてでもインストールが可能です。両方がインストールされている場合はワークロードが Karpenter または EKS Auto Mode に関連付けられるようにノードプールを設定します。

詳細については、Amazon EC2 ユーザーガイドの「Amazon EC2 管理されたインスタンス」を参照してください。

比較表

スタンダード EC2 インスタンス	EKS Auto Mode マネージドインスタンス
インスタンスのパッチ適用と更新はユーザーの責任となります。	AWS で自動的にインスタンスのパッチ適用と更新が行われます。
EKS はインスタンス上のソフトウェアに関与しません。	EKS は`kubelet`、コンテナランタイム、オペレーティングシステムなど、インスタンス上の特定のソフトウェアに関与します。
EC2 API を使用して EC2 インスタンスを削除できます。	EKS によりアカウントにデプロイされたインスタンスの数が決定されます。ワークロードを削除すると、EKS によりアカウント内のインスタンスの数が減ります。
SSH を使用して EC2 インスタンスにアクセスできます。	ポッドとコンテナをマネージドインスタンスにデプロイできます。
ユーザーがオペレーティングシステムとイメージ (AMI) を決定します。	AWS によりオペレーティングシステムとイメージが決定されます。
Windows または Ubuntu の機能に依存するワークロードをデプロイできます。	Linux に基づいてコンテナをデプロイできますが、特定の OS 依存関係はありません。
ユーザーが起動するインスタンスタイプとファミリーを決定します。	AWS により起動するインスタンスタイプとファミリーが決定されます。ノードプールを使用して、EKS Auto Mode が選択するインスタンスタイプを制限できます。

次の機能はマネージドインスタンスとスタンダード EC2 インスタンスの両方で機能します:

- インスタンスストレージは、ワークロードのエフェメラルストレージとして使用できます。

AMI サポート

EKS Auto Mode では、AWS がコンピューティングノードに使用されるイメージ (AMI) を決定します。AWS は新しい EKS Auto Mode AMI バージョンのロールアウトをモニタリングします。AMI バージョンに関連するワークロードの問題が発生した場合は、サポートケースを作成します。詳細については、AWS サポートユーザーガイドの「サポートケースの作成およびケース管理」を参照してください。

EKS の場合、通常、CVE とセキュリティ修正を含む新しい AMI が毎週リリースされます。

マネージドリソースの可視性

> **Note**
>
> 2026 年 4 月 22 日以降、EKS Auto Mode によって作成された新しい Amazon EC2 マネージドインスタンスと関連リソース (EC2 起動テンプレート、EBS ボリューム、ネットワークインターフェイス (ENI) など) は、EC2 コンソールビューと `describe` API リストオペレーションからデフォルトで非表示になります。その日付より前にアカウントに既に存在していたマネージドリソースは表示され続けます。この動作は、マネージドリソースの可視性設定を使用して変更できます。

Amazon EC2 には、マネージドリソースが EC2 コンソールビューと API リストオペレーション (`DescribeInstances` など) に表示されるかどうかを制御するマネージドリソースの可視性設定が用意されています。

マネージドリソースが非表示になっている場合、EKS Auto Mode マネージドインスタンスとその関連リソース (EBS ボリューム、起動テンプレート、ネットワークインターフェイス) は EC2 コンソールまたは `describe` API レスポンスに表示されません。これにより、ガバナンスダッシュボードを簡素化し、オブザーバビリティツールのノイズを減らし、マネージドリソースを顧客の設定ミスとしてフラグ付けするクラウドセキュリティ体制管理 (CSPM) スキャナーの誤検出を防ぐことができます。

可視性設定は、すべての IAM プリンシパルにアカウント全体で適用されます。リソースタイプまたはリソースを作成したサービスによって、マネージドリソースを選択的に表示または非表示にすることはできません。非表示のリソースは完全に動作し、請求可能です。

可視性設定は、Amazon EC2 コンソールまたは AWS CLI を使用していつでも変更できます。詳細については、Amazon EC2 ユーザーガイドの「マネージドリソースの可視性設定」を参照してください。

> **Note**
>
> マネージドリソースが EC2 コンソールビューとリスト API から非表示になっていても、以下を通じて EKS Auto Mode インスタンスを表示できます。
>
> - クラスターの [コンピューティング] タブの下にある Amazon EKS コンソール。
> - Kubernetes API (`kubectl get nodes` など)。
> - インスタンス ID によるダイレクト EC2 API クエリ (`describe-instances --instance-ids i-0123456789abcdef0` など)。
> - `include-managed-resources` パラメータを使用した `DescribeInstances` API。
> - マネージドリソースの可視性設定を変更した後の EC2 コンソール。

EKS Auto Mode でサポートされているインスタンスリファレンス

EKS Auto Mode は、サポートされているタイプの、最小サイズ要件を満たしたインスタンスのみを作成します。

EKS Auto Mode では、次のインスタンスタイプがサポートされています:

ファミリー	インスタンスタイプ
コンピューティング最適化 (C)	c9g、c9gd、c8a、c8g、c8gb、c8gd、c8gn、c8i、c8i-flex、c8ib、c8id、c7a、c7g、c7gd、c7gn、c7i、c7i-flex、c6a、c6g、c6gd、c6gn、c6i、c6id、c6in、c5、c5a、c5ad、c5d、c5n、c4
汎用 (M)	m9g、m9gd、m8a、m8azn、m8g、m8gb、m8gd、m8gn、m8i、m8i-flex、m8id、m7a、m7g、m7gd、m7i、m7i-flex、m6a、m6g、m6gd、m6i、m6id、m6idn、m6in、m5、m5a、m5ad、m5d、m5dn、m5n、m5zn、m4

ファミリー	インスタンスタイプ
メモリ最適化 (R)	r9g、r8a、r8g、r8gb、r8gd、r8gn、r8i、r8i-flex、r8id、r7a、r7g、r7gd、r7i、r7iz、r6a、r6g、r6gd、r6i、r6id、r6idn、r6in、r5、r5a、r5ad、r5b、r5d、r5dn、r5n、r4
バースト可能 (T)	t4g、t3、t3a、t2
ハイメモリ (Z/X)	z1d、x8aedz、x8g、x8i、x2gd
ストレージ最適化 (I/D)	i8g、i8ge、i7i、i7ie、i4g、is4gen、im4gn、i4i、i3、i3en、d3、d3en
高速コンピューティング (Trn/P/Inf/G)	trn2、trn1、trn1n、p6-b200、p6-b300、p5、p5e、p5en、p4d、p4de、p3、p3dn、inf2、inf1、g7e、g6、gr6、g6e、g5、g5g、g4ad、g4dn
ハイパフォーマンスコンピューティング (X2/HPC)	x2idn、x2iedn、x2iezn、hpc8a

さらに、EKS Auto Mode は、次の要件を満たす EC2 インスタンスのみを作成します。

- 複数の CPU
- インスタンスサイズがナノ、マイクロ、スモールではない

詳細については、「Amazon EC2 インスタンスタイプの命名規則」を参照してください。

インスタンスメタデータサービス

- EKS Auto Mode では、デフォルトでホップ制限が 1 の IMDSv2 が実施され、AWS セキュリティのベストプラクティスに準拠します。
- このデフォルト設定は Auto Mode では変更できません。
- 通常は IMDS アクセスが必要なアドオンの場合、IMDS ルックアップを避けるため、インストール中にパラメータ (AWS リージョンなど) を指定します。詳細については、「the section called “カスタマイズ可能なフィールド”」を参照してください。

- Auto Mode で実行するときにポッドが IMDS アクセスを絶対に必要とする場合は、ポッドを `hostNetwork: true` で実行するように設定する必要があります。ポッドがインスタンスメタデータサービスに直接アクセスできるようになります。
- ポッドにインスタンスメタデータへのアクセスを付与する場合、セキュリティ上の影響などを配慮してください。

Amazon EC2 インスタンスメタデータサービス (IMDS) に関する詳細については、Amazon EC2 ユーザーガイドの「インスタンスメタデータサービスのオプション設定」を参照してください。

考慮事項

- NodeClass で設定されたエフェメラルストレージがインスタンスの NVMe ローカルストレージよりも小さい場合、EKS Auto Mode は次のアクションを自動的に実行することで、手動設定の必要性を排除します。
  - より小さい (20 GiB) Amazon EBS データボリュームを使用し、コストを削減します。
  - エフェメラルデータ用に NVMe ローカルストレージをフォーマットして設定します。複数の NVMe ドライブがある場合、これには RAID 0 アレイの設定が含まれます。
- `ephemeralStorage.size` がローカルの NVMe 容量と等しいかそれより大きい場合、以下のアクションが実行されます。
  - Auto Mode では、容量の少ない EBS ボリュームはスキップされます。
  - NVMe ドライブは、ワークロードに直接公開されます。
- Amazon EKS Auto Mode は、以下の AWS Fault Injection Service アクションをサポートしていません。
  - `ec2:RebootInstances`
  - `ec2:SendSpotInstanceInterruptions`
  - `ec2:StartInstances`
  - `ec2:StopInstances`
  - `ec2:TerminateInstances`
  - `ec2:PauseVolumeIO`
- Amazon EKS Auto Mode は、AWS Fault Injection Service EKS ポッドアクションをサポートしています。詳細については、AWS レジリエンスハブユーザーガイドの「フォールトインジェクションサービスの実験管理」と「AWS FIS aws:eks:pod アクション」を参照してください。
- EKS Auto Mode のノードに `Neuron Device Plugin` をインストールする必要はありません。

クラスター内に他のタイプのノードがある場合、Auto Mode のノードで実行されないように Neuron Device プラグインを設定する必要があります。詳細については、「the section called “デプロイを制御する”」を参照してください。

EKS Auto Mode での ID とアクセスについての説明

このトピックでは、EKS Auto Mode を使用するのに必要な Identity and Access Management (IAM) ロールとアクセス許可について説明します。EKS Auto Mode は、クラスター IAM ロールとノード IAM ロールという 2 つのプライマリ IAM ロールを使用します。これらのロールは、EKS Pod Identity および EKS アクセスエントリと連携して動作し、EKS クラスターの包括的なアクセス管理を提供します。

EKS Auto Mode を設定するときには、AWS サービスがクラスターリソースとやり取りできるようにする特定のアクセス許可を持つこれらの IAM ロールを設定する必要があります。これには、コンピューティングリソース、ストレージボリューム、ロードバランサー、ネットワーキングコンポーネントを管理するためのアクセス許可が含まれます。これらのロール設定を理解することは、適切なクラスターオペレーションおよびセキュリティに不可欠です。

EKS Auto Mode では、AWS IAM ロールは EKS アクセスエントリを介して Kubernetes アクセス許可に自動的にマッピングされるため、`aws-auth` ConfigMaps またはカスタムバインディングを手動で設定する必要はありません。新しい Auto Mode クラスターを作成すると、EKS はアクセスエントリを使用して対応する Kubernetes アクセス許可を自動的に作成し、AWS サービスとクラスターコンポーネントが AWS と Kubernetes の両方の認証システムで適切なアクセスレベルになるようにします。この自動統合により、設定の複雑さが軽減され、EKS クラスターの管理時によく発生するアクセス許可関連の問題を防ぐことができます。

クラスター IAM ロール

クラスター IAM ロールは、Kubernetes クラスターのアクセス許可を管理するために Amazon EKS が使用する AWS Identity and Access Management (IAM) ロールです。このロールは、クラスターに代わって他の AWS サービスとやり取りするのに必要なアクセス許可を Amazon EKS に付与し、EKS アクセスエントリにより Kubernetes アクセス許可が付与されて自動的に設定されます。

- AWS IAM ポリシーをこのロールにアタッチする必要があります。
- EKS Auto Mode は、EKS アクセスエントリを使用して、このロールに Kubernetes アクセス許可を自動的にアタッチします。

- EKS Auto Mode を使用する場合、AWS では AWS アカウントごとに 1 つのクラスター IAM ロールを作成することをお勧めします。
- AWS では、このロールの名前を とすることをお勧めします。`AmazonEKSAutoClusterRole`
- このロールには、EBS ボリューム、Elastic Load Balancer、EC2 インスタンスなどのリソースを管理するために、複数の AWS サービスに対するアクセス許可が必要です。
- このロールの推奨設定には、EKS Auto Mode のさまざまな機能に関連する複数の AWS マネージド IAM ポリシーが含まれます。
  - `AmazonEKSComputePolicy`
  - `AmazonEKSBlockStoragePolicyV2`
  - `AmazonEKSLoadBalancingPolicy`
  - `AmazonEKSNetworkingPolicy`
  - `AmazonEKSClusterPolicy`

クラスター IAM ロールと AWS マネージド IAM ポリシーの詳細については、次を参照してください。

- the section called “ AWS マネージドポリシー”
- the section called “クラスター IAM ロール”

Kubernetes アクセスの詳細については、次を参照してください。

- the section called “アクセスポリシーを確認する”

ノード の IAM ロール

ノード IAM ロールは、Kubernetes クラスター内のワーカーノードのアクセス許可を管理するために Amazon EKS が使用する AWS Identity and Access Management (IAM) ロールです。このロールは、Kubernetes ノードとして実行されている EC2 インスタンスに、AWS サービスおよびリソースを操作するのに必要なアクセス許可を付与し、EKS アクセスエントリにより Kubernetes RBAC アクセス許可が付与されて自動的に設定されます。

- AWS IAM ポリシーをこのロールにアタッチする必要があります。
- EKS Auto Mode は、EKS アクセスエントリを使用して、このロールに Kubernetes RBAC アクセス許可を自動的にアタッチします。
- AWS では、このロールの名前を とすることをお勧めします。`AmazonEKSAutoNodeRole`

- EKS Auto Mode を使用する場合、AWS では AWS アカウントごとに 1 つのノード IAM ロールを作成することをお勧めします。
- このロールには制限されたアクセス許可があります。その主なアクセス許可には、Pod Identity ロールの引き受け、ECR からのイメージのプルなどがあります。
- AWS では、以下の AWS マネージド IAM ポリシーをお勧めします。
  - `AmazonEKSWorkerNodeMinimalPolicy`
  - `AmazonEC2ContainerRegistryPullOnly`

ノード IAM ロールと AWS マネージド IAM ポリシーの詳細については、次を参照してください:

- the section called “ AWS マネージドポリシー”
- the section called “ノード の IAM ロール”

Kubernetes アクセスの詳細については、次を参照してください。

- the section called “アクセスポリシーを確認する”

サービスにリンクされたロール

Amazon EKS は、特定のオペレーションにサービスリンクロール (SLR) を使用します。サービスにリンクされたロールは Amazon EKS に直接リンクされた一意のタイプの IAM ロールです。サービスにリンクされたロールはAmazon EKS で事前定義されています。このロールにはサービスがユーザーに代わって他の AWS のサービスを呼び出すために必要な、すべてのアクセス許可が付与されています。

AWS では SLR が自動的に作成および設定されます。先に関連リソースを削除した後にのみ、SLR を削除できます。これにより、リソースへのアクセス許可を不用意に削除することが防止され、Amazon EKS リソースが保護されます。

SLR ポリシーは、EC2 リソース (インスタンス、ネットワークインターフェイス、セキュリティグループ)、ELB リソース (ロードバランサー、ターゲットグループ)、CloudWatch 機能 (ログ記録とメトリクス)、「eks」というプレフィックスが付いた IAM ロールなどのコアインフラストラクチャコンポーネントを監視および削除する Amazon EKS アクセス許可を付与します。また、VPC/ホストゾーンの関連付けによるプライベートエンドポイントネットワーキングを有効にし、EKS タグ付きリソースの EventBridge モニタリングとクリーンアップのためのアクセス許可を備えています。

詳細については、以下を参照してください。

- the section called “AWS マネージドポリシー: AmazonEKSServiceRolePolicy”
- the section called “Amazon EKS でのサービスにリンクされた役割のアクセス許可”

アクセスポリシーのリファレンス

EKS Auto Mode で使用される Kubernetes アクセス許可の詳細については、「the section called “アクセスポリシーを確認する”」を参照してください。

EKS 自動モードI での VPC ネットワーキングとロードバランシングの説明

このトピックではEKS 自動モードI で Virtual Private クラウド (VPC ネットワーキングおよびロードバランシング機能を設定する方法について説明します。EKS Auto Mode はほとんどのネットワークコンポーネントを自動的に管理しますが、NodeClass リソースとロードバランサーの注釈を使用してクラスターのネットワーキング設定の特定の側面をカスタマイズできます。

EKS 自動モードI を使用すると、AWS でクラスターの VPC Container Network Interface (CNI 設定とロードバランサーのプロビジョニングが管理されます。NodeClass オブジェクトを定義し、サービスリソースとイングレスリソースに特定の注釈を適用することで、EKS Auto Mode が提供する自動運用モデルを維持しながら、ネットワーキング動作に関与することができます。

ネットワーク機能

EKS Auto Mode には、ノードとポッドのネットワークを処理する新しいネットワーク機能があります。この機能は、NodeClass Kubernetes オブジェクトを作成することで設定できます。

以前の AWS VPC CNI の設定オプションは EKS Auto Mode には適用されません。

`NodeClass` を使用したネットワークの設定

EKS Auto Mode の NodeClass リソースを使用すると、ネットワーク機能の特定の側面をカスタマイズできます。NodeClass を使用すると、セキュリティグループ選択の指定、VPC サブネット間のノード配置の制御、SNAT ポリシーの設定、ネットワークポリシーの設定、ネットワークイベントのログ記録の有効化を行うことができます。このアプローチはEKS 自動モードI の自動運用モデルを維持しながら、ネットワークのカスタマイズで柔軟性が得られます。

NodeClass を使用して以下のことができます。

- ノードのセキュリティグループを選択する
- VPC サブネットにノードを配置する方法を制御する

- ノード SNAT ポリシーを `random` または `disabled` に設定する
- 以下を含む Kubernetes ネットワークポリシーを有効にする。
  - ネットワークポリシーを「デフォルトで拒否」または「デフォルトで許可」に設定する
  - ファイルへのネットワークイベントのログ記録を有効にする。
- 異なるサブネットにポッドをアタッチして、ノードトラフィックからポッドトラフィックを分離する。

Amazon EKS ノードクラス の作成方法を参照してください。

考慮事項

EKS 自動モードI は以下をサポートしています：

- EKS ネットワークポリシー。
- Kubernetes ポッドの `HostPort` および `HostNetwork` オプション。
- パブリックサブネットまたはプライベートサブネットのノードとポッド。
- ノードでの DNS クエリキャッシュ。
- EFA ワークロードの静的ネットワークインターフェイス設定。詳細については、「the section called “静的ネットワークインターフェイスの設定”」を参照してください。

EKS 自動モードI は以下をサポートしていません：

- ポッドあたりのセキュリティグループ (SGPP)。Auto Mode で個別のセキュリティグループをポッドトラフィックに適用するには、代わりに `NodeClass` で `podSecurityGroupSelectorTerms` を使用します。詳細については、「the section called “ポッドのサブネットとセキュリティグループを分離する”」を参照してください。
- `ENIConfig` のカスタムネットワーキング。ポッドを複数のサブネットに配置することも、the section called “ポッドのサブネットとセキュリティグループを分離する” を使用してノードトラフィックから排他的に分離することもできます。
- ウォーム IP、ウォームプレフィックス、ウォーム ENI 設定。
- 最小 IP ターゲット設定。
- オープンソース AWS VPC CNI でサポートされているその他の設定。
- conntrack タイマーのカスタマイズなどのネットワークポリシー設定 (デフォルトは 300 秒)。
- CloudWatch へのネットワークイベントログのエクスポート。

- 代替 CNI プラグイン (Cilium や Calico など) と代替ネットワークポリシープラグイン。Amazon EKS Auto Mode は、前述の組み込みネットワーキング機能を使用します。詳細については、「Amazon EKS Auto Mode に関する考慮事項」を参照してください。

ネットワークリソース管理

EKS Auto Mode では、NodeClass リソースのネットワーク設定をモニタリングすることで、プレフィックス、IP アドレス指定、ネットワークインターフェイス管理を処理します。このサービスでは、次のいくつかの主要なオペレーションが自動的に実行されます。

プレフィックスの委任

EKS 自動モードは、デフォルトでポッドネットワークにプレフィックス委任 (/28 プレフィックス) を使用し、IP リソースのウォームプールを事前定義されたとおりに維持し、スケジュールされたポッドの数に基づいてスケールします。ポッドサブネットの断片化が検出されると、自動モードはセカンダリ IP アドレス (/32) をプロビジョニングします。このデフォルトのポッドネットワークアルゴリズムにより、自動モードはインスタンスタイプごとにサポートされている ENI と IP の数に基づいてノードあたりの最大ポッド数を計算します (断片化の最悪のケースを想定)。インスタンスタイプあたりの ENI と IP の最大数の詳細については、「EC2 ユーザーガイド」の「ネットワークインターフェイスあたりの最大 IP アドレス数」を参照してください。新世代 (Nitro v6 以降) のインスタンスファミリーでは通常、インスタンスタイプあたりの ENI と IP の数が増加しており、自動モードはそれに応じて最大ポッド数の計算を調整します。

IPv6 クラスターでは、プレフィックス委任のみが使用されます。Auto Mode は同じ IPv4 ベースの計算を適用し、110 未満の場合は計算値を使用して、ノードごとに最大ポッド数を 110 に制限します。

クールダウン管理

このサービスでは、使用されなくなったプレフィックスまたはセカンダリ IPv4 アドレスのクールダウンプールを実装します。クールダウン期間が終了すると、これらのリソースは VPC に解放されます。ただし、こうしたリソースは、クールダウン期間にポッドで再利用されると、クールダウンプールから復元されます。

IPv6 のサポート

IPv6 クラスターの EKS Auto Mode では、プライマリネットワークインターフェイスのノードごとに `/80` IPv6 プレフィックスをプロビジョニングします。`podSubnetSelectorTerms` を使用する場合、プレフィックスは代わりにポッドサブネットのセカンダリネットワークインターフェイスに割り当てられます。

また、すべてのネットワークインターフェイスの適切な管理とガベージコレクションを確実に行います。

負荷分散

EKS 自動モードI でプロビジョニングされた AWS 弾性ロードバランサー はサービスリソースとイングレスリソースの注釈を使用して設定します。

詳細については「the section called “ingress class の作成”」または「the section called “サービスの作成”」を参照してください。

EKS 自動モードI によるロードバランシングに関する考慮事項

- デフォルトのターゲットモードは IP モードであり、インスタンスモードではありません。
- EKS 自動モードI はネットワークロードバランサー のセキュリティグループモードのみをサポートします。
- AWS はセルフマネージド AWS ロードバランサーコントローラー から EKS 自動モードI による管理へのロードバランサーの移行をサポートしていません。
- `TargetGroupBinding` 仕様の `networking.ingress.ipBlock` フィールドはサポートされていません。
- ワーカーノードがカスタムセキュリティグループ (`eks-cluster-sg-` 命名パターンではない) を使用している場合、クラスターロールには追加の IAM アクセス許可が必要です。デフォルトの EKS マネージドポリシーではEKS による `eks-cluster-sg-` という名前のセキュリティグループの変更のみを許可します。カスタムセキュリティグループを変更する許可がないと、EKS は ALB/NLB トラフィックがポッドに到達できるようにするのに必要な進入ルールを追加できません。

CoreDNS に関する考慮事項

EKS 自動モードは、クラスター内で DNS を解決する場合に、従来の CoreDNS デプロイを使用しません。代わりに、自動モードノードが、各ノードでシステムサービスとして直接実行される CoreDNS を使用します。従来のクラスターを自動モードに移行する場合、ワークロードを自動モードノードに移動したら、クラスターから CoreDNS デプロイを削除できます。

> **Important**
>
> 自動モードノードと非自動モードノードの両方でクラスターを維持する場合は、CoreDNS デプロイを保持する必要があります。非自動モードノードは、自動モードが備えるノードレ

ベルの DNS サービスにアクセスできないため、従来の CoreDNS ポッドがないと DNS を解決できません。

EKS Auto Mode のオブザーバビリティ

この章では、Amazon EKS Auto Mode クラスターのオブザーバビリティオプションについて説明します。

トピック

- EKS Auto の AWS マネージドコンポーネントログにアクセスする

EKS Auto の AWS マネージドコンポーネントログにアクセスする

EKS Auto Mode から AWS マネージドコンポーネントログにアクセスして、クラスターオペレーションをより詳細に観察できます。EKS Auto Mode は、次のソースのログをサポートします。

- コンピューティングの自動スケーリング – Karpenter
- ブロックストレージ – EBS CSI
- ロードバランシング – AWS ロードバランサーコントローラー
- ポッドネットワーク – VPC CNI IP アドレス管理

ログは、任意の 送信先 に配信できます。

EKS Auto Mode クラスターを作成する際に、コントロールプレーンのログ (API server、Audit、Authenticator、Controller manager、Scheduler) を有効にするオプションがあります。EKS Auto マネージドコンポーネントログ (コンピューティング、ブロックストレージ、ロードバランシング、IPAM など) には、ログ配信による個別の設定が必要です。

ログ配信の設定

EKS Auto Mode クラスターの AWS マネージドコンポーネントログ配信を設定するには、Amazon CloudWatch Logs API を使用します。詳細な設定手順については、「Amazon CloudWatch Logs ユーザーガイド」の「Enabling logging from AWS services」を参照してください。各 Auto Mode 機能は、個々の CloudWatch Vended Logs 配信ソースとして設定でき、アクセスするログを選択できます。

EKS Auto Mode では次のログタイプがサポートされています。

- AUTO_MODE_COMPUTE_LOGS
- AUTO_MODE_BLOCK_STORAGE_LOGS
- AUTO_MODE_LOAD_BALANCING_LOGS
- AUTO_MODE_IPAM_LOGS

Amazon CloudWatch API の使用

ログ記録の設定には次の 3 つのステップが必要です。

1. CloudWatch PutDeliverySource API を使用して機能の配信ソースを作成する
2. PutDeliveryDestination を使用して送信先を作成する
3. CreateDelivery を使用して送信元と送信先を接続する配信を作成する

CloudWatch PutDeliveryDestination API の deliveryDestinationConfiguration オブジェクトを使用して、Auto Mode のログの送信先の詳細を設定できます。CloudWatch ロググループ、S3 バケット、または Amazon Data Firehose 配信ストリームのいずれかの ARN を取得します。

複数の配信を作成することで、1 つの Auto Mode 機能 (配信ソース) から複数の送信先にログを送信するよう設定できます。あるいは複数の配信の作成により、複数の配信ソースから同じ送信先にログを送信するように設定することもできます。

IAM アクセス許可

選択した送信先によっては、ログ配信が成功するように CloudWatch ロググループ、S3 バケット、および Amazon Data Firehose の IAM ポリシーまたはロールを設定する必要がある場合があります。さらに、AWS アカウント間でログを送受信する場合は、PutDeliveryDestinationPolicy API を使用して、送信先への配信を許可する IAM ポリシーを設定する必要があります。詳細については、「CloudWatch Vended Logs permissions」のドキュメントを参照してください。

ログの表示

ログ配信が設定されると、ログは指定された送信先に配信されます。ログにアクセスする方法は、選択した送信先タイプによって異なります。

- CloudWatch Logs – CloudWatch Logs コンソールでログを表示する、AWS CLI コマンドを使用する、または CloudWatch Logs Insights でクエリを実行する
- Amazon S3 – S3 コンソール、AWS CLI、または Amazon Athena などの分析ツールを使用して、S3 バケット内のオブジェクトとしてログにアクセスする

- Amazon Data Firehose – ログは、設定された Firehose ターゲット (S3、OpenSearch Service、Redshift など) にストリーミングされます。

料金

CloudWatch Vended Logs の料金は、選択した送信先に基づいてログの配信とストレージに適用されます。CloudWatch Vended Logs は、標準の CloudWatch Logs と比較して、組み込みの AWS 認証と認可により、信頼性が高く安全なログ配信を低価格で実現します。詳細については、「CloudWatch 料金ページのVended Logs セクション」を参照してください。

関連リソース

- Amazon EKS コントロールプレーンのログ記録
- CloudWatch Logs API リファレンスの PutDeliverySource API
- CloudWatch Logs API リファレンスの PutDeliveryDestination API
- CloudWatch Logs API リファレンスの CreateDelivery API

EKS Auto Mode のトラブルシューティング

EKS Auto Mode ではAWS が AWS アカウント内の EC2 インスタンスに対してより多くの責任を負います。EKS はノード上のコンテナランタイム、ノード上のオペレーティングシステム、および特定のコントローラーに対して責任を負います。これにはブロックストレージコントローラー、ロードバランシングコントローラー、コンピューティングコントローラーなどがあります。

ノードのトラブルシューティングにはAWS API および Kubernetes API を使用する必要があります。次のようにできます:

- Kubernetes `NodeDiagnostic` リソースを使用し、the section called “ノードモニタリングエージェント”を使用することでノードログを取得します。詳細な手順については、the section called “ノードログの取得” を参照してください。
- Kubernetes `NodeDiagnostic` リソースを使用し、ノード上のネットワークトラフィックをキャプチャします。詳細な手順については、the section called “ネットワークトラフィックをキャプチャする” を参照してください。
- AWS EC2 CLI コマンド `get-console-output` を使用して、ノードからコンソール出力を取得します。詳細な手順については、the section called “AWS EC2 CLI を使用して EC2 マネージドインスタンスからコンソール出力を取得する” を参照してください。

- Kubernetes デバッグコンテナを使用してノードログを取得します。詳細な手順については、the section called “デバッグコンテナと kubectl CLI を使用してノードログを取得する” を参照してください。

> **Note**
>
> EKS Auto Mode は EC2 マネージドインスタンスを使用します。SSH による場合を含め、EC2 マネージドインスタンスに直接アクセスすることはできません。

EKS Auto Mode コンポーネントに固有の解決策がある、次のような問題が発生する可能性があります。

- Auto Mode ノードにスケジュールされていない Pod が Pending 状態でスタックする。解決策については、「the section called “Auto Mode ノードにスケジュールできない Pod のトラブルシューティング”」を参照してください。
- EC2 マネージドインスタンスが Kubernetes ノードとしてクラスターに参加しない。解決策については、「the section called “クラスターに参加しないノードのトラブルシューティング”」を参照してください。
- EKS Auto Mode に含まれているコントローラーを使用する NodePools、PersistentVolumes、Services に関するエラーと問題。解決策については、「the section called “Auto Mode に含まれているコントローラーのトラブルシューティング”」を参照してください。
- 強化された Pod のセキュリティにより、Pod 間でボリュームを共有できない。解決策については、「the section called “Pod 間でボリュームを共有する”」を参照してください。

EKS Auto Mode コンポーネントのトラブルシューティングには、次の方法を使用できます。

- the section called “AWS EC2 CLI を使用して EC2 マネージドインスタンスからコンソール出力を取得する”
- the section called “デバッグコンテナと kubectl CLI を使用してノードログを取得する”
- the section called “AWS コンソールで EKS Auto Mode に関連付けられたリソースを表示する”
- the section called “AWS アカウントの IAM エラーを表示します”
- the section called “VPC Reachability Analyzer でノード接続の問題を検出する”

ノードモニタリングエージェント

EKS Auto Mode には Amazon EKS ノードモニタリングエージェントが備わっています。このエージェントを使用して、ノードに関するトラブルシューティングとデバッグの情報を表示できます。ノードモニタリングエージェントはKubernetes `events` とノードの `conditions` を発行します。詳細については、the section called “ノードのヘルスと修復” を参照してください。

AWS EC2 CLI を使用して EC2 マネージドインスタンスからコンソール出力を取得する

この手順はブートタイムまたはカーネルレベルの問題のトラブルシューティングに役立ちます。

まず、ワークロードに関連付けられたインスタンスの EC2 インスタンス ID を決定する必要があります。次に、AWS CLI を使用してコンソール出力を取得します。

1. `kubectl` がインストールされ、クラスターに接続されていることを確認します
2. (任意 Kubernetes デプロイの名前を使用して、関連するポッドを一覧表示します。

```
kubectl get pods -l app=<deployment-name>
```

3. Kubernetes ポッドの名前を使用して、関連付けられたノードの EC2 インスタンス ID を決定します。

```
kubectl get pod <pod-name> -o wide
```

4. EC2 インスタンス ID を使用してコンソール出力を取得します。

```
aws ec2 get-console-output --instance-id <instance id> --latest --output text
```

デバッグコンテナと **`kubectl`** CLI を使用してノードログを取得する

EKS Auto Mode ノードからログを取得するために推奨される方法は、`NodeDiagnostic` リソースを使用することです。手順については、「the section called “ノードログの取得”」を参照してください。

ただし、`kubectl debug node` コマンドを使用してインスタンスからログをライブストリーミングすることができます。このコマンドにより、デバッグするノードで新しい Pod が起動されて、インタラクティブに使用することができます。

1. デバッグコンテナを起動します。次のコマンドは、ノードのインスタンス ID に i-0123456789012345 6 を使用し、インタラクティブな使用のために -it で tty を割り当てて stdin をアタッチし、kubeconfig ファイルの sysadmin プロファイルを使用します。

```
kubectl debug node/i-0123456789012345 6 -it --profile=sysadmin --image=public.ecr.aws/
amazonlinux/amazonlinux:2023
```

出力例は次のとおりです。

```
Creating debugging pod node-debugger-i-0123456789012345 6-nxb9c with container
 debugger on node i-0123456789012345 6.
If you don't see a command prompt, try pressing enter.
bash-5.2#
```

2. シェルから、nsenter コマンドを提供する util-linux-core をインストールできるようになりました。nsenter を使用してホスト上の PID 1 (init) のマウント名前空間を入力し、journalctl コマンドを実行して kubelet からログをストリーミングします。

```
yum install -y util-linux-core
nsenter -t 1 -m journalctl -f -u kubelet
```

セキュリティのため、Amazon Linux コンテナイメージはデフォルトでは多くのバイナリをインストールしません。yum whatprovides コマンドを使用して、特定のバイナリを提供するためにインストールする必要があるパッケージを識別できます。

```
yum whatprovides ps
```

```
Last metadata expiration check: 0:03:36 ago on Thu Jan 16 14:49:17 2025.
procps-ng-3.3.17-1.amzn2023.0.2.x86_64 : System and process monitoring utilities
Repo        : @System
Matched from:
Filename    : /usr/bin/ps
Provide    : /bin/ps

procps-ng-3.3.17-1.amzn2023.0.2.x86_64 : System and process monitoring utilities
Repo        : amazonlinux
Matched from:
Filename    : /usr/bin/ps
Provide    : /bin/ps
```

AWS コンソールで EKS Auto Mode に関連付けられたリソースを表示する

AWS コンソールを使用して、EKS Auto Mode クラスターに関連付けられているリソースのステータスを表示できます。

- EBS ボリューム
  - タグキー `eks:eks-cluster-name` を検索して EKS Auto Mode ボリュームを表示します
- ロードバランサー
  - タグキー `eks:eks-cluster-name` を検索して EKS Auto Mode ロードバランサーを表示します
- EC2 インスタンス

  EKS Auto Mode インスタンスは EC2 マネージドインスタンスです。2026 年 4 月 22 日以降、新しいマネージドインスタンスと関連リソースは、EC2 コンソールビューと API リストオペレーション (`DescribeInstances` など) からデフォルトで非表示になります。EC2 コンソールに Auto Mode インスタンスが表示されない場合は、マネージドリソースの可視性設定を確認してください。

  1. Amazon EC2 コンソールの https://console.aws.amazon.com/ec2/ を開いてください。
  2. ナビゲーションペインで、ダッシュボードを選択してください。
  3. アカウント属性カードの設定で、マネージドリソースの可視性を選択します。
  4. [管理] を選択し、マネージドインスタンスの可視性トグルを選択します。
  5. [Save changes] (変更の保存) をクリックします。

  可視性を有効にすると、タグキー `eks:eks-cluster-name` を検索して EKS Auto Mode インスタンスを表示できます。

  または、以下を通じて Auto Mode ノードを表示できます。

  - クラスターの [コンピューティング] タブの下にある Amazon EKS コンソール。
  - 設定済みの Kubernetes クライアントからの `kubectl get nodes`。
  - インスタンス ID または `include-managed-resources` パラメータを使用したダイレクト EC2 API クエリ。

  詳細については、「Amazon EC2 ユーザーガイド」の「マネージドリソースの可視性設定」を参照してください。

AWS アカウントの IAM エラーを表示します

1. CloudTrail コンソールに移動します。
2. 左側のナビゲーションペインで [イベント履歴] を選択します。
3. イベント履歴テーブルの右上にある設定 (歯車) アイコンを選択し、[エラーコード] 列を有効にします。
4. [エラーコード] 列で、`AccessDenied`、`UnauthorizedOperation`、`InvalidClientTokenId` などのアクセス許可関連のエラーをレビューします。リストを絞り込むには、[イベントソース] (`ec2.amazonaws.com` や `eks.amazonaws.com` など) と時間範囲でフィルタリングします。

より大きな時間枠でエラーコードによるより正確なフィルタリングを行うには、CloudTrail Lake イベントデータストアをクエリするか、Athena テーブルを作成します (イベント履歴ページの上部にある [Lake でのクエリ] と [Athena テーブルの作成] オプション)。

EKS クラスターに関連するエラーを探します。エラーメッセージを使用して、EKS アクセスエントリ、クラスター IAM ロール、またはノード IAM ロールを更新します。EKS Auto Mode に対するアクセス許可が付与された状態で、これらのロールを新しいポリシーにアタッチすることが必要になる場合があります。

Auto Mode ノードにスケジュールできない Pod のトラブルシューティング

ポッドが `Pending` 状態のままになり、Auto Mode ノードにスケジュールされない場合は、ポッドまたはデプロイマニフェストに `nodeSelector` が含まれているかどうかを確認します。`nodeSelector` が存在する場合は、EKS Auto Mode によって作成されたノードでスケジュールされる `eks.amazonaws.com/compute-type: auto` が使用されていることを確認します。EKS Auto Mode で使用されるノードラベルの詳細については、「the section called “デプロイを制御する”」を参照してください。

クラスターに参加しないノードのトラブルシューティング

EKS Auto Mode は、クラスターエンドポイントやクラスター認証局 (CA) などの、クラスターに参加するための正しい情報を使用して新しい EC2 インスタンスを自動的に設定します。しかし、それでもこれらのインスタンスがノードとして EKS クラスターに参加できない場合があります。次のコマンドを実行して、クラスターに参加しなかったインスタンスを特定します。

1. `kubectl get nodeclaim` を実行して、NodeClaims が `Ready = False` であるかどうかを確認します。

```
kubectl get nodeclaim
```

2. `kubectl describe nodeclaim <node_claim>` を実行して、[ステータス] でノードがクラスターに参加できない問題を見つけます。

```
kubectl describe nodeclaim <node_claim>
```

一般的なエラーメッセージ:

`Error getting launch template configs`

デフォルトのクラスター IAM ロールのアクセス許可を使用して `NodeClass` でカスタムタグを設定する場合、このエラーが表示されることがあります。the section called “ID とアクセス” を参照してください。

`Error creating fleet`

EC2 API から `RunInstances` コールを呼び出すと、認可の問題が発生する可能性があります。AWS CloudTrail でエラーを確認し、必要な IAM アクセス許可については、「the section called “Amazon EKS 自動モードI クラスター IAM ロール”」を参照してください。

`VPC Reachability Analyzer` でノード接続の問題を検出する

> **Note**
>
> VPC Reachability Analyzer によって実行される分析ごとに課金されます。料金の詳細については、「Amazon VPC の料金」を参照してください。

インスタンスがクラスターに参加しない理由の 1 つは、API サーバーに到達できないネットワーク接続の問題です。この問題を診断するために、VPC Reachability Analyzer を使用して、クラスターに参加できないノードと API サーバー間の接続の分析を実行することができます。次の 2 つの情報が必要になります。

- クラスターに参加できないノードのインスタンス ID
- Kubernetes API サーバーエンドポイントの IP アドレス

インスタンス ID を取得するには、クラスターにワークロードを作成して、EKS Auto Mode で EC2 インスタンスを起動する必要があります。これにより、インスタンス ID を持つ NodeClaim オブジェクトもクラスター内に作成されます。kubectl get nodeclaim -o yaml を実行して、クラスター内のすべての NodeClaims を出力します。各 NodeClaim にはインスタンス ID がフィールドとして含まれ、providerID にも再度含まれています。

```
kubectl get nodeclaim -o yaml
```

出力例は次のとおりです。

```
    nodeName: i-01234567890123456
    providerID: aws:///us-west-2a/i-01234567890123456
```

Kubernetes API サーバーエンドポイントは、kubectl get endpoint kubernetes -o yaml を実行して判断できます。このアドレスは addresses フィールドにあります。

```
kubectl get endpoints kubernetes -o yaml
```

出力例は次のとおりです。

```
apiVersion: v1
kind: Endpoints
metadata:
  name: kubernetes
  namespace: default
subsets:
- addresses:
  - ip: 10.0.143.233
  - ip: 10.0.152.17
  ports:
  - name: https
    port: 443
    protocol: TCP
```

これら 2 つの情報を使用して分析を実行できます。まず、AWS マネジメントコンソールで VPC Reachability Analyzer に移動します。

1. [パスの作成と分析] を選択します
2. 分析の名前を指定します (例: 「Node Join Failure」)
3. [ソースタイプ] には [インスタンス] を選択します

4. 障害が発生したノードのインスタンス ID を [ソース] として入力します
5. [パスの送信先] に [IP アドレス] を選択します
6. API サーバーの IP アドレスの 1 つを [送信先アドレス] として入力します
7. [追加のパケットヘッダーの設定セクション] を展開します
8. [送信先ポート] に 443 と入力します
9. まだ選択されていない場合は、[プロトコル] を [TCP] として選択します
10. [パスの作成と分析] を選択します
11. 分析の完了には数分かかることがあります。分析結果で到達可能性の失敗が示された場合は、ネットワークパスのどこで障害が生じたかが示され、問題を解決することができます。

Pod 間でボリュームを共有する

EKS Auto Mode ノードは、SELinux の強制モードで設定され、同じノードで実行されている Pod 間の分離を強化します。SELinux を有効にすると、特権のないほとんどのポッドに、独自のマルチカテゴリセキュリティ (MCS) ラベルが自動的に適用されます。この MCS ラベルは Pod ごとに一意であり、1 つの Pod のプロセスで他の Pod またはホストのプロセスを操作できないように設計されています。ラベル付き Pod がルートとして実行され、ホストファイルシステムにアクセスできる場合でも、ファイルの操作、ホストでの機密システムの呼び出し、コンテナランタイムへのアクセス、kubelet のシークレットキーマテリアルの取得を行うことはできません。

このため、Pod 間でデータを共有しようとすると問題が発生する可能性があります。例えば、アクセスモードが `ReadWriteOnce` の `PersistentVolumeClaim` でも、複数の Pod がボリュームに同時にアクセスすることはできません。

Pod 間でこの共有を有効にするには、Pod の `seLinuxOptions` を使用して、これらの Pod に同じ MCS ラベルを設定します。この例では、Pod に 3 つのカテゴリ (`c123,c456,c789`) を割り当てます。2 つのカテゴリが割り当てられるだけであるため、これによりノード上の Pod に自動的に割り当てられたカテゴリとの競合が発生することはありません。

```
securityContext:
  seLinuxOptions:
    level: "s0:c123,c456,c789"
```

コントロールプレーンログで Karpenter イベントを表示する

コントロールプレーンログが有効になっている EKS クラスターの場合、ログをクエリすることで Karpenter のアクションと意思決定プロセスに関するインサイトを得ることができます。これは、

ノードのプロビジョニング、スケーリング、終了に関連する EKS Auto Mode の問題のトラブルシューティングに特に役立ちます。Karpenter 関連のイベントを表示するには、次の CloudWatch Logs Insights クエリを使用します。

```
fields @timestamp, @message
| filter @logStream like /kube-apiserver-audit/
| filter @message like 'DisruptionBlocked'
or @message like 'DisruptionLaunching'
or @message like 'DisruptionTerminating'
or @message like 'DisruptionWaitingReadiness'
or @message like 'Unconsolidatable'
or @message like 'FailedScheduling'
or @message like 'NoCompatibleInstanceTypes'
or @message like 'NodeRepairBlocked'
or @message like 'Disrupted'
or @message like 'Evicted'
or @message like 'FailedDraining'
or @message like 'TerminationGracePeriodExpiring'
or @message like 'TerminationFailed'
or @message like 'FailedConsistencyCheck'
or @message like 'InsufficientCapacityError'
or @message like 'UnregisteredTaintMissing'
or @message like 'NodeClassNotReady'
| sort @timestamp desc
```

このクエリは、kube-apiserver 監査ログ内の特定の Karpenter 関連イベントをフィルタリングします。イベントには、さまざまな中断状態、スケジュールの失敗、容量の問題、ノード関連の問題が含まれます。これらのログを分析することで、以下をよりよく理解できます。

- Karpenter が特定のアクションを実行する理由。
- ノードの適切なプロビジョニング、スケーリング、または終了を妨げる問題。
- インスタンスタイプで発生する可能性のある容量または互換性の問題。
- 中断、削除、終了などのノードのライフサイクルイベント。

このクエリを使用するには:

1. CloudWatch コンソールに移動する
2. 左のナビゲーションペインから、[Logs Insights] を選択する
3. EKS クラスターのコントロールプレーンログのロググループを選択する

4. クエリエディタにそのクエリを貼り付ける
5. 必要に応じて時間範囲を調整する
6. クエリを実行します

結果には Karpenter 関連イベントのタイムラインが表示されるため、問題のトラブルシューティングや、クラスター内の EKS Auto Mode の動作の理解に役立ちます。特定のノードに対する Karpenter のアクションを確認するには、インスタンス ID を指定する次の行フィルターをクエリに追加します。

```
|filter @message like /i-12345678910123456/
```

> **Note**
>
> このクエリを使用するには、EKS クラスターでコントロールプレーンのログ記録を有効にする必要があります。これをまだ実行していない場合は、「the section called “コントロールプレーンのログ”」を参照してください。

Auto Mode に含まれているコントローラーのトラブルシューティング

コントローラーに問題がある場合は以下について調べる必要があります:

- そのコントローラーに関連付けられているリソースが適切にフォーマットされ、有効であるかどうか。
- AWS IAM リソースおよび Kubernetes RBAC リソースがクラスター用に適切に設定されているかどうか。詳細については、「the section called “ID とアクセス”」を参照してください。

関連リソース

高度なトラブルシューティング手順については、AWS re:Post の以下の記事を参照してください。

- EKS Auto Mode でよくあるスケールの問題をトラブルシューティングするにはどうすればよいか
- Amazon EKS Auto Mode でカスタムノードプールとノードクラスのプロビジョニングに関する問題をトラブルシューティングするにはどうすればよいか
- EKS Auto Mode に組み込みのノードプールでステータスが不明になる問題をトラブルシューティングするにはどうすればよいか

EKS Auto Mode のリリースノートを確認する

このページでは、Amazon EKS Auto Mode の更新について説明します。ここでは、機能、バグ修正、既知の問題、廃止された機能に関するお知らせを定期的に確認できます。

この特定のドキュメントページへのすべてのソースファイルの変更通知を受け取るには、RSS リーダーを使用して次の URL をサブスクライブできます。

```
https://github.com/awsdocs/amazon-eks-user-guide/commits/mainline/latest/ug/automode/auto-change.adoc.atom
```

2026 年 10 月 1 日

機能: EKS 1.37 以降、新しく作成された Auto Mode ノードプールでは、デフォルト設定が `WhenEmptyOrUnderutilized` ではなく `consolidationPolicy: Balanced` になります。`WhenEmptyOrUnderutilized` は、(1 時間あたりわずか 0.01 USD であっても) コスト削減が見込める代替候補が見つかるたびにノードのワークロードを中断させますが、`Balanced` は、1 時間あたりのコスト削減額がそのノードのポッドを中断させる際にかかるコストに見合うと判断した場合にのみ、中断を承認します。ほぼ同じコストでポッドのエビクションが減少するはずです。

新しい EKS 1.37 Auto Mode ノードプールで以前のデフォルトの動作を維持するには、明示的に設定を行ってください。

```
apiVersion: karpenter.sh/v1
kind: NodePool
metadata:
  name: example
spec:
  ...
  disruption:
    consolidationPolicy: WhenEmptyOrUnderutilized
```

GitOps (Argo CD や Flux など) を使用して Auto Mode ノードプールを管理する場合:

- `consolidationPolicy` を省略したマニフェストの場合、Git ソースにはまだ値が設定されていないにもかかわらず、1.37 以降ではライブオブジェクトに `Balanced` が表示されるようになります。Git とクラスターを一致させる場合は、マニフェストで `consolidationPolicy` を明示的に設定してください。

- デフォルト値は、リソースの作成時にのみ適用されます。未設定の Auto Mode ノードプール (または 1.37 でブートストラップされた新しいクラスター) を削除して再作成する GitOps 同期では、「同じ」Auto Mode ノードプールの設定が以前 `WhenEmptyOrUnderutilized` であった場合でも `Balanced` が適用されます。再作成時にサイレントフリップが発生しないようにフィールド値を設定してください。

EKS 1.37 では、2 つの EKS Auto Mode ノードプール (汎用、システム) も `consolidationPolicy: Balanced` に更新されます。EKS Auto はこれらを 1 時間ごとに調整するため、このフィールド値をその場で上書きすることはできません。特定のワークロードで以前の `WhenEmptyOrUnderutilized` 動作が必要な場合は、そのワークロードを更新して、`consolidationPolicy: WhenEmptyOrUnderutilized` が設定された NodePool をターゲットにします。

2026 年 9 月 14 日

非推奨: `volume-modifier-for-k8s` プロジェクトは、ネイティブの Kubernetes `VolumeAttributesClass` API への移行に伴い、非推奨となりました。このプロジェクトは、EBS ボリュームを変更するために PersistentVolumeClaim に注釈を設定する目的で使用されていました。2026 年 10 月 31 日以降、Amazon EKS Auto Mode ではこれらの注釈はサポートされなくなります。その日付以降に EBS ボリュームを変更するには、代わりに Kubernetes `VolumeAttributesClass` API を使用してください。詳細については、「the section called “StorageClass を作成する”」を参照してください。

2026 年 8 月 17 日

ドキュメント: 自動的に作成された `AWSServiceRoleForAmazonEKS` アクセスエントリの `eks:managed` グループを通じて、EKS Auto Mode の管理コントローラーに追加の Kubernetes RBAC を付与する手順を追加しました。これにより、ロードバランサーコントローラーに特定の `Secret` へのアクセス権を付与し、ALB `Ingress` 上の OIDC 設定を解決できるようにするなどのユースケースが可能になります。詳細については、「the section called “マネージドコントローラーに RBAC を付与する”」および「the section called “LBC にシークレットへのアクセスを許可する”」のウォークスルーを参照してください。

2026 年 8 月 12 日

機能: Amazon EKS Auto Mode のロードバランサーコントローラーが、AWS Load Balancer Controller の v3.4 までの機能をサポートするようになりました。なお、ゲートウェイ API はまだサポートされていません。

- イングレス向けの JWT 検証 – 新しいイングレスレベルのアクションを通じて、リクエストがバックエンドに到達する前に、Application Load Balancer (ALB) のリスナールールレベルで JSON Web Token (JWT) を検証します。詳細については、「Application Load Balancer が JWT 検証によるクライアント認証情報フローをサポートするようになりました」を参照してください。
- Network Load Balancer (NLB) の加重ターゲットグループ – NLB サービス上で、重みに基づいて複数のターゲットグループにトラフィックを分散します。これにより、ブルーグリーンおよびカナリアデプロイパターンがサポートされます。少なくとも 1 つの他のターゲットグループがゼロ以外の重みを持つ場合は、重みを 0 に設定することも含まれます。詳細については、「Network Load Balancer が加重ターゲットグループをサポートするようになりました」を参照してください。
- TargetGroupBinding の調整イベント – Amazon EKS は、TargetGroupBinding の調整の失敗時に Kubernetes イベントを発行するようになり、ターゲット登録の問題をコントローラーのログだけでなく `kubectl describe` を通じて確認できるようになりました。
- サブネットの順序を保持 – `aws-load-balancer-subnets` アノテーションが、内部的にサブネットを並べ替えるのではなく、指定した順序を尊重するようになりました。
- ALB のクロスゾーンロードバランシング – Application Load Balancer でクロスゾーンロードバランシングを明示的に無効化できるようになりました。

2026 年 8 月 5 日

機能: Amazon EKS Auto Mode ロードバランサーコントローラーは、マルチクラスターターゲットグループをサポートして、AWS Load Balancer Controller の動作と一致するようになりました。この機能を使用すると、同じターゲットグループ ARN を複数の `TargetGroupBinding` リソース間で共有できるため、1 つのターゲットグループが複数の Kubernetes クラスター (同じ VPC 内) を提供したり、他のソースからターゲットを受け入れたりできます。詳細については、「the section called “マルチクラスターターゲットグループ”」を参照してください。

2026 年 7 月 27 日

機能: Amazon Elastic Kubernetes Service (Amazon EKS) Auto Mode ロードバランサーコントローラーは、AWS Load Balancer Controller v2.13 および v2.14 の機能をサポートするようになりました。

- Application Load Balancer (ALB) URL の書き換え – アプリケーションを変更することなく、リクエストがバックエンドサービスに到達する前に、リクエスト URL とホストヘッダーを変換できるようになりました。詳細については、「AWS Application Load Balancer を使用した URL とホストヘッダーの書き換えの紹介」を参照してください。

- IngressClassParams の PrefixListsIDs と LoadBalancerName – Application Load Balancer のセキュリティグループプレフィックスリストとカスタムロードバランサー名を設定できるようになりました。どちらも Ingress 注釈と IngressClassParams (prefixListsIDs と loadBalancerName) のフィールドとしてサポートされています。IngressClassParams で設定すると、設定は IngressClass 内のすべての Ingress に適用されます。すべての Ingress リソースに個別に注釈を付ける必要がなくなりました。
- Ingress 用フロントエンド NLB – Network Load Balancer (NLB) を Application Load Balancer の前に配置できるようになりました。これは、NLB 静的 IP アドレスと AWS PrivateLink を ALB の Layer-7 ルーティング機能と組み合わせています。alb.ingress.kubernetes.io/enable-frontend-nlb 注釈を使用してこの機能を有効にします。詳細については、「Network Load Balancer の Application Load Balancer タイプターゲットグループ」を参照してください。
- TCP_UDP リスナーサポート – NLB サービスでは TCP_UDP リスナーを使用できるようになりました。これにより、同じポートで TCP トラフィックと UDP トラフィックの両方が許可されます。service.beta.kubernetes.io/aws-load-balancer-enable-tcp-udp-listener 注釈を使用してこの機能を有効にします。
- ターゲットグループごとのプロキシプロトコル – すべてのターゲットグループに設定を均一に適用するのではなく、service.beta.kubernetes.io/aws-load-balancer-proxy-protocol-per-target-group 注釈を使用して、個別のターゲットグループレベルで Proxy Protocol v2 ヘッダーを設定できるようになりました。
- IngressClassParams の targetType フィールド – IngressClassParams でデフォルトのターゲットタイプ (インスタンスまたは IP) を直接設定できるようになりました。これにより、すべての Ingress リソースに個別に注釈を付ける必要がなくなります。
- 到達可能性によるサブネット検出 – サブネットの選択に kubernetes.io/role タグが厳密に必要なくなりました。タグがない場合、コントローラーはルートテーブルベースの到達可能性分析にフォールバックするようになりました。コントローラーは現在、ip-address-type のデュアルスタックでロードバランサーに対してこのフォールバックをサポートしていません。
- ALB の IPv4 IP Address Manager (IPAM) サポート – インターネット向け ALB は、AWS マネージドアドレス範囲からの代わりに、Amazon Virtual Private Cloud (Amazon VPC) IPAM プールからパブリック IPv4 アドレスを取得できるようになりました。これにより、許可リストの予測可能な IP アドレスブロックが得られます。alb.ingress.kubernetes.io/ipam-ipv4-pool-id 注釈を使用してプールを指定します。詳細については、「IPAM を使用した ALB のパブリック IP アドレス割り当ての簡素化」を参照してください。

機能: EKS Auto Mode NodePools の `Balanced` 統合ポリシーを追加しました。`spec.disruption.consolidationPolicy: Balanced` を設定すると、コンピュー

ティングコスト削減を中断コストに対して比較することで、各統合アクションがスコアリングされます。中断が削減額を上回るアクションはスキップされます。現在 `WhenEmpty` を使用している場合は、`Balanced` に切り替えて統合によるコスト削減を実現できます。現在 `WhenEmptyOrUnderutilized` を使用している場合は、`Balanced` に切り替えてわずかなメリットのためのポッドの中断をなくすことができます。`WhenEmpty` と `WhenEmptyOrUnderutilized` は変更されず、既存の NodePools は現在の動作を維持します。詳細については、the section called “ノードプールの作成” と Karpenter の「中断」を参照してください。

2026 年 7 月 21 日

機能: NodeClass での静的ネットワークインターフェイス設定のサポートが追加されました。動的キャパシティプロビジョニングと静的キャパシティプロビジョニングの両方に `advancedNetworking.networkInterfaces` を使用して Elastic Fabric Adapter (EFA) ネットワークインターフェイスを設定できるようになりました。これにより、EFA 対応ノードを分散トレーニングと推論のワークロードで有効にできます。詳細については、「the section called “静的ネットワークインターフェイスの設定”」を参照してください。

2026 年 6 月 30 日

機能: EKS Auto Mode ロードバランサーコントローラーは、AWS Load Balancer Controller v2.10、v2.11、および v2.12 の機能をサポートするようになりました。

アップストリーム v2.12.0 から:

- リスナールールの優先度管理 – コントローラーはリスナールールの優先度を明示的に設定して順序を変更し、複数の Ingress ルールが同じリスナーをターゲットとする場合に順序の競合を解決できるようになりました

アップストリーム v2.11.0 から:

- Load Balancer Capacity Unit (LCU) 予約 – Application Load Balancer (ALB) と Network Load Balancer (NLB) の両方でキャパシティユニットを予約できるようになりました。これにより、既知のトラフィックパターンを持つワークロードの予測可能なパフォーマンスを確保できます

アップストリーム v2.10.0 から:

- ALB Shield Advanced 保護 – ALB リソースを alb.ingress.kubernetes.io/shield-advanced-protection 注釈を使用して AWS Shield Advanced で保護できるようになりました

- 独自のカスタム TargetGroupBinding の使用 – コントローラーによって作成されていない既存のターゲットグループを参照できるようになりました。これにより、外部マネージドインフラストラクチャとの統合が可能になります
- IPv6 クラスターでのデュアルスタック NLB の UDP サポート – IPv6 クラスターでの NLB サービスが UDP プロトコルリスナーをサポートするようになりました
- ALB HTTP および HTTPS リスナー属性 – 注釈によるリスナーレベルの属性 (ルーティング動作、ヘッダー変更など) のきめ細かな制御

マネージドポリシーでの更新

AWS は、これらの新機能をサポートするために AmazonEKSServiceRolePolicy と AmazonEKSLoadBalancingPolicy を更新しました。

カスタム IAM ポリシーを使用するお客様に必要なアクション

AWS マネージド AmazonEKSLoadBalancingPolicy を使用する代わりに EKS Auto Mode クラスターロールに独自のカスタム IAM ポリシーを提供する場合は、ポリシーに上記のアクセス許可が含まれていることを確認する必要があります。カスタムポリシーを更新しないと、新しい機能の使用時にアクセス拒否エラーが発生します。

パリティを検証するには、カスタムポリシーを最新バージョンの AmazonEKSLoadBalancingPolicy と比較します。

具体的には、ポリシーに以下が含まれていることを確認します。

- elasticloadbalancing:ModifyCapacityReservation、elasticloadbalancing:ModifyIpPools、elasticloadbalanci および elasticloadbalancing:SetRulePriorities
- ec2:DescribeIpamPools および ec2:DescribeRouteTables
- shield:CreateProtection、shield:DeleteProtection、および shield:TagResource

2026 年 6 月 9 日

機能: EKS Auto Mode にインスタンスステータスチェックモニタリングを追加しました。コンピューティングコントローラーで、EC2 `DescribeInstanceStatus` がポーリングされて、スケジュールされたメンテナンスイベントとインスタンスまたはシステムのステータスチェックの失敗が検出され、異常なノードが自動的に置き換えられるようになりました。

2026 年 6 月 4 日

ドキュメント: EKS Auto Mode でのコンピューティングコストの制御に関するガイダンスを追加しました (統合の仕組み、統合を妨げるもの、バーストワークロードの推奨パターンなど)。詳細については、「EKS Auto Mode でのコスト最適化」を参照してください。

2026 年 6 月 3 日

機能: EKS Auto Mode での中断可能なキャパシティ予約のサポートが追加されました。詳細については、「the section called “キャパシティ予約を制御する”」を参照してください。

2026 年 5 月 5 日

機能: EKS Auto Mode での EC2 プレイスメントグループのサポートが追加されました。詳細については、「the section called “ノードクラスの指定”」を参照してください。

2026 年 4 月 10 日

新しくサポートされるインスタンスタイプ: p6-b200、p6-b300、p5e、p5en、trn2、hpc8a、x8aedz、x8i。サポートされているインスタンスの完全なリストについては、the section called “管理されたインスタンス” を参照してください。

2026 年 4 月 2 日

タスク: NodeClass のドライラン検証では、リンクされた NodePool に基づいて動的に選択されたインスタンスタイプを使用するようになります。

2026 年 2 月 2 日

機能: EKS Auto Mode IPv6 クラスター内の IPv6 ポッドからの v4Egress トラフィックを無効にできるサポートが追加されました。詳細については、the section called “IPv6 クラスター内の IPv6 ポッドからの IPv4 egress を無効にする。” を参照してください。

2025 年 12 月 19 日

機能: オートノードにプレフィックスの代わりにセカンダリ IP アドレスをプロビジョニングするセカンダリ IP モードのサポートが追加されました。このモードは、1 つのセカンダリ IP を MinimalIPTarget として維持することで、セカンダリ IP またはプレフィックスをウォームアップする必要のないお客様の IP リソースを節約します。詳細については、「the section called “ノードクラスの指定”」および「the section called “ポッドのセカンダリ IP モード”」を参照してください。

2025 年 11 月 19 日

機能: ローカル NVMe ストレージにより、G、P、および Trn ファミリーインスタンスの Seekable OCI (SOCI) 並列プルと解凍を可能にしました。SOCI 並列プルと解凍は、これらのインスタンスファミリーが EKS Auto Mode であれば常に使用されるため、わざわざ設定を変更しなくても有効になります。SOCI の詳細については、「リリースブログ」を参照してください。

2025 年 11 月 19 日

機能: 静的キャパシティノードプールを新たにサポートして、決まった数のノードを維持できるようにしました。詳細については、the section called “静的ノードプールを作成する” を参照してください。

2025 年 10 月 23 日

機能: クラスターが米国リージョンにあるユーザーは、NodeClass 定義に `spec.advancedSecurity.fips` を指定することで、FIPS 互換 AMI の使用をリクエストできるようになりました。

2025 年 10 月 1 日

機能: EKS Auto Mode で、AWS ローカルゾーンへのノードのデプロイがサポートされるようになりました。詳細については、the section called “Local Zone を使用する” を参照してください。

2025 年 9 月 30 日

機能: `spec.role` フィールドから相互に排他的である NodeClass `spec.instanceProfile` に instanceProfile のサポートを追加しました。

2025 年 9 月 29 日

DRA は、現在 EKS Auto Mode ではサポートされていません。

2025 年 9 月 10 日

内部改善: Auto Mode コンピューティングコントローラーから発生したイベントは、`karpenter` ではなく `eks-auto-mode/compute` という名前を使用するようになります。

2025 年 8 月 24 日

バグ修正: VPC に使用されている DHCP オプションセットでカスタムドメイン名に大文字が含まれていた場合、無効なホスト名が生成されてノードがクラスターに参加できませんでした。この問題を解決し、ドメイン名に大文字が含まれていても正しく機能するようになりました。

2025 年 8 月 15 日

バグ修正: Pod Identity Agent は、IPv4 EKS クラスター内の IPv4 リンクローカルアドレスのみをリッスンします。これにより、Pod が IPv6 アドレスに到達できないという問題を回避します。

2025 年 8 月 6 日

機能: EKS Auto Mode ノードにパブリック IP アドレスが割り当てられないようにするための新しい設定を NodeClass `spec.advancedNetworking.associatePublicIPAddress` に追加しました。

2025 年 6 月 30 日

機能: Auto Mode NodeClass は設定されたカスタム KMS キーを使用して、読み取り/書き込みデータボリュームに加えて、インスタンスの読み取り専用ルートボリュームも暗号化するようになりました。以前、カスタム KMS キーはデータボリュームの暗号化にのみ使用されていました。

2025 年 6 月 20 日

機能: EC2 オンデマンドキャパシティ予約 (ODCR) へのワークロードのデプロイを制御するためのサポート。オプションのキー `capacityReservationSelectorTerms` が NodeClass に追加され、ワークロードが使用する ODCR を明示的に制御できます。詳細については、the section called “キャパシティ予約を制御する” を参照してください。

2025 年 6 月 13 日

機能: `NodeClass` での個別のポッドサブネットのサポート。これにより、オプションのキー `podSubnetSelectorTerms` と `podSecurityGroupSelectorTerms` が追加され、ポッドのサブネットとセキュリティグループが設定されます。詳細については、the section called “ポッドのサブネットとセキュリティグループを分離する” を参照してください。

2025 年 4 月 30 日

機能: NodeClass でのフォワードネットワークプロキシのサポート。これにより、HTTPS プロキシを設定するためのオプションキー advancedNetworking が追加されます。詳細については、the section called “ノードクラスの指定” を参照してください。

2025 年 4 月 18 日

機能: ユニキャスト DNS を介したローカルドメイン (通常はマルチキャスト DNS 用に予約) の解決をサポートします。

2025 年 4 月 11 日

機能: certificateBundles と ephemeralStorage.kmsKeyID を NodeClass に追加しました。詳細については、the section called “ノードクラスの指定” を参照してください。

機能: イメージのプル速度が向上しました。特に、高速なイメージ解凍を活用できるローカルインスタンスストレージを持つインスタンスタイプで、この効果を得られます。

バグ修正: FailedCreatePodSandBox のエラー原因である競合状態を解決しました。エラー詳細: Error while dialing: dial tcp 127.0.0.1:50051: connect: connection refused to sometimes occur for Pods scheduling to a Node immediately at startup

2025 年 4 月 4 日

機能: registryPullQPS を 5 から 25、registryBurst を 10 から 50 に増やして、クライアントで強制されるイメージプルスロットリングを減らします (Failed to pull image xyz: pull QPS exceeded)

2025 年 3 月 31 日

バグ修正: CoreDNS ポッドを Auto Mode ノードで実行している場合に、ノードのポッドからの DNS クエリがノードローカル DNS サーバーではなくその CoreDNS ポッドにヒットする問題を修正しました。Auto Mode ノードのポッドからの DNS クエリは、常にノードローカル DNS に送信されます。

2025 年 3 月 21 日

バグ修正: Auto Mode ノードは、クラスターに kube-dns サービスがインストールされていない場合にも kube-dns.kube-system.svc.cluster.local が正しく解決されるようになりました。GitHub の問題 #2546 に対処します。

2025 年 3 月 14 日

機能: IPv6 クラスターで IPv4 egress が有効になりました。IPv6 Auto Mode クラスターから送信される IPv4 トラフィックは、ノードプライマリ ENI の v4 アドレスに自動的に変換されるようになりました。

Amazon EKS クラスターのライフサイクルと設定

このセクションでは、Kubernetes クラスターの完全なライフサイクル管理と、それらを設定するさまざまな方法に関する詳細なガイダンスを提供します。クラスターの作成、クラスターの状態のモニタリング、Kubernetes バージョンの更新、クラスターの削除について説明します。また、エンドポイントアクセスの設定、Windows サポートの有効化/無効化、プライベートクラスターの設定、自動スケールの実装、マルチ AZ セットアップでのトラフィックリダイレクトのゾーンシフトによる耐障害性を高める方法に関する高度なトピックも含まれています。

トピック

- Amazon EKS Auto Mode クラスターを作成する
- Amazon EKS クラスターを作成します。
- Amazon EKS プロビジョンドコントロールプレーン
- 高度な Kubernetes コントロールプレーン設定
- Kubernetes バージョンアップグレードの準備およびクラスターインサイトでの設定ミスのトラブルシューティング
- 既存のクラスターを新しい Kubernetes バージョンに更新する
- 以前の Kubernetes バージョンにクラスターをロールバックする
- クラスターを削除
- クラスター API サーバーエンドポイント
- EKS クラスターに WiWindows ノードをデプロイする
- Windows サポートを無効にする
- インターネットアクセスが制限されたプライベートクラスターをデプロイする
- コントロールプレーンエグレスルーティングの設定
- Karpenter と Cluster Autoscaler を使用したクラスターコンピューティングのスケーリング
- Amazon EKS での Amazon Application Recovery Controller (ARC) のゾーンシフトの詳細
- EKS ゾーンシフトを有効にしてアベイラビリティーゾーンの障害を回避する

Amazon EKS Auto Mode クラスターを作成する

このトピックでは、高度な設定オプションを使用して Amazon EKS Auto Mode クラスターを作成する詳細な手順について説明します。前提条件、ネットワーキングオプション、アドオン設定を取り上

げています。このプロセスには、IAM ロールの設定、クラスター設定の構成、ネットワーキングパラメータの指定、アドオンの選択が含まれます。ユーザーは、AWS マネジメントコンソールまたは AWS CLI のいずれかを使用してクラスターを作成でき、どちらの方法でもステップバイステップのガイダンスが付いています。

より複雑でないセットアッププロセスを求める場合は、クラスター作成手順の簡略化について以下を参照してください。

- the section called “eksctl CLI”
- the section called “ AWS CLI”
- the section called “マネジメントコンソール”

この高度な設定ガイドは、EKS Auto Mode クラスターのセットアップをより細かく制御する必要があり、Amazon EKS の概念と要件に精通しているユーザーを対象としています。高度な設定に進む前に、すべての前提条件を満たし、EKS Auto Mode クラスターのネットワーキングおよび IAM 要件を十分に理解してください。

EKS Auto Mode には、追加の IAM アクセス許可が必要です。詳細については、以下を参照してください。

- the section called “EKS Auto Mode クラスターの IAM ロール”
- the section called “ID とアクセス”

> **Note**
>
> EKS Auto Mode を使用せずにクラスターを作成する場合は、「the section called “クラスターを作成する”」を参照してください。
> このトピックでは、高度な設定を取り上げています。EKS Auto Mode を使用して開始する場合は、「the section called “クラスターを作成する”」を参照してください。

前提条件

- Amazon EKS の要件 を満たす既存の VPC とサブネット。本番用にクラスターをデプロイする前に、VPC とサブネットの要件を十分に理解しておくことをお勧めします。VPC とサブネットがない場合はAmazon EKS に用意されている AWS クラウドフォーメーション テンプレートを使用して作成できます。

- デバイスまたは AWS CloudShell に、kubectl コマンドラインツールがインストールされていること。バージョンはご使用のクラスターの Kubernetes バージョンと同じか、1 つ前のマイナーバージョン以前、あるいはそれより新しいバージョンが使用できます。例えば、クラスターのバージョンが 1.29 である場合、kubectl のバージョン 1.28、1.29、または 1.30 が使用できます。kubectl をインストールまたはアップグレードする方法については、the section called “kubectl と eksctl のセットアップ” を参照してください。
- ご使用のデバイスまたは AWS CloudShell で、バージョン 2.12.3 以降、または AWS コマンドラインインターフェイス (AWS CLI) のバージョン 1.27.160 以降がインストールおよび設定されていること。現在のバージョンを確認するには、aws --version を参照してください。最新バージョンをインストールするには、「 AWS コマンドラインインターフェイスユーザーガイド」の「インストール」および「aws configure を使用したクイック設定」を参照してください。
- EKS および IAM リソースを作成および変更するアクセス許可を持つ IAM プリンシパル。

クラスターの作成 - AWS コンソール

1. Amazon EKS コンソールを開きます。
2. [クラスターを追加]、[作成] の順にクリックします。
3. [設定オプション] で [カスタム設定] を選択します。
   - このトピックでは、カスタム設定を取り上げています。クイック設定の詳細については、「the section called “マネジメントコンソール”」を参照してください。
4. [EKS Auto Mode を使用する] が有効になっていることを確認します。
   - このトピックでは、EKS Auto Mode でのクラスターの作成を取り上げています。EKS Auto Mode を使用せずにクラスターを作成する方法の詳細については、「the section called “クラスターを作成する”」を参照してください。
5. [クラスターの設定] ページで、次のフィールドに入力します：
   - [名前] - クラスターの名前。この名前には英数字 (大文字と小文字が区別されます)、ハイフン、下線のみを使用できます。先頭の文字は英数字である必要があります。また、100 文字より長くすることはできません。名前はクラスターを作成する AWS リージョンおよび AWS アカウント内で一意である必要があります。
   - [クラスター IAM ロール] – ユーザーに代わって AWS リソースを管理することを Kubernetes コントロールプレーンに許可するために作成した Amazon EKS クラスター IAM ロールを選択します。EKS Auto Mode 用のクラスター IAM ロールをまだ作成していない場合は、IAM コンソールで [推奨ロールを作成] ボタンを選択して、必要なアクセス許可を持つロールを作成します。

- [Kubernetes バージョン] – クラスターで使用する Kubernetes のバージョン。以前のバージョンが必要でない限り、最新バージョンを選択することをお勧めします。
- [アップグレードポリシー] – クラスターに対して設定する Kubernetes バージョンポリシー。クラスターを標準サポートバージョンでのみ実行したい場合は、[標準] を選択できます。バージョンの標準サポート終了時にクラスターの延長サポートに入りたい場合は、[延長] を選択できます。現在延長サポートを利用中の Kubernetes バージョンを選択した場合、オプションとして標準サポートを選択することはできません。

6. [クラスターを設定] ページの [Auto Mode コンピューティング] セクションで、次のフィールドに入力します。
   - ノードプール – 組み込みノードプールを使用するかどうかを決定します。詳細については、「the section called “組み込みノードプールを確認する”」を参照してください。
   - ノード IAM ロール – 組み込みノードプールのいずれかを有効にする場合は、ノード IAM ロールを選択する必要があります。EKS Auto Mode は、このロールを新しいノードに割り当てます。クラスターの作成後はこの値を変更することはできません。EKS Auto Mode 用のノード IAM ロールをまだ作成していない場合は、[推奨ロールを作成] ボタンを選択して、必要なアクセス許可を持つロールを作成します。このロールの詳細については、「the section called “ID とアクセス”」を参照してください。
7. [クラスターの設定] ページの [クラスターアクセス] セクションで、次のフィールドに入力します。
   - [クラスター管理者アクセスをブートストラップ] – クラスター作成者は自動的に Kubernetes 管理者になります。これを無効にする場合は、[クラスター管理者アクセスを拒否] を選択します。
   - [クラスター認証モード] – EKS Auto Mode には、EKS アクセスエントリ、EKS API 認証モードが必要です。必要に応じて、[EKS API と ConfigMap] を選択して `ConfigMap` 認証モードを有効にできます。
8. 「クラスターの設定」ページの残りのフィールドに入力します。
   - [Secret encryption] (シークレット暗号化) – (オプション) KMS キーを使用して Kubernetes シークレットのシークレット暗号化を有効にするよう選択します。クラスターを作成した後で、これを有効にすることもできます。この機能を有効にする前に、the section called “シークレット暗号化を有効にする” の情報をよく理解していることを確認してください。
   - [ARC ゾーンシフト] – EKS Auto Mode は ARC ゾーンシフトをサポートしていません。
   - [タグ] - (オプション) クラスターにタグを追加します。詳細については、the section called “リソースのタグ付け” を参照してください。

このページを読み終えたら、[次へ] を選択してください。

9. [ネットワーキングの指定] ページで、次のフィールドの値を選択してください：

- [VPC] – Amazon EKS VPC 要件を満たす既存の VPC を選択し、そこでクラスターを作成します。VPC を選択する前に、「the section called “VPC およびサブネットの要件”」の要件と考慮事項をすべて理解しておくことをお勧めします。クラスターの作成後は使用する VPC を変更できません。VPC が表示されていない場合はまず作成する必要があります。詳細については、the section called “「VPC を作成する」” を参照してください。
- [サブネット] - デフォルトで、前のフィールドで指定した VPC 内の利用可能なすべてのサブネットがあらかじめ選択されています。少なくとも 2 つ選択する必要があります。

  選択するサブネットは Amazon EKS サブネットの要件を満たす必要があります。サブネットを選択する前に、Amazon EKS VPC およびサブネットの要件と考慮事項をすべて理解しておくことをお勧めします。

  [セキュリティグループ] - (オプション Amazon EKS が作成するネットワークインターフェイスに関連付ける 1 つまたは複数のセキュリティグループを指定します。

  セキュリティグループを選択するかどうかにかかわらず、Amazon EKS はクラスターと VPC 間の通信を可能にするセキュリティグループを作成します。Amazon EKS はこのセキュリティグループおよびユーザーが選択したセキュリティグループを、作成するネットワークインターフェイスに関連付けます。Amazon EKS が作成するクラスターセキュリティグループの詳細については「the section called “セキュリティグループの要件”」を参照してください。Amazon EKS が作成するクラスターセキュリティグループのルールを変更できます。
- クラスターの IP アドレスファミリーの選択 – IPv4 または IPv6 を選択できます。

  デフォルトで、Kubernetes は `IPv4` アドレスを Pod とサービスに割り当てます。`IPv6` ファミリーの使用を決定する前に、VPC の要件と考慮事項、the section called “サブネットの要件と考慮事項”、the section called “セキュリティグループの要件”、および the section called “IPv6” のトピックの考慮事項と要件をすべて理解していることを確認します。`IPv6` ファミリーを選択すると、Kubernetes が `IPv6` サービスアドレスを割り当てる範囲を `IPv4` ファミリーで指定できるようには指定できません。Kubernetes は、一意のローカルアドレス範囲 (`fc00::/7`) からサービスにアドレスを割り当てます。
- (オプション) [Kubernetes サービス IP アドレスの範囲を設定する] を選択し、[サービスの **IPv4** 範囲] を指定します。

独自の範囲を指定すると、Kubernetes サービスと VPC にピアリングまたは接続されたその他のネットワークとの間の競合を防ぐことができます。CIDR 表記で範囲を入力します。例: `10.2.0.0/16`。

この　CIDR ブロックでは以下の要件を満たす必要があります：

- `10.0.0.0/8`、`172.16.0.0/12`、または `192.168.0.0/16` のいずれかの範囲内にある。
- 最小サイズが `/24`、最大サイズが `/12`。
- Amazon EKS リソースの VPC の範囲と重複しない。

このオプションを指定できるのは、`IPv4` アドレスファミリーを使用してクラスターを作成するときのみです。これを指定しない場合、Kubernetes は、`10.100.0.0/16` または `172.20.0.0/16` のいずれかの CIDR ブロックからサービス IP アドレスを割り当てます。

- [クラスターエンドポイントのアクセス] で、オプションを選択してください。クラスターを作成した後で、このオプションを変更できます。デフォルト以外のオプションを選択する前に、オプションとその意味を理解しておいてください。詳細については、the section called “クラスターエンドポイントのアクセス” を参照してください。

  このページを読み終えたら、[次へ] を選択してください。

10. (オプション) [オブザーバビリティの設定] ページで、有効にする [メトリクス] と [コントロールプレーンのロギング] オプションを選択してください。デフォルトではそれぞれのログタイプは無効化されています。
    - Prometheus メトリクスの詳細については、the section called “ステップ 1: Prometheus メトリクスを有効にする” を参照してください。
    - コントロールプレーンのログ記録 オプションの詳細については、the section called “コントロールプレーンのログ” を参照してください。
    - このページを読み終えたら、[次へ] を選択してください。
11. [アドオンの選択] ページで、クラスターに追加するアドオンを選択してください。[Amazon EKS アドオン] と [AWS マーケットプレイス アドオン] は必要な数だけ選択できます。インストールする [AWS Marketplace アドオン] がリストされていない場合は、ページ番号を選択して追加のページ結果を表示するか、検索ボックスにテキストを入力して使用可能な [AWS Marketplace アドオン] を検索できます。[カテゴリ]、[ベンダー]、または [料金モデル] でフィルタリングして、検索結果からアドオンを選択することもできます。クラスターを作成する際に、「the section called “Pod Identity”」で詳述されているように、EKS ポッドアイデンティティー をサポートするアドオンを表示、選択、インストールできます。

- EKS Auto Mode は、特定のアドオンの機能を自動化します。EKS Auto Mode クラスターに EKS マネージドノードグループをデプロイする場合は、[追加の Amazon EKS アドオン] を選択し、オプションを確認します。CoreDNS や kube-proxy などのアドオンをインストールすることが必要になる場合があります。EKS は、このセクションのアドオンをセルフマネージドノードとノードグループにのみインストールします。
- このページを読み終えたら、[次へ] を選択してください。

12[選択したアドオン設定の構成] ページで、インストールするバージョンを選択し、[次へ] を選択してください。クラスターを作成した後はいつでも新しいバージョンに更新できます。

EKS ポッドアイデンティティー をサポートするアドオンの場合、コンソールを使用して、アドオン専用に事前入力された名前、AWS マネージドポリシー、信頼ポリシーを使用してロールを自動的に生成できます。既存のロールを再利用したり、サポートされているアドオン用の新しいロールを作成したりできます。コンソールを使用して EKS ポッドアイデンティティー をサポートするアドオンのロールを作成するためのステップについては「the section called “アドオンの作成 (AWS コンソール) ”」を参照してください。アドオンが EKS Pod Identity をサポートしていない場合、クラスターの作成後にウィザードを使用してサービスアカウント用の IAM ロール (IRSA 作成する手順を示すメッセージが表示されます。

クラスターの作成後に、各アドオンの設定を更新できます。アドオンの設定の詳細については「the section called “アドオンを更新する”」を参照してください。このページを読み終えたら、[次へ] を選択してください。

13[確認と作成] ページで、前のページで入力または選択した情報を確認します。変更する必要がある場合は[編集] を選択してください。そのままでよければ、[作成] を選択してください。クラスターがプロビジョニングされている間、[ステータス] フィールドに [作成中] と表示されます。

> **Note**
>
> リクエストで指定したアベイラビリティーゾーンのいずれかに、Amazon EKS クラスターの作成に十分な容量がない場合にはエラーが表示されることがあります。このエラー出力には新しいクラスターをサポートできるアベイラビリティーゾーンが表示されます。アカウント向けにサポートされているアベイラビリティーゾーンにある 2 つ以上のサブネットを使用して、クラスターを作成します。詳細については、the section called “容量不足” を参照してください。

クラスターのプロビジョニングには数分かかります。

クラスターの作成 - AWS CLI

次の CLI の手順では、IAM リソースの作成とクラスターの作成について説明します。

EKS Auto Mode クラスター IAM ロールを作成する

ステップ 1: 信頼ポリシーを作成する

Amazon EKS サービスがロールを引き受けることを許可する信頼ポリシーを作成します。ポリシーを `trust-policy.json` として保存します。

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "eks.amazonaws.com"
      },
      "Action": [
        "sts:AssumeRole",
        "sts:TagSession"
      ]
    }
  ]
}
```

ステップ 2: IAM ロールを作成する

信頼ポリシーを使用して、クラスター IAM ロールを作成します。

```
aws iam create-role \
    --role-name AmazonEKSAutoClusterRole \
    --assume-role-policy-document file://trust-policy.json
```

ステップ 3: ロール ARN をメモする

以降のステップで使用する新しいロールの ARN を取得して保存します。

```
aws iam get-role --role-name AmazonEKSAutoClusterRole --query "Role.Arn" --output text
```

ステップ 4: 必要なポリシーをアタッチする

次の AWS マネージドポリシーをクラスター IAM ロールにアタッチして、必要なアクセス許可を付与します。

AmazonEKSClusterPolicy:

```
aws iam attach-role-policy \
    --role-name AmazonEKSAutoClusterRole \
    --policy-arn arn:aws:iam::aws:policy/AmazonEKSClusterPolicy
```

AmazonEKSコンピュートポリシー:

```
aws iam attach-role-policy \
    --role-name AmazonEKSAutoClusterRole \
    --policy-arn arn:aws:iam::aws:policy/AmazonEKSComputePolicy
```

AmazonEKSBlockStoragePolicyV2:

```
aws iam attach-role-policy \
    --role-name AmazonEKSAutoClusterRole \
    --policy-arn arn:aws:iam::aws:policy/AmazonEKSBlockStoragePolicyV2
```

AmazonEKSロードバランシングポリシー:

```
aws iam attach-role-policy \
    --role-name AmazonEKSAutoClusterRole \
    --policy-arn arn:aws:iam::aws:policy/AmazonEKSLoadBalancingPolicy
```

AmazonEKSNetworkingPolicy:

```
aws iam attach-role-policy \
    --role-name AmazonEKSAutoClusterRole \
    --policy-arn arn:aws:iam::aws:policy/AmazonEKSNetworkingPolicy
```

EKS Auto Mode ノード IAM ロールを作成する

ステップ 1: 信頼ポリシーを作成する

Amazon EKS サービスがロールを引き受けることを許可する信頼ポリシーを作成します。ポリシーを `node-trust-policy.json` として保存します。

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "ec2.amazonaws.com"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
```

ステップ 2: ノード IAM ロールを作成する

前のステップの node-trust-policy.json ファイルを使用して、ロールを引き受けることができるエンティティを定義します。次のコマンドを実行してノード IAM ロールを作成します。

```
aws iam create-role \
    --role-name AmazonEKSAutoNodeRole \
    --assume-role-policy-document file://node-trust-policy.json
```

ステップ 3: ロール ARN をメモする

ロールを作成したら、ノード IAM ロールの ARN を取得して保存します。以降のステップでこの ARN が必要になります。次のコマンドを使用して ARN を取得します。

```
aws iam get-role --role-name AmazonEKSAutoNodeRole --query "Role.Arn" --output text
```

ステップ 4: 必要なポリシーをアタッチする

次の AWS マネージドポリシーをノード IAM ロールにアタッチして、必要なアクセス許可を付与します。

AmazonEKSWorkerNodeMinimalPolicy:

```
aws iam attach-role-policy \
    --role-name AmazonEKSAutoNodeRole \
    --policy-arn arn:aws:iam::aws:policy/AmazonEKSWorkerNodeMinimalPolicy
```

AmazonEC2ContainerRegistryPullOnly:

```
aws iam attach-role-policy \
    --role-name AmazonEKSAutoNodeRole \
    --policy-arn arn:aws:iam::aws:policy/AmazonEC2ContainerRegistryPullOnly
```

クラスターを作成する

1. 下記のコマンドを使用して、クラスターを作成します。コマンドを実行する前に、次の置き換えを行います：
   - *#####* はクラスターを作成する AWS リージョンに置き換えます。
   - *#######* の部分は自分のクラスター名に置き換えます。この名前には英数字 (大文字と小文字が区別されます)、ハイフン、下線のみを使用できます。先頭の文字は英数字である必要があります。また、100 文字より長くすることはできません。名前はクラスターを作成する AWS リージョンおよび AWS アカウント内で一意である必要があります。
   - *1.30* を Amazon EKS がサポートする任意のバージョンに置き換えます。
   - *111122223333* は、自分のアカウント ID に置き換えます。
   - クラスターロールとノードロールに異なる名前の IAM ロールを作成した場合は、ARN を置き換えます。
   - subnetIds の値を独自の値に置き換えます。さらに ID を追加することもできます。少なくとも 2 つのサブネット ID を指定する必要があります。

     選択するサブネットは Amazon EKS サブネットの要件を満たす必要があります。サブネットを選択する前に、Amazon EKS VPC およびサブネットの要件と考慮事項をすべて理解しておくことをお勧めします。
   - セキュリティグループ ID を指定しない場合はコマンドから ,securityGroupIds=sg-<ExampleID1> を削除します。1 つまたは複数のセキュリティグループ ID を指定する場合はsecurityGroupIds の値を独自の値に置き換えます。さらに ID を追加することもできます。

     セキュリティグループを選択するかどうかにかかわらず、Amazon EKS はクラスターと VPC 間の通信を可能にするセキュリティグループを作成します。Amazon EKS はこのセキュリティグループおよびユーザーが選択したセキュリティグループを、作成するネットワークインターフェイスに関連付けます。Amazon EKS が作成するクラスターセキュリティグループの詳細については「the section called “セキュリティグループの要件”」を参照してください。Amazon EKS が作成するクラスターセキュリティグループのルールを変更できます。

```
aws eks create-cluster \
  --region region-code \
  --name my-cluster \
  --kubernetes-version 1.30 \
  --role-arn arn:aws:iam::111122223333:role/AmazonEKSAutoClusterRole \
  --resources-vpc-config '{"subnetIds": ["subnet-ExampleID1","subnet-ExampleID2"],
 "securityGroupIds": ["sg-ExampleID1"], "endpointPublicAccess": true,
 "endpointPrivateAccess": true}' \
  --compute-config '{"enabled": true, "nodeRoleArn":
 "arn:aws:iam::111122223333:role/AmazonEKSAutoNodeRole", "nodePools": ["general-
purpose", "system"]}' \
  --kubernetes-network-config '{"elasticLoadBalancing": {"enabled": true}}' \
  --storage-config '{"blockStorage": {"enabled": true}}' \
  --access-config '{"authenticationMode": "API"}'
```

> **Note**
>
> リクエストで指定したアベイラビリティーゾーンのいずれかに、Amazon EKS クラスターの作成に十分な容量がない場合にはエラーが表示されることがあります。このエラー出力には新しいクラスターをサポートできるアベイラビリティーゾーンが表示されます。アカウント向けにサポートされているアベイラビリティーゾーンにある 2 つ以上のサブネットを使用して、クラスターを作成します。詳細については、the section called “容量不足” を参照してください。

必要に応じて前のコマンドに追加する必要があるオプションの設定を次に示します。これらのオプションは、クラスターの作成時にのみ有効にすることができ、作成後は有効にできません。

- どの IPv4 Classless Inter-domain Routing (CIDR) ブロックから Kubernetes がサービス IP アドレスを割り当てるかを指定する場合、`--kubernetes-network-config serviceIpv4Cidr=<cidr-block>` を次のコマンドに追加することによって指定する必要があります。

  独自の範囲を指定すると、Kubernetes サービスと VPC にピアリングまたは接続されたその他のネットワークとの間の競合を防ぐことができます。CIDR 表記で範囲を入力します。例: `10.2.0.0/16`。

  この　CIDR ブロックでは以下の要件を満たす必要があります：

- 10.0.0.0/8、172.16.0.0/12、または 192.168.0.0/16 のいずれかの範囲内にある。
- 最小サイズが /24、最大サイズが /12。
- Amazon EKS リソースの VPC の範囲と重複しない。

このオプションを指定できるのは、IPv4 アドレスファミリーを使用してクラスターを作成するときのみです。これを指定しない場合、Kubernetes は、10.100.0.0/16 または 172.20.0.0/16 のいずれかの CIDR ブロックからサービス IP アドレスを割り当てます。

- クラスターを作成していて、そのクラスターで IPv4 アドレスではなく IPv6 アドレスを Pod とサービスに割り当てるようにする場合は、次のコマンドに --kubernetes-network-config ipFamily=ipv6 を追加します。

デフォルトで、Kubernetes は IPv4 アドレスを Pod とサービスに割り当てます。IPv6 ファミリーの使用を決定する前に、VPC の要件と考慮事項、the section called “サブネットの要件と考慮事項”、the section called “セキュリティグループの要件”、および the section called “IPv6” のトピックの考慮事項と要件をすべて理解していることを確認します。IPv6 ファミリーを選択すると、Kubernetes が IPv6 サービスアドレスを割り当てる範囲を IPv4 ファミリーで指定できるようには指定できません。Kubernetes は、一意のローカルアドレス範囲 (fc00::/7) からサービスにアドレスを割り当てます。

2. クラスターがプロビジョニングされるまでに数分かかります。クラスターのステータスのクエリを実行するには次のコマンドを使用します。

```
aws eks describe-cluster --region region-code --name my-cluster --query
 "cluster.status"
```

次のステップ

- the section called “kubectl を使用してクラスターにアクセスする”
- the section called “アクセスエントリ”
- クラスターのシークレット暗号化を有効にする。
- クラスターのログ記録を設定する。
- クラスターにノードを追加する。

Amazon EKS クラスターを作成します。

> **Note**
>
> このトピックでは EKS Auto Mode を使用しない Amazon EKS クラスターの作成を取り上げています。
> EKS Auto Mode クラスターの詳細な作成手順については、「the section called “自動クラスターの作成”」を参照してください。
> EKS Auto Mode を使用して開始するには、「the section called “クラスターを作成する (EKS Auto Mode)”」を参照してください。

このトピックでは使用可能なオプションの概要と、Amazon EKS クラスターの作成時に考慮すべき点を説明します。ノードのコンピューティングとしてオンプレミスインフラストラクチャを使用してクラスターを作成する必要がある場合は「the section called “クラスターを作成する”」を参照してください。Amazon EKS クラスターを初めて作成する場合ははじめに のガイドのいずれかに従うことをお勧めします。これらのガイドは使用可能なすべてのオプションを展開することなく、シンプルでデフォルトのクラスターを作成するのに役立ちます。

前提条件

- Amazon EKS の要件 を満たす既存の VPC とサブネット。本番用にクラスターをデプロイする前に、VPC とサブネットの要件を十分に理解しておくことをお勧めします。VPC とサブネットがない場合は、Amazon EKS に用意されている AWS CloudFormation テンプレートを使用して作成できます。
- デバイスまたは AWS CloudShell に、`kubectl` コマンドラインツールがインストールされていること。バージョンはご使用のクラスターの Kubernetes バージョンと同じか、1 つ前のマイナーバージョン以前、あるいはそれより新しいバージョンが使用できます。`kubectl` をインストールまたはアップグレードする方法については、the section called “kubectl と eksctl のセットアップ” を参照してください。
- ご使用のデバイスまたは AWS CloudShell で、バージョン `2.12.3` 以降、または AWS コマンドラインインターフェイス (AWS CLI) のバージョン `1.27.160` 以降がインストールおよび設定されていること。現在のバージョンを確認するには、`aws --version | cut -d / -f2 | cut -d ' ' -f1` を参照してください。`yum`、`apt-get`、macOS 用の Homebrew などのパッケージマネージャーは、多くの場合 AWS CLI の最新バージョンより数バージョン古くなっています。最新バージョンをインストールするには、 AWS コマンドラインインターフェイスユーザーガイドの「インストール」および「aws configure を使用したクイック設定」を参照してください。AWS

CloudShell にインストールされている AWS CLI バージョンも最新バージョンより数バージョン遅れることがあります。更新するには、「AWS CloudShell ユーザーガイド」の「ホームディレクトリへの AWS CLI のインストール」を参照してください。

- Amazon EKS クラスターを `create` および `describe` するための許可を持つ IAM プリンシパル。詳細については、「the section called “Outpost にローカル Kubernetes クラスターを作成する”」および「the section called “すべてのクラスターの一覧表示または説明”」を参照してください。

ステップ 1: クラスター IAM ロールを作成する

1. 既にクラスター IAM ロールがある場合、または `eksctl` を使用してクラスターを作成する場合はこのステップはスキップできます。デフォルトでは`eksctl` により、ロールが自動的に作成されます。
2. IAM 信頼ポリシー用の JSON ファイルを作成するには次のコマンドを実行してください。

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "eks.amazonaws.com"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
```

3. Amazon EKS クラスターの IAM ロールを作成します。必要であれば、前のステップでファイルを書き込んだコンピュータ上のパスを *eks-cluster-role-trust-policy.json* の前につけます。このコマンドは前のステップで作成した信頼ポリシーをロールに関連付けます。IAM ロールを作成するには、ロールを作成する IAM プリンシパル に `iam:CreateRole` アクション (許可) を割り当てる必要があります。

```
aws iam create-role --role-name myAmazonEKSClusterRole --assume-role-policy-document
 file://"eks-cluster-role-trust-policy.json"
```

4. Amazon EKS 管理のポリシーを割り当てるか、独自のカスタムポリシーを作成できます。カスタムポリシーで使用する必要がある最小限の許可については、「the section called “クラスター IAM ロール”」を参照してください。

   AmazonEKSClusterPolicy という名前の Amazon EKS マネージド型ポリシーをロールにアタッチします。IAM ポリシーを IAM プリンシパルにアタッチするにはポリシーのアタッチを行っているプリンシパルに、次のいずれかの IAM アクション (許可) を割り当てる必要があります: `iam:AttachUserPolicy` または `iam:AttachRolePolicy`。

   ```
   aws iam attach-role-policy --policy-arn arn:aws:iam::aws:policy/
   AmazonEKSClusterPolicy --role-name myAmazonEKSClusterRole
   ```

サービスにリンクされたロール

Amazon EKS は、`AWSServiceRoleForAmazonEKS` というサービスにリンクされたロールを自動的に作成します。

これはクラスター IAM ロールに追加されます。サービスにリンクされたロールはAmazon EKS に直接リンクされた一意のタイプの IAM ロールです。このロールにより、Amazon EKS はアカウント内のクラスターを管理できます。詳細については、「the section called “クラスターの ロール”」を参照してください。

EKS クラスターを作成するために使用する IAM プリンシパルには、サービスにリンクされたロールを作成するためのアクセス許可が必要です。これには `iam:CreateServiceLinkedRole` 権限が含まれます。

サービスにリンクされたロールがまだ存在せず、現在の IAM ロールにそれを作成するための十分なアクセス許可がない場合、クラスター作成オペレーションは失敗します。

ステップ 2: クラスターを作成する

以下を使用してクラスターを作成できます:

- eksctl
- の AWS マネジメントコンソール
- AWS CLI

クラスターを作成します – eksctl

1. デバイスまたは AWS CloudShell にインストールされている `eksctl` コマンドラインツールのバージョン `0.215.0` 以降が必要です。`eksctl` をインストールまたはアップグレードするには、`eksctl` ドキュメントの「インストール」を参照してください。
2. デフォルトの AWS リージョンに、Amazon EKS のデフォルト Kubernetes バージョンを使用して、Amazon EKS IPv4 クラスターを作成します。コマンドを実行する前に、次の置き換えを行います:
3. *region-code* はクラスターを作成する AWS リージョンに置き換えます。
4. ******* の部分は自分のクラスター名に置き換えます。この名前には英数字 (大文字と小文字が区別されます) とハイフンのみを使用できます。先頭の文字は英数字である必要があります。また、100 文字より長くすることはできません。名前はクラスターを作成する AWS リージョンおよび AWS アカウント内で一意である必要があります。
5. *1.37* を Amazon EKS がサポートする任意のバージョンに置き換えます。
6. 要件を満たすように `vpc-private-subnets` の値を変更します。さらに ID を追加することもできます。少なくとも 2 つのサブネット ID を指定する必要があります。パブリックサブネットを指定する場合は、`--vpc-private-subnets` を `--vpc-public-subnets` に変更できます。パブリックサブネットにはインターネットゲートウェイへのルートに関連付けられたルートテーブルがありますが、プライベートサブネットには関連付けられたルートテーブルがありません。可能な限り、プライベートサブネットを使用することをお勧めします。

   選択するサブネットは Amazon EKS サブネットの要件を満たす必要があります。サブネットを選択する前に、Amazon EKS VPC およびサブネットの要件と考慮事項をすべて理解しておくことをお勧めします。
7. 次のコマンドを実行してください:

   ```
   eksctl create cluster --name my-cluster --region region-code --version 1.37 --vpc-private-subnets subnet-ExampleID1,subnet-ExampleID2 --without-nodegroup
   ```

   クラスターのプロビジョニングには数分かかります。クラスターの作成中は数行の出力が表示されます。出力の最後の行は次のサンプル行のようになります。

   ```
   [#]  EKS cluster "my-cluster" in "region-code" region is ready
   ```
8. 「the section called “ステップ 3: kubeconfig を更新する”」に進みます

オプション設定

eksctl を使用してクラスターを作成するときに指定できるほとんどのオプションを表示するにはeksctl create cluster --help コマンドを使用します。使用可能なオプションをすべて表示するには、config ファイルを使用します。詳細については、eksctl ドキュメントの「設定ファイルの使用」と「設定ファイルのスキーマ」を参照してください。設定ファイルの例は、GitHub で見つけることができます。

必要に応じて前のコマンドに追加する必要があるオプションの設定を次に示します。これらのオプションはクラスターの作成時にのみ有効にすることができ、作成後は有効にできません。これらのオプションを指定する必要がある場合は前のコマンドを使用するのではなく、eksctl config ファイルを使用してクラスターを作成し、設定を指定する必要があります。

- Amazon EKS が作成するネットワークインターフェイスに割り当てる 1 つまたは複数のセキュリティグループを指定する場合は、securityGroup オプションを指定します。

  セキュリティグループを選択するかどうかにかかわらず、Amazon EKS はクラスターと VPC 間の通信を可能にするセキュリティグループを作成します。Amazon EKS はこのセキュリティグループおよびユーザーが選択したセキュリティグループを、作成するネットワークインターフェイスに関連付けます。Amazon EKS が作成するクラスターセキュリティグループの詳細については、「the section called “セキュリティグループの要件”」を参照してください。Amazon EKS が作成するクラスターセキュリティグループのルールを変更できます。
- どの IPv4 Classless Inter-domain Routing (CIDR) ブロックから Kubernetes がサービス IP アドレスを割り当てるかを指定する場合、serviceIPv4CIDR オプションを指定します。

  独自の範囲を指定すると、Kubernetes サービスと VPC にピアリングまたは接続されたその他のネットワークとの間の競合を防ぐことができます。CIDR 表記で範囲を入力します。例: 10.2.0.0/16。

  この　CIDR ブロックでは以下の要件を満たす必要があります:

  - 10.0.0.0/8、172.16.0.0/12、または 192.168.0.0/16 のいずれかの範囲内にある。
  - 最小サイズが /24、最大サイズが /12。
  - Amazon EKS リソースの VPC の範囲と重複しない。

    このオプションを指定できるのは、IPv4 アドレスファミリーを使用してクラスターを作成するときのみです。これを指定しない場合、Kubernetes は、10.100.0.0/16 または 172.20.0.0/16 のいずれかの CIDR ブロックからサービス IP アドレスを割り当てます。ハ

イブリッドノード]が有効になっているクラスターでは、Amazon EKS は、設定されたリモートノードとポッドネットワークとの競合を避けるために、別の CIDR を選択する場合があります。

- クラスターを作成していて、そのクラスターで IPv4 アドレスではなく IPv6 アドレスを Pod とサービスに割り当てるようにする場合は、ipFamily オプションを指定します。

  デフォルトで、Kubernetes は IPv4 アドレスを Pod とサービスに割り当てます。IPv6 ファミリーの使用を決定する前に、「the section called “VPC の要件と考慮事項”」、「the section called “サブネットの要件と考慮事項”」、「the section called “セキュリティグループの要件”」、および「the section called “IPv6”」のトピックの考慮事項と要件をすべて理解していることを確認します。IPv6 ファミリーを選択すると、Kubernetes が IPv6 サービスアドレスを割り当てる範囲を IPv4 ファミリーで指定できるようには指定できません。Kubernetes は、一意のローカルアドレス範囲 (fc00::/7) からサービスにアドレスを割り当てます。

クラスターの作成 – AWS コンソール

1. Amazon EKS コンソールを開きます。
2. [クラスターを追加]、[作成] の順にクリックします。
3. [設定オプション] で [カスタム設定] を選択してください。
   - EKS Auto Mode でクラスターをすばやく作成する方法については、「the section called “マネジメントコンソール”」を参照してください。
4. [EKS Auto Mode] で、[EKS Auto Mode を使用する] をオフに切り替えます。
   - カスタム設定で EKS Auto Mode クラスターを作成する方法については、「the section called “自動クラスターの作成”」を参照してください。
5. [クラスターの設定] ページで、次のフィールドに入力します:
   - [名前] – クラスターの名前。この名前には英数字 (大文字と小文字が区別されます)、ハイフン、下線のみを使用できます。先頭の文字は英数字である必要があります。また、100 文字より長くすることはできません。名前はクラスターを作成する AWS リージョンおよび AWS アカウント内で一意である必要があります。
   - [クラスター IAM ロール] – ユーザーに代わって AWS リソースを管理することを Kubernetes コントロールプレーンに許可するために作成した Amazon EKS クラスター IAM ロールを選択します。
   - [Kubernetes バージョン] – クラスターで使用する Kubernetes のバージョン。以前のバージョンが必要でない限り、最新バージョンを選択することをお勧めします。

- [サポートタイプ] – クラスターに設定する Kubernetes バージョンポリシー。クラスターを標準サポートバージョンでのみ実行する場合は [標準サポート] を選択できます。バージョンの標準サポート終了時にクラスターで延長サポートに入る場合は [延長サポート] を選択できます。現在延長サポートを利用中の Kubernetes バージョンを選択した場合、オプションとして標準サポートを選択することはできません。
- [シークレット暗号化] – (オプション) KMS キーを使用して Kubernetes シークレットのシークレット暗号化を有効にするよう選択します。クラスターを作成した後で、これを有効にすることもできます。この機能を有効にする前に、the section called “シークレット暗号化を有効にする” の情報をよく理解していることを確認してください。
- [タグ] – (オプション) クラスターにタグを追加します。詳細については、「the section called “リソースのタグ付け”」を参照してください。
- ARC ゾーンシフト – (オプション) Amazon Application Recovery Controller を使用して、アベイラビリティーゾーンの障害を軽減できます。詳細については、「the section called “ゾーンシフトの詳細”」を参照してください。

6. [クラスターの設定] ページの [クラスターアクセス] セクションで、次のフィールドに入力します:
   - [クラスター管理者アクセスをブートストラップ] – クラスター作成者は自動的に Kubernetes 管理者になります。これを無効にする場合は、[クラスター管理者アクセスを拒否] を選択してください。
   - [クラスター認証モード] – IAM ユーザーおよびロールに Kubernetes API へのアクセスを許可する方法を決定します。詳細については、the section called “クラスター認証モードを設定する” を参照してください。

     このページを読み終えたら、[次へ] を選択してください。
7. [ネットワーキングの指定] ページで、次のフィールドの値を選択してください:
   - [VPC] – Amazon EKS VPC 要件を満たす既存の VPC を選択し、そこでクラスターを作成します。VPC を選択する前に、「the section called “VPC およびサブネットの要件”」の要件と考慮事項をすべて理解しておくことをお勧めします。クラスターの作成後は使用する VPC を変更できません。VPC が表示されていない場合はまず作成する必要があります。詳細については、the section called “「VPC を作成する」” を参照してください。
   - [サブネット] – デフォルトで、前のフィールドで指定した VPC 内の利用可能なすべてのサブネットがあらかじめ選択されています。少なくとも 2 つ選択する必要があります。

     選択するサブネットは Amazon EKS サブネットの要件を満たす必要があります。サブネットを選択する前に、Amazon EKS VPC およびサブネットの要件と考慮事項をすべて理解しておくことをお勧めします。

[セキュリティグループ] – (オプション Amazon EKS が作成するネットワークインターフェイスに関連付ける 1 つまたは複数のセキュリティグループを指定します。

セキュリティグループを選択するかどうかにかかわらず、Amazon EKS はクラスターと VPC 間の通信を可能にするセキュリティグループを作成します。Amazon EKS はこのセキュリティグループおよびユーザーが選択したセキュリティグループを、作成するネットワークインターフェイスに関連付けます。Amazon EKS が作成するクラスターセキュリティグループの詳細については、「the section called “セキュリティグループの要件”」を参照してください。Amazon EKS が作成するクラスターセキュリティグループのルールを変更できます。

- クラスターの IP アドレスファミリーの選択 – IPv4 または IPv6 を選択できます。

  デフォルトで、Kubernetes は IPv4 アドレスを Pod とサービスに割り当てます。IPv6 ファミリーの使用を決定する前に、VPC の要件と考慮事項、the section called “サブネットの要件と考慮事項”、the section called “セキュリティグループの要件”、および the section called “IPv6” のトピックの考慮事項と要件をすべて理解していることを確認します。IPv6 ファミリーを選択すると、Kubernetes が IPv6 サービスアドレスを割り当てる範囲を IPv4 ファミリーで指定できるようには指定できません。Kubernetes は、一意のローカルアドレス範囲 (fc00::/7) からサービスにアドレスを割り当てます。

- (オプション) [Kubernetes サービス IP アドレスの範囲を設定する] を選択し、[サービスの **IPv4** 範囲] を指定します。

  独自の範囲を指定すると、Kubernetes サービスと VPC にピアリングまたは接続されたその他のネットワークとの間の競合を防ぐことができます。CIDR 表記で範囲を入力します。例: 10.2.0.0/16。

  この　CIDR ブロックでは以下の要件を満たす必要があります:

  - 10.0.0.0/8、172.16.0.0/12、または 192.168.0.0/16 のいずれかの範囲内にある。
  - 最小サイズが /24、最大サイズが /12。
  - Amazon EKS リソースの VPC の範囲と重複しない。

このオプションを指定できるのは、IPv4 アドレスファミリーを使用してクラスターを作成するときのみです。これを指定しない場合、Kubernetes は、10.100.0.0/16 または 172.20.0.0/16 のいずれかの CIDR ブロックからサービス IP アドレスを割り当てます。ハイブリッドノードが有効になっているクラスターでは、Amazon EKS は、設定されたリモートノードとポッドネットワークとの競合を避けるために、別の CIDR を選択する場合があります。

- [クラスターエンドポイントのアクセス] で、オプションを選択してください。クラスターを作成した後で、このオプションを変更できます。デフォルト以外のオプションを選択する前に、オプションとその意味を理解しておいてください。詳細については、「the section called “クラスターエンドポイントのアクセス”」を参照してください。
- (オプション) コントロールプレーンエグレスでは、コントロールプレーンから VPC へのエグレストラフィックのルーティング方法を選択します。デフォルトは Amazon EKS マネージドです。顧客ルーティングを選択した場合、ユーザーにはコントロールプレーンから VPC リソースへのネットワークパスに対する責任が発生します。詳細については、「the section called “コントロールプレーンエグレス”」を参照してください。

  このページを読み終えたら、[次へ] を選択してください。

8. (オプション) [オブザーバビリティの設定] ページで、有効にする [メトリクス] と [コントロールプレーンのロギング] オプションを選択してください。デフォルトではそれぞれのログタイプは無効化されています。
   - Prometheus メトリクスの詳細については、the section called “ステップ 1: Prometheus メトリクスを有効にする” を参照してください。
   - コントロールプレーンのログ記録 オプションの詳細については、the section called “コントロールプレーンのログ” を参照してください。

   このページを読み終えたら、[次へ] を選択してください。
9. [アドオンの選択] ページで、クラスターに追加するアドオンを選択してください。特定のアドオンが事前に選択されています。[Amazon EKS アドオン] と [AWS マーケットプレイス アドオン] は必要な数だけ選択できます。インストールする [AWS Marketplace アドオン] がリストされていない場合は、ページ番号を選択して追加のページ結果を表示するか、検索ボックスにテキストを入力して使用可能な [AWS Marketplace アドオン] を検索できます。[カテゴリ]、[ベンダー]、または [料金モデル] でフィルタリングして、検索結果からアドオンを選択することもできます。クラスターを作成する際に、「the section called “Pod Identity”」で詳述されているように、EKS Pod Identity をサポートするアドオンを表示、選択、インストールできます。

   このページを読み終えたら、[次へ] を選択してください。

   Amazon VPC CNI、CoreDNS、kube-proxy などの一部のアドオンはデフォルトでインストールされます。デフォルトのアドオンのいずれかを無効にすると、Kubernetes アプリケーションを実行する機能に影響する場合があります。
10. [選択したアドオン設定の構成] ページで、インストールするバージョンを選択してください。クラスターを作成した後はいつでも新しいバージョンに更新できます。

EKS Pod Identity をサポートするアドオンの場合、コンソールを使用して、アドオン専用に事前入力された名前、AWS マネージドポリシー、信頼ポリシーを使用してロールを自動的に生成できます。既存のロールを再利用したり、サポートされているアドオン用の新しいロールを作成したりできます。コンソールを使用して EKS Pod Identity をサポートするアドオンのロールを作成するためのステップについては、「the section called “アドオンの作成 (AWS コンソール) ”」を参照してください。アドオンが EKS Pod Identity をサポートしていない場合、クラスターの作成後にウィザードを使用してサービスアカウント用の IAM ロール (IRSA 作成する手順を示すメッセージが表示されます。

クラスターの作成後に、各アドオンの設定を更新できます。アドオンの設定の詳細については、「the section called “アドオンを更新する”」を参照してください。このページを読み終えたら、[次へ] を選択してください。

11[確認と作成] ページで、前のページで入力または選択した情報を確認します。変更する必要がある場合は[編集] を選択してください。そのままでよければ、[作成] を選択してください。クラスターがプロビジョニングされている間、[ステータス] フィールドに [作成中] と表示されます。

> **Note**
>
> リクエストで指定したアベイラビリティーゾーンのいずれかに、Amazon EKS クラスターの作成に十分な容量がない場合にはエラーが表示されることがあります。このエラー出力には新しいクラスターをサポートできるアベイラビリティーゾーンが表示されます。アカウント向けにサポートされているアベイラビリティーゾーンにある 2 つ以上のサブネットを使用して、クラスターを作成します。詳細については、the section called “容量不足” を参照してください。

クラスターのプロビジョニングには数分かかります。

12.「the section called “ステップ 3: kubeconfig を更新する”」に進みます

クラスターの作成 – AWS CLI

1. 下記のコマンドを使用して、クラスターを作成します。コマンドを実行する前に、次の置き換えを行います:
   - *region-code* はクラスターを作成する AWS リージョンに置き換えます。

- *my-cluster* の部分は自分のクラスター名に置き換えます。この名前には英数字 (大文字と小文字が区別されます)、ハイフン、下線のみを使用できます。先頭の文字は英数字である必要があります。また、100 文字より長くすることはできません。名前はクラスターを作成する AWS リージョンおよび AWS アカウント内で一意である必要があります。
- *1.37* を Amazon EKS がサポートする任意のバージョンに置き換えます。
- *111122223333* をアカウント ID に、*myAmazonEKSClusterRole* をクラスター IAM ロールの名前に置き換えます。
- subnetIds の値を独自の値に置き換えます。さらに ID を追加することもできます。少なくとも 2 つのサブネット ID を指定する必要があります。

  選択するサブネットは Amazon EKS サブネットの要件を満たす必要があります。サブネットを選択する前に、Amazon EKS VPC およびサブネットの要件と考慮事項をすべて理解しておくことをお勧めします。
- セキュリティグループ ID を指定しない場合はコマンドから ,securityGroupIds=sg-<ExampleID1> を削除します。1 つまたは複数のセキュリティグループ ID を指定する場合はsecurityGroupIds の値を独自の値に置き換えます。さらに ID を追加することもできます。

  セキュリティグループを選択するかどうかにかかわらず、Amazon EKS はクラスターと VPC 間の通信を可能にするセキュリティグループを作成します。Amazon EKS はこのセキュリティグループおよびユーザーが選択したセキュリティグループを、作成するネットワークインターフェイスに関連付けます。Amazon EKS が作成するクラスターセキュリティグループの詳細については、「the section called “セキュリティグループの要件”」を参照してください。Amazon EKS が作成するクラスターセキュリティグループのルールを変更できます。
- (オプション) コントロールプレーンから VPC へのエグレスパスを自分で管理するには、controlPlaneEgressMode=CUSTOMER_ROUTED を --resources-vpc-config に追加します。デフォルトは AWS_MANAGED です。クラスターの作成後は、この設定も変更できます。詳細については、「the section called “コントロールプレーンエグレス”」を参照してください。

```
aws eks create-cluster --region region-code --name my-cluster --kubernetes-version
 1.37 \
   --role-arn arn:aws:iam::111122223333:role/myAmazonEKSClusterRole \
   --resources-vpc-config subnetIds=subnet-ExampleID1,subnet-
ExampleID2,securityGroupIds=sg-ExampleID1
```

> **Note**
>
> リクエストで指定したアベイラビリティーゾーンのいずれかに、Amazon EKS クラスターの作成に十分な容量がない場合にはエラーが表示されることがあります。このエラー出力には新しいクラスターをサポートできるアベイラビリティーゾーンが表示されます。アカウント向けにサポートされているアベイラビリティーゾーンにある 2 つ以上のサブネットを使用して、クラスターを作成します。詳細については、the section called “容量不足” を参照してください。

必要に応じて前のコマンドに追加する必要があるオプションの設定を次に示します。これらのオプションはクラスターの作成時にのみ有効にすることができ、作成後は有効にできません。

- デフォルトではEKS はクラスターの作成時に複数のネットワーキングアドオンをインストールします。これには Amazon VPC CNI、CoreDNS、kube-proxy が含まれます。

  これらのデフォルトのネットワーキングアドオンのインストールを無効にする場合は以下のパラメータを使用します。これは Cilium などの代替 CNI に使用できます。詳細については、「EKS API リファレンス」を参照してください。

  ```
  aws eks create-cluster --no-bootstrap-self-managed-addons …
  ```

- どの IPv4 Classless Inter-domain Routing (CIDR) ブロックから Kubernetes がサービス IP アドレスを割り当てるかを指定する場合、`--kubernetes-network-config serviceIpv4Cidr=<cidr-block>` を次のコマンドに追加することによって指定する必要があります。

  独自の範囲を指定すると、Kubernetes サービスと VPC にピアリングまたは接続されたその他のネットワークとの間の競合を防ぐことができます。CIDR 表記で範囲を入力します。例: `10.2.0.0/16`。

  この　CIDR ブロックでは以下の要件を満たす必要があります:

  - `10.0.0.0/8`、`172.16.0.0/12`、または `192.168.0.0/16` のいずれかの範囲内にある。
  - 最小サイズが `/24`、最大サイズが `/12`。
  - Amazon EKS リソースの VPC の範囲と重複しない。

このオプションを指定できるのは、IPv4 アドレスファミリーを使用してクラスターを作成するときのみです。これを指定しない場合、Kubernetes は、`10.100.0.0/16` または `172.20.0.0/16`

のいずれかの CIDR ブロックからサービス IP アドレスを割り当てます。ハイブリッドノードが有効になっているクラスターでは、Amazon EKS は、設定されたリモートノードとポッドネットワークとの競合を避けるために、別の CIDR を選択する場合があります。

- クラスターを作成していて、そのクラスターで IPv4 アドレスではなく IPv6 アドレスを Pod とサービスに割り当てるようにする場合は、次のコマンドに `--kubernetes-network-config ipFamily=ipv6` を追加します。

  デフォルトで、Kubernetes は IPv4 アドレスを Pod とサービスに割り当てます。IPv6 ファミリーの使用を決定する前に、VPC の要件と考慮事項、the section called “サブネットの要件と考慮事項”、the section called “セキュリティグループの要件”、および the section called “IPv6” のトピックの考慮事項と要件をすべて理解していることを確認します。IPv6 ファミリーを選択すると、Kubernetes が IPv6 サービスアドレスを割り当てる範囲を IPv4 ファミリーで指定できるようには指定できません。Kubernetes は、一意のローカルアドレス範囲 (`fc00::/7`) からサービスにアドレスを割り当てます。

2. クラスターがプロビジョニングされるまでに数分かかります。クラスターのステータスのクエリを実行するには次のコマンドを使用します。

```
aws eks describe-cluster --region region-code --name my-cluster --query
 "cluster.status"
```

   出力が「ACTIVE」を返すまで、次のステップに進まないでください。

3. 「the section called “ステップ 3: kubeconfig を更新する”」に進みます

ステップ 3: kubeconfig を更新する

1. `eksctl` を使用してクラスターを作成した場合、このステップはスキップできます。`eksctl` によってこのステップはすでに完了しているからです。新しいコンテキストを `kubectl config` ファイルに追加して、`kubectl` がクラスターと通信できるようにします。ファイルを作成および更新する方法の詳細については、「the section called “kubectl を使用してクラスターにアクセスする”」を参照してください。

```
aws eks update-kubeconfig --region region-code --name my-cluster
```

   出力例は次のとおりです。

```
Added new context arn:aws:eks:region-code:111122223333:cluster/my-cluster to /home/
username/.kube/config
```

2. 次のコマンドを実行して、クラスターとの通信を確認します。

```
kubectl get svc
```

出力例は次のとおりです。

```
NAME         TYPE        CLUSTER-IP   EXTERNAL-IP   PORT(S)   AGE
kubernetes   ClusterIP   10.100.0.1   <none>        443/TCP   28h
```

ステップ 4: クラスターのセットアップ

1. (推奨) Amazon EKS アドオンを使用するか、固有の AWS Identity and Access Management (IAM) アクセス許可を個々の Kubernetes ワークロードに付与できるようにするには、クラスター用に IAM OpenID Connect (OIDC) プロバイダーを作成します。クラスター用に IAM OIDC プロバイダーを作成する必要があるのは 1 回だけです。Amazon EKS アドオンの詳細については、「the section called “Amazon EKS アドオン”」を参照してください。ワークロードに特定の IAM アクセス許可を割り当てる方法については、the section called “IRSA による認証情報” を参照してください。
2. (推奨) Amazon EC2 ノードをクラスターにデプロイする前に Amazon VPC CNI plugin for Kubernetes プラグイン用にクラスターを設定します。デフォルトではプラグインはクラスターとともにインストールされています。Amazon EC2 ノードをクラスターに追加すると、プラグインは追加する各 Amazon EC2 ノードに自動的にデプロイされます。プラグインでは次の IAM ポリシーのいずれかを IAM ロールにアタッチする必要があります。クラスターが IPv4 ファミリーを使用する場合はAmazonEKS_CNI_Policy マネージド IAM ポリシーを使用します。クラスターが IPv6 ファミリーを使用している場合、自ら作成した IAM ポリシーを使用します。

   ポリシーをアタッチする IAM ロールはノード IAM ロール、またはプラグインにのみ使用される専用ロールです。このロールにポリシーをアタッチすることをお勧めします。ロールの作成の詳細については、「the section called “IRSA 用の設定”」または「Amazon EKS ノードの IAM ロール」を参照してください。
3. AWS マネジメントコンソール を使用してクラスターをデプロイした場合、このステップはスキップできます。AWS マネジメントコンソールでは、デフォルトで、Amazon VPC CNI plugin for Kubernetes、CoreDNS、および kube-proxy Amazon EKS アドオンがデプロイされます。

`eksctl` または AWS CLI のいずれかを使用してクラスターをデプロイする場合、Amazon VPC CNI plugin for Kubernetes、CoreDNS、および `kube-proxy` セルフマネージド型アドオンがデプロイされます。クラスターと共に Amazon EKS アドオンにデプロイされる、Amazon VPC CNI plugin for Kubernetes、CoreDNS、および `kube-proxy` セルフマネージド型アドオンを移行できます。詳細については、「the section called “Amazon EKS アドオン”」を参照してください。

4. (オプション) まだ作成していない場合は、クラスターの Prometheus メトリクスを有効にできます。詳細については、Amazon Managed Service for Prometheus ユーザーガイドの「スクレイパーの作成」を参照してください。
5. Amazon EBS ボリュームを使用するクラスターにワークロードをデプロイする予定の場合、ワークロードをデプロイする前に、Amazon EBS CSI をクラスターにインストールする必要があります。

次のステップ

- クラスターを作成した IAM プリンシパルはクラスターにアクセスできる唯一のプリンシパルです。他の IAM プリンシパルに許可を付与して、クラスターにアクセスできるようにします。
- クラスターを作成した IAM プリンシパルが、前提条件で参照されている最低限の IAM 許可しか持たない場合、そのプリンシパルに Amazon EKS 許可を追加することができます。Amazon EKS 許可を IAM プリンシパルに付与する方法については、「the section called “IAM リファレンス”」を参照してください。
- クラスターを作成した IAM プリンシパル、またはその他のプリンシパルに Amazon EKS コンソールで Kubernetes リソースを表示させる場合は、必要なアクセス許可をエンティティに付与します。
- ノードと IAM プリンシパルが VPC 内からクラスターにアクセスできるようにする場合、クラスターのプライベートエンドポイントを有効にします。デフォルトではパブリックエンドポイントは有効です。プライベートエンドポイントを有効にした後、必要に応じてパブリックエンドポイントを無効にできます。詳細については、「the section called “クラスターエンドポイントのアクセス”」を参照してください。
- クラスターのシークレット暗号化を有効にする。
- クラスターのログ記録を設定する。
- クラスターにノードを追加する。

Amazon EKS プロビジョンドコントロールプレーン

概要

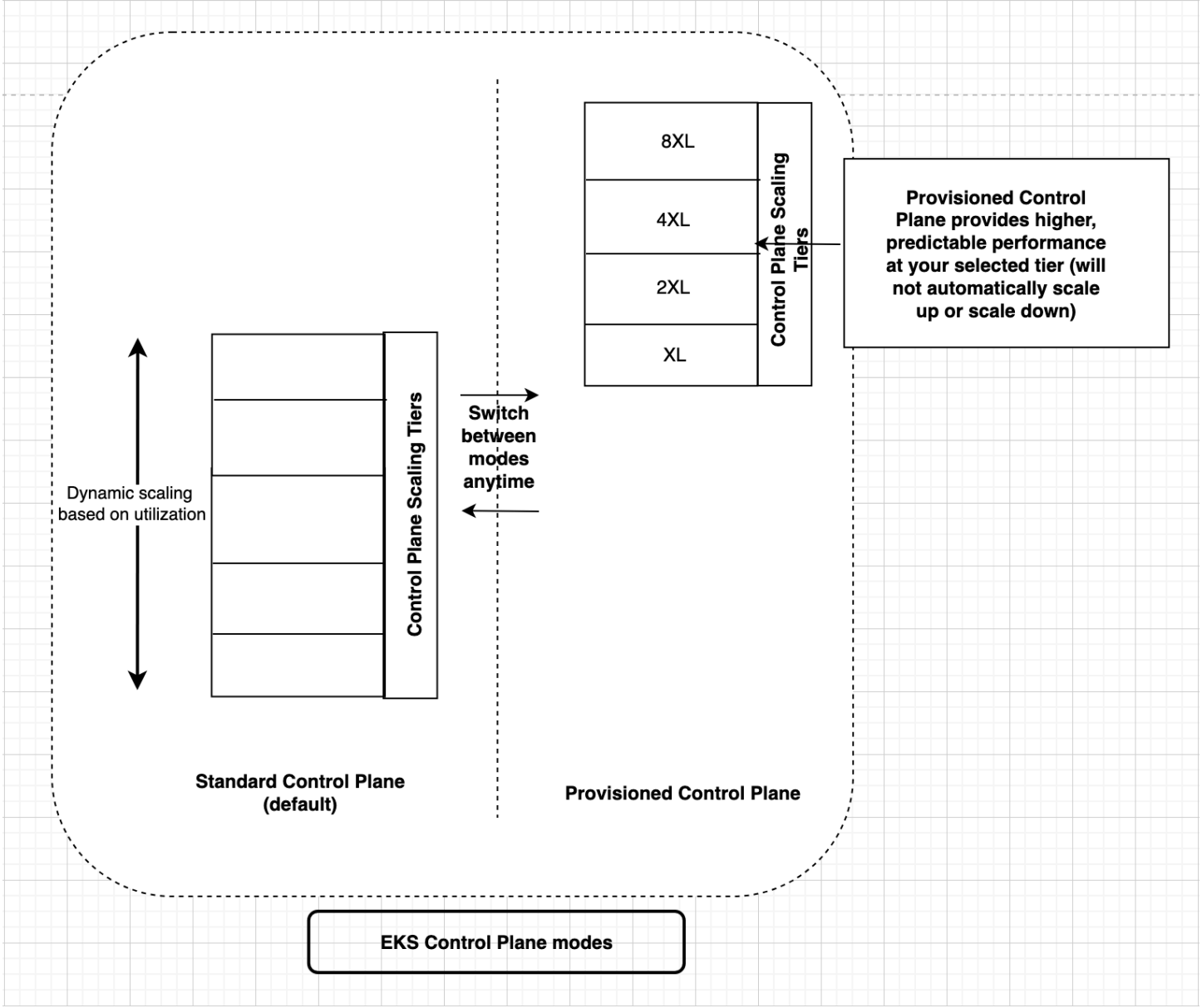
Amazon EKS プロビジョンドコントロールプレーンは、クラスター管理者が一連のスケーリング階層から選択し、選択した階層を指定して、クラスターのコントロールプレーンから非常に高く予測可能なパフォーマンスを実現できるようにする機能です。これにより、クラスター管理者は、コントロールプレーンが常に指定された容量でプロビジョニングされるようにできます。

Amazon EKS には、クラスターのコントロールプレーン用に 2 つのオペレーションモードが用意されています。デフォルトでは、Amazon EKS クラスターは スタンダードモード を使用します。このモードでは、コントロールプレーンはワークロードの需要に応じて自動的にスケールアップおよびスケールダウンします。スタンダードモードでは、ワークロードのニーズを満たすのに十分なコントロールプレーン容量が動的に割り当てられ、ほとんどのユースケースで推奨されるソリューションです。ただし、コントロールプレーンのスケーリングによるパフォーマンスの変動を一切許容できない特殊なワークロードや、非常に大量のコントロールプレーン容量を必要とするワークロードの場合は、オプションで プロビジョンドモード を使用できます。プロビジョンドモードでは、要求の厳しいワークロード要件に常に対応できるコントロールプレーン容量を事前に割り当てることができます。

> **Note**
>
> プロビジョンドモードは、デフォルトのスタンダードモードと共に追加のコントロールプレーンオペレーションモードです。プロビジョンドモードを導入しても、スタンダードモードの動作は変わりません。

EKS プロビジョンドコントロールプレーンを使用すると、クラスター管理者は必要なコントロールプレーン容量を事前にプロビジョニングできるため、常に利用可能なクラスターのコントロールプレーンから予測可能で高いパフォーマンスが得られます。また、EKS プロビジョンドコントロールプレーンは、ステージングから本番稼働、ディザスタリカバリサイトまで、環境間で同じコントロールプレーン容量をクラスター管理者がプロビジョニングできるようにします。これは、環境間で得られるコントロールプレーンのパフォーマンスに一貫性があり、予測可能であることを確実にするために重要なことです。最後に、EKS プロビジョンドコントロールプレーンでは、非常に高いレベルのコントロールプレーンパフォーマンスにアクセスできるため、Kubernetes で非常にスケーラブルな AI ワークロード、高性能コンピューティング、大規模なデータ処理ワークロードを実行できます。

既存および新規の Amazon EKS クラスターはすべて、デフォルトでスタンダードモードで動作します。コントロールプレーンから高い予測可能なパフォーマンスを必要とするクラスターでは、EKS プロビジョンドコントロールプレーン機能の使用をオプトインできます。標準または延長サポートの EKS 時間単位料金に加えて、特定のコントロールプレーンのスケーリング階層に対する時間単位料金が請求されます。料金に関する詳細については、「Amazon EKS の料金」を参照してください。



ユースケース

EKS プロビジョンドコントロールプレーンは、高い予測可能なコントロールプレーンのパフォーマンスがオペレーションにとって重要な特定のシナリオに対処するように設計されています。これらの

ユースケースを理解することで、EKS プロビジョンドコントロールプレーンがワークロードに適したソリューションかどうかを判断できます。

パフォーマンスクリティカルなワークロード – Kubernetes コントロールプレーンから最小のレイテンシーと最大のパフォーマンスを必要とするワークロードの場合、EKS プロビジョンドコントロールプレーンは、コントロールプレーンのスケーリングによるパフォーマンスの変動を排除する容量を提供します。

大規模にスケーラブルなワークロード – AI トレーニングや推論、高性能コンピューティング、大規模なデータ処理など、クラスターで実行されている多数のノードを必要とする非常にスケーラブルなワークロードを実行する場合、プロビジョンドコントロールプレーンは、これらの要求の厳しいワークロードをサポートするために必要なコントロールプレーン容量を提供します。

予想される需要の高いイベント – e コマースの販売やプロモーション、製品の発売、ホリデーシーズンのショッピング、スポーツやエンターテインメントの大規模なイベントなど、今後のイベントが原因でコントロールプレーンのリクエストが急増すると予想される場合、プロビジョンドコントロールプレーンでは、コントロールプレーンの容量を事前にスケールできます。このプロアクティブアプローチにより、自動スケーリングが需要に応答するのを待たずに、増加した負荷をコントロールプレーンが処理できるようになります。

ミッションクリティカルな高可用性 – 最高レベルの可用性を必要とするワークロードの場合、プロビジョンドコントロールプレーンは、1 分間隔で測定される 99.99% のサービスレベルアグリーメント (SLA) を提供します。この拡張 SLA は、5 分間隔で測定される標準コントロールプレーンモードの 99.95% SLA と比較して、より厳格な可用性保証を提供します。

環境の一貫性 – プロビジョンドコントロールプレーンを使用すると、ステージング環境と本番稼働環境間でコントロールプレーンの容量とパフォーマンスを照合できるため、本番稼働環境へのデプロイ前に潜在的な問題を特定できます。環境間で同じコントロールプレーン層を維持することで、テスト結果に本番稼働の動作を正確に反映し、ロールアウト中に発生するパフォーマンス関連の予期しない問題のリスクを減らすことができます。

ディザスタリカバリと事業継続性 – ディザスタリカバリのシナリオでは、プロビジョンドコントロールプレーンを使用すると、プライマリ環境と同じレベルの容量でフェイルオーバー環境をプロビジョニングできます。これにより、ディザスタリカバリクラスターは、アクティブ化された瞬間から本番稼働クラスターと同じコントロールプレーンのパフォーマンス特性を持つため、フェイルオーバーイベント中の中断を最小限に抑え、迅速な復旧が可能になります。

コントロールプレーンのスケーリング階層

EKS プロビジョンドコントロールプレーンは、T シャツサイズ (XL、2XL、4XL、8XL) で命名されたスケーリング階層を提供します。各階層は、クラスターのコントロールプレーンのパフォーマンス特性を決定する 4 つの主要な属性によって容量を定義します。これらの属性を理解することで、ワークロード要件に適した階層を選択できます。

API リクエスト同時実行数 は、Kubernetes コントロールプレーンの API サーバーが同時に処理できるリクエストの数を測定します。これは、高スループットワークロードにとって重要です。

ポッドスケジューリングレート は、デフォルトの Kubernetes スケジューラがノードでポッドをスケジュールできる速度を示し、1 秒あたりのポッド数で測定されます。

クラスターデータベースのサイズ は、クラスターの状態/メタデータを保持するデータベースである etcd に割り当てられたストレージスペースを示します。

SLA (サービスレベルアグリーメント) は、EKS クラスターの Kubernetes エンドポイントの月間稼働率に関する保証を示します。

プロビジョンドコントロールプレーンを使用して特定のスケーリング階層でクラスターのコントロールプレーンをプロビジョニングすると、EKS はクラスターのコントロールプレーンがその階層に対応する制限を維持するようにします。コントロールプレーンのスケーリング階層の制限は、次の表に示すように、Kubernetes のバージョンによって異なります。

EKS v1.30 ~ v1.33

プロビジョンドコントロールプレーンのスケーリング階層	API リクエスト同時実行数 (同時実行枠)	ポッドスケジューリングレート (ポッド/秒)	クラスターデータベースサイズ (GB)	SLA (1 分間隔で測定)
XL	1700	167	16	99.99%
2XL	3400	283	16	99.99%
4XL	6800	400	16	99.99%
8XL	13600	400	16	99.99%

EKS v1.34 以降

プロビジョンドコントロールプレーンのスケーリング階層	API リクエスト同時実行数 (同時実行枠)	ポッドスケジューリングレート (ポッド/秒)	クラスターデータベースサイズ (GB)	SLA (1 分間隔で測定)
XL	2000	167	16	99.99%
2XL	4000	283	16	99.99%
4XL	8000	400	16	99.99%
8XL	16000	400	16	99.99%

スケーリング階層の値をプログラムで取得する

`DescribeClusterVersions` API を使用して、Kubernetes バージョンごとにスケーリング階層の属性とその値を取得できます。これは、階層の選択を自動化する場合や、バージョン間で階層の機能をプログラムで比較する必要がある場合に役立ちます。詳細については、Amazon EKS API リファレンスの「DescribeClusterVersions」を参照してください。

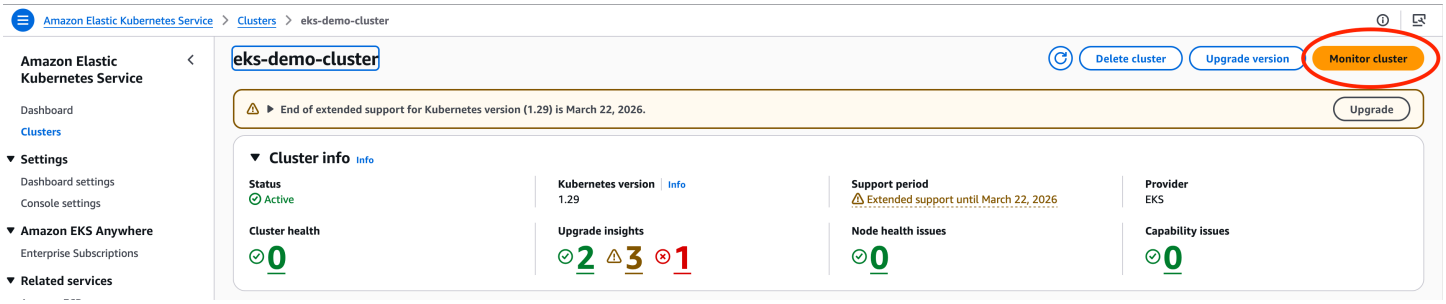
コントロールプレーンのスケーリング階層の使用率のモニタリング

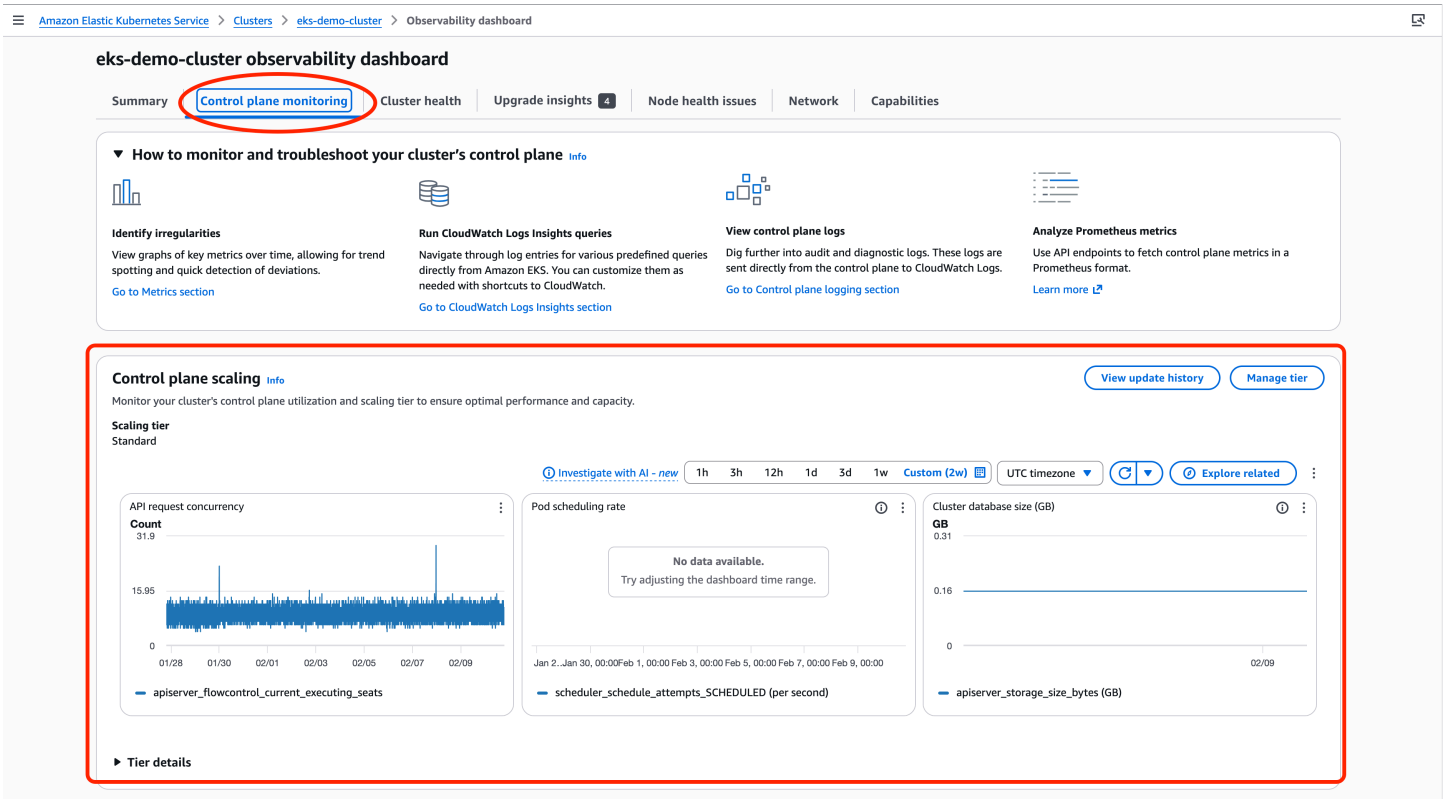
Amazon EKS には、コントロールプレーンの階層使用率のモニタリングに役立つメトリクスがいくつか用意されています。これらのメトリクスは Amazon CloudWatch メトリクス として公開されており、CloudWatch および EKS コンソールからアクセスできます。さらに、これらのメトリクスは EKS クラスターの Prometheus エンドポイントからスクレイピングできます (こちら を参照)。

	Prometheus メトリクス	CloudWatch メトリクス
API リクエスト同時実行数	apiserver_flowcontrol_current_executing_seats	apiserver_flowcontrol_current_executing_seats
ポッドのスケジュールレート	scheduler_schedule_attempts_total	scheduler_schedule_attempts_total、scheduler_schedule_attempts_SCHEDULED、

	Prometheus メトリクス	CloudWatch メトリクス
		scheduler_schedule_attempts_UNSCHEDULABLE
クラスターデータベースサイズ	apiserver_storage_size_bytes (注: etcd_mvcc_db_total_size_in_use_in_bytes は、2026 年後半までにすべての EKS クラスターに Prometheus メトリクスとして展開される予定です)	etcd_mvcc_db_total_size_in_use_in_bytes

Amazon EKS コンソールでコントロールプレーンの使用率を表示できます。クラスターの概要ページから、[クラスターをモニタリング] を選択してオブザーバビリティダッシュボードにアクセスし、[コントロールプレーンのモニタリング] タブを選択して、[コントロールプレーンのスケーリング]セクションでコントロールプレーンの使用率を表示します。





階層容量と実際のパフォーマンスについて

プロビジョンドコントロールプレーンのスケーリング階層を選択すると、階層属性は Amazon EKS がコントロールプレーンに適用する基盤となる設定を表します。ただし、実際のパフォーマンスは、特定のワークロードパターン、設定、Kubernetes のベストプラクティスへの準拠によって異なります。例えば、4XL 階層では API Priority and Fairness (APF) が 6,800 の同時リクエスト枠で設定されていますが、コントロールプレーンから得られる実際のリクエストスループットは、実行されるオペレーションのタイプによって異なります。例えば、Kubernetes は get リクエストよりも list リクエストに対して大きなペナルティを課すため、コントロールプレーンで同時に処理される list リクエストの実効数は、get リクエストより少なくなります。詳しくは、EKS ベストプラクティスガイドの「API の優先順位と公平性」を参照してください。同様に、4XL 階層ではデフォルトのスケジューラ QPS が 400 に設定されていますが、実際のポッドスケジューリングレートは、ノードの準備状況やスケジューリング時のヘルスなどの要因によって異なります。最適なパフォーマンスを実現するには、アプリケーションが Kubernetes のベストプラクティスに従っており、ワークロードの特性に合わせて適切に設定されていることを確認します。詳細については、EKS ベストプラクティスガイドの「スケーラビリティのベストプラクティス」参照してください。

Horizontal Pod Autoscaler の同期同時実行性

コントロールプレーンスケーリング階層属性に加えて、EKS プロビジョンドコントロールプレーンクラスターは、より高い Horizontal Pod Autoscaler (HPA) 同期同時実行性で調整されます。HPA 同期同時実行性は、Kubernetes コントローラーマネージャーが並行して処理する HPA オブジェクトの数です。アップストリーム Kubernetes のデフォルトは 5 です。高い同時実行性では、コントロールプレーンはより多くの HPA オブジェクトを同時に調整できるため、負荷の変化と対応するスケーリングアクションの間の時間を短縮できます。これは、多数の HPA オブジェクトを実行するクラスターにとって有益であり、オブジェクトを連続して処理すると自動スケーリングが遅れる可能性があります。

各プロビジョンドコントロールプレーンスケーリング階層は、次の HPA 同期同時実行性で設定されます。

プロビジョンドコントロールプレーンのスケーリング階層	HPA 同期同時実行性
XL	50
2XL	100
4XL	200
8XL	200

HPA コントローラーの Kubernetes コントローラーマネージャーのワークキューメトリクスを通じて、より高い HPA 同期同時実行性の影響を確認できます。

- `workqueue_depth{name="horizontalpodautoscaler"}` – HPA コントローラーによって処理されるのを待機している HPA オブジェクトの数。同時実行性が高いほど、より多くのワーカーがこのキューを並行してドレインするため、深度は低く安定したままになります (各調整期間内にゼロに達するのが理想的です)。ワークキューの深度がゼロに近い場合は、コントロールプレーンがクラスター内の HPA オブジェクトに追いついており、自動スケーリングの決定がバックログによって遅延しないことを示します。

観察している実際の自動スケーリングの応答性は、コントロールプレーン以外の要因にも依存します。各 HPA 調整は、スケーリングの決定を左右するメトリクスを取得します。通常は CPU とメモリの使用率に関する Kubernetes メトリクスサーバー、またはカスタムまたは外部メトリクスアダプ

ターから取得します。クラスター内の HPA オブジェクトの数とこれらのメトリクスルックアップのレイテンシーはどちらも、コントローラーがキューをどの程度速く処理できるかに影響を与えます。メトリクスソースがリクエストに十分に迅速に対応できない場合、並列処理される数に関係なく、調整が遅くなります。リクエストボリュームに対応し、メトリクスクエリを効率的にスコープするために十分な Metrics Server レプリカを実行すると、高い HPA 同期同時実行性の利点が最大限に得られます。Horizontal Pod Autoscaler の仕組みとメトリクスの取得場所の詳細については、Kubernetes ウェブサイトの「Horizontal Pod Autoscaling」を参照してください。

考慮事項

- スタンダードコントロールプレーン容量 – EKS スタンダードコントロールプレーンモードは、価格対パフォーマンス比に優れており、ほとんどのユースケースに推奨されるオプションです。ただし、コントロールプレーンのスケーリングによるパフォーマンスの変動を一切許容できない特殊なワークロードや、非常に大量のコントロールプレーン容量を必要とするワークロードの場合は、オプションでプロビジョンドモードの使用を検討できます。
- オプトインが必要 – 既存のクラスターは、スタンダードコントロールプレーンから、より高い料金 の EKS プロビジョンドコントロールプレーン階層へ自動的にスケールアップされることはありません。新しい EKS プロビジョンドコントロールプレーンスケーリング階層のいずれかに明示的にオプトインする必要があります。
- 終了制限 – スタンダードコントロールプレーンモードは、最大 8 GB のクラスターデータベース (etcd) サイズをサポートします。プロビジョンドモードの使用中にクラスターのデータベースサイズが 8 GB を超える場合は、データベースサイズを 8 GB 未満に減らすまで、スタンダードモードに戻すことはできません。例えば、プロビジョンドモードで 14 GB のデータベースストレージを使用している場合は、スタンダードモードに戻る前に、まずデータベース使用率を 8 GB 未満に減らす必要があります。
- 自動階層スケーリングなし – EKS プロビジョンドコントロールプレーンは、階層間で自動的にスケーリングされません。スケーリング階層を選択すると、クラスターのコントロールプレーンはその階層に固定されたままになり、一貫した予測可能なパフォーマンスが確保されます。ただし、階層使用率メトリクスをモニタリングし、EKS プロビジョンドコントロールプレーン API を使用して、定義したしきい値をこれらのメトリクスが超えたときにスケールアップまたはスケールダウンすることで、独自の自動スケーリングソリューションを柔軟に実装できるため、スケーリング戦略とコスト最適化を完全に制御できます。
- 現在の階層の表示 – Amazon EKS コンソール、Amazon Web Services CLI、または API を使用して、現在のコントロールプレーンのスケーリング階層を表示できます。CLI では、次の `describe-cluster` コマンドを実行できます。`aws eks describe-cluster --name cluster-name`

- 階層の移行時間 – Amazon EKS コンソール、Amazon EKS API、または CLI を使用して、スケーリング階層を終了または移動できます。Amazon EKS では、`ScalingTierConfigUpdate` という新しいクラスター更新タイプが導入されています。この更新タイプでは、内容を確認し、移行の進行状況をモニタリングできます。階層変更コマンドを実行した後、クラスターの更新を一覧表示すると、ステータスが `Updating` の `ScalingTierConfigUpdate` タイプの新しい更新が表示されます。更新が完了するとステータスは `Successful` に変わり、エラーが発生した場合は `Failed` に変わります。更新のエラーフィールドには、失敗の理由が示されます。階層を切り替える頻度に制限はありません。コントロールプレーン階層の変更には数分かかります。このプロセスでは、EKS が古い API サーバーを終了する前に新しい API サーバーを起動するため、API サーバーのダウンタイムはありません。
- 最適な階層の選択 – クラスターに最適なプロビジョンドコントロールプレーンのスケーリング階層を決定するために、最上位の階層 (8XL) でクラスターをプロビジョニングして負荷テストを実行できます。次に、負荷テストを実行して、クラスターのコントロールプレーンに対するピーク需要をシミュレートします。ピーク負荷時のコントロールプレーン階層使用率メトリクスを観察し、これらの観測値を指針として使用し、プロビジョンドモードに適した階層を選択します。
- プロビジョンドコントロールプレーンの料金 – クラスターが稼働しているプロビジョンドコントロールプレーンのスケーリング階層について、時間単位の料金が請求されます。これは、標準または延長サポートの時間単位の料金に追加されます。詳細については、「Amazon EKS の料金」ページ を参照してください。
- より大きなスケーリング階層 – 8XL を超えるスケーリング階層でクラスターを実行する場合は、Amazon Web Services アカウントチームに連絡して、追加の料金情報を確認してください。
- Kubernetes バージョンとリージョンのサポート – EKS プロビジョンドコントロールプレーンは、すべての Amazon Web Services 商用リージョン、GovCloud リージョン、中国リージョンでサポートされています。プロビジョンドコントロールプレーンは EKS v1.28 以降で動作します。
- より高い SLA – EKS プロビジョンドコントロールプレーンは、標準コントロールプレーンよりも高い EKS SLA を提供します。詳細については、EKS SLA ページ を参照してください。

Amazon EKS プロビジョンドコントロールプレーンの開始方法

このガイドでは、AWS CLI と AWS マネジメントコンソールを使用して、EKS プロビジョンドコントロールプレーンをセットアップして使用する手順について説明します。

前提条件

開始する前に、以下があることを確認してください。

- AWS CLI – Amazon EKS など AWS のサービスを操作するためのコマンドラインツールです。詳細については AWS コマンドラインインターフェイスユーザーガイドの「インストール」を参照してください。AWS CLI のインストール後は設定も行っておくことをお勧めします。詳細については AWS コマンドラインインターフェイスユーザーガイドの「aws configure を使用したクイック設定」を参照してください。なお、このページにある [update-kubeconfig] オプションを使用するにはAWS CLI v2 が必要です。
- 必要な IAM アクセス許可 – 使用している IAM セキュリティプリンシパルには Amazon EKS の IAM ロール、サービスにリンクされたロール、AWS クラウドフォーメーション、VPC、その関連リソースを操作するための権限が必要になります。詳細については IAM ユーザーガイドの「アクション」および「サービスにリンクされたロールの使用」を参照してください。このガイドのすべてのステップは1 つのユーザーとして実行する必要があります。現在のユーザーを確認するには次のコマンドを実行してください：

```
aws sts get-caller-identity
```

> **Note**
>
> このトピック内のステップはバッシュ シェル内で実行することが推奨されます。Bash シェルを使用していない場合、行継続文字や、変数の設定と使用に関する方法など、一部のスクリプトコマンドのためにシェルの調整が必要となります。さらに、シェルの引用規則とエスケープ規則は異なる場合があります。詳細については AWS コマンドラインインターフェイスのユーザーガイドの「AWS CLI での文字列への引用符の使用」を参照してください。

EKS プロビジョンドコントロールプレーン - AWS CLI

EKS プロビジョンドコントロールプレーンスケーリング階層を使用してクラスターを作成する

```
aws eks create-cluster --name prod-cluster \
--role-arn arn:aws:iam::012345678910:role/eks-service-role-AWSServiceRoleForAmazonEKS-
J7ONKE3BQ4PI \
--resources-vpc-config subnetIds=subnet-6782e71e,subnet-
e7e761ac,securityGroupIds=sg-6979fe18 \
--control-plane-scaling-config tier=tier-xl
```

レスポンス:

```
{
    "cluster": {
        "name": "my-eks-cluster",
        "arn": "arn:aws:eks:us-east-2:111122223333:cluster/my-eks-cluster",
        "createdAt": "2024-03-14T11:31:44.348000-04:00",
        "version": "1.26",
        "endpoint": "https://JSA79429HJDASKJDJ8223829MNDNASW.yl4.us-
east-2.eks.amazonaws.com",
        "roleArn": "arn:aws:iam::111122223333:role/eksctl-my-eks-cluster-cluster-
ServiceRole-zMF6CBakwwbW",
        "resourcesVpcConfig": {
            "subnetIds": [
                "subnet-0fb75d2d8401716e7",
                "subnet-02184492f67a3d0f9",
                "subnet-04098063527aab776",
                "subnet-0e2907431c9988b72",
                "subnet-04ad87f71c6e5ab4d",
                "subnet-09d912bb63ef21b9a"
            ],
            "securityGroupIds": [
                "sg-0c1327f6270afbb36"
            ],
            "clusterSecurityGroupId": "sg-01c84d09d70f39a7f",
            "vpcId": "vpc-0012b8e1cc0abb17d",
            "endpointPublicAccess": true,
            "endpointPrivateAccess": true,
            "publicAccessCidrs": [
                "22.19.18.2/32"
            ]
        },
        "controlPlaneScalingConfig": {
            "tier": "tier-xl"
        },
        "kubernetesNetworkConfig": {
            "serviceIpv4Cidr": "10.100.0.0/16",
            "ipFamily": "ipv4"
        },
        "logging": {
            "clusterLogging": [
                {
                    "types": [
                        "api",
                        "audit",
```

```
                        "authenticator",
                        "controllerManager",
                        "scheduler"
                    ],
                    "enabled": true
                }
            ]
        },
        "identity": {
            "oidc": {
                "issuer": "https://oidc.eks.us-east-2.amazonaws.com/id/
JSA79429HJDASKJDJ8223829MNDNASW"
            }
        },
        "status": "CREATING",
        "certificateAuthority": {
            "data": "CA_DATA_STRING..."
        },
        "platformVersion": "eks.14",
        "tags": {
            "aws:cloudformation:stack-name": "eksctl-my-eks-cluster-cluster",
            "alpha.eksctl.io/cluster-name": "my-eks-cluster",
            "karpenter.sh/discovery": "my-eks-cluster",
            "aws:cloudformation:stack-id": "arn:aws:cloudformation:us-
east-2:111122223333:stack/eksctl-my-eks-cluster-cluster/e752ea00-e217-11ee-
beae-0a9599c8c7ed",
            "auto-delete": "no",
            "eksctl.cluster.k8s.io/v1alpha1/cluster-name": "my-eks-cluster",
            "EKS-Cluster-Name": "my-eks-cluster",
            "alpha.eksctl.io/cluster-oidc-enabled": "true",
            "aws:cloudformation:logical-id": "ControlPlane",
            "alpha.eksctl.io/eksctl-version": "0.173.0-dev
+a7ee89342.2024-03-01T03:40:57Z",
            "Name": "eksctl-my-eks-cluster-cluster/ControlPlane"
        },
        "health": {
            "issues": []
        },
        "accessConfig": {
            "authenticationMode": "API_AND_CONFIG_MAP"
        }
    }
}
```

クラスターのコントロールプレーンスケーリング階層を表示する

```
aws eks describe-cluster --name prod-cluster
```

レスポンス:

```
{
    "cluster": {
        "name": "my-eks-cluster",
        "arn": "arn:aws:eks:us-east-2:111122223333:cluster/my-eks-cluster",
        "createdAt": "2024-03-14T11:31:44.348000-04:00",
        "version": "1.26",
        "endpoint": "https://JSA79429HJDASKJDJ8223829MNDNASW.yl4.us-
east-2.eks.amazonaws.com",
        "roleArn": "arn:aws:iam::111122223333:role/eksctl-my-eks-cluster-cluster-
ServiceRole-zMF6CBakwwbW",
        "resourcesVpcConfig": {
            "subnetIds": [
                "subnet-0fb75d2d8401716e7",
                "subnet-02184492f67a3d0f9",
                "subnet-04098063527aab776",
                "subnet-0e2907431c9988b72",
                "subnet-04ad87f71c6e5ab4d",
                "subnet-09d912bb63ef21b9a"
            ],
            "securityGroupIds": [
                "sg-0c1327f6270afbb36"
            ],
            "clusterSecurityGroupId": "sg-01c84d09d70f39a7f",
            "vpcId": "vpc-0012b8e1cc0abb17d",
            "endpointPublicAccess": true,
            "endpointPrivateAccess": true,
            "publicAccessCidrs": [
                "22.19.18.2/32"
            ]
        },
        "controlPlaneScalingConfig": {
            "tier": "tier-xl"
        },
        "kubernetesNetworkConfig": {
            "serviceIpv4Cidr": "10.100.0.0/16",
            "ipFamily": "ipv4"
        },
```

```
        "logging": {
            "clusterLogging": [
                {
                    "types": [
                        "api",
                        "audit",
                        "authenticator",
                        "controllerManager",
                        "scheduler"
                    ],
                    "enabled": true
                }
            ]
        },
        "identity": {
            "oidc": {
                "issuer": "https://oidc.eks.us-east-2.amazonaws.com/id/
JSA79429HJDASKJDJ8223829MNDNASW"
            }
        },
        "status": "ACTIVE",
        "certificateAuthority": {
            "data": "CA_DATA_STRING..."
        },
        "platformVersion": "eks.14",
        "tags": {
            "aws:cloudformation:stack-name": "eksctl-my-eks-cluster-cluster",
            "alpha.eksctl.io/cluster-name": "my-eks-cluster",
            "karpenter.sh/discovery": "my-eks-cluster",
            "aws:cloudformation:stack-id": "arn:aws:cloudformation:us-
east-2:111122223333:stack/eksctl-my-eks-cluster-cluster/e752ea00-e217-11ee-
beae-0a9599c8c7ed",
            "auto-delete": "no",
            "eksctl.cluster.k8s.io/v1alpha1/cluster-name": "my-eks-cluster",
            "EKS-Cluster-Name": "my-eks-cluster",
            "alpha.eksctl.io/cluster-oidc-enabled": "true",
            "aws:cloudformation:logical-id": "ControlPlane",
            "alpha.eksctl.io/eksctl-version": "0.173.0-dev
+a7ee89342.2024-03-01T03:40:57Z",
            "Name": "eksctl-my-eks-cluster-cluster/ControlPlane"
        },
        "health": {
            "issues": []
        },
```

```
        "accessConfig": {
            "authenticationMode": "API_AND_CONFIG_MAP"
        }
    }
}
```

EKS プロビジョンドコントロールプレーンを使用するようにクラスターを更新する

```
aws eks update-cluster-config --name prod-cluster \
--control-plane-scaling-config tier=tier-2xl
```

レスポンス:

```
{
    "update": {
        "id": "7551c64b-1d27-4b1e-9f8e-c45f056eb6fd",
        "status": "InProgress",
        "type": "ScalingTierConfigUpdate",
        "params": [
            {
                "type": "UpdatedTier",
                "value": "tier-2xl"
            },
            {
                "type": "PreviousTier",
                "value": "tier-xl"
            }
        ],
        "createdAt": 1565807210.37,
        "errors": []
    }
}
```

コントロールプレーンスケーリングの更新を表示する

```
aws eks list-updates --name example
```

レスポンス:

```
{
    "updateIds": [
```

```
        "7551c64b-1d27-4b1e-9f8e-c45f056eb6fd1"
    ]
}
```

プロビジョンドコントロールプレーンから標準コントロールプレーンに移行する

```
aws eks update-cluster-config --name prod-cluster \
--control-plane-scaling-config tier=standard
```

レスポンス:

```
{
    "update": {
        "id": "7551c64b-1d27-4b1e-9f8e-c45f056eb6fd",
        "status": "InProgress",
        "type": "ScalingTierConfigUpdate",
        "params": [
            {
                "type": "UpdatedTier",
                "value": "standard"
            },
            {
                "type": "PreviousTier",
                "value": "tier-2xl"
            }
        ],
        "createdAt": 1565807210.37,
        "errors": []
    }
}
```

EKS プロビジョンドコントロールプレーン - AWS マネジメントコンソール

1. Amazon EKS コンソール を開きます。
2. [クラスターを作成] を選択します。
3. [設定オプション] で [カスタム設定] を選択します。
4. 下にスクロールして、[コントロールプレーンスケーリング階層] に移動します。[スケーリング階層を使用] を選択して、プロビジョンドコントロールプレーンを有効にします。
5. XL、2XL、4XL、8XL などのさまざまなスケーリング階層オプションから、クラスターにプロビジョニングするコントロールプレーンスケーリング階層を選択します。

6. 必要に応じて、他のクラスター設定オプションを選択します。最後のステップで、[クラスターの作成] を選択します。クラスターの作成が完了するまでに数分かかる場合があります。

高度な Kubernetes コントロールプレーン設定

概要

Amazon EKS は、API サーバー、スケジューラー、コントローラーマネージャーを含む、クラスターの Kubernetes コントロールプレーンを管理します。EKS はこれらのコンポーネントを、大半のワークロードで適切に機能するデフォルトのアップストリーム Kubernetes 設定で実行しており、ほとんどのクラスターでは変更する必要はありません。ただし、ワークロードによっては別のコントロールプレーン設定が有利になる場合があります。コンピューティングコストを削減するためにスケジューラーでポッドをより少ないノードに詰め込みたい、クラスターデータベース (etcd) の増大を抑えるために Kubernetes イベントの保持期間を短くしたい、または自動スケーリングの判断をより頻繁に評価したいと考える場合があります。

高度な Kubernetes コントロールプレーン設定では、これらのパラメータをクラスターに直接設定します。EKS はそれらをコントロールプレーンに適用し、クラスターは同じ可用性とパフォーマンス特性で動作を継続します。

これらは高度な設定です。各設定パラメータは、クラスターで実行されるワークロードに対して Kubernetes コントロールプレーンのコアコンポーネントの動作を変更するものであり、適切な値はワークロードによって異なります。パラメータを変更する前に、以下のセクションでその考慮事項を読み、非本番クラスターで変更をテストしてください。

高度なコントロールプレーン設定パラメータは、クラスターの作成時に設定することも、既存のクラスターでいつでも更新することもできます。この機能では、既存の `CreateCluster` および `UpdateClusterConfig` オペレーションを新しいパラメータとともに使用するため、AWS マネジメントコンソール、AWS CLI、AWS SDK、または AWS CloudFormation を通じて設定できます。EKS は適用前に各設定を検証し、変更を AWS CloudTrail に記録します。

高度なコントロールプレーンパラメータは、クラスター全体およびそこで実行されるすべてのワークロードに適用されます。これらを個々の名前空間やワークロードに限定することはできません。EKS は各パラメータを検証済みの範囲に制限します。各パラメータでサポートされる値は、以下のセクションでそのパラメータとともに示されています。

サポートされている Kubernetes コントロールプレーンパラメータ

Amazon EKS は以下のパラメータをサポートしています。各パラメータはコントロールプレーンコンポーネントに属し、そのコンポーネントの設定フィールド (kubeSchedulerConfig、kubeControllerManagerConfig、または kubeApiServerConfig) を通じて設定します。

コンポーネント	パラメータ	サポートされる値	デフォルト	プロビジョンドコントロールプレーンが必要
kube-scheduler	nodeResourcesFit.scoringStrategy	LeastAllocated , MostAllocated	LeastAllocated (cpu: 1 および memory: 1 を含む)	いいえ
kube-controller-manager	horizontalPodAutoscalerControllerConfig.horizontalPodAutoscalerSyncPeriod	10s ~ 15s	15s (秒)	はい
kube-controller-manager	podGcControllerConfig.terminatedPodGcThreshold	10000 ~ 12500	12500	はい
kube-apiserver	eventTtl	10m ~ 60m	60m (分)	いいえ
kube-apiserver	serviceNodePortRange	minPort および maxPort(1026 の間)	minPort: 30000, maxPort: 32767	いいえ

このトピックのデフォルト値およびサポートされている値は、公開時点で利用可能な Kubernetes バージョン (EKS v1.31 以降) に適用され、後のバージョンで変更される可能性があります。DescribeClusterVersions オペレーションは、各パラメータおよび Kubernetes バージョンの現在のデフォルト値とサポートされている値を報告するため、複数のバージョンにまたがってクラスターを管理する場合やクラスター設定を自動化する場合は、これを信頼できる情報源として使用してください。詳細については、「the section called “コントロールプレーンパラメータを設定する”」を参照してください。

以下のセクションでは、各パラメータ、それを変更するタイミング、および変更前に考慮すべき事項について説明します。

スケジューラー: ノードリソースの適合

スケジューラーは 2 つのフェーズでポッドをノードに割り当てます。最初にポッドを実行できるノードをフィルタリングし、次に残った候補をスコアリングして、最もスコアの高いノードにポッドを配置します。nodeResourcesFit プラグインは、ノードにポッドが要求するリソースがあるかどうかを確認し、スコアリング戦略に従ってノードをスコアリングします。

フィールド	説明	サポートされる値	デフォルト
nodeResourcesFit.scoringStrategy.type	リソース割り当てによってノードをスコアリングするために使用される戦略。	LeastAllocated , MostAllocated	LeastAllocated
nodeResourcesFit.scoringStrategy.resources	スコアリング時に考慮されるリソース。それぞれに相対的な重みが付きます。	cpu、memory、nvm/gpu、aws.amazon.com/neuron 、aws.amazon.com/neuroncore 。重みは 1～100 の範囲です。	cpu: 1, memory: 1

LeastAllocated はリソース割り当ての少ないノードを優先し、クラスター内のノード全体にポッドを分散させ、各ノードに余裕を残します。これは Kubernetes のデフォルトの動作であり、既存のポッドの増加を吸収できる容量をすべてのノードで利用可能にしたい場合に適した選択です。

MostAllocated はリソース割り当てが既に多いノードを優先し、より少ないノードにポッドを詰め込みます。ワークロードが占有する合計容量が少なくなるため、より少ない数のノードで実行でき、コンピューティング費用を削減できます。時間の経過とともに、この詰め込み動作により使用率の低いノードに新しいワークロードが割り当てられなくなるため、統合をサポートするノードプールはそれらを削除できます。

EKS は LeastAllocated および MostAllocated 戦略をサポートしています。アップストリーム Kubernetes の RequestedToCapacityRatio 戦略はサポートされていません。

リソースの重み

必要に応じて、スコアリングの判断でどのリソースを最も重視するかに影響を与えるカスタム重みを持つ resources 配列を指定できます。これは、特定のリソースがクラスターの制約となっている場合に役立ちます。例えば、アクセラレーター (GPU) が不足しているリソースであるクラスターでは、nvidia.com/gpu の重みを CPU やメモリよりも高くすると、アクセラレーターを要求するポッドが、既に部分的に使用されているノードに集中します。

重みは絶対的ではなく相対的です。cpu: 100 と memory: 1 を設定しても、スケジューラーがメモリを無視するわけではありません。スコアリング式において、CPU をメモリより 100 倍重く評価します。すべての候補ノードの CPU 可用性が同一であれば、CPU ではそれらを区別できなくなり、スコアリングは事実上メモリに移ります。

リソースを省略することは、低い重みを付けることとは異なります。resources 配列を指定すると、リストしたリソースのみがスコアリングされます。除外したリソースは、計算から完全に除外されます。例えば、cpu: 100 に memory のエントリがない場合、CPU のみでノードをスコアリングし、メモリの可用性は結果に影響しません。リソースを計算に含めたまま影響を減らすには、省略するのではなく、低い重みを付けてリストします。

nvidia.com/gpu などのアクセラレーターリソースに重みを付けると、そのリソースに対して実際に resources.requests を宣言しているポッドのスコアリングにのみ影響します。アクセラレーターを要求しないポッドは、アクセラレーターの重みの影響を受けません。

3 つのアクセラレーターリソースは Kubernetes の拡張リソースであり、組み込みではなく、プラグインによって Kubernetes にアドバタイズされるノードレベルのリソースです。これらは、デバイスプラグインがデバイスプラグイン API を通じて kubelet にアドバタイズした場合にのみスコアリングされます。デバイスドライバーだけによってノードで利用可能になったリソー

スは、nodeResourcesFit プラグインからは認識されず、スコアリングされません。Dynamic Resource Allocation (DRA) を通じて管理されるリソースは、別のプラグインによってスケジュールされ、nodeResourcesFit のスコアリングには含まれないため、DRA を有効にしてもこのパラメータの動作は変わりません。NVIDIA デバイスプラグインの設定の詳細については、「NVIDIA DRA とデバイスプラグイン」を参照してください。Neuron デバイスの設定の詳細については、「Neuron デバイス管理」を参照してください。

スコアリング戦略は、スケジューラーが各ノードのスコアを計算するために使用する複数の入力の 1 つです。スケジューラーがノードをフィルタリングおよびスコアリングする方法の詳細については、Kubernetes ドキュメントの「スケジューリングフレームワーク」を参照してください。

スコアリング戦略に関する考慮事項

- 実行中のポッドは移動されません。Kubernetes スケジューラーは、既に実行中のポッドを再配置することはありません。スコアリング戦略の変更は今後のスケジューリング判断にのみ影響し、既存のポッドの配置は永続的です。既に実行中のポッドのバランスを再調整するには、ポッドを エビクションまたは再起動します。
- フィルタリング動作は変わりません。スコアリング戦略は、スケジューラーが選好度に応じてノードをランク付けするスコアリングフェーズにのみ影響します。ポッドがノードで実行できるかどうかを判断するフィルタリングフェーズは変更されません。ノードに適合しないポッドは、どちらの戦略でもそのノードにスケジュールされません。
- **MostAllocated** は障害の影響範囲を集中させます。ワークロードをより少ないノードに詰め込むと、ノードが異常になった場合、インスタンスが廃止された場合、またはアベイラビリティーゾーンが中断された場合に、一度に影響を受けるポッドが増えます。ポッドの入れ替わりが激しい状況では、高密度に詰め込まれたノードもより早くいっぱいになり、新しい容量がプロビジョニングされる間、ポッドが Pending 状態のままになる可能性があります。
- スケジューラーとノード管理は異なるレイヤーで動作します。スコアリング戦略は、既にポッドを実行できるノードの間でポッドがどこに配置されるかに影響します。EKS Auto Mode または Karpenter がノードをプロビジョニングまたは削除する方法は変わりません。設定を変更する前に、ワークロードの組み合わせた動作を検証してください。

コントローラーマネージャー: Horizontal Pod Autoscaler の同期期間

コントローラーマネージャーは、Horizontal Pod Autoscaler (HPA) コントローラーを含む、クラスターの状態を望ましい状態に近づける Kubernetes コントローラーを実行します。各サイクルで、HPA コントローラーは各 HorizontalPodAutoscaler オブジェクトのメトリクスを取得し、望ましいレプリカ数を計算し、数が変更されていればターゲットワークロードを更新します。

フィールド	説明	サポートされる値	デフォルト
horizontalPodAutoscalerControllerConfig.horizontalPodAutoscalerSyncPeriod	HPA コントローラーがスケーリングの判断を評価する頻度。	10s ~ 15s	15s

同期期間を短縮すると、負荷の増加後に容量を追加するまで 1 サイクル全体を待つのではなく、ワークロードがより早くスケールします。

このパラメータを設定するには、クラスターが Amazon EKS プロビジョンドコントロールプレーン上にある必要があります。間隔を短縮すると、HPA コントローラーがクラスター内のすべての HorizontalPodAutoscaler オブジェクトを調整する頻度が高まり、より多くの API リクエストが生成されます。各調整では少なくとも 1 つの API リクエストを消費し、レプリカ数を変更する調整ではさらに 2 つ消費します。プロビジョンドコントロールプレーンクラスターは、要求の厳しいワークロードに常に備えたコントロールプレーン容量を事前に割り当てるため、その追加負荷を吸収できるようにサイズ設定されています。詳細については、「the section called “プロビジョンドコントロールプレーンの作成”」を参照してください。

クラスターがサポートする HPA オブジェクトの数への影響

- 同期期間を短縮すると、同じキューを処理する時間が短くなるため、コントロールプレーンがスケジュールどおりに調整できる **HorizontalPodAutoscaler** オブジェクトの数が減少します。期間を 15s から 10s に短縮すると、サポートされるオブジェクト数がおよそ 3 分の 1 減少します。同期期間を短縮する前に、クラスター内の HorizontalPodAutoscaler オブジェクトを数え、短縮後の期間でもスケーリング階層でその数がサポートされることを確認してください:

  ```
  kubectl get hpa --all-namespaces --no-headers | wc -l
  ```

- EKS は、HPA オブジェクト数に対して同期期間を検証しません。クラスターに短縮後の期間でサポートされる数より多くの HorizontalPodAutoscaler オブジェクトが既にある場合でも、設定変更は成功します。変更を行う前に、自分で数を確認してください。
- サポートされる数を超えると、自動スケーリングが通知なしに低下します。コントローラーが期間内にすべてのオブジェクトを処理できない場合、一部のオブジェクトはスケジュールどおりに調整されません。EKS はこの状態に対してアラームや Kubernetes イベントを発行せず、自動スケーリングの応答に予想より時間がかかるようになります。これは意図した効果とは逆です。同期期間を

短縮した後にスケーリングの遅延が発生した場合は、パラメータをデフォルトの 15s に戻してください。

- 同期期間はクラスター内のすべての HPA オブジェクトに適用されます。オブジェクトや名前空間ごとに異なる同期期間を設定することはできません。

コントローラーマネージャー: 終了したポッドのガベージコレクションしきい値

コントローラーマネージャーは、終了したポッドのガベージコレクター (ポッド GC コントローラー) を実行します。このコントローラーは、クラスター内の終了したポッドの数がしきい値を超えると、終了したポッド (Succeeded または Failed フェーズのポッド) を削除します。terminatedPodGcThreshold パラメータがそのしきい値を設定します。

フィールド	説明	サポートされる値	デフォルト
podGcControllerConfig.terminatedPodGcThreshold	終了したポッドのガベージコレクターが終了したポッドの削除を開始する前に存在できる、終了したポッドの数。	10000 ~ 12500	12500

ガベージコレクターは 20 秒の固定サイクルで実行されます。しきい値を下げると、コレクターは次のサイクルで、数が新しいしきい値に達するまで、最も古い終了ポッドの強制削除を開始します。

このパラメータを設定するには、クラスターが Amazon EKS プロビジョンドコントロールプレーン上にある必要があります。しきい値を下げると、コントローラーがクラスターデータベース (etcd) に対して実行するガベージコレクションの作業が増加します。各コレクションパスで、より多くの終了したポッドをクエリ、処理、削除できるようになります。プロビジョンドコントロールプレーンクラスターは、要求の厳しいワークロードに常に備えたコントロールプレーン容量を事前に割り当てるため、その追加負荷を吸収できます。詳細については、「the section called “プロビジョンドコントロールプレーンの作成”」を参照してください。

終了したポッドのガベージコレクションしきい値に関する考慮事項

- しきい値を下げると、余分な終了ポッドが直ちに削除されます。しきい値を下げると (例えば、12500 から 10000 に)、ガベージコレクションコントローラーは次のサイクルで最も古い終了ポッドの強制削除を開始します。コントローラーは、終了ポッド数が新しいしきい値に達するまで続行します。削減は段階的ではありません。

- しきい値は、所有者に関係なくすべての終了ポッドに適用されます。これは、Job、CronJob、または Deployment に所有されているか、スタンドアロンであるかに関係なく、`Succeeded` または `Failed` フェーズのポッドに影響します。実際には、Job および CronJob のポッドが、終了ポッド数の最も一般的な要因です。
- しきい値を下げると、デバッグの時間枠が短くなります。完了したポッドと失敗したポッドは、`kubectl get pods` と `kubectl logs` からより早く消えます。完了した Job ポッドの終了コードやログを検査するオートメーションは、動作できる時間枠が小さくなります。
- しきい値はクラスター全体の設定です。名前空間ごとまたは Job ごとに設定することはできません。個々の Job のポッドのライフサイクルを制御するには、その Job で `ttlSecondsAfterFinished` を使用します。

API サーバー: イベントの保持

API サーバーは Kubernetes コントロールプレーンのフロントエンドです。Kubernetes API を提供し、クラスターの状態をクラスターデータベース (etcd) に永続化します。Kubernetes は、ポッドのスケジューリング判断、イメージのプル、ヘルスチェックの失敗、スケーリングアクションなど、クラスターで発生していることを説明するイベントを記録します。

フィールド	説明	サポートされる値	デフォルト
`eventTtl`	API サーバーが Kubernetes イベントを削除する前に保持する期間。	`10m` ~ `60m`	`60m`

大規模なバッチジョブ、AI ワークロード、CI/CD パイプライン、頻繁な CronJob など、入れ替わりの激しいワークロードを実行するクラスターは、数千のイベントを急速に蓄積します。保持されるすべてのイベントは、クラスターの実行に必要なオブジェクトと競合するクラスターデータベース領域を消費し、イベントコレクションが大きくなると、API サーバーのリスト操作の処理コストが高くなります。

イベントの保持期間を短縮すると、この短命な診断データがより早く消去され、クラスターデータベースのストレージ負荷が軽減され、イベントの多いクエリに対する API サーバーの応答時間が改善されます。

保持期間を短くすることが適しているのは、次の場合です:

- クラスターが、大量のイベントを生成するバッチ、CI/CD、AI、または CronJob ワークロードを実行している場合。
- クラスターデータベースのストレージが限界に向けて増加しているのを確認した場合。
- イベントを永続的にキャプチャするために外部システムに依存しており、履歴デバッグに `kubectl get events` を利用していない場合。

イベントの保持に関する考慮事項

- 変更は新しいイベントにのみ適用されます。Kubernetes は、イベントの作成時にその有効期限を設定します。既に存在するイベントは、作成時に有効だった保持期間を維持し、そのスケジュールで期限切れになります。`eventTtl` を短縮しても、クラスターデータベースに既にあるイベントの有効期間は短くならないため、ストレージの削減は既存のイベントが期限切れになるにつれて徐々に効果を発揮します。
- 削除されたイベントは復元できません。Kubernetes がイベントを削除すると、そのイベントは永久に失われます。意図した以上に保持期間を短縮してイベント履歴を失った場合、それを復元する方法はありません。この値を短縮する前に、トラブルシューティングで依存している情報がクラスター外で収集されていることを確認してください。
- イベントは設定された期間をわずかに超えて保持されることがあります。状況によっては、コントロールプレーンのリーダー選出中に発生する可能性のある etcd リースの更新により、イベントの有効期限が設定した値を超えて延長されることがあります。
- デバッグウィンドウの短縮。保持期間を短くすると、`kubectl get events` と `kubectl describe` に表示されるウィンドウが狭くなります。クラスターからイベントを収集するモニタリングツールで利用できるデータが少なくなります。ストレージ効率とデバッグワークフローのバランスが取れる値を選択してください。
- この設定はクラスター全体に適用されます。保持期間は、ポッドのスケジューリング、ノードの状態、スケーリングイベントなど、すべての名前空間のすべてのイベントに適用されます。名前空間ごとに異なる保持期間を設定することはできません。

API サーバー: サービスノードポート範囲

Kubernetes は、ポートを必要とする各サービスのために、この範囲からすべてのノード上にポートを割り当てます。これにはタイプ `NodePort` のサービスが含まれ、デフォルトではタイプ `LoadBalancer` のサービスも含まれます。

フィールド	説明	サポートされる値	デフォルト
serviceNodePortRange.minPort	範囲内の最小ポート。	10260 - 32767	30000
serviceNodePortRange.maxPort	範囲内の最大ポート。	10260 - 32767	32767

minPort は maxPort 以下である必要があります。Amazon EKS は、minPort が maxPort より大きい設定を拒否します。

範囲を変更することで、ノードポートの割り当てを組織が既に適用しているネットワークポリシーやファイアウォールポリシーに合わせることができます。範囲を広げると、1 つのクラスターでサポートできるサービスの数も増えます。このパラメータは、移行時に特に役立ちます。Amazon EKS に移行するアプリケーションと、それらを呼び出すクライアントは、特定の固定ポートでサービスが提供されることを期待していることがよくあります。それらのポートがデフォルト範囲外にある場合、通常の選択肢は、アプリケーションを変更するか、その前にプロキシを配置することです。範囲をアプリケーションが既に使用しているポートに合わせることで、その作業が不要になり、アプリケーションを書き換えたり、維持するネットワークコンポーネントを追加したりすることなく、ワークロードを EKS に移行できます。

範囲が 10260 と 32767 で制限されている理由

10260 の下限により、NodePort の割り当ては、ノード上の Kubernetes システムコンポーネントが既に使用しているポート (kubelet ヘルスポート (10248) や kube-proxy ヘルスチェックポート (10256) など) を回避できます。

32767 の上限により、範囲は、通常 32768 から始まる Linux のエフェメラルポート範囲を回避できます。NodePort が エフェメラル範囲内にある場合、カーネルがノードからの送信接続用にそのポートを選択し、サービスと競合する可能性があります。

サービスノードのポート範囲に関する考慮事項

- 既存のサービスは割り当てられたポートを保持します。範囲を狭めた場合、新しい範囲外のポートを既に保持しているサービスは引き続き機能し、kube-proxy は引き続きそれらにトラフィックを

ルーティングします。Amazon EKS は、範囲が変更されても既存のサービスのポートを再割り当てしません。

- サービスを再作成すると、そのポートが再割り当てされます。範囲外のポートを保持しているサービスが削除されて再作成された場合、そのポートは割り当てられなくなります。特に、デプロイプロセスがサービスを更新するのではなく再作成する場合、既存のサービスが依存している範囲を狭める前に、これに備えて計画してください。
- 範囲外の新しい割り当ては拒否されます。設定された範囲外のポートを必要とするサービスを作成または更新すると、API サーバーからの検証エラーで失敗します。
- 明示的に指定されたポートも検証されます。サービスが Kubernetes に割り当てさせるのではなく、`nodePort` 値を直接指定する場合、そのポートは設定された範囲内にある必要があります。範囲外の静的ポートリクエストは、以前に設定していた現在の範囲より広い範囲では同じポートが有効だったとしても、拒否されます。
- 範囲はクラスター全体に適用されます。名前空間ごとに異なる範囲を設定することはできません。

このパラメータを変更する前に、セキュリティグループとネットワーク ACL が新しい範囲のトラフィックを許可していること、および範囲がノード上の他のソフトウェアで使用されているポートと競合しないことを確認してください。

考慮事項

高度なコントロールプレーンパラメータを設定する前に、以下を確認してください。

- クラスター全体のスコープ – コントロールプレーンパラメータは、クラスター全体とそこで実行されているすべてのワークロードに適用されます。これらを個々の名前空間やワークロードに限定することはできません。本番環境に適用する前に、非本番クラスターでパラメータ変更をテストしてください。
- Horizontal Pod Autoscaler の同期期間と終了したポッドのガベージコレクションしきい値にはプロビジョンドコントロールプレーンが必要 – `horizontalPodAutoscalerSyncPeriod` および `terminatedPodGcThreshold` パラメータは、Amazon EKS プロビジョンドコントロールプレーンを使用するクラスターでのみ使用できます。Amazon EKS は、コントロールプレーンのリソース消費を大幅に増加させるパラメータを、コントロールプレーン容量が事前に割り当てられたクラスターに限定します。標準コントロールプレーンモードのクラスターでいずれかのパラメータを設定すると失敗します。これらを使用するには、まずクラスターをプロビジョンドコントロールプレーンスケーリング階層に移行します。詳細については、「the section called “プロビジョンドコントロールプレーンの作成”」を参照してください。

- Horizontal Pod Autoscaler の同期期間と終了したポッドのガベージコレクションしきい値の終了制限 – `horizontalPodAutoscalerSyncPeriod` または `terminatedPodGcThreshold` がデフォルト以外の値に設定されている場合、クラスターのコントロールプレーンをプロビジョンドモードから標準モードに戻すことはできません。標準モードに戻すには、まず両方のパラメータをデフォルト (`15s` と `12500`) に戻し、次にコントロールプレーンのスケーリング階層を `standard` に変更します。
- デフォルト値への戻し – Amazon EKS は専用のリセット操作を提供しておらず、更新でフィールドを省略すると、そのフィールドはクリアされずに現在の値が保持されます。パラメータをデフォルトに戻すには、明示的にデフォルト値に設定します。クラスターが実行している Kubernetes バージョンのデフォルト値は、`DescribeClusterVersions` で取得します。詳細については、「the section called “コントロールプレーンパラメータを設定する”」を参照してください。
- 更新のセマンティクス – 更新は既存の設定とマージされます。指定したフィールドのみが変更され、省略したフィールドは現在の値を保持します。これは、コンポーネント間および単一コンポーネント内の両方に適用されます。例えば、スケジューラ設定のみを指定した更新では、コントローラーマネージャーと API サーバーの設定は変更されません。
- 現在の設定の表示 – `describe-cluster` 操作は、カスタマイズしていないパラメータとそのデフォルト値を含め、コントロールプレーンで実行されている完全な設定を返します。
- デフォルト値とサポートされる値は Kubernetes バージョン間で変わる可能性があります – このトピックに記載されている値は、公開時に利用可能な Kubernetes バージョンに適用されます。`DescribeClusterVersions` を使用して、各パラメータおよび Kubernetes バージョンの現在のデフォルト値とサポートされる値を取得します。「the section called “コントロールプレーンパラメータを設定する”」を参照してください。
- 既存のクラスターは変更されません – Amazon EKS は既存のクラスターの動作を変更しません。明示的にパラメータを設定するまで、すべてのクラスターはデフォルトのパラメータ値で実行され続けます。
- 変更は即座に適用されません – `UpdateClusterConfig` が戻った時点では、設定変更は有効になっていません。Amazon EKS はコントロールプレーンのローリング更新を通じて新しい設定を適用するため、変更が完全に有効になるまで数分かかります。更新が完了すると、クラスターは `ACTIVE` ステータスに戻ります。DescribeUpdate 操作を使用して進行状況を追跡するか、`aws eks wait cluster-active` を使用して変更が完了するまでブロックできます。
- 監査可能性 – Amazon EKS は適用前に各設定を検証し、設定変更を AWS CloudTrail に記録します。
- ツールのサポート – 高度な Kubernetes コントロールプレーン設定は、リリース時に AWS マネジメントコンソール、eksctl、AWS CLI、Amazon EKS API、AWS CloudFormation、AWS CDK を

通じて利用できます。AWS Controllers for Kubernetes (ACK) および Terraform のサポートは近日提供予定です。

- Kubernetes バージョンのサポート – 高度な Kubernetes コントロールプレーン設定は、Kubernetes バージョン 1.31 以降を実行している新規および既存のクラスターでサポートされています。
- AWS リージョンのサポート – 高度な Kubernetes コントロールプレーン設定は、Amazon EKS が利用可能なすべての AWS 商用リージョン、AWS GovCloud (US) リージョン、および AWS 中国リージョンで利用できます。
- 料金 – コントロールプレーンパラメータの設定に追加料金はかかりません。`horizontalPodAutoscalerSyncPeriod` を使用するにはプロビジョンドコントロールプレーンが必要で、スケーリング階層の時間単位の料金で請求されます。詳細については、「Amazon EKS の料金」を参照してください。

次のステップ

- the section called “コントロールプレーンパラメータを設定する” – AWS CLI と AWS マネジメントコンソール を使用してコントロールプレーンパラメータを設定および表示します。
- the section called “プロビジョンドコントロールプレーンの作成” – 予測可能で高いパフォーマンスを実現するために、コントロールプレーン容量を事前に割り当てます。

高度な Kubernetes コントロールプレーンパラメータを設定する

このガイドでは、AWS CLI と AWS マネジメントコンソール を使用して高度な Kubernetes コントロールプレーンパラメータを設定および表示する手順を説明します。各パラメータとそれがクラスターに与える影響の説明については、「the section called “高度なコントロールプレーン設定”」を参照してください。

前提条件

開始する前に、以下の準備が整っていることを確認します。

- Kubernetes バージョン 1.31 以降を実行している Amazon EKS クラスター – 高度な Kubernetes コントロールプレーン設定は、Kubernetes バージョン 1.31 以降を実行している新規および既存のクラスターでサポートされています。このトピックのすべてのパラメータに、この最小バージョンが適用されます。

- AWS CLI – Amazon EKS など AWS のサービスを操作するためのコマンドラインツールです。詳細については、AWS コマンドラインインターフェイスユーザーガイドの「インストール」を参照してください。AWS CLI のインストール後は設定も行っておくことをお勧めします。詳細については AWS コマンドラインインターフェイスユーザーガイドの「aws configure を使用したクイック設定」を参照してください。
- 必要な IAM アクセス許可 – 使用している IAM プリンシパルには、Amazon EKS クラスターに対して describe および update を実行するためのアクセス許可が必要です。現在のプリンシパルを確認するには、次のコマンドを実行します。

```
aws sts get-caller-identity
```

> **Note**
>
> このトピック内のステップはバッシュ シェル内で実行することが推奨されます。Bash シェルを使用していない場合、行継続文字や、変数の設定と使用に関する方法など、一部のスクリプトコマンドのためにシェルの調整が必要となります。さらに、シェルの引用規則と工スケープ規則は異なる場合があります。詳細については AWS コマンドラインインターフェイスのユーザーガイドの「AWS CLI での文字列への引用符の使用」を参照してください。

このトピックのコマンドを実行する前に、ご希望のクラスター名とリージョンを設定します。

```
export CLUSTER=my-eks-cluster
export AWS_REGION=us-west-2
```

サポートされる値

Amazon EKS は各パラメータを次の範囲に対して検証し、その範囲外のリクエストは拒否します。

コンポーネントパラメータ	サポートされる値	デフォルト
kubeSchedulerConfig.nodeResourcesFit.scoringStrategy.type	LeastAllocated , MostAllocated	LeastAllocated

コンポーネントパラメータ	サポートされる値	デフォルト
kubeSchedulerConfig.nodeResourcesFit.scoringStrategy.resources[].weight	1～100	cpu: 1, memory: 1
kubeControllerManagerConfig.horizontalPodAutoscalerControllerConfig.horizontalPodAutoscalerSyncPeriod	10s～15s	15s
kubeControllerManagerConfig.podGcControllerConfig.terminatedPodGcThreshold	10000～12500	12500
kubeApiServerConfig.eventTtl	10m～60m	60m
kubeApiServerConfig.serviceNodePortRange.minPort	10260～32767	30000
kubeApiServerConfig.serviceNodePortRange.maxPort	10260～32767	32767

これらの値は公開時点で利用可能な Kubernetes バージョンに適用され、今後のバージョンで変更される可能性があります。特定のバージョンの現在の値を取得するには、次のセクションを参照してください。

Kubernetes バージョンのデフォルト値とサポートされている値を確認する

DescribeClusterVersions オペレーションは、Kubernetes バージョンごとに、すべてのコントロールプレーンパラメータのデフォルト値とサポートされている値を報告します。特にクラスター設定を自動化したり、複数の Kubernetes バージョンにまたがってクラスターを管理したりする場合は、値をハードコーディングするのではなく、このオペレーションから読み取ってください。

```
aws eks describe-cluster-versions --cluster-versions 1.35
```

応答には、コントロールプレーンコンポーネント別に整理された controlPlaneComponentConfig オブジェクトが含まれます。各パラメータは defaultValue と constraints オブジェクトを報告します。

```
{
    "clusterVersions": [
        {
            "clusterVersion": "1.35",
            "clusterType": "eks",
            "defaultVersion": false,
            "releaseDate": "2026-01-23T00:00:00+00:00",
            "endOfStandardSupportDate": "2027-03-23T00:00:00+00:00",
            "endOfExtendedSupportDate": "2028-03-23T00:00:00+00:00",
            "status": "STANDARD_SUPPORT",
            "controlPlaneScalingTiers": [
                {
                    "tierName": "tier-xl",
                    "apiRequestConcurrency": 2000,
                    "podSchedulingRatePerSecond": 167,
                    "clusterDatabaseSizeGb": 16
                },
                {
                    "tierName": "tier-2xl",
                    "apiRequestConcurrency": 4000,
                    "podSchedulingRatePerSecond": 283,
                    "clusterDatabaseSizeGb": 16
                },
                {
                    "tierName": "tier-4xl",
                    "apiRequestConcurrency": 8000,
                    "podSchedulingRatePerSecond": 400,
                    "clusterDatabaseSizeGb": 16
                },
```

```
                {
                    "tierName": "tier-8xl",
                    "apiRequestConcurrency": 16000,
                    "podSchedulingRatePerSecond": 400,
                    "clusterDatabaseSizeGb": 16
                }
            ],
            "controlPlaneComponentConfig": {
                "kubeApiServerConfig": {
                    "eventTtl": {
                        "defaultValue": "60m",
                        "constraints": {
                            "min": "10m",
                            "max": "60m"
                        }
                    },
                    "serviceNodePortRange": {
                        "defaultValue": {
                            "minPort": 30000,
                            "maxPort": 32767
                        },
                        "constraints": {
                            "minPort": {"min": 10260, "max": 32767},
                            "maxPort": {"min": 10260, "max": 32767}
                        }
                    }
                },
                "kubeSchedulerConfig": {
                    "nodeResourcesFit": {
                        "scoringStrategy": {
                            "defaultValue": {
                                "type": "LeastAllocated",
                                "resources": [
                                    {"name": "cpu", "weight": 1},
                                    {"name": "memory", "weight": 1}
                                ]
                            },
                            "constraints": {
                                "scoringStrategy": {
                                    "allowedValues": ["LeastAllocated",
"MostAllocated"]
                                },
                                "resources": {
                                    "name": {
```

```
                                    "allowedValues": [
                                        "cpu",
                                        "memory",
                                        "nvidia.com/gpu",
                                        "aws.amazon.com/neuron",
                                        "aws.amazon.com/neuroncore"
                                    ]
                                },
                                "weight": {"min": 1, "max": 100}
                            }
                        }
                    }
                },
                "kubeControllerManagerConfig": {
                    "horizontalPodAutoscalerControllerConfig": {
                        "horizontalPodAutoscalerSyncPeriod": {
                            "defaultValue": "15s",
                            "constraints": {
                                "min": "10s",
                                "max": "15s"
                            }
                        }
                    },
                    "podGcControllerConfig": {
                        "terminatedPodGcThreshold": {
                            "defaultValue": 12500,
                            "constraints": {
                                "min": 10000,
                                "max": 12500
                            }
                        }
                    }
                }
            }
        }
    ]
}
```

前述のレスポンスには、バージョンのサポート日、プロビジョンドコントロールプレーンのスケーリング階層、およびコントロールプレーン設定フィールドが表示されます。詳細については、Amazon EKS API リファレンスの「DescribeClusterVersions」を参照してください。

設定フィールドのみを返すには、--query オプションを使用します。

```
aws eks describe-cluster-versions --cluster-versions 1.35 \
--query 'clusterVersions[0].controlPlaneComponentConfig'
```

コントロールプレーン設定 – AWS CLI

各コントロールプレーンコンポーネントには、--kube-scheduler-config、--kube-controller-manager-config、--kube-api-server-config といった独自のパラメータがあります。カスタマイズするコンポーネントのみを指定します。各パラメータはインライン JSON を受け入れます。

コントロールプレーンパラメータを指定してクラスターを作成する

次のコマンドは、新しい Kubernetes イベントをデフォルトの 60 分ではなく 10 分間保持するクラスターを作成します。指定しないパラメータにはデフォルト値が使用されます。

```
aws eks create-cluster \
--name "$CLUSTER" \
--kubernetes-version 1.35 \
--role-arn arn:aws:iam::111122223333:role/eks-service-role \
--resources-vpc-config subnetIds=subnet-abc123,subnet-def456,securityGroupIds=sg-xyz789
 \
--kube-api-server-config '{
  "eventTtl": "10m"
}'
```

レスポンス:

```
{
    "cluster": {
        "name": "my-eks-cluster",
        "arn": "arn:aws:eks:us-west-2:111122223333:cluster/my-eks-cluster",
        "createdAt": 1709640000.0,
        "version": "1.35",
        "roleArn": "arn:aws:iam::111122223333:role/eks-service-role",
        "status": "CREATING",
        "kubeSchedulerConfig": {
            "nodeResourcesFit": {
                "scoringStrategy": {
                    "type": "LeastAllocated"
                }
```

```
            }
        },
        "kubeControllerManagerConfig": {
            "horizontalPodAutoscalerControllerConfig": {
                "horizontalPodAutoscalerSyncPeriod": "15s"
            }
        },
        "kubeApiServerConfig": {
            "eventTtl": "10m",
            "serviceNodePortRange": {
                "minPort": 30000,
                "maxPort": 32767
            }
        }
    }
}
```

クラスターのコントロールプレーン設定を表示する

```
aws eks describe-cluster --name "$CLUSTER"
```

レスポンスに
は、kubeSchedulerConfig、kubeControllerManagerConfig、kubeApiServerConfig がトップレベルのフィールドとして含まれます。これらのフィールドは常に存在し、カスタマイズしていないパラメータとそのデフォルト値を含め、コントロールプレーンで実行されている完全な設定を示します。

単一のコンポーネントの設定を返すには、--query オプションを使用します。

```
aws eks describe-cluster --name "$CLUSTER" \
--query 'cluster.kubeSchedulerConfig'
```

レスポンス:

```
{
    "nodeResourcesFit": {
        "scoringStrategy": {
            "type": "LeastAllocated"
        }
    }
}
```

スケジューラのスコアリング戦略を設定する

デフォルトでは、スケジューラは LeastAllocated 戦略を使用し、ポッドをノード全体に分散させます。次のコマンドはそれを MostAllocated に変更します。これは、既にリソース割り当てが高いノードを優先し、より少ないノードにポッドを詰め込みます。

```
aws eks update-cluster-config \
--name "$CLUSTER" \
--kube-scheduler-config '{
  "nodeResourcesFit": {
    "scoringStrategy": {
      "type": "MostAllocated"
    }
  }
}'
```

レスポンス:

```
{
    "update": {
        "id": "a1b2c3d4-5678-90ab-cdef-EXAMPLE11111",
        "status": "InProgress",
        "type": "ControlPlaneComponentConfigUpdate",
        "params": [
            {
                "type": "kubeSchedulerConfig",
                "value": "{\"nodeResourcesFit\":{\"scoringStrategy\":{\"type\":
\"MostAllocated\"}}}"
            }
        ],
        "createdAt": 1709643600.0,
        "errors": []
    }
}
```

また、resources 配列を指定してスコアリング計算の個々のリソースに重みを付けることもできます。これは、クラスター内で特定のリソースが制約となっている場合に役立ちます。詳細については、「the section called “高度なコントロールプレーン設定”」を参照してください。

他のコントロールプレーンパラメータも同じパターンに従い、それぞれに独自のコンポーネントフラグがあります: eventTtl と serviceNodePortRange には --kube-api-server-config、Horizontal Pod Autoscaler の同期期間には --kube-controller-manager-config で

す。各パラメータの機能と、変更する前に考慮すべき事項については、「the section called “高度なコントロールプレーン設定”」を参照してください。

一度に複数のコンポーネントを更新する

1 回の呼び出しで複数のコンポーネントパラメータを指定できます。

```
aws eks update-cluster-config \
--name "$CLUSTER" \
--kube-scheduler-config '{"nodeResourcesFit":{"scoringStrategy":
{"type":"MostAllocated"}}}' \
--kube-api-server-config '{"eventTtl":"30m"}'
```

更新は既存の設定とマージされます。指定したフィールドのみが変更され、省略したフィールドは現在の値を保持します。前述のコマンドは、コントローラーマネージャーの設定を変更しません。

コントロールプレーン設定の更新を監視する

update-cluster-config が返された時点では、設定変更は有効になっていません。Amazon EKS はコントロールプレーンのローリング更新を通じて新しい設定を適用します。これには数分かかります。変更が完了するまでブロックするには、次のようにします。

```
aws eks wait cluster-active --name "$CLUSTER"
```

このコマンドは、クラスターが ACTIVE ステータスに達したときに返されます。通常は数分以内に達します。

クラスターの更新を一覧表示するには、次のようにします。

```
aws eks list-updates --name "$CLUSTER"
```

レスポンス:

```
{
    "updateIds": [
        "a1b2c3d4-5678-90ab-cdef-EXAMPLE11111"
    ]
}
```

特定の更新のステータスを表示するには、次のようにします。

```
aws eks describe-update --name "$CLUSTER" \
--update-id a1b2c3d4-5678-90ab-cdef-EXAMPLE11111
```

ステータスは、更新が完了すると InProgress から Successful に変わり、エラーが発生すると Failed に変わります。errors フィールドは失敗の理由を示します。

コントロールプレーン設定 – AWS マネジメントコンソール

クラスターを作成するときに高度な Kubernetes コントロールプレーンパラメータを設定するには、以下の手順を行います。

1. Amazon EKS コンソール を開きます。
2. [クラスターを作成] を選択します。
3. [設定オプション] で [カスタム設定] を選択します。
4. [コントロールプレーン設定] まで下にスクロールします。
5. [コントロールプレーン設定を有効にする] を選択します。
6. 希望のパラメータを設定します。変更しないパラメータにはデフォルト値が使用されます。
7. 必要に応じて、他のクラスター設定オプションを選択します。最後のステップで、[クラスターを作成] を選択します。クラスターの作成が完了するまでに数分かかる場合があります。

既存のクラスターで高度な Kubernetes コントロールプレーンパラメータを設定するには、次のようにします。

1. Amazon EKS コンソール を開きます。
2. 更新するクラスターを選択します。
3. [概要] タブを選択し、[コントロールプレーン設定] まで下にスクロールします。
4. [管理] を選択し、次に [コントロールプレーン設定を有効にする] を選択して、変更するパラメータを設定します。次に、変更の保存を選択します。

Kubernetes バージョンアップグレードの準備およびクラスターインサイトでの設定ミスのトラブルシューティング

Amazon EKS クラスターインサイトでは、問題の検出と解決のための推奨事項が提供され、クラスターの管理に役立ちます。すべての Amazon EKS クラスターは、Amazon EKS がキュレートしたインサイトのリストと照合して、自動的かつ定期的にチェックされます。これらのインサイトチェッ

クは Amazon EKS によって完全に管理され、どのような検出結果にも対処するための推奨事項を提示します。

クラスターインサイトタイプ

- 設定インサイト: クラスターまたはワークロードの機能を損なう可能性のある EKS Hybrid Nodes の設定ミスを特定します。
- アップグレードインサイト: Kubernetes の新しいバージョンにアップグレードする機能に影響を与える可能性のある問題を特定します。
- ロールバックの準備状況インサイト: アップグレード後に以前の Kubernetes バージョンにロールバックする機能に影響を与える可能性のある問題を特定します。

考慮事項

- 頻度: Amazon EKS は、クラスターインサイトを 24 時間ごとに更新します。また、最新のステータスを確認するときには、手動で更新することもできます。例えば、問題に対処した後でクラスターインサイトを手動で更新して、実際に問題が解決したかどうかを確認できます。
- アクセス許可: Amazon EKS は、すべての EKS クラスターでクラスターインサイトのクラスターアクセスエントリを自動的に作成します。このエントリは、クラスターに関する情報を表示するアクセス許可を EKS に付与します。Amazon EKS は、この情報を使用してインサイトを生成します。詳細については、「the section called “AmazonEKSClusterInsightsPolicy”」を参照してください。
- ロールバックの準備状況の可用性: ロールバックの準備状況インサイトは、過去 7 日以内にアップグレードされたクラスターでのみ使用できます。7 日間のロールバック資格期間が終了すると、これらのインサイトはクラスターに対して生成されなくなります。

ユースケース

Amazon EKS のクラスターインサイトは、Kubernetes クラスターの健全性、信頼性、および最適な設定を維持するのに役立つ自動チェックを提供します。以下は、アップグレードの準備状況や設定のトラブルシューティングを含む、クラスターインサイトの主要なユースケースです。

アップグレードインサイト

アップグレードインサイトは、クラスターインサイト内の特定のタイプのインサイトチェックです。これらのチェックは Kubernetes バージョンアップグレードの準備状況に関するインサイトを返しま

す。Amazon EKS は、すべての EKS クラスターでアップグレードインサイトチェックを実行します。

> **Important**
>
> Amazon EKS では、特定のクラスターインサイトの問題が発生した際に `--force` フラグでのクラスターアップグレードが必要となる機能が一時的にロールバックされています。詳細については、GitHub の「Temporary rollback of enforcing upgrade insights on update cluster version」を参照してください。
> クラスターの更新に関する詳細については、「the section called “ステップ 3: クラスターコントロールプレーンを更新する”」を参照してください。

クラスターの Kubernetes バージョンを更新する前に、Amazon EKS コンソールのオブザーバビリティダッシュボードにある [アップグレード] タブを使用できます。クラスターに問題が見つかった場合は、確認して適切な修正を行います。問題には、Amazon EKS および Kubernetes のドキュメントへのリンクも含まれます。問題を修正したら、オンデマンドでクラスターインサイトを更新して、最新のインサイトを取得してください。すべての問題が解決したら、クラスターを更新します。

Amazon EKS は Kubernetes バージョンアップグレードの準備状況に関するインサイトを返します。アップグレードインサイトは、Kubernetes クラスターのアップグレードに影響する可能性のある問題を特定します。これにより、管理者がアップグレードの準備に費やす労力が最小限に抑えられ、新しい Kubernetes バージョンでのアプリケーションの信頼性が高まります。クラスターは、Amazon EKS によって Kubernetes バージョンアップグレードに影響する可能性のある問題のリストと照合して自動的にスキャンされます。Amazon EKS は、各 Kubernetes バージョンリリースで行われた変更のレビューに基づいて、インサイトチェックのリストを頻繁に更新します。

Amazon EKS のアップグレードインサイトにより、新しいバージョンのテストと検証のプロセスがスピードアップします。また、クラスター管理者やアプリケーション開発者は、懸念事項を浮き彫りにし、修正に関するアドバイスを提供することで、最新の Kubernetes 機能を活用できます。

設定インサイト

EKS クラスターインサイトは、ハイブリッドノードを使用して Amazon EKS クラスターを自動的にスキャンし、Kubernetes コントロールプレーンからウェブフックへの通信、exec やログなどの kubectl コマンドなどに障害をもたらす設定の問題を特定します。設定インサイトは問題を明らかにし、修復に関する推奨事項を提供し、完全に機能するハイブリッドノードのセットアップまでの時間を短縮します。

ロールバックの準備状況インサイト

ロールバックの準備状況インサイトは、クラスターインサイト内の特定のタイプのインサイトチェックです。これらのチェックは Kubernetes バージョンのロールバック準備状況に関するインサイトを返します。Amazon EKS は、過去 7 日以内にアップグレードされたクラスターに対してロールバックの準備状況インサイトチェックを実行します。ロールバックの準備状況インサイトはポイントインタイムチェックであり、継続的にではなく、評価時のクラスターの状態を反映します。

クラスターの Kubernetes バージョンをアップグレードした後、Amazon EKS コンソールのオブザーバビリティダッシュボードにある [アップグレードインサイト] タブを使用して、ロールバック準備インサイトをレビューできます。クラスターに問題が見つかった場合は、確認して適切な修正を行います。問題には、Amazon EKS および Kubernetes のドキュメントへのリンクも含まれます。問題を修正したら、オンデマンドでクラスターインサイトを更新して、最新のインサイトを取得してください。すべての問題が解決したら、ロールバックを続行できます。

Amazon EKS は `ROLLBACK_READINESS` カテゴリの下に Kubernetes バージョンのロールバック準備状況に関するインサイトを返します。ロールバックの準備状況インサイトは、Kubernetes クラスターバージョンのロールバックに影響を与える可能性のある問題を特定します。これにより、管理者がロールバックの準備に費やす労力が最小限に抑えられ、以前のバージョンへの復元の信頼性が高まります。クラスターは、API の使用の互換性 (以前の API バージョンが削除された API バージョン卒業中の非互換性チェック、目的のバージョンに存在しない新しいリソース、および新しいフィールドや列挙型の変更をカバーする)、クラスターの状態、kubelet と kube-proxy のバージョンスキュー、EKS マネージドアドオンの互換性、および Auto Mode クラスターの中断予算と注釈チェックなど、ロールバックに影響する可能性のある問題のリストに対して Amazon EKS によって自動的にスキャンされます。

> **Note**
>
> ロールバックの準備状況インサイトは、EKS マネージドアドオンバージョンのみをチェックします。セルフマネージドアドオンの場合、または EKS アドオンライフサイクルの外部でマネージドアドオンのバージョンを上書きした場合、インサイトはバージョンの非互換性を検出しません。ユーザーには、ロールバックする前に、これらのアドオンとターゲットバージョンとの互換性を検証する責任があります。

ERROR または UNKNOWN ステータスがあるインサイトは、問題が解決されるまでロールバックをブロックします。自分のリスクで続行する場合は、`--force` フラグを使用してインサイトチェックをバイパスできます。WARNING ステータスがあるインサイトは助言的なものであり、ロールバッ

クを妨げません。詳細については、「the section called “Kubernetes バージョンのロールバック”」を参照してください。

はじめに

実行されたインサイトチェックのリストと、Amazon EKS によって特定された関連する問題を確認するには、AWS マネジメントコンソール、AWS CLI、AWS SDK、および Amazon EKS `ListInsights` API オペレーションを使用できます。開始するには、the section called “クラスターインサイトを表示する” を参照してください。

クラスターインサイトを表示する

Amazon EKS は、設定インサイト、アップグレードインサイト、およびロールバックの準備状況インサイトの 3 種類のインサイトを提供します。設定インサイトは、クラスターまたはワークロードの機能を損なう可能性のある EKS Hybrid Nodes の設定ミスを特定します。アップグレードインサイトは、Kubernetes の新しいバージョンにアップグレードする機能に影響を与える可能性のある問題を特定します。ロールバックの準備状況インサイトは、アップグレード後に以前の Kubernetes バージョンにロールバックする機能に影響を与える可能性のある問題を特定します。

実行されたインサイトチェックのリストと、Amazon EKS によって特定された関連する問題を確認するには、AWS マネジメントコンソール、AWS CLI、AWS SDK、および Amazon EKS `ListInsights` API オペレーションを使用できます。

設定インサイトを表示する (コンソール)

1. Amazon EKS コンソール を開きます。
2. クラスターのリストから、インサイトを確認する Amazon EKS クラスターの名前を選択します。
3. [クラスターを監視する] を選択します。
4. [クラスターヘルス] タブを選択します。
5. [設定インサイト] テーブルには、次の列が表示されます。
   - 名前 — Amazon EKS がクラスターに対して実行したチェック。
   - インサイトステータス – ステータスが `Error` のインサイトは、クラスター機能に影響を与える可能性のある設定ミスがあることを意味します。ステータスが `Warning` のインサイトは、設定が文書化されたアプローチと一致しないことを意味しますが、クラスター機能を意図的に設定すると機能する可能性があります。ステータスが `Passing` のインサイトは、Amazon EKS でこのインサイトチェックに関連する問題がクラスター内で検出されなかったことを意味します。

- バージョン – 該当するバージョン。
- 最終更新時刻 – このクラスターについてのインサイトのステータスが最後に更新された時刻。
- 説明 — アラートと修復のための推奨アクションを含むインサイトチェックの情報。

アップグレードインサイトを表示する (コンソール)

1. Amazon EKS コンソール を開きます。
2. クラスターのリストから、インサイトを確認する Amazon EKS クラスターの名前を選択します。
3. [クラスターを監視する] を選択します。
4. [アップグレードインサイト] タブを選択します。
5. 最新のデータを表示するには、[インサイトを更新] ボタンを選択し、更新操作が完了するまで待ちます。
6. [アップグレードインサイト] テーブルには、次の列が表示されます。
   - 名前 — Amazon EKS がクラスターに対して実行したチェック。
   - インサイトステータス — ステータスが「エラー」のインサイトは、通常、影響を受ける Kubernetes バージョンが現在のクラスターバージョンの N+1 であることを意味し、ステータスが「警告」とは、インサイトが将来の Kubernetes バージョン N+2 以上に適用されることを意味します。ステータスが「合格」のインサイトは、Amazon EKS がこのインサイトチェックに関連する問題をクラスター内で発見しなかったことを意味します。インサイトステータスが「不明」の場合は、Amazon EKS はクラスターがこのインサイトチェックの影響を受けているかどうかを判断できないことを意味します。
   - バージョン — インサイトにより問題の可能性がチェックされた Kubernetes バージョン。
   - 最終更新時刻 – このクラスターについてのインサイトのステータスが最後に更新された時刻。
   - 前回の移行時刻 – このインサイトのステータスが最後に変更された時刻。
   - 説明 — アラートと修復のための推奨アクションを含むインサイトチェックの情報。

> **Note**
>
> ロールバックの準備状況インサイトは、アップグレードの実行後に同じタブに表示され、7 日間のロールバック資格ウィンドウ中も表示されます。

クラスターインサイトを表示する (AWS CLI)

1. 最新のデータを表示するには、指定されたクラスターのインサイトを更新します。必要に応じてコマンドに次の変更を加えてから、そのコマンドを実行してください。
   - *region-code* を、AWS リージョンのコードに置き換えます。
   - ******* の部分は自分のクラスター名に置き換えます。

   ```
   aws eks start-insights-refresh --region region-code --cluster-name my-cluster
   ```

2. インサイト更新のステータスを追跡するには、次のコマンドを実行します。******* の部分は自分のクラスター名に置き換えます。

   ```
   aws eks describe-insights-refresh --cluster-name my-cluster
   ```

   出力例は次のとおりです。

   ```
   {
       "message": "Insights refresh is in progress",
       "status": "IN_PROGRESS",
       "startedAt": "2025-07-30T13:36:09-07:00"
   }
   ```

3. 指定されたクラスターのインサイトをリストします。必要に応じてコマンドに次の変更を加えてから、そのコマンドを実行してください。
   - *region-code* を、AWS リージョンのコードに置き換えます。
   - ******* の部分は自分のクラスター名に置き換えます。

   ```
   aws eks list-insights --region region-code --cluster-name my-cluster
   ```

   出力例は次のとおりです。

   ```
   {
   "insights":
       [
           {
               "id": "a1b2c3d4-5678-90ab-cdef-EXAMPLE11111",
   ```

```
            "name": "Deprecated APIs removed in Kubernetes vX.XX",
            "category": "UPGRADE_READINESS",
            "kubernetesVersion": "X.XX",
            "lastRefreshTime": 1734557315.000,
            "lastTransitionTime": 1734557309.000,
            "description": "Checks for usage of deprecated APIs that are scheduled
for removal in Kubernetes vX.XX. Upgrading your cluster before migrating to the
updated APIs supported by vX.XX could cause application impact.",
            "insightStatus":
            {
                "status": "PASSING",
                "reason": "No deprecated API usage detected within the last 30
days.",
            },
        },
        {
            "id": "a1b2c3d4-5678-90ab-cdef-EXAMPLE22222",
            "name": "Kubelet version skew",
            "category": "UPGRADE_READINESS",
            "kubernetesVersion": "X.XX",
            "lastRefreshTime": 1734557309.000,
            "lastTransitionTime": 1734557309.000,
            "description": "Checks for kubelet versions of worker nodes in the
cluster to see if upgrade would cause non compliance with supported Kubernetes
kubelet version skew policy.",
            "insightStatus":
            {
                "status": "UNKNOWN",
                "reason": "Unable to determine status of node kubelet versions.",
            },
        },
        {
            "id": "a1b2c3d4-5678-90ab-cdef-EXAMPLE33333",
            "name": "Deprecated APIs removed in Kubernetes vX.XX",
            "category": "UPGRADE_READINESS",
            "kubernetesVersion": "X.XX",
            "lastRefreshTime": 1734557315.000,
            "lastTransitionTime": 1734557309.000,
            "description": "Checks for usage of deprecated APIs that are scheduled
for removal in Kubernetes vX.XX. Upgrading your cluster before migrating to the
updated APIs supported by vX.XX could cause application impact.",
            "insightStatus":
            {
                "status": "PASSING",
```

```
                "reason": "No deprecated API usage detected within the last 30
days.",
            },
        },
        {
            "id": "a1b2c3d4-5678-90ab-cdef-EXAMPLEaaaaa",
            "name": "Cluster health issues",
            "category": "UPGRADE_READINESS",
            "kubernetesVersion": "X.XX",
            "lastRefreshTime": 1734557314.000,
            "lastTransitionTime": 1734557309.000,
            "description": "Checks for any cluster health issues that prevent
successful upgrade to the next Kubernetes version on EKS.",
            "insightStatus":
            {
                "status": "PASSING",
                "reason": "No cluster health issues detected.",
            },
        },
        {
            "id": "a1b2c3d4-5678-90ab-cdef-EXAMPLEbbbbb",
            "name": "EKS add-on version compatibility",
            "category": "UPGRADE_READINESS",
            "kubernetesVersion": "X.XX",
            "lastRefreshTime": 1734557314.000,
            "lastTransitionTime": 1734557309.000,
            "description": "Checks version of installed EKS add-ons to ensure they
are compatible with the next version of Kubernetes. ",
            "insightStatus": { "status": "PASSING", "reason": "All installed EKS
add-on versions are compatible with next Kubernetes version."},
        },
        {
            "id": "a1b2c3d4-5678-90ab-cdef-EXAMPLEccccc",
            "name": "kube-proxy version skew",
            "category": "UPGRADE_READINESS",
            "kubernetesVersion": "X.XX",
            "lastRefreshTime": 1734557314.000,
            "lastTransitionTime": 1734557309.000,
            "description": "Checks version of kube-proxy in cluster to see if
upgrade would cause non compliance with supported Kubernetes kube-proxy version
skew policy.",
            "insightStatus":
            {
                "status": "PASSING",
```

```
                "reason": "kube-proxy versions match the cluster control plane
version.",
            },
        },
        {
            "id": "a1b2c3d4-5678-90ab-cdef-EXAMPLEddddd",
            "name": "Deprecated APIs removed in Kubernetes vX.XX",
            "category": "UPGRADE_READINESS",
            "kubernetesVersion": "X.XX",
            "lastRefreshTime": 1734557315.000,
            "lastTransitionTime": 1734557309.000,
            "description": "Checks for usage of deprecated APIs that are scheduled
for removal in Kubernetes vX.XX. Upgrading your cluster before migrating to the
updated APIs supported by vX.XX could cause application impact.",
            "insightStatus":
            {
                "status": "PASSING",
                "reason": "No deprecated API usage detected within the last 30
days.",
            },
        },
        {
            "id": "a1b2c3d4-5678-90ab-cdef-EXAMPLE44444",
            "name": "Incompatible API usage",
            "category": "ROLLBACK_READINESS",
            "kubernetesVersion": "X.XX",
            "lastRefreshTime": 1734557315.000,
            "lastTransitionTime": 1734557309.000,
            "description": "Checks for usage of APIs that are not compatible
with the previous Kubernetes version. Rolling back your cluster before removing
incompatible API usage could cause data loss or application impact.",
            "insightStatus":
            {
                "status": "ERROR",
                "reason": "Incompatible API usage detected.",
            },
        },
        {
            "id": "a1b2c3d4-5678-90ab-cdef-EXAMPLE55555",
            "name": "Kubelet version rollback compatibility",
            "category": "ROLLBACK_READINESS",
            "kubernetesVersion": "X.XX",
            "lastRefreshTime": 1734557309.000,
            "lastTransitionTime": 1734557309.000,
```

```
            "description": "Checks for kubelet versions of worker nodes in the
 cluster to see if rollback would cause non compliance with supported Kubernetes
 kubelet version skew policy.",
            "insightStatus":
            {
                "status": "ERROR",
                "reason": "At least one node kubelet version matches the cluster
 control plane version.",
            },
        },
        {
            "id": "a1b2c3d4-5678-90ab-cdef-EXAMPLE66666",
            "name": "EKS add-on version rollback compatibility",
            "category": "ROLLBACK_READINESS",
            "kubernetesVersion": "X.XX",
            "lastRefreshTime": 1734557314.000,
            "lastTransitionTime": 1734557309.000,
            "description": "Checks version of installed EKS add-ons to ensure they
 are compatible with the previous version of Kubernetes.",
            "insightStatus":
            {
                "status": "PASSING",
                "reason": "All installed EKS add-on versions are compatible with
 previous Kubernetes version.",
            },
        },
    ],
"nextToken": null,
}
```


4. インサイトの詳細を表示するには、次のコマンドを実行します。必要に応じてコマンドに次の変更を加えてから、そのコマンドを実行してください。
   - *region-code* を、AWS リージョンのコードに置き換えます。
   - *a1b2c3d4-5678-90ab-cdef-EXAMPLE22222* を、クラスターインサイトのリストから取得したインサイト ID に置き換えます。
   - *#######* の部分は自分のクラスター名に置き換えます。

   ```
   aws eks describe-insight --region region-code --id a1b2c3d4-5678-90ab-cdef-
   EXAMPLE22222 --cluster-name my-cluster
   ```

   出力例は次のとおりです。

```
{
  "insight":
    {
      "id": "a1b2c3d4-5678-90ab-cdef-EXAMPLE22222",
      "name": "Kubelet version skew",
      "category": "UPGRADE_READINESS",
      "kubernetesVersion": "1.27",
      "lastRefreshTime": 1734557309.000,
      "lastTransitionTime": 1734557309.000,
      "description": "Checks for kubelet versions of worker nodes in the cluster
 to see if upgrade would cause non compliance with supported Kubernetes kubelet
 version skew policy.",
      "insightStatus":
        {
          "status": "UNKNOWN",
          "reason": "Unable to determine status of node kubelet versions.",
        },
      "recommendation": "Upgrade your worker nodes to match the Kubernetes version
 of your cluster control plane.",
      "additionalInfo":
        {
          "Kubelet version skew policy": "https://kubernetes.io/releases/version-
skew-policy/#kubelet",
          "Updating a managed node group": "https://docs.aws.amazon.com/eks/latest/
userguide/update-managed-node-group.html",
        },
      "resources": [],
      "categorySpecificSummary":
        { "deprecationDetails": [], "addonCompatibilityDetails": [] },
    },
}
```


前述の出力には、UPGRADE_READINESS と ROLLBACK_READINESS の両方のインサイトが表示されます。ロールバックの準備状況インサイトは、過去 7 日以内にアップグレードされたクラスターにのみ存在します。

カテゴリでインサイトをフィルタリングする

カテゴリでインサイトをフィルタリングして、特定のタイプのみを表示できます:

```
aws eks list-insights \
```

```
  --cluster-name my-cluster \
  --region region-code \
  --filter '{"categories": ["ROLLBACK_READINESS"]}'
```

ステータスでフィルタリングして、ブロックの問題のみを表示することもできます:

```
aws eks list-insights \
  --cluster-name my-cluster \
  --region region-code \
  --filter '{"categories": ["ROLLBACK_READINESS"], "statuses": ["ERROR"]}'
```

ロールバックの準備状況インサイトの詳細を表示する

ロールバックの準備状況インサイトでは、describe-insight コマンドは影響を受けるリソースと修復ステップに関する同様の情報を返します。例えば、次のようになります。

```
{
    "clusterId": "73a0e91f-f016-4555-bb7c-177496c47c9d",
    "insight": {
        "category": "ROLLBACK_READINESS",
        "name": "API usage rollback compatibility",
        "kubernetesVersion": "1.32",
        "insightStatus": {
            "status": "ERROR",
            "reason": "Detected incompatible API objects with version rollback."
        },
        "resources": [
            {
                "kubernetesResourceUri": "/apis/networking.k8s.io/v1/servicecidrs/
kubernetes",
                "status": {
                    "status": "ERROR",
                    "reason": "networking.k8s.io/v1 is not compatible with Kubernetes
 version 1.32"
                }
            }
        ]
    }
}
```

"status": "ERROR" でロールバックの準備状況インサイトが見つかった場合は、ロールバックを実行する前に問題に対処するか、--force フラグを使用してインサイトチェックをバイパスする必

要があります。ロールバックプロセスの詳細については、「the section called “Kubernetes バージョンのロールバック”」を参照してください。

インサイトが UNKNOWN ステータスを示す場合、EKS はインサイトを評価できていません。ロールバックは、インサイトが正常に評価されるか、--force フラグを使用してインサイトチェックをバイパスするまでブロックされます。

既存のクラスターを新しい Kubernetes バージョンに更新する

> **Tip**
>
> 今後開催予定の Amazon EKS ワークショップに登録してください。

Amazon EKS で利用可能な新しい Kubernetes バージョンがある場合には、Amazon EKS クラスターを最新バージョンに更新できます。

> **Important**
>
> アップグレード後、問題が発生した場合は 7 日以内に以前のバージョンにロールバックできます。詳細については、「the section called “Kubernetes バージョンのロールバック”」を参照してください。新しい Kubernetes バージョンに更新する前に、「Understand the Kubernetes version lifecycle on EKS」の情報と、本トピック内の更新手順を確認することをお勧めします。

新しい Kubernetes バージョンでは、大幅な変更が加えられている場合があります。このため、本稼働用クラスターで更新を行う前に、新しいバージョンの Kubernetes に対するアプリケーションの動作をテストしておくことをお勧めします。この際は、継続的な統合ワークフローを構築し、新しい Kubernetes バージョンに移行する前にアプリケーションの動作をテストします。

更新プロセスに含まれている Amazon EKS が、更新された Kubernetes バージョンを使用しながら新しい API サーバーノードを起動することで、既存のバージョンを置き換えます。Amazon EKS は、これらの新しいノードで、ネットワークトラフィックの標準インフラストラクチャと準備状況に関するヘルスチェックを実行し、想定どおりに動作していることを確認します。ただし、クラスターのアップグレードを開始すると、一時停止または停止することはできません。これらのヘルスチェックのいずれかが失敗すると、Amazon EKS はインフラストラクチャのデプロイを元に戻します。ク

ラスターは前の Kubernetes バージョンのままになります。この際も、実行中のアプリケーションは影響を受けません。また、クラスターが不確定または回復不可能な状態のままになることもありません。Amazon EKS は定期的にすべてのマネージド型クラスターをバックアップします。さらに、必要に応じてクラスターを復元するメカニズムも存在します。Kubernetes インフラストラクチャの管理プロセスは、継続的に評価、改善されています。

クラスターをアップグレードする際、Amazon EKS には、クラスターの作成時に指定したサブネット内に、最大で 5 つの使用可能な IP アドレスが必要となります。Amazon EKS は、指定したサブネットのいずれかに、新しいクラスターの Elastic Network Interface (ネットワークインターフェイス) を作成します。新しいネットワークインターフェイスは、既存のネットワークインターフェイスがあるのとは b サブネット内に作成される場合があります。ですので、クラスター作成時に指定したサブネットのいずれでも必要なクラスターとの通信が許可されるよう、セキュリティグループルールを設定します。クラスターの作成時に指定したサブネットのいずれかが存在しない、使用できる十分な IP アドレスがない、または必要なクラスターとの通信を許可するセキュリティグループルールがない場合、更新が失敗する可能性があります。

クラスターの API サーバーエンドポイントに常にアクセスできるように、Amazon EKS では高可用性を備えた Kubernetes コントロールプレーンを提供しており、アップデート実行中に API サーバーインスタンスのローリングアップデートを行います。Kubernetes API サーバーエンドポイントをサポートする API サーバーインスタンスの IP アドレスの変更を考慮するために、API サーバークライアントが再接続を適切に管理しているか確認する必要があります。`kubectl` の最新バージョンと公式にサポートされている Kubernetes クライアントライブラリは、この再接続プロセスを透過的に実行します。

> **Note**
>
> クラスターの更新の詳細については、「EKS Best Practices Guide」の「Best Practices for Cluster Upgrades」を参照してください。このリソースは、アップグレードを計画し、クラスターのアップグレード戦略を理解するのに役立ちます。

Amazon EKS Auto Mode の考慮事項

- Amazon EKS Auto Mode のコンピューティング機能は、Kubernetes バージョンのノードを制御します。コントロールプレーンをアップグレードすると、EKS Auto Mode はマネージドノードの段階的な更新を開始します。EKS Auto Mode は、ポッドの停止状態の予算を尊重します。
- コンピューティングの自動スケーリング、ブロックストレージ、ロードバランシング機能など、Amazon EKS Auto Mode の機能を手動でアップグレードする必要はありません。

> **Note**
>
> アップグレード後に問題が発生した場合は、ロールバックを開始できます。EKS Auto Mode を実行しているクラスターでは、EKS はコントロールプレーンを元に戻す前にワーカーノードを自動的にロールバックします。詳細については、「the section called “Kubernetes バージョンのロールバック”」を参照してください。

概要

Amazon EKS クラスターのアップグレードプロセスの概要は以下のとおりです。

1. クラスターがアップグレードをサポートする状態であることを確認します。これには、クラスターにデプロイされたリソースで使用されている Kubernetes API をチェックし、クラスターに正常性の問題がないことを確認することが含まれます。クラスターのアップグレードの準備状況を評価する際には、Amazon EKS のアップグレードインサイトを使用する必要があります。
2. コントロールプレーンを次のマイナーバージョンにアップグレードします (1.36 から 1.37 など)。
3. データプレーン内のノードをコントロールプレーンのノードと一致するようにアップグレードします。
4. その他にクラスターで実行されているアプリケーション (`cluster-autoscaler` など) がある場合は、そのアプリケーションをアップグレードします。
5. Amazon EKS が提供するアドオン (デフォルトで含まれるアドオンなど) をアップグレードします。
   - Amazon VPC CNI の推奨バージョン
   - CoreDNS の推奨バージョン
   - `kube-proxy` の推奨バージョン
6. クラスターと通信しているクライアントがある場合は、そのクライアントをアップグレードします (`kubectl` など)。

ステップ 1: アップグレードの準備をする

クラスターコントロールプレーンの Kubernetes バージョンと、ノードの Kubernetes バージョンを比較します。

- クラスターコントロールプレーンの Kubernetes バージョンを取得します。

```
kubectl version
```

- ノードの Kubernetes バージョンを取得します。このコマンドでは、セルフマネージド型およびマネージド型の Amazon EC2、Fargate、およびハイブリッドノードがすべて返されます。各 Fargate Pod は、独自のノードとしてリストされます。

```
kubectl get nodes
```

コントロールプレーンを新しい Kubernetes バージョンに更新する前に、クラスター内のマネージド型ノードと Fargate ノードの双方の Kubernetes マイナーバージョンが、コントロールプレーンのバージョンと同じであることを確認してください。例えば、コントロールプレーンがバージョン `1.29` を実行し、かつノードの 1 つがバージョン `1.28` を実行している場合、コントロールプレーンを 1.30 に更新する前に、ノードをバージョン `1.29` に更新する必要があります。また、コントロールプレーンを更新する前に、セルフマネージド型ノードとハイブリッドノードをコントロールプレーンと同じバージョンに更新することをお勧めします。詳細については、the section called “更新”、the section called “更新方式”、および the section called “ハイブリッドノードをアップグレードする” を参照してください。Fargate ノードのマイナーバージョンがコントロールプレーンのバージョンよりも古い場合、まずノードの示す Pod を削除します。次に、コントロールプレーンを更新します。残りの Pod は、再デプロイ後に新しいバージョンに更新されます。

ステップ 2: アップグレードに関する考慮事項を確認する

Amazon EKS クラスターインサイトは、非推奨の Kubernetes API の使用など、Kubernetes バージョンのアップグレードに影響を与える可能性のある問題のリストに照らして、クラスターを自動的にスキャンします。Amazon EKS は、Kubernetes プロジェクトの変更の評価に基づいて、実行するインサイトチェックのリストを定期的に更新します。また、Amazon EKS は、新しいバージョンに伴って Amazon EKS サービスに変更が導入された際にも、インサイトチェックのリストを更新します。詳細については、「the section called “クラスターのインサイト”」を参照してください。

Kubernetes ドキュメントの「Deprecated API Migration Guide」を確認してください。

アップグレードインサイトを確認する

Amazon EKS アップグレードインサイトを使用して問題を特定します。詳細については、「the section called “アップグレードインサイトを表示する (コンソール)”」を参照してください。

詳細な考慮事項

- Amazon EKS は可用性の高いコントロールプレーンを実行しているため、一回に更新できるのはマイナーバージョン 1 つのみです。この要件の詳細については、「Kubernetes Version and Version Skew Support Policy」(Kubernetes のバージョンおよびバージョンスキューのサポートポリシー) を参照してください。現在のクラスターバージョンが `1.28` であり、`1.30` に更新することを仮定します。最初にバージョン `1.28` クラスターをバージョン `1.29` に更新し、次にバージョン `1.29` クラスターをバージョン `1.30` に更新する必要があります。
- ノード上の Kubernetes `kube-apiserver` と `kubelet` 間のバージョンスキューを確認してください。
  - Kubernetes バージョン `1.28` 以降、`kubelet` は `kube-apiserver` より最大 3 マイナーバージョン古い場合があります。「Kubernetes アップストリームバージョンのスキューポリシー」を参照してください。
  - マネージドノードと Fargate ノードの `kubelet` が Kubernetes バージョン `1.25` 以降の場合は、`kubelet` バージョンを更新せずに最大 3 バージョン先までクラスターを更新できます。例えば、`kubelet` がバージョン `1.25` である場合、`kubelet` のバージョンを `1.25` のままにしていても、Amazon EKS クラスターのバージョンは `1.25` から `1.26`、`1.27`、`1.28` に更新できます。
- 更新を開始する前のベストプラクティスとして、ノード上の `kubelet` がコントロールプレーンと同じ Kubernetes バージョンであることを確認してください。
- クラスターが `1.8.0` より前のバージョンの Amazon VPC CNI Plugin for Kubernetes で設定されている場合は、クラスターを更新する前に、プラグインを最新バージョンに更新することをお勧めします。プラグインのアップデートについては、「the section called “Amazon VPC CNI”」を参照してください。
- Amazon EKS クラスターのバックアップを取得することで、アップグレードプロセス中に障害が発生した場合に、Amazon EKS クラスターの状態と永続的ストレージを復元できます。「the section called “ AWS バックアップ”」を参照してください。

ステップ 3: クラスターコントロールプレーンを更新する

> **Important**
>
> Amazon EKS では、特定のクラスターインサイトの問題が発生した際に `--force` フラグでのクラスターアップグレードが必要となる機能が一時的にロールバックされています。詳細

については、GitHub の「Temporary rollback of enforcing upgrade insights on update cluster version」を参照してください。
Amazon EKS は、「最終更新時間」から 24 時間後にクラスターインサイトを更新します。問題に対処した時刻とクラスターインサイトの「最終更新時間」を比較できます。
さらに、非推奨 API の使用に対処した後、インサイトのステータスが更新されるまでに最大 30 日かかる場合があります。アップグレードインサイトは、常に、過去 30 日の期間中の非推奨 API の使用を確認しています。

EKS コントロールプレーンバージョンのアップグレードリクエストは、以下を使用して送信できます。

- eksctl
- AWS コンソール
- AWS CLI

クラスターの更新 - eksctl

この手順には、`eksctl` バージョン `0.215.0` 以降が必要です。お使いのバージョンは、以下のコマンドを使用して確認できます。

```
eksctl version
```

`eksctl` のインストールと更新の手順については、`eksctl` ドキュメントの「インストール」を参照してください。

Amazon EKS コントロールプレーンの Kubernetes バージョンを更新します。`<cluster-name>` をクラスター名に置き換えます。`<version-number>` は、Amazon EKS がサポートする、クラスターの更新先のバージョン番号に置き換えてください。サポートされているバージョン番号のリストについては、「Amazon EKS supported versions」を参照してください。

```
eksctl upgrade cluster --name <cluster-name> --version <version-number> --approve
```

この更新は完了までに数分かかることがあります。

「the section called “ステップ 4: クラスターコンポーネントを更新する”」に進みます。

クラスターの更新 - AWS コンソール

1. Amazon EKS コンソールを開きます。
2. アップグレードするクラスターに対して [今すぐアップグレード] を選択します。
3. クラスターの更新先のバージョンを選択し、[アップグレード] を選択します。
4. この更新は完了までに数分かかることがあります。「the section called “ステップ 4: クラスターコンポーネントを更新する”」に進みます。

クラスターの更新 - AWS CLI

1. AWS CLI がインストールされ、ログインしていることを確認します。詳細については、「AWS CLI の最新バージョンのインストールまたは更新」を参照してください。
2. 次の AWS CLI コマンドを使用して、Amazon EKS クラスターを更新します。アップグレードするクラスターの <cluster-name> と <region-code> を置き換えます。<version-number> は、Amazon EKS がサポートする、クラスターの更新先のバージョン番号に置き換えてください。サポートされているバージョン番号のリストについては、「Amazon EKS supported versions」を参照してください。

```
aws eks update-cluster-version --name <cluster-name> \
  --kubernetes-version <version-number> --region <region-code>
```

出力例は次のとおりです。

```
{
    "update": {
        "id": "<update-id>",
        "status": "InProgress",
        "type": "VersionUpdate",
        "params": [
            {
                "type": "Version",
                "value": "<version-number>"
            },
            {
                "type": "PlatformVersion",
                "value": "eks.1"
            }
        ],
[...]
```

```
        "errors": []
    }
```

3. この更新は完了までに数分かかることがあります。クラスター更新のステータスをモニタリングするには、次のコマンドを使用します。同じ <cluster-name> と <region-code> を使用することに加えて、前のコマンドで返された <update-id> を使用します。

```
aws eks describe-update --name <cluster-name> \
   --region <region-code> --update-id <update-id>
```

   Successful ステータスが表示されると、更新は完了です。

4. 「the section called “ステップ 4: クラスターコンポーネントを更新する”」に進みます。

ステップ 4: クラスターコンポーネントを更新する

1. クラスターの更新が完了したら、更新したクラスターでの Kubernetes と同じマイナーバージョンに、ノードを更新する必要があります。詳細については、the section called “更新方式”、the section called “更新”、および the section called “ハイブリッドノードをアップグレードする” を参照してください。Fargate で起動される新しい Pod であれば、クラスターのバージョンと一致する kubelet バージョンを持っています。それまでに存在していた Fargate Pod は変更されていません。
2. (オプション) クラスターを更新する前に、そのクラスターに Kubernetes Cluster Autoscaler をデプロイしてある場合は、更新後の Kubernetes のメジャーバージョンとマイナーバージョンに一致するように、Cluster Autoscaler を最新バージョンに更新します。
   a. ウェブブラウザで Cluster Autoscaler のリリースページを開き、クラスターの Kubernetes メジャーバージョンとマイナーバージョンに一致する最新の Cluster Autoscaler バージョンを見つけます。例えば、クラスターの Kubernetes バージョンが 1.30 である場合、1.30 で始まる最新の Cluster Autoscaler リリースを見つけます。次のステップで使用するため、そのリリースのセマンティックバージョン番号 (例: 1.30.n) を書き留めておきます。
   b. 次のコマンドを使用して、Cluster Autoscaler イメージタグを、前のステップで書き留めたバージョンに設定します。必要に応じて、X.XX.X を独自の値に置き換えます。

```
kubectl -n kube-system set image deployment.apps/cluster-autoscaler cluster-
autoscaler=registry.k8s.io/autoscaling/cluster-autoscaler:vX.XX.X
```

3. (GPU ノードを含むクラスターのみ) クラスターに GPU 対応のノードグループ (p3.2xlarge など) がある場合は、クラスターの NVIDIA device plugin for Kubernetes DaemonSet を更新する

必要があります。次のコマンドを実行する前に、<vX.X.X> を必要となる NVIDIA/k8s-device-plugin バージョンに置き換えます。

```
kubectl apply -f https://raw.githubusercontent.com/NVIDIA/k8s-device-plugin/<vX.X.X>/deployments/static/nvidia-device-plugin.yml
```

4. Amazon VPC CNI Plugin for Kubernetes、CoreDNS、および kube-proxy アドオンを更新します。サービスアカウントトークンにリストされている最小バージョンに対してアドオンを更新することをお勧めします。
   - Amazon EKS アドオンを使用している場合は、Amazon EKS コンソールで [クラスター] をクリックし、左のナビゲーションペインで更新したクラスター名を選択します。通知がコンソールに表示されます。更新可能なアドオンごとに、新しいバージョンが利用可能であることが通知されます。アドオンを更新するには、[アドオン] タブを選択します。更新があるアドオンが表示されているボックスで [今すぐ更新] を選択し、使用可能なバージョンを選択してから、[更新] を選択します。
   - 別の方法として、AWS CLI または eksctl を使用してアドオンを更新することもできます。詳細については、「the section called “アドオンを更新する”」を参照してください。
5. 必要に応じて、kubectl のバージョンを更新します。Amazon EKS クラスターコントロールプレーンとのマイナーバージョンの相違が 1 つ以内である kubectl バージョンを使用する必要があります。

Amazon EKS クラスターの Kubernetes バージョンをダウングレードする

Amazon EKS クラスターの Kubernetes バージョンを、アップグレードから 7 日以内に以前のマイナーバージョンにロールバックできます。詳細については、「the section called “Kubernetes バージョンのロールバック”」を参照してください。7 日間の期間が過ぎると、ダウングレードすることはできません。代わりに、以前の Amazon EKS バージョンで新しいクラスターを作成し、ワークロードを移行します。

以前の Kubernetes バージョンにクラスターをロールバックする

Amazon EKS バージョンロールバックを使用すると、インプレースアップグレードの実行後に、クラスターの Kubernetes コントロールプレーンを以前のマイナーバージョンに戻すことができます。アップグレード後に、アプリケーションの非互換性、廃止された API の使用、予期しない動作などの問題が発生した場合は、ロールバックしてクラスターを既知の正常な状態に復元できます。

ロールバック中、Amazon EKS は Kubernetes API サーバーとコントロールプレーンのコンポーネントを以前のバージョンに戻すと同時に、すべての etcd データ、顧客ワークロード、および永続ボリュームを保持します。

ロールバックされるもの

以下のコンポーネントがロールバックされます:

- Kubernetes API サーバーバージョン
- コントロールプレーンコンポーネントとその設定
- プラットフォームバージョン (以前の Kubernetes バージョンの最新のプラットフォームバージョンに戻ります)
- EKS Auto Mode ワーカーノード。EKS Auto Mode を実行しているクラスターでは、Amazon EKS はコントロールプレーンを元に戻す前に Auto Mode ワーカーノードのロールバックを自動的に管理します。詳細については、「the section called “EKS Auto Mode クラスターをロールバックする”」を参照してください。

ロールバックされないもの

以下のコンポーネントはロールバックされません:

- etcd データ。クラスターの状態、リソース、設定はすべて保持されます。
- 顧客ワークロード。ポッド、デプロイ、サービスは引き続き実行されます。
- EKS アドオン。アドオンバージョンは変更されません。これらを個別に管理します。
- 永続的なボリュームとデータ。すべての顧客データはそのまま残ります。
- セルフマネージドノードとハイブリッドノード。これらをロールバックするのはユーザーの責任です。
- マネージドノードグループ `UpdateNodegroupVersion` API を使用して、これらを個別にロールバックする必要があります。

前提条件

クラスターをロールバックする前に、以下の条件をすべて満たす必要があります:

要件	詳細
7 日間のウィンドウ	ロールバックは、アップグレードの完了から 7 日以内に開始する必要があります。7 日を過ぎると、ロールバックは使用できなくなります。
アップグレードされたクラスター	クラスターは、インプレースアップグレードを通じて現在のバージョンにアップグレードされている必要があります。現在のバージョンで作成されたクラスターはロールバックできません。
単一バージョンのみ	ロールバックできるのは 1 つのマイナーバージョン (N から N-1) のみです。1.31 から 1.32 にアップグレードしてから 1.33 にアップグレードした場合、1.32 のみにロールバックでき、1.31 にはロールバックできません。
サポートされるバージョン	バージョンロールバックは、現在サポートされている Amazon EKS バージョンで使用できます。
延長サポートポリシー	サポートが拡張されているバージョンにロールバックするには、まずクラスターのアップグレードポリシーを EXTENDED に変更する必要があります。
延長サポート終了時の自動アップグレードなし	延長サポートの終了時にクラスターが自動的にアップグレードされた場合、以前のバージョンにロールバックすることはできません。クラスターが標準サポートの終了時に自動的にアップグレードされた場合は、ロールバックできますが、まずアップグレードポリシーを EXTENDED に変更する必要があります。
クラスターステータス	クラスターは ACTIVE ステータスである必要があります。別の更新の進行中にロールバックを開始することはできません。
EKS 機能の互換性	クラスターで有効になっている EKS 機能が以前のバージョンでサポートされていない場合、ロールバックリクエストは失敗します。このチェックを --force でバイパスすることはできません。

上記の要件に加えて、特定の条件では、`--force` フラグを使用してもロールバックが不可能になります。これらの条件には、クラスターが現在のバージョンで作成されたか、アップグレードから 7 日よりも経過しているか、クラスターが既に新しいバージョンに再度アップグレードされているか、下位互換性のない EKS 機能が現在のバージョン境界で有効になっていることが含まれます。

概要

Amazon EKS クラスターのロールバックプロセスの概要は以下のとおりです:

1. ロールバックの準備状況インサイトを確認して、ロールバックに影響する可能性のある問題を特定します。
2. ブロックの問題 (ERROR ステータスのインサイト) を解決するか、`--force` を使用してインサイトチェックをバイパスします。
3. アプリケーション、カスタムコントローラー、およびサードパーティーツールが以前の Kubernetes バージョンと互換性があることを確認します。
4. ワーカーノードがコントロールプレーンと同じ Kubernetes バージョンを実行している場合は、まずワーカーノードをロールバックします。
5. 以前の Kubernetes バージョンと互換性のないバージョンを実行しているアドオンがある場合は、互換性のあるバージョンにダウングレードします。
6. コントロールプレーンのロールバックを開始します。
7. ロールバックの進行状況をモニタリングします。

> **Important**
>
> EKS Auto Mode を実行しているクラスターでは、ステップ 4 は自動的に処理されます。ロールバックを開始すると、Amazon EKS はコントロールプレーンの前に Auto Mode ノードをロールバックします。詳細については、「the section called “EKS Auto Mode クラスターをロールバックする”」を参照してください。

ステップ 1: ロールバックの準備状況インサイトを確認する

Amazon EKS は、ポイントインタイムロールバック準備状況チェックに対してクラスターを自動的に評価し、クラスターインサイトを通じて問題を `ROLLBACK_READINESS` カテゴリに表示します。これらのインサイトは、アップグレードの実行後に表示され、7 日間のロールバック資格ウィンドウ中も引き続き利用できます。

ロールバックの準備状況インサイトの表示

AWS コンソール:

1. Amazon EKS コンソール を開きます。
2. クラスターを選択します。
3. [アップグレードインサイト] タブを選択します。ロールバックの準備状況インサイトは、アップグレード後に表示されます。
4. ERROR または WARNING ステータスのインサイトを確認します。

AWS CLI:

```
aws eks list-insights \
  --cluster-name my-cluster \
  --region us-west-2 \
  --filter '{"categories": ["ROLLBACK_READINESS"]}'
```

特定のインサイトの詳細を取得するには:

```
aws eks describe-insight \
  --cluster-name my-cluster \
  --region us-west-2 \
  --id <insight-id>
```

インサイトの更新

Amazon EKS では、24 時間ごとにインサイトが更新されます。Amazon EKS コンソールで [更新] ボタンを選択するか、CLI を使用して、問題を解決した後に手動で更新をトリガーできます。

```
aws eks start-insights-refresh \
  --cluster-name my-cluster \
  --region us-west-2
```

> **Note**
>
> Amazon EKS は、ロールバックを開始するとインサイトを自動的に更新し、最新のクラスター状態に対してチェックが実行されることを確認します。

インサイトステータスの動作

次の表に、各インサイトステータスの意味とロールバックへの影響を示します。

ステータス	意味	ロールバックへの影響
PASSING	このチェックで問題が検出されませんでした	ロールバックが許可されました
WARNING	潜在的な問題が検出されましたが、ブロックはされていません	ロールバックが許可されました (アドバイザリのみ)
ERROR	ブロックの問題が検出されました	解決されるまでロールバックがブロックされます。または、`--force` を使用してバイパスします
UNKNOWN	ステータスを判断できません	解決されるまでロールバックがブロックされます。または、`--force` を使用してバイパスします

ERROR または UNKNOWN ステータスのインサイトはロールバックをブロックします。PASSING または WARNING ステータスのインサイトは、ロールバックを妨げません。

ロールバック準備状況チェック

Amazon EKS は、ロールバックの準備状況インサイトの一部として一連のチェックを実行します。これらのチェックでは、API の使用の互換性 (フィールドレベルの変更検出を含む)、クラスターの状態、kubelet バージョンスキュー、kube-proxy バージョンスキュー、およびアドオンバージョンの互換性を評価します。EKS Auto Mode を実行しているクラスターでは、追加のチェックは NodePool 中断予算、do-not-disrupt 注釈、PodDisruptionBudget 設定を評価します。

--force フラグの使用

ロールバックの準備状況インサイトに ERROR ステータスが表示され、問題を解決せずに続行する場合は、`--force` フラグを使用してすべてのインサイトチェックをバイパスできます。

```
aws eks update-cluster-version \
  --name my-cluster \
  --kubernetes-version 1.30 \
```

```
  --force \
  --region us-west-2
```

> **Warning**
>
> `--force` を使用すると、すべてのインサイトチェック (ERROR、WARNING、UNKNOWN) がバイパスされ、ロールバックが直接続行されます。Amazon EKS は、インサイトチェックがバイパスされたときにロールバックの安全性を保証できません。発生した問題については、ユーザーが全責任を負うものとします。

`--force` フラグはインサイトチェックのみをバイパスします。7 日間のウィンドウ、作成バージョンチェック、シーケンシャルロールバックチェックなどの前提条件の検証はバイパスされません。Auto Mode クラスターでは、`--force` は中断コントロールを上書きしません。NodePool の中断予算、PDB、および do-not-disrupt 注釈は引き続き適用されます。

ステップ 2: ワーカーノードを準備する

コントロールプレーンをロールバックする前に、ワーカーノードがターゲットバージョンと互換性があることを確認してください。Kubernetes バージョンスキューポリシーでは、ワーカーノードがコントロールプレーンより新しいバージョンを実行できないことが必要です。

EKS 自動モード

対処は必要ありません。ロールバックを開始すると、Amazon EKS はコントロールプレーンの前に自動モードノードを自動的にロールバックします。詳細については、「the section called “EKS Auto Mode クラスターをロールバックする”」を参照してください。

マネージドノードグループ (MNG)

コントロールプレーンをロールバックする前に、マネージドノードグループを以前のバージョンにロールバックする必要があります。`UpdateNodegroupVersion` API を使用します:

```
aws eks update-nodegroup-version \
  --cluster-name my-cluster \
  --nodegroup-name my-nodegroup \
  --kubernetes-version 1.30 \
  --region us-west-2
```

ノードグループの更新は、構成された更新設定 (maxUnavailable または maxUnavailablePercentage) と更新戦略 (Rolling または Force) を尊重します。

セルフマネージドノードとハイブリッドノード

セルフマネージドノードとハイブリッドノードのロールバックはユーザーの責任です。コントロールプレーンをロールバックする前に、以前の Kubernetes バージョンを使用するようにノード AMI または設定を更新します。

Fargate

バージョンロールバックは Fargate ワーカーノードではサポートされていません。Fargate を使用するクラスターのコントロールプレーンはロールバックできますが、コントロールプレーンと同じ Kubernetes バージョンを実行している Fargate ポッドは、ERROR ステータスの kubelet バージョンスキューインサイトをトリガーします。

Amazon EKS は、Fargate ポッドを古い kubelet バージョンに自動的にロールバックすることはできません。

回避策: コントロールプレーンと同じ Kubernetes バージョンを実行している Fargate ポッドがある場合は、ロールバックを開始する前にそれらのポッドを削除します。次に、コントロールプレーンをロールバックします。残りのポッドは、再デプロイ時にロールバックバージョンで起動します。

または、--force を使用してインサイトチェックをバイパスします。ただし、kubelet バージョンスキュー違反を続行すると、それらのポッドが置き換えられるまで、Fargate ワークロードに予期しない動作が発生する可能性があります。

ステップ 3: クラスターコントロールプレーンをロールバックする

ロールバックを開始するには、AWS コンソール、AWS CLI、または EKS API を使用します。

AWS コンソールを使用してクラスターをロールバックする

1. Amazon EKS コンソール を開きます。
2. クラスターを選択します。
3. [アクション] ドロップダウンメニューを選択します。
4. ロールバッククラスターバージョンを選択します。
5. インサイトの警告を含むロールバックの概要を確認します。
6. ロールバックバージョンを選択します。

ロールバックは完了までに数分かかることがあります。Auto Mode クラスターの場合、ノードのロールバックフェーズに時間がかかる場合があります。詳細については、「the section called “EKS Auto Mode クラスターをロールバックする”」を参照してください。

AWS CLI を使用してクラスターをロールバックする

以前の (N-1) Kubernetes バージョンで既存の `update-cluster-version` コマンドを使用します。

```
aws eks update-cluster-version \
  --name my-cluster \
  --kubernetes-version 1.30 \
  --region us-west-2
```

レスポンスの例:

```
{
    "update": {
        "id": "e4091a28-ea14-48fd-a8c7-975aeb469e8a",
        "status": "InProgress",
        "type": "VersionRollback",
        "params": [
            {
                "type": "Version",
                "value": "1.30"
            },
            {
                "type": "PlatformVersion",
                "value": "eks.16"
            }
        ],
        "createdAt": "2026-05-12T16:56:01.082000-04:00",
        "errors": []
    }
}
```

> **Note**
>
> Amazon EKS は、インサイトデータが古い場合、ロールバックを実行する前にインサイトの更新を実行します。

ステップ 4: ロールバックの進行状況をモニタリングする

Amazon EKS コンソールまたは AWS CLI を使用して、クラスターロールバックのステータスをモニタリングできます。

AWS CLI:

```
aws eks describe-update \
  --name my-cluster \
  --region us-west-2 \
  --update-id e4091a28-ea14-48fd-a8c7-975aeb469e8a
```

AWS コンソール:

1. Amazon EKS コンソール を開きます。
2. クラスターを選択します。
3. [更新履歴] タブを選択します。
4. ロールバックに関連付けられた更新 ID を見つけて、現在のステータスを表示します。

ステータスの遷移

標準クラスターの場合 (自動モードなし):

```
InProgress # Successful
InProgress # Failed
```

Auto Mode クラスターでは、ノードがロールバックしている間はクラスターのステータスは ACTIVE のまま維持され、コントロールプレーンのロールバックが開始された場合にのみ UPDATING に変わります。describe-update を使用して、全体的なロールバックの進行状況を追跡します。詳細については、「the section called “EKS Auto Mode クラスターをロールバックする”」を参照してください。

Successful ステータスが表示されると、ロールバックは完了です。

考慮事項と警告

インサイトはベストエフォートかつポイントインタイムです

クラスターインサイトは、ロールバックがトリガーされた時点で評価されます。インサイトがチェックされた後であるが、ロールバックが完了する前にクラスターに変更を加えた場合 (新しい API を使用したリソースの作成など)、これらの変更は最初のインサイトチェックではキャプチャされず、ロールバックの完了後に問題が発生する可能性があります。

etcd データの保持

Amazon EKS はロールバック中に etcd データを保持します。`--force` フラグを使用してバイパスされた互換性のないリソースは保持され、ガベージコレクションされません。

延長サポート料金

標準サポートのバージョンから延長サポートのバージョンにロールバックすると、クラスターで延長サポート料金が発生します。例えば、1.30 (延長サポート) から 1.31 (標準サポート) にアップグレードしてから 1.30 にロールバックすると、延長サポート料金が再開されます。

ロールバックの責任共有モデル

Amazon EKS は、Kubernetes コントロールプレーンを希望のバージョンにロールバックします。責任共有モデルの一環として、ユーザーはアプリケーションの以前のバージョンとの互換性を検証する責任があります。

- Amazon EKS は、コントロールプレーンコンポーネントを安全に元に戻す責任があります。
- ユーザーは、アプリケーション、設定、および依存関係が以前のバージョンと互換性があることを確認する責任があります。
- ユーザーは、バージョン間の非互換性を確認し、クラスターへの影響を評価し、問題を軽減する必要があります。

CloudFormation スタックのロールバック動作

AWS CloudFormation スタックの更新が失敗し、スタックのロールバックがトリガーされた場合、より低い Kubernetes バージョンを指定する以前のテンプレートバージョンに戻しても、クラスターバージョンのロールバックはトリガーされません。バージョンのロールバックは、`UpdateClusterVersion` API、CLI、またはコンソールを使用して明示的に開始する必要があります。

ロールバックとアドオン

Amazon EKS は、クラスターバージョンのロールバック中にアドオンバージョンを自動的にロールバックしません。アドオンバージョンは個別に管理する必要があります。

コントロールプレーンをロールバックする前に:

1. ロールバックの準備状況インサイトを使用して、ターゲットバージョンとのアドオンの互換性を確認します。
2. アドオンバージョンが以前の Kubernetes バージョンと互換性がない場合は、まずそれをダウングレードします。

```
aws eks update-addon \
  --cluster-name my-cluster \
  --addon-name vpc-cni \
  --addon-version v1.22.4-eksbuild.3 \
  --region us-west-2
```

3. コントロールプレーンのロールバックが完了したら、すべてのアドオンが正しく機能していることを確認します。

> **Note**
>
> ロールバックの準備状況インサイトは、EKS マネージドアドオンバージョンのみをチェックします。セルフマネージドアドオンでは、ユーザーはロールバックする前にターゲットバージョンとの互換性を検証する責任があります。

関連リソース

- the section called “EKS Auto Mode クラスターをロールバックする”
- the section called “Kubernetes バージョンを更新する”
- the section called “クラスターのインサイト”
- Amazon EKS での Kubernetes バージョンのライフサイクルを理解する
- マネージドノードグループを更新する
- クラスターのアップグレードのベストプラクティス

EKS Auto Mode クラスターをロールバックする

EKS Auto Mode を実行しているクラスター上でバージョンロールバックを開始すると、Amazon EKS はコントロールプレーンを元に戻す前に Auto Mode ワーカーノードのロールバックを自動的に管理します。このページでは、Auto Mode ノードのロールバックの仕組み、それを高速化する方法、および必要に応じてキャンセルする方法について説明します。

前提条件、インサイトチェック、ロールバックプロセス全体など、バージョンロールバックに関する一般的な情報については、「the section called “Kubernetes バージョンのロールバック”」を参照してください。

Auto Mode のロールバックの仕組み

EKS Auto Mode は Karpenter ベースのシステムを使用して、Kubernetes バージョンのアップグレードやロールバックなどのワーカーノードインフラストラクチャを管理します。Auto Mode が有効になっているクラスター上で以前の (N-1) バージョンで `UpdateClusterVersion` を呼び出すと、EKS は次のシーケンスを実行します:

1. 前提条件を検証し、ロールバックの準備状況インサイトを更新します。
2. Karpenter ベースのシステムを使用してノードを目的のロールバックバージョンにドリフトし、設定された中断コントロールを尊重します。
3. すべてのノードが目的のロールバックバージョンの Kubernetes バージョンスキューポリシー内にあるようにした後、EKS はインサイトを再チェックし、コントロールプレーンのロールバックに進みます。

コントロールプレーンは現在の (新しい) バージョンのままで、ノードがロールバックしている間もトラフィックを正常に処理し続けます。Kubernetes バージョンスキューポリシーでは、kube-apiserver より古い最大 3 つのマイナーバージョンをノードで実行できるため、この中間状態は有効です。

> **Note**
>
> the section called “Kubernetes バージョンのロールバック” で説明されているのと同じ API とプロセスを使用して、ロールバックをトリガーします。Auto Mode ノードのロールバック用の個別の API はありません。

> **Note**
>
> ノードのロールバックフェーズ (ステップ 2) は、中断コントロールに応じて数分から 7 日かかる場合があります。設定されたタイムアウト内にノードのロールバックが完了しない場合、更新は失敗としてマークされます。

> **Note**
>
> ロールバックの進行中、他の顧客トリガーのコントロールプレーンの更新はブロックされます。別の更新を実行するには、まず CancelUpdate API を使用してロールバックをキャンセルします。

ロールバック中のクラスターステータス

フェーズ	クラスターステータス	起きていること
ノードのロールバックが進行中	ACTIVE	Karpenter はノードを以前のバージョンの AMI に置き換えています。コントロールプレーンは正常で、現在のバージョンでトラフィックを処理しています。
コントロールプレーンのロールバック	UPDATING	API サーバーとコントロールプレーンのコンポーネントは、以前のバージョンに戻されています。
ロールバックが完了	ACTIVE	クラスターは以前のバージョンで完全に動作しています。

クラスターのステータスは、ノードのロールバックフェーズ中も `ACTIVE` のままです。`ListUpdates` または `DescribeUpdate` を使用して、ロールバックが進行中かどうかを判断します。Amazon EKS コンソールで、クラスターに移動し、更新履歴タブを開いて、ロールバックに関連付けられた更新 ID のステータスを表示します。

ロールバック中に個々のノードの進行状況を追跡するには、Auto Mode ノードの Kubernetes バージョンを確認します。

```
kubectl get nodes -l karpenter.sh/nodepool=<nodepool-name> -o wide
```

中断コントロール

Auto Mode のロールバックは、既存のすべての中断コントロールを尊重します。これらのコントロールは、ノードを置き換える速さを決定し、ロールバック期間に大きな影響を与える可能性があります。

NodePool 中断予算

NodePool 中断予算は、同時に中断できるノードの数を制御します。ロールバック中、Karpenter はノードを以前のバージョンにドリフトするときにこれらの予算を尊重します。

- ドリフトに対する `nodes: 0` の予算は、ロールバックを無期限にブロックします。これにより、ERROR インサイトがトリガーされます。
- 制限された予算 (例: `nodes: 1`) はロールバックを遅らせますが、前進を許可します。

ノードの 10% を一度に置き換えることができる中断予算を持つ NodePool の例:

```
apiVersion: karpenter.sh/v1
kind: NodePool
metadata:
  name: default
spec:
  disruption:
    budgets:
      - nodes: "10%"
        reasons:
          - Drifted
```

NodePool 中断予算の設定の詳細については、「EKS Auto Mode 用のノードプールを作成する」および「Karpenter 中断予算」を参照してください。

PodDisruptionBudgets

Kubernetes PodDisruptionBudgets はノードの交換時に優先されます。PDB がポッドのエビクションを妨げる場合、ノードの中断は最大 TerminationGracePeriod 遅延します。

- `maxUnavailable: 0` がある PDB はノードの中断を遅延させます。これにより、WARNING インサイトがトリガーされます。
- PDB はロールバックを完全にブロックしませんが、大幅に遅くなる可能性があります。

詳細については、「PDB を使用して重要なワークロードを保護する」および「Kubernetes の PDB ドキュメント」を参照してください。

Do-not-disrupt 注釈

`karpenter.sh/do-not-disrupt` 注釈はノードまたはポッド上で設定できます。

ノード上: ノードの中断を無期限にブロックします。これにより ERROR インサイトがトリガーされ、そのノード上でロールバックを続行する前に削除する必要があります。

ポッド上: 最大 TerminationGracePeriod ノードの中断を遅延します。これにより WARNING インサイトがトリガーされますが、ロールバックは完全にブロックされません。

Karpenter の中断動作の詳細については、「Karpenter の中断ドキュメント」を参照してください。

ロールバックの高速化

ロールバックに予想以上に時間がかかる場合は、ロールバックの進行中に中断コントロールを調整できます。

NodePool 中断予算を増やす

NodePool リソースを編集して、より多くの同時ノード置換を許可します。

```
kubectl edit nodepool default
```

予算をより高い値に変更します:

```
spec:
  disruption:
    budgets:
      - nodes: "50%"
        reasons:
          - Drifted
```

ノードから do-not-disrupt 注釈を削除する

注釈を使用してノードを一覧表示し、それを削除します。

```
# List nodes with the annotation
kubectl get nodes -o json | jq '.items[] | select(.metadata.annotations["karpenter.sh/
do-not-disrupt"] == "true") | .metadata.name'

# Remove from a specific node
kubectl annotate node <node-name> karpenter.sh/do-not-disrupt-
```

PodDisruptionBudgets を調整する

PDB がポッドの削除を遅くしている場合は、一時的にそれらを調整します:

```
kubectl edit pdb <pdb-name> -n <namespace>
```

> **Warning**
>
> 中断コントロールを調整すると、アプリケーションの可用性の保証に影響します。本番環境で変更を加える前に、必ず影響を理解してください。

中断コントロールの管理の詳細については、「Amazon EKS Auto Mode でのポッドとノードの中断の防止」を参照してください。

ロールバックのキャンセル

ロールバックは、ノードがロールバックしている間のみ、キャンセル可能な状態になります。このフェーズで CancelUpdate を使用してオペレーションを停止できます。

```
aws eks cancel-update \
  --name my-cluster \
  --update-id <update-id> \
  --region us-west-2
```

キャンセル動作

側面	動作
キャンセル可能な場合	Auto Mode ノードがロールバックされている間のみ (コントロールプレーンのロールバックが開始される前)
セマンティクス	ベストエフォートが停止。ノードのロールバックオペレーションを停止します。
中断中のノード	ノードがキャンセル時に中断中である場合、現在のオペレーションを完了します。
キャンセル後のステータス	`Cancelling` から `Cancelled` への移行を更新します。
クラスターステータス	全体を通して `ACTIVE` のままです。

キャンセル後

キャンセルが成功すると、ノードは通常どおり現在のクラスターバージョンにドリフトします。更新は `Cancelling` から `Cancelled` に移行します。

キャンセル後、すぐに以下を行えます:

- ロールバックを再試行する (7 日間の資格ウィンドウ内である限り)。
- 別のクラスター更新を実行する。
- クラスターを現在のバージョンのままにする。

キャンセルできない場合

ノードのロールバックが既に完了していてコントロールプレーンのロールバックが開始されている場合、または更新が `Successful` または `Failed` ステータスで既に完了している場合、キャンセルは失敗します。

> **Note**
>
> CloudFormation と Terraform は CancelUpdate API を直接サポートしていません。IaC を通じて開始されたロールバックをキャンセルする必要がある場合は、API を直接呼び出す必要があります。

ロールバックのタイムアウト

Auto Mode ノードのロールバックには、`rollbackConfig` の `timeoutMinutes` パラメータによって制御される設定可能なタイムアウトがあります。デフォルトのタイムアウトは 720 分 (12 時間) です。値は 120 分 (2 時間) ~ 10080 分 (7 日) で設定できます。タイムアウトは、最小許容プロパティです。つまり、指定した時間よりも早く発生することはありませんが、その後すぐに発生する可能性があります。

```
aws eks update-cluster-version \
  --name my-cluster \
  --kubernetes-version 1.30 \
  --rollback-config timeoutMinutes=1440 \
  --region us-west-2
```

すべてのノードが指定されたタイムアウト内にロールバックを完了していない場合:

1. ロールバックはタイムアウトします。
2. ノードは現在のクラスターバージョンにドリフトし始めます。
3. コントロールプレーンは現在のバージョンのままです (ロールバックされていません)。
4. 更新ステータスは `Failed` に移行します。

タイムアウト後、元のアップグレードから 7 日間のロールバック資格ウィンドウ内にある場合は、ロールバックを再試行できます。実際には、ノードのロールバックが 7 日目にタイムアウトする場合、ロールバック資格ウィンドウも 7 日間であるため、それが期限切れになっている可能性があります。

タイムアウトを回避するには、ロールバックを開始する前にロールバックの準備状況インサイトを確認してください。これらのインサイトは、ノードのロールバックプロセスを遅らせる可能性のある中断予算や注釈について警告します。

--force フラグと Auto Mode

`UpdateClusterVersion` 上の `--force` フラグは、クラスターインサイトチェックのみをバイパスします。Auto Mode ノードの中断動作には影響しません。

`--force` を使用しても、次のようになります:

- NodePool 中断予算は引き続き優先されます。
- PodDisruptionBudgets は引き続き優先されます。
- Do-not-disrupt 注釈は引き続き尊重されます。
- 7 日間のノードロールバックタイムアウトは引き続き適用されます。

ノードのロールバックを高速化する唯一の方法は、中断コントロール自体を調整することです。詳細については、「the section called “ロールバックの高速化”」を参照してください。

IaC タイムアウトの競合

Infrastructure-as-Code ツールには、Auto Mode のロールバック期間と競合する可能性のあるタイムアウト制限があります。CloudFormation では、リソースごとに最大 36 時間許可されます。オペレーションがタイムアウトすると、CloudFormation はこれを no-op として扱います。これにより、クラスターがドリフト状態になり、テンプレートに実際のクラスターバージョンが反映されない可能性があります。バージョンロールバックは明示的に開始される必要があります。Terraform Enterprise/Cloud のタイムアウトは約 24 時間ですが、クライアント側のタイムアウトは認証情報の有効期限やその他の要因によって異なる場合があります。

ロールバック期間を IaC ツールに合わせるには、`rollbackConfig` の `timeoutMinutes` パラメータを使用して適切なタイムアウトを設定します。IaC ツールがタイムアウトした場合は、CancelUpdate API を直接使用してコントロールを回復します。制限付きの中断予算がある場合は、IaC の代わりに CLI または API から直接ロールバックを開始することを検討してください。

ノードのロールバック中のシステム更新

Auto Mode ノードがロールバックしている間、EKS はコントロールプレーンの安全性と可用性を引き続き維持します。

顧客トリガーの更新 (UpdateClusterVersion や UpdateClusterConfig など) は、ノードのロールバックの進行中にブロックされます。優先度の高い更新を実行する必要がある場合は、まず

CancelUpdate を使用してロールバックをキャンセルしてから、更新を実行し、資格ウィンドウ内にある場合はロールバックを再開します。

関連リソース

- the section called “Kubernetes バージョンのロールバック”
- EKS Auto Mode の概要
- EKS 自動モードI 用のノードプールを作成する
- EKS Auto Mode クラスターの Kubernetes バージョンを更新する
- Amazon EKS Auto Mode でのポッドとノードの中断の防止
- EKS 自動モードI のトラブルシューティング
- Karpenter の中断に関するドキュメント
- Kubernetes PodDisruptionBudgets

クラスターを削除

Amazon EKS クラスターの使用が終了したら、関連付けられたリソースを削除して、不要なコストが発生しないようにする必要があります。

eksctl、AWS マネジメントコンソール、またはAWS CLI を使用してクラスターを削除できます。

考慮事項

- クラスター作成者が削除されたためにエラーが発生した場合は、この記事 を参照して解決してください。
- Amazon Managed Service for Prometheus リソースはクラスターのライフサイクル外にあるため、クラスターとは別途維持する必要があります。クラスターを削除するときは、対象コストを抑えるために、該当するスクレイパーも削除してください。詳細については、「Amazon Managed Service for Prometheus ユーザーガイド」の「スクレイパーの検出と作成」を参照してください。
- 接続されたクラスターを削除するには、the section called “クラスターの登録解除” を参照してください。
- クラスターを削除する前に、クラスターの削除保護が無効になっていることを確認してください。
- クラスターに EKS 機能 (ACK、Argo CD、または kro) がある場合は、クラスターを削除する前にそれらを削除する必要があります。詳細については、「the section called “機能を削除する”」を参照してください。

EKS Auto Mode に関する考慮事項

- EC2 マネージドインスタンスを含む、すべての EKS Auto Mode ノードが削除されます
- すべてのロードバランサーが削除されます

詳細については、the section called “EKS Auto Mode を無効にする” を参照してください。

前提となる手順

クラスターを削除する前に最初に実行する必要があるステップを以下に示します。これらのステップは、クラスターの削除に使用する方法に関係なく適用されます。

1. クラスターで実行されているすべてのサービスを一覧表示します。

```
kubectl get svc --all-namespaces
```

2. 関連付けられた EXTERNAL-IP 値のあるサービスすべてを削除します。これらのサービスの前面には Elastic Load Balancing のロードバランサーが置かれているため、そのロードバランサーや関連するリソースを適切に解放するためには、これらのサービスを Kubernetes から削除する必要があります。*service-name* を説明に従って各サービスの名前に置き換えます。

```
kubectl delete svc service-name
```

3. Ingress リソースも削除します。Ingress リソースを削除しない場合、クラスターを削除しても Application Load Balancer は残ります。*ingress-name* をお使いの Ingress リソースの名前に置き換えます。

```
kubectl get ingress --all-namespaces
```

```
kubectl delete ing ingress-name
```

クラスターの削除 (eksctl)

この手順には、eksctl バージョン 0.215.0 以降が必要です。お使いのバージョンは、以下のコマンドを使用して確認できます。

```
eksctl version
```

eksctl のインストールまたはアップグレードの手順については、eksctl ドキュメントの「インストール」を参照してください。

1. 前提条件の手順を実行します。その後、以下のコマンドを使用して、クラスターとそれに関連するノードを削除します。この時、*prod* は実際のクラスター名に置き換えてください。

```
eksctl delete cluster --name prod
```

出力:

```
[#]  using region region-code
[#]  deleting EKS cluster "prod"
[#]  will delete stack "eksctl-prod-nodegroup-standard-nodes"
[#]  waiting for stack "eksctl-prod-nodegroup-standard-nodes" to get deleted
[#]  will delete stack "eksctl-prod-cluster"
[#]  the following EKS cluster resource(s) for "prod" will be deleted: cluster. If in
 doubt, check CloudFormation console
```

クラスターを削除する (AWS コンソール)

1. 前提条件の手順を実行します。その後、すべてのノードグループ、Fargate プロファイル、および機能を削除します。
   a. Amazon EKS コンソールを開きます。
   b. 左のナビゲーションペインで Amazon EKS [クラスター] を選択し、クラスターのタブ付きリストから、削除するクラスター名を選択します。
   c. [コンピューティング] タブを選択し、削除するノードグループを選択します。[削除] を選択し、ノードグループの名前を入力し、[削除] を選択します。クラスター内のすべてのノードグループを削除します。

> **Note**
>
> ここでリストに表示されるノードグループは、マネージド型ノードグループのみです。

   d. 削除する [Fargate プロファイル] を選択し、[削除] を選択し、プロファイルの名前を入力した上で [削除] を選択します。クラスター内のすべての Fargate プロファイルを削除します。

e. [機能] タブを選択し、削除する機能を選択します。[機能を削除] を選択し、機能の名前を入力して、[削除] を選択します。クラスター内のすべての機能を削除します。詳細については、「the section called “機能を削除する”」を参照してください。

2. すべてのセルフマネージド型ノードの AWS CloudFormation スタックを削除します。

   a. AWS CloudFormation コンソールを開きます。

   b. 削除するノードスタックを選択し、[削除] を選択します。

   c. [スタックの削除] 確認ダイアログボックスで、[スタックを削除] をクリックします。クラスター内のすべてのセルフマネージド型ノードスタックを削除します。

3. クラスターを削除します。

   a. Amazon EKS コンソールを開きます。

   b. 削除するクラスターを選択し、[削除] を選択します。

   c. クラスターの削除確認画面で、[削除] を選択します。

4. (オプション) VPC AWS CloudFormation スタックを削除します。

   a. AWS CloudFormation コンソールを開きます。

   b. 削除する VPC スタックを選択し、[削除] をクリックします。

   c. [スタックの削除] 確認ダイアログボックスで、[スタックを削除] をクリックします。

クラスターを削除する (AWS CLI)

1. 前提条件の手順を実行します。その後、すべてのノードグループ、Fargate プロファイル、および機能を削除します。

   a. 次のコマンドを使用して、クラスター内のノードグループを一覧表示します。

```
aws eks list-nodegroups --cluster-name my-cluster
```

> **Note**
>
> ここでリストに表示されるノードグループは、マネージド型ノードグループのみです。

   b. 次のコマンドを使用して、各ノードグループを削除します。クラスター内のすべてのノードグループを削除します。

```
aws eks delete-nodegroup --nodegroup-name my-nodegroup --cluster-name my-cluster
```

c. 次のコマンドを使用して、クラスターの Fargate プロファイルを一覧表示します。

```
aws eks list-fargate-profiles --cluster-name my-cluster
```

d. 次のコマンドを使用して、各 Fargate プロファイルを削除します。クラスター内のすべての Fargate プロファイルを削除します。

```
aws eks delete-fargate-profile --fargate-profile-name my-fargate-profile --
cluster-name my-cluster
```

e. 次のコマンドを使用して、クラスター上の機能を一覧表示します。

```
aws eks list-capabilities --cluster-name my-cluster
```

f. 次のコマンドを使用して、各機能を削除します。クラスター内のすべての機能を削除します。詳細については、「the section called “機能を削除する”」を参照してください。

```
aws eks delete-capability --cluster-name my-cluster --capability-name my-
capability-name
```

2. すべてのセルフマネージド型ノードの AWS CloudFormation スタックを削除します。

a. 次のコマンドを使用して、使用可能な AWS CloudFormation スタックを一覧表示します。出力結果から、ノードテンプレート名を見つけます。

```
aws cloudformation list-stacks --query "StackSummaries[].StackName"
```

b. 次のコマンドを使用して、各ノードスタックを削除します。この際、*node-stack* の部分は実際のノードスタック名に置き換えます。クラスター内のすべてのセルフマネージド型ノードスタックを削除します。

```
aws cloudformation delete-stack --stack-name node-stack
```

3. 以下のコマンドを使用して、クラスターを削除します。*my-cluster* は実際のクラスター名に置き換えてください。

```
aws eks delete-cluster --name my-cluster
```

4. (オプション) VPC AWS CloudFormation スタックを削除します。

a. 次のコマンドを使用して、使用可能な AWS CloudFormation スタックを一覧表示します。結果の出力で VPC テンプレート名を見つけます。

```
aws cloudformation list-stacks --query "StackSummaries[].StackName"
```

b. 以下のコマンドを使用して、VPC スタックを削除します。*my-vpc-stack* は実際の VPC スタック名に置き換えてください。

```
aws cloudformation delete-stack --stack-name my-vpc-stack
```

EKS クラスターを誤って削除しないように保護する

EKS クラスターを誤って削除すると、Kubernetes クラスターのオペレーションが損なわれる可能性があります。

EKS クラスターを誤って削除しないように保護できるようになりました。クラスターで削除保護を有効にすると、クラスターを削除する際は、まず削除保護を無効化しなければならなくなります。

削除保護の目的は、事故を防ぐことです。クラスターを削除する権限を持つユーザーを慎重に制限する必要があります。

削除保護が有効になっているアクティブなクラスターを削除しようとすると、`InvalidRequestException` が返されます。

> **Important**
>
> クラスターで削除保護を有効にした場合、削除保護を削除してからクラスターを削除するためには、UpdateClusterConfig と DeleteCluster IAM アクセス許可の両方が必要です。

> **Note**
>
> クラスターの状態が作成中、失敗、または削除中の場合は、削除保護が有効になっている場合でもクラスターを削除できます。

既存のクラスターの削除保護を有効にするには

これは、アクティブなステータスのクラスターでのみ実行できます。

```
aws eks update-cluster-config --name <cluster-name> --region <aws-region> --deletion-
protection
```

既存のクラスターの削除保護を無効にするには

```
aws eks update-cluster-config --name <cluster-name> --region <aws-region> --no-
deletion-protection
```

クラスター API サーバーエンドポイント

このトピックでは、Amazon EKS クラスターの Kubernetes API サーバーエンドポイントや上限へのプライベートアクセスを有効化する方法、あるいは、インターネットからのパブリックアクセスを完全に無効化する方法について説明します。

新しいクラスターを作成すると、Amazon EKS によってマネージド型の Kubernetes API サーバーのエンドポイントが作成されます。ユーザーはこのエンドポイントを、(kubectl などの Kubernetes 管理ツールにより) クラスターとの通信に使用します。デフォルトでは、この API サーバーエンドポイントはインターネットに公開されます。API サーバーへのアクセスの保護には、AWS Identity and Access Management (IAM) と、ネイティブの Kubernetes ロールベースアクセスコントロール (RBAC) が組み合わせて使用されます。このエンドポイントはクラスターパブリックエンドポイントと呼ばれます。また、クラスタープライベートエンドポイントもあります。クラスタープライベートエンドポイントの詳細についてはセクション the section called “クラスタープライベートエンドポイント” を参照してください。

IPv6 クラスターエンドポイント形式

EKS は2024 年 10 月以降に作成された新しい IPv6 クラスターに対して、次の形式で一意のデュアルスタックエンドポイントを作成します。IPv6 クラスターはクラスターの IP ファミリー (ipFamily) 設定で IPv6 を選択するクラスターです。

Example

AWS

EKS クラスターパブリック / プライベートエンドポイント: eks-cluster.*region*.api.aws

AWS GovCloud (US)

EKS クラスターパブリック / プライベートエンドポイント: eks-cluster.*region*.api.aws

Amazon Web Services in China

EKS クラスターパブリック / プライベートエンドポイント: eks-cluster.*region*.api.amazonwebservices.com.cn

> **Note**
>
> デュアルスタッククラスターエンドポイントは2024 年 10 月に導入されました。IPv6 クラスターの詳細については、「the section called “IPv6”」を参照してください。2024 年 10 月より前に作成されたクラスターは代わりに次のエンドポイント形式を使用します。

IPv4 クラスターエンドポイント形式

EKS は、クラスターの IP ファミリー (ipFamily) 設定で IPv4 を選択したクラスターごとに、次の形式で一意のエンドポイントを作成します:

Example

AWS

EKS クラスターパブリック / プライベートエンドポイント eks-cluster.*region*.eks.amazonaws.com

AWS GovCloud (US)

EKS クラスターパブリック / プライベートエンドポイント eks-cluster.*region*.eks.amazonaws.com

Amazon Web Services in China

EKS クラスターパブリック / プライベートエンドポイント eks-cluster.*region*.amazonwebservices.com.cn

> **Note**
>
> 2024 年 10 月以前、IPv6 クラスターはこのエンドポイント形式も使用していました。これらのクラスターではパブリックエンドポイントとプライベートエンドポイントの両方において、このエンドポイントから解決できるのは IPv4 アドレスのみです。

クラスタープライベートエンドポイント

Kubernetes API サーバーへのプライベートアクセスを有効にすると、ノードと API サーバー間のすべての通信が VPC 内で行われるようになります。インターネットから API サーバーにアクセスできる IP アドレスを制限したり、API サーバーへのインターネットアクセスを完全に無効にしたりできます。

> **Note**
>
> このエンドポイントは Kubernetes API サーバー用であり、AWS API と通信するための従来の AWS PrivateLink エンドポイントではないため、Amazon VPC コンソール上にはエンドポイントとして表示されません。

クラスターでエンドポイントへのプライベートアクセスを有効にすると、Amazon EKS によって自動的に ルート 53 のプライベートホストゾーンが作成され、クラスターの VPC に関連付けられます。このプライベートホストゾーンは Amazon EKS によって管理され、アカウントの ルート 53 リソースには表示されません。プライベートホストゾーンが API サーバーに正しくトラフィックをルーティングするためにはVPC で `enableDnsHostnames` と `enableDnsSupport` が `true` に設定され、VPC 用に設定された DHCP オプションで、ドメイン名サーバーリストに `AmazonProvidedDNS` が含まれている必要があります。詳細についてはAmazon VPC ユーザーガイドの「VPC の DNS サポートを表示および更新する」を参照してください。

API サーバーエンドポイントのアクセス要件は新しいクラスターを作成するときに定義できます。また、クラスターの API サーバーエンドポイントのアクセスは随時更新できます。

> **Note**
>
> エンドポイントアクセスは、Kubernetes API サーバーに到達できるユーザーを制御します。コントロールプレーンからの出力トラフィックが VPC 内のリソース (ウェブフックサーバーや OIDC プロバイダーなど) に到達する方法も管理する場合は、the section called “コントロールプレーンエグレス” を参照してください。

クラスターエンドポイントのアクセスの変更

既存クラスターのエンドポイントのアクセスを変更するにはこのセクションの手順に従ってください。次の表はサポートされている API サーバーエンドポイントのアクセスの組み合わせとそれらに関連付けられている動作を示しています。

エンドポイントのパブリックアクセス	エンドポイントのプライベートアクセス	行動
有効	無効	• これは新しい Amazon EKS クラスターのデフォルトの動作です。 • クラスターの VPC 内から送信される Kubernetes API リクエスト (ノードからコントロールプレーンへの通信など) は、VPC の外部に伝達されますが、Amazon のネットワークからは離れません。 • クラスター API サーバーにはインターネットからアクセスできます。必要に応じて、パブリックアクセス CIDR のリストを使用して、CIDR ブロックのリストでパブリックエンドポイントへのアクセスを制御できます。特定の CIDR ブロックへのアクセスを制限する場合は、同時にプライベートエンドポイントも有効にすることをお勧めします。あるいは、ノードと (存在する場合は) Fargate Pod がパブリックエンドポイントにアクセスするためのアドレスを対象の CIDR ブロックに含めるようにします。 • ワーカーノードとコントロールプレーン間のトラフィックには、標準 AWS データ転送料金が適用されます。Amazon EKS は、接続のコントロールプレーン側でデータ転送コストを吸収します。詳細については、「the section called “Amazon EKS の料金”」を参照してください。
有効	有効	• クラスターの VPC 内の Kubernetes API リクエスト (ノードとコントロールプレーン間の通信など) では、プライベート VPC エンドポイントが使用されます。ク

エンドポイントのパブリックアクセス	エンドポイントのプライベートアクセス	行動
		ラスターセキュリティグループに対してルールを使用して、プライベートエンドポイントへのアクセスを制御します。 • クラスター API サーバーにはインターネットからアクセスできます。必要に応じて、パブリックアクセス CIDR のリストを使用して、CIDR ブロックのリストでパブリックエンドポイントへのアクセスを制御できます。 • Amazon EKS クラスターでハイブリッドノードを使用している場合は、パブリックとプライベートの両方のエンドポイントアクセスを有効にすることはお勧めしません。ハイブリッドノードは VPC の外部で実行されているため、クラスターエンドポイントはパブリック IP アドレスに解決されます。ハイブリッドノードを持つクラスターにはパブリックまたはプライベートのいずれかのエンドポイントアクセスを使用することをお勧めします。

エンドポイントのパブリックアクセス	エンドポイントのプライベートアクセス	行動
無効	有効	• クラスター API サーバーへのすべてのトラフィックはクラスターの VPC または接続されたネットワーク内から送信する必要があります。 • インターネットから API サーバーへのパブリックアクセスは存在しません。kubectl コマンドはすべて、VPC または接続されたネットワーク内から実行する必要があります。接続オプションについては、「the section called “プライベート専用 API サーバーへのアクセス”」を参照してください。 • クラスターセキュリティグループに対してルールを使用して、プライベートエンドポイントへのアクセスを制御します。パブリックアクセス CIDR はプライベートエンドポイントに影響しないことに注意してください。 • クラスターの API サーバーエンドポイントはパブリック DNS サーバーによって VPC のプライベート IP アドレスに解決されます。これまではエンドポイントは VPC 内からしか解決できませんでした。 エンドポイントが既存のクラスターの VPC 内のプライベート IP アドレスに解決されない場合は次の操作を実行できます: • パブリックアクセスを有効にし、再度無効にします。この操作はクラスターに対して 1 回行うだけで済みます。それ以降、エンドポイントはプライベート IP アドレスに解決されます。 • クラスターを更新します。

エンドポイントアクセスコントロール

以下のエンドポイントアクセスを制御する方法のいずれも、それぞれのエンドポイントにのみ影響することに注意してください。

クラスターセキュリティグループ

クラスターセキュリティグループは、kubelet API とプライベートエンドポイントへの接続という 2 つのタイプの接続を制御します。kubelet API への接続は、kubectl attach、kubectl cp、kubectl exec、kubectl logs、kubectl port-forward の各コマンドで使用されます。クラスターセキュリティグループがパブリックエンドポイントに影響することはありません。

パブリックアクセス CIDR

パブリックアクセス CIDR は、CIDR ブロックのリストでパブリックエンドポイントへのアクセスを制御します。パブリックアクセス CIDR はプライベートエンドポイントに影響しないことに注意してください。パブリックアクセス CIDR の動作は、以下で説明するように、IPv6 クラスターと IPv4 クラスターが作成された日付によって異なります。

パブリックエンドポイントの CIDR ブロック (**IPv6** クラスター)

パブリックエンドポイントはデュアルスタックであるため、IPv6 クラスターのパブリックエンドポイントに IPv6 および IPv4 CIDR ブロックを追加できます。これは、2024 年 10 月以降に作成した ipFamily が IPv6 に設定されている新しいクラスターにのみ適用されます。これらのクラスターは、新しいエンドポイントドメイン名 api.aws で識別できます。

パブリックエンドポイントの CIDR ブロック (**IPv4** クラスター)

IPv4 クラスターのパブリックエンドポイントに IPv4 CIDR ブロックを追加できます。IPv4 クラスターのパブリックエンドポイントに IPv6 CIDR ブロックを追加することはできません。試みると、EKS は次のエラーメッセージを返します: The following CIDRs are invalid in publicAccessCidrs

パブリックエンドポイントの CIDR ブロック (2024 年 10 月以前に作成された **IPv6** クラスター)

2024 年 10 月以前に作成した古い IPv6 クラスターのパブリックエンドポイントに IPv4 CIDR ブロックを追加できます。これらのクラスターは eks.amazonaws.com エンドポイントで識別できます。2024 年 10 月以前に作成したこれらの古い IPv6 クラスターのパブリックエンドポイントに IPv6 CIDR ブロックを追加することはできません。試みると、EKS は次のエラーメッセージを返します: The following CIDRs are invalid in publicAccessCidrs

プライベート専用 API サーバーへのアクセス

クラスターの Kubernetes API サーバーエンドポイントに対するパブリックアクセスを無効にした場合は、VPC または接続されたネットワーク内からのみ API サーバーにアクセスできます。Kubernetes API サーバーエンドポイントにアクセスする方法はいくつかあります。

接続されたネットワーク

AWS トランジットゲートウェイまたはその他の接続オプションを使用してネットワークを VPC に接続し、接続されたネットワークのコンピュータを使用します。接続されたネットワークからのポート 443 でのイングレストラフィックを許可するためのルールが、Amazon EKS コントロールプレーンセキュリティグループに含まれていることを確認する必要があります。

Amazon EC2 踏み台ホスト

Amazon EC2 インスタンスをクラスターの VPC のパブリックサブネットで起動し、SSH 経由でそのインスタンスにログインして `kubectl` コマンドが実行できます。詳細については AWS での Linux 踏み台ホストを参照してください。踏み台ホストからのポート 443 でのイングレストラフィックを許可するためのルールが、Amazon EKS コントロールプレーンセキュリティグループに含まれていることを確認する必要があります。詳細については、「the section called “セキュリティグループの要件”」を参照してください。

踏み台ホスト用に `kubectl` を設定するときにはクラスターの RBAC 設定に既にマッピングされている AWS 認証情報を使用するか、踏み台が使用する IAM プリンシパル を RBAC 設定に追加してから、エンドポイントのパブリックアクセスを削除します。詳細については the section called “Kubernetes API アクセス” および the section called “許可されていないか、アクセスが拒否されました (`kubectl`)” を参照してください。

AWS Cloud9 IDE

AWS Cloud9 はブラウザだけでコードを記述、実行、およびデバッグできるクラウドベースの統合開発環境 (IDE) です。クラスターの VPC に AWS Cloud9 IDE を作成し、その IDE を使用してクラスターと通信できます。詳細については、「AWS Cloud9 で環境を作成する」を参照してください。Amazon EKS コントロールプレーンセキュリティグループに、IDE セキュリティグループからのポート 443 でのイングレストラフィックを許可するためのルールが、含まれていることを確認する必要があります。詳細については、「the section called “セキュリティグループの要件”」を参照してください。

AWS Cloud9 IDE 用に `kubectl` を設定するときにはクラスターの RBAC 設定に既にマッピングされている AWS 認証情報を使用するか、IDE が使用する IAM プリンシパルを RBAC 設定に追

加してから、エンドポイントのパブリックアクセスを削除してください。詳細については、「the section called “Kubernetes API アクセス”」および「the section called “許可されていないか、アクセスが拒否されました (kubectl)”」を参照してください。

AWSCloudShell

Amazon EKS コンソールのクラスターの詳細ページで [接続] を選択します。プライベートクラスターの場合、CloudShell はクラスターのプライベート API サーバーエンドポイントに到達できる VPC 環境を起動します。詳細については、「the section called “kubectl を使用してクラスターにアクセスする”」を参照してください。

GitHub でこのページを編集する

クラスター API サーバーエンドポイントへのネットワークアクセスを設定する

次のセクションでは、AWS マネジメントコンソール または AWS CLI を使用して、クラスター API サーバーのエンドポイントアクセスを変更できます。

エンドポイントアクセスの設定 - AWS コンソール

1. Amazon EKS コンソールを開きます。
2. クラスター名を選択すると、そのクラスターの情報を表示されます。
3. [ネットワーキング] タブを選択し、[エンドポイントアクセスを管理] を選択します。
4. [プライベートアクセス] の場合はクラスターの Kubernetes API サーバーエンドポイントに対するプライベートアクセスを有効にするか無効にするかを選択してください。プライベートアクセスを有効にした場合、クラスターの VPC 内から送信される Kubernetes API リクエストはプライベート VPC エンドポイントを使用します。パブリックアクセスを無効にするにはプライベートアクセスを有効にする必要があります。
5. [パブリックアクセス] の場合はクラスターの Kubernetes API サーバーエンドポイントに対するパブリックアクセスを有効にするか無効にするかを選択してください。パブリックアクセスを無効にすると、クラスターの Kubernetes API サーバーはクラスター VPC 内からのみリクエストを受信できます。
6. (オプション) [パブリックアクセス] で有効化を行うと、インターネットからパブリックエンドポイントと通信するためのアドレスを指定できるようになります。[詳細設定] を選択してください。*203.0.113.5/32* などの CIDR ブロックを入力します。ブロックに予約済みアドレスを含めることはできません。[ソースの追加] を選択すると、追加のブロックを入力できます。指

定できる CIDR ブロックには最大数があります。詳細については、「the section called “Service Quotas”」を参照してください。ブロックをまったく指定しない場合、パブリック API サーバーエンドポイントは、IPv4 (0.0.0.0/0) と、さらにデュアルスタック IPv6 クラスターの IPv6 (::/0) の両方のすべての IP アドレスからのリクエストを受信します。CIDR ブロックを使用してパブリックエンドポイントへのアクセスを制限する場合は、同時にプライベートエンドポイントアクセスも有効化することをお勧めします。これにより、ノードと (使用している場合は) Fargate Pods がクラスターと通信できるようになります。プライベートエンドポイントが有効になっていない場合はパブリックアクセスエンドポイントの CIDR ソースに、VPC からの出力ソースを含める必要があります。例えば、プライベートサブネットに NAT ゲートウェイを介してインターネットと通信するノードがある場合、パブリックエンドポイントで許可された CIDR ブロックの一部として、NAT ゲートウェイのアウトバウンド IP アドレスを追加する必要があります。

7. [更新] を選択して終了します。

エンドポイントアクセスの設定 - AWS CLI

AWS CLI バージョン 1.27.160 以降を使用して、次のステップを実行してください。現在のバージョンはaws --version で確認できます。AWS CLI をインストールまたはアップグレードするには「AWS CLI のインストール」を参照してください。

1. 次の AWS CLI コマンドを使用してクラスター API サーバーエンドポイントのアクセスを更新します。クラスター名と必要なエンドポイントアクセス値を置き換えます。endpointPublicAccess=true を設定した場合は、必要に応じて publicAccessCidrs の 1 つの CIDR ブロックまたは複数の CIDR ブロックからなるカンマ区切りリストを入力できます。ブロックに予約済みアドレスを含めることはできません。CIDR ブロックを指定すると、パブリック API サーバーエンドポイントはリストされたブロックからのリクエストのみを受信します。指定できる CIDR ブロックには最大数があります。詳細については「the section called “Service Quotas”」を参照してください。CIDR ブロックを使用してパブリックエンドポイントへのアクセスを制限する場合は同時にプライベートエンドポイントアクセスも有効化することをお勧めします。これにより、ノードと (存在している場合は Fargate Pods がクラスターと通信できるようになります。プライベートエンドポイントが有効になっていない場合はパブリックアクセスエンドポイントの CIDR ソースに、VPC からの出力ソースを含める必要があります。例えば、プライベートサブネットに NAT ゲートウェイを介してインターネットと通信するノードがある場合、パブリックエンドポイントで許可された CIDR ブロックの一部として、NAT ゲートウェイのアウトバウンド IP アドレスを追加する必要があります。CIDR ブロックをまったく指定しない場合、パブリック API サーバーエンドポイントはすべての (0.0.0.0/0) IP アドレスからのリクエストと、さらにデュアルスタック IPv6 クラスターの IPv6 (::/0) も受信します。

> **Note**
>
> 次のコマンドはAPI サーバーエンドポイントの 1 つの IP アドレスからのプライベートアクセスとパブリックアクセスを有効にします。*203.0.113.5/32* は単一の CIDR ブロック、またはネットワークアクセスを制限する CIDR ブロックのカンマ区切りリストに置き換えます。

```
aws eks update-cluster-config \
    --region region-code \
    --name my-cluster \
    --resources-vpc-config
 endpointPublicAccess=true,publicAccessCidrs="203.0.113.5/32",endpointPrivateAccess=true
```

出力例は次のとおりです。

```
{
    "update": {
        "id": "e6f0905f-a5d4-4a2a-8c49-EXAMPLE00000",
        "status": "InProgress",
        "type": "EndpointAccessUpdate",
        "params": [
            {
                "type": "EndpointPublicAccess",
                "value": "true"
            },
            {
                "type": "EndpointPrivateAccess",
                "value": "true"
            },
            {
                "type": "publicAccessCidrs",
                "value": "[\"203.0.113.5/32\"]"
            }
        ],
        "createdAt": 1576874258.137,
        "errors": []
    }
}
```

2. 次のコマンドでエンドポイントアクセス更新のステータスをモニタリングします。この際、以前のコマンドで返ったクラスター名と更新 ID を使用します。ステータスが Successful と表示されたら、更新は完了です。

```
aws eks describe-update \
    --region region-code \
    --name my-cluster \
    --update-id e6f0905f-a5d4-4a2a-8c49-EXAMPLE00000
```

出力例は次のとおりです。

```
{
    "update": {
        "id": "e6f0905f-a5d4-4a2a-8c49-EXAMPLE00000",
        "status": "Successful",
        "type": "EndpointAccessUpdate",
        "params": [
            {
                "type": "EndpointPublicAccess",
                "value": "true"
            },
            {
                "type": "EndpointPrivateAccess",
                "value": "true"
            },
            {
                "type": "publicAccessCidrs",
                "value": "[\"203.0.113.5/32\"]"
            }
        ],
        "createdAt": 1576874258.137,
        "errors": []
    }
}
```

GitHub でこのページを編集する

EKS クラスターに WiWindows ノードをデプロイする

Amazon EKS クラスターの Windows サポートを有効にして管理し、Linux コンテナと一緒に Windows コンテナを実行する方法について説明します。

考慮事項

Windows ノードをデプロイする前に、以下の考慮事項を確認してください。

- EKS Auto Mode は Windows ノードをサポートしていません
- `HostProcess` ポッドを使用して Windows ノードでホスト ネットワークを使用できます。詳細については、Kubernetes ドキュメントの「Create a Windows HostProcess Pod」を参照してください。
- CoreDNS など、Linux でのみ動作するコアシステム Pod を実行するには、Amazon EKS クラスターに 1 つ以上の Linux ノードまたは Fargate ノードが含まれている必要があります。
- `kubelet` および `kube-proxy` イベントログは `EKS Windows` Event Log にリダイレクトされ、200 MB の制限に設定されます。
- Windows ノードで実行されている Pod で、[セキュリティグループを個別のポッドに割り当てる] を使用することはできません。
- Windows ノードでカスタムネットワークを使用することはできません。
- `IPv6` を Windows ノードで使用することはできません。
- Windows ノードは、ノードごとに 1 つの Elastic Network Interface をサポートします。デフォルトでは、Windows ノードごとに実行できる Pod の数は、ノードのインスタンスタイプの Elastic Network Interface ごとに使用できる IP アドレスの数から 1 を引いた数に等しくなります。詳細については、「Amazon EC2 ユーザーガイド」の「各インスタンスタイプのネットワークインターフェイスあたりの IP アドレス」を参照してください。
- Amazon EKS クラスターでは、ロードバランサーを持つ単一のサービスが、最大 1,024 個のバックエンド Pod をサポートできます。各ポッドには固有の IP アドレスがあります。OS ビルド 17763.2746 で始まる Windows Server 更新プログラム以降、以前の 64 個の Pod という上限はなくなりました。
- Windows コンテナは Fargate の Amazon EKS Pod ではサポートされていません。
- Windows を搭載した Amazon EKS Hybrid Nodes をホストのオペレーティングシステムとして使用することはできません。
- `vpc-resource-controller` ポッドからはログを取得できません。コントローラーをデータプレーンにデプロイしていたときには取得できていました。

- IPv4 アドレスが新しいポッドに割り当てられるまでにはクールダウン期間があります。これは、古い kube-proxy ルールが原因で、同じ IPv4 アドレスを持つ古いポッドにトラフィックが流れるのを防ぎます。
- コントローラーのソースは GitHub で管理されます。コントローラーに関する投稿や、問題の登録を行うには、GitHub の project にアクセスしてください。
- Windows マネージド型ノードグループのカスタム AMI ID を指定するときは、AWS IAM オーセンティケーター設定マップに eks:kube-proxy-windows を追加します。詳細については、「the section called “AMI ID を指定する場合の制限と条件”」を参照してください。
- サブネットで使用可能な IPv4 アドレスを保持することが重要な場合は、「EKS Best Practices Guide - Windows Networking IP Address Management」のガイダンスを参照してください。
- EKS アクセスエントリに関する考慮事項
  - Windows ノードで使用するアクセスエントリには、タイプとして EC2_WINDOWS を指定する必要があります。詳細については、「the section called “アクセスエントリを作成する”」を参照してください。

    Windows ノード用のアクセスエントリを作成するには、次のように指定します。

```
aws eks create-access-entry --cluster-name my-cluster --principal-arn
 arn:aws:iam::111122223333:role/<role-name> --type EC2_Windows
```

前提条件

- 既存のクラスター。
- CoreDNS を実行するには、クラスターに少なくとも 1 つ (推奨値は少なくとも 2 つ) の Linux ノードまたは Fargate Pod が必要です。レガシー Windows サポートを有効にする場合は、CoreDNS の実行に Linux ノードを使用する必要があります (Fargate Pod は使用できません)。
- 既存の Amazon EKS クラスター IAM ロール。

Windows サポートを有効にする

1. クラスターに Amazon Linux ノードがなく、Pod のセキュリティグループを使用する場合は、次のステップに進みます。それ以外の場合は、クラスターのロールに AmazonEKSVPCResourceController マネージドポリシーがアタッチされていることを確認します。*eksClusterRole* は、自分のクラスターのロール名に置き換えてください。

```
aws iam list-attached-role-policies --role-name eksClusterRole
```

出力例は次のとおりです。

```
{
    "AttachedPolicies": [
        {
            "PolicyName": "AmazonEKSClusterPolicy",
            "PolicyArn": "arn:aws:iam::aws:policy/AmazonEKSClusterPolicy"
        },
        {
            "PolicyName": "AmazonEKSVPCResourceController",
            "PolicyArn": "arn:aws:iam::aws:policy/AmazonEKSVPCResourceController"
        }
    ]
}
```

前の出力のようにポリシーがアタッチされている場合は、次のステップをスキップします。

2. AmazonEKSVPCResourceController マネージド型ポリシーを Amazon EKS クラスターの IAM ロールにアタッチします。*eksClusterRole* は、自分のクラスターのロール名に置き換えてください。

```
aws iam attach-role-policy \
  --role-name eksClusterRole \
  --policy-arn arn:aws:iam::aws:policy/AmazonEKSVPCResourceController
```

3. VPC CNI ConfigMap を更新して Windows IPAM を有効にします。Helm チャートを使用するか、Amazon EKS アドオンとして VPC CNI がクラスターにインストールされた場合、ConfigMap を直接変更できない場合がありますのでご注意ください。Amazon EKS アドオンの設定に関する詳細については、「the section called “カスタマイズ可能なフィールド”」を参照してください。

   a. 次の内容で、*vpc-resource-controller-configmap.yaml* という名前のファイルを作成します。

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: amazon-vpc-cni
  namespace: kube-system
data:
```

```
  enable-windows-ipam: "true"
```

b. ConfigMap をクラスターに適用する

```
kubectl apply -f vpc-resource-controller-configmap.yaml
```

4. クラスターの認証モードが aws-auth configmap を有効にするように設定されている場合は、次の手順を実行します。

   - aws-auth ConfigMap に、Windows ノードのインスタンスロールに eks:kube-proxy-windows RBAC 権限グループを含めるマッピングが含まれていることを確認します。次のコマンドを使用してインストールします。

     ```
     kubectl get configmap aws-auth -n kube-system -o yaml
     ```

     出力例は次のとおりです。

     ```
     apiVersion: v1
     kind: ConfigMap
     metadata:
       name: aws-auth
       namespace: kube-system
     data:
       mapRoles: |
         - groups:
           - system:bootstrappers
           - system:nodes
           - eks:kube-proxy-windows # This group is required for Windows DNS resolution
      to work
           rolearn: arn:aws:iam::111122223333:role/eksNodeRole
           username: system:node:{{EC2PrivateDNSName}}
     [...]
     ```

     グループの下に eks:kube-proxy-windows がリストされているはずです。グループが指定されていない場合は、必要なグループを含むように ConfigMap を更新するか、作成する必要があります。aws-auth ConfigMapの詳細については、「the section called “aws-auth ConfigMap をクラスターに適用する”」を参照してください。

5. クラスターの認証モードが aws-auth configmap を無効にするように設定されている場合は、EKS アクセスエントリを使用できます。Windows インスタンスで使用する新しいノードロー

ルを作成すると、EKS によって EC2_WINDOWS タイプのアクセスエントリが自動的に作成されます。

Windows ポッドをデプロイする

クラスターにポッドをデプロイするときに、さまざまなノードタイプを混在させて実行する場合に使用するオペレーティングシステムを指定する必要があります。

Linux Pod の場合は、マニフェストで以下のノードセレクターテキストを使用します。

```
nodeSelector:
        kubernetes.io/os: linux
        kubernetes.io/arch: amd64
```

Windows Pod の場合は、マニフェストで以下のノードセレクターテキストを使用します。

```
nodeSelector:
        kubernetes.io/os: windows
        kubernetes.io/arch: amd64
```

サンプルアプリケーションをデプロイして、使用されているノードセレクタを確認できます。

Windows ノードでより高い Pod 密度をサポートする

Amazon EKS では、各 Pod に VPC の IPv4 アドレスが割り当てられます。このため、ノード上でさらに多くの Pod を実行するのに十分なリソースがある場合でも、ノードにデプロイできる Pod の数の上限は、使用可能な IP アドレスによって制限されます。Windows ノードでサポートされる Elastic Network Interface は 1 つだけなので、デフォルトでは Windows ノードで使用できる IP アドレスの最大数は次のようになります。

```
Number of private IPv4 addresses for each interface on the node - 1
```

1 つの IP アドレスがネットワークインターフェイスのプライマリ IP アドレスとして使用されるため、Pod に割り当てることはできません。

IP プレフィックスの委任を有効にすると、Windows ノードで Pod 密度をより高めることができます。この機能により、セカンダリ IPv4 アドレスを割り当てる代わりに、プライマリネットワークイ

ンターフェイスに /28IPv4 プレフィックスを割り当てることができます。IP プレフィックスを割り当てると、ノードで使用できる IPv4 アドレスの最大数が次のように増加します。

```
(Number of private IPv4 addresses assigned to the interface attached to the node - 1) *
 16
```

使用可能な IP アドレス数が大幅に増加したため、使用可能な IP アドレスによってノード上の Pod の数をスケーリングできる機能が制限されることはありません。詳細については、「the section called “IP アドレスを増やす”」を参照してください。

Windows サポートを無効にする

1. クラスターに Amazon Linux ノードが含まれていて、ポッドのセキュリティグループをそれらとともに使用する場合は、このステップをスキップしてください。

   クラスターのロールから AmazonVPCResourceController マネージド IAM ポリシーを削除します。*eksClusterRole* は、実際に使用するクラスターのロールに置き換えてください。

   ```
   aws iam detach-role-policy \
       --role-name eksClusterRole \
       --policy-arn arn:aws:iam::aws:policy/AmazonEKSVPCResourceController
   ```

2. amazon-vpc-cni ConfigMap で Windows IPAM を無効にします。

   ```
   kubectl patch configmap/amazon-vpc-cni \
                       -n kube-system \
                       --type merge \
                       -p '{"data":{"enable-windows-ipam":"false"}}'
   ```

インターネットアクセスが制限されたプライベートクラスターをデプロイする

このトピックでは、AWS Cloud にデプロイされているが、アウトバウンドインターネットアクセスがない Amazon EKS クラスターをデプロイする方法について説明します。AWS Outposts にローカルクラスターがある場合、このトピックの代わりに「the section called “ノード”」を参照してください。

> **Note**
>
> このトピックでは、データプレーンエグレス (アウトバウンドインターネットアクセスのないノードとワークロード) について説明します。クラスターコントロールプレーンからのエグレストラフィックが VPC に到達する方法を制御するには、the section called “コントロールプレーンエグレス” を参照してください。

Amazon EKS でのネットワークに詳しくない場合は、「De-mystifying cluster networking for Amazon EKS worker nodes (Amazon EKS ワーカーノードのクラスターネットワークを解明する)」を参照してください。クラスターにアウトバウンドインターネットアクセスがない場合、次の要件を満たす必要があります。

クラスターアーキテクチャの要件

- クラスターは VPC 内のコンテナレジストリからイメージを取得する必要があります。VPC 内に Amazon Elastic Container Registry を作成し、そこにコンテナイメージをコピーしてノードの取得元にすることができます。詳細については、the section called “リポジトリにイメージをコピーする” を参照してください。
- クラスターでは、エンドポイントのプライベートアクセスが有効になる必要があります。これは、ノードをクラスターエンドポイントに登録するために必要です。エンドポイントのパブリックアクセスはオプションです。詳細については、the section called “クラスターエンドポイントのアクセス” を参照してください。

ノードの要件

- セルフマネージド型 Linux ノードおよび Windows ノードには、起動する前に次のブートストラップ引数を含める必要があります。これらの引数は Amazon EKS のイントロスペクションをバイパスするため、VPC 内からの Amazon EKS API へのアクセスは不要です。

  a. 次のコマンドを使用して、クラスターのエンドポイントの値を確認します。`#######` の部分は自分のクラスター名に置き換えます。

  ```
  aws eks describe-cluster --name my-cluster --query cluster.endpoint --output text
  ```

  出力例は次のとおりです。

```
https://EXAMPLE108C897D9B2F1B21D5EXAMPLE.sk1.region-code.eks.amazonaws.com
```

b. 次のコマンドを使用して、クラスターの認証機関の値を確認します。`#######` の部分は、自分のクラスター名に置き換えます。

```
aws eks describe-cluster --name my-cluster --query cluster.certificateAuthority --
output text
```

返された出力は長い文字です。

c. NodeConfig オブジェクト内の `apiServerEndpoint` および `certificateAuthority` の値を、前のコマンドで返された出力の値に置き換えます。セルフマネージド型 Amazon Linux 2023 ノードを起動する際にブートストラップ引数を指定する方法の詳細については、the section called “Amazon Linux” および the section called “Windows” を参照してください。

- Linux ノードの場合:

```
---
MIME-Version: 1.0
Content-Type: multipart/mixed; boundary="BOUNDARY"

--BOUNDARY
Content-Type: application/node.eks.aws

---
apiVersion: node.eks.aws/v1alpha1
kind: NodeConfig
spec:
  cluster:
    name: my-cluster
    apiServerEndpoint: https://EXAMPLE108C897D9B2F1B21D5EXAMPLE.sk1.region-
code.eks.amazonaws.com
    certificateAuthority: Y2VydGlmaWNhdGVBdXRob3JpdHk=
    ...
```

その他の引数については、GitHub の「ブートストラップのスクリプト」を参照してください。

- Windows ノードの場合:

> **Note**
>
> カスタムサービス CIDR を使用している場合は `-ServiceCIDR` パラメータを使用して指定する必要があります。そうしないと、クラスター内の Pod の DNS 解決が失敗します。

```
-APIServerEndpoint cluster-endpoint -Base64ClusterCA certificate-authority
```

その他の引数については、the section called “ブートストラップスクリプトの設定パラメータ”を参照してください。

- クラスターの `aws-auth ConfigMap` は VPC 内から作成する必要があります。エントリの作成と `aws-auth ConfigMap` への追加の詳細については、ターミナルで `eksctl create iamidentitymapping --help` と入力してください。サーバーに `ConfigMap` が存在しない場合は、このコマンドを使用して ID マッピングを追加したときに `eksctl` によって作成されます。

ポッドの要件

- Pod Identity – EKS Pod Identity と共に設定されたポッドは、EKS Auth API から認証情報を取得します。アウトバウンドインターネットアクセスがない場合は、EKS Auth API の VPC エンドポイントを作成して使用する必要があります: `com.amazonaws.region-code.eks-auth`。EKS と EKS Auth VPC エンドポイントの詳細については、the section called “ AWS プライベートリンク”を参照してください。
- IRSA - サービスアカウントの IAM ロール で設定されたポッドは、AWS Security Token Service (AWS STS) API コールから認証情報を取得します。アウトバウンドインターネットアクセスがない場合は、VPC 内で AWS STS VPC エンドポイントを作成して使用する必要があります。ほとんどの AWS v1 SDK は、AWS STS VPC エンドポイントを使用しないグローバル AWS STS エンドポイント (`sts.amazonaws.com`) をデフォルトで使用します。AWS STS VPC エンドポイントを使用するには、リージョンの AWS STS エンドポイント (`sts.region-code.amazonaws.com`) を使用するように SDK を構成する必要がある場合があります。詳細については、「the section called “STS エンドポイント”」を参照してください。

  IRSA をセットアップするには、VPC 内からクラスターの OIDC 検出/JWKS エンドポイントに到達する必要があります。例えば、クラスターの IAM OIDC ID プロバイダーを作成したり、独自のトークンバリデーターを実行したりすると、これに到達します。アウトバウンドインターネッ

トアクセスがない場合は、クラスターの OIDC エンドポイント用の VPC エンドポイントを作成して使用します: `com.amazonaws.region-code.oidc-eks`。これは AWS STS VPC エンドポイントとは別のものです。STS は AWS 内からクラスターの JWKS を取得するため、OIDC エンドポイントは VPC が発信する OIDC トラフィックをプライベート化し、STS エンドポイントは `AssumeRoleWithWebIdentity` 呼び出しをプライベート化します。詳細については、「the section called “AWS PrivateLink を使用してクラスター OIDC エンドポイントにアクセスする”」を参照してください。

- Pod がアクセスする必要のあるすべての AWS サービスについて、クラスターの VPC サブネットに VPC インターフェイスエンドポイントが必要です。詳細については、「インターフェイス VPC エンドポイントを使用して AWS サービスにアクセスする」を参照してください。下表には、一般的に使用されるサービスとエンドポイントがリスト表示されています。エンドポイントの詳細なリストについては、AWS PrivateLink ガイド の「AWS PrivateLink と連携する AWS サービス」を参照してください。

  VPC エンドポイントのために [プライベート DNS 名を有効にする] ことをお勧めします。これにより、ワークロードはパブリック AWS サービスエンドポイントを問題なく引き続き使用できます。

サービス	エンドポイント
Amazon EC2	com.amazonaws.*region-code* .ec2
Amazon Elastic Container Registry (コンテナイメージの取得用)	com.amazonaws.*region-code* .ecr.api、com.amazonaws.*region-code* .ecr.dkr、および com.amazonaws.*region-code* .s3
Amazon Application Load Balancer および Network Load Balancer	com.amazonaws.*region-code* .elasticloadbalancing
(オプション) AWS X-Ray (トレースを AWS X-Ray に送信する場合に必要)	com.amazonaws.*region-code* .xray
(オプション) Amazon SSM (SSM エージェントがノード管理タスクを実施する場合に必要。SSH の代替)	com.amazonaws.*region-code* .ssm

サービス	エンドポイント
Amazon CloudWatch Logs (ノードログとポッドログを Amazon CloudWatch Logs に送信する場合に必要)	com.amazonaws.*region-code* .logs
AWS Security Token Service (サービスアカウントに IAM ロールを使用している場合に必要)	com.amazonaws.*region-code* .sts
Amazon EKS クラスター OIDC エンドポイント (VPC 内からサービスアカウントの IAM ロールをセットアップするために必要)	com.amazonaws.*region-code* .oidc-eks
Amazon EKS Auth (Pod Identity 関連付けを使用する場合には必須)	com.amazonaws.*region-code* .eks-auth
Amazon EKS	com.amazonaws.*region-code* .eks
Amazon Route 53	com.amazonaws.route53

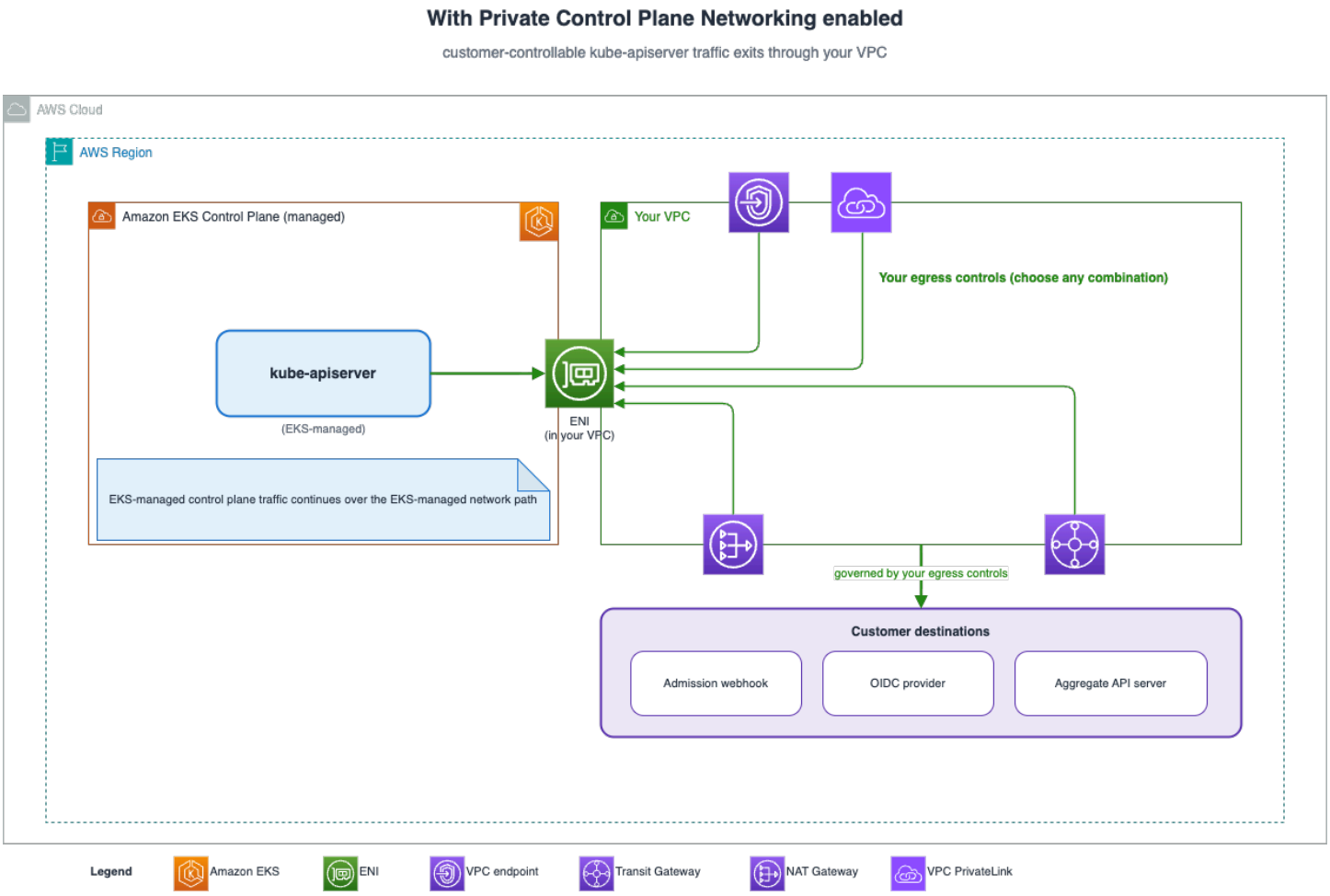
- セルフマネージド型ノードはすべて、必要な VPC インターフェイスエンドポイントを持つサブネットにデプロイする必要があります。マネージド型ノードグループを作成する場合には、VPC インターフェイスエンドポイントのセキュリティグループでサブネットの CIDR を許可するか、ノードのセキュリティグループを作成し VPC インターフェイスエンドポイントのセキュリティグループに追加する必要があります。
- EFS ストレージ - Pod で Amazon EFS ボリュームを使用する場合、「Store an elastic file system with Amazon EFS」をデプロイする前に、Amazon EKS クラスターと同じ AWS リージョンを使用するように、ドライバーの kustomization.yaml ファイルを変更して、コンテナイメージを設定する必要があります。
- EKS 最適化 AMI を使用した場合は、上記の表に挙げた `ec2` エンドポイントを有効にする必要があります。あるいは、ノードの DNS 名を手動で設定することもできます。最適化 AMI では、EC2 API を使用して、ノードの DNS 名を自動的に設定します。
- AWS ロードバランサーコントローラー を使用して、AWS Application Load Balancer (ALB) および Network Load Balancer をプライベートクラスターにデプロイできます。デプロイするときは、コマンドラインフラグ を使用して、`enable-shield`、`enable-waf`、および `enable-wafv2` を false に設定する必要があります。Ingress オブジェクトのホスト名を使用した 証明書の検出 は、サポートされていません。これは、コントローラーが VPC インターフェイスエンドポイントを持たない AWS Certificate Manager に到達する必要があるからです。

このコントローラーでは、Fargate で必要な IP ターゲットを持つネットワークロードバランサをサポートします。詳細については、the section called “アプリケーション負荷分散” および the section called “ネットワークロードバランサーを作成する” を参照してください。

- Cluster Autoscaler がサポートされています。クラスター自動スケーラー Pod をデプロイする場合は、コマンドラインに `--aws-use-static-instance-list=true` が含まれていることを確認してください。詳細については、GitHub で「静的インスタンス・リストを使用」を参照してください。ワーカーノード VPC には、AWS STS VPC エンドポイントと自動スケーリング VPC エンドポイントも含める必要があります。
- 一部のコンテナソフトウェア製品では、AWS Marketplace Metering Service にアクセスする API コールを使用して、使用状況をモニタリングします。プライベートクラスターではこれらの呼び出しが許可されないため、これらのコンテナタイプはプライベートクラスターには使用できません。

コントロールプレーンエグレスルーティングの設定

デフォルトでは、Amazon EKS は Kubernetes コントロールプレーンから VPC 内のリソースへのエグレスネットワーキングを管理します。コントロールプレーンエグレスルーティングを使用して、この動作を変更し、ネットワークパスを自分で管理します。これにより、コントロールプレーンの Elastic Network Interface (ENI) からのトラフィックが VPC リソースに到達する方法を完全に制御できます。独自の NAT ゲートウェイ、ファイアウォール、または検査アプライアンスを介してルーティングできます。



エグレスルーティングモード

Amazon EKS は、次のコントロールプレーンエグレスルーティングモードをサポートしています。

モード	説明
AWS_MANAGED	デフォルトの動作。Amazon EKS は、コントロールプレーン ENI からのエグレスパスを管理します。コントロールプレーントラフィック用に NAT ゲートウェイやその他のルーティングインフラストラクチャを設定する必要はありません。
CUSTOMER_ROUTED	VPC サブネットのコントロールプレーンからのエグレスパスを管理します。ユーザーは、コントロールプレーンが必要なエンドポイント (ウェブフックサーバー、OIDC プロバイダー、その他のリソースなど) に到達できることを確認する責任があります。NAT ゲートウェイ、NAT インスタンス、トランジットゲートウェイ、ファイアウォールアプライアン

モード	説明
	スなどのエグレスパスを指定します。このトラフィックを許可するルートテーブル、ネットワーク ACL、およびセキュリティグループルールも設定します。

> **Important**
>
> CUSTOMER_ROUTED モードでは、ユーザーは、コントロールプレーンからの適切なネットワーク接続を確保する責任があります。VPC ネットワークの設定ミスにより、コントロールプレーンオペレーションが失敗する可能性があります。これらの設定ミスには、欠落したエグレスパス、制限的なネットワーク ACL、または誤ったセキュリティグループが含まれます。影響を受けるオペレーションには、アドミッションウェブフック呼び出しと OIDC 認証が含まれます。

前提条件

VPC とサブネットは、標準の Amazon EKS ネットワーキング要件を満たしている必要があります。詳細については、「the section called “VPC およびサブネットの要件”」を参照してください。

CUSTOMER_ROUTED モードでは、Kubernetes API サーバーは、クロスアカウントネットワークインターフェイスを介して顧客向けトラフィックをアウトバウンドに送信します。Amazon EKS は、コントロールプレーンからノードへの通信用にこれらのインターフェイスをサブネットに既に作成しています。このトラフィックには、アドミッションウェブフックと OIDC プロバイダーへの呼び出しが含まれます。Amazon EKS は、個別のエグレスネットワークインターフェイスを作成しません。このモードでは、既存のインターフェイスの使用方法が変更されます。これらのインターフェイスを保持するサブネットは、次の要件を満たしている必要があります。

- サブネットには、コントロールプレーンが到達する必要があるエンドポイント (ウェブフックサーバーや OIDC プロバイダーなど) へのルートがある必要があります。VPC 外のエンドポイントでは、これは通常、エグレスデバイスへのデフォルトルートを意味します。デフォルトルートは、IPv4 では `0.0.0.0/0` で、IPv6 では `::/0` です。エグレスデバイスは、NAT ゲートウェイ、NAT インスタンス、ファイアウォール、または一元化された出力 VPC へのトランジットゲートウェイである場合があります。エグレスデバイスの選択はユーザーが行います。Amazon EKS では、パスが機能することのみが必要です。

- クロスアカウントネットワークインターフェイス上のセキュリティグループは、ワークロードに必要なポート (ウェブフックや OIDC プロバイダーのポート 443 など) でのアウトバウンドトラフィックを許可する必要があります。
- サブネット上のネットワーク ACL は、アウトバウンドトラフィックと、リターントラフィックに対応するインバウンドエフェメラルポート範囲を許可する必要があります。

CUSTOMER_ROUTED モードでは、コントロールプレーンは VPC の DNS 設定を使用してホスト名を解決します。これにより、コントロールプレーンは Route 53 プライベートホストゾーンと Route 53 Resolver エンドポイントを介して転送されるオンプレミス DNS のエンドポイントに到達できます。

- VPC DHCP オプションセットでは、ドメインネームサーバーリストに AmazonProvidedDNS が含まれる必要があります。これは、コントロールプレーンが VPC 内の DNS 名を解決するために必要です。クラスターが外部ウェブフックエンドポイントまたはパブリック DNS 名を持つ OIDC プロバイダーを使用している場合、リゾルバーはパブリックホスト名も解決する必要があります。リゾルバーが VPC 解決とパブリック DNS 解決の両方を処理できることを確認します。

次の表は、コントロールプレーンが CUSTOMER_ROUTED モードで VPC 経由で送信するトラフィックをまとめたものです。

トラフィック	送信先	ポート	注意事項
アドミッションウェブフック	ウェブフックエンドポイント (顧客定義の URL)	443 (通常)	ウェブフックが設定されている場合のみ。エンドポイントが外部の場合、エグレスデバイスを介して VPC から離れます。
OIDC 検出	OIDC 発行者 URL	443	OIDC プロバイダーが設定されている場合のみ。発行者が外部の場合、エグレスデバイスを介して VPC から離れます。
集約 API サーバー	カスタマー API サーバーエンドポイント	443	設定されている場合にのみ。エンドポイントが外部の場合、エグレスデバ

トラフィック	送信先	ポート	注意事項
			イスを介して VPC から離れます。
Kubelet API	ワーカーノードの IP アドレス	10250	これはクラスター ENI を介したコントロールプレーンとノード間のトラフィックであり、エグレスデバイスを通過しません。これには、ルートテーブル、セキュリティグループ、ネットワーク ACL が、コントロールプレーンとノード間のクラスター ENI を介したトラフィックを許可する必要があります。

> **Note**
>
> このテーブルにリストされているトラフィックのみがエグレス設定の影響を受けます。EKS マネージドコントロールプレーントラフィック (etcd、CloudWatch Logs、および内部 EKS サービスとの通信など) は AWS マネージドネットワークパスを経由し、VPC 設定の影響を受けません。

顧客ルーティングエグレスでクラスターを作成する

新しいクラスターを作成するときに、コントロールプレーンのエグレスモードを指定できます。

Example

AWS CLI

以下のコマンドを実行してください。**##########**を、独自の値に置き換えます。

```
aws eks create-cluster \
```

```
    --name my-cluster \
    --role-arn arn:aws:iam::111122223333:role/myAmazonEKSClusterRole \
    --resources-vpc-config "subnetIds=subnet-ExampleID1,subnet-
ExampleID2,securityGroupIds=sg-ExampleID1,controlPlaneEgressMode=CUSTOMER_ROUTED" \
    --kubernetes-network-config "ipFamily=ipv4" \
    --region region-code
```

IPv6 クラスターには ipFamily=ipv6 を使用できます。CUSTOMER_ROUTED モードで IPv6 を使用する場合は、IPv4 トラフィック用の NAT ゲートウェイに加えて、IPv6 トラフィック用のエグレス専用インターネットゲートウェイがサブネットにあることを確認します。

Example

AWS マネジメントコンソール

1. Amazon EKS コンソールを開きます。
2. [クラスターを追加]、[作成] の順に選択します。
3. [ネットワーク] ページの [コントロールプレーンエグレス] で、[顧客ルーティング] を選択します。
4. 残りのクラスター設定を完了し、[作成] を選択します。

AWS CloudFormation では、ResourcesVpcConfig で ControlPlaneEgressMode: CUSTOMER_ROUTED を設定します。このフィールドの Terraform サポートは、AWS プロバイダーの今後のリリースで利用可能になります。

> **Note**
>
> CUSTOMER_ROUTED への切り替えは一方向操作です。クラスターで顧客ルーティングエグレスを有効にした後は、AWS_MANAGED に戻すことはできません。

既存のクラスターの更新

update-cluster-config コマンドを使用して、既存のクラスターのコントロールプレーンエグレスモードを変更できます。

```
aws eks update-cluster-config \
    --name my-cluster \
    --resources-vpc-config "controlPlaneEgressMode=CUSTOMER_ROUTED" \
```

```
    --region region-code
```

更新のステータスを監視します。

```
aws eks describe-update \
    --name my-cluster \
    --update-id update-id \
    --region region-code
```

ステータスが Successful を示すと更新が完了します。更新タイプは ControlPlaneEgressUpdate です。更新は通常、10 分以内に完了します。

> **Important**
>
> CUSTOMER_ROUTED への切り替えは一方向操作です。クラスターで顧客ルーティングエグレスを有効にした後は、AWS_MANAGED に戻すことはできません。
> 切り替える前に、VPC が the section called “前提条件” の要件を満たしていることを確認します。更新後にコントロールプレーンが必要なエンドポイントへの接続が失われると、アドミッションウェブフック呼び出しや OIDC 認証などのオペレーションが失敗する可能性があります。

IPv6 に関する考慮事項

顧客ルーティングエグレスで IPv6 クラスターを実行する場合は、IPv4 と IPv6 の両方のエグレスパスを設定する必要があります。

CUSTOMER_ROUTED エグレスで IPv6 クラスター (ipFamily=ipv6) を実行する場合:

- コントロールプレーン ENI には、IPv4 アドレスと IPv6 アドレスの両方が割り当てられます。
- IPv4 と IPv6 の両方のエグレスパスを設定する必要があります。
  - IPv4: エグレスデバイス (NAT ゲートウェイなど) へのデフォルトルート (0.0.0.0/0)。
  - IPv6: IPv6 エグレスデバイス (エグレス専用インターネットゲートウェイなど) への ::/0 ルート。
- セキュリティグループと NACL は、両方の IP バージョンでトラフィックを許可する必要があります。
- OIDC プロバイダーまたはウェブフックエンドポイントが IPv4 専用である場合は、IPv4 NAT が機能していることを確認します。

考慮事項

顧客ルーティングコントロールプレーンエグレスを使用する場合は、次の点に注意してください。

- ユーザーの責任: CUSTOMER_ROUTED モードでは、コントロールプレーンから外部エンドポイントへのネットワークパスはユーザーが管理します。そのパスが破損した場合、そのパスに依存するコントロールプレーンオペレーション (アドミッションウェブフック呼び出しや OIDC 認証など) は、接続を復元するまで失敗する可能性があります。
- VPC 内部トラフィックは影響を受けない: クラスター ENI を介したコントロールプレーンとノード (ポート 10250 の kubelet API など) 間のトラフィックは、エグレスデバイスに依存しません。
- EKS Auto Mode: コントロールプレーンのエグレスルーティングは、コントロールプレーンアーキテクチャが同一であるため、標準クラスターと自動モードクラスターの両方で同じように機能します。
- EKS 機能: EKS 機能 (ArgoCD、ACK、KRO など) は、個別の AWS マネージドインフラストラクチャで実行されます。EKS 機能コントローラーからのトラフィックは、この機能によって VPC 経由でルーティングされません。
- オブザーバビリティ: VPC またはクラスターサブネットで VPC フローログを有効にすると、VPC を経由する送信トラフィックを監視できます。これには、ウェブフックと OIDC エンドポイントへの呼び出しが含まれます。VPC フローログが有効になっていない場合、このトラフィックはログに記録されません。

IAM 条件キー

Amazon EKS は `eks:controlPlaneEgressMode` 条件キーをサポートしています。このキーを IAM ポリシーまたはサービスコントロールポリシー (SCP) で使用して、発信者がクラスターを作成または更新するときに指定できるエグレスモードを制御できます。

条件キーは、次のアクションに適用されます。

- `eks:CreateCluster`
- `eks:UpdateClusterConfig`

例えば、次の SCP は、発信者が CUSTOMER_ROUTED を指定しない限り、クラスターの作成と設定の更新を拒否します。

```
{
```

```
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "RequireCustomerRoutedControlPlane",
      "Effect": "Deny",
      "Action": [
        "eks:CreateCluster",
        "eks:UpdateClusterConfig"
      ],
      "Resource": "*",
      "Condition": {
        "StringNotEquals": {
          "eks:controlPlaneEgressMode": "CUSTOMER_ROUTED"
        }
      }
    }
  ]
}
```

このポリシーを使用して、組織内のすべての新規および更新されたクラスターが CUSTOMER_ROUTED エグレスモードを使用するように適用します。

OIDC プロバイダー設定

クラスターが OIDC ID プロバイダーを使用している場合、コントロールプレーンは HTTPS (ポート 443) 経由で OIDC 検出エンドポイントに到達できる必要があります。これは、サービスアカウントの IAM ロール、またはクラスター認証に関連付ける OIDC ID プロバイダーに適用されます。OIDC 固有の設定はありません。前提条件で設定したのと同じ出力パスを使用します。これを許可するには:

1. コントロールプレーンサブネットに OIDC エンドポイントをカバーするルート (通常は NAT ゲートウェイなどのエグレスデバイスへのデフォルトルート) があることを確認します。
2. クラスターセキュリティグループがアウトバウンド TCP 443 を許可していることを確認します。
3. サブネット NACL がアウトバウンド TCP 443 およびインバウンドエフェメラルリターントラフィック (ポート 1024 ~ 65535) を許可していることを確認します。

エンドポイントはプロバイダーによって異なります。

- Amazon EKS OIDC プロバイダー (デフォルト): `oidc.eks.region-code.amazonaws.com`
- カスタム OIDC プロバイダー: 設定した発行者 URL。

OIDC 認証が失敗した場合のトラブルシューティング手順については、「the section called “OIDC プロバイダーにアクセスできない”」を参照してください。

接続を確認する

CUSTOMER_ROUTED エグレスを設定したら、コントロールプレーンが VPC リソースに到達できることを確認します。

1. 現在のエグレスモードをチェックする: クラスターが想定どおりのモードを使用していることを確認します。

```
aws eks describe-cluster --name my-cluster \
    --query "cluster.resourcesVpcConfig.controlPlaneEgressMode" \
    --region region-code
```

2. クラスターのステータスをチェックする: クラスターは ACTIVE 状態になっている必要があります。

```
aws eks describe-cluster --name my-cluster --query "cluster.status" --region region-code
```

3. ウェブフック接続をテストする: アドミッションウェブフックを設定している場合は、ウェブフックをトリガーするリソースを作成し、成功することを確認します。
4. ノード登録を確認する: ノードを起動し、クラスターを正常に結合することを確認します。

```
kubectl get nodes
```

5. OIDC をチェックする: サービスアカウント (IRSA) に IAM ロールを使用する場合は、ポッドが IAM ロールを引き受けることができることを確認します。

一般的な問題のトラブルシューティングについては、the section called “エグレスのトラブルシューティング” を参照してください。

GitHub でこのページを編集する

コントロールプレーンのエグレスに関する問題のトラブルシューティング

CUSTOMER_ROUTED コントロールプレーンエグレスモードを使用する場合、コントロールプレーン ENI からのネットワーク接続はユーザーの責任になります。このページでは、一般的な問題とその解決策について説明します。

失敗したウェブフックを検出する

コントロールプレーンがウェブフックサーバーまたは OIDC プロバイダーに到達できない場合、通常、症状はウェブフックタイムアウトとして表れます。確認するには、ウェブフックをトリガーするリソースを作成または変更し、エラーを確認します。

```
kubectl apply -f my-resource.yaml
```

接続または DNS 障害では通常、次のようなエラーを返します:

```
Error from server (InternalError): error when creating "my-resource.yaml": Internal
 error occurred:
failed calling webhook "my-webhook.example.com": failed to call webhook:
Post "https://my-webhook.example.com/validate?timeout=10s": context deadline exceeded
```

クラスター全体のウェブフックエラーの最近のイベントを確認することもできます:

```
kubectl get events --all-namespaces --field-selector reason=FailedCreate
```

- エラーがタイムアウト (`context deadline exceeded`) か、接続の拒否である場合、コントロールプレーンはウェブフックエンドポイントに到達できません。「the section called “必要なエンドポイントへのエグレスルートがない”」、「the section called “ウェブフックまたはコントロールプレーントラフィックをブロックする NACL”」、および「the section called “アクセスできないようにするセキュリティグループ”」を参照してください。
- エラーが DNS に関するものであるか、そのようなホスト障害ではない場合、コントロールプレーンはエンドポイントを解決できません。「the section called “DHCP オプションセットの更新の失敗”」を参照してください。

必要なエンドポイントへのエグレスルートがない

症状:

- アドミッションウェブフックがタイムアウトする。
- OIDC プロバイダーの検出が失敗する。
- クラスターの作成または更新が停止する。

原因:

コントロールプレーンのネットワークインターフェイスサブネットには、コントロールプレーンが到達する必要があるエンドポイントへの有効なルートがありません。通常、サブネットルートテーブルには、エグレスデバイスへのデフォルトルートがありません。または、そのデバイスが誤って設定されています。通常、エグレスデバイスは NAT ゲートウェイです。ただし、それを NAT インスタンス、ファイアウォールまたはプロキシアプライアンス、または一元化されたエグレス VPC へのトランジットゲートウェイにすることもできます。

解決策:

1. クラスターがコントロールプレーンネットワークインターフェイスに使用するサブネットを特定します:

```
aws eks describe-cluster --name my-cluster \
    --query "cluster.resourcesVpcConfig.subnetIds"
```

2. サブネットごとに、関連付けられたルートテーブルをチェックします:

```
aws ec2 describe-route-tables \
    --filters "Name=association.subnet-id,Values=subnet-ExampleID1"
```

3. エグレスデバイスを指す `0.0.0.0/0` のルート (またはエンドポイントをカバーするルート) が存在することを確認します。見つからない場合は、ルートを追加します。次の例では、NAT ゲートウェイルートを追加します。独自のエグレスターゲット (トランジットゲートウェイやネットワークインターフェイスなど) に置き換えてください:

```
aws ec2 create-route \
    --route-table-id rtb-ExampleID \
    --destination-cidr-block 0.0.0.0/0 \
    --nat-gateway-id nat-ExampleID
```

ウェブフックまたはコントロールプレーントラフィックをブロックする NACL

症状:

- アドミッションウェブフック呼び出しがタイムアウトする (エラー: `failed calling webhook`)。
- ミューテーティングウェブフックまたは検証ウェブフックを使用する Kubernetes リソースを作成または変更する際の断続的な障害。

原因:

コントロールプレーン ENI サブネット上のネットワーク ACL が、ウェブフックエンドポイントへのアウトバウンドトラフィックをブロックするか、インバウンドエフェメラルポートのリターントラフィックをブロックします。

解決策:

1. コントロールプレーンサブネットに関連付けられている NACL を特定します。

```
aws ec2 describe-network-acls \
    --filters "Name=association.subnet-id,Values=subnet-ExampleID1"
```

2. 次のルールが存在することを確認します:

Direction	プロトコル	ポート範囲	送信先/送信元	アクション
アウトバウンド	TCP	443	0.0.0.0/0 (またはウェブフック CIDR)	許可
アウトバウンド	TCP	10250	VPC CIDR	許可
インバウンド	TCP	1024–65535	0.0.0.0/0	許可 (エフェメラルリターントラフィック)

> **Note**
>
> NACL はステートレスです。インバウンドルールで、エフェメラルポート (1024 ~ 65535) でリターントラフィックを明示的に許可する必要があります。
> これらのルールは、2 つの異なるパスを対象としています。ポート 443 ルールは、ウェブフックと OIDC エンドポイントへのアウトバウンドトラフィック用であり、それはエグレスデバイスを介して VPC から離れます。ポート 10250 ルールは kubelet API 用であり、それはコントロールプレーンとノードの間の VPC 内に留まります。エグレスデバイスが存在しない場合、ポート 10250 には影響しませんが、制限的なネットワーク ACL によってブロックされる可能性があります。

アクセスできないようにするセキュリティグループ

症状:

- Webhook 呼び出しが失敗する。
- コントロールプレーンはノード (ポート 10250) の kubelet API に到達できない。
- `kubectl exec`、`kubectl logs`、または `kubectl port-forward` が失敗する。

原因:

コントロールプレーン ENI にアタッチされたセキュリティグループ (クラスターセキュリティグループ) は、必要なポートでのアウトバウンドトラフィックを許可しません。

解決策:

1. クラスターセキュリティグループを特定する :

```
aws eks describe-cluster --name my-cluster \
    --query "cluster.resourcesVpcConfig.clusterSecurityGroupId"
```

2. アウトバウンドルールで以下が許可されていることを確認します:

プロトコル	ポート	送信先
TCP	443	0.0.0.0/0 (ウェブフックエンドポイント、OIDC プロバイダー)
TCP	10250	ノードセキュリティグループまたは VPC CIDR (kubelet API)

3. アウトバウンドルールが制限されている場合は、必要なトラフィックのルールを追加します。

```
aws ec2 authorize-security-group-egress \
    --group-id sg-ExampleClusterSG \
    --protocol tcp \
    --port 443 \
    --cidr 0.0.0.0/0
```

> **Note**
>
> 厳格なエグレス要件があり、ウェブフックエンドポイントと OIDC エンドポイントの IP 範囲がわかっている場合は、ポート 443 ルールを `0.0.0.0/0` の代わりに、それらの特定の CIDR にスコープできます。ポート 10250 (kubelet API) ルールは VPC 内部のものであり、インターネットではなくノードセキュリティグループまたは VPC CIDR にスコープします。

DHCP オプションセットの更新の失敗

症状:

- コントロールプレーンから DNS 解決が失敗する。
- DNS ルックアップ (OIDC 検出、ウェブフック解決) を必要とするクラスターオペレーションが失敗する。
- 問題が、VPC DHCP オプションが変更されるか、コントロールプレーンが更新された後に発生する。

原因:

VPC DHCP オプションセットが変更されました。または、ドメインネームサーバーに `AmazonProvidedDNS` が含まれていません。また、コントロールプレーンが必要な名前を解決できる別のリゾルバーがない可能性もあります。コントロールプレーンは DHCP オプションセットの変更を自動的に検出し、通常は 1 時間以内に新しい DNS 設定を適用します。コントロールプレーンは、クラスター IAM ロールが、必要な Amazon EC2 読み取りアクセス許可を付与する場合にのみこれを行うことができます。

解決策:

1. VPC の DHCP オプションセットを確認します。

```
aws ec2 describe-vpcs --vpc-ids vpc-ExampleID \
    --query "Vpcs[0].DhcpOptionsId" \
    --region region-code
```

```
aws ec2 describe-dhcp-options --dhcp-options-ids dopt-ExampleID --region region-code
```

2. `domain-name-servers` に `AmazonProvidedDNS` (VPC IPv4 CIDR のベースに 2 つ加えた Amazon が提供する DNS リゾルバー)、またはコントロールプレーンに必要な名前を解決できる別のリゾルバーが含まれていることを確認します。
3. クラスターの IAM ロールが `ec2:DescribeVpcs` と `ec2:DescribeDhcpOptions` を付与していることを確認します。これらのアクセス許可がないと、コントロールプレーンは更新された DHCP オプションを読み取れず、DNS 設定を更新できません。詳細については、「Amazon EKS クラスターの IAM ロール」を参照してください。
4. DHCP オプションが変更されたら、コントロールプレーンが新しい設定を自動的に検出して適用するまで最大 1 時間かかります。クラスターの更新やインスタンスの置換は必要ありません。1 時間後も DNS 解決が失敗し、上記のアクセス許可が設定されている場合は、AWS サポートにお問い合わせください。

IPv6 ルーティングの問題

症状:

- IPv6 クラスターが外部 OIDC エンドポイントまたはウェブフックエンドポイントに到達できない。
- ノード登録が IPv4 では機能するが、IPv6 サービスは失敗する。

原因:

サブネットルートテーブルに Egress-Only インターネットゲートウェイへの `::/0` ルートがないか、セキュリティグループ/NACL が IPv6 トラフィックを許可していません。

解決策:

1. Egress-Only インターネットゲートウェイが存在し、VPC にアタッチされていることを確認します。

```
aws ec2 describe-egress-only-internet-gateways \
    --filters "Name=attachment.vpc-id,Values=vpc-ExampleID"
```

2. コントロールプレーンサブネットのルートテーブルに `::/0` ルートがあることを確認します。

```
aws ec2 describe-route-tables \
    --filters "Name=association.subnet-id,Values=subnet-ExampleID1" \
    --query "RouteTables[0].Routes[?DestinationIpv6CidrBlock=='::/0']"
```

3. 見つからない場合は、ルートを追加します。

```
aws ec2 create-route \
    --route-table-id rtb-ExampleID \
    --destination-ipv6-cidr-block ::/0 \
    --egress-only-internet-gateway-id eigw-ExampleID
```

4. NACL とセキュリティグループが、ポート 443 のアウトバウンドおよびインバウンドエフェメラルポートで IPv6 を許可していることを確認します。

OIDC プロバイダーにアクセスできない

症状:

- `IAM roles for service accounts` (IRSA) が失敗 – ポッドがロールを引き受けることができない。
- クラスターイベントに OIDC 検出エラーが表示される。

原因:

エグレスがブロックされているため、コントロールプレーンは OIDC プロバイダーエンドポイント (例: `oidc.eks.region-code.amazonaws.com`) に到達できません。

解決策:

1. エグレスパスとルートテーブルがアウトバウンド HTTPS トラフィックを許可していることを確認します。エグレスルートがないか、設定が間違っている場合のトラブルシューティング手順については、the section called “必要なエンドポイントへのエグレスルートがない” を参照してください。
2. クラスターセキュリティグループが `0.0.0.0/0` へのアウトバウンド TCP 443 を許可していることを確認します (the section called “アクセスできないようにするセキュリティグループ” を参照)。

GitHub でこのページを編集する

Karpenter と Cluster Autoscaler を使用したクラスターコンピューティングのスケーリング

自動スケーリングとは、需要の変化に合わせてリソースを自動的に増減させる機能です。これは Kubernetes の主要な機能であり、この機能がなければ、手動で実行するために膨大な人的リソースが必要になります。

EKS Auto Mode

Amazon EKS Auto Mode は、クラスターコンピューティングリソースを自動的にスケールします。ポッドが既存のノードに収まらない場合、EKS Auto Mode は新しいノードを作成します。また、EKS Auto Mode はワークロードの統合とノードの削除も行います。EKS Auto Mode は Karpenter に基づいて構築されます。

詳細については、以下を参照してください。

- EKS Auto Mode
- the section called “ノードプールの作成”
- the section called “拡張ワークロードのデプロイ”

その他のソリューション

Amazon EKS はさらに 2 つの自動スケーリング製品をサポートしています。

Karpenter

Karpenter は、アプリケーションの可用性とクラスターの効率の向上に役立つ柔軟で高性能な Kubernetes クラスターオートスケーラーです。Karpenter は、アプリケーションの負荷の変化に応じて、適切なサイズのコンピューティングリソース (Amazon EC2 インスタンスなど) を 1 分未満で起動します。Kubernetes と AWS との統合を通じて、Karpenter は、ワークロードの要件を正確に満たすジャストインタイムコンピューティングリソースをプロビジョンできます。Karpenter は、クラスターワークロードの具体的な要件に基づいて、新しいコンピューティングリソースを自動的にプロビジョンします。これには、コンピューティング、ストレージ、高速化、スケジューリングの各要件が含まれます。Amazon EKS は Karpenter を使用したクラスターをサポートしていますが、Karpenter は適合するいずれの Kubernetes クラスターでも動作します。詳細については、Karpenter のドキュメントを参照してください。

> **Important**
>
> Karpenter は、AWS ユーザーが Kubernetes クラスターでインストール、設定、管理を行うオープンソースソフトウェアです。AWS は、Amazon EKS クラスターで互換性のあるバージョンを使用して Karpenter を変更せずに実行した場合にテクニカルサポートを提供します。他のカスタマーマネージドソフトウェアと同様に、お客様が Karpenter コントローラーの可用性とセキュリティ、および、Karpenter コントローラーやその実行中の Kubernetes クラスターをアップグレードするときの適切なテスト手順を維持することが重要です。Karpenter の AWS サービスレベルアグリーメント (SLA) はなく、お客様は Karpenter によって起動された EC2 インスタンスがビジネス要件を満たしていることを確認する必要があります。

Cluster Autoscaler

Kubernetes の Cluster Autoscalerは、ポッドが失敗したり、他のノードに再スケジュールされた場合に、クラスター内のノード数を自動的に調整します。Cluster Autoscaler は Auto Scaling グループを使用します。詳細については、「AWS の Cluster Autoscaler」を参照してください。

Amazon EKS での Amazon Application Recovery Controller (ARC) のゾーンシフトの詳細

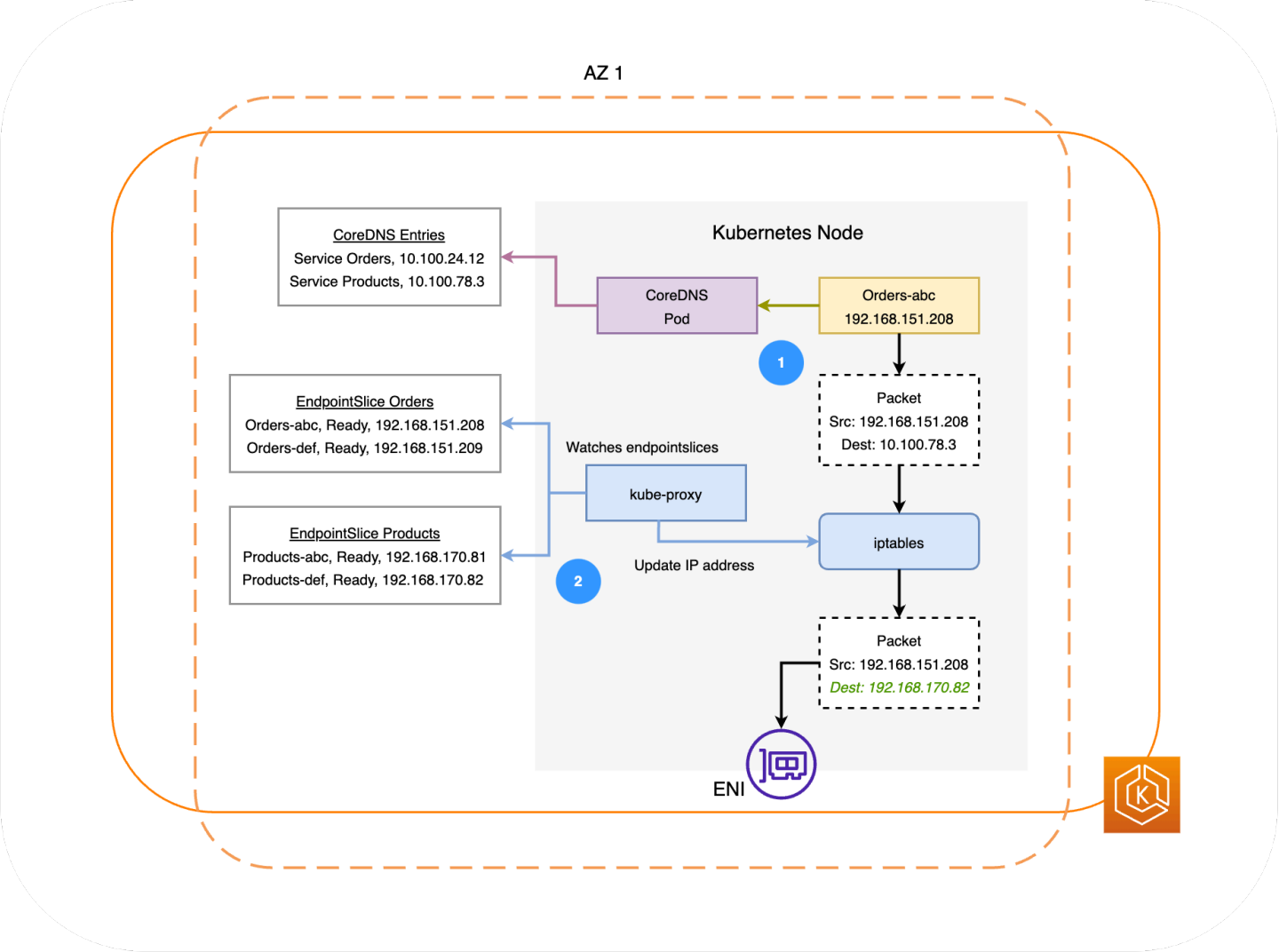
Kubernetes には、アベイラビリティーゾーン (AZ) のヘルスの低下や障害などのイベントに対するアプリケーションの回復力を高めるネイティブ機能があります。Amazon EKS クラスターでワークロードを実行する場合、Amazon Application Recovery Controller (ARC) のゾーンシフトまたはゾーンオートシフトを使用して、アプリケーション環境の耐障害性とアプリケーション復旧をさらに改善できます。ARC ゾーンシフトは、ゾーンシフトが期限切れになるまで、またはキャンセルするまで、障害のある AZ からリソースのトラフィックを移動できる一時的な手段として設計されています。必要に応じてゾーンシフトを拡張できます。

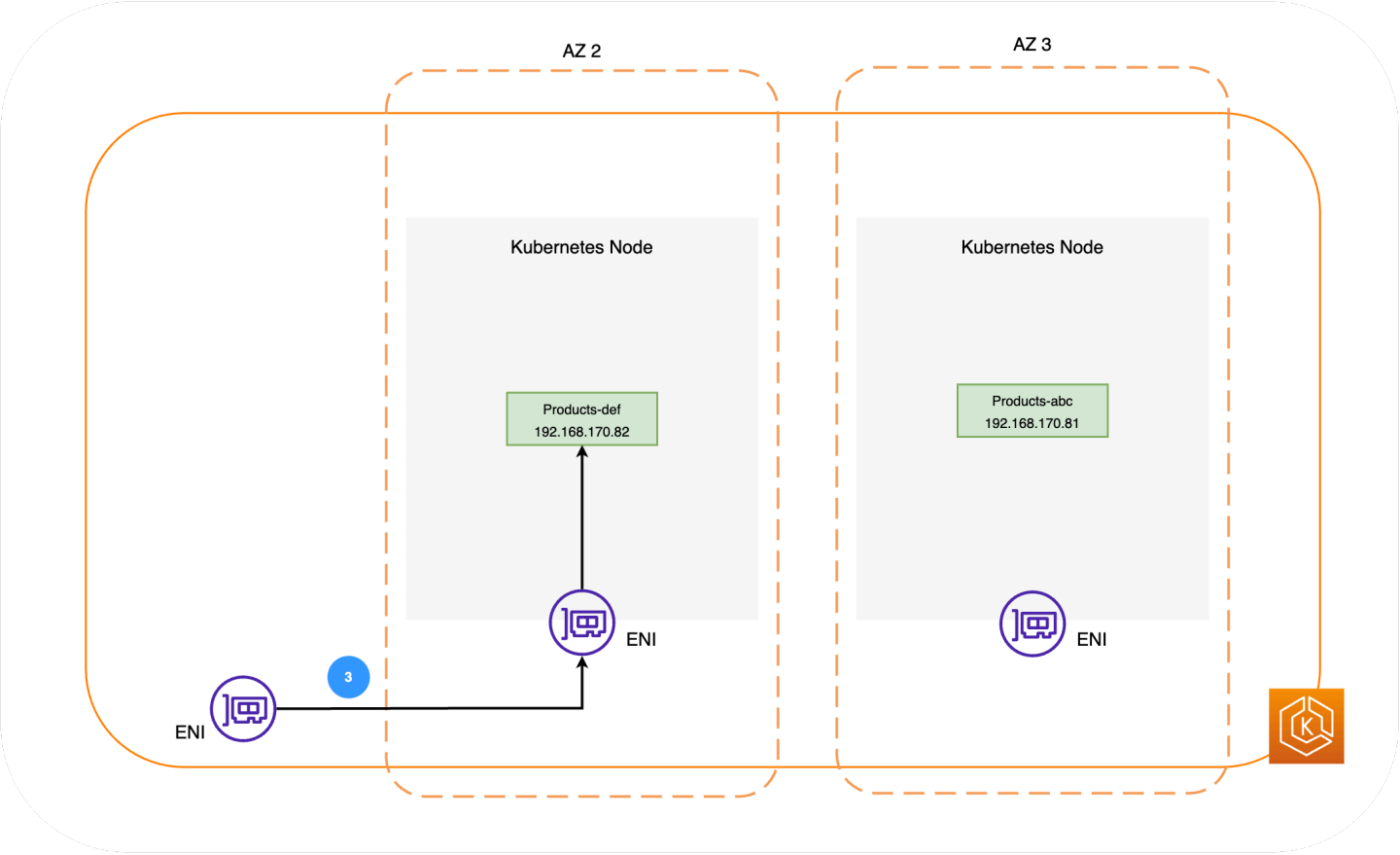
EKS クラスターのゾーンシフトを開始することも、ゾーンオートシフトを有効にして AWS がトラフィックをシフトできるようにすることもできます。このシフトは、正常な AZ のワーカーノードで実行されているポッドのネットワークエンドポイントのみを使用するようにクラスター内の東西ネットワークトラフィックのフローを更新します。さらに、EKS クラスター内のアプリケーションの入力トラフィックを処理する ALB または NLB は、トラフィックを正常な AZ 内のターゲットに自動的にルーティングします。可用性を最大限に高めることを目標にしているお客様にとって、AZ で障害が発生した場合は、障害のある AZ が回復するまでその AZ からすべてのトラフィックを移動できる

ことが重要です。そのためには、ARC ゾーンシフト で ALB または NLB を有効にすることもできます。

ポッド間の東西ネットワークトラフィックフローを理解する

次の図は、2 つのワークロードの例である Orders と Products を示しています。この例の目的は、さまざまな AZ でワークロードとポッドが通信する方法を示します。





1. Orders が Products と通信するには、まず Orders が送信先のサービスの DNS 名を解決する必要があります。Orders は CoreDNS と通信して、そのサービスの仮想 IP アドレス (クラスター IP) を取得します。Orders が Products のサービス名を解決すると、そのターゲット IP アドレスにトラフィックが送信されます。
2. kube-proxy はクラスター内のすべてのノードで実行され、サービス用の EndpointSlices を継続的に監視します。サービスが作成されると、EndpointSlice コントローラーによってバックグラウンドで EndpointSlice が作成され、管理されます。各 EndpointSlice には、ポッドアドレスのサブセットと、それらが実行されているノードを含むエンドポイントのリストまたは表があります。kube-proxy は、ノードで `iptables` を使用して、これらのポッドエンドポイントごとにルーティングルールを設定します。また、kube-proxy は、基本的なロードバランシングも行います。サービスのクラスターの IP アドレス宛てのトラフィックを、ポッドの IP アドレスに直接送信するようリダイレクトします。kube-proxy は、送信接続の送信先 IP アドレスを書き換えることでこれを行います。
3. 次に、前の図に示されているように、ネットワークパケットは、各ノードの ENI を使用して AZ 2 の Products ポッドに送信されます。

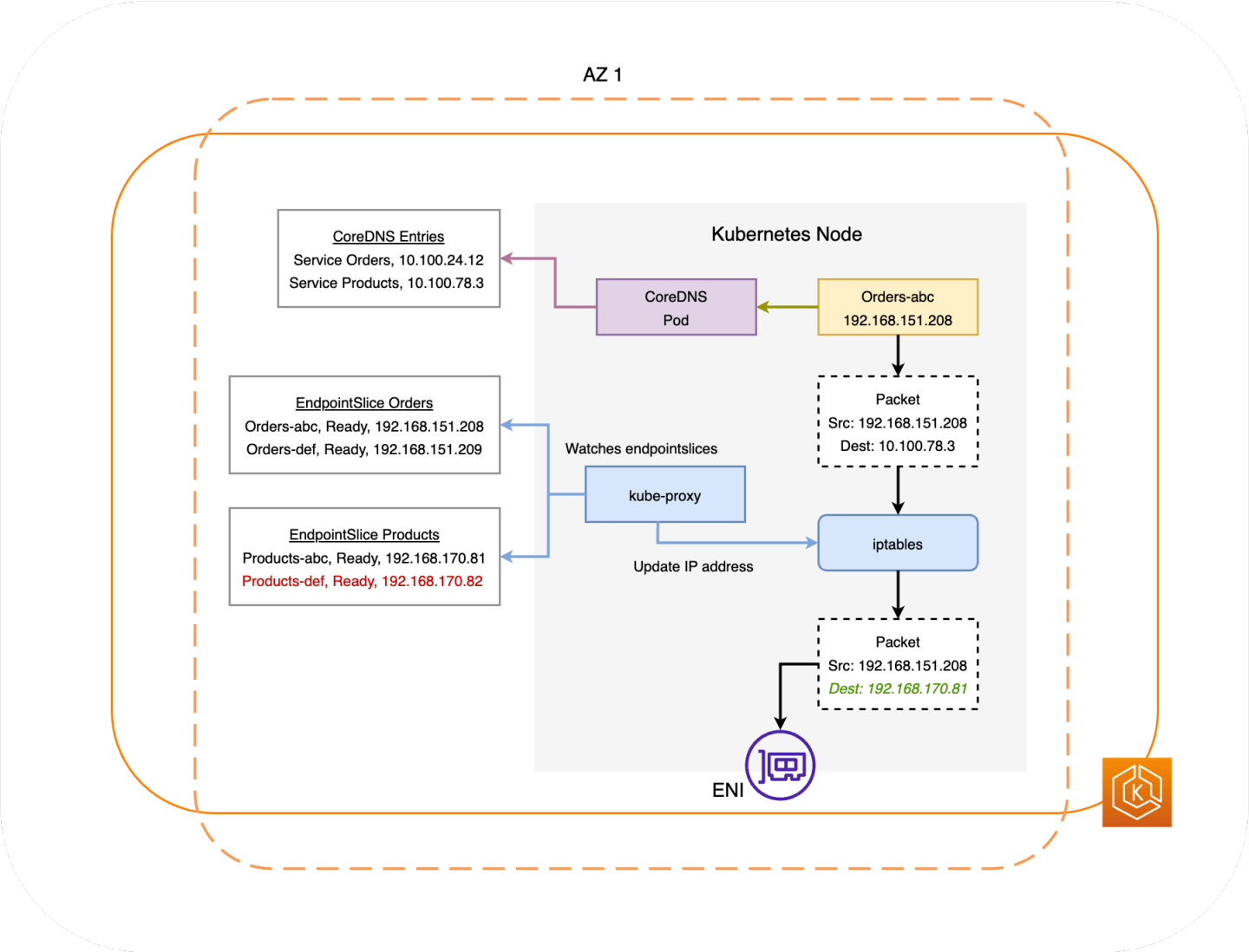
Amazon EKS での ARC ゾーンシフトについて

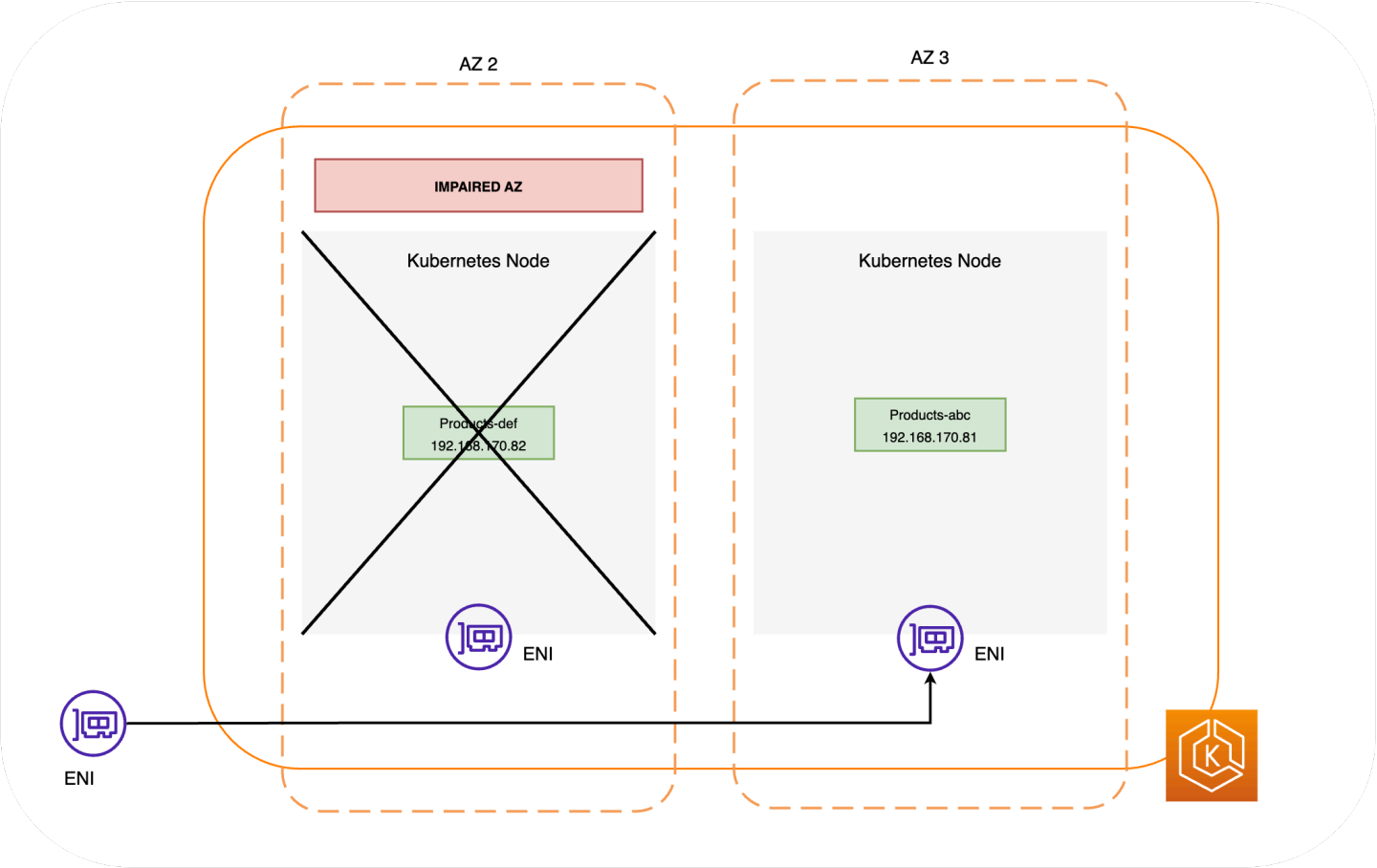
環境に AZ の障害がある場合は、EKS クラスター環境のゾーンシフトを開始できます。または、ゾーンオートシフトを使用して AWS がトラフィックのシフトを管理できます。ゾーンオートシフトでは、AWS は AZ の全体的なヘルスをモニタリングし、クラスター環境内の障害のある AZ からトラフィックを自動的に移動することで、潜在的な AZ 障害に対応します。

ARC で Amazon EKS クラスターのゾーンシフトを有効にすると、ARC コンソール、AWS CLI、またはゾーンシフト API やゾーンオートシフト API を使用して、ゾーンシフトをトリガーしたり、ゾーンオートシフトを有効にしたりできます。EKS ゾーンシフト中、次のことが自動的に行われます。

- 影響を受ける AZ 内のすべてのノードが遮断されます。これにより、Kubernetes スケジューラーが異常な AZ のノードに新しいポッドをスケジューリングできなくなります。
- EKS Auto Mode を使用している場合、EKS Auto Mode は障害のある AZ 内の新しいノードのプロビジョニングを自動的に停止します。EKS Auto Mode は、障害のある AZ に影響を与える統合やドリフトなどの自発的な中断アクションも一時停止し、障害のある AZ での自発的なインスタンスの終了を停止して、再試行ストームを防ぎます。障害のある AZ を対象とする厳格なスケジューリング要件 (永続的なボリュームバインド、ノードアフィニティ、厳格なトポロジー分散制約など) を持つポッドでは、その AZ への新しいノード起動は試行されません。
- マネージドノードグループを使用している場合、アベイラビリティーゾーンの再分散は停止され、Auto Scaling グループが更新され、新しい EKS データプレーンノードが正常な AZ でのみ起動されるようにします。
- 異常な AZ のノードは終了されず、ポッドはこれらのノードから削除されません。これは、ゾーンシフトの有効期限が切れたりキャンセルされたりしたときに、トラフィックがフル容量で AZ に安全に戻ることができるようにするためです。
- EndpointSlice コントローラーは、障害のある AZ 内のすべてのポッドエンドポイントを検索し、関連する EndpointSlices から削除します。これにより、正常な AZ のポッドエンドポイントのみがネットワークトラフィックの受信対象になります。ゾーンシフトがキャンセルまたは期限切れになると、EndpointSlice コントローラーは EndpointSlices を更新して、復元された AZ にエンドポイントを含めます。

以下の図は、EKS ゾーンシフトによって、クラスター環境で正常なポッドエンドポイントのみがターゲットにされる方法についての大まかな概要を示しています。


AZ 1
Kubernetes Node
CoreDNS Entries
Service Orders, 10.100.24.12
Service Products, 10.100.78.3
CoreDNS
Pod
Orders-abc
192.168.151.208
Packet
Src: 192.168.151.208
Dest: 10.100.78.3
EndpointSlice Orders
Orders-abc, Ready, 192.168.151.208
Orders-def, Ready, 192.168.151.209
Watches endpointslices
kube-proxy
EndpointSlice Products
Products-abc, Ready, 192.168.170.81
Products-def, Ready, 192.168.170.82
Update IP address
iptables
Packet
Src: 192.168.151.208
Dest: 192.168.170.81
ENI



EKS ゾーンシフトの要件

> **Important**
>
> ゾーンシフトは、クラスター内のすべてのトラフィックをアベイラビリティーゾーンからシフトします。ワークロードが十分なレプリカを持つ複数の AZ 全体に分散されていない場合、ゾーンシフトを開始すると、トラフィックを受信する正常なエンドポイントがないため、アプリケーションの可用性に影響する可能性があります。ゾーンオートシフトを有効にするか、手動ゾーンシフトを開始する前に、クラスター環境が 1 つ少ない AZ で動作することを検証する必要があります。

ゾーンシフトを EKS で正常に機能させるには、事前に AZ の障害に対する回復性を実現できるようにクラスター環境を設定する必要があります。以下は、回復性を確保するのに役立つ設定オプションのリストです。

- 複数の AZ にクラスターのワーカーノードをプロビジョニングする

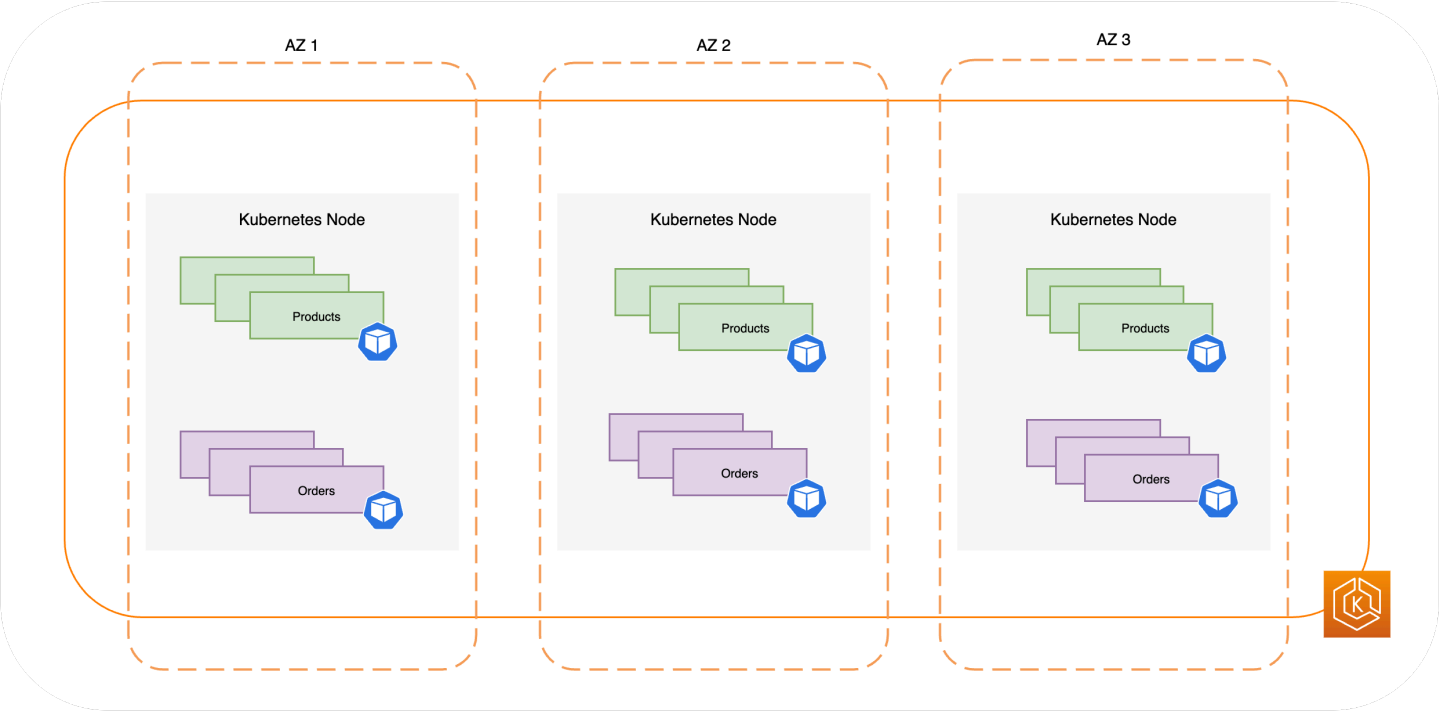
- 1 つの AZ を削除しても対応できる十分なコンピューティング容量をプロビジョニングする
- CoreDNS を含むポッドをすべての AZ でプリスケールする
- すべての AZ に複数のポッドレプリカを分散して、単一の AZ から移動した場合でも十分な容量を確保できるようにする
- 相互依存ポッドまたは関連ポッドを同じ AZ に配置する
- AZ からのゾーンシフトを手動で開始することで、1 つの AZ がない状態でもクラスター環境が期待どおりに動作することをテストします。または、ゾーンオートシフトを有効にし、オートシフトの練習運用に頼ることもできます。手動または練習のゾーンシフトによるテストは、EKS でゾーンシフトを機能させるために必須ではありませんが、強くお勧めします。

複数のアベイラビリティーゾーンに EKS ワーカーノードをプロビジョニングする

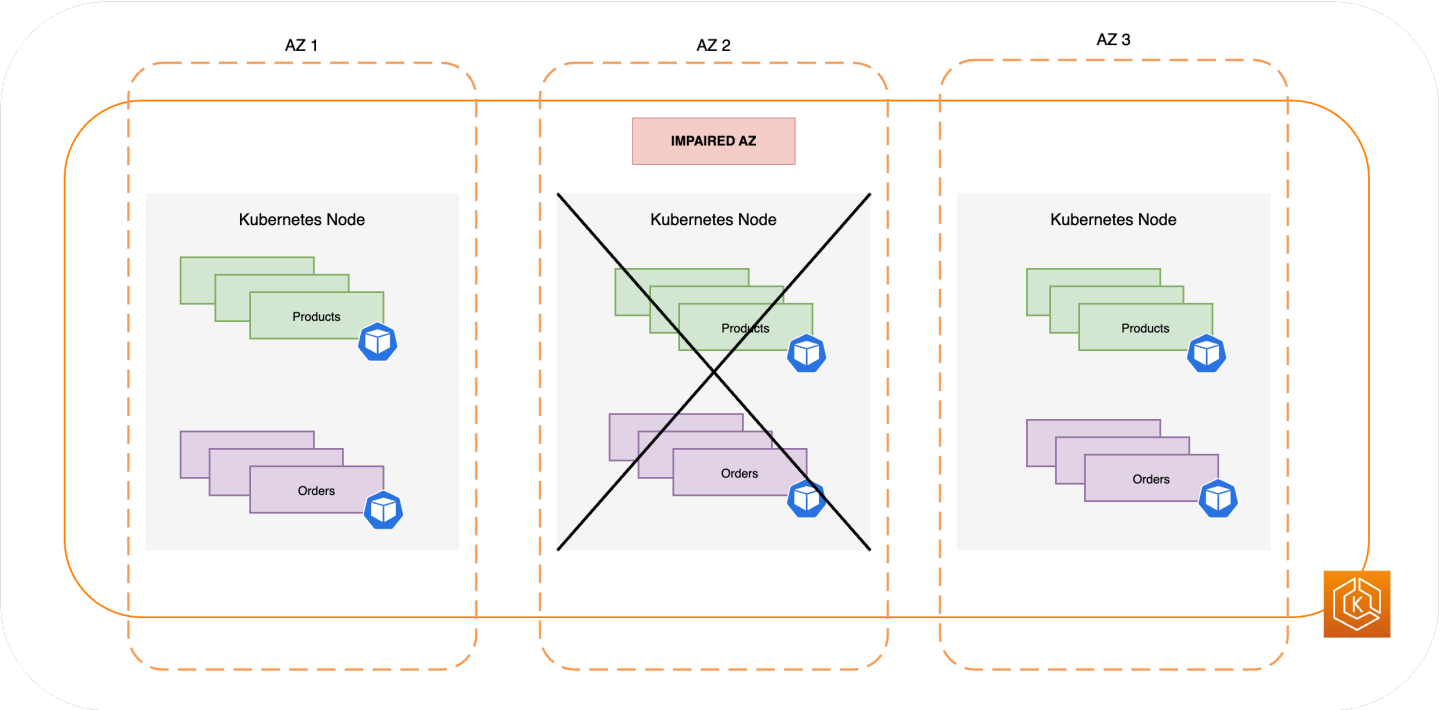
AWS リージョンには、アベイラビリティーゾーン (AZ) と呼ばれる物理データセンターがある複数の別々の場所があります。AZ は、リージョン全体に影響を与える可能性のある同時障害を回避するために、互いに物理的に離れた場所に配置されています。EKS クラスターをプロビジョニングする場合は、リージョン内の複数の AZ にワーカーノードをデプロイすることをお勧めします。これにより、クラスター環境は 1 つの AZ の障害に対する回復力が向上し、他の AZ で実行されるアプリケーションの高可用性を維持できます。影響を受ける AZ からゾーンシフトを開始すると、EKS 環境のクラスター内ネットワークは自動的に更新され、正常な AZ のみを使用するようになり、クラスターの高可用性を維持するのに役立ちます。

EKS 環境にマルチ AZ 設定を実施することで、システム全体の信頼性が向上します。ただし、マルチ AZ 環境は、アプリケーションデータの転送と処理方法に影響を与え、環境のネットワーク料金に影響します。特に、頻繁な egress クロスゾーントラフィック (AZ 間に分散されたトラフィック) は、ネットワーク関連のコストに大きな影響を与える可能性があります。さまざまな戦略を適用して、EKS クラスター内のポッド間のクロスゾーントラフィックの量を制御し、関連するコストを削減できます。高可用性 EKS 環境を実行するときにネットワークコストを最適化する方法の詳細については、このベストプラクティスガイドを参照してください。

次の図は、3 つの正常な AZ を持つ高可用性 EKS 環境を示しています。



次の図は、3 つの AZ を持つ EKS 環境が AZ の障害に対して回復力があり、他の 2 つの正常な AZ が残っているために高可用性を維持する方法を示しています。



1 つのアベイラビリティーゾーンの削除に耐えるのに十分なコンピューティング容量をプロビジョニングする

EKS データプレーンでコンピューティングインフラストラクチャのリソース使用率とコストを最適化するには、コンピューティング容量をワークロード要件に合わせることをお勧めします。ただし、すべてのワーカーノードがフル容量である場合、新しいポッドをスケジュールする前に、新しいワーカーノードを EKS データプレーンに追加する必要があります。重要なワークロードを実行する場合、通常、負荷の急増やノードのヘルスの問題などのシナリオに対処するために、冗長な容量をオンラインで運用することをお勧めします。ゾーンシフトを使用する予定がある場合は、障害が発生したときに AZ 全体の容量を削除することになります。そのため、いずれか 1 つの AZ がオフラインになっても負荷を十分に処理できるよう、冗長なコンピューティング容量を調整する必要があります。

コンピューティングリソースをスケールすると、EKS データプレーンに新しいノードを追加するプロセスには一定の時間がかかります。これは、特にゾーン障害が発生した場合に、アプリケーションのリアルタイムのパフォーマンスと可用性に影響を与える可能性があります。EKS 環境は、エンドユーザーやクライアントのエクスペリエンスが低下させることなく、1 つの AZ を失った場合の負荷を吸収できる必要があります。これは、新しいポッドが必要な時点からワーカーノードで実際にスケジュールされる時点までの遅延を最小限に抑えるか、排除することを意味します。

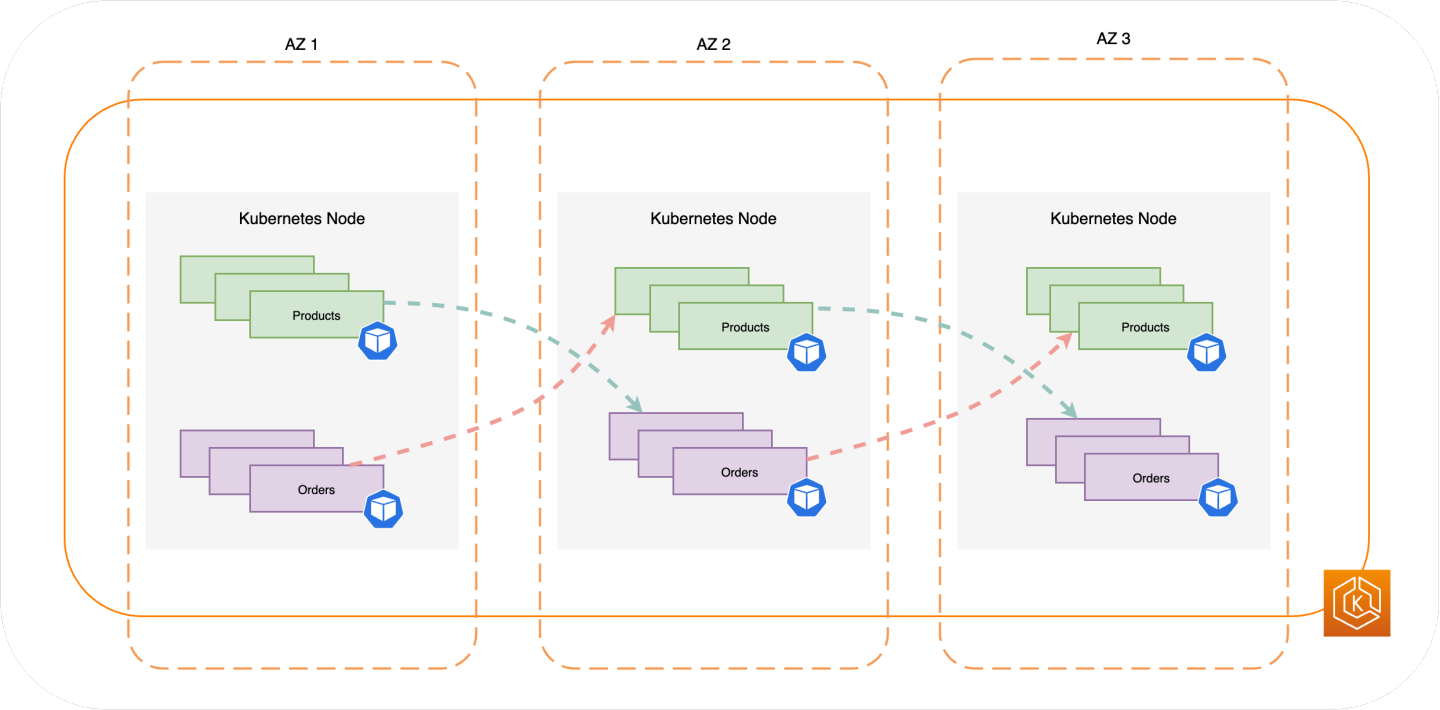
さらに、ゾーンの障害が発生した場合は、コンピューティング容量の制約に直面するリスクを軽減する必要があります。これにより、新しく必要なノードが正常な AZ の EKS データプレーンに追加されるのを防ぐことができます。

これらの潜在的な悪影響のリスクを軽減するために、各 AZ の一部のワーカーノードでコンピューティングキャパシティを過剰にプロビジョニングすることをお勧めします。これにより、Kubernetes スケジューラには、新しいポッド配置に使用できる既存の容量が確保されます。これは、環境内でいずれか 1 つの AZ を失った場合に特に重要です。

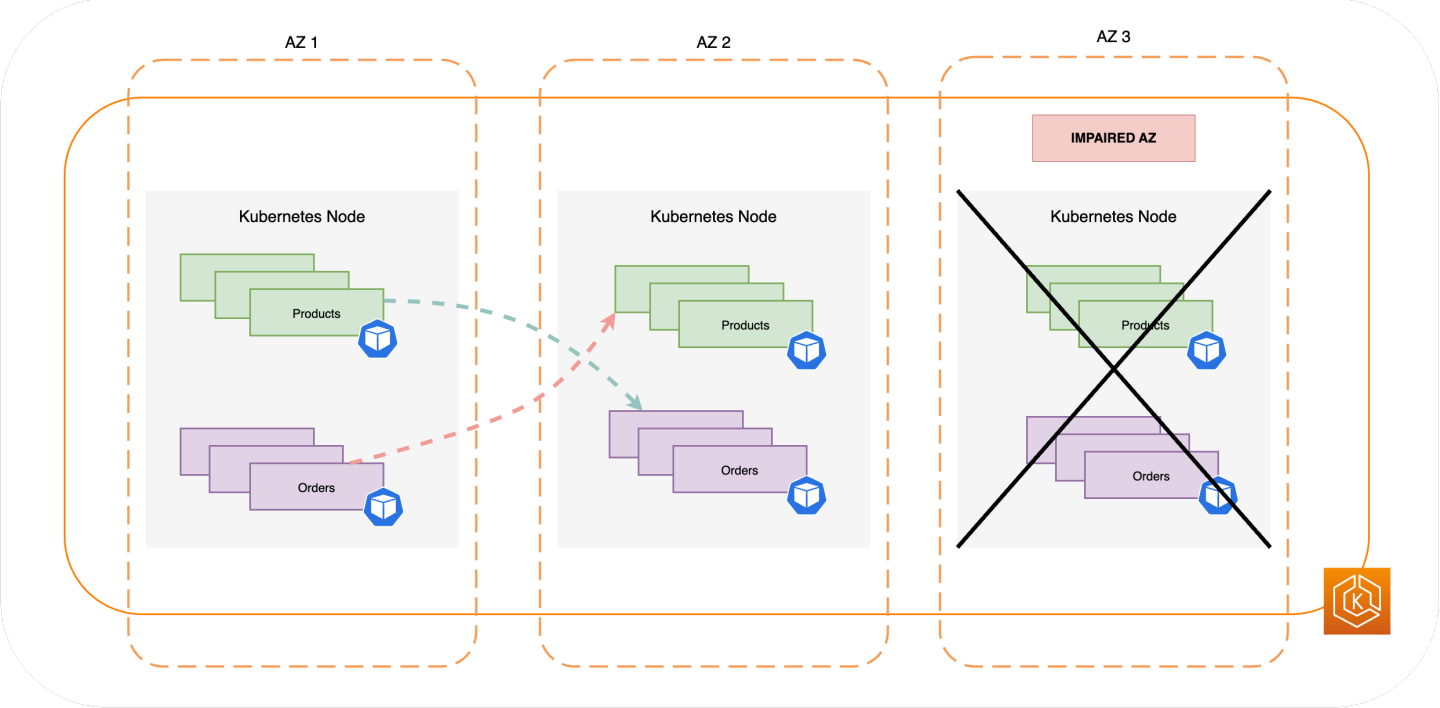
複数のポッドレプリカを実行してアベイラビリティーゾーンに分散する

Kubernetes を使用すると、1 つのアプリケーションの複数のインスタンス (ポッドレプリカ) を実行してワークロードを事前スケールできます。アプリケーションに対して複数のポッドレプリカを実行すると、単一障害点がなくなり、単一のレプリカにかかるリソース負荷が軽減されることで、全体的なパフォーマンスが向上します。ただし、アプリケーションの高可用性と耐障害性向上を実現するには、トポロジドメインとも呼ばれる異なる障害ドメインにまたがってアプリケーションの複数のレプリカを分散させて実行することをお勧めします。このシナリオの障害ドメインはアベイラビリティーゾーンです。トポロジの分散制約を使用することで、既存の静的安定性を持つようにアプリケーションをセットアップできます。その結果、AZ の障害が発生した場合でも、環境には正常な AZ に十分なレプリカがあり、トラフィックの急増やスパイクにすぐに対応できます。

下図は、すべての AZ が正常である場合の、東西トラフィックフローが発生している EKS 環境を示しています。



次の図は、1 つの AZ で障害が発生し、ゾーンシフトを開始した、東西トラフィックフローが発生している EKS 環境を示しています。



次のコードスニペットは、Kubernetes で複数のレプリカを使用してワークロードをセットアップする方法の例です。

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: orders
spec:
  replicas: 9
  selector:
    matchLabels:
      app: orders
  template:
    metadata:
      labels:
        app: orders
        tier: backend
    spec:
      topologySpreadConstraints:
      - maxSkew: 1
        topologyKey: "topology.kubernetes.io/zone"
        whenUnsatisfiable: ScheduleAnyway
        labelSelector:
          matchLabels:
            app: orders
```

最も重要なのは、DNS サーバーソフトウェア (CoreDNS/kube-dns) の複数のレプリカを実行し、それらがデフォルトで設定されていない場合は、同様のトポロジ分散制約を適用することです。これにより、単一の AZ 障害がある場合に、正常な AZ に十分な DNS ポッドがあり、クラスター内の他の通信ポッドのサービス検出リクエストを引き続き処理できます。CoreDNS EKS アドオンには、複数の AZ に使用可能なノードがある場合に、CoreDNS ポッドがクラスターのアベイラビリティーゾーンに分散されるようにするデフォルト設定があります。必要に応じて、これらのデフォルト設定を独自のカスタム設定に置き換えることもできます。

Helm で CoreDNS をインストールする場合、values.yaml ファイルで `replicaCount` を更新して、各 AZ に十分なレプリカを確保できます。さらに、これらのレプリカがクラスター環境内の異なる AZ に分散されるようにするには、同じ `values.yaml` ファイルで `topologySpreadConstraints` プロパティを更新してください。次のコードスニペットは、これを行うために CoreDNS を設定する方法を示しています。

CoreDNS Helm の values.yaml

```
replicaCount: 6
topologySpreadConstraints:
  - maxSkew: 1
    topologyKey: topology.kubernetes.io/zone
    whenUnsatisfiable: ScheduleAnyway
    labelSelector:
      matchLabels:
        k8s-app: kube-dns
```

AZ の障害が発生した場合は、CoreDNS の自動スケーリングシステムを使用して CoreDNS ポッドの負荷の増加を吸収できます。必要な DNS インスタンスの数は、クラスターで実行されているワークロードの数によって異なります。CoreDNS は CPU バインドであり、Horizontal Pod Autoscaler (HPA) を使用して CPU に基づいてスケールできます。以下は、ニーズに合わせて変更できる例です。

```
apiVersion: autoscaling/v1
kind: HorizontalPodAutoscaler
metadata:
  name: coredns
  namespace: default
spec:
  maxReplicas: 20
  minReplicas: 2
  scaleTargetRef:
    apiVersion: apps/v1
    kind: Deployment
    name: coredns
  targetCPUUtilizationPercentage: 50
```

また、EKS は、CoreDNS の EKS アドオンバージョンで CoreDNS デプロイの自動スケーリングを管理できます。この CoreDNS オートスケーラーは、ノード数や CPU コア数など、クラスターの状態を継続的にモニタリングします。この情報に基づいて、コントローラーは EKS クラスター内の CoreDNS デプロイのレプリカ数を動的に調整します。

CoreDNS EKS アドオンで自動スケーリング設定を有効にするには、次の構成設定を使用します。

```
{
  "autoScaling": {
    "enabled": true
  }
}
```

NodeLocal DNS またはクラスター比例オートスケーラーを使用して CoreDNS をスケールすることもできます。詳細については、「CoreDNS の水平スケーリング」を参照してください。

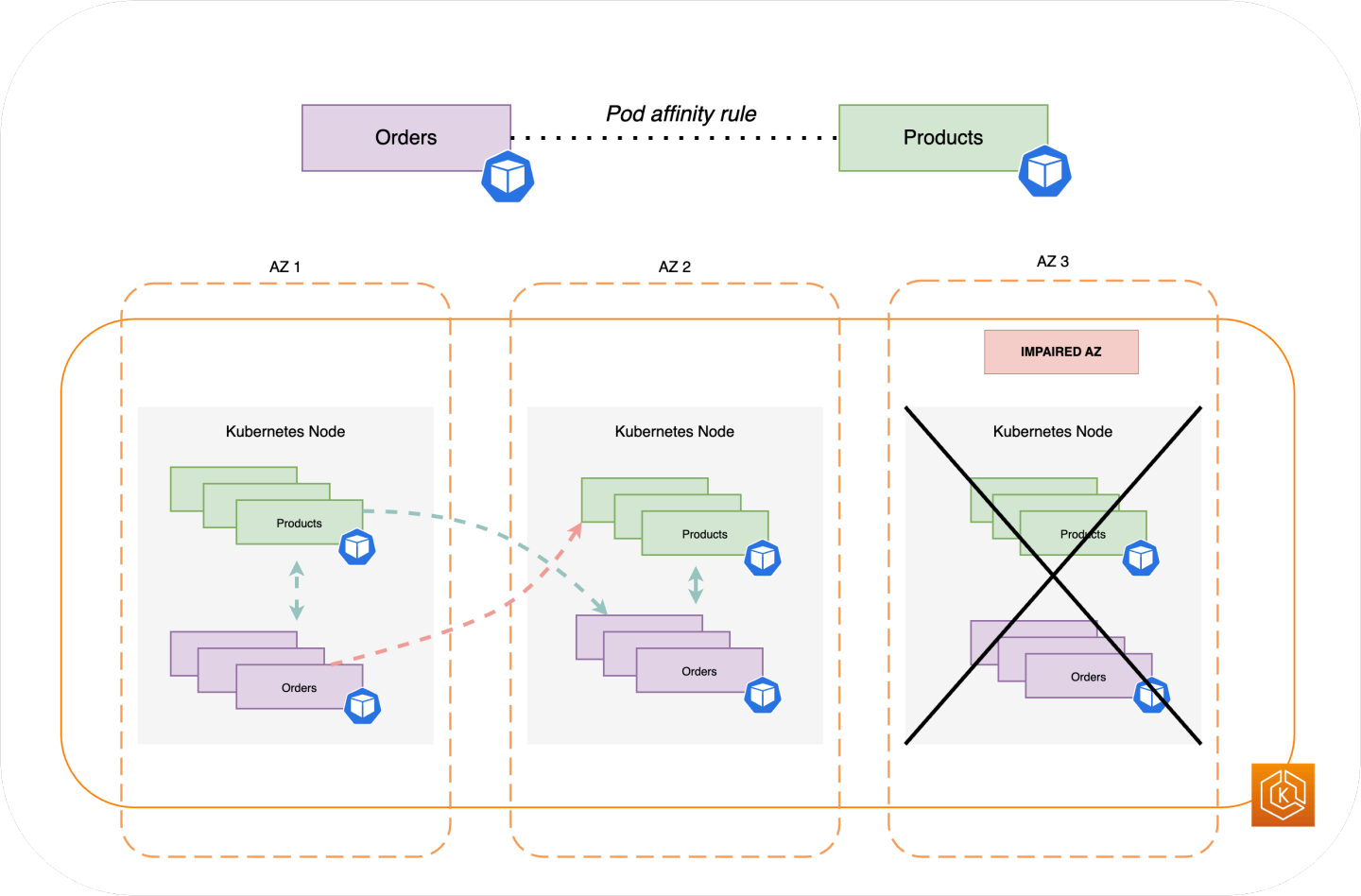
同じアベイラビリティーゾーンで相互依存ポッドを同じ場所に配置する

通常、アプリケーションには、エンドツーエンドのプロセスを正常に完了するために相互に通信する必要がある、個別のワークロードがあります。これらの異なるアプリケーションが異なる AZ に分散しており、同じ AZ に配置されていない場合、単一の AZ の障害がエンドツーエンドのプロセスに影響を与える可能性があります。例えば、アプリケーション A が AZ 1 と AZ 2 に複数のレプリカを持ち、アプリケーション B が AZ 3 にすべてのレプリカを持つ場合、AZ 3 が失われると、アプリケーション A とアプリケーション B の 2 つのワークロード間のエンドツーエンドのプロセスに影響が生じます。トポロジの分散制約とポッドのアフィニティを組み合わせると、すべての AZ にポッドを分散することで、アプリケーションの回復力を高めることができます。さらに、これにより、特定のポッドがコロケーションされるようにするためのポッド間の関係が設定されます。

ポッドのアフィニティのルールを使用すると、ワークロード間の関係を定義して Kubernetes スケジューラーの動作に影響を与え、同じワーカーノードまたは同じ AZ にポッドを配置できます。このスケジューリング制約をどの程度厳格にするかを設定することもできます。

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: products
  namespace: ecommerce
  labels:
    app.kubernetes.io/version: "0.1.6"
spec:
  template:
    spec:
      serviceAccountName: graphql-service-account
      affinity:
        podAffinity:
          requiredDuringSchedulingIgnoredDuringExecution:
          - labelSelector:
              matchExpressions:
              - key: app
                operator: In
                values:
                - orders
            topologyKey: "kubernetes.io/hostname"
```

次の図は、ポッドアフィニティルールを使用して同じノードにコロケーションされた複数のポッドを示しています。



クラスター環境が AZ の損失を処理できることをテストする

前のセクションで説明した要件を完了したら、次のステップは、AZ の損失を処理するのに十分なコンピューティングとワークロード容量があることをテストすることです。これを行うには、EKS でゾーンシフトを手動で開始します。または、ゾーンオートシフトを有効にして練習実行を設定して、クラスター環境内の AZ を 1 つ減らしてアプリケーションが期待どおりに機能することをテストすることもできます。

よくある質問

この機能を使用する理由は何ですか?

EKS クラスターで ARC ゾーンシフトまたはゾーンオートシフトを使用することで、クラスター内ネットワークトラフィックを障害のある AZ から移行する迅速な復旧プロセスを自動化すること

で、Kubernetes アプリケーションの可用性をより適切に維持できます。ARC を使用すると、障害のある AZ イベント中に回復期間を延長することがある、長く複雑なステップを回避できます。

この機能は他の AWS サービスとどのように連携しますか?

EKS は ARC と統合され、AWS で復旧オペレーションを実行するためのプライマリインターフェイスを提供します。クラスター内トラフィックが障害のある AZ から適切にルーティングされるように、EKS は Kubernetes データプレーンで実行されているポッドのネットワークエンドポイントのリストを変更します。クラスターへの外部トラフィックのルーティングに Elastic Load Balancing を使用している場合は、ロードバランサーを ARC に登録し、ゾーンシフトを開始して、トラフィックが機能低下した AZ に流れるのを防ぐことができます。EKS Auto Mode を使用している場合、EKS Auto Mode はノードのプロビジョニングを正常な AZ に自動的に制限します。ゾーンシフトは、EKS マネージド型ノードグループによって作成された Amazon EC2 Auto Scaling グループでも機能します。障害のある AZ が新しい Kubernetes ポッドまたはノードの起動に使用されるのを防ぐため、EKS は障害のある AZ を Auto Scaling グループから削除します。

この機能はデフォルトの Kubernetes 保護とどのように異なりますか?

この機能は、顧客のアプリケーションの回復力を高めるいくつかの Kubernetes 組み込み保護と連携して機能します。ポッドの準備状況とライブネスプローブを設定して、ポッドがいつトラフィックを取るかを決定できます。これらのプローブで障害が発生すると、Kubernetes はこれらのポッドをサービスのターゲットとして削除し、トラフィックはポッドに送信されなくなります。これは便利ですが、AZ が機能低下したときに障害が発生することが保証されるように、これらのヘルスチェックを設定するのは簡単ではありません。ARC ゾーンシフト機能には、Kubernetes のネイティブ保護が十分でない場合に、機能低下した AZ を完全に分離するのに役立つ追加のセーフティネットが用意されています。ゾーンシフトにはまた、アーキテクチャの運用準備状況と回復性を簡単にテストする方法も用意されています。

AWS はユーザーに代わってゾーンシフトを開始できますか?

はい。ARC ゾーンシフトを完全に自動化する方法が必要な場合は、ARC ゾーンオートシフトを有効にできます。ゾーンオートシフトでは、AWS を使用して EKS クラスターの AZ の状態を監視し、AZ の障害が検出されるとゾーンシフトを自動的に開始できます。

この機能を使用し、ワーカーノードとワークロードが事前にスケーリングされていない場合はどうなりますか?

ゾーンシフト中に事前スケールされておらず、追加のノードやポッドのプロビジョニングに依存している場合、復旧が遅れるリスクがあります。Kubernetes データプレーンに新しいノードを追加する

プロセスには時間がかかるため、特にゾーンの障害が発生した場合に、アプリケーションのリアルタイムのパフォーマンスと可用性に影響を与える可能性があります。さらに、ゾーンの障害が発生した場合は、コンピューティング容量の制約により、新たに必要なノードが正常な AZ に追加できなくなる可能性があります。

EKS Auto Mode を使用している場合、EKS Auto Mode はスケジュール不能なポッドの需要を満たすために、正常な AZ に新しいノードを自動的にプロビジョニングします。ただし、新しいノードが起動して準備完了になるにはまだ時間がかかります。最速の復旧を実現するには、複数の AZ 全体でワークロードを事前にスケーリングすることをお勧めします。

ワークロードが事前にスケールされておらず、クラスター内のすべての AZ に分散されていない場合、ゾーンの障害により、影響を受ける AZ のワーカーノードでのみ実行されているアプリケーションの可用性に影響する可能性があります。アプリケーションの可用性が完全に停止するリスクを軽減するため、EKS にはフェイルセーフ機能があります。この機能により、ワークロードのすべてのエンドポイントが障害が発生している AZ にある場合でも、障害ゾーンのポッドエンドポイントにトラフィックを送信することが可能です。ただし、ゾーンの問題が発生した場合でも可用性を維持するために、アプリケーションを事前にスケールしてすべての AZ に分散することを強くお勧めします。

ステートフルアプリケーションを実行している場合、これはどのように機能しますか?

ステートフルアプリケーションを実行している場合は、ユースケースとアーキテクチャに基づいて、耐障害性を評価する必要があります。アクティブ/スタンバイのアーキテクチャまたはパターンがある場合、アクティブが障害のある AZ にある可能性があります。アプリケーションレベルでは、スタンバイがアクティブ化されていない場合、アプリケーションに問題が発生する可能性があります。また、正常な AZ で新しい Kubernetes ポッドを起動すると、障害のある AZ にバインドされた永続的なボリュームにアタッチできないため、問題が発生する可能性があります。

この機能は EKS Auto Mode で機能しますか?

はい。EKS Auto Mode クラスターでゾーンシフトを有効にすると、EKS Auto Mode はゾーンシフトイベントに自動的に応答します。ゾーンシフト中、EKS Auto Mode は障害のある AZ 内の新しいノードのプロビジョニングを停止し、障害のある AZ に影響を与える任意の中断アクション (統合やドリフトなど) を一時停止し、障害のある AZ を対象とする厳格なスケジューリング要件を持つポッドのキャパシティの起動を回避します。クラスターでゾーンシフトを有効にする以外に追加の設定は必要ありません。

この機能はセルフマネージド Karpenter で機能しますか?

セルフマネージド Karpenter のサポートは、Karpenter バージョン 1.12 以降の EKS の ARC ゾーンシフトとゾーンオートシフトで利用できます。

この機能は EKS Fargate で機能しますか?

この機能は EKS Fargate では機能しません。デフォルトでは、EKS Fargate がゾーンヘルスイベントを認識すると、ポッドは他の AZ での動作を優先します。

EKS マネージド Kubernetes コントロールプレーンは影響を受けますか?

いいえ。デフォルトでは、Amazon EKS は、複数の AZ で Kubernetes コントロールプレーンを実行およびスケールして、高可用性を実現します。ARC ゾーンシフトとゾーンオートシフトは、Kubernetes データプレーンでのみ動作します。

この新機能に関連するコストはありますか?

EKS クラスターでは、ARC ゾーンシフトとゾーンオートシフトを追加料金なしで使用できます。ただし、プロビジョニングされたインスタンスの料金は引き続きお支払いいただきます。この機能を使用する前に、Kubernetes データプレーンを事前にスケールすることを強くお勧めします。コストとアプリケーションの可用性のバランスを考慮する必要があります。

その他のリソース

- ゾーンシフトの仕組み
- ARC のゾーンシフトのベストプラクティス
- ゾーンシフトおよびゾーンオートシフトでサポートされているリソースとシナリオ
- Amazon EKS での回復力のあるワークロードの運用
- ポッドの優先度とオーバープロビジョニングにより Kubernetes ノードスケーリングの遅延を排除する
- CoreDNS ポッドを高い DNS トラフィックにスケールする

EKS ゾーンシフトを有効にしてアベイラビリティーゾーンの障害を回避する

Amazon Application Recovery Controller (ARC) は、アベイラビリティーゾーン (AZ) 間でアプリケーションのリカバリを管理および調整し、Amazon EKS を含む多くのサービスと連携動作します。ARC ゾーンシフトの EKS サポートでは、クラスター内ネットワークトラフィックを障害のある AZ から移行できます。また、AWS に AZ の健全性を監視させ、異常な AZ から別の場所へ一時的にネットワークトラフィックを切り替える権限を付与することもできます。

EKS ゾーンシフトの使用方法:

1. Amazon Application Recovery Controller (ARC) で EKS クラスターを有効にします。これは、Amazon EKS コンソール、AWS CLI、CloudFormation、または eksctl を使用してクラスターレベルで行われます。
2. 有効にすると、ARC コンソール、AWS CLI、またはゾーンシフト API およびゾーンオートシフト API を使用して、ゾーンシフトまたはゾーンオートシフトを管理できます。

EKS クラスターを ARC に登録した後も、ARC を設定する必要があります。例えば、ARC コンソールを使用してゾーンオートシフトを設定できます。

EKS ゾーンシフトの仕組みと、障害のあるアベイラビリティーゾーンを処理するためのワークロードの設計方法についての詳細は、「the section called “ゾーンシフトの詳細”」を参照してください。

考慮事項

- 各リクエストが適切に処理されるため、ゾーンシフト操作の間に少なくとも 60 秒待つことをお勧めします。

  ゾーンシフトを迅速に連続的に (互いに 60 秒以内の間隔で) 実行しようとすると、Amazon EKS サービスがすべてのシフトリクエストを適切に処理しない可能性があります。これはクラスターのゾーン状態を更新する現在のポーリングメカニズムが原因です。複数のゾーンシフトを実行する必要がある場合、システムがすべての変更を処理するため、操作間に十分な時間間隔を確保してください。

Amazon Application Recovery Controller とは

Amazon Application Recovery Controller (ARC) は、AWS 上で実行しているアプリケーションのリカバリ操作の準備や迅速な実行に役立ちます。ゾーンシフトを使用すると、サポートされているリソースのトラフィックを アベイラビリティーゾーン (AZ) から AWS リージョン内の正常な AZ に一時的に移動することで、AZ の障害からすばやく回復できます。

Amazon Application Recovery Controller (ARC) の詳細はこちら

ゾーンシフトとは

ゾーンシフトは ARC の機能であり、EKS クラスターや Elastic Load Balancer などのリソースのトラフィックを AWS リージョンのアベイラビリティーゾーンから移動できます。これにより、問題を

すばやく軽減し、アプリケーションをすばやく回復できます。例えば、不適切なデプロイが原因でレイテンシーの問題が発生していたり、アベイラビリティーゾーンで障害が発生していたりする場合、トラフィックをシフトすることを選択できます。ゾーンシフトでは、事前設定手順は必要ありません。ただし、シフトを開始する前に、ワークロードがゾーン的に回復力を持つには、特定の要件を満たしている必要があります。詳細については、「the section called “ゾーンシフトの詳細”」を参照してください。

ARC ゾーンシフトの詳細はこちら

ゾーンオートシフトとは

ゾーンオートシフトは ARC の機能であり、有効にすると AWS がユーザーに代わってサポートされているリソースのトラフィックを AZ から AWS リージョンの正常な AZ に移動できます。AWS は、顧客に影響を与える可能性のある、リージョン内の AZ の障害が内部テレメトリによって示されると、オートシフトを開始します。内部テレメトリには、AWS ネットワーク、Amazon EC2、Elastic Load Balancing サービスなど、複数のソースからのメトリクスが組み込まれています。

AWS は、インジケータが問題や潜在的な問題がなくなったことを示すと、オートシフトを終了します。

ARC ゾーンオートシフトの詳細はこちら

ゾーンシフト中に EKS は何をしますか?

EKS は、障害のある AZ へのトラフィックを停止するようにネットワーク設定を更新します。特定のコンピューティング動作は、クラスターのデータプレーン設定によって異なります。

- EKS Auto Mode: EKS Auto Mode は、障害のある AZ 内の新しいノードのプロビジョニングを自動的に停止し、障害のある AZ に影響を与える自発的なノードの中断 (統合など) を停止します。また、障害のある AZ を対象とする厳格なスケジューリング要件 (永続的なボリュームバインドや厳格なトポロジー分散制約など) を持つポッドの容量の起動を回避します。クラスターでゾーンシフトを有効にする以外に追加の設定は必要ありません。
- セルフマネージド Karpenter (v1.12+): `--enable-zonal-shift` フラグ付きで Karpenter v1.12 以降を実行している場合、Karpenter は障害のある AZ でキャパシティのプロビジョニングを停止し、障害のある AZ に依存する自発的な中断を停止します。
- マネージドノードグループ: EKS は、アベイラビリティーゾーンの再調整を停止し、Auto Scaling グループを更新することで、ゾーンシフト中に正常な AZ でのみ新しいノードを起動します。

シフトの有効期限が切れたりキャンセルされたりすると、以前に異常として検出された AZ を含めるようにネットワーク設定とコンピューティング動作が復元されます。

EKS ゾーンシフトの詳細はこちら

EKS クラスターを Amazon Application Recovery Controller (ARC) に登録する (AWS コンソール)

1. ARC に登録する EKS クラスターの名前とリージョンを見つけます。
2. そのリージョンの EKS コンソールに移動し、クラスターを選択します。
3. [クラスター情報] ページの [概要] タブを選択します。
4. ゾーンシフトの見出しで、[管理] ボタンを選択します。
5. [EKS ゾーンシフト] に [有効] または [無効] を選択します。

これで、EKS クラスターが ARC に登録されました。

AWS にアベイラビリティーゾーンの障害を検出して回避させるには、ARC ゾーンオートシフトを設定する必要があります。例えば、ARC コンソールで以下のことを行うことができます。

次のステップ

- ゾーンオートシフトを有効にする方法について説明します。
- 手動でゾーンシフトを開始する方法について説明します。

Amazon EKS でのアクセスコントロールの仕組みについて説明します。

Amazon EKS クラスターへのアクセスを管理する方法について説明します。Amazon EKS を使用するには、Kubernetes と AWS Identity and Access Management (AWS IAM) がどのようにアクセスコントロールをするのかについての知識が必要になります。

このセクションには以下が含まれます。

the section called “Kubernetes API アクセス” — アプリケーションまたはユーザーが Kubernetes API を認証できるようにする方法について説明します。アクセスエントリ、aws-auth ConfigMap、または外部の OIDC プロバイダーを使用できます。

the section called “クラスターリソースにアクセスする” — Amazon EKS クラスターと通信するように AWS マネジメントコンソールを設定する方法について説明します。コンソールを使用して、名前空間、ノード、Pod など、クラスター内の Kubernetes リソースを表示します。

the section called “クラスターリソースを変更する” – Kubernetes リソースの変更に必要なアクセス許可について説明します。

the section called “kubectl を使用してクラスターにアクセスする” — Amazon EKS クラスターと通信するように kubectl を設定する方法について説明します。AWS CLI を使用して kubeconfig ファイルを作成します。

the section called “AWS へのワークロードアクセス ” —Kubernetes サービスアカウントを AWS IAM ロールに関連付ける方法について説明します。サービスアカウント (IRSA) には、Pod Identity または IAM ロールを使用できます。

一般的なタスク

- 開発者に Kubernetes API へのアクセスを許可します。AWS マネジメントコンソールに Kubernetes リソースを表示します。
  - 解決策: アクセスエントリを使用して、Kubernetes RBAC アクセス許可を AWS IAM ユーザーまたはロールに関連付けます。
- AWS 認証情報を使用して、kubectl を Amazon EKS クラスターと通信するように設定します。
  - 解決策: AWS CLI を使用して kubeconfig ファイルを作成 します。

- Ping Identity など外部 ID プロバイダーを使用して、Kubernetes API に対してユーザーを認証します。
  - 解決策: 外部の OIDC プロバイダーをリンクします。
- Kubernetes クラスターのワークロードに AWS API を呼び出す機能を付与します。
  - 解決策: Pod Identity を使用して AWS IAM ロールを Kubernetes サービスアカウントに関連付けます。

背景

- Kubernetes サービスアカウントの仕組みを学びます。
- Kubernetes ロールベースアクセスコントロール (RBAC) モデルを確認します。
- AWS リソースへのアクセスを管理する方法については、「AWS IAM ユーザーガイド」を参照してください。あるいは、無料の AWS IAM の使用に関する入門トレーニング も受講できます。

EKS Auto Mode に関する考慮事項

EKS Auto Mode は、EKS Pod Identity および EKS アクセスエントリと統合されます。

- EKS Auto Mode は、アクセスエントリを使用して EKS コントロールプレーンに Kubernetes アクセス許可を付与します。例えば、アクセスポリシーにより、EKS Auto Mode はネットワークエンドポイントとサービスに関する情報を読み取ることができます。
  - EKS Auto Mode クラスターのアクセスエントリを無効にすることはできません。
  - 任意で `aws-auth ConfigMap` を有効にできます。
  - EKS Auto Mode のアクセスエントリは自動的に設定されます。このアクセスエントリは表示できますが、変更することはできません。
  - NodeClass を使用してカスタムノード IAM ロールを作成する場合は、AmazonEKSAutoNodePolicy アクセスポリシーを使用してロールのアクセスエントリを作成する必要があります。
- AWS サービスのアクセス許可をワークロードに付与する場合は、EKS Pod Identity を使用します。
  - EKS Auto Mode クラスターに Pod Identity エージェントをインストールする必要はありません。

IAM ユーザーおよびロールに Kubernetes API へのアクセスを付与する

クラスターには Kubernetes API エンドポイントがあります。Kubectl はこの API を使用します。この API は、次の 2 種類の ID を使用して認証することができます。

- AWS Identity and Access Management (IAM) プリンシパル (ロールまたはユーザー) — このタイプには IAM への認証が必要です。IAM ユーザーとして、または ID ソースから提供された認証情報を使用して、フェデレーテッド ID で AWS にサインインできます。IAM ロールを使用して前もって管理者により ID フェデレーションが設定されている場合、フェデレーティッド ID としてのみサインインできます。フェデレーションを使用して AWS にアクセスする場合、間接的にロールを引き受けることになります。ユーザーがこのタイプの ID を使用すると、次のことが行われます。
  - ユーザーがクラスター上の Kubernetes オブジェクトを操作できるように Kubernetes アクセス許可を割り当てることができます。IAM プリンシパルにアクセス許可を割り当てて、クラスター上の Kubernetes オブジェクトにアクセスできるようにする方法の詳細については、「the section called “アクセスエントリ”」を参照してください。
  - IAM アクセス許可を割り当てて、ユーザーが Amazon EKS API、AWS CLI、AWS CloudFormation、AWS マネジメントコンソール、または `eksctl` を使用して Amazon EKS クラスターとそのリソースを操作できるようにすることができます。詳細については、サービス認可リファレンスの「Amazon Elastic Kubernetes Service で定義されるアクション」を参照してください。
  - ノードは IAM ロールを引き受けることでクラスターに参加します。IAM プリンシパルを使用してクラスターにアクセスできます。プリンシパルは、Amazon EKS コントロールプレーンで実行される AWS IAM Authenticator for Kubernetes によって提供されます。
- 独自の OpenID Connect (OIDC) プロバイダーのユーザー — このタイプでは、OIDC プロバイダーへの認証が必要です。Amazon EKS クラスターで独自の OIDC プロバイダーを設定する方法については、「the section called “OIDC プロバイダーをリンクする”」を参照してください。ユーザーがこのタイプの ID を使用すると、次のことが行われます。
  - ユーザーがクラスター上の Kubernetes オブジェクトを操作できるように Kubernetes アクセス許可を割り当てることができます。
  - IAM アクセス許可を割り当てて、ユーザーが Amazon EKS API、AWS CLI、AWS CloudFormation、AWS マネジメントコンソール、または `eksctl` を使用して Amazon EKS クラスターとそのリソースを操作できるようにすることはできません。

クラスターでは両方のタイプの ID を使用できます。IAM の認証方法を無効にすることはできません。OIDC 認証は任意です。

IAM アイデンティティと Kubernetes のアクセス許可を関連付ける

AWS IAM Authenticator for Kubernetes はクラスターのコントロールプレーンにインストールされます。これにより、クラスター上の Kubernetes リソースへのアクセスを許可する AWS Identity and Access Management (IAM) プリンシパル (ロールとユーザー) が有効になります。次のいずれかの方法を使用して、IAM プリンシパルにクラスター上の Kubernetes オブジェクトへのアクセスを許可できます。

- アクセスエントリの作成 — クラスターが、クラスターの Kubernetes バージョンの「前提条件」セクションに記載されているプラットフォームバージョンと同じかそれ以降の場合は、このオプションを使用することをお勧めします。

  アクセスエントリを使用して、クラスター外から IAM プリンシパルの Kubernetes アクセス許可を管理します。クラスターへのアクセスの追加および管理には、EKS API、AWS コマンドラインインターフェイス、AWS SDK、AWS CloudFormation、および AWS マネジメントコンソール を使用できます。つまり、クラスターを作成したのと同じツールでユーザーを管理できるということです。

  開始するには、「アクセスエントリを使用するように認証モードを変更する」に従い、次に「既存の aws-auth ConfigMap エントリをアクセスエントリに移行する」に従います。
- **aws-auth ConfigMap** へのエントリの追加 — クラスターのプラットフォームバージョンが「前提条件」セクションに記載されているバージョンよりも前の場合は、このオプションを使用する必要があります。クラスターのプラットフォームバージョンが、クラスターの Kubernetes バージョンの「前提条件」セクションに記載されているプラットフォームバージョンと同じかそれ以降で、ConfigMap にエントリを追加した場合は、それらのエントリをアクセスエントリに移行することをお勧めします。ただし、マネージド型ノードグループで使用される IAM ロールのエントリや Fargate プロファイルなど、Amazon EKS が ConfigMap に追加したエントリは移行できません。詳細については、「the section called “Kubernetes API アクセス”」を参照してください。
  - aws-auth ConfigMap オプションを使用する必要がある場合は、eksctl create iamidentitymapping コマンドを使用してエントリを ConfigMap に追加できます。詳細については、eksctlドキュメントの「IAM ユーザーとロールの管理」を参照してください。

クラスター認証モードを設定する

各クラスターには認証モードがあります。認証モードによって、IAM プリンシパルがクラスター上の Kubernetes オブジェクトにアクセスできるようにするために使用できる方法が決まります。認証モードは 3 つあります。

> **Important**
>
> アクセス入力メソッドを一度有効にすると、無効にすることはできません。ConfigMap メソッドがクラスターの作成時に有効になっていない場合、後で有効にすることはできません。アクセスエントリの導入前に作成されたすべてのクラスターでは、ConfigMap メソッドが有効になっています。
> クラスターでハイブリッドノードを使用している場合は、APIまたは API_AND_CONFIG_MAP クラスター認証モードを使用する必要があります。

クラスター内部の **aws-auth ConfigMap**

これは Amazon EKS クラスターのオリジナルの認証モードです。クラスターを作成した IAM プリンシパルは、kubectl を使用してクラスターにアクセスできる初期ユーザーです。初期ユーザーは、aws-auth ConfigMap のリストに他のユーザーを追加し、クラスター内の他のユーザーに影響を与えるアクセス許可を割り当てる必要があります。ConfigMap には管理するエントリがないため、これらの他のユーザーは初期ユーザーを管理したり削除したりできません。

ConfigMap とアクセスエントリの両方

この認証モードでは、両方の方法を使用して IAM プリンシパルをクラスターに追加できます。それぞれの方法には別々のエントリが保存されることに注意してください。例えば、AWS CLI からアクセスエントリを追加しても、aws-auth ConfigMap は更新されません。

アクセスエントリのみ

この認証モードでは、EKS API、AWS コマンドラインインターフェイス、AWS SDK、AWS CloudFormation、および AWS マネジメントコンソール を使用して IAM プリンシパルのクラスターへのアクセスを管理できます。

各アクセスエントリにはタイプがあり、プリンシパルを特定の名前空間に制限するアクセススコープと、事前設定された再利用可能なアクセス許可ポリシーを設定するアクセスポリシーを組み合わせて使用できます。また、STANDARD タイプと Kubernetes RBAC グループを使用してカスタムアクセス許可を割り当てることもできます。

[Authentication mode] (認証モード)	方法
`ConfigMap` のみ (`CONFIG_MAP`)	`aws-auth ConfigMap`
EKS API と `ConfigMap` (`API_AND_CONFIG_MAP`)	EKS API、AWS コマンドラインインターフェイス、AWS SDK、AWS CloudFormation、AWS マネジメントコンソール、および `aws-auth ConfigMap` のアクセスエントリ
EKS API のみ (`API`)	EKS API、AWS コマンドラインインターフェイス、AWS SDK、AWS CloudFormation、および AWS マネジメントコンソール のアクセスエントリ

> **Note**
>
> Amazon EKS Auto Mode にはアクセスエントリが必要です。

EKS アクセスエントリを使用して Kubernetes へのアクセスを IAM ユーザーに許可する

このセクションでは、アクセスエントリとポリシーを使用して、Amazon Elastic Kubernetes Service (EKS) の Kubernetes クラスターへの IAM プリンシパルアクセスを管理する方法について説明します。認証モードの変更、レガシー `aws-auth` ConfigMap エントリからの移行、アクセスエントリの作成、更新、削除、ポリシーとエントリの関連付け、事前定義されたポリシーのアクセス許可の確認、安全なアクセス管理の主な前提条件と考慮事項について詳しく説明します。

概要

EKS アクセスエントリはユーザーに Kubernetes API へのアクセスを許可する最善の方法です。例えば、アクセス エントリを使用して、開発者に kubectl を使用するアクセス権を付与できます。基本的に、EKS アクセスエントリは一連の Kubernetes アクセス許可を IAM ロールなどの IAM アイデンティティに関連付けます。例えば、開発者は IAM ロールを引き受け、それを使用して EKS クラスターを認証できます。

機能

- 認証と認可の一元化: Amazon EKS API を介して Kubernetes クラスターへのアクセスを直接制御するため、ユーザーアクセス許可のために AWS と Kubernetes API を切り替える必要がなくなります。
- 詳細なアクセス権限の管理: アクセスエントリとアクセスポリシーを使用して、クラスター作成者からのクラスター管理者権限の変更または取り消しなど、AWS IAM プリンシパルの詳細な権限を定義できます。
- IaC ツールの統合: AWS CloudFormation、Terraform、AWS CDK などの Infrastructure as Code (IaC) ツールをに対応し、クラスターの作成時にアクセス構成を定義できます。
- 設定ミスリカバリ: Kubernetes API に直接アクセスすることなく、Amazon EKS API を介してクラスターアクセスの復元を許可します。
- オーバーヘッドの削減とセキュリティの強化: CloudTrail 監査ログ記録や多要素認証などの AWS IAM 機能を活用しながら、オペレーションを一元化してオーバーヘッドを削減します。

アクセス許可をアタッチする方法

Kubernetes アクセス許可をアタッチして、次の 2 つの方法でエントリにアクセスできます：

- アクセス ポリシーを使用します。アクセスポリシーは、AWS によって管理される事前定義された Kubernetes アクセス許可テンプレートです。詳細については「the section called “アクセスポリシーを確認する”」を参照してください。
- Kubernetes グループを参照してください。IAM アイデンティティを Kubernetes グループに関連付けると、グループのアクセス許可を付与する Kubernetes リソースを作成できます。詳細についてはKubernetes ドキュメントの「RBAC 認可を使用する」を参照してください。

考慮事項

既存のクラスターで EKS アクセスエントリを有効にするときは、次の点に注意してください。

- レガシークラスターの動作: アクセスエントリが導入される前に作成されたクラスター (プラットフォームバージョン要件で指定されているものよりも前の初期プラットフォームバージョンを持つもの) の場合、EKS は既存のアクセス許可を反映するアクセスエントリを自動的に作成します。このエントリには、最初にクラスターを作成した IAM ID と、クラスターの作成中にその ID に付与された管理アクセス許可が含まれます。

- レガシー **aws-auth** ConfigMap の処理: クラスターがアクセス管理のためにレガシー aws-auth ConfigMap に依存している場合、アクセスエントリを有効にすると、元のクラスター作成者のアクセスエントリのみが自動的に作成されます。ConfigMap に追加された追加のロールまたはアクセス許可 (デベロッパーまたはサービスのカスタム IAM ロールなど) は自動的に移行されません。これに対処するには、対応するアクセスエントリを手動で作成します。

はじめに

1. 使用する IAM アイデンティティとアクセスポリシーを決定します。
   - the section called “アクセスポリシーを確認する”
2. クラスターで EKS アクセスエントリを有効にします。サポートされているプラットフォームバージョンがあることを確認します。
   - the section called “[Authentication mode] (認証モード)”
3. IAM アイデンティティを Kubernetes 権限に関連付けるアクセスエントリを作成します。
   - the section called “アクセスエントリを作成する”
4. IAM アイデンティティを使用してクラスターを認証します。
   - the section called “AWS CLI の設定”
   - the section called “kubectl と eksctl のセットアップ”

アクセスポリシーをアクセスエントリに関連付ける

1 つ以上のアクセスポリシーを タイプ STANDARD の アクセスエントリ に割り当てることができます。Amazon EKS は、クラスターで正しく機能するために必要なアクセス許可を他のタイプのアクセスエントリに自動的に付与します。Amazon EKS アクセスポリシーには、IAM アクセス許可ではなく Kubernetes アクセス許可が含まれます。アクセスポリシーをアクセスエントリに関連付ける前に、各アクセスポリシーに含まれる Kubernetes アクセス許可を既に理解できていることを確かめてください。詳細については、the section called “アクセスポリシーを確認する” を参照してください。どのアクセスポリシーも要件を満たさない場合は、アクセスポリシーをアクセスエントリに関連付けないでください。代わりに、アクセスエントリに 1 つ以上の グループ名 を指定し、Kubernetes ロールベースのアクセス制御オブジェクトを作成して管理します。詳細については、the section called “アクセスエントリを作成する” を参照してください。

- 既存のアクセスエントリ。作成する場合は、the section called “アクセスエントリを作成する” を参照してください。

- `ListAccessEntries`、`DescribeAccessEntry`、`UpdateAccessEntry`、`ListAccessPolicies`、A および `DisassociateAccessPolicy` のアクセス許可を持つ AWS Identity and Access Management ロールまたはユーザー。詳細については、「サービス認証リファレンス」の「Actions defined by Amazon Elastic Kubernetes Service」を参照してください。

アクセスポリシーをアクセスエントリに関連付ける前に、以下の要件を考慮してください。

- 各アクセスエントリには複数のアクセスポリシーを関連付けることができますが、各ポリシーを 1 つのアクセスエントリに関連付けることができるのは 1 回だけです。複数のアクセスポリシーを関連付ける場合、アクセスエントリの IAM プリンシパルには、関連するすべてのアクセスポリシーに含まれるすべてのアクセス許可が付与されます。
- アクセスポリシーをクラスター上のすべてのリソースに範囲指定することも、1 つ以上の Kubernetes 名前空間の名前を指定して範囲指定することもできます。名前空間の名前にはワイルドカード文字を使用できます。例えば、`dev-` で始まるすべての名前空間にアクセスポリシーを適用する場合は、名前空間の名前として `dev-*` を指定できます。名前空間がクラスターに存在し、スペルがクラスター上の実際の名前空間の名前と一致していることを確認してください。Amazon EKS は、クラスター上の名前空間のスペルや存在を確認しません。
- アクセスポリシーをアクセスエントリに関連付けた後、アクセスポリシーの アクセス範囲 を変更できます。アクセスポリシーの範囲を Kubernetes 名前空間に設定した場合は、必要に応じて関連付けの名前空間を追加および削除できます。
- グループ名 も指定されているアクセスエントリにアクセスポリシーを関連付けると、IAM プリンシパルには、関連するすべてのアクセスポリシーのすべてのアクセス許可が付与されます。また、グループ名を指定する Kubernetes `Role` および `RoleBinding` オブジェクトで指定されている Kubernetes `Role` または `ClusterRole` オブジェクトのすべてのアクセス許可も保持されます。
- `kubectl auth can-i --list` コマンドを実行しても、コマンドを実行したときに使用している IAM プリンシパルのアクセスエントリに関連付けられているアクセスポリシーによって割り当てられた Kubernetes アクセス許可は表示されません。このコマンドは、アクセスエントリに指定したグループ名またはユーザー名にバインドした Kubernetes `Role` または `ClusterRole` オブジェクトでアクセス許可を付与した場合にのみ、Kubernetes アクセス許可を表示します。
- `--as` *`username`* または `--as-group` *`group-name`* で `kubectl` コマンドを使用するなど、クラスター上の Kubernetes オブジェクトを操作する際に Kubernetes ユーザーまたはグループになりすますと、Kubernetes RBAC 認可の使用を強制することになります。その結果、IAM プリンシパルには、アクセスエントリに関連付けられたアクセスポリシーによって割り当てられるアクセス許可はありません。IAM プリンシパルを偽装するユーザーまたはグループが持つ Kubernetes アクセス許可は、グループ名またはユーザー名にバインドした Kubernetes `Role` また

は `ClusterRole` オブジェクトで付与した Kubernetes アクセス許可だけです。IAM プリンシパルが関連するアクセスポリシー内のアクセス許可を持つようにするには、Kubernetes ユーザーまたはグループを偽装しないでください。IAM プリンシパルには、アクセスエントリに指定したグループ名またはユーザー名にバインドした Kubernetes `Role` または `ClusterRole` オブジェクトで付与したアクセス許可も引き続き保持されます。詳細については、Kubernetes ドキュメントの「ユーザーの偽装」を参照してください。

AWS マネジメントコンソール または AWS CLI を使用してアクセスポリシーをアクセスエントリに関連付けることができます。

AWS マネジメントコンソール

1. Amazon EKS コンソール を開きます。
2. アクセスポリシーを関連付けるアクセスエントリがあるクラスターの名前を選択します。
3. アクセス タブを選択します。
4. アクセスエントリのタイプが 標準 の場合は、Amazon EKS アクセスポリシー を関連付けたり関連付けを解除したりできます。アクセスエントリのタイプが 標準 以外の場合、このオプションは使用できません。
5. [アクセスポリシーを関連付ける] を選択します。
6. [ポリシー名] には、IAM プリンシパルに付与したいアクセス許可を持つポリシーを選択します。各ポリシーに含まれるアクセス許可を表示するには、the section called “アクセスポリシーを確認する” を参照してください。
7. [アクセススコープ] では、アクセス範囲を選択します。[クラスター] を選択すると、すべての Kubernetes 名前空間のリソースについて、アクセスポリシー内のアクセス許可が IAM プリンシパルに付与されます。[Kubernetes 名前空間] を選択した場合は、[新しい名前空間の追加] を選択できます。表示される [名前空間] フィールドには、クラスター上の Kubernetes 名前空間の名前を入力できます。IAM プリンシパルに複数の名前空間のアクセス許可を持たせたい場合は、複数の名前空間を入力できます。
8. [アクセスポリシーを追加] を選択します。

AWS CLI

1. ご使用のデバイスまたは AWS CloudShell で、バージョン `2.12.3` 以降、または AWS コマンドラインインターフェイス (AWS CLI) のバージョン `1.27.160` 以降がインストールおよび設定されていること。現在のバージョンを確認するには、`aws --version | cut -d / -f2 | cut`

`-d ' ' -f1` を参照してください。yum、apt-get、macOS 用の Homebrew などのパッケージマネージャーは、多くの場合 AWS CLI の最新バージョンより数バージョン古くなっています。最新バージョンをインストールするには、「 AWS コマンドラインインターフェイスユーザーガイド」の「インストール」および「aws configure を使用したクイック設定」を参照してください。AWS クラウドシェル にインストールされている AWS CLI バージョンも最新バージョンより数バージョン遅れることがあります。更新するには、「 AWS CloudShell ユーザーガイド」の「ホームディレクトリへの AWS CLI のインストール」を参照してください。

2. 利用可能なアクセスポリシーを表示します。

```
aws eks list-access-policies --output table
```

出力例は次のとおりです。

```
----------------------------------------------------------------------------------------------------
|                                          ListAccessPolicies                                      |
|                  |                                                                              |
+-----------------------------------------------------------------------------------------------------
+
||                                            accessPolicies                                        |
|                 ||                                                                               |
|+-------------------------------------------------------------------------
+-------------------------------+|
||                                arn                                     |
 name              ||                                                                              |
|+-------------------------------------------------------------------------
+-------------------------------+|
||  {arn-aws}eks::aws:cluster-access-policy/AmazonEKSAdminPolicy        |
 AmazonEKSAdminPolicy         ||                                                                   |
||  {arn-aws}eks::aws:cluster-access-policy/AmazonEKSClusterAdminPolicy |
 AmazonEKSClusterAdminPolicy  ||                                                                   |
||  {arn-aws}eks::aws:cluster-access-policy/AmazonEKSEditPolicy         |
 AmazonEKSEditPolicy          ||                                                                   |
||  {arn-aws}eks::aws:cluster-access-policy/AmazonEKSViewPolicy         |
 AmazonEKSViewPolicy          ||                                                                   |
|+-------------------------------------------------------------------------
+-------------------------------+|
```

各ポリシーに含まれるアクセス許可を表示するには、the section called “アクセスポリシーを確認する” を参照してください。

3. 既存のアクセスエントリを表示します。`#######` の部分は自分のクラスター名に置き換えます。

```
aws eks list-access-entries --cluster-name my-cluster
```

出力例は次のとおりです。

```
{
    "accessEntries": [
        "arn:aws:iam::111122223333:role/my-role",
        "arn:aws:iam::111122223333:user/my-user"
    ]
}
```

4. アクセスポリシーをアクセスエントリに関連付けます。次の例では、AmazonEKSViewPolicy アクセスポリシーをアクセスエントリに関連付けます。*my-role* IAM ロールがクラスター上の Kubernetes オブジェクトにアクセスしようとするたびに、Amazon EKS は、ポリシー内のアクセス許可を使用して *my-namespace1* と *my-namespace2* の Kubernetes 名前空間のみの Kubernetes オブジェクトにアクセスするアクセス許可をロールに付与します。*my-cluster* をクラスターの名前、*111122223333* を AWS アカウント ID に、*my-role* を Amazon EKS に Kubernetes クラスターオブジェクトへのアクセスを許可してほしい IAM ロールの名前に置き換えます。

```
aws eks associate-access-policy --cluster-name my-cluster --principal-arn
 arn:aws:iam::111122223333:role/my-role \
    --access-scope type=namespace,namespaces=my-namespace1,my-namespace2 --policy-arn
 arn:aws:eks::aws:cluster-access-policy/AmazonEKSViewPolicy
```

IAM プリンシパルにクラスター全体のアクセス許可を持たせたい場合は、type=namespace,namespaces=*my-namespace1*,*my-namespace2* を type=cluster に置き換えてください。複数のアクセスポリシーをアクセスエントリに関連付ける場合は、それぞれに固有のアクセスポリシーを指定してコマンドを複数回実行します。関連するアクセスポリシーにはそれぞれ独自の範囲があります。

> **Note**
>
> 関連するアクセスポリシーの範囲を後で変更する場合は、新しい範囲で前のコマンドをもう一度実行してください。例えば、*my-namespace2* を削除したい場合は、type=namespace,namespaces=*my-namespace1* のみを使用してコマンドを再実行します。範囲を namespace から cluster に変更する場合は、type=cluster を使用

して type=namespace,namespaces=*my-namespace1*,*my-namespace2* を削除して コマンドを再実行します。

5. どのアクセスポリシーをアクセスエントリに関連付けるかを決定します。

```
aws eks list-associated-access-policies --cluster-name my-cluster --principal-arn
 arn:aws:iam::111122223333:role/my-role
```

出力例は次のとおりです。

```
{
    "clusterName": "my-cluster",
    "principalArn": "arn:aws:iam::111122223333:role/my-role",
    "associatedAccessPolicies": [
        {
            "policyArn": "arn:aws:eks::aws:cluster-access-policy/
AmazonEKSViewPolicy",
            "accessScope": {
                "type": "cluster",
                "namespaces": []
            },
            "associatedAt": "2023-04-17T15:25:21.675000-04:00",
            "modifiedAt": "2023-04-17T15:25:21.675000-04:00"
        },
        {
            "policyArn": "arn:aws:eks::aws:cluster-access-policy/
AmazonEKSAdminPolicy",
            "accessScope": {
                "type": "namespace",
                "namespaces": [
                    "my-namespace1",
                    "my-namespace2"
                ]
            },
            "associatedAt": "2023-04-17T15:02:06.511000-04:00",
            "modifiedAt": "2023-04-17T15:02:06.511000-04:00"
        }
    ]
}
```

前の例では、このアクセスエントリの IAM プリンシパルには、クラスター上のすべての名前空間に対する表示アクセス許可と、2 つの Kubernetes 名前空間に対する管理者アクセス許可があります。

6. アクセスポリシーとアクセスエントリの関連付けを解除します。この例では、AmazonEKSAdminPolicy ポリシーとアクセスエントリの関連付けが解除されます。ただし、IAM プリンシパルは *my-namespace1* と *my-namespace2* 名前空間のオブジェクトに対する AmazonEKSViewPolicy アクセスポリシー内のアクセス許可を保持します。これは、そのアクセスポリシーがアクセスエントリとの関連付けを解除されていないためです。

```
aws eks disassociate-access-policy --cluster-name my-cluster --principal-arn
 arn:aws:iam::111122223333:role/my-role \
    --policy-arn arn:aws:eks::aws:cluster-access-policy/AmazonEKSAdminPolicy
```

利用可能なアクセスポリシーを一覧表示するには、the section called “アクセスポリシーを確認する”を参照してください。

既存の aws-auth ConfigMap エントリをアクセスエントリに移行する

クラスター上の aws-auth ConfigMap にエントリを追加した場合は、aws-auth ConfigMap 内の既存のエントリのアクセスエントリを作成することをお勧めします。アクセスエントリを作成したら、そのエントリを ConfigMap から削除できます。aws-authConfigMap のエントリにアクセスポリシーを関連付けることはできません。IAM プリンシパルにアクセスポリシーを関連付けるには、アクセスエントリを作成します。

> **Important**
>
> - クラスターが API_AND_CONFIGMAP 認証モードで、aws-auth ConfigMap およびアクセスエントリの両方に同じ IAMロールのマッピングがある場合、ロールによってアクセスエントリのマッピングが認証に使用されます。アクセスエントリは、同じ IAM プリンシパルの ConfigMap エントリよりも優先されます。
> - マネージドノードグループまたは Fargate プロファイル用に Amazon EKS によってお使いのクラスターに作成された既存の aws-auth ConfigMap エントリを削除する際は、事前にそれらの特定リソースの正しいアクセスエントリが Amazon EKS クラスターに存在するかどうかを再確認してください。同等のアクセスエントリがない状態で Amazon EKS が ConfigMap で作成したエントリを削除した場合、クラスターが正しく機能しません。

前提条件

- アクセスエントリとアクセスポリシーに関する知識。詳細については、「the section called “アクセスエントリ”」および「the section called “アソシエイトアクセスポリシー”」を参照してください。
- 「the section called “アクセスエントリ”」のトピックの前提条件に記載されているバージョンと同じかそれ以降のプラットフォームバージョンの既存のクラスター。
- デバイスまたは AWS CloudShell にインストールされている `eksctl` コマンドラインツールのバージョン `0.215.0` 以降。`eksctl` をインストールまたはアップグレードするには、`eksctl` ドキュメントの「インストール」を参照してください。
- `kube-system` 名前空間内の `aws-auth` `ConfigMap` を変更する Kubernetes アクセス許可。
- `CreateAccessEntry` および `ListAccessEntries` のアクセス許可を持つ AWS Identity and Access Management ロールまたはユーザー。詳細については、サービス認可リファレンスの「Amazon Elastic Kubernetes Service で定義されるアクション」を参照してください。

`eksctl`

1. `aws-auth` `ConfigMap` 内の既存のエントリを表示します。*`my-cluster`* の部分は自分のクラスター名に置き換えます。

```
eksctl get iamidentitymapping --cluster my-cluster
```

出力例は次のとおりです。

```
ARN
            USERNAME                                GROUPS
                        ACCOUNT
arn:aws:iam::111122223333:role/EKS-my-cluster-Admins
            Admins                                  system:masters
arn:aws:iam::111122223333:role/EKS-my-cluster-my-namespace-Viewers
            my-namespace-Viewers                    Viewers
arn:aws:iam::111122223333:role/EKS-my-cluster-self-managed-ng-1
            system:node:{{EC2PrivateDNSName}}       system:bootstrappers,system:nodes
arn:aws:iam::111122223333:user/my-user
            my-user
arn:aws:iam::111122223333:role/EKS-my-cluster-fargateprofile1
                                system:node:{{SessionName}}
 system:bootstrappers,system:nodes,system:node-proxier
```

```
arn:aws:iam::111122223333:role/EKS-my-cluster-managed-ng
            system:node:{{EC2PrivateDNSName}}       system:bootstrappers,system:nodes
```

2. 前の出力で返された作成済み ConfigMap エントリのいずれかへの the section called "アクセスエントリを作成する"。アクセスエントリを作成するときは、出力で返されるARN、USERNAME、GROUPS および ACCOUNT と同じ値を指定してください。出力例では、最後の 2 つのエントリを除くすべてのエントリのアクセスエントリを作成します。これらのエントリは Fargate プロファイルとマネージド型ノードグループ用に Amazon EKS によって作成されたものだからです。
3. 作成したすべてのアクセスエントリのエントリを ConfigMap から削除します。ConfigMap からエントリを削除しない場合、IAM プリンシパル ARN のアクセスエントリの設定によって ConfigMap エントリが上書きされます。*111122223333* は AWS アカウント ID に、*EKS-my-cluster-my-namespace-Viewers* は、ConfigMap のエントリに含まれるロールの名前に置き換えます。削除するエントリが IAM ロールではなく IAM ユーザー用のものである場合は、role を user に、*EKS-my-cluster-my-namespace-Viewers* をユーザー名に置き換えます。

```
eksctl delete iamidentitymapping --arn arn:aws:iam::111122223333:role/EKS-my-cluster-
my-namespace-Viewers --cluster my-cluster
```

アクセスポリシーアクセス許可を確認する

アクセスポリシーには、Kubernetes verbs (アクセス許可) と resources を含む rules が含まれます。アクセスポリシーには IAM のアクセス許可やリソースは含まれません。Kubernetes Role および ClusterRole オブジェクトと同様に、アクセスポリシーには allow rules のみが含まれます。アクセスポリシーの内容は変更できません。独自のアクセスポリシーを作成することはできません。アクセスポリシーのアクセス許可がニーズに合わない場合は、Kubernetes RBAC オブジェクトを作成し、アクセスエントリのグループ名を指定します。詳細については、「the section called "アクセスエントリを作成する"」を参照してください。アクセスポリシーに含まれるアクセス許可は、Kubernetes ユーザー向けのクラスターロールのアクセス許可と似ています。詳細については、Kubernetes ドキュメントの「ユーザー向け Role」を参照してください。

すべてのポリシーをリストする

このページに記載されているいずれかのアクセスポリシーを使用するか、AWS CLI を使用して使用可能なすべてのアクセスポリシーのリストを取得します。

```
aws eks list-access-policies
```

予想される出力は次のようになります (簡潔にするために省略しています)。

```
{
    "accessPolicies": [
        {
            "name": "AmazonAIOpsAssistantPolicy",
            "arn": "arn:aws:eks::aws:cluster-access-policy/AmazonAIOpsAssistantPolicy"
        },
        {
            "name": "AmazonARCRegionSwitchScalingPolicy",
            "arn": "arn:aws:eks::aws:cluster-access-policy/
AmazonARCRegionSwitchScalingPolicy"
        },
        {
            "name": "AmazonEKSAdminPolicy",
            "arn": "arn:aws:eks::aws:cluster-access-policy/AmazonEKSAdminPolicy"
        },
        {
            "name": "AmazonEKSAdminViewPolicy",
            "arn": "arn:aws:eks::aws:cluster-access-policy/AmazonEKSAdminViewPolicy"
        },
        {
            "name": "AmazonEKSAutoNodePolicy",
            "arn": "arn:aws:eks::aws:cluster-access-policy/AmazonEKSAutoNodePolicy"
        }
        // Additional policies omitted
    ]
}
```

AmazonEKSAdminPolicy

このアクセスポリシーには、リソースに対するほとんどのアクセス許可を IAM プリンシパルに付与するアクセス許可が含まれています。アクセスエントリに関連付けられている場合、そのアクセス範囲は通常 1 つ以上の Kubernetes 名前空間です。IAM プリンシパルにクラスター上のすべてのリソースへの管理者アクセスを許可する場合は、代わりに the section called “AmazonEKSClusterAdminPolicy” アクセスポリシーをアクセスエントリに関連付けてください。

ARN — `arn:aws:eks::aws:cluster-access-policy/AmazonEKSAdminPolicy`

Kubernetes API グループ	Kubernetes resources	Kubernetes 動詞 (許可)
apps	daemonsets , deployments , deployments/rollback , deployments/scale , replicasets , replicasets/scale, statefulsets , statefulsets/scale	create, delete, deletecollection , patch, update
apps	controllerrevisions , daemonsets , daemonsets/status , deployments , deployments/scale , deployments/status , replicasets , replicasets/scale , replicasets/status , statefulsets , statefulsets/scale , statefulsets/status	get, list, watch
authorization.k8s.io	localsubjectaccessreviews	create
autoscaling	horizontalpodautoscalers	create, delete, deletecollection , patch, update
autoscaling	horizontalpodautoscalers , horizontalpodautoscalers/status	get, list, watch
batch	cronjobs, jobs	create, delete, deletecollection , patch, update

Kubernetes API グループ	Kubernetes resources	Kubernetes 動詞 (許可)
batch	cronjobs, cronjobs/status , jobs, jobs/status	get, list, watch
discovery.k8s.io	endpointslices	get, list, watch
extensions	daemonsets , deployments , deployments/rollback , deployments/scale , ingresses , networkpolicies , replicasets , replicasets/scale , replicationcontrollers/scale	create, delete, deletecollection , patch, update
extensions	daemonsets , daemonsets/status , deployments , deployments/scale , deployments/status , ingresses , ingresses/status , networkpolicies , replicasets , replicasets/scale , replicasets/status , replicationcontrollers/scale	get, list, watch
networking.k8s.io	ingresses , ingresses/status , networkpolicies	get, list, watch
networking.k8s.io	ingresses , networkpolicies	create, delete, deletecollection , patch, update
policy	poddisruptionbudgets	create, delete, deletecollection , patch, update

Kubernetes API グループ	Kubernetes resources	Kubernetes 動詞 (許可)
policy	poddisruptionbudgets , poddisruptionbudgets/status	get, list, watch
rbac.authorization.k8s.io	rolebindings , roles	create, delete, deletecollection , get, list, patch, update, watch
	configmaps , endpoints , persistentvolumeclaims , persistentvolumeclaims/status , pods, replicationcontrollers , replicationcontrollers/scale , serviceaccounts , services, services/status	get, list, watch
	pods/attach , pods/exec , pods/portforward , pods/proxy , secrets, services/proxy	get, list, watch
	configmaps , events, persistentvolumeclaims , replicationcontrollers , replicationcontrollers/scale , secrets, serviceaccounts , services, services/proxy	create, delete, deletecollection , patch, update

Kubernetes API グループ	Kubernetes resources	Kubernetes 動詞（許可）
	pods, pods/attach , pods/exec , pods/port forward , pods/proxy	create, delete, deletecollection , patch, update
	serviceaccounts	impersonate
	bindings, events, limitranges , namespaces/status , pods/log, pods/status , replicationcontrollers/status , resourcequotas , resourcequotas/status	get, list, watch
	namespaces	get, list, watch

AmazonEKSClusterAdminPolicy

このアクセスポリシーには、IAM プリンシパル管理者にクラスターへのアクセス権を付与するアクセス許可が含まれています。アクセスエントリに関連付けられている場合、そのアクセス範囲は通常、Kubernetes 名前空間ではなくクラスターになります。IAM プリンシパルの管理範囲をより限定したい場合は、代わりに the section called “AmazonEKSAdminPolicy” アクセスポリシーをアクセスエントリに関連付けることを検討してください。

ARN — arn:aws:eks::aws:cluster-access-policy/AmazonEKSClusterAdminPolicy

Kubernetes API グループ	Kubernetes nonResourceURL	Kubernetes resources	Kubernetes 動詞（許可）
*		*	*
	*		*

AmazonEKSAdminViewPolicy

このアクセスポリシーには、IAM プリンシパルにクラスターの全リソースをリスト/表示するアクセス権を付与するアクセス許可が含まれています。これには Kubernetes シークレットが含まれることに注意してください。

ARN — `arn:aws:eks::aws:cluster-access-policy/AmazonEKSAdminViewPolicy`

Kubernetes API グループ	Kubernetes resources	Kubernetes 動詞 (許可)
*	*	get, list, watch

AmazonEKSEditPolicy

このアクセスポリシーには、IAM プリンシパルがほとんどの Kubernetes リソースを編集できるようにするアクセス許可が含まれています。

ARN — `arn:aws:eks::aws:cluster-access-policy/AmazonEKSEditPolicy`

Kubernetes API グループ	Kubernetes resources	Kubernetes 動詞 (許可)
apps	daemonsets , deployments , deployments/rollback , deployments/scale , replicasets , replicasets/scale, statefulsets , statefulsets/scale	create, delete, deletecollection , patch, update
apps	controllerrevisions , daemonsets , daemonsets/status , deployments , deployments/scale , deployments/status , replicasets , replicasets/scale , replicasets/status , statefulsets , statefulsets/	get, list, watch

Kubernetes API グループ	Kubernetes resources	Kubernetes 動詞（許可）
	scale , statefulsets/status	
autoscaling	horizontalpodautoscalers , horizontalpodautoscalers/status	get, list, watch
autoscaling	horizontalpodautoscalers	create, delete, deletecollection , patch, update
batch	cronjobs, jobs	create, delete, deletecollection , patch, update
batch	cronjobs, cronjobs/status , jobs, jobs/status	get, list, watch
discovery.k8s.io	endpointslices	get, list, watch
extensions	daemonsets , deployments , deployments/rollback , deployments/scale , ingresses , networkpolicies , replicasets , replicasets/scale , replicationcontrollers/scale	create, delete, deletecollection , patch, update

Kubernetes API グループ	Kubernetes resources	Kubernetes 動詞 (許可)
extensions	daemonsets , daemonsets/status , deployments , deployments/scale , deployments/status , ingresses , ingresses/status , networkpolicies , replicasets , replicasets/scale , replicasets/status , replicationcontrollers/scale	get, list, watch
networking.k8s.io	ingresses , networkpolicies	create, delete, deletecollection , patch, update
networking.k8s.io	ingresses , ingresses/status , networkpolicies	get, list, watch
policy	poddisruptionbudgets	create, delete, deletecollection , patch, update
policy	poddisruptionbudgets , poddisruptionbudgets/status	get, list, watch
	namespaces	get, list, watch
	pods/attach , pods/exec , pods/portforward , pods/proxy , secrets, services/proxy	get, list, watch
	serviceaccounts	impersonate

Kubernetes API グループ	Kubernetes resources	Kubernetes 動詞 (許可)
	pods, pods/attach , pods/exec , pods/port forward , pods/proxy	create, delete, deletecollection , patch, update
	configmaps , events, persistentvolumeclaims , replicationcontrollers , replicationcontrollers/scale , secrets, serviceaccounts , services, services/proxy	create, delete, deletecollection , patch, update
	configmaps , endpoints , persistentvolumeclaims , persistentvolumeclaims/status , pods, replicationcontrollers , replicationcontrollers/scale , serviceaccounts , services, services/status	get, list, watch
	bindings, events, limitranges , namespaces/status , pods/log, pods/status , replicationcontrollers/status , resourcequotas , resourcequotas/status	get, list, watch

AmazonEKSViewPolicy

このアクセスポリシーには、IAM プリンシパルがほとんどの Kubernetes リソースを表示できるようにするアクセス許可が含まれています。

ARN — `arn:aws:eks::aws:cluster-access-policy/AmazonEKSViewPolicy`

Kubernetes API グループ	Kubernetes resources	Kubernetes 動詞 (許可)
apps	controllerrevisions , daemonsets , daemonsets/status , deployments , deployments/scale , deployments/status , replicasets , replicasets/scale , replicasets/status , statefulsets , statefulsets/scale , statefulsets/status	get, list, watch
autoscaling	horizontalpodautoscalers , horizontalpodautoscalers/status	get, list, watch
batch	cronjobs, cronjobs/status , jobs, jobs/status	get, list, watch
discovery.k8s.io	endpointslices	get, list, watch
extensions	daemonsets , daemonsets/status , deployments , deployments/scale , deployments/status , ingresses , ingresses/status , networkpolicies , replicasets ,	get, list, watch

Kubernetes API グループ	Kubernetes resources	Kubernetes 動詞 (許可)
	replicasets/scale , replicasets/status , replicationcontrollers/scale	
networking.k8s.io	ingresses , ingresses/status , networkpolicies	get, list, watch
policy	poddisruptionbudgets , poddisruptionbudgets/status	get, list, watch
	configmaps , endpoints , persistentvolumeclaims , persistentvolumeclaims/status , pods, replicationcontrollers , replicationcontrollers/scale , serviceaccounts , services, services/status	get, list, watch
	bindings, events, limitranges , namespaces/status , pods/log, pods/status , replicationcontrollers/status , resourcequotas , resourcequotas/status	get, list, watch
	namespaces	get, list, watch

AmazonEKSPodIdentityPolicy

このアクセスポリシーに含まれているアクセス許可により、IAM プリンシパルは Kubernetes Pods を一覧表示することができます。このポリシーは、Amazon EKS サービスにリンクされたロールによって内部で使用されます。

> **Note**
>
> このポリシーは AWS サービスリンクロール専用で、カスタマーマネージドロールには使用できません。

ARN — `arn:aws:eks::aws:cluster-access-policy/AmazonEKSPodIdentityPolicy`

Kubernetes API グループ	Kubernetes resources	Kubernetes 動詞 (許可)
	`pods`	`list`

AmazonEKSSecretAdminPolicy

このアクセスポリシーに含まれているアクセス許可により、IAM プリンシパルは Kubernetes シークレットにフルアクセスできます。

ARN — `arn:aws:eks::aws:cluster-access-policy/AmazonEKSSecretAdminPolicy`

Kubernetes API グループ	Kubernetes resources	Kubernetes 動詞 (許可)
	`secrets`	*

AmazonEKSSecretReaderPolicy

このアクセスポリシーに含まれているアクセス許可により、IAM プリンシパルは Kubernetes シークレットを読み取ることができます。

ARN — `arn:aws:eks::aws:cluster-access-policy/AmazonEKSSecretReaderPolicy`

Kubernetes API グループ	Kubernetes resources	Kubernetes 動詞 (許可)
	secrets	get, list, watch

AmazonEKSAutoNodePolicy

ARN — arn:aws:eks::aws:cluster-access-policy/AmazonEKSAutoNodePolicy

このポリシーには、Amazon EKS コンポーネントが以下のタスクを完了できるようにする次のアクセス許可が含まれています。

- kube-proxy – ネットワークエンドポイントおよびサービスをモニタリングし、関連するイベントを管理します。これにより、クラスター全体のネットワークプロキシ機能が有効になります。
- ipamd – AWS VPC ネットワーキングリソースと Container Network Interface (CNI) を管理します。これにより、IP アドレス管理デーモンがポッドネットワーキングを処理できるようになります。
- coredns – エンドポイントやサービスなどのサービス検出リソースにアクセスします。これにより、クラスター内の DNS 解決が有効になります。
- ebs-csi-driver – Amazon EBS ボリュームのストレージ関連リソースを操作します。これにより、永続ボリュームの動的なプロビジョニングとアタッチが可能になります。
- neuron – AWS Neuron デバイスのノードとポッドをモニタリングします。これにより、AWS Inferentia および Trainium アクセラレーターを管理できます。
- node-monitoring-agent – ノード診断およびイベントにアクセスします。これにより、クラスターのヘルスモニタリングと診断の収集が可能になります。

各コンポーネントは専用のサービスアカウントを使用し、その特定の機能に必要なアクセス許可のみに制限されます。

NodeClass でノード IAM ロールを手動で指定する場合には、新しいノード IAM ロールをこのアクセスポリシーに関連付けるアクセスエントリを作成する必要があります。

Amazon EKSブロックストレージポリシー

> **Note**
>
> このポリシーは AWS サービスリンクロール専用で、カスタマーマネージドロールには使用できません。

ARN — `arn:aws:eks::aws:cluster-access-policy/AmazonEKSBlockStoragePolicy`

このポリシーには、Amazon EKS がストレージオペレーションのリーダー選出および調整リソースを管理できるようにするアクセス許可が含まれています。

- `coordination.k8s.io` – リーダー選出用のリースオブジェクトを作成および管理します。これにより、EKS ストレージコンポーネントは、リーダー選出メカニズムを通じてクラスター全体のアクティビティを調整できます。

ポリシーは、クラスター内の他の調整リソースへのアクセス競合を防ぐために、EKS ストレージコンポーネントで使用される特定のリースリソースを対象とします。

Amazon EKS は、Auto Mode が有効の場合、クラスター IAM ロールのこのアクセスポリシーを使用してアクセスエントリを自動的に作成し、ブロックストレージ機能が正しく機能するのに必要なアクセス許可を設定します。

AmazonEKSLoadBalancingPolicy

ARN — `arn:aws:eks::aws:cluster-access-policy/AmazonEKSLoadBalancingPolicy`

このポリシーには、Amazon EKS がロードバランシングのリーダー選出リソースを管理できるようにするアクセス許可が含まれています。

- `coordination.k8s.io` – リーダー選出用のリースオブジェクトを作成および管理します。これにより、EKS ロードバランシングコンポーネントはリーダーを選出することで、複数のレプリカ間でアクティビティを調整できます。

このポリシーは、クラスター内の他のリースリソースへのアクセスを防止しながら、適切な調整を実現するために、特にロードバランシングリースリソースを対象とします。

Amazon EKS は、Auto Mode が有効の場合、クラスター IAM ロールのこのアクセスポリシーを使用してアクセスエントリを自動的に作成し、ネットワーク機能が正しく機能するのに必要なアクセス許可を設定します。

AmazonEKSNetworkingPolicy

ARN — `arn:aws:eks::aws:cluster-access-policy/AmazonEKSNetworkingPolicy`

このポリシーには、Amazon EKS がネットワーキングのリーダー選出リソースを管理できるようにするアクセス許可が含まれています。

- `coordination.k8s.io` – リーダー選出用のリースオブジェクトを作成および管理します。これにより、EKS ネットワークコンポーネントはリーダーを選出することで、IP アドレス割り当てアクティビティを調整できます。

このポリシーは、クラスター内の他のリースリソースへのアクセスを防止しながら、適切な調整を実現するために、特にネットワークリースリソースを対象とします。

Amazon EKS は、Auto Mode が有効の場合、クラスター IAM ロールのこのアクセスポリシーを使用してアクセスエントリを自動的に作成し、ネットワーク機能が正しく機能するのに必要なアクセス許可を設定します。

Amazon EKSコンピュートポリシー

> **Note**
>
> このポリシーは AWS サービスリンクロール専用で、カスタマーマネージドロールには使用できません。

ARN — `arn:aws:eks::aws:cluster-access-policy/AmazonEKSComputePolicy`

このポリシーには、Amazon EKS がコンピューティングオペレーションのリーダー選出リソースを管理できるようにするアクセス許可が含まれています。

- `coordination.k8s.io` – リーダー選出用のリースオブジェクトを作成および管理します。これにより、EKS コンピューティングコンポーネントは、リーダーを選出することでノードスケーリングアクティビティを調整できます。

このポリシーは、クラスター内のすべてのリースリソースへの基本的な読み取りアクセス (get、watch) を許可しながら、特にコンピューティング管理リースリソースを対象とします。

Amazon EKS は、Auto Mode が有効の場合、クラスター IAM ロールのこのアクセスポリシーを使用してアクセスエントリを自動的に作成し、コンピューティング管理機能が正しく機能するのに必要なアクセス許可を設定します。

AmazonEKSBlockStorageClusterPolicy

> **Note**
>
> このポリシーは AWS サービスリンクロール専用で、カスタマーマネージドロールには使用できません。

ARN — `arn:aws:eks::aws:cluster-access-policy/AmazonEKSBlockStorageClusterPolicy`

このポリシーは、Amazon EKS Auto Mode のブロックストレージ機能に必要なアクセス許可を付与します。これにより、Amazon EKS クラスター内のブロックストレージリソースを効率的に管理できます。 ポリシーには以下のアクセス許可が含まれています。

CSI ドライバー管理:

- CSI ドライバー (特にブロックストレージ用) の作成、読み取り、更新、削除を行います。

ボリューム管理:

- 永続ボリュームの一覧表示、監視、作成、更新、パッチ適用、削除を行います。
- 永続ボリュームのクレームを一覧表示、監視、更新します。
- パッチ永続ボリュームのクレームのステータス。

ノードとポッドのインタラクション:

- ノードおよびポッドの情報を読み取ります。
- ストレージオペレーションに関連するイベントを管理します。

ストレージクラスおよび属性:

- ストレージクラスと CSI ノードを読み取ります。
- ボリューム属性クラスを読み取ります。

ボリュームのアタッチ:

- ボリュームのアタッチとそのステータスを一覧表示、監視、変更します。

スナップショットオペレーション:

- ボリュームのスナップショット、スナップショットコンテンツ、スナップショットクラスを管理します。
- ボリュームグループスナップショットおよび関連リソースのオペレーションを処理します。

このポリシーは、Auto Mode で実行されている Amazon EKS クラスター内の包括的なブロックストレージ管理をサポートするように設計されています。ブロックストレージボリュームのプロビジョニング、アタッチ、サイズ変更、スナップショット作成など、さまざまなオペレーションに対するアクセス許可を組み合わせます。

Amazon EKS は、Auto Mode が有効の場合、クラスター IAM ロールのこのアクセスポリシーを使用してアクセスエントリを自動的に作成し、ブロックストレージ機能が正しく機能するのに必要なアクセス許可を設定します。

AmazonEKSComputeClusterPolicy

> **Note**
>
> このポリシーは AWS サービスリンクロール専用で、カスタマーマネージドロールには使用できません。

ARN — `arn:aws:eks::aws:cluster-access-policy/AmazonEKSComputeClusterPolicy`

このポリシーは、Amazon EKS Auto Mode のコンピューティング管理機能に必要なアクセス許可を付与します。これにより、Amazon EKS クラスター内のコンピューティングリソースを効率的にオーケストレーションおよびスケールできます。 ポリシーには以下のアクセス許可が含まれています。

ノード管理:

- NodePools および NodeClaims のステータスの作成、読み取り、更新、削除、管理を行います。
- 作成、変更、削除を含め、NodeClasses を管理します。

スケジューリングとリソース管理:

- ポッド、ノード、永続ボリューム、永続ボリュームのクレーム、レプリケーションコントローラー、名前空間への読み取りアクセス。
- ストレージクラス、CSI ノード、ボリュームのアタッチへの読み取りアクセス。
- デプロイ、デーモンセット、レプリカセット、ステートフルセットを一覧表示および監視します。
- ポッド中断の予算を読み取ります。

イベント処理:

- クラスターイベントの作成、読み取り、管理を行います。

ノードのプロビジョニング解除とポッド エビクション:

- ノードの更新、パッチ適用、削除を行います。
- 必要に応じて、ポッド エビクションを作成してポッドを削除します。

カスタムリソース定義 (CRD) 管理:

- 新しい CRD を作成します。
- ノード管理に関連する特定の CRD (NodeClasses、NodePools、NodeClaims、NodeDiagnostics) を管理します。

このポリシーは、Auto Mode で実行されている Amazon EKS クラスター内の包括的なコンピューティング管理をサポートするように設計されています。ノードのプロビジョニング、スケジューリング、スケーリング、リソースの最適化など、さまざまなオペレーションのアクセス許可を組み合わせます。

Amazon EKS は、Auto Mode が有効の場合、クラスター IAM ロールのこのアクセスポリシーを使用してアクセスエントリを自動的に作成し、コンピューティング管理機能が正しく機能するのに必要なアクセス許可を設定します。

AmazonEKSLoadBalancingClusterPolicy

ARN — `arn:aws:eks::aws:cluster-access-policy/AmazonEKSLoadBalancingClusterPolicy`

このポリシーは、Amazon EKS Auto Mode のロードバランシング機能に必要なアクセス許可を付与します。これにより、Amazon EKS クラスター内のロードバランシングリソースを効率的に管理および設定できます。 ポリシーには以下のアクセス許可が含まれています。

イベントとリソースの管理:

- イベントを作成してパッチを適用します。
- ポッド、ノード、エンドポイント、名前空間への読み取りアクセス。
- ポッドのステータスを更新します。

サービスおよびイングレス管理:

- サービスとそのステータスの完全な管理。
- イングレスとそのステータスの包括的な制御。
- エンドポイントスライスとイングレスクラスへの読み取りアクセス。

ターゲットグループのバインディング:

- ターゲットグループのバインディングとそのステータスを作成および変更します。
- イングレスクラスパラメータへの読み取りアクセス。

カスタムリソース定義 (CRD) 管理:

- すべての CRD の作成および読み取りを行います。
- targetgroupbindings.eks.amazonaws.com および ingressclassparams.eks.amazonaws.com CRD の特定の管理。

ウェブフック設定:

- 変異および検証ウェブフック設定の作成および読み取りを行います。
- eks-load-balancing-webhook 設定を管理します。

このポリシーは、Auto Mode で実行されている Amazon EKS クラスター内の包括的なロードバランシング管理をサポートするように設計されています。サービスの公開、イングレスルーティング、AWS ロードバランシングサービスとの統合など、さまざまなオペレーションのアクセス許可を組み合わせます。

Amazon EKS は、Auto Mode が有効の場合、クラスター IAM ロールのこのアクセスポリシーを使用してアクセスエントリを自動的に作成し、ロードバランシング機能が正しく機能するのに必要なアクセス許可を設定します。

AmazonEKSNetworkingClusterPolicy

ARN — `arn:aws:eks::aws:cluster-access-policy/AmazonEKSNetworkingClusterPolicy`

AmazonEKSNetworkingClusterPolicy

このポリシーは、Amazon EKS Auto Mode のネットワーキング機能に必要なアクセス許可を付与します。これにより、Amazon EKS クラスター内のネットワーキングリソースを効率的に管理および設定できます。 ポリシーには以下のアクセス許可が含まれています。

ノードとポッドの管理:

- NodeClasses とそのステータスへの読み取りアクセス。
- NodeClaims とそのステータスへの読み取りアクセス。
- ポッドへの読み取りアクセス。

CNI ノード管理:

- 作成、読み取り、更新、削除、パッチ適用などの、CNINodes とそのステータスのアクセス許可。

カスタムリソース定義 (CRD) 管理:

- すべての CRD の作成および読み取りを行います。
- cninodes.eks.amazonaws.com CRD の特定の管理 (更新、パッチ適用、削除)。

イベント管理:

- イベントを作成してパッチを適用します。

このポリシーは、Auto Mode で実行されている Amazon EKS クラスター内の包括的なネットワーク管理をサポートするように設計されています。ノードネットワーキング設定、CNI (Container Network Interface) 管理、関連するカスタムリソース処理など、さまざまなオペレーションのアクセス許可を組み合わせます。

このポリシーにより、ネットワーキングコンポーネントはノード関連のリソースを操作し、CNI 固有のノード設定を管理し、クラスター内のネットワーキングオペレーションに不可欠なカスタムリソースを処理できます。

Amazon EKS は、Auto Mode が有効の場合、クラスター IAM ロールのこのアクセスポリシーを使用してアクセスエントリを自動的に作成し、ネットワーク機能が正しく機能するのに必要なアクセス許可を設定します。

AmazonEKSHybridPolicy

> **Note**
>
> このポリシーは AWS サービスリンクロール専用で、カスタマーマネージドロールには使用できません。

このアクセスポリシーには、クラスターのノードへの EKS アクセスを許可するアクセス許可が含まれています。アクセスエントリに関連付けられている場合、そのアクセス範囲は通常、Kubernetes 名前空間ではなくクラスターになります。このポリシーは、Amazon EKS Hybrid Nodes で使用されます。

ARN — `arn:aws:eks::aws:cluster-access-policy/AmazonEKSHybridPolicy`

Kubernetes API グループ	Kubernetes nonResourceURL	Kubernetes resources	Kubernetes 動詞 (許可)
*		nodes	list

AmazonEKSClusterInsightsPolicy

> **Note**
>
> このポリシーは AWS サービスリンクロール専用で、カスタマーマネージドロールには使用できません。

ARN — `arn:aws:eks::aws:cluster-access-policy/AmazonEKSClusterInsightsPolicy`

このポリシーは、Amazon EKS クラスターインサイト機能の読み取り専用アクセス許可を付与します。 ポリシーには以下のアクセス許可が含まれています。

ノードアクセス: - クラスターノードの一覧表示および表示 - ノードのステータス情報の読み取り

DaemonSet アクセス: - kube-proxy 設定に対する読み取りアクセス

このポリシーは、クラスターインサイトの EKS サービスによって自動的に管理されます。詳細については、「the section called “クラスターのインサイト”」を参照してください。

AWSBackupFullAccessPolicyForBackup

ARN — `arn:aws:eks::aws:cluster-access-policy/AWSBackupFullAccessPolicyForBackup`

AWSBackupFullAccessPolicyForBackup

このポリシーは、AWS バックアップが EKS クラスターのバックアップを管理および作成するために必要なアクセス許可を付与します。このポリシーには、以下のアクセス許可が含まれています。

Kubernetes API グループ	Kubernetes resources	Kubernetes 動詞 (許可)
*	*	list, get

AWSBackupFullAccessPolicyForRestore

ARN — `arn:aws:eks::aws:cluster-access-policy/AWSBackupFullAccessPolicyForRestore`

AWSBackupFullAccessPolicyForRestore

このポリシーは、AWS バックアップが EKS クラスターのバックアップを管理および復元するために必要なアクセス許可を付与します。このポリシーには、以下のアクセス許可が含まれています。

Kubernetes API グループ	Kubernetes resources	Kubernetes 動詞 (許可)
*	*	list, get, create
rbac.authorization.k8s.io	roles, clusterroles	create, update, escalate, bind

AmazonEKSACKPolicy

ARN — arn:aws:eks::aws:cluster-access-policy/AmazonEKSACKPolicy

このポリシーは、Kubernetes 用 AWS コントローラー (ACK) 機能が Kubernetes から AWS リソースを管理するために必要なアクセス許可を付与します。 ポリシーには以下のアクセス許可が含まれています。

ACK カスタムリソース管理:

- S3、RDS、DynamoDB、Lambda、EC2 など、50 を超える AWS サービス全体ですべての ACK サービスカスタムリソースにフルアクセスします。
- ACK カスタムリソース定義を作成、読み取り、更新、および削除します。

名前空間アクセス:

- リソースを編成するために名前空間に読み取りアクセスします。

リーダー選出:

- リーダーを選出するために調整リースを作成および読み取ります。
- 特定の ACK サービスコントローラーリースを更新および削除します。

イベント管理:

- ACK のオペレーション用にイベントを作成してパッチを適用します。

このポリシーは、Kubernetes API を介した包括的な AWS リソース管理をサポートするように設計されています。ACK 機能の作成時に機能 IAM ロールを指定した場合、Amazon EKS により、このアクセスポリシーに従ってそのロールへのアクセスエントリが自動的に作成されます。

Kubernetes API グループ	Kubernetes resources	Kubernetes 動詞 (許可)
	namespaces	get, watch, list
services.k8s.aws , acm.services.k8s.aws , acmpca.services.k8s.aws , apigateway.services.k8s.aws , apigatewayv2.services.k8s.aws , applicationautoscaling.services.k8s.aws , athena.services.k8s.aws , bedrock.services.k8s.aws , bedrockagent.services.k8s.aws , bedrockagentcorecontrol.services.k8s.aws , cloudfront.services.k8s.aws , cloudtrail.services.k8s.aws , cloudwatch.services.k8s.aws , cloudwatchlogs.services.k8s.aws , codeartifact.services.k8s.aws , cognitoidentityprovider.services.k8s.aws , documentdb.service	*	*

Kubernetes API グループ	Kubernetes resources	Kubernetes 動詞 (許可)
s.k8s.aws , dynamodb.services.k8s.aws , ec2.services.k8s.aws , ecr.services.k8s.aws , ecrpublic.services.k8s.aws , ecs.services.k8s.aws , efs.services.k8s.aws , eks.services.k8s.aws , elasticache.services.k8s.aws , elbv2.services.k8s.aws , emrcontainers.services.k8s.aws , eventbridge.services.k8s.aws , iam.services.k8s.aws , kafka.services.k8s.aws , keyspaces.services.k8s.aws , kinesis.services.k8s.aws , kms.services.k8s.aws , lambda.services.k8s.aws , memorydb.services.k8s.aws , mq.services.k8s.aws , networkfirewall.services.k8s.aws , opensearchservice.services.k8s.aws , organizations.services.k8s.aws , pipes.services.k8s		

Kubernetes API グループ	Kubernetes resources	Kubernetes 動詞 (許可)
.aws , prometheusservice.services.k8s.aws , ram.services.k8s.aws , rds.services.k8s.aws , recyclebin.services.k8s.aws , route53.services.k8s.aws , route53resolver.services.k8s.aws , s3.services.k8s.aws , s3control.services.k8s.aws , sagemaker.services.k8s.aws , secretsmanager.services.k8s.aws , ses.services.k8s.aws , sfn.services.k8s.aws , sns.services.k8s.aws , sqs.services.k8s.aws , ssm.services.k8s.aws , wafv2.services.k8s.aws		
coordination.k8s.io	leases	create, get, list, watch
coordination.k8s.io	leases (特定の ACK サービスコントローラーリース専用)	delete, update, patch
	events	create, patch
apiextensions.k8s.io	customresourcedefinitions	*

AmazonEKSArgoCDClusterPolicy

ARN — `arn:aws:eks::aws:cluster-access-policy/AmazonEKSArgoCDClusterPolicy`

このポリシーは、Argo CD 機能がリソースを検出し、クラスター範囲のオブジェクトを管理するために必要なクラスターレベルのアクセス許可を付与します。 ポリシーには以下のアクセス許可が含まれています。

名前空間の管理:

- アプリケーションの名前空間を管理するための名前空間を作成、読み取り、更新、および削除します。

カスタムリソース定義の管理:

- Argo CD 固有の CRD (Application、AppProject、ApplicationSet) を管理します。

API 検出:

- リソースを検出するために Kubernetes API エンドポイントに読み取りアクセスします。

このポリシーは、名前空間の管理や CRD のインストールなど、クラスターレベルの Argo CD オペレーションをサポートするように設計されています。Argo CD 機能の作成時に機能 IAM ロールを指定した場合、Amazon EKS により、このアクセスポリシーに従ってそのロールへのアクセスエントリが自動的に作成されます。

Kubernetes API グループ	Kubernetes nonResourceURL	Kubernetes resources	Kubernetes 動詞 (許可)
		`namespaces`	`create`, `get`, `update`, `patch`, `delete`
`apiextensions.k8s.io`		`customresourcedefinitions`	`create`
`apiextensions.k8s.io`		`customresourcedefi`	`get`, `update`, `patch`, `delete`

Kubernetes API グループ	Kubernetes nonResourceURL	Kubernetes resources	Kubernetes 動詞 (許可)
		nitions (Argo CD CRD のみ)	
	/api, /api/*, /apis, /apis/*		get

AmazonEKSArgoCDPolicy

ARN — arn:aws:eks::aws:cluster-access-policy/AmazonEKSArgoCDPolicy

このポリシーは、Argo CD 機能がアプリケーションをデプロイおよび管理するために必要な名前空間レベルのアクセス許可を付与します。 ポリシーには以下のアクセス許可が含まれています。

シークレットの管理:

- Git 認証情報用のシークレットとクラスターシークレットにフルアクセスします。

ConfigMap アクセス:

- ConfigMaps に読み取りアクセスして、サポート対象外の Argo CD ConfigMaps を使用しようとしたら警告を送信します。

イベント管理:

- アプリケーションライフサイクルを追跡するためのイベントを読み取りおよび作成します。

Argo CD リソース管理:

- Application、ApplicationSet、AppProject にフルアクセスします。
- Argo CD リソースのファイナライザーおよびステータスを管理します。

このポリシーは、アプリケーションのデプロイや管理など、名前空間レベルの Argo CD オペレーションをサポートするように設計されています。Argo CD 機能の作成時に機能 IAM ロールを指定した場合、Amazon EKS により、このアクセスポリシーに従ってそのロールへのアクセスエントリが自動的に作成され、その範囲が Argo CD 名前空間に限定されます。

Kubernetes API グループ	Kubernetes resources	Kubernetes 動詞 (許可)
	secrets	*
	configmaps	get, list, watch
	events	get, list, watch, patch, create
argoproj.io	applications , applications/finalizers , applications/status , applicationsets , applicationsets/finalizers , applicationsets/status , appprojects , appprojects/finalizers , appprojects/status	*

AmazonEKSKROPolicy

ARN — arn:aws:eks::aws:cluster-access-policy/AmazonEKSKROPolicy

このポリシーは、kro (Kube Resource Orchestrator) 機能がカスタム Kubernetes API を作成および管理するために必要なアクセス許可を付与します。 ポリシーには以下のアクセス許可が含まれています。

kro リソース管理:

- ResourceGraphDefinitions やカスタムリソースインスタンスを含めすべての kro リソースにフルアクセスします。

カスタムリソース定義の管理:

- ResourceGraphDefinitions に定義されたカスタム API の CRD を作成、読み取り、更新、および削除します。

リーダー選出:

- リーダーを選出するために調整リースを作成および読み取ります。
- kro コントローラーリースを更新および削除します。

イベント管理:

- kro のオペレーション用にイベントを作成してパッチを適用します。

このポリシーは、kro を介した包括的なリソース構成とカスタム API 管理をサポートするように設計されています。kro 機能の作成時に機能 IAM ロールを指定した場合、Amazon EKS により、このアクセスポリシーに従ってそのロールへのアクセスエントリが自動的に作成されます。

Kubernetes API グループ	Kubernetes resources	Kubernetes 動詞 (許可)
`kro.run`	`*`	`*`
`apiextensions.k8s.io`	`customresourcedefinitions`	`*`
`coordination.k8s.io`	`leases`	`create`, `get`, `list`, `watch`
`coordination.k8s.io`	`leases` (kro コントローラーリースのみ)	`delete`, `update`, `patch`
	`events`	`create`, `patch`

アクセスポリシーの更新

アクセスポリシーが導入されてからのアクセスポリシーへの更新に関する詳細を表示します。このページの変更に関する自動通知については、ドキュメント履歴 の RSS フィードを購読してください。

変更	説明	日付
EKS 機能用のポリシーを追加する	EKS 機能を管理できるように `AmazonEKSACKPolicy` 、`AmazonEKS`	2025 年 11 月 22 日

変更	説明	日付
	ArgoCDClusterPolicy 、AmazonEKSArgoCDPolicy 、AmazonEKSKROPolicy を発行します。	
AmazonEKSSecretReaderPolicy を追加します。	読み取り専用アクセス用の新しいポリシーをシークレットに追加する	2025 年 11 月 6 日
EKS クラスターインサイトのポリシーを追加する	AmazonEKSClusterInsightsPolicy を発行	2024 年 12 月 2 日
Amazon EKS Hybrid のポリシーを追加	AmazonEKSHybridPolicy を発行	2024 年 12 月 2 日
Amazon EKS Auto Mode のポリシーを追加	このアクセスポリシーは、クラスター IAM ロールとノード IAM ロールに Kubernetes API を呼び出すアクセス許可を付与します。AWS ではこれらを使用して、ストレージ、コンピューティング、およびネットワーキングリソースのルーチンタスクを自動化します。	2024 年 12 月 2 日
AmazonEKSAdminViewPolicy を追加します。	Secrets などのリソースを含む、拡張ビューアクセス用の新しいポリシーを追加します。	2024 年 4 月 23 日
アクセスポリシーが導入されました。	Amazon EKS はアクセスポリシーを導入しました。	2023 年 5 月 29 日

アクセスエントリを使用するように認証モードを変更する

アクセスエントリを使い始めるにはクラスターの認証モードを API_AND_CONFIG_MAP または API モードに変更する必要があります。これにより、アクセスエントリ用の API が追加されます。

AWS コンソール

1. Amazon EKS コンソールを開きます。
2. アクセスエントリを作成するクラスターの名前を選択してください。
3. [リモートアクセス] タブを選択してください。
4. [認証モード] にはクラスターの現在の認証モードが表示されます。モードが EKS API と表示されている場合は既にアクセスエントリを追加でき、残りの手順は省略できます。
5. [アクセスの管理] を選択してください。
6. クラスター認証モードの場合は、EKS API でモードを選択してください。認証モードを、EKS API およびアクセスエントリを削除するモードに戻すことはできないことに注意してください。
7. [Save changes] (変更の保存) をクリックします。Amazon EKS がクラスターの更新を開始し、クラスターのステータスが「更新中」に変わり、変更が [更新履歴] タブに記録されます。
8. クラスターのステータスが [アクティブ] に戻るまで待ちます。クラスターが [アクティブ] になったら、「the section called “アクセスエントリを作成する”」の手順に従って IAM プリンシパルのクラスターへのアクセスを追加できます。

AWS CLI

1. 「AWS コマンドラインインターフェイスユーザーガイド」の「インストール」の説明に従って、AWS CLI をインストールします。
2. 以下のコマンドを実行してください。`#######` の部分は自分のクラスター名に置き換えます。ConfigMap メソッドを完全に無効にする場合はAPI_AND_CONFIG_MAP を API に置き換えてください。

   Amazon EKS がクラスターの更新を開始し、クラスターのステータスが [UPDATING] に変わり、変更が aws eks list-updates に記録されます。

```
aws eks update-cluster-config --name my-cluster --access-config
 authenticationMode=API_AND_CONFIG_MAP
```

3. クラスターのステータスが [アクティブ] に戻るまで待ちます。クラスターが [アクティブ] になったら、「the section called “アクセスエントリを作成する”」の手順に従って IAM プリンシパルのクラスターへのアクセスを追加できます。

必要なプラットフォームバージョン

アクセスエントリを使用するには、クラスターのプラットフォームバージョンが、次の表に記載されているバージョンと同じかそれ以降、または表に記載されている Kubernetes バージョンよりも新しいバージョンである必要があります。Kubernetes バージョンがリストにない場合は、すべてのプラットフォームバージョンでアクセスエントリがサポートされます。

Kubernetes バージョン	プラットフォームバージョン
リストに記載なし	すべてサポート
1.30	eks.2
1.29	eks.1
1.28	eks.6

詳細については、「platform-versions」を参照してください。

アクセスエントリを作成する

アクセスエントリを作成する前に、次の点を考慮してください:

- 適切に設定された認証モード。「the section called “[Authentication mode] (認証モード)”」を参照してください。
- アクセスエントリには1 つだけの既存の IAM プリンシパルの Amazon リソースネーム (ARN) が含まれます。IAM プリンシパルは複数のアクセスエントリに含めることはできません。指定する ARN に関するその他の考慮事項
  - IAM のベストプラクティスでは長期認証情報を持つ IAM ユーザーではなく、短期認証情報を持つ IAM ロールを使用してクラスターにアクセスすることを推奨しています。詳細は「IAM ユーザーガイド」の「人間のユーザーが一時的な認証情報を使用して AWS にアクセスするにはID プロバイダーとのフェデレーションの使用が必要です」を参照してください。

  - ARN が IAM ロール用の場合はパスを含めることができます。aws-auth ConfigMapエントリの ARN には、パスを含めることができません。例えば、ARN は arn:aws:iam::<111122223333>:role/<development/apps/my-role> または arn:aws:iam::<111122223333>:role/<my-role> にすることができます。
  - アクセスエントリのタイプが STANDARD 以外の場合 (タイプに関する次の考慮事項を参照)、ARN はクラスターと同じ AWS アカウントにある必要があります。タイプが STANDARD の場合、ARN はクラスターが属するアカウントと同じ、または異なる AWS アカウントでもかまいません。
  - アクセスエントリの作成後に IAM プリンシパルを変更することはできません。
  - この ARN を含む IAM プリンシパルを削除しても、アクセスエントリは自動的に削除されません。削除する IAM プリンシパルの ARN を含むアクセスエントリを削除することをお勧めします。アクセスエントリを削除せず、IAM プリンシパルを再作成すると、同じ ARN があっても、アクセスエントリは機能しません。これは再作成された IAM プリンシパルの ARN は同じですが、再作成された IAM プリンシパルの roleID または userID (aws sts get-caller-identity AWS CLI コマンドで確認できます) は元の IAM プリンシパルの ARN とは異なるためです。アクセスエントリの IAM プリンシパルの roleID または userID が表示されない場合でも、Amazon EKS はそれをアクセスエントリとともに保存します。
- 各アクセスエントリにはタイプがあります。アクセスエントリの種類は関連付けられているリソースの種類によって異なり、アクセス許可を定義しません。タイプを指定しない場合、Amazon EKS は自動的にタイプを STANDARD に設定します。
  - EC2_LINUX - Linux または Bottlerocket セルフマネージド型ノードで使用される IAM ロール用
  - EC2_WINDOWS - Windows セルフマネージド型ノードで使用される IAM ロール用
  - FARGATE_LINUX - AWS Fargate (Fargate) で使用される IAM ロール用
  - HYBRID_LINUX - ハイブリッドノードで使用される IAM ロール用
  - STANDARD - 指定しない場合のデフォルトタイプ
  - EC2 - EKS Auto Mode カスタムノードクラス用。詳細については、「the section called “ノードクラスのアクセスエントリを作成する”」を参照してください。
  - アクセスエントリの作成後にタイプを変更することはできません。
- マネージドノードグループまたは Fargate プロファイルに使用される IAM ロールのアクセスエントリを作成する必要はありません。EKS はアクセス エントリを作成するか (有効な場合)、認証設定マップを更新します (アクセス エントリが利用できない場合)。
- アクセスエントリのタイプが STANDARD の場合、そのアクセスエントリにユーザー名を指定できます。ユーザー名の値を指定しない場合、Amazon EKS はアクセスエントリのタイプと、指定した IAM プリンシパルが IAM ロールか IAM ユーザーかに応じて、以下のいずれかの値を設定しま

す。独自のユーザー名を指定する特別な理由がない限り、ユーザー名を指定せず、Amazon EKS に自動生成させることをお勧めします。独自のユーザー名を指定する場合:

- `system:`、`eks:`、`aws:`、`amazon:`、または `iam:` で始めることはできません。
- ユーザー名が IAM ロール用の場合は、ユーザー名の末尾に `{{SessionName}}` または `{{SessionNameRaw}}` を追加することをお勧めします。ユーザー名に `{{SessionName}}` または `{{SessionNameRaw}}` を追加する場合、ユーザー名の {{SessionName}} の前にコロンを含める必要があります。このロールを引き受けると、ロールを引き受けるときに指定された AWS STS セッションの名前が自動的にクラスターに渡され、CloudTrail ログに表示されます。例えば、`john{{SessionName}}` というユーザー名にはできません。ユーザー名は `:john{{SessionName}}` または `jo:hn{{SessionName}}` でなければなりません。コロンは `{{SessionName}}` の前になければいけません。次の表の Amazon EKS によって生成されたユーザー名には ARN が含まれています。ARN にはコロンが含まれているため、この要件を満たしています。ユーザー名に `{{SessionName}}` を含めなければ、コロンは不要です。`{{SessionName}}` の場合、セッション名の特殊文字「@」は「-」に置き換えられます。`{{SessionNameRaw}}` の場合、セッション名の特殊文字はすべて保持されます。

IAM プリンシパルタイプ	タイプ	Amazon EKS が自動的に設定するユーザー名の値
ユーザー	`STANDARD`	ユーザーの ARN。例: `arn:aws:iam::<111122223333>:user/<my-user>`

IAM プリンシパルタイプ	タイプ	Amazon EKS が自動的に設定するユーザー名の値
ロール	STANDARD	想定されるロールの STS ARN。Amazon EKS では{{SessionName}} がロールに追加されます。 例: arn:aws:sts::<111122223333>:assumed-role/<my-role>/{{SessionName}} 指定したロールの ARN にパスが含まれている場合、Amazon EKS は生成されたユーザー名からそのパスを削除してください。
ロール	EC2_LINUX -または-EC2_WINDOWS	system:node:{{EC2PrivateDNSName}}
ロール	FARGATE_LINUX	system:node:{{SessionName}}
ロール	HYBRID_LINUX	system:node:{{SessionName}}

アクセスエントリの作成後にユーザー名を変更することができます。

- アクセスエントリのタイプが STANDARD で、Kubernetes RBAC 認可を使用する場合はアクセスエントリに 1 つ以上のグループ名を追加できます。アクセスエントリを作成したら、グループ名を追加および削除できます。IAM プリンシパルがクラスター上の Kubernetes オブジェクトにアクセスできるようにするには Kubernetes ロールベース認可 (RBAC) オブジェクトを作成して管理する必要があります。クラスター上で Kubernetes RoleBinding または ClusterRoleBinding オブジェクトを作成し、kind: Group の subject としてグループ名を指定します。Kubernetes は、Kubernetes Role で指定したクラスターオブジェクト、またはバインディングの roleRef で

も指定した ClusterRole オブジェクトへの IAM プリンシパルアクセスを許可します。グループ名を指定する場合は、Kubernetes ロールベース認可 (RBAC) オブジェクトについて理解しておくことをお勧めします。詳細についてはKubernetes ドキュメントの「RBAC 認可を使用する」を参照してください。

> **Important**
>
> Amazon EKS はクラスターに存在する Kubernetes RBAC オブジェクトに、指定したグループ名が含まれていることを確認しません。例えば、現在存在しないグループのアクセスエントリを作成すると、Amazon EKS はエラーを返すことなく設定を受け入れますが、一致する Kubernetes RBAC リソースが作成されるまで IAM プリンシパルにはアクセス許可がありません。

クラスター上の Kubernetes オブジェクトに IAM プリンシパルがアクセスするのを許可する Kubernetes の代わりに、またはそれに加えて、Amazon EKS アクセスポリシーをアクセスエントリに関連付けることができます。Amazon EKS はIAM プリンシパルがアクセスポリシーのアクセス許可を使用してクラスター上の Kubernetes オブジェクトにアクセスすることを許可します。アクセスポリシーのアクセス許可の範囲は指定した Kubernetes 名前空間に限定できます。アクセスポリシーを使用する場合、Kubernetes RBAC オブジェクトを管理する必要はありません。詳細については、「the section called “アソシエイトアクセスポリシー”」を参照してください。

- タイプが EC2_LINUX または EC2_WINDOWS のアクセスエントリを作成する場合、アクセスエントリを作成する IAM プリンシパルには iam:PassRole アクセス許可が必要です。詳細については、IAM ユーザーガイドの「AWS サービスにロールを渡す許可をユーザーに付与する」を参照してください。
- 標準的な IAM 動作と同様に、アクセスエントリの作成と更新は最終的には一貫性があり、最初の API コールが正常に戻ってから有効になるまでに数秒かかる場合があります。発生する可能性のあるこれらの遅延を考慮して、アプリケーションを設計する必要があります。アプリケーションの重要で高可用性のコードパスにはアクセスエントリの作成や更新を含めないことをお勧めします。代わりに、実行頻度が低い別の初期化またはセットアップルーチンに の変更を加えます。また、本番稼働ワークフローが依存する前に、変更が伝達済みであることを確認します。
- アクセスエントリはサービスリンクロールをサポートしていません。プリンシパル ARN がサービスリンクロールである場合、アクセスエントリを作成することはできません。サービスリンクロールは arn:aws:iam::*:role/aws-service-role/* 形式の ARN によって識別できます。

AWS マネジメントコンソール または AWS CLI を使用してアクセスエントリを作成できます。

AWS マネジメントコンソール

1. Amazon EKS コンソールを開きます。
2. アクセスエントリを作成するクラスターの名前を選択してください。
3. [リモートアクセス] タブを選択してください。
4. [アクセスエントリの作成] を選択してください。
5. IAM プリンシパルには既存の IAM ロールまたはユーザーを選択してください。IAM のベストプラクティスでは長期認証情報を持つ IAM ユーザーではなく、短期認証情報を持つ IAM ロールを使用してクラスターにアクセスすることを推奨しています。詳細は「IAM ユーザーガイド」の「人間のユーザーが一時的な認証情報を使用して AWS にアクセスするにはID プロバイダーとのフェデレーションの使用が必要です」を参照してください。
6. [タイプ] で、アクセスエントリがセルフマネージド型の Amazon EC2 ノードに使用されるノードロール用のものである場合は[EC2 Linux] または [EC2 Windows] を選択してください。それ以外の場合はデフォルト (標準) をそのまま使用します。
7. 選択した タイプが標準で、ユーザー名を指定する場合はユーザー名を入力します。
8. 選択したタイプが標準で、IAM プリンシパルに Kubernetes RBAC 認可を使用したい場合はグループに 1 つ以上の名前を指定します。グループ名を指定せず、Amazon EKS 認可を使用する場合は後のステップで、またはアクセスエントリの作成後にアクセスポリシーを関連付けることができます。
9. (オプション) [タグ] ではアクセスエントリにラベルを割り当てます。例えば、同じタグを持つすべてのリソースを検索しやすくするためです。
10. [次へ] を選択します。
11. [アクセスポリシーを追加] ページで、選択したタイプが標準で、Amazon EKS が IAM プリンシパルにクラスター上の Kubernetes オブジェクトへのアクセス許可を与えることを許可する場合は次のステップを実行してください。それ以外の場合は[次へ] を選択してください。
    a. [ポリシー名] にはアクセスポリシーを選択してください。アクセスポリシーのアクセス許可は表示できませんが、Kubernetes ユーザー向け `ClusterRole` オブジェクトと同様のアクセス許可が含まれています。詳細については、Kubernetes ドキュメントの「ユーザー向け Role」を参照してください。
    b. 以下のオプションのいずれかを選択してください。
        - クラスター – Amazon EKS が IAM プリンシパルに、クラスター上のすべての Kubernetes オブジェクトのアクセスポリシー内のアクセス許可があるように承認する場合はこのオプションを選択してください。

- Kubernetes 名前空間 – Amazon EKS が IAM プリンシパルに、クラスター上の特定の Kubernetes 名前空間のすべての Kubernetes オブジェクトのアクセスポリシー内のアクセス許可があるように承認する場合はこのオプションを選択してください。[名前空間] にはクラスター上の Kubernetes 名前空間の名前を入力します。名前空間をさらに追加する場合は[新しい名前空間を追加する] を選択し、名前空間名を入力します。

c. ポリシーを追加するには[ポリシーを追加] を選択してください。ポリシーごとに異なる範囲を設定できますが、各ポリシーは 1 回しか追加できません。

d. [次へ] を選択してください。

12.アクセスエントリの設定を確認してください。何かが正しくないと思われる場合は[戻る] を選択してステップに戻り、エラーを修正します。設定が正しければ、[作成] を選択してください。

AWS CLI

1. AWS コマンドラインインターフェイスユーザーガイドの「インストール」の説明に従って、AWS CLI をインストールします。
2. アクセスエントリを作成するには以下のいずれかの例を使用してアクセスエントリを作成できます:

- セルフマネージド型 Amazon EC2 Linux ノードグループのアクセスエントリを作成します。*######* はクラスターの名前、*111122223333* は自分の AWS アカウント ID に、*EKS-######-self-managed-ng-1* はノード IAM ロールの名前に置き換えます。ノードグループが Windows ノードグループの場合は *EC2_LINUX* を EC2_WINDOWS に置き換えてください。

```
aws eks create-access-entry --cluster-name my-cluster --principal-arn
 arn:aws:iam::111122223333:role/EKS-my-cluster-self-managed-ng-1 --type EC2_LINUX
```

STANDARD 以外のタイプを指定する場合、--kubernetes-groups オプションは使用できません。タイプが STANDARD 以外の値であるため、このアクセスエントリにアクセスポリシーを関連付けることはできません。

- Amazon EC2 セルフマネージド型ノードグループには使用されていない IAM ロールを許可するアクセスエントリを作成し、Kubernetes にクラスターへのアクセスを許可してもらいます。*######* をクラスターの名前に、*111122223333* を AWS アカウント ID に、*my-role* を IAM ロールの名前に置き換えます。*Viewers* はクラスター上の Kubernetes RoleBinding または ClusterRoleBinding オブジェクトで指定したグループの名前に置き換えます。

```
aws eks create-access-entry --cluster-name my-cluster --principal-arn
 arn:aws:iam::111122223333:role/my-role --type STANDARD --username Viewers --
kubernetes-groups Viewers
```

- IAM ユーザーがクラスターに対して認証できるようにするアクセスエントリを作成します。これも可能であるためこの例が示されていますが、IAM のベストプラクティスでは長期認証情報を持つ IAM ユーザーではなく、短期認証情報を持つ IAM ロールを使用してクラスターにアクセスすることを推奨しています。詳細は「IAM ユーザーガイド」の「人間のユーザーが一時的な認証情報を使用して AWS にアクセスするにはID プロバイダーとのフェデレーションの使用が必要です」を参照してください。

  ```
  aws eks create-access-entry --cluster-name my-cluster --principal-arn
   arn:aws:iam::111122223333:user/my-user --type STANDARD --username my-user
  ```

  このユーザーに Kubernetes API ディスカバリーロールのアクセス許可よりも多くのクラスターへのアクセスを許可したい場合は `--kubernetes-groups` オプションが使用されていないため、アクセスエントリにアクセスポリシーを関連付ける必要があります。詳細については、「the section called “アソシエイトアクセスポリシー”」と Kubernetes ドキュメントの「API ディスカバリーロール」を参照してください。

アクセスエントリを更新する

AWS マネジメントコンソール または AWS を使用してアクセスエントリを更新できます。

AWS マネジメントコンソール

1. アマゾン EKS コンソールを開きます。
2. アクセスエントリを作成するクラスターの名前を選択してください。
3. [リモートアクセス] タブを選択してください。
4. 更新するアクセスエントリを選択します。
5. [編集] を選択します。
6. [ユーザー名] では、既存の値を変更できます。
7. [グループ]では、既存のグループ名を削除したり、新しいグループ名を追加したりできます。system:nodes または system bootstrappers というグループ名が存在する場合は、削除しないでください。これらのグループを削除すると、クラスターが正しく機能しなくなる可能性があり

ます。グループ名を指定せず、Amazon EKS 認可を使用する場合は、後のステップで、アクセスポリシーを関連付けます。

8. [タグ] では、アクセスエントリにラベルを割り当てます。例えば、同じタグを持つすべてのリソースを検索しやすくするためです。また、既存のタグを削除することもできます。
9. [Save changes] (変更の保存) をクリックします。
10. アクセスポリシーをエントリに関連付ける場合は、the section called “アソシエイトアクセスポリシー” を参照してください。

AWS CLI

1. AWS コマンドラインインターフェイスユーザーガイドの「インストール」の説明に従って、AWS CLI をインストールします。
2. アクセスエントリを更新するには、*my-cluster* はクラスターの名前、*111122223333* は AWS アカウント ID に、*EKS-my-cluster-my-namespace-Viewers* は IAM ロールの名前に置き換えます。

```
aws eks update-access-entry --cluster-name my-cluster --principal-arn
 arn:aws:iam::111122223333:role/EKS-my-cluster-my-namespace-Viewers --kubernetes-
groups Viewers
```

アクセスエントリのタイプが STANDARD 以外の値の場合、--kubernetes-groups オプションは使用できません。また、アクセスポリシーを STANDARD 以外のタイプのアクセスエントリに関連付けることもできません。

アクセスエントリを削除する

アクセスエントリを誤って削除したことが判明した場合は、いつでも再作成できます。削除するアクセスエントリがアクセスポリシーに関連付けられている場合、その関連付けは自動的に削除されます。アクセスエントリを削除する前に、アクセスポリシーとアクセスエントリの関連付けを解除する必要はありません。

AWS マネジメントコンソール または AWS CLI を使用してアクセスエントリを削除できます。

AWS マネジメントコンソール

1. アマゾン EKS コンソールを開きます。
2. アクセスエントリを削除するクラスターの名前を選択します。

3. [リモートアクセス] タブを選択してください。
4. [アクセスエントリ] リストで、削除するアクセスエントリを選択します。
5. [Delete] を選択します。
6. 確認ダイアログボックスで、[削除] を選択します。

AWS CLI

1. AWS コマンドラインインターフェイスユーザーガイドの「インストール」の説明に従って、AWS CLI をインストールします。
2. アクセスエントリを削除するには、*my-cluster* をクラスターの名前、*111122223333* を AWS アカウント ID に、*my-role* をクラスターにアクセスさせたくない IAM ロールの名前に置き換えます。

```
aws eks delete-access-entry --cluster-name my-cluster --principal-arn
 arn:aws:iam::111122223333:role/my-role
```

EKS アクセスエントリ用のカスタムユーザー名を設定する

Amazon EKS 用のアクセスエントリを作成するときに、自動生成されたユーザー名を使用することも、カスタムユーザー名を指定することもできます。このページでは、両方のオプションについて説明し、カスタムユーザー名を設定する手順を示します。

概要:

アクセスエントリにユーザー名を使用すると、Kubernetes のログと監査証跡で IAM プリンシパルを識別できます。デフォルトでは、Amazon EKS は IAM ID の ARN に基づいてユーザー名を生成しますが、必要に応じてカスタムユーザー名を指定することもできます。

デフォルトのユーザー名の生成

ユーザー名の値を指定しないと、Amazon EKS は IAM ID に基づいてユーザー名を自動的に生成します。

- IAM ユーザーの場合:
  - EKS は、Kubernetes ユーザー名を IAM ユーザーの ARN に設定します。
  - 例:

```
{arn-aws}iam::<111122223333>:user/<my-user>
```

- IAM ロールの場合:
  - EKS は、IAM ロールの ARN に基づいて Kubernetes ユーザー名を設定します。
  - 想定されるロールの STS ARN。Amazon EKS では{{SessionName}} がロールに追加されます。指定したロールの ARN にパスが含まれている場合、Amazon EKS は生成されたユーザー名からそのパスを削除してください。
  - 例:

```
{arn-aws}sts::<111122223333>:assumed-role/<my-role>/{{SessionName}}
```

独自のユーザー名を指定する特別な理由がない限り、ユーザー名を指定せず、Amazon EKS に自動生成させることをお勧めします。

カスタムユーザー名の設定

アクセスエントリを作成するときは、--username パラメータを使用して、カスタムユーザー名を指定できます。

```
aws eks create-access-entry --cluster-name <cluster-name> --principal-arn <iam-
identity-arn> --type STANDARD --username <custom-username>
```

カスタムユーザー名の要件

カスタムユーザー名を指定する場合:

- ユーザー名を system:、eks:、aws:、amazon:、または iam: で始めることはできません。
- ユーザー名が IAM ロール用の場合は、ユーザー名の末尾に {{SessionName}} または {{SessionNameRaw}} を追加することをお勧めします。
  - ユーザー名に {{SessionName}} または {{SessionNameRaw}} を追加する場合、ユーザー名の {{SessionName}} の前にコロンを含める必要があります。

アクセスポリシーと AWS CLI を使用して IAM ロールまたは IAM ユーザーのアクセスエントリを作成する

Amazon EKS アクセスエントリを作成すると、AWS マネージド EKS アクセスポリシーを使用して、Kubernetes クラスターにアクセスおよび管理するための標準化されたアクセス許可を IAM ID に付与できるようになります。

概要

Amazon EKS のアクセスエントリでは、IAM ID (ユーザーとロール) が Kubernetes クラスターにアクセスしてやり取りする方法を定義します。EKS アクセスポリシーを使用してアクセスエントリを作成することで、次のことができます。

- EKS クラスターへのアクセス許可を特定の IAM ユーザーまたは IAM ロールに付与する
- AWS マネージド EKS アクセスポリシーでアクセス許可セットを標準化および事前定義してアクセス許可を制御する
- アクセス許可のスコープを特定の名前空間またはクラスター全体に指定する
- `aws-auth` ConfigMap の変更も Kubernetes RBAC リソースの作成も不要にして、アクセス管理を簡素化する
- AWS 統合アプローチを使用した Kubernetes アクセスコントロールにより、よくあるユースケースに対応すると共に、セキュリティベストプラクティスを維持する

AWS マネージドの標準化されたアクセス許可を実現し、Kubernetes RBAC を手動で設定する必要がないため、ほとんどのユースケースにお勧めのアプローチです。EKS アクセスポリシーにより、Kubernetes RBAC リソースを手動で設定する必要がなくなり、よくあるユースケースに対応したアクセス許可セットを事前定義できます。

前提条件

- アクセスエントリを有効にするようにクラスターの認証モードを設定する必要があります。詳細については、「the section called “[Authentication mode] (認証モード)”」を参照してください。
- AWS コマンドラインインターフェイスユーザーガイドの「インストール」の説明に従って、AWS CLI をインストールして設定します。

ステップ 1: アクセスエントリを定義する

1. アクセス許可を付与する IAM ID (ユーザーやロールなど) の ARN を確認します。

- IAM ID ごとに EKS アクセスエントリを 1 つのみ含めることができます。

2. Amazon EKS アクセスポリシーアクセス許可を特定の Kubernetes 名前空間にのみ適用するか、クラスター全体に適用するかを決定します。
   - アクセス許可を特定の名前空間に制限する場合は、その名前空間の名前を書き留めます。
3. IAM ID に使用する EKS アクセスポリシーを選択します。このポリシーでは、クラスター内アクセス許可を付与します。ポリシーの ARN を書き留めます。
   - ポリシーのリストについては、「available access policies」を参照してください。
4. 自動生成したユーザー名がアクセスエントリに適しているかどうかや、ユーザー名を手動で指定する必要があるかどうかを判断します。
   - AWS は、IAM ID に基づいてこの値を自動生成します。カスタムのユーザー名を設定できます。この名前が Kubernetes ログに表示されます。
   - 詳細については、「the section called “カスタムユーザー名を設定する”」を参照してください。

ステップ 2: アクセスエントリを作成する

アクセスエントリを作成する準備を整えたら、AWS CLI を使用して作成します。

以下に、よくあるユースケースの例を示します。すべての設定オプションについて CLI リファレンスを表示します。

次のステップでアクセスポリシーをアタッチします。

```
aws eks create-access-entry --cluster-name <cluster-name> --principal-arn <iam-
identity-arn> --type STANDARD
```

ステップ 3: アクセスポリシーを関連付ける

使用するコマンドは、ポリシーを指定の Kubernetes 名前空間に制限するかどうかによって異なります。

アクセスポリシーの ARN が必要になります。利用可能なアクセスポリシーをレビューします。

名前空間スコープを指定せずにポリシーを作成する

```
aws eks associate-access-policy --cluster-name <cluster-name> --principal-arn <iam-
identity-arn> --access-scope type=cluster --policy-arn <access-policy-arn>
```

名前空間スコープを指定してポリシーを作成する

```
aws eks associate-access-policy --cluster-name <cluster-name> --principal-arn <iam-
identity-arn> \
    --access-scope type=namespace,namespaces=my-namespace1,my-namespace2 --policy-arn
 <access-policy-arn>
```

次のステップ

- kubectl を IAM ID と共に使用できるように kubeconfig を作成する

AWS CLI で Kubernetes グループを使用してアクセスエントリを作成する

作成する Amazon EKS アクセスエントリは、認可に Kubernetes グループを使用し、RBAC を手動で設定する必要があるものとします。

> **Note**
>
> 通常は、このページで説明している Kubernetes グループによるアプローチではなく、EKS アクセスポリシーを使用する方法をお勧めします。EKS アクセスポリシーでは、AWS との統合を進めたシンプルな方法でアクセスを管理するため、RBAC を手動で設定する必要がありません。Kubernetes グループによるアプローチを使用するのは、EKS アクセスポリシーよりもさらにきめ細かく制御する必要がある場合だけにしてください。

概要

アクセスエントリでは、IAM ID (ユーザーとロール) が Kubernetes クラスターにアクセスする方法を定義します。Kubernetes グループによるアプローチでは、標準の Kubernetes RBAC グループを介して EKS クラスターにアクセスするためのアクセス許可を IAM ユーザーまたは IAM ロールに付与します。この方法では、Kubernetes RBAC リソース (Roles、RoleBindings、ClusterRoles、ClusterRoleBindings) を作成および管理する必要があります。高度にカスタマイズされたアクセス許可セットが必要なときや複雑な認可要件が必要なとき、あるいはハイブリッド Kubernetes 環境全体にわたって一貫したアクセスコントロールパターンを維持するときに、お勧めの方法です。

このトピックでは、Amazon EC2 インスタンスが EKS クラスターに参加する際に使用する IAM ID 用のアクセスエントリの作成については取り上げません。

前提条件

- アクセスエントリを有効にするようにクラスターの認証モードを設定する必要があります。詳細については、「the section called “[Authentication mode] (認証モード)”」を参照してください。
- AWS コマンドラインインターフェイスユーザーガイドの「インストール」の説明に従って、AWS CLI をインストールして設定します。
- Kubernetes RBAC をよく理解しておくことをお勧めします。詳細についてはKubernetes ドキュメントの「RBAC 認可を使用する」を参照してください。

ステップ 1: アクセスエントリを定義する

1. アクセス許可を付与する IAM ID (ユーザーやロールなど) の ARN を確認します。
   - IAM ID ごとに EKS アクセスエントリを 1 つのみ含めることができます。
2. この IAM ID に関連付ける Kubernetes グループを決定します。
   - そのグループを参照する Kubernetes `Role`/`ClusterRole` と `RoleBinding`/`ClusterRoleBinding` リソースを作成するか、既存のものを使用する必要があります。
3. 自動生成したユーザー名がアクセスエントリに適しているかどうかや、ユーザー名を手動で指定する必要があるかどうかを判断します。
   - AWS は、IAM ID に基づいてこの値を自動生成します。カスタムのユーザー名を設定できます。この名前が Kubernetes ログに表示されます。
   - 詳細については、「the section called “カスタムユーザー名を設定する”」を参照してください。

ステップ2: Kubernetes グループと共にアクセスエントリを作成する

アクセスエントリを作成する準備を整えたら、AWS CLI を使用して適切な Kubernetes グループと共に作成します。

```
aws eks create-access-entry --cluster-name <cluster-name> --principal-arn <iam-
identity-arn> --type STANDARD --kubernetes-groups <groups>
```

置換:

- `<cluster-name>` を EKS クラスター名に置き換えます。
- `<iam-identity-arn>` を IAM ユーザーまたはロールの ARN に置き換えます。

- <groups> を Kubernetes グループのカンマ区切りリスト (例えば「system:developers,system:readers」)に置き換えます。

すべての設定オプションについて CLI リファレンスを表示します。

ステップ 3: Kubernetes RBAC を設定する

IAM プリンシパルがクラスター上の Kubernetes オブジェクトにアクセスできるようにするには、Kubernetes ロールベースアクセスコントロール (RBAC) オブジェクトを作成して管理する必要があります。

1. アクセス許可を定義する Kubernetes オブジェクト Role または ClusterRole を作成します。
2. クラスター上で Kubernetes RoleBinding または ClusterRoleBinding オブジェクトを作成し、kind: Group の subject としてグループ名を指定します。

Kubernetes でグループとアクセス許可を設定する方法の詳細については、Kubernetes のドキュメントの「Using RBAC Authorization」を参照してください。

次のステップ

- kubectl を IAM ID と共に使用できるように kubeconfig を作成する

ConfigMap を使用して IAM ユーザーに Kubernetes へのアクセスを許可する

> **Important**
>
> aws-auth ConfigMap は非推奨になりました。Kubernetes API へのアクセスを管理する推奨手段については、「the section called “アクセスエントリ”」を参照してください。

IAM プリンシパルを使用してクラスターにアクセスします。プリンシパルは、Amazon EKS コントロールプレーンで実行される AWS IAM Authenticator for Kubernetes によって有効になります。オーセンティケーターは、その設定情報を aws-auth ConfigMap から取得します。すべての aws-auth ConfigMap 設定については、GitHub の「Full Configuration Format」を参照してください。

Amazon EKS クラスターに IAM プリンシパルを追加する

Amazon EKS クラスターを作成すると、このクラスターを作成する IAM プリンシパルには、Amazon EKS コントロールプレーンのクラスターロールベースアクセスコントロール (RBAC) 設定で、`system:masters` 許可が自動的に付与されます。このプリンシパルは表示可能な設定の中には表示されません。したがって、どのプリンシパルが最初にクラスターを作成したのかを追跡する必要があります。追加の IAM プリンシパルがクラスターとインタラクションできるようにするには Kubernetes 内の `aws-auth ConfigMap` を編集し、`aws-auth ConfigMap` で指定した `group` の名前を使用して、Kubernetes `rolebinding` または `clusterrolebinding` を作成します。

> **Note**
>
> Kubernetes のロールベースアクセスコントロール (RBAC) 設定の詳細については、Kubernetes ドキュメントの「RBAC 認可を使用する」を参照してください。

1. クラスターへのアクセスに `kubectl` が使用している認証情報を判別します。使用中のコンピュータ上で、次のコマンドを使用して、`kubectl` が使用する認証情報を判別できます。デフォルトのパスを使用しない場合は、*`~/.kube/config`* を `kubeconfig` ファイルへのパスに置き換えます。

   ```
   cat ~/.kube/config
   ```

   出力例は次のとおりです。

   ```
   [...]
   contexts:
   - context:
       cluster: my-cluster.region-code.eksctl.io
       user: admin@my-cluster.region-code.eksctl.io
     name: admin@my-cluster.region-code.eksctl.io
   current-context: admin@my-cluster.region-code.eksctl.io
   [...]
   ```

   前述の出力例では、*`admin`* という名前のユーザーの認証情報が *`my-cluster`* という名前のクラスター用に設定されています。このユーザーがクラスターを作成したユーザーである場合は、既にそのクラスターにアクセスできます。クラスターを作成したユーザーでない場合は、以降の手順を実行して、他の IAM プリンシパルのクラスターへのアクセスを有効にする必要があります。IAM のベストプラクティスでは、ユーザーではなくロールに許可を付与することが推奨され

ています。次のコマンドを使用すると、現在クラスターへのアクセスが可能な、他のプリンシパルを確認できます。

```
kubectl describe -n kube-system configmap/aws-auth
```

出力例は次のとおりです。

```
Name:         aws-auth
Namespace:    kube-system
Labels:       <none>
Annotations:  <none>

Data
====
mapRoles:
----
- groups:
  - system:bootstrappers
  - system:nodes
  rolearn: arn:aws:iam::111122223333:role/my-node-role
  username: system:node:{{EC2PrivateDNSName}}


BinaryData
====

Events:  <none>
```

前の例は、デフォルトの aws-auth ConfigMap です。ノードインスタンスのロールだけがクラスターにアクセスできます。

2. IAM プリンシパルをマッピングできる既存の Kubernetes roles と rolebindings または clusterroles と clusterrolebindings があることを確認してください。これらのリソースの詳細については、Kubernetes ドキュメントの「RBAC 認可を使用する」を参照してください。

   a. 既存の Kubernetes の roles または clusterroles を表示します。Roles は namespace スコープされ、clusterroles はクラスターにスコープされます。

   ```
   kubectl get roles -A
   ```

   ```
   kubectl get clusterroles
   ```

b. 前の出力で返された role または clusterrole の詳細を表示します。クラスター内で IAM プリンシパルに必要となる許可 (rules) があることを確認します。

前のコマンド出力で返された *role-name* を、role の名前に置き換えます。*kube-system* を、role の名前空間に置き換えます。

```
kubectl describe role role-name -n kube-system
```

cluster-role-name を、前のコマンドの出力で返された clusterrole 名に置き換えます。

```
kubectl describe clusterrole cluster-role-name
```

c. 既存の Kubernetes の rolebindings または clusterrolebindings を表示します。Rolebindings は namespace スコープされ、clusterrolebindings はクラスターにスコープされます。

```
kubectl get rolebindings -A
```

```
kubectl get clusterrolebindings
```

d. rolebinding または clusterrolebinding の詳細を表示して、前のステップで roleRef としてリストされた role または clusterrole、そして subjects にリストされたグループ名があることを確認します。

role-binding-name を、前のコマンドの出力で返された rolebinding の名前に置き換えます。*kube-system* は、rolebinding の namespace で置き換えます。

```
kubectl describe rolebinding role-binding-name -n kube-system
```

出力例は次のとおりです。

```
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: eks-console-dashboard-restricted-access-role-binding
  namespace: default
subjects:
- kind: Group
```

```
  name: eks-console-dashboard-restricted-access-group
  apiGroup: rbac.authorization.k8s.io
roleRef:
  kind: Role
  name: eks-console-dashboard-restricted-access-role
  apiGroup: rbac.authorization.k8s.io
```

cluster-role-binding-name は、前のコマンドの出力で返された clusterrolebinding の名前で置き換えます。

```
kubectl describe clusterrolebinding cluster-role-binding-name
```

出力例は次のとおりです。

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: eks-console-dashboard-full-access-binding
subjects:
- kind: Group
  name: eks-console-dashboard-full-access-group
  apiGroup: rbac.authorization.k8s.io
roleRef:
  kind: ClusterRole
  name: eks-console-dashboard-full-access-clusterrole
  apiGroup: rbac.authorization.k8s.io
```

3. aws-auth ConfigMap を編集します。eksctl のようなツールにより ConfigMap を更新することができます。あるいは、手動で編集しての更新も可能です。

> **Important**
>
> ConfigMap の編集には、eksctl や、その他のツールを使用することをお勧めします。使用できるその他のツールについては、Amazon EKS ベストプラクティスガイドの「ツールを使用して aws-auth ConfigMap を変更する」を参照してください。aws-auth ConfigMap の形式が不適切な場合、クラスターへのアクセスが失われる場合があります。

- eksctl で configmap を編集するステップを表示します。

- configmap を手動で編集するステップを表示します。

eksctl で ConfigMap を編集する

1. デバイスまたは AWS CloudShell にインストールされている eksctl コマンドラインツールのバージョン 0.215.0 以降が必要です。eksctl をインストールまたはアップグレードするには、eksctl ドキュメントの「インストール」を参照してください。
2. ConfigMap に現在のマッピングを表示します。*my-cluster* の部分は自分のクラスター名に置き換えます。*region-code* を、クラスターのある AWS リージョンに置き換えます。

```
eksctl get iamidentitymapping --cluster my-cluster --region=region-code
```

出力例は次のとおりです。

```
ARN
                USERNAME                                GROUPS
 ACCOUNT
arn:aws:iam::111122223333:role/eksctl-my-cluster-my-nodegroup-
NodeInstanceRole-1XLS7754U3ZPA    system:node:{{EC2PrivateDNSName}}
 system:bootstrappers,system:nodes
```

3. ロールのマッピングを追加します。*my-role* を実際のロールの名前に置き換えます。*eks-console-dashboard-full-access-group* を、Kubernetes の RoleBinding または ClusterRoleBinding オブジェクトで指定されたグループの名前に置き換えます。*111122223333* は、ご自身のアカウント ID に置き換えます。*###*を任意の名前に置き換えることができます。

```
eksctl create iamidentitymapping --cluster my-cluster --region=region-code \
    --arn arn:aws:iam::111122223333:role/my-role --username admin --group eks-
console-dashboard-full-access-group \
    --no-duplicate-arns
```

> **Important**
>
> ロールの ARN には、role/my-team/developers/my-role などのパスを含めることはできません。ロールの ARN は、 arn:aws:iam::*111122223333*:role/*my-role*

のような形式にする必要があります。この例では、my-team/developers/ を削除する必要があります。

出力例は次のとおりです。

```
[...]
2022-05-09 14:51:20 [#]  adding identity "{arn-aws}iam::111122223333:role/my-role" to
 auth ConfigMap
```

4. ユーザーのマッピングを追加します。IAM のベストプラクティスでは、ユーザーではなくロールに許可を付与することが推奨されています。*my-user* を、ご自身のユーザー名に置き換えます。*eks-console-dashboard-restricted-access-group* を、Kubernetes RoleBinding または ClusterRoleBinding オブジェクトで指定されたグループの名前に置き換えます。*111122223333* は、ご自身のアカウント ID に置き換えます。*my-user* を任意の名前に置き換えることができます。

```
eksctl create iamidentitymapping --cluster my-cluster --region=region-code \
    --arn arn:aws:iam::111122223333:user/my-user --username my-user --group eks-
console-dashboard-restricted-access-group \
    --no-duplicate-arns
```

出力例は次のとおりです。

```
[...]
2022-05-09 14:53:48 [#]  adding identity "arn:aws:iam::111122223333:user/my-user" to
 auth ConfigMap
```

5. ConfigMap のマッピングを再度表示します。

```
eksctl get iamidentitymapping --cluster my-cluster --region=region-code
```

出力例は次のとおりです。

```
ARN
            USERNAME                                GROUPS
      ACCOUNT
arn:aws:iam::111122223333:role/eksctl-my-cluster-my-nodegroup-
NodeInstanceRole-1XLS7754U3ZPA    system:node:{{EC2PrivateDNSName}}
 system:bootstrappers,system:nodes
```

```
arn:aws:iam::111122223333:role/admin
          my-role                                     eks-console-dashboard-full-access-
group
arn:aws:iam::111122223333:user/my-user
           my-user                                     eks-console-dashboard-restricted-
access-group
```

ConfigMap を手動で編集する

1. 編集する ConfigMap を開きます。

```
kubectl edit -n kube-system configmap/aws-auth
```

> **Note**
>
> 「Error from server (NotFound): configmaps "aws-auth" not found」というエラーが表示された場合は、「aws-auth ConfigMap をクラスターに適用する」の手順を使用して、ストック ConfigMap を適用します。

2. IAM プリンシパルを ConfigMap に追加します。IAM グループは IAM プリンシパルではないため、ConfigMap に追加できません。
   - IAM ロールを (例えば、フェデレーティッドユーザーに) 追加するには、ConfigMap の mapRoles セクションの data の下にロールの詳細を追加します。ファイルに存在しない場合は、このセクションを追加します。各エントリは、次のパラメータをサポートしています。
     - rolearn: 追加する IAM ロールの ARN。この値にパスを含めることはできません。例えば、 arn:aws:iam::*111122223333*:role/my-team/developers/*role-name* のような ARN を指定することはできません。ARN は、arn:aws:iam::*111122223333*:role/*role-name* のようにする必要があります。
     - username: IAM ロールにマッピングする Kubernetes 内のユーザー名。
     - groups: ロールをマップする Kubernetes グループのグループまたはリスト。グループには、デフォルトグループか、clusterrolebinding または rolebinding で指定されたグループを使用します。詳細については、Kubernetes ドキュメントの「デフォルトのロールとロールバインディング」を参照してください。
   - IAM ユーザーを追加するには: IAM のベストプラクティスでは、ユーザーではなくロールに許可を付与することが推奨されています。ConfigMap の mapUsers セクション (data の下) に

ユーザーの詳細を追加します。ファイルに存在しない場合は、このセクションを追加します。各エントリは、次のパラメータをサポートしています。

- userarn: 追加する IAM ユーザーの ARN。
- username: IAM ユーザーにマッピングする Kubernetes 内のユーザー名。
- groups: ユーザーをマップする Kubernetes グループのグループまたはリスト。グループには、デフォルトグループか、`clusterrolebinding` または `rolebinding` で指定されたグループを使用します。詳細については、Kubernetes ドキュメントの「デフォルトのロールとロールバインディング」を参照してください。

=== `username` および `groups` のテンプレート変数

`mapRoles` エントリと `mapUsers` エントリの両方の `username` および `groups` フィールドは、認証時に AWS IAM Authenticator が置き換えるテンプレート変数をサポートします。以下のテンプレート変数が利用可能です。

`{{AccountID}}`

認証された IAM プリンシパルの 12 桁の AWS アカウント ID。

`{{SessionName}}`

STS ロールセッション名。@ 文字は - に置き換えられます。EC2 インスタンスロールの場合は、インスタンス ID です。フェデレーティッドロールの場合、これはフェデレーティッドアイデンティティです。`sts:AssumeRole` で引き受けるロールの場合、これは発信者によって設定された `RoleSessionName` パラメータの値です。

`{{SessionNameRaw}}`

文字置き換えなしの raw STS ロールセッション名。

`{{AccessKeyID}}`

リクエストの認証に使用される AWS アクセスキー ID。

`{{EC2PrivateDNSName}}`

EC2 インスタンスのプライベート DNS 名。この変数では、セッション名が有効な EC2 インスタンス ID である必要があり、`ec2:DescribeInstances` API コールを行います。

> **Important**
>
> ロールが sts:AssumeRole で引き受けられると、{{SessionName}} と {{SessionNameRaw}} の値はユーザー制御されます。発信者は、RoleSessionName パラメータをほぼ任意の文字列に設定します。username フィールドでこれらの変数を使用すると、マッピングされたロールを引き受けることができる発信者は、予約されたプレフィックス (system:、eks:、aws:、amazon:、iam: など) と一致しない Kubernetes ユーザー名を偽装できます。groups フィールドでこれらの変数を使用する場合、発信者はセッション名を作成して独自の Kubernetes グループメンバーシップを選択できます。ロールを引き受けることができるすべてのプリンシパルを信頼する場合にのみ、username または groups で {{SessionName}} または {{SessionNameRaw}} を使用します。

例えば、次の YAML ブロックには次が含まれます。

- ノードがそれ自体をクラスターに登録できるように、my-console-viewer-role IAM ノードインスタンスを Kubnetes グループにマッピングしている mapRoles セクション。および、すべてのクラスターですべての Kubernetes リソースを表示することが可能な、Kubernetes グループにマッピングされた IAM ロール。my-console-viewer-role IAM ロールに必要な IAM と Kubernetes グループのアクセス許可のリストについては、「the section called “必要なアクセス許可”」を参照してください。
- デフォルトの AWS アカウントからの admin IAM ユーザーを system:masters Kubernetes グループにマッピングする mapUsers セクション。および、特定の名前空間で Kubernetes リソースを表示することが可能な、Kubernetes グループにマッピングされた、他の AWS アカウントからの my-user ユーザー。my-user IAM ユーザーに必要な IAM と Kubernetes グループのアクセス許可のリストについては、「the section called “必要なアクセス許可”」を参照してください。

必要に応じて行を追加または削除し、すべての値の例を、実際の値で置き換えます。

```
# Please edit the object below. Lines beginning with a '#' will be ignored,
# and an empty file will abort the edit. If an error occurs while saving this file will
 be
# reopened with the relevant failures.
#
apiVersion: v1
data:
  mapRoles: |
    - groups:
```

```
      - system:bootstrappers
      - system:nodes
      rolearn: arn:aws:iam::111122223333:role/my-role
      username: system:node:{{EC2PrivateDNSName}}
    - groups:
      - eks-console-dashboard-full-access-group
      rolearn: arn:aws:iam::111122223333:role/my-console-viewer-role
      username: my-console-viewer-role
  mapUsers: |
    - groups:
      - system:masters
      userarn: arn:aws:iam::111122223333:user/admin
      username: admin
    - groups:
      - eks-console-dashboard-restricted-access-group
      userarn: arn:aws:iam::444455556666:user/my-user
      username: my-user
```

1. ファイルを保存し、テキストエディタを終了します。

aws-auth ConfigMap をクラスターに適用する

マネージド型ノードグループを作成するとき、または aws-auth を使用してノードグループを作成するときに、ConfigMap eksctl が自動的に作成され、クラスターに適用されます。これはノードをクラスターに追加するために当初作成されたものですが、この ConfigMap を使用して、ロールベースアクセスコントロール (RBAC) アクセスを IAM プリンシパルに追加することもできます。セルフマネージド型ノードが起動済みで、クラスターに aws-auth ConfigMap を適用していない場合は、次の手順に従って適用できます。

1. aws-auth ConfigMap が適用済みであるかどうかを確認します。

   ```
   kubectl describe configmap -n kube-system aws-auth
   ```

   「Error from server (NotFound): configmaps "aws-auth" not found」という エラーが表示された場合は、以下のステップを実行してストック ConfigMap を適用します。

2. AWS オーセンティケーター設定マップをダウンロード、編集、適用します。

   a. 設定マップをダウンロードします。

```
curl -O https://s3.us-west-2.amazonaws.com/amazon-eks/cloudformation/2020-10-29/
aws-auth-cm.yaml
```

b. aws-auth-cm.yaml ファイルで、ノードに関連付けられている IAM ロールの Amazon リソースネーム (ARN) に rolearn を設定します。これを行うには、テキストエディタを使用するか、*my-node-instance-role* を置き換えて次のコマンドを実行します。

```
sed -i.bak -e 's|<ARN of instance role (not instance profile)>|my-node-instance-
role|' aws-auth-cm.yaml
```

このファイルの他の行は変更しないでください。

> **Important**
>
> ロールの ARN には、role/my-team/developers/my-role などのパスを含めることはできません。ロールの ARN は、 arn:aws:iam::*111122223333*:role/*my-role* のような形式にする必要があります。この例では、my-team/developers/ を削除する必要があります。

ノードグループの AWS CloudFormation スタック出力を検査し、次の値を探すことができます。

- InstanceRoleARN – eksctl で作成されたノードグループ用
- NodeInstanceRole – AWS マネジメントコンソール で、Amazon EKS で発行された AWS CloudFormation テンプレートで作成されたノードグループ用

c. 設定を適用します。このコマンドが完了するまで数分かかることがあります。

```
kubectl apply -f aws-auth-cm.yaml
```

> **Note**
>
> 認可またはリソースタイプのエラーが発生した場合は、トラブルシューティングトピックの the section called “許可されていないか、アクセスが拒否されました (kubectl)” を参照してください。

3. ノードのステータスを監視し、Ready ステータスになるまで待機します。

```
kubectl get nodes --watch
```

Ctrl+C を入力して、シェルプロンプトに戻ります。

外部 OIDC プロバイダーを使用して Kubernetes へのアクセスをユーザーに許可する

Amazon EKS では、クラスターに対してユーザーを認証する方法として OpenID Connect (OIDC) ID プロバイダーがサポートされます。OIDC ID プロバイダーは AWS アイデンティティとアクセス管理 (IAM) と併用、または代替として使用できます。IAM の使用の詳細については、「the section called "Kubernetes API アクセス"」を参照してください。クラスターに認証を設定した後は、Kubernetes `roles` および `clusterroles` を作成しロールにアクセス許可を割り当て、さらに、このロールを Kubernetes `rolebindings` および`clusterrolebindings` を使用する ID にバインドできます。詳細についてはKubernetes ドキュメントの「RBAC 認可を使用する」を参照してください。

- Kubernetes は OIDC ID プロバイダーを提供しません。既存のパブリック OIDC ID プロバイダーを使用することも、独自の ID プロバイダーを実行することもできます。認定プロバイダーのリストについては、OpenID サイトの「OpenID Certification」(OpenID 認定) を参照してください。
- Amazon EKS が署名キーを検出できるように、OIDC ID プロバイダーの発行者 URL は公的にアクセス可能である必要があります。Amazon EKS は、自己署名証明書を持つ OIDC ID プロバイダーをサポートしていません。
- ノードをクラスターに結合する時に必要なため、クラスターへの IAM 認証を無効にすることはできません。
- Amazon EKS クラスターは、OIDC ID プロバイダーユーザーではなく、AWS IAM プリンシパルが作成する必要があります。これは、クラスター作成者が Kubernetes API ではなく Amazon EKS API と対話するためです。
- コントロールプレーンで CloudWatch ログが有効になっている場合、OIDC ID プロバイダー認証ユーザーは、クラスターの監査ログに一覧表示されます。詳細については、「the section called "コントロールプレーンログを有効または無効にする"」を参照してください。
- OIDC プロバイダーのアカウントで AWS マネジメントコンソールにサインインすることはできません。AWS アイデンティティとアクセス管理 アカウントで AWS マネジメントコンソールにサインインした場合にのみ、the section called "クラスターリソースにアクセスする" が可能となります。

クラスターに関連付けることができる OIDC ID プロバイダーの数は、クラスターの Kubernetes バージョンによって異なります。Kubernetes バージョン 1.32 以降を実行しているクラスターでは、最大 10 個の OIDC ID プロバイダーを関連付けることができます。すべての OIDC プロバイダー設定の合計サイズは 12 KB 未満である必要があります。1.32 より前のバージョンを実行しているクラスターでは、OIDC ID プロバイダーを 1 つだけ関連付けることができます。

- 発行者の URL (`issuerUrl`) と名前 (`name`) は、クラスターに関連付けられたすべての OIDC ID プロバイダー間で一意である必要があります。同じプロバイダーを複数回関連付けることはできません。
- OIDC ID プロバイダーの関連付けはクラスターの更新です。クラスターは `UPDATING` 状態になり、変更がクラスターの API サーバーに完全に適用されるまでに数分かかることがあります。DescribeUpdate オペレーションで更新の進行状況を追跡できます。

> **Note**
>
> コントロールプレーンは発行者 URL からプロバイダーの署名キーを取得するので、発行者はコントロールプレーンから到達可能である必要があります。クラスターがカスタマールートのコントロールプレーン出力 (`controlPlaneEgressMode=CUSTOMER_ROUTED`) を使用している場合は、VPC で設定した出力パスを介して発行者のエンドポイントに到達できることを確認します。詳細については、「the section called “コントロールプレーンエグレス”」を参照してください。

OIDC ID プロバイダーを関連付ける

OIDC ID プロバイダーをクラスターに関連付けるには、プロバイダーから以下の情報が必要です。

発行者 URL

API サーバーがトークン検証のために公開署名キーを検出できるようにする OIDC ID プロバイダーの URL。URL は `https://` で始まり、プロバイダーの OIDC ID トークンで `iss` クレームに対応している必要があります。OIDC 標準に従って、パスコンポーネントは許可されますが、クエリパラメータは許可されません。通常 URL は `https://server.example.org` または `https://example.com` のように、ホスト名のみで構成されます。この URL は以下 `.well-known/openid-configuration` のレベルを指し、インターネット経由で公的にアクセス可能である必要があります。発行者の URL は、クラスターに関連付けられたすべての OIDC ID プロバイダー間で一意である必要があります。

クライアント ID (オーディエンスとも呼ばれる)

OIDC ID プロバイダーに認証リクエストを行うクライアントアプリケーションの ID。

eksctl または AWS マネジメントコンソール を使用して、ID プロバイダーを関連付けることができます。

eksctl を使用して ID プロバイダーを関連付ける

1. 次の内容で、associate-identity-provider.yaml という名前のファイルを作成します。値の例は、実際に使用する値に置き換えます。identityProviders セクションの値は、OIDC ID プロバイダーから取得します。値はidentityProviders での name、type、issuerUrl、および clientId の設定のみで必要です。

```
---
apiVersion: eksctl.io/v1alpha5
kind: ClusterConfig

metadata:
  name: my-cluster
  region: your-region-code

identityProviders:
  - name: my-provider
    type: oidc
    issuerUrl: https://example.com
    clientId: kubernetes
    usernameClaim: email
    usernamePrefix: my-username-prefix
    groupsClaim: my-claim
    groupsPrefix: my-groups-prefix
    requiredClaims:
      string: string
    tags:
      env: dev
```

> **Important**
>
> groupsPrefix または usernamePrefix に対してはsystem: またはその文字列の一部でも指定しないでください。

2. プロバイダーを作成します。

```
eksctl associate identityprovider -f associate-identity-provider.yaml
```

3. kubectl を使用してクラスターと OIDC ID プロバイダーを操作する方法については、Kubernetes ドキュメントの「kubectl の使用」を参照してください。

AWS コンソールを使用して ID プロバイダーを関連付ける

1. Amazon EKS コンソールを開きます。
2. クラスターを選択し、[アクセス] タブを選択してください。
3. [OIDC ID プロバイダー] セクションで、[ID プロバイダーを関連付け] を選択します。
4. [OIDC ID プロバイダーの関連付け] ページで、以下のオプションを入力または選択した上で、[関連付け] を選択します。
   - [名前] に、プロバイダーの一意の名前を入力します。
   - [発行者 URL] に、プロバイダーの URL を入力してください。この URL はインターネットからアクセス可能である必要があります。
   - [クライアント ID] に、OIDC ID プロバイダーのクライアント ID (別名、対象者) を入力します。
   - [ユーザー名のクレーム] に、ユーザー名として使用するクレームを入力してください。
   - [グループクレーム] に、ユーザーのグループとして使用するクレームを入力します。
   - (オプション) [詳細オプション] で、以下の情報を入力または選択してください。
     - [ユーザー名のプレフィックス] – ユーザー名クレームの前に付加するプレフィックスを入力してください。プレフィックスは既存の名前との衝突を防ぐために、ユーザー名クレームの前に付加されます。値を入力せず、ユーザーネームが email 以外の場合、プレフィックスはデフォルトで [発行者 URL] の値になります。値 - を使用すると、すべてのプレフィックスを無効にできます。system: またはその文字列の一部でも指定しないでください。
     - [グループプレフィックス] – グループクレームの前に付加するプレフィックスを入力してください。プレフィックスは、既存の名前 (system:groups など) との競合を防ぐために、グループクレームの前に付加されます。例えば、値 oidc: はoidc:engineering および oidc:infra などのグループ名を作成します。system: またはその文字列の一部でも指定しないでください。
     - [必要なクレーム] – [クレームを追加] 選択し、クライアント ID トークンに必要なクレームを記述する、キーと値のペアを 1 つ以上入力してください。ペアはID トークンで必要なクレームを記述します。設定されている場合、各クレームは一致する値を持つ ID トークンに存在することが検証されます。

a. `kubectl` を使用してクラスターと OIDC ID プロバイダーを操作する方法については、Kubernetes ドキュメントの「kubectl の使用」を参照してください。

複数の ID プロバイダーのベストプラクティス

ユーザー名とグループのプレフィックスを使用する: 各 OIDC ID プロバイダーにユーザー名プレフィックス (`usernamePrefix`) とグループプレフィックス (`groupsPrefix`) を設定して、異なるプロバイダーのアイデンティティが重複する可能性を減らします。プレフィックスを使用しない場合は、プロバイダーごとに一意のユーザー名クレームとグループクレームを使用します。どちらの方法でも、2 つのプロバイダーが同じ Kubernetes ユーザー名またはグループ名を生成できないようになります。

トークンクレームを検証する: 必須クレーム (`requiredClaims`) を使用して、ID トークンに存在する必要があるキーと値のペアを指定し、クラスターが承認したトークンのみを受け入れるようにします。

各トークンのクレームを制限する: クラスターに必要なクレームのみを含むトークンを発行するように ID プロバイダーを設定します。クレームを制限すると、トークンが侵害された場合に公開される情報を減らすことができます。

IAM ポリシーの例

OIDC ID プロバイダーとクラスターの関連付けを防ぐには、以下の IAM ポリシーを作成して、Amazon EKS 管理者の IAM アカウントに関連付けます。詳細については、IAM ユーザーガイドの「IAM ポリシーを作成する」と「IAM ID アクセス許可の追加」、およびサービス認可リファレンスの「アクション」に関するページを参照してください。

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "denyOIDC",
            "Effect": "Deny",
            "Action": [
                "eks:AssociateIdentityProviderConfig"
            ],
            "Resource": "arn:aws:eks:us-west-2:111122223333:cluster/*"

        },
        {
```

```
            "Sid": "eksAdmin",
            "Effect": "Allow",
            "Action": [
                "eks:*"
            ],
            "Resource": "*"
        }
    ]
}
```

以下のポリシーの例では、clientID が kubernetes で、issuerUrl が https://cognito-idp.us-west-2.amazonaws.com/* の場合、OIDC ID プロバイダーの関連付けを許可します。

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "AllowCognitoOnly",
            "Effect": "Deny",
            "Action": "eks:AssociateIdentityProviderConfig",
            "Resource": "arn:aws:eks:us-west-2:111122223333:cluster/my-instance",
            "Condition": {
                "StringNotLikeIfExists": {
                    "eks:issuerUrl": "https://cognito-idp.us-west-2.amazonaws.com/*"
                }
            }
        },
        {
            "Sid": "DenyOtherClients",
            "Effect": "Deny",
            "Action": "eks:AssociateIdentityProviderConfig",
            "Resource": "arn:aws:eks:us-west-2:111122223333:cluster/my-instance",
            "Condition": {
                "StringNotEquals": {
                    "eks:clientId": "kubernetes"
                }
            }
        },
        {
            "Sid": "AllowOthers",
            "Effect": "Allow",
            "Action": "eks:*",
            "Resource": "*"
```

```
        }
    ]
}
```

OIDC ID プロバイダーとクラスターの関連付けを解除します。

OIDC ID プロバイダーとクラスターの関連付けを解除する場合、プロバイダーに含まれるユーザーはクラスターにアクセスできなくなります。ただし、IAM プリンシパルを使用して、クラスターへ引き続きアクセスすることは可能です。

OIDC ID プロバイダーの関連付けを解除することは、クラスターの更新です。クラスターは UPDATING 状態になり、変更がクラスターの API サーバーに完全に適用されるまでに数分かかることがあります。DescribeUpdate オペレーションで更新の進行状況を追跡できます。クラスターに複数の OIDC ID プロバイダーが関連付けられている場合は、一度に 1 つずつ関連付けを解除する必要があります。次のプロバイダーの関連付けを解除する前に、各クラスターの更新が完了するまで待ちます。

1. Amazon EKS コンソールを開きます。
2. [OIDC Identity Providers] (OIDC ID プロバイダー) セクションで、[Disassociate] (関連付けの解除) を選択しID プロバイダー名を入力した上で、[Disassociate] を選択します。

AWS マネジメントコンソール に Kubernetes リソースを表示する

クラスターにデプロイされた Kubernetes リソースを AWS マネジメントコンソール で表示できます。Kubernetes リソースは、AWS CLIまたは eksctl で表示することはできません。コマンドラインツールを使用して Kubernetes リソースを表示するには、kubectl を使用します。

> **Note**
>
> AWS マネジメントコンソール の [リソース] タブと [コンピューティング] タブの [ノード] セクションを表示するには、使用している IAM プリンシパルに特定の IAM と Kubernetes 許可が必要です。詳細については、「the section called “必要なアクセス許可”」を参照してください。

1. Amazon EKS コンソールを開きます。
2. [Clusters] (クラスター) のリストで、表示したい Kubernetes リソースを含んでいるクラスターを選択します。

3. [リソース] タブを選択します。
4. リソースを表示したい [リソースタイプ] グループを選択します。例えば、[ワークロード] などです。そのグループ内のリソースタイプのリストが表示されます。
5. リソースタイプを選択します。例えば、[Workloads] (ワークロード) グループ内の [Deployments] (デプロイメント) などです。リソースタイプの簡単な説明、リソースタイプの詳細な説明が参照できる Kubernetes ドキュメントへのリンク、およびクラスターにデプロイされていてそのタイプに分類されるリソースのリストが表示されます。リストが空の場合、そのタイプのリソースはクラスターにデプロイされていません。
6. 詳細情報を表示するには、そのリソースを選択します。次の例を試してください。
   - [ワークロード] グループを選択し、[デプロイメント] のリソースタイプを選択し、次に [CoreDNS] リソースを選択します。リソースを選択すると、デフォルトで [構造化ビュー] が表示されます。リソースタイプによっては、[構造化ビュー] に [ポッド] セクションが表示されます。このセクションでは、ワークロードによって管理される Pod が一覧表示されます。一覧表示されている任意の Pod を選択し、Pod の情報を表示できます。すべてのリソースタイプについて、[Structured View] (構造化ビュー) で情報が表示されるわけではありません。リソース表示画面で右上隅にある [Raw view] (raw ビュー) を選択すると、そのリソースについて Kubernetes API からの完全な JSON レスポンスが表示されます。
   - [Cluster] (クラスター) グループを選択してから、[Nodes] (ノード) リソースタイプを選択します。クラスター内のすべてのノードのリストが表示されます。ノードは、任意の Amazon EKS ノードタイプとなります。これはクラスターの [コンピューティング] () タブを選択したときに [ノード] セクションに表示されるのと同じリストです。リストからノードリソースを選択します。[構造化ビュー] では、[ポッド] セクションも表示されます。このセクションでは、ノード上で実行中のすべての Pod が表示されます。

必要なアクセス許可

AWS マネジメントコンソール の [リソース] タブと [コンピューティング] タブの [ノード] セクションを表示するには、使用している IAM プリンシパルに特定の最小限の IAM と Kubernetes 許可が必要です。IAM と Kubernetes の両方の RBAC アクセス許可を正しく設定する必要があります。次のステップを実行して、IAM プリンシパルに必要な許可を割り当てます。

1. `eks:AccessKubernetesApi`、および Kubernete リソースを表示するために必要なその他の IAM アクセス許可が、使用している IAM プリンシパルに割り当てられていることを確認してください。IAM プリンシパルの許可を編集する方法の詳細については、「IAM ユーザーガイド」の「プリンシパルへのアクセスの制御」を参照してください。ロールのアクセス許可を編集する方

法の詳細については、IAM ユーザーガイドの「ロールのアクセス許可ポリシーの変更 (コンソール)」を参照してください。

次のポリシーの例には、アカウント内のすべてのクラスター内にある Kubernetes リソースを表示するために必要な、プリンシパルへの許可が含まれています。*111122223333* を自分の AWS アカウント ID に置き換えます。

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "eks:ListFargateProfiles",
                "eks:DescribeNodegroup",
                "eks:ListNodegroups",
                "eks:ListUpdates",
                "eks:AccessKubernetesApi",
                "eks:ListAddons",
                "eks:DescribeCluster",
                "eks:DescribeAddonVersions",
                "eks:ListClusters",
                "eks:ListIdentityProviderConfigs",
                "iam:ListRoles"
            ],
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": "ssm:GetParameter",
            "Resource": "arn:aws:ssm:*:111122223333:parameter/*"
        }
    ]
}
```

接続されたクラスター内のノードを表示するには、Amazon EKS コネクタの IAM ロールがクラスター内のプリンシパルを偽装できる必要があります。これにより、Amazon EKS Connector はプリンシパルを Kubernetes ユーザーにマッピングできます。

2. EKS アクセスエントリを使用して Kubernetes RBAC アクセス許可を設定します。

   EKS アクセスエントリとは

EKS アクセスエントリは、IAM プリンシパル (ユーザーとロール) に Kubernetes クラスターへのアクセス許可を付与する効率的な方法です。アクセスエントリを使用することで、Kubernetes RBAC リソースや `aws-auth` ConfigMap を手動で管理する代わりに、AWS が提供するマネージドポリシーを使用して IAM と Kubernetes アクセス許可間のマッピングを自動的に処理することができます。アクセスエントリの詳細情報については「the section called “アクセスエントリ”」を参照してください。利用可能なアクセスポリシーとそのアクセス許可の詳細については、「アクセスポリシーアクセス許可を確認する」を参照してください。

Kubernetes アクセス許可をアタッチして、次の 2 つの方法でエントリにアクセスできます :

- アクセスポリシーを使用する: アクセスポリシーは、AWS によって管理される事前定義された Kubernetes アクセス許可テンプレートです。これらは、一般的なユースケース用の標準化されたアクセス許可セットを提供します。
- Kubernetes グループを参照する: IAM アイデンティティを Kubernetes グループに関連付けると、グループのアクセス許可を付与する Kubernetes リソースを作成できます。詳細については Kubernetes ドキュメントの「RBAC 認可を使用する」を参照してください。

  a. AWS CLI を使用して IAM プリンシパルのアクセスエントリを作成します。*#######* の部分は自分のクラスター名に置き換えます。*111122223333* は、ご自分のアカウント ID に置き換えます。

  ```
  aws eks create-access-entry \
      --cluster-name my-cluster \
      --principal-arn arn:aws:iam::111122223333:role/my-console-viewer-role
  ```

  出力例は次のとおりです。

  ```
  {
      "accessEntry": {
          "clusterName": "my-cluster",
          "principalArn": "arn:aws:iam::111122223333:role/my-console-viewer-role",
          "kubernetesGroups": [],
          "accessEntryArn": "arn:aws:eks:region-code:111122223333:access-
  entry/my-cluster/role/111122223333/my-console-viewer-role/
  abc12345-1234-1234-1234-123456789012",
          "createdAt": "2024-03-15T10:30:45.123000-07:00",
          "modifiedAt": "2024-03-15T10:30:45.123000-07:00",
          "tags": {},
          "username": "arn:aws:iam::111122223333:role/my-console-viewer-role",
          "type": "STANDARD"
  ```

```
    }
}
```

b. ポリシーをアクセスエントリに関連付けます。Kubernetes リソースを表示するには、AmazonEKSViewPolicy を使用します。

```
aws eks associate-access-policy \
    --cluster-name my-cluster \
    --principal-arn arn:aws:iam::111122223333:role/my-console-viewer-role \
    --policy-arn arn:aws:eks::aws:cluster-access-policy/AmazonEKSViewPolicy \
    --access-scope type=cluster
```

出力例は次のとおりです。

```
{
    "clusterName": "my-cluster",
    "principalArn": "arn:aws:iam::111122223333:role/my-console-viewer-role",
    "associatedAt": "2024-03-15T10:31:15.456000-07:00"
}
```

名前空間固有のアクセスでは、ポリシー範囲を特定の名前空間に限定できます。

```
aws eks associate-access-policy \
    --cluster-name my-cluster \
    --principal-arn arn:aws:iam::111122223333:role/my-console-viewer-role \
    --policy-arn arn:aws:eks::aws:cluster-access-policy/AmazonEKSViewPolicy \
    --access-scope type=namespace,namespaces=default,kube-system
```

c. アクセスエントリが正常に作成されたことを確認します。

```
aws eks describe-access-entry \
    --cluster-name my-cluster \
    --principal-arn arn:aws:iam::111122223333:role/my-console-viewer-role
```

d. 関連付けられたポリシーを一覧表示して、ポリシーの関連付けを確認します。

```
aws eks list-associated-access-policies \
    --cluster-name my-cluster \
    --principal-arn arn:aws:iam::111122223333:role/my-console-viewer-role
```

出力例は次のとおりです。

```
{
    "associatedAccessPolicies": [
        {
            "policyArn": "arn:aws:eks::aws:cluster-access-policy/
AmazonEKSViewPolicy",
            "accessScope": {
                "type": "cluster"
            },
            "associatedAt": "2024-03-15T10:31:15.456000-07:00",
            "modifiedAt": "2024-03-15T10:31:15.456000-07:00"
        }
    ]
}
```

CloudTrail の可視性

Kubernetes リソースを表示すると、CloudTrail ログに次のオペレーション名が表示されます。

- `AccessKubernetesApi` – リソースの読み取りまたは表示時

この CloudTrail イベントは、Kubernetes リソースへの読み取りアクセスの監査証跡を提供します。

> **Note**
>
> このオペレーション名は、監査目的でのみ CloudTrail ログに表示されます。これは IAM アクションではなく、IAM ポリシーステートメントでは使用できません。IAM ポリシーを使用して Kubernetes リソースへの読み取りアクセスを制御するには、「the section called “必要なアクセス許可”」セクションに示すように `eks:AccessKubernetesApi` アクセス許可を使用します。

Kubernetes API への書き込みアクセスを AWS サービスに付与する

必要なアクセス許可

AWS サービスが Amazon EKS クラスター内の Kubernetes リソースに対して書き込みオペレーションを実行できるようにするには、`eks:AccessKubernetesApi` と `eks:MutateViaKubernetesApi` の両方の IAM アクセス許可を付与する必要があります。

例えば、Amazon SageMaker HyperPod はこれらのアクセス許可を使用して、SageMaker AI Studio からのモデルデプロイを有効にします。詳細については、Amazon SageMaker AI デベロッパーガイドの「Set up optional JavaScript SDK permissions」を参照してください。

> **Important**
>
> 作成、更新、削除などの書き込みオペレーションには両方のアクセス許可が必要です。いずれかのアクセス許可がない場合、書き込みオペレーションは失敗します。

CloudTrail の可視性

Kubernetes リソースに対して書き込みオペレーションを実行すると、CloudTrail ログに特定のオペレーション名が表示されます。

- `createKubernetesObject` – 新しいリソースを作成する場合
- `updateKubernetesObject` – 既存のリソースを変更する場合
- `deleteKubernetesObject` – リソースを削除する場合

これらの CloudTrail イベントは、Kubernetes リソースに加えられたすべての変更の詳細な監査証跡を提供します。

> **Note**
>
> これらのオペレーション名は、監査目的でのみ CloudTrail ログに表示されます。これらは IAM アクションではなく、IAM ポリシーステートメントでは使用できません。IAM ポリシーを使用して Kubernetes リソースへの書き込みアクセスを制御するには、「the section called

"必要なアクセス許可"」セクションに示すように eks:MutateViaKubernetesApi アクセス許可を使用します。

kubeconfig ファイルを作成して kubectl を EKS クラスターに接続する

> **Tip**
> 今後開催予定の Amazon EKS ワークショップに登録してください。

このトピックでは、クラスター用の kubeconfig ファイルを作成します (または既存のファイルを更新します)。

kubectl コマンドラインツールは、kubeconfig ファイルの設定情報を使用して、クラスターの API サーバーと通信します。詳細については、Kubernetes ドキュメントの「kubeconfig ファイルを使用してクラスターアクセスを組織する」を参照してください。

Amazon EKS はクラスター認証に kubectl で aws eks get-token コマンドを使用します。デフォルトでは、AWS CLI は次のコマンドで返されるものと同じ認証情報を使用します。

```
aws sts get-caller-identity
```

- 既存の Amazon EKS クラスター。デプロイするには、はじめに を参照してください。
- デバイスまたは AWS CloudShell に、kubectl コマンドラインツールがインストールされていること。バージョンはご使用のクラスターの Kubernetes バージョンと同じか、1 つ前のマイナーバージョン以前、あるいはそれより新しいバージョンが使用できます。例えば、クラスターのバージョンが 1.29 である場合、kubectl のバージョン 1.28、1.29、または 1.30 が使用できます。kubectl をインストールまたはアップグレードする方法については、the section called "kubectl と eksctl のセットアップ" を参照してください。
- ご使用のデバイスまたは AWS CloudShell で、バージョン 2.12.3 以降、または AWS コマンドラインインターフェイス (AWS CLI) のバージョン 1.27.160 以降がインストールおよび設定されていること。現在のバージョンを確認するには、aws --version | cut -d / -f2 | cut -d ' ' -f1 を参照してください。yum、apt-get、macOS 用の Homebrew などのパッケージマネージャーは、多くの場合 AWS CLI の最新バージョンより数バージョン古くなっています。最新バージョンをインストールするには、「 AWS コマンドラインインターフェイスユーザーガイド」

の「インストール」および「aws configure を使用したクイック設定」を参照してください。AWS クラウドシェル にインストールされている AWS CLI バージョンも最新バージョンより数バージョン遅れることがあります。更新するには、「AWS CloudShell ユーザーガイド」の「ホームディレクトリへの AWS CLI のインストール」を参照してください。

- 指定したクラスターに対して `eks:DescribeCluster` API アクションを使用するアクセス許可を持つ IAM ユーザーまたはロール。詳細については、the section called “アイデンティティベースのポリシー” を参照してください。独自の OpenID Connect プロバイダーの ID を使用してクラスターにアクセスする場合は、Kubernetes ドキュメントの「kubectl の使用」を参照して、`kubeconfig` ファイルを作成または更新してください。

EKS コンソールを使用して接続する (CloudShell)

AWS CloudShell を使用して、ローカルセットアップなしで AWS マネジメントコンソール からクラスターに直接接続できます。

1. https://console.aws.amazon.com/eks/home#/clusters で Amazon EKS コンソールを開きます。
2. 接続するクラスターの名前を選択します。
3. クラスターの詳細ページで、右上のアクションバーから [接続] を選択します。
4. AWS CloudShell は、クラスター用に事前設定された `kubectl` で開きます。

接続したら、`kubectl` コマンドをすぐに実行できます。

```
kubectl get nodes
kubectl get pods -A
```

CloudShell セッションには、`kubectl`、AWS CLI、および標準の CloudShell ユーティリティが含まれます。この機能は、パブリック API サーバーエンドポイントとプライベート API サーバーエンドポイントの両方を持つクラスターをサポートします。プライベートエンドポイントを持つクラスターの場合、CloudShell は自動的に VPC 環境を起動してクラスターのプライベート API サーバーに到達し、環境の名前を入力するように求めます。

AWS CLI を使用して **`kubeconfig`** ファイルを作成する

- ご使用のデバイスまたは AWS CloudShell で、バージョン `2.12.3` 以降、または AWS コマンドラインインターフェイス (AWS CLI) のバージョン `1.27.160` 以降がインストールおよび設定されていること。現在のバージョンを確認するには、`aws --version | cut -d / -f2 | cut -`

d ' ' -f1 を参照してください。yum、apt-get、macOS 用の Homebrew などのパッケージマネージャーは、多くの場合 AWS CLI の最新バージョンより数バージョン古くなっています。最新バージョンをインストールするには、「 AWS コマンドラインインターフェイスユーザーガイド」の「インストール」および「aws configure を使用したクイック設定」を参照してください。AWS クラウドシェル にインストールされている AWS CLI バージョンも最新バージョンより数バージョン遅れることがあります。更新するには、「AWS CloudShell ユーザーガイド」の「ホームディレクトリへの AWS CLI のインストール」を参照してください。

- 指定したクラスターに対して eks:DescribeCluster API アクションを使用するアクセス許可。詳細については、the section called “アイデンティティベースのポリシー” を参照してください。

1. クラスター用の kubeconfig ファイルを作成もしくは更新します。*region-code* はクラスターがある AWS リージョンに、*my-cluster* はクラスターの名前に置き換えます。

```
aws eks update-kubeconfig --region region-code --name my-cluster
```

デフォルトでは、最終的な設定ファイルは、ホームディレクトリのデフォルト kubeconfig パス (.kube) に作成されるか、その場所で既存 config ファイルとマージされます。別のパスは --kubeconfig オプションを使用して指定できます。

kubectl コマンドを発行する場合、認証に使用する --role-arn オプションで IAM ロール ARN を指定できます。それ以外の場合は、デフォルトの AWS CLI または SDK の認証情報チェーンの中の IAM プリンシパルが使用されます。デフォルトの AWS CLI または SDK の ID を表示するには、 aws sts get-caller-identity コマンドを実行します。

使用可能なすべてのオプションについては、aws eks update-kubeconfig help コマンドを実行するか、「AWS CLI コマンドリファレンス」の「update-kubeconfig」を参照してください。

2. 設定をテストします。

```
kubectl get svc
```

出力例は次のとおりです。

```
NAME             TYPE        CLUSTER-IP   EXTERNAL-IP   PORT(S)   AGE
svc/kubernetes   ClusterIP   10.100.0.1   <none>        443/TCP   1m
```

認可またはリソースタイプのエラーが発生した場合は、トラブルシューティングトピックの the section called “許可されていないか、アクセスが拒否されました (kubectl)” を参照してください。

Kubernetes ワークロードに Kubernetes サービスアカウントを使用して AWS へのアクセスを許可する

Kubernetes サービスアカウントは、ポッドで実行されるプロセスの ID を提供します。詳細については、Kubernetes ドキュメントの「Managing Service Accounts」(サービスアカウントの管理) を参照してください。ポッドが AWS サービスにアクセスする必要がある場合、サービスアカウントを AWS Identity and Access Management のアイデンティティにマッピングして、アクセス権を付与することができます。詳細については「the section called “IRSA による認証情報”」または「the section called “Pod Identity”」を参照してください。

サービスアカウントトークン

BoundServiceAccountTokenVolume 機能は、Kubernetes バージョンでデフォルトで有効になっています。この機能により、Kubernetes で実行されているワークロードは、対象者、時間、および鍵に紐付けされた JSON ウェブトークンをリクエストできるようになり、サービスアカウントトークンのセキュリティが向上します。サービスアカウントトークンの有効期限は 1 時間です。Kubernetes の以前のバージョンでは、トークンに有効期限はありませんでした。つまり、これらのトークンに依存するクライアントは 1 時間以内にトークンを更新する必要があります。以下の Kubernetes クライアント SDK は、必要な期間内にトークンを自動的に更新します。

- Go バージョン `0.15.7` 以降
- Python バージョン `12.0.0` 以降
- Java バージョン `9.0.0` 以降
- JavaScript バージョン `0.10.3` 以降
- Ruby `master` ブランチ
- Haskell バージョン`0.3.0.0`
- C# バージョン `7.0.5` 以降

ワークロードで古いバージョンのクライアントを使用している場合は、更新する必要があります。有効期限付きの新しいサービスアカウントトークンにクライアントがスムーズに移行できるようにする

ため、Kubernetes ではデフォルトの 1 時間を超えてサービスアカウントトークンの有効期限が延長されます。Amazon EKS クラスターの場合、延長できる有効期限は 90 日です。Amazon EKS クラスターの Kubernetes API サーバーでは、90 日を超えるトークンのリクエストは拒否されます。アプリケーションとその依存関係を確認し、Kubernetes クライアント SDK が前記のバージョンと同じかそれ以降であることを確認することをお勧めします。

API サーバーが 1 時間を超える古いトークンによるリクエストを受信すると、API 監査ログイベントに `annotations.authentication.k8s.io/stale-token` で注釈を付けます。注釈の値は次の例のようになります。

```
subject: system:serviceaccount:common:fluent-bit, seconds after warning threshold:
 4185802.
```

クラスターで コントロールプレーンのログ記録 が有効になっている場合、注釈は監査ログに記録されます。以下の CloudWatch Log Insights クエリを使用すると、Amazon EKS クラスター内で古いトークンを使っているすべての Pod を特定できます。

```
fields @timestamp
|filter @logStream like /kube-apiserver-audit/
|filter @message like /seconds after warning threshold/
|parse @message "subject: *, seconds after warning threshold:*\"" as subject,
 elapsedtime
```

`subject` は、Pod が使用したサービスアカウントを示します。`elapsedtime` は、最新のトークンを読み込んでからの経過時間 (秒単位) を示します。`elapsedtime` が 90 日 (7,776,000 秒間) を超えると、API サーバーへのリクエストは拒否されます。トークンの自動更新を行う前記のバージョンのいずれかを使用するよう、アプリケーションの Kubernetes クライアント SDK をプロアクティブに更新する必要があります。使用しているサービスアカウントトークンが 90 日に近く、トークンの有効期限が切れるまでクライアント SDK のバージョンを更新するのに十分な時間がない場合、既存の Pod を終了して新しい Pod を作成できます。サービスアカウントトークンが再フェッチされるので、クライアントバージョン SDK を更新するのに 90 日が追加されたことになります。

Pod がデプロイの一部である場合、高可用性を維持しながら Pod を終了する方法として、次のコマンドによるロールアウトの実行をお勧めします。*my-deployment* をデプロイの名前で置き換えます。

```
kubectl rollout restart deployment/my-deployment
```

クラスターアドオン

以下のクラスターアドオンが更新され、サービスアカウントトークンを自動的に再フェッチする Kubernetes クライアント SDK が使用できるようになりました。リストされているバージョン、またはそれ以降のバージョンを、 クラスターにインストールすることをお勧めします。

- Amazon VPC CNI Plugin for Kubernetes およびメトリクスヘルパーのプラグインのバージョン `1.8.0` 以降。現在のバージョンを確認したり、更新したりするには、the section called “Amazon VPC CNI” および cni-metrics-helper を参照してください。
- CoreDNS のバージョン `1.8.4` 以降。現在のバージョンを確認したり、更新したりするには、the section called “CoreDNS” を参照してください。
- AWS ロードバランサーコントローラーのバージョン `2.0.0` 以降。現在のバージョンを確認したり、更新したりするには、the section called “ AWS ロードバランサーコントローラー” を参照してください。
- 現在の `kube-proxy` バージョン 現在のバージョンを確認したり、更新したりするには、the section called “kube-proxy” を参照してください。
- AWS for Fluent Bit バージョン `2.25.0` 以降。現在のバージョンを更新するには、GitHub で「Releases」(リリース) を参照してください。
- Fluentd イメージバージョン 1.14.6-1.2 以降、および Kubernetes メタデータバージョン 2.11.1 以降用の Fluentd フィルタープラグイン。

Amazon Elastic Kubernetes Service クラスター上のワークロードへの AWS Identity and Access Management 許可の付与

Amazon EKS には、Amazon EKS クラスターで実行されるワークロードに AWS Identity and Access Management 許可を付与する方法が 2 つあります。サービスアカウント用の IAM ロール と EKS Pod Identity です。

サービスアカウントの IAM ロール

サービスアカウント用の IAM ロール (IRSA) は、Amazon S3 バケットや Amazon DynamoDB テーブルなど、他の様々な AWS リソースにアクセスするためのきめ細かい IAM アクセス許可で AWS 上で動作する Kubernetes アプリケーションを構成します。同じ Amazon EKS クラスター内で複数のアプリケーションを同時に実行し、各アプリケーションに必要な最小限のアクセス権限のみを持たせることができます。IRSA は、Amazon EKS、Amazon EKS Anywhere、AWS での Red Hat OpenShift サービス、Amazon EC2 インスタンスでのセルフマネージド型 Kubernetes

クラスターなどの、AWS でサポートされるさまざまな Kubernetes デプロイオプションをサポートするように構築されました。そのため、IRSA は IAM のような基本的な AWS サービスを使用して構築されており、Amazon EKS サービスや EKS API に直接依存することはありませんでした。詳細については、「the section called “IRSA による認証情報”」を参照してください。

EKS Pod Identity

EKS Pod Identity は、Amazon S3 バケット、Amazon DynamoDB テーブルなどのさまざまな AWS リソースにアクセスするためのアプリケーションを認証するためのシンプルなワークフローをクラスター管理者に提供します。EKS Pod Identity は EKS 専用であるため、クラスター管理者が Kubernetes アプリケーションを設定して IAM アクセス許可を取得する方法が簡単になります。これらの権限は、AWS マネジメントコンソール、EKS API、および AWS CLI から直接行う、より少ない手順で簡単に設定できるようになり、クラスター内のどの Kubernetes オブジェクトに対しても実行する必要がありません。クラスター管理者は EKS サービスと IAM サービスを切り替えたり、特権的な IAM オペレーションを使用してアプリケーションに必要な権限を設定したりする必要がありません。新しいクラスターを作成するときにロールの信頼ポリシーを更新しなくても、IAM ロールを複数のクラスターで使用できるようになりました。EKS Pod Identity が提供する IAM 認証情報には、クラスター名、名前空間、サービスアカウント名などの属性を含むロールセッションタグが含まれます。ロールセッションタグを使用すると、管理者は一致するタグに基づいて AWS リソースへのアクセスを許可することで、複数のサービスアカウントで機能する単一のロールを作成できます。詳細については、the section called “Pod Identity” を参照してください。

EKS Pod Identity と IRSA の比較

大まかに言うと、EKS Pod Identity と IRSA はどちらも、Kubernetes クラスター上で実行されているアプリケーションに IAM アクセス許可を付与できるようにします。ただし、設定方法、サポートされる制限、有効になる機能は根本的に異なります。以下では、両ソリューションの重要な側面をいくつか比較します。

> **Note**
>
> AWS では、可能な限り EKS Pod Identity を使用して AWS リソースへのアクセスをポッドに許可することをお勧めします。詳細については、the section called “Pod Identity” を参照してください。

属性	EKS Pod Identity	IRSA
ロール拡張性	新しく導入された Amazon EKS サービスプリンシパル `pods.eks.amazonaws.com` との信頼を確立するには、各ロールを一度設定する必要があります。この 1 回限りのステップを済ませると、そのロールを新しいクラスターで使用するたびにそのロールの信頼ポリシーを更新する必要がなくなります。	新しいクラスターでロールを使用するたびに、IAM ロールの信頼ポリシーを新しい EKS クラスター OIDC プロバイダーエンドポイントで更新する必要があります。
クラスターのスケーラビリティ	EKS Pod Identity では、ユーザーが IAM OIDC プロバイダーをセットアップする必要がないため、この制限は適用されません。	各 EKS クラスターには、OpenID Connect (OIDC) 発行者の URL が関連付けられています。IRSA を使用するには、IAM の EKS クラスターごとに固有の OpenID Connect プロバイダーを作成する必要があります。IAM のデフォルトのグローバル制限は、AWS アカウントにつき 100 の OIDC プロバイダーです。IRSA で AWS アカウントにつき 100 を超える EKS クラスターを使用する予定の場合は、IAM OIDC プロバイダーの制限に達します。
ロールのスケーラビリティ	EKS Pod Identity では、ユーザーが信頼ポリシーで IAM ロールとサービスアカウント間の信頼関係を定義する必要がないため、この制限は適用されません。	IRSA では、IAM ロールとサービスアカウント間の信頼関係を、ロールの信頼ポリシーで定義します。デフォルトでは、信頼ポリシーの長さは 2048 です。つまり、通常、1 つの

属性	EKS Pod Identity	IRSA
		信頼ポリシーで 4 つの信頼関係を定義できます。信頼ポリシーの長さの上限を増やすことはできますが、通常、1 つの信頼ポリシー内で作成できる信頼関係は最大 8 つに制限されます。
STS API クォータの使用	EKS Pod Identity は、ポッドへの AWS 認証情報の配信を簡素化し、AWS Security Token Service (STS) でコードを直接呼び出す必要はありません。EKS サービスはロールの引き受けを処理し、ポッドが AWS STS と通信したり STS API Quota を使用したりすることなく、ポッド内の AWS SDK を使用して記述されたアプリケーションに認証情報を配信します。	IRSA では、ポッドの AWS SDK を使用して記述されたアプリケーションは、トークンを使用して AWS Security Token Service (STS) で `AssumeRoleWithWebIdentity` API を呼び出します。AWS SDK 上のコードのロジックによっては、コードが AWS STS への不必要な呼び出しを行い、スロットリングエラーを受信する可能性があります。

属性	EKS Pod Identity	IRSA
ロールの再利用性	EKS Pod Identity によって提供される一時的な AWS STS 認証情報には、クラスター名、名前空間、サービスアカウント名などのロールセッションタグが含まれます。ロールセッションタグを使用すると、管理者は、アタッチされたタグに基づいて AWS リソースへのアクセスを許可することで、複数のサービスアカウントで使用できる単一の IAM ロールを作成し、そのロールを有効な権限を変えることができます。これは、属性ベースのアクセス制御 (ABAC) とも呼ばれます。詳細については、the section called “Pod にアクセス権を付与する” を参照してください。	AWS STS セッションタグはサポートされていません。ロールはクラスタ間で再利用できますが、すべてのポッドにそのロールのすべての権限が付与されます。
サポートされている環境	EKS Pod Identity は Amazon EKS でのみ使用できます。	IRSA は、Amazon EKS、Amazon EKS Anywhere、AWS での Red Hat OpenShift サービス、Amazon EC2 インスタンスでのセルフマネージド型 Kubernetes クラスターなどに使用できます。

属性	EKS Pod Identity	IRSA
サポートされている EKS バージョン	サポート対象のすべての EKS クラスターバージョン。特定のバージョンについては、the section called “EKS Pod Identity クラスターバージョン” を参照してください。	サポート対象のすべての EKS クラスターバージョン。

サービスアカウントの IAM ロール

> **Tip**
>
> 今後開催予定の Amazon EKS ワークショップに登録してください。

ポッドのコンテナ内のアプリケーションは AWS SDK または AWS CLI で、AWS Identity and Access Management (IAM) アクセス許可を使用した AWS サービスへの API リクエストを行うことができます。アプリケーションは AWS 認証情報で AWS API リクエストに署名する必要があります。サービスアカウントの IAM ロール (IRSA) には、Amazon EC2 インスタンスプロファイルから Amazon EC2 インスタンスに認証情報を提供する場合と同じような方法で、アプリケーションの認証情報を管理する機能があります。AWS 認証情報を作成してコンテナに配布したり、Amazon EC2 インスタンスのロールを使用したりする必要はありません。IAM ロールを Kubernetes サービスアカウントと関連付けて、サービスアカウントを使用するように Pod を設定できます。AWS Outposts の Amazon EKS 用ローカルクラスターで、サービスアカウントに IAM ロールを使用することはできません。

サービスアカウントの IAM ロールには、次の利点があります。

- 最小特権 – IAM アクセス許可の範囲をサービスアカウントに設定すると、そのサービスアカウントを使用する Pod にのみそのアクセス許可を付与できます。また、この機能により、`kiam` や `kube2iam` などのサードパーティーのソリューションが不要になります。
- 認証情報の分離 – Amazon EC2 インスタンスメタデータサービス (IMDS) へのアクセスが制限されている場合、ポッド内のコンテナは、そのコンテナが使用するサービスアカウントに関連付けられている IAM ロールの認証情報のみを取得できます。コンテナは、他の Pod 内のコンテナで使われている認証情報にアクセスすることはできません。IMDS が制限されていない場合、ポッドのコンテナは「Amazon EKS ノード IAM ロール」にもアクセスでき、コンテナは同じノード上にある

他のポッドの IAM ロールの認証情報にアクセスできる可能性があります。詳細については「ワーカーノードに割り当てられたインスタンスプロファイルへのアクセスを制限する」を参照してください。

> **Note**
>
> `hostNetwork: true` で設定されたポッドは常に IMDS アクセスできますが、AWS SDK および CLI は有効にすると IRSA 認証情報が使用されます。

- 監査性 – AWS CloudTrail にアクセスしイベントのログ記録を利用することで、遡及的な監査を確実に行えます。

> **Important**
>
> コンテナはセキュリティ境界ではなく、サービスアカウントに IAM ロールを使用してもこの点は変更されません。同じノードに割り当てられたポッドはカーネルを共有し、ポッド設定によっては他のリソースも共有される可能性があります。別々のノードで実行されているポッドはコンピューティングレイヤーで分離されますが、Kubernetes API に対し個別のインスタンスの範囲を超えて追加のアクセス許可を持つノードアプリケーションがあります。例としては `kubelet`、`kube-proxy`、CSI ストレージドライバー、ユーザー独自の Kubernetes アプリケーションなどがあります。

以下の手順を実行してサービスアカウントの IAM ロールを有効にします。

1. クラスターの IAM OIDC プロバイダーを作成する – この手順は、クラスターごとに 1 回だけ実行する必要があります。

   > **Note**
   >
   > クラスターの VPC にアウトバウンドインターネットアクセスがなく、クラスター OIDC エンドポイントへのプライベートアクセスをセットアップしていない場合、`eksctl` で OIDC プロバイダーを作成するなど、VPC 内からそのエンドポイントに到達するオペレーションでは、OIDC 発行者のホスト名を解決できません。エラーメッセージの例は次のとおりです。

```
server cant find oidc.eks.region.amazonaws.com: NXDOMAIN
```

VPC からクラスター OIDC エンドポイントにプライベートにアクセスするには、プライベート DNS を有効にして、それの VPC インターフェイスエンドポイント (com.amazonaws.*region-code*.oidc-eks) を作成します。詳細については、「the section called “ AWS プライベートリンク”」を参照してください。
または、VPC の外部 (AWS CloudShell など) でコマンドを実行するか、スプリットホライズン条件付きリゾルバーを作成できます。例については、GitHub の「Amazon EKS feature request」を参照してください。

2. Kubernetes サービスアカウントに IAM ロールを割り当てる - この手順は、アプリケーションに付与する固有のアクセス許可のセットごとに実行します。
3. Kubernetes サービスアカウントを使用するように Pod を設定する – この手順は、AWS サービスへのアクセスを必要とする Pod ごとに実行します。
4. AWS SDK で IRSA を使用する – ワークロードがサポートされているバージョンの AWS SDK を使用していること、およびワークロードがデフォルトの認証情報チェーンを使用していることを確認します。

IAM、Kubernetes、OpenID Connect (OIDC) の背景情報

2014 年に、OpenID Connect (OIDC) を使用したフェデレーティッドアイデンティティのサポートが、AWS Identity and Access Management に追加されました。この機能により、サポートされている ID プロバイダーで AWS API コールを認証し、有効な OIDC JSON ウェブトークン (JWT) を受け取ることができます。このトークンを AWS STS の `AssumeRoleWithWebIdentity` API オペレーションに渡し、一時的な IAM ロールの認証情報を受け取ることができます。これらの認証情報を使用して、Amazon S3 や DynamoDB などの任意の AWS サービスとやり取りできます。

各 JWT トークンは署名キーペアによって署名されます。キーは Amazon EKS が管理する OIDC プロバイダーで提供され、プライベートキーは 7 日ごとにローテーションされます。Amazon EKS はパブリックキーの有効期限が切れるまで保持します。外部 OIDC クライアントに接続する場合は、パブリックキーが期限切れになる前に、キーを更新する必要があることに注意してください。署名キーを取得して OIDC トークンを検証する方法について説明します。

Kubernetes は、独自の内部 ID システムとして長い間、サービスアカウントを使用してきました。ポッドは、Kubernetes API サーバーのみが検証できる自動マウントトークン (OIDC JWT ではない) を使用して Kubernetes API サーバーで認証できます。これらのレガシーサービスアカウン

トトークンは期限切れにならず、署名キーの更新は難しいプロセスです。Kubernetes バージョン 1.12 では、新しい ProjectedServiceAccountToken 機能のサポートが追加されました。この機能は、サービスアカウントアイデンティティを含む OIDC JSON ウェブトークンで、設定可能なオーディエンスをサポートします。

Amazon EKS は、ProjectedServiceAccountToken JSON ウェブトークンの署名キーを含むクラスターごとにパブリック OIDC 検出エンドポイントをホストするため、IAM などの外部システムで、Kubernetes によって発行された OIDC トークンを検証して受け入れることができます。

クラスターの IAM OIDC プロバイダーを作成するには

クラスターには、OpenID Connect (OIDC) 発行者の URL が関連付けられています。サービスアカウントで AWS Identity and Access Management (IAM) ロールを使用するには、クラスターの OIDC 発行者 URL に対して IAM OIDC プロバイダーが存在している必要があります。

前提条件

- 既存の Amazon EKS クラスター。デプロイするには、はじめに を参照してください。
- ご使用のデバイスまたは AWS CloudShell で、バージョン 2.12.3 以降、または AWS コマンドラインインターフェイス (AWS CLI) のバージョン 1.27.160 以降がインストールおよび設定されていること。現在のバージョンを確認するには、aws --version | cut -d / -f2 | cut -d ' ' -f1 を参照してください。yum、apt-get、macOS 用の Homebrew などのパッケージマネージャーは、多くの場合 AWS CLI の最新バージョンより数バージョン古くなっています。最新バージョンをインストールするには、「 AWS コマンドラインインターフェイスユーザーガイド」の「インストール」および「aws configure を使用したクイック設定」を参照してください。AWS クラウドシェル にインストールされている AWS CLI バージョンも最新バージョンより数バージョン遅れることがあります。更新するには、「AWS CloudShell ユーザーガイド」の「ホームディレクトリへの AWS CLI のインストール」を参照してください。
- デバイスまたは AWS CloudShell に、kubectl コマンドラインツールがインストールされていること。バージョンはご使用のクラスターの Kubernetes バージョンと同じか、1 つ前のマイナーバージョン以前、あるいはそれより新しいバージョンが使用できます。例えば、クラスターのバージョンが 1.29 である場合、kubectl のバージョン 1.28、1.29、または 1.30 が使用できます。kubectl をインストールまたはアップグレードする方法については、the section called “kubectl と eksctl のセットアップ” を参照してください。
- クラスター構成を含む既存の kubectl config ファイル。kubectl config ファイルの作成については、the section called “kubectl を使用してクラスターにアクセスする” を参照してください。

eksctl または AWS マネジメントコンソール を使用して、クラスターの IAM OIDC プロバイダーを作成できます。

OIDC プロバイダーの作成 (eksctl)

1. デバイスまたは AWS CloudShell にインストールされている eksctl コマンドラインツールのバージョン 0.215.0 以降。eksctl をインストールまたはアップグレードするには、eksctl ドキュメントの「インストール」を参照してください。
2. クラスターの OIDC 発行者 ID を決定します。

   クラスターの OIDC 発行者 ID を取得し、変数に格納します。<my-cluster> を独自の値に置き換えます。

   ```
   cluster_name=<my-cluster>
   oidc_id=$(aws eks describe-cluster --name $cluster_name --query
    "cluster.identity.oidc.issuer" --output text | cut -d '/' -f 5)
   echo $oidc_id
   ```

3. クラスターの発行者 ID を持つ IAM OIDC プロバイダーが既にアカウントにあるかどうかを確認します。

   ```
   aws iam list-open-id-connect-providers | grep $oidc_id | cut -d "/" -f4
   ```

   出力が返された場合は、既にクラスター用の IAM OIDC プロバイダーが存在するため、次の手順をスキップできます。出力が返されない場合はクラスター用の IAM OIDC プロバイダーを作成する必要があります。

4. 次のコマンドを使用して、クラスターの IAM OIDC ID プロバイダーを作成します。

   ```
   eksctl utils associate-iam-oidc-provider --cluster $cluster_name --approve
   ```

   > **Note**
   >
   > クラスターの VPC にアウトバウンドインターネットアクセスがなく、クラスター OIDC エンドポイントへのプライベートアクセスをセットアップしていない場合、eksctl で OIDC プロバイダーを作成するなど、VPC 内からそのエンドポイントに到達するオペレーションでは、OIDC 発行者のホスト名を解決できません。エラーメッセージの例は次のとおりです。

```
** server cant find oidc.eks.<region-code>.amazonaws.com: NXDOMAIN
```

VPC からクラスター OIDC エンドポイントにプライベートにアクセスするには、プライベート DNS を有効にして、それの VPC インターフェイスエンドポイント (com.amazonaws.<region-code>.oidc-eks) を作成します。詳細については、「AWS PrivateLink を使用してクラスター OIDC エンドポイントにアクセスする」を参照してください。

または、VPC の外部 (AWS CloudShell など) でコマンドを実行するか、スプリットホライズン条件付きリゾルバーを作成できます。例については、GitHub の「Amazon EKS feature request」を参照してください。

OIDC プロバイダーの作成 (AWS コンソール)

1. Amazon EKS コンソールを開きます。
2. 左側ペインで、[クラスター] を選択し、[クラスター] ページでご自身のクラスターの名前を選択します。
3. [概要] タブの [詳細] セクションで、[OpenID Connect プロバイダーの URL] の値を書き留めます。
4. IAM コンソール (https://console.aws.amazon.com/iam/) を開きます。
5. 左のナビゲーションペインの [アクセス管理] で [ID プロバイダー] を選択します。[プロバイダー] がクラスターの URL と一致するリストに表示されている場合は、クラスターのプロバイダーがすでに存在します。クラスターの URL に一致するプロバイダーがリストにない場合、プロバイダーを作成する必要があります。
6. プロバイダーを作成するには、[プロバイダーを追加] を選択します。
7. [プロバイダータイプ] に、[OpenID Connect] を選択します。
8. [プロバイダー URL] に、クラスター用の OIDC プロバイダー URL 入力します。
9. [対象者] には sts.amazonaws.com を入力します。
10. (オプション) 任意のタグ、例えば、このプロバイダーのクラスターを特定するためのタグを追加します。

11[プロバイダーを追加] をクリックします。

次のステップ: the section called “IAM ロールを割り当てる”

IAM ロールを Kubernetes サービスアカウントに割り当てる

このトピックでは、AWS Identity and Access Management (IAM) ロール を引き受けるように Kubernetes サービスアカウントを設定する方法について説明します。任意の Pod はサービスアカウントを使用するように設定すると、ロールにアクセス許可がある AWS サービスすべてにアクセスできます。

前提条件

- 既存のクラスター。まだ所有していない場合は、はじめに でのガイドのいずれかに従って作成できます。
- クラスターの既存 IAM OpenID Connect (OIDC) プロバイダー 既に所有中かどうかの確認、または作成方法については、the section called “IAM OIDC プロバイダー” を参照してください。
- ご使用のデバイスまたは AWS CloudShell で、バージョン `2.12.3` 以降、または AWS コマンドラインインターフェイス (AWS CLI) のバージョン `1.27.160` 以降がインストールおよび設定されていること。現在のバージョンを確認するには、`aws --version | cut -d / -f2 | cut -d ' ' -f1` を参照してください。`yum`、`apt-get`、macOS 用の Homebrew などのパッケージマネージャーは、多くの場合 AWS CLI の最新バージョンより数バージョン古くなっています。最新バージョンをインストールするには、「 AWS コマンドラインインターフェイスユーザーガイド」の「インストール」および「aws configure を使用したクイック設定」を参照してください。AWS クラウドシェル にインストールされている AWS CLI バージョンも最新バージョンより数バージョン遅れることがあります。更新するには、「AWS CloudShell ユーザーガイド」の「ホームディレクトリへの AWS CLI のインストール」を参照してください。
- デバイスまたは AWS CloudShell に、`kubectl` コマンドラインツールがインストールされていること。バージョンはご使用のクラスターの Kubernetes バージョンと同じか、1 つ前のマイナーバージョン以前、あるいはそれより新しいバージョンが使用できます。例えば、クラスターのバージョンが `1.29` である場合、`kubectl` のバージョン `1.28`、`1.29`、または `1.30` が使用できます。`kubectl` をインストールまたはアップグレードする方法については、the section called “kubectl と eksctl のセットアップ” を参照してください。
- クラスター構成を含む既存の `kubectl config` ファイル。`kubectl config` ファイルの作成については、the section called “kubectl を使用してクラスターにアクセスする” を参照してください。

ステップ 1: IAM ポリシーの作成

既存の IAM ポリシーを IAM ロールに関連付ける場合は、次のステップにスキップします。

1. IAM ポリシーを作成します。ポリシーを自作することも、必要となるアクセス権限のいくつかが既に付与されている AWS マネージドポリシーをコピーし、特定の要件に応じてカスタマイズすることもできます。詳細については、「IAM ユーザーガイド」の「IAM ポリシーの作成」を参照してください。
2. Pod にアクセスさせる AWS サービスの権限を含むファイルを作成します。すべての AWS サービスに対するアクションの全リストについては、「サービス認可リファレンス」を参照してください。

   次のコマンドを実行して、Amazon S3 バケットへの読み取り専用アクセスを許可するサンプルポリシーファイルを作成できます。必要に応じて、このバケットに設定情報またはブートストラップスクリプトを格納すると、Pod 内のコンテナがバケットからファイルを読み取り、アプリケーションにロードできます。このサンプルポリシーを作成する場合は、次のコンテンツをデバイスにコピーします。*my-pod-secrets-bucket* をバケット名に置き換え、コマンドを実行します。

   ```
   {
       "Version":"2012-10-17",
       "Statement": [
           {
               "Effect": "Allow",
               "Action": "s3:GetObject",
               "Resource": "arn:aws:s3:::my-pod-secrets-bucket"
           }
       ]
   }
   ```
3. IAM ポリシーを作成します。

   ```
   aws iam create-policy --policy-name my-policy --policy-document file://my-policy.json
   ```

ステップ 2: IAM ロールを作成して関連付ける

IAM ロールを作成し、Kubernetes サービスアカウントに関連付けます。`eksctl` または AWS CLI を使用できます。

ロールの作成と関連付け (eksctl)

この `eksctl` コマンドは指定された名前空間に Kubernetes サービスアカウントを作成し、指定された名前で (存在しない場合は) IAM ロールを作成し、既存の IAM ポリシー ARN をロールにアタッチ

し、サービスアカウントに IAM ロール ARN を注釈付けします。このコマンドのサンプルプレースホルダー値は、必ず特定の値に置き換えてください。eksctl をインストールまたはアップグレードするには、eksctl ドキュメントの「インストール」を参照してください。

```
eksctl create iamserviceaccount --name my-service-account --namespace default --cluster
 my-cluster --role-name my-role \
    --attach-policy-arn arn:aws:iam::111122223333:policy/my-policy --approve
```

> **Important**
>
> ロールまたはサービスアカウントが既に存在する場合、前のコマンドは失敗する可能性があります。eksctl には、そのような状況で使用できるさまざまなオプションがあります。詳細については、eksctl create iamserviceaccount --help を実行してください。

ロールの作成と関連付け (AWS CLI)

IAM ロールを引き受ける既存の Kubernetes サービスアカウントがある場合は、この手順を省略できます。

1. Kubernetes サービスアカウントを作成します。次のコンテンツをデバイスにコピーします。*my-service-account* を目的の名前に置き換え、必要に応じて *default* を別の名前空間に置き換えます。*default* を変更する場合、名前空間は既に存在している必要があります。

   ```
   cat >my-service-account.yaml <<EOF
   apiVersion: v1
   kind: ServiceAccount
   metadata:
     name: my-service-account
     namespace: default
   EOF
   kubectl apply -f my-service-account.yaml
   ```

2. 次のコマンドを使用して、AWS アカウント ID を環境変数に設定します。

   ```
   account_id=$(aws sts get-caller-identity --query "Account" --output text)
   ```

3. 次のコマンドを使用して、クラスターの OIDC ID プロバイダーを環境変数に設定します。*#####* の部分は自分のクラスター名に置き換えます。

```
oidc_provider=$(aws eks describe-cluster --name my-cluster --region $AWS_REGION --
query "cluster.identity.oidc.issuer" --output text | sed -e "s/^https:\/\///")
```

4. サービスアカウントの名前空間と名前の変数を設定します。*my-service-account* を、ロールを引き受けさせる Kubernetes サービスアカウントに置き換えます。*default* は、サービスアカウントの名前空間に置き換えます。

```
export namespace=default
export service_account=my-service-account
```

5. IAM ロール用の信頼ポリシーファイルを作成するには、次のコマンドを実行します。名前空間内のすべてのサービスアカウントにロールの使用を許可する場合は、次の内容をデバイスにコピーします。*StringEquals* を StringLike に置き換え、*$service_account* を * に置き換えます。以下の StringEquals または StringLike 条件に複数のエントリを追加して、複数のサービスアカウントまたは名前空間がロールを引き受けられるようにできます。クラスターが属するアカウントとは異なる AWS アカウントのロールがロールを引き受けられるようにする方法については、the section called “クロスアカウント IAM” を参照してください。

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Federated": "arn:aws:iam::123456789012:oidc-provider/$oidc_provider"
      },
      "Action": "sts:AssumeRoleWithWebIdentity",
      "Condition": {
        "StringEquals": {
          "$oidc_provider:aud": "sts.amazonaws.com",
          "$oidc_provider:sub": "system:serviceaccount:$namespace:$service_account"
        }
      }
    }
  ]
}
```

6. ロールを作成します。*my-role* を IAM ロールの名前に置き換え、*my-role-description* をロールの説明に置き換えます。

```
aws iam create-role --role-name my-role --assume-role-policy-document file://trust-
relationship.json --description "my-role-description"
```

7. IAM ポリシーをロールにアタッチします。*my-role* を IAM ロールの名前に置き換え、*my-policy* を、作成した既存のポリシーの名前に置き換えます。

```
aws iam attach-role-policy --role-name my-role --policy-arn=arn:aws:iam::
$account_id:policy/my-policy
```

8. サービスアカウントに、サービスアカウントで引き受ける IAM ロールの Amazon リソースネーム (ARN) の注釈を付けます。*my-role* を既存の IAM ロールの名前に置き換えます。前の手順で、クラスターが属するアカウントとは異なる AWS アカウントのロールに、ロールの引き受けを許可したと仮定します。その場合、AWS アカウントおよび他のアカウントからのロールを必ず指定してください。詳細については、the section called “クロスアカウント IAM” を参照してください。

```
kubectl annotate serviceaccount -n $namespace $service_account eks.amazonaws.com/
role-arn=arn:aws:iam::$account_id:role/my-role
```

9. (オプション) サービスアカウントの AWS Security Token Service エンドポイントを設定します。AWS では、グローバルエンドポイントの代わりにリージョン AWS STS エンドポイントを使用することをお勧めしています。これにより、レイテンシーが減少し、組み込みの冗長性が提供され、セッショントークンの有効性が向上します。

ステップ 3: 設定を確認する

1. IAM ロールの信頼ポリシーが正しく設定されていることを確認します。

```
aws iam get-role --role-name my-role --query Role.AssumeRolePolicyDocument
```

出力例は次のとおりです。

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
```

```
                "Federated": "arn:aws:iam::111122223333:oidc-provider/oidc.eks.us-
east-1.amazonaws.com/id/EXAMPLED539D4633E53DE1B71EXAMPLE"
            },
            "Action": "sts:AssumeRoleWithWebIdentity",
            "Condition": {
                "StringEquals": {
                    "oidc.eks.us-east-1.amazonaws.com/id/
EXAMPLED539D4633E53DE1B71EXAMPLE:sub": "system:serviceaccount:default:my-service-
account",
                    "oidc.eks.us-east-1.amazonaws.com/id/
EXAMPLED539D4633E53DE1B71EXAMPLE:aud": "sts.amazonaws.com"
                }
            }
        }
    ]
}
```

2. 前の手順でロールにアタッチしたポリシーが、そのロールにアタッチされていることを確認します。

```
aws iam list-attached-role-policies --role-name my-role --query
 "AttachedPolicies[].PolicyArn" --output text
```

出力例は次のとおりです。

```
                arn:aws:iam::111122223333:policy/my-policy
```

3. 使用するポリシーの Amazon リソースネーム (ARN) を保存する変数を設定します。*my-policy* を、アクセス許可を確認するポリシーの名前に置き換えます。

```
export policy_arn=arn:aws:iam::111122223333:policy/my-policy
```

4. ポリシーのデフォルトバージョンを確認します。

```
aws iam get-policy --policy-arn $policy_arn
```

出力例は次のとおりです。

```
{
    "Policy": {
```

```
        "PolicyName": "my-policy",
        "PolicyId": "EXAMPLEBIOWGLDEXAMPLE",
        "Arn": "arn:aws:iam::111122223333:policy/my-policy",
        "Path": "/",
        "DefaultVersionId": "v1",
        [...]
    }
}
```

5. ポリシーの内容を表示して、Pod で必要な権限がすべて含まれていることを確認します。必要であれば、次のコマンドの *1* を、前の出力で返されたバージョンに置き換えます。

```
aws iam get-policy-version --policy-arn $policy_arn --version-id v1
```

出力例は次のとおりです。

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "s3:GetObject",
            "Resource": "arn:aws:s3:::my-pod-secrets-bucket"
        }
    ]
}
```

前の手順でサンプルポリシーを作成した場合、出力は同じになります。別のポリシーを作成した場合、*####* の内容は異なります。

6. Kubernetes サービスアカウントにロールが注釈されていることを確認します。

```
kubectl describe serviceaccount my-service-account -n default
```

出力例は次のとおりです。

```
Name:                my-service-account
Namespace:           default
Annotations:         eks.amazonaws.com/role-arn: arn:aws:iam::111122223333:role/my-
role
Image pull secrets:  <none>
```

```
Mountable secrets:   my-service-account-token-qqjfl
Tokens:              my-service-account-token-qqjfl
[...]
```

次のステップ

- the section called “Pod への割り当て”

Kubernetes サービスアカウントを使用するように Pod を設定するには

Pod が AWS サービスにアクセスする必要がある場合は、Kubernetes サービスアカウントを使用するように設定する必要があります。サービスアカウントは、AWS サービスにアクセスする権限がある AWS Identity and Access Management (IAM) ロールに関連付ける必要があります。

- 既存のクラスター。まだ所有していない場合は、はじめに でのガイドのいずれかを参照しながら作成できます。
- クラスターの既存 IAM OpenID Connect (OIDC) プロバイダー 既に所有中かどうかの確認、または作成方法については、the section called “IAM OIDC プロバイダー” を参照してください。
- IAM ロールに関連付けられている既存の Kubernetes サービスアカウント。サービスアカウントには、IAM ロールの Amazon リソースネーム (ARN) の注釈を付ける必要があります。ロールには、Pod が AWS サービスを使用するアクセス許可を含む IAM ポリシーが関連付けられている必要があります。サービスアカウントとロールの作成および設定方法については、the section called “IAM ロールを割り当てる” を参照してください。
- ご使用のデバイスまたは AWS CloudShell で、バージョン `2.12.3` 以降、または AWS コマンドラインインターフェイス (AWS CLI) のバージョン `1.27.160` 以降がインストールおよび設定されていること。現在のバージョンを確認するには、`aws --version | cut -d / -f2 | cut -d ' ' -f1` を参照してください。`yum`、`apt-get`、macOS 用の Homebrew などのパッケージマネージャーは、多くの場合 AWS CLI の最新バージョンより数バージョン古くなっています。最新バージョンをインストールするには「AWS コマンドラインインターフェイスユーザーガイド」の「インストール」および「aws configure を使用したクイック設定」を参照してください。AWS クラウドシェル にインストールされている AWS CLI バージョンも最新バージョンより数バージョン遅れることがあります。更新するには、「AWS CloudShell ユーザーガイド」の「ホームディレクトリへの AWS CLI のインストール」を参照してください。
- デバイスまたは AWS CloudShell に、`kubectl` コマンドラインツールがインストールされていること。バージョンはご使用のクラスターの Kubernetes バージョンと同じか、1 つ前のマイナーバージョン以前、あるいはそれより新しいバージョンが使用できます。例えば、クラスターの

バージョンが 1.29 である場合、kubectl のバージョン 1.28、1.29、または 1.30 が使用できます。kubectl をインストールまたはアップグレードする方法については、the section called “kubectl と eksctl のセットアップ” を参照してください。

- クラスター構成を含む既存の kubectl config ファイル。kubectl config ファイルの作成については、the section called “kubectl を使用してクラスターにアクセスする” を参照してください。

1. 次コマンドを使用して、Pod をデプロイして設定を確認できるデプロイマニフェストを作成します。example の値は独自の値に置き換えます。

```
cat >my-deployment.yaml <<EOF
apiVersion: apps/v1
kind: Deployment
metadata:
  name: my-app
spec:
  selector:
    matchLabels:
      app: my-app
  template:
    metadata:
      labels:
        app: my-app
    spec:
      serviceAccountName: my-service-account
      containers:
      - name: my-app
        image: public.ecr.aws/nginx/nginx:X.XX
EOF
```

2. マニフェストをクラスターにデプロイします。

```
kubectl apply -f my-deployment.yaml
```

3. Pod に必要な環境変数が存在することを確認してください。

   a. 前の手順のデプロイ時にデプロイされた Pod を確認します。

```
kubectl get pods | grep my-app
```

   出力例は次のとおりです。

```
my-app-6f4dfff6cb-76cv9   1/1     Running   0          3m28s
```

b. Pod が使用している IAM ロールの ARN を確認します。

```
kubectl describe pod my-app-6f4dfff6cb-76cv9 | grep AWS_ROLE_ARN:
```

出力例は次のとおりです。

```
AWS_ROLE_ARN:                 arn:aws:iam::111122223333:role/my-role
```

ロール ARN は、既存のサービスアカウントに注釈を付けたロール ARN と一致する必要があります。サービスアカウントへの注釈付けの詳細については、「the section called “IAM ロールを割り当てる”」を参照してください。

c. Pod にウェブ ID トークンファイルのマウントがあることを確認します。

```
kubectl describe pod my-app-6f4dfff6cb-76cv9 | grep AWS_WEB_IDENTITY_TOKEN_FILE:
```

出力例は次のとおりです。

```
AWS_WEB_IDENTITY_TOKEN_FILE:  /var/run/secrets/eks.amazonaws.com/serviceaccount/
token
```

kubelet が Pod に代わってトークンをリクエストして格納します。デフォルトで kubelet は、トークンが合計有効期限の 80% を超えている場合、またはトークンが 24 時間を超えている場合、そのトークンを更新します。Pod 仕様の設定を使用して、デフォルトのサービスアカウントを除くすべてのアカウントの有効期限を変更できます。詳細については、Kubernetes ドキュメントの「Service Account Token Volume Projection (サービスアカウントトークンボリュームのプロジェクション)」を参照してください。

クラスターの Amazon EKS Pod Identity ウェブフックは、次の注釈が付いたサービスアカウントを使用している Pod をモニタリングします。

```
eks.amazonaws.com/role-arn: arn:aws:iam::111122223333:role/my-role
```

このウェブフックは、そうした Pod に以前の環境変数を適用します。クラスターでは、環境変数とトークンファイルのマウントを設定するために、ウェブフックを使用する必要はありません。これらの環境変数を保持するよう、Pod を手動で設定できます。AWS SDK のサポートされているバージョンは、最初に認証情報チェーンプロバイダーでこれらの環境変数を探します。ロールの認証情報は、この条件を満たす Pod で使用されます。

4. ロールにアタッチされた IAM ポリシーで割り当てたアクセス許可を使用して、Pod が AWS サービスとやり取りできることを確認します。

> **Note**
>
> サービスアカウントに関連付けられた IAM ロールの AWS 認証情報を Pod が使用する場合、AWS CLI またはその Pod のコンテナ内にある他の SDK は、そのロールから提供される認証情報を使用します。Amazon EKS ノードの IAM ロールに提供された認証情報へのアクセスを制限しない場合、ポッドは引き続きこれらの認証情報にアクセスできます。詳細については「ワーカーノードに割り当てられたインスタンスプロファイルへのアクセスを制限する」を参照してください。

Pod が想定通りにサービスとやり取りできない場合は、次の手順を実行して、すべてが正しく設定されていることを確認してください。

a. OpenID Connect ウェブアイデンティティトークンファイルを介した IAM ロールの引き受けをサポートする AWS SDK バージョンを Pod が使用していることを確認します。詳細については、the section called “サポートされている SDK” を参照してください。

b. デプロイがサービスアカウントを使用していることを確認します。

```
kubectl describe deployment my-app | grep "Service Account"
```

出力例は次のとおりです。

```
Service Account:  my-service-account
```

c. それでも Pod がサービスにアクセスできない場合は、「Kubernetes サービスアカウントに IAM ロールを割り当てる」で説明されている手順を参照して、ロールとサービスアカウントが正しく設定されていることを確認します。

サービスアカウントの AWS Security Token Service エンドポイントを設定する

Kubernetes サービスアカウントをサービスアカウント用の IAM ロールと共に使用している場合、そのサービスアカウントで使用される AWS Security Token Service エンドポイントのタイプを設定できます。

AWS では、グローバルエンドポイントの代わりに地域の AWS STS エンドポイントを使用することを推奨しています。これにより、レイテンシーが減少し、組み込みの冗長性が提供され、セッショントークンの有効性が向上します。Pod が実行中の AWS リージョンで、AWS Security Token Service がアクティブであること。さらに、AWS リージョン内のサービスに障害が発生した場合に別の AWS リージョンを使用できるよう、アプリケーションに冗長性が組み込まれている必要があります。詳細については、IAM ユーザーガイドの「AWS リージョンでの AWS STS の管理」を参照してください。

- 既存のクラスター。まだ所有していない場合は、はじめに でのガイドのいずれかを参照しながら作成できます。
- クラスター用の既存の IAM OIDC プロバイダー。詳細については、「the section called “IAM OIDC プロバイダー”」を参照してください。
- サービスアカウント用の Amazon EKS IAM 機能を使用するように設定された、既存の Kubernetes サービスアカウント。

> **Note**
>
> インターネットへの出力なしで IRSA 認証情報リクエストを VPC 内でプライベートにするには、AWS STS インターフェイスエンドポイント (`com.amazonaws.region-code.sts`) を作成し、このトピックで説明されているようにリージョンのエンドポイントを設定します。グローバルエンドポイント (`sts.amazonaws.com`) は VPC エンドポイントをバイパスします。また、クラスター OIDC 検出/JWKS エンドポイントにプライベートで到達するには (プロバイダーのセットアップとトークンの検証用)、別の VPC エンドポイント (`com.amazonaws.region-code.oidc-eks`) を作成します。詳細については、「the section called “AWS PrivateLink を使用してクラスター OIDC エンドポイントにアクセスする”」を参照してください。

次の例ではすべて、Amazon VPC CNI プラグインで使用される aws-node Kubernetes サービスアカウントを使用しています。*`example values`* は、ご自身のサービスアカウント、ポッド、名前空間、その他のリソースに置き換えることができます。

1. エンドポイントを変更したいサービスアカウントを使用する Pod を選択します。Pod を実行する AWS リージョンを決定します。*`aws-node-6mfgv`* を Pod 名に置き換え、*`kube-system`* をポッドの名前空間に置き換えます。

```
kubectl describe pod aws-node-6mfgv -n kube-system |grep Node:
```

出力例は次のとおりです。

```
ip-192-168-79-166.us-west-2/192.168.79.166
```

前の出力では、Pod は us-west-2 AWS リージョンのノードで実行されています。

2. Pod のサービスアカウントが使用しているエンドポイントタイプを確認します。

```
kubectl describe pod aws-node-6mfgv -n kube-system |grep AWS_STS_REGIONAL_ENDPOINTS
```

出力例は次のとおりです。

```
AWS_STS_REGIONAL_ENDPOINTS: regional
```

現在のエンドポイントがグローバルの場合、この出力には global が返されます。出力が返されない場合、デフォルトのエンドポイントタイプが、上書きされないまま使用されています。

3. サービスアカウントが使用するエンドポイントタイプは、次のいずれかのコマンドを使用して、デフォルトのタイプから別のタイプに変更できます。*aws-node* は、サービスアカウントの名前に置き換え、*kube-system* は、サービスアカウントの名前空間に置き換えます。

- デフォルトまたは現在のエンドポイントタイプがグローバルであり、それをリージョン別に変更する場合は、以下の手順を実行します。

  ```
  kubectl annotate serviceaccount -n kube-system aws-node eks.amazonaws.com/sts-regional-endpoints=true
  ```

  Pod のコンテナで実行されているアプリケーションで、事前署名された S3 URL を生成するためにサービスアカウントの IAM ロールを使用している場合、リージョン別エンドポイントの URL の形式は、次の例のようになります。

  ```
  https://bucket.s3.us-west-2.amazonaws.com/path?...&X-Amz-Credential=your-access-key-id/date/us-west-2/s3/aws4_request&...
  ```

- デフォルトまたは現在のエンドポイントタイプがリージョン別であり、それをグローバルに変更する場合は、次の手順を実行します。

```
kubectl annotate serviceaccount -n kube-system aws-node eks.amazonaws.com/sts-
regional-endpoints=false
```

アプリケーションが、AWS STS のグローバルエンドポイントに対し明示的にリクエストを行っており、Amazon EKS クラスターにおいて、リージョン別エンドポイントを使用する際のデフォルトの動作がオーバーライドされていない場合は、リクエストが失敗しエラーが返されます。詳細については、「the section called “ポッドコンテナは次のエラーを受け取ります: An error occurred (SignatureDoesNotMatch) when calling the GetCallerIdentity operation: Credential should be scoped to a valid region”」を参照してください。

Pod のコンテナで実行されているアプリケーションで、事前署名された S3 URL を生成するためにサービスアカウントの IAM ロールを使用している場合、グローバルエンドポイントの URL の形式は、次の例のようになります。

```
https://bucket.s3.amazonaws.com/path?...&X-Amz-Credential=your-access-key-id/date/
us-west-2/s3/aws4_request&...
```

使用している自動化処理で、事前署名付き URL に特定の形式を想定している場合、または事前署名付き URL を使用するアプリケーションやダウンストリームの依存関係に、ターゲットとして想定する AWS リージョンがある場合、適切な AWS STS エンドポイントを使用するために必要な変更を加えます。

4. 認証情報環境変数を適用するために、サービスアカウントに関連付けられている既存の Pod を削除して再作成します。変更するウェブフックは、既に実行されている Pod には適用されません *Pod*、*kube-system*、および *-l k8s-app=aws-node* を、アノテーションを設定する Pod の情報に置き換えることができます。

```
kubectl delete Pods -n kube-system -l k8s-app=aws-node
```

5. ポッドがすべて再起動したことを確認します。

```
kubectl get Pods -n kube-system -l k8s-app=aws-node
```

6. Pod のいずれかの環境変数を表示します。AWS_STS_REGIONAL_ENDPOINTS の値が、以前のステップで設定した値であることを確認します。

```
kubectl describe pod aws-node-kzbtr -n kube-system |grep AWS_STS_REGIONAL_ENDPOINTS
```

出力例は次のとおりです。

```
AWS_STS_REGIONAL_ENDPOINTS=regional
```

IRSA で別のアカウントに対して認証する

別のアカウントのクラスターから ID プロバイダーを作成するか、連鎖した AssumeRole オペレーションを使用することで、クロスアカウントの IAM アクセス許可を設定できます。次の例では、アカウント A は、サービスアカウントの IAM ロールをサポートする Amazon EKS クラスターを所有しています。そのクラスターで実行されている Pod は、アカウント B の IAM アクセス許可を引き受ける必要があります。

- オプション 1 の方がシンプルですが、アカウント B でアカウント A のクラスターの OIDC ID プロバイダーを作成および管理する必要があります。
- オプション 2 は、OIDC が常にアカウント A で管理されますが、AssumeRole を 2 つ呼び出してロールチェイニングを行う必要があります。

オプション 1: 別のアカウントのクラスターから ID プロバイダーを作成する

この例では、アカウント A はアカウント B にクラスターからの OpenID Connect (OIDC) 発行者 URL を提供します。アカウント B は、アカウント A のクラスターからの OIDC 発行者 URL を使用して、「クラスターの IAM OIDC プロバイダーを作成する」および「the section called “IAM ロールを割り当てる”」の手順に従います。次に、クラスター管理者は、アカウント A のクラスターのサービスアカウントに、アカウント B (*444455556666*) のロールを使用するように注釈を付けます。

```
apiVersion: v1
kind: ServiceAccount
metadata:
  annotations:
    eks.amazonaws.com/role-arn: arn:aws:iam::444455556666:role/account-b-role
```

オプション 2: 連鎖された **AssumeRole** オペレーションを使用する

このアプローチでは、アカウントごとに IAM ロールが作成されます。アカウント B のロールは、アカウント A を信頼します。アカウント A のロールは、OIDC フェデレーションを使用して、クラスターから認証情報を取得します。ポッドは、AWS CLI プロファイルを使用して、2 つのロールを連鎖させます。

ステップ 1: アカウント B にターゲットロールを作成する

アカウント B (*444455556666*) は、アカウント A のクラスター内のポッドで必要になるアクセス許可と共に IAM ロールを作成します。アカウント B は、このロールに目的のアクセス許可ポリシーをアタッチし、次の信頼ポリシーを追加します。

アカウント B のロール用の信頼ポリシー — アカウント A の特定の IRSA ロールがこのロールを引き受けることを許可します。

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "AWS": "arn:aws:iam::111122223333:root"
      },
      "Action": "sts:AssumeRole",
      "Condition": {}
    }
  ]
}
```

> **Important**
>
> 最小特権の場合、アカウントルート (`arn:aws:iam::111122223333:root`) を使用するのではなく、`Principal` ARN をアカウント A の特定のロール ARN に置き換えます。アカウントルートを使用すると、アカウント A のいずれの IAM プリンシパルもこのロールを引き受けることができます。

ステップ 2: アカウント A に IRSA ロールを作成する

アカウント A (*111122223333*) では信頼ポリシーを定義したロールを作成し、その信頼ポリシーではクラスターの OIDC 発行者アドレスで作成された ID プロバイダーから認証情報を取得することを許可します。

アカウント A のロール (OIDC フェデレーション) 用の信頼ポリシー — EKS クラスターの OIDC プロバイダーがこのロールの認証情報を発行することを許可します。

```
{
```

```
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Federated": "arn:aws:iam::111122223333:oidc-provider/oidc.eks.us-
east-1.amazonaws.com/id/EXAMPLED539D4633E53DE1B71EXAMPLE"
      },
      "Action": "sts:AssumeRoleWithWebIdentity",
      "Condition": {
        "StringEquals": {
          "oidc.eks.us-east-1.amazonaws.com/id/EXAMPLED539D4633E53DE1B71EXAMPLE:aud":
 "sts.amazonaws.com"
        }
      }
    }
  ]
}
```

Important

最小特権の場合、sub クレームの StringEquals 条件を追加して、このロールを特定の Kubernetes サービスアカウントに制限します。sub 条件がない場合、クラスター内のいずれのサービスアカウントもこのロールを引き受けることができます。sub 値では、system:serviceaccount:*NAMESPACE*:*SERVICE_ACCOUNT_NAME* という形式を使用します。例えば、default 名前空間で my-service-account という名前のサービスアカウントに制限するには、次のようにします。

```
"oidc.eks.region-code.amazonaws.com/id/EXAMPLED539D4633E53DE1B71EXAMPLE:sub":
 "system:serviceaccount:default:my-service-account"
```

ステップ 3: アカウント A のロールに AssumeRole アクセス許可をアタッチする

アカウント A は、ステップ 2 で作成されたロールにアクセス許可ポリシーをアタッチします。このポリシーにより、ロールがアカウント B のロールを引き受けることが許可されます。

アカウント A のロール用のアクセス許可ポリシー — アカウント B のターゲットロールに sts:AssumeRole を付与します。

```
{
```

```
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "sts:AssumeRole",
            "Resource": "arn:aws:iam::444455556666:role/account-b-role"
        }
    ]
}
```

ステップ 4: ロールを連鎖するようにポッドを設定する

アカウント B のロールを引き受ける Pod のアプリケーションコードは、account_b_role と account_a_role の 2 つのプロファイルを使用します。account_b_role プロファイルでは、ソースとして account_a_role プロファイルを使用します。AWS CLI では、~/.aws/config ファイルは以下のようになります。

```
[profile account_b_role]
source_profile = account_a_role
role_arn=arn:aws:iam::444455556666:role/account-b-role

[profile account_a_role]
web_identity_token_file = /var/run/secrets/eks.amazonaws.com/serviceaccount/token
role_arn=arn:aws:iam::111122223333:role/account-a-role
```

他の AWS SDK の連鎖されたプロファイルを指定するには、使用する SDK のドキュメントを参照してください。詳細については、「AWS で構築するツール」を参照してください。

AWS SDK で IRSA を使用する

認証情報の使用

サービスアカウントの IAM ロール (IRSA) の認証情報を使用するために、コードで任意の AWS SDK を使用して SDK を含む AWS サービスのクライアントを作成できます。デフォルトでは、SDK は使用する AWS Identity and Access Management 認証情報を一連の場所で検索します。クライアントの作成時や SDK の初期化時に認証情報プロバイダーを指定しなかった場合は、サービスアカウントの認証情報用の IAM ロールが使用されます。

これがうまくいくのは、サービスアカウント用の IAM ロールがデフォルトの認証情報チェーンのステップとして追加されたからです。現在、ワークロードが認証情報チェーンの前にある認証情報を使

用している場合は、同じワークロードのサービスアカウントに IAM ロールを設定しても、その認証情報は引き続き使用されます。

SDK は、`AssumeRoleWithWebIdentity` アクションを使用してサービスアカウント OIDC トークンを AWS Security Token Service からの一時的な認証情報と自動的に交換します。Amazon EKS とこの SDK アクションは、有効期限が切れる前に一時認証情報を更新することで、引き続きローテーションを行います。

サービスアカウントの IAM ロールを使用すると、Pod 内のコンテナは、OpenID Connect ウェブ ID トークンファイルを介した IAM ロールの引き受けをサポートする AWS SDK バージョンを使用する必要があります。お使いの AWS SDK には、必ず次のバージョン以降を使用してください。

- Java (バージョン 2) – 2.10.11
- Java – 1.12.782
- AWS SDK for Go v1 – 1.23.13
- AWS SDK for Go v2 – すべてのバージョンがサポートされています
- Python (Boto3) – 1.9.220
- Python (botocore) – 1.12.200
- AWS CLI – 1.16.232
- ノード - 2.525.0 と 3.27.0
- Ruby – 3.58.0
- C++ – 1.7.174
- .NET - 3.3.659.1 - `AWSSDK.SecurityToken` も含めることを確認します。
- PHP – 3.110.7

Cluster Autoscaler、AWS Load Balancer Controller を使用してインターネットトラフィックをルーティングする、Amazon VPC CNI plugin for Kubernetes など、一般的な Kubernetes アドオンの多くは、サービスアカウントの IAM ロールをサポートしています。

サポートされている SDK を使用していることを確認するには、コンテナを構築する際に、「AWS での構築ツール」で、希望する SDK のインストール手順に従ってください。

考慮事項

Java

Java を使用する場合は、クラスパスに sts を含める必要があります。詳細については、Java SDK ドキュメントの「WebIdentityTokenFileCredentialsProvider」を参照してください。

OIDC トークンを検証するための署名キーを取得する

Kubernetes は、ProjectedServiceAccountToken を各 Kubernetes サービスアカウントに発行します。このトークンは OIDC トークンであり、さらに JSON ウェブトークン (JWT) の一種でもあります。Amazon EKS は、外部システムでトークンを検証できるように、トークンの署名キーを含むクラスターごとにパブリック OIDC エンドポイントをホストします。

ProjectedServiceAccountToken を検証するには、JSON ウェブキーセット (JWKS) とも呼ばれる OIDC パブリック署名キーを取得する必要があります。アプリケーションでこれらのキーを使用してトークンを検証します。例えば、PyJWT Python ライブラリを使用して、これらのキーを使用してトークンを検証できます。ProjectedServiceAccountToken の詳細については、「the section called “IAM、Kubernetes、OpenID Connect (OIDC) の背景情報”」を参照してください。

> **Note**
>
> バリデーターがアウトバウンドインターネットアクセスなしで VPC で実行されている場合、クラスター OIDC エンドポイント (com.amazonaws.region-code.oidc-eks) の AWS PrivateLink インターフェイスエンドポイントを作成することで、検出ドキュメントと JWKS をプライベートに取得できます。詳細については、「the section called “AWS PrivateLink を使用してクラスター OIDC エンドポイントにアクセスする”」を参照してください。

前提条件

- クラスターの既存の AWS Identity and Access Management (IAM) OpenID Connect (OIDC) プロバイダー。既に存在しているかどうかを確認する、または作成するには、the section called “IAM OIDC プロバイダー” を参照してください。
- AWS CLI – Amazon EKS など AWS のサービスを操作するためのコマンドラインツールです。詳細については、AWS コマンドラインインターフェイスユーザーガイドの「インストール」を参照してください。AWS CLI のインストール後は設定も行っておくことをお勧めします。詳細につい

ては、AWS コマンドラインインターフェイスユーザーガイドの「aws configure を使用したクイック設定」を参照してください。

手順

1. AWS CLI を使用して、Amazon EKS クラスターの OIDC URL を取得します。

```
$ aws eks describe-cluster --name my-cluster --query 'cluster.identity.oidc.issuer'
"https://oidc.eks.us-west-2.amazonaws.com/id/8EBDXXXX00BAE"
```

2. curl または同様のツールを使用して、パブリック署名キーを取得します。結果は JSON ウェブキーセット (JWKS) です。

> **Important**
>
> Amazon EKS は、OIDC エンドポイントへの呼び出しをスロットリングします。パブリック署名キーをキャッシュする必要があります。レスポンスに含まれる `cache-control` ヘッダーを考慮します。

> **Important**
>
> Amazon EKS は、OIDC 署名キーを 7 日ごとにローテーションします。

```
$ curl https://oidc.eks.us-west-2.amazonaws.com/id/8EBDXXXX00BAE/keys
{"keys":
[{"kty":"RSA","kid":"2284XXXX4a40","use":"sig","alg":"RS256","n":"wklbXXXXMVfQ","e":"AQAB"}]
```

EKS Pod Identity が Pod に AWS サービスへのアクセス権を付与する仕組みを学ぶ

ポッドのコンテナ内のアプリケーションは AWS SDK または AWS CLI で、AWS Identity and Access Management (IAM) アクセス許可を使用した AWS サービスへの API リクエストを行うことができます。アプリケーションは AWS 認証情報で AWS API リクエストに署名する必要があります。

EKS Pod Identity は、Amazon EC2 インスタンスプロファイルから Amazon EC2 インスタンスに認証情報を提供する場合と同じような方法で、アプリケーションの認証情報を管理する機能があります。AWS 認証情報を作成してコンテナに配布したり、Amazon EC2 インスタンスのロールを使用したりする必要はありません。IAM ロールを Kubernetes サービスアカウントと関連付けて、サービスアカウントを使用するように Pod を設定できます。

各 EKS Pod Identity の関連付けは、指定されたクラスター内のネームスペース内のサービスアカウントにロールをマップします。複数のクラスターに同じアプリケーションがある場合、ロールの信頼ポリシーを変更することなく、各クラスターで同一の関連付けを行うことができます。

ポッドが関連付けられたサービスアカウントを使用する場合、Amazon EKS はポッドのコンテナに環境変数を設定します。環境変数は、AWS CLI を含む AWS SDK が EKS Pod Identity の認証情報を使用するように設定します。

EKS Pod Identity の利点

EKS Pod Identity には次の利点があります。

- 最小特権 – IAM アクセス許可の範囲をサービスアカウントに設定すると、そのサービスアカウントを使用する Pod にのみそのアクセス許可を付与できます。また、この機能により、`kiam` や `kube2iam` などのサードパーティーのソリューションが不要になります。

- 認証情報の分離 – Amazon EC2 インスタンスメタデータサービス (IMDS) へのアクセスが制限されている場合、ポッド内のコンテナは、そのコンテナが使用するサービスアカウントに関連付けられている IAM ロールの認証情報のみを取得できます。コンテナは、他の Pod 内のコンテナで使われている認証情報にアクセスすることはできません。IMDS が制限されていない場合、ポッドのコンテナは「Amazon EKS ノード IAM ロール」にもアクセスでき、コンテナは同じノード上にある他のポッドの IAM ロールの認証情報にアクセスできる可能性があります。詳細については「ワーカーノードに割り当てられたインスタンスプロファイルへのアクセスを制限する」を参照してください。

> **Note**
>
> `hostNetwork: true` で設定されたポッドは常に IMDS にアクセスできますが、AWS SDK および CLI は、有効である場合は Pod Identity 認証情報を使用します。

- 監査性 – 遡及的な監査を確実に行うため、AWS CloudTrail を介してアクセスとイベントのロギングを利用できます。

> **Important**
>
> コンテナはセキュリティ境界ではなく、Pod Identity を使用してもこの点は変わりません。同じノードに割り当てられたポッドはカーネルを共有し、ポッド設定によっては他のリソースも共有される可能性があります。別々のノードで実行されているポッドはコンピューティングレイヤーで分離されますが、Kubernetes API に対し個別のインスタンスの範囲を超えて追加のアクセス許可を持つノードアプリケーションがあります。例としては `kubelet`、`kube-proxy`、CSI ストレージドライバー、ユーザー独自の Kubernetes アプリケーションなどがあります。

EKS Pod Identity は、OIDC ID プロバイダーを使用しないため、the section called “IRSA による認証情報” よりも簡単な方法です。EKS Pod Identity には次の機能強化があります。

- 独立オペレーション – 多くの組織では、OIDC ID プロバイダーの作成を Kubernetes クラスターの管理とは別のチームが担当しています。EKS Pod Identity には明確な役割分担があり、EKS Pod Identity の関連付けの設定はすべて Amazon EKS で行われ、IAM アクセス許可の設定はすべて IAM で行われます。

- 再利用性 – EKS Pod Identity は、サービスアカウントの IAM ロールが使用するクラスターごとの個別のプリンシパルの代わりに、単一の IAM プリンシパルを使用します。IAM 管理者は以下のプリンシパルを任意のロールの信頼ポリシーに追加して、EKS Pod Identity で使用できるようにします。

```
            "Principal": {
                "Service": "pods.eks.amazonaws.com"
            }
```

- スケーラビリティ – 一時的な認証情報の各セットは、各ポッドで実行する各 AWS SDK ではなく、EKS Pod Identity 内の EKS Auth サービスによって引き継がれます。次に、各ノードで実行される Amazon EKS Pod Identity エージェントが SDK に認証情報を発行します。そのため、負荷はノードごとに 1 回に減り、各ポッドで重複することはありません。詳細については、「the section called “仕組み”」を参照してください。

この 2 つの選択肢を比較する詳細については、「the section called “AWS へのワークロードアクセス”」を参照してください。

EKS Pod Identity のセットアップの概要

以下の手順を実行して EKS Pod Identity を有効にします。

1. the section called “エージェントのセットアップ” — この手順は、クラスターごとに 1 回だけ実行します。クラスターで EKS 自動モードが有効になっている場合は、このステップを完了する必要はありません。
2. the section called “IAM ロールを割り当てる” — この手順は、アプリケーションに付与する固有の権限セットごとに実行します。
3. the section called “Pod サービスアカウント” — この手順を AWS サービスへのアクセスが必要な Pod ごとに実行します。
4. the section called “サポートされている SDK” – ワークロードがサポートされているバージョンの AWS SDK を使用していること、およびワークロードがデフォルトの認証情報チェーンを使用していることを確認します。

制限

- IAM ロールを Kubernetes サービスアカウントにマッピングするため、クラスターごとに最大 5,000 個の EKS Pod Identity の関連付けがサポートされています。

考慮事項

- IAM ロールの関連付け: クラスター内の各 Kubernetes サービスアカウントは、クラスターと同じ AWS アカウントにある 1 つの IAM ロールに関連付けることができます。ロールを変更するには、EKS Pod Identity の関連付けを編集します。クロスアカウントアクセスの場合、IAM ロールを使用してロールへのアクセスを委任します。詳細については、「IAM ユーザーガイド」の「AWS アカウント間の IAM ロールを使用したアクセスの委任」を参照してください。
- EKS Pod Identity エージェント: EKS Pod Identity を使用するには、Pod Identity エージェントが必要です。このエージェントはクラスターノード上で Kubernetes `DaemonSet` として実行され、同じノード上のポッドにのみ認証情報を提供します。ノードの `hostNetwork` を使用し、リンクローカルアドレス (IPv4 の場合は `169.254.170.23`、IPv6 の場合は `[fd00:ec2::23]`) でポート `80` および `2703` を占有します。クラスターで IPv6 が無効になっている場合は、Pod Identity エージェントの IPv6 を無効にしてください。詳細については、「EKS Pod Identity エージェントで IPv6 を無効にする」を参照してください。
- 結果的な整合性: EKS Pod Identity の関連付けは結果整合性を持ち、API コール後に整合性が確保されるまで数秒の遅延が発生する可能性があります。重要な高可用性コードパスで関連付けを作成または更新することは避けてください。代わりに、そのようなアクションは別々の、より頻度の低い初期化ルーチンまたはセットアップルーチンで実行します。詳細については、「EKS ベストプラクティスガイド」の「Security Groups Per Pod」を参照してください。
- プロキシとセキュリティグループの考慮事項: プロキシを使用するポッドの場合、`169.254.170.23` (IPv4) および `[fd00:ec2::23]` (IPv6) を `no_proxy/NO_PROXY` 環境変数に追加し、EKS Pod Identity エージェントへのリクエストの失敗を防ぎます。AWS VPC CNI でポッドのセキュリティグループを使用する場合、`ENABLE_POD_ENI` フラグを「true」に設定して `POD_SECURITY_GROUP_ENFORCING_MODE` フラグを「standard」に設定します。詳細については、「セキュリティグループを個別のポッドに割り当てる」を参照してください。

EKS Pod Identity クラスターバージョン

EKS Pod Identity を使用するには、クラスターのプラットフォームバージョンが次のテーブルに記載されているバージョン以上であるか、Kubernetes バージョンがテーブルに記載されているよりも新しいバージョンである必要があります。Kubernetes バージョンに対し推奨される Amazon EKS Pod Identity エージェントのバージョンを確認するには、「the section called “互換性を確認する”」を参照してください。

Kubernetes バージョン	プラットフォームバージョン
Kubernetes バージョンの記載がない	サポートされるプラットフォームのすべてのバージョン
1.28	eks.4

EKS Pod Identity の制限事項

EKS Pod Identity は以下で使用できます。

- 前のトピック the section called “EKS Pod Identity クラスターバージョン” に記載されている Amazon EKS クラスターバージョン。
- Linux Amazon EC2 インスタンスであるクラスターのワーカーノード。

EKS Pod Identity は以下では使用できません。

- AWS Outposts。
- Amazon EKS Anywhere
- Amazon EC2 で作成して実行する Kubernetes クラスター。EKS Pod Identity コンポーネントは Amazon EKS でのみ使用できます。

EKS Pod Identity は以下では使用できません。

- Linux Amazon EC2 インスタンス以外の場所で実行されるポッド。AWS Fargate (Fargate) 上で実行される Linux ポッドと Windows ポッドはサポートされていません。Windows Amazon EC2 インスタンスで実行されるポッドはサポートされていません。

EKS Pod Identity の仕組みを理解する

EKS Pod Identity の関連付けは、Amazon EC2 インスタンスプロファイルから Amazon EC2 インスタンスに認証情報を提供する場合と同じような方法で、アプリケーションの認証情報を管理する機能があります。

Amazon EKS Pod Identity は、追加の EKS Auth API と各ノードで実行されるエージェントポッドを使用して、ワークロードに認証情報を提供します。

Amazon EKS アドオン、セルフマネージドコントローラー、オペレーター、その他のアドオンなどのアドオンでは、作成者は最新の AWS SDK を使用するようにソフトウェアを更新する必要があります。EKS Pod Identity と Amazon EKS によって作成されたアドオンとの互換性のリストについては、前の the section called “EKS Pod Identity の制限事項” セクションを参照してください。

EKS Pod Identity をコードで使用する

コードでは、AWS SDK を使用して AWS サービスにアクセスできます。SDK を使用して AWS サービスのクライアントを作成するコードを記述すると、デフォルトで SDK は使用する AWS Identity and Access Management 認証情報を一連の場所で検索します。有効な認証情報が見つかると、検索は停止されます。使用されるデフォルトの場所について詳しくは、「AWS SDK and Tools リファレンスガイド」の「認証情報プロバイダーチェーン」を参照してください。

EKS Pod Identity がコンテナ認証情報プロバイダーに追加されました。このプロバイダーはデフォルトの認証情報チェーンのステップで検索されます。現在、ワークロードが認証情報チェーンの前にある認証情報を使用している場合、同じワークロードに EKS Pod Identity の関連付けを設定しても、それらの認証情報は引き続き使用されます。この方法では、古い認証情報を削除する前に、まず関連付けを作成することで、他の種類の認証情報から安全に移行できます。

コンテナ認証情報プロバイダーは、各ノードで実行されるエージェントからの一時的な認証情報を提供します。Amazon EKS では、エージェントは Amazon EKS Pod Identity エージェントであり、Amazon Elastic Container Service ではエージェントは `amazon-ecs-agent` です。SDK は環境変数を使用して接続するエージェントを検索します。

一方、サービスアカウント用の IAM ロールは、AWS SDK が `AssumeRoleWithWebIdentity` を使用して AWS Security Token Service と交換しなければならないウェブ ID トークンを提供します。

EKS Pod Identity エージェントが Pod とどのように連携するか

1. Amazon EKS が起動した新しい Pod で EKS Pod Identity 関連付けのあるサービスアカウントが使用されている場合、クラスターは次のコンテンツを Pod マニフェストに追加します。

```
    env:
    - name: AWS_CONTAINER_AUTHORIZATION_TOKEN_FILE
      value: "/var/run/secrets/pods.eks.amazonaws.com/serviceaccount/eks-pod-
identity-token"
    - name: AWS_CONTAINER_CREDENTIALS_FULL_URI
      value: "http://169.254.170.23/v1/credentials"
    volumeMounts:
    - mountPath: "/var/run/secrets/pods.eks.amazonaws.com/serviceaccount/"
```

```
      name: eks-pod-identity-token
  volumes:
  - name: eks-pod-identity-token
    projected:
      defaultMode: 420
      sources:
      - serviceAccountToken:
          audience: pods.eks.amazonaws.com
          expirationSeconds: 86400 # 24 hours
          path: eks-pod-identity-token
```

2. Kubernetes は Pod を実行するノードを選択します。次に、ノード上の Amazon EKS Pod Identity エージェントは AssumeRoleForPodIdentity アクションを使用して EKS Auth API から一時的な認証情報を取得します。
3. EKS Pod Identity エージェントは、コンテナ内で実行する AWS SDK でこれらの認証情報を利用できるようにします。
4. デフォルトの認証情報チェーンを使用する認証情報プロバイダーを指定しなくても、アプリケーションで SDK を使用できます。または、コンテナ認証情報プロバイダーを指定します。使用されるデフォルトの場所について詳しくは、「AWS SDK and Tools リファレンスガイド」の「認証情報プロバイダーチェーン」を参照してください。
5. SDK は環境変数を使用して EKS Pod Identity エージェントに接続し、認証情報を取得します。

> **Note**
>
> ワークロードが現在、認証情報チェーンの早い段階にある認証情報を使用している場合、同じワークロードに EKS Pod Identity の関連付けを設定しても、その認証情報は引き続き使用されます。

Amazon EKS Pod Identity エージェントのセットアップ

EKS Pod Identity の関連付けは、Amazon EC2 インスタンスプロファイルから Amazon EC2 インスタンスに認証情報を提供する場合と同じような方法で、アプリケーションの認証情報を管理する機能があります。

Amazon EKS Pod Identity は、追加の EKS Auth API と各ノードで実行されるエージェントポッドを使用して、ワークロードに認証情報を提供します。

> **Tip**
>
> EKS Auto Mode クラスターに EKS Pod Identity エージェントをインストールする必要はありません。この機能は EKS Auto Mode に組み込まれています。

考慮事項

- デフォルトでは、EKS Pod Identity Agent は EKS 自動モードクラスターにプリインストールされます。詳細については、EKS Auto Mode を参照してください。
- EKS Pod Identity エージェントはデフォルトでポッドが認証情報をリクエストするために IPv4 と IPv6 のアドレスをリッスンします。エージェントは、IPv4 のループバック (localhost) IP アドレス 169.254.170.23 と IPv6 の localhost IP アドレス [fd00:ec2::23] を使用します。
- IPv6 アドレスを無効にするか、または localhost IPv6 IP アドレスを禁止すると、エージェントは起動できません。IPv6 を使用できないノードでエージェントを起動するには、the section called “IPv6 を無効にする” の手順に従って IPv6 設定を無効にします。

Amazon EKS Pod Identity エージェントの作成

エージェントの前提条件

> **Important**
>
> EKS Pod Identity Agent が実行されるノードは、EKS Auth API にアクセスできる必要があります。ノードにプライベートサブネットを使用している場合は、EKS Auth API の AWS PrivateLink インターフェイスエンドポイントを設定する必要があります。詳細については、the section called “ AWS プライベートリンク” を参照してください。

- 既存の Amazon EKS クラスター。デプロイするには、はじめに を参照してください。クラスターバージョンとプラットフォームバージョンは、EKS Pod Identity クラスターバージョン に記載されているバージョン以降である必要があります。
- ノードロールには、エージェントが EKS Auth API で AssumeRoleForPodIdentity アクションを実行する権限があります。AWS マネージドポリシー: AmazonEKSWorkerNodePolicy を使用するか、次のようなカスタムポリシーを追加できます。

```
{
```

```
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "eks-auth:AssumeRoleForPodIdentity"
            ],
            "Resource": "*"
        }
    ]
}
```

このアクションをタグで制限して、エージェントを使用するポッドが引き受けることができるロールを制限できます。

- ノードは Amazon ECR にアクセスしてイメージをダウンロードできます。アドオンのコンテナイメージは、「Amazon EKS アドオンの Amazon コンテナイメージレジストリの表示」に記載されているレジストリにあります。

  なお、イメージの場所を変更して、AWS マネジメントコンソール の [オプションの設定] や AWS CLI の `--configuration-values` で EKS アドオンの `imagePullSecrets` を提供することができます。

AWS コンソールでエージェントを設定する

1. Amazon EKS コンソール を開きます。
2. 左のナビゲーションペインで、[クラスター] を選択し、次にアドオンを設定するEKS Pod Identity エージェントを選択します。
3. [アドオン] タブを選択してください。
4. [その他のアドオンを入手] を選択してください。
5. EKS Pod Identity のアドオンボックスの右上にあるボックスを選択し、[次へ] を選択します。
6. [選択したアドオン設定を構成する] ページの [バージョン] ドロップダウンリストで任意のバージョンを選択します。
7. (オプション) [オプションの設定] を展開して追加の設定を入力します。例えば、代替のコンテナイメージの場所と `ImagePullSecrets` を指定できます。許可されたキーのある JSON スキーマは、[アドオン設定スキーマ] に表示されます。

   設定キーと値を [設定値] に入力します。

8. [次へ] を選択します。
9. EKS Pod Identity がクラスター上で実行されていることを確認してください。

```
kubectl get pods -n kube-system | grep 'eks-pod-identity-agent'
```

出力例は次のとおりです。

```
eks-pod-identity-agent-gmqp7                                          1/1     Running
   1 (24h ago)   24h
eks-pod-identity-agent-prnsh                                          1/1     Running
   1 (24h ago)   24h
```

これで、クラスターで EKS Pod Identity の関連付けを使用できるようになりました。詳細については、the section called “IAM ロールを割り当てる” を参照してください。

AWS CLI でエージェントを設定する

1. 次の AWS CLI コマンドを実行します。`my-cluster` を自分のクラスター名に置き換えます。

```
aws eks create-addon --cluster-name my-cluster --addon-name eks-pod-identity-agent --
addon-version v1.0.0-eksbuild.1
```

> **Note**
>
> EKS Pod Identity エージェントは、サービスアカウントの IAM ロール 用の `service-account-role-arn` を使用しません。EKS Pod Identity エージェントにはノードロールの権限を付与する必要があります。

2. EKS Pod Identity がクラスター上で実行されていることを確認してください。

```
kubectl get pods -n kube-system | grep 'eks-pod-identity-agent'
```

出力例は次のとおりです。

```
eks-pod-identity-agent-gmqp7                                          1/1     Running
   1 (24h ago)   24h
```

```
eks-pod-identity-agent-prnsh                                    1/1     Running
   1 (24h ago)   24h
```

これで、クラスターで EKS Pod Identity の関連付けを使用できるようになりました。詳細については、the section called “IAM ロールを割り当てる” を参照してください。

IAM ロールを Kubernetes サービスアカウントに割り当てる

このトピックでは、EKS Pod Identity に関連付けられている AWS Identity and Access Management (IAM) ロールを引き受けるように Kubernetes サービスアカウントを設定する方法について説明します。任意の Pod はサービスアカウントを使用するように設定すると、ロールにアクセス許可がある AWS サービスすべてにアクセスできます。

EKS Pod Identity の関連付けを作成するには、1 つの手順しかありません。AWS マネジメントコンソール、AWS CLI、AWS SDK、AWS CCloudFormation などのツールを使用して EKS で関連付けを作成します。クラスター内の関連付けに関するデータやメタデータはどの Kubernetes オブジェクトにもなく、サービスアカウントに注釈を追加することもありません。

前提条件

- 既存のクラスター。まだ所有していない場合は、はじめに でのガイドのいずれかに従って作成できます。
- 関連付けを作成している IAM プリンシパルには `iam:PassRole` が必要です。
- ご使用のデバイスまたは AWS CloCloudShell に最新バージョンの AWS CLI がインストールおよび設定されていること。現在のバージョンは `aws --version | cut -d / -f2 | cut -d ' ' -f1` で確認できます。`yum`、`apt-get`、macOS 用の Homebrew などのパッケージマネージャーは、多くの場合 AWS CLI の最新バージョンより数バージョン古くなっています。最新バージョンをインストールするには、「AWS コマンドラインインターフェイスユーザーガイド」の「インストール」および「aws configure を使用したクイック設定」を参照してください。AWS CloudShell にインストールされている AWS CLI バージョンは、最新バージョンより数バージョン遅れている可能性もあります。更新するには、「AWS CloudShell ユーザーガイド」の「ホームディレクトリへの AWS CLI のインストール」を参照してください。
- デバイスまたは AWS CloudShell に、`kubectl` コマンドラインツールがインストールされていること。バージョンはご使用のクラスターの Kubernetes バージョンと同じか、1 つ前のマイナーバージョン以前、あるいはそれより新しいバージョンが使用できます。例えば、クラスターのバージョンが `1.29` である場合、`kubectl` のバージョン `1.28`、`1.29`、または `1.30` が使用で

きます。kubectl をインストールまたはアップグレードする方法については、the section called “kubectl と eksctl のセットアップ” を参照してください。

- クラスター構成を含む既存の kubectl config ファイル。kubectl config ファイルの作成については、the section called “kubectl を使用してクラスターにアクセスする” を参照してください。

Pod Identity の関連付けを作成する (AWS コンソール)

1. Amazon EKS コンソール を開きます。
2. 左のナビゲーションペインで、[クラスター] を選択し、次にアドオンを設定するEKS Pod Identity エージェントを選択します。
3. [アクセス] タブを選択します。
4. [Pod Identity の関連付け] で [作成] を選択します。
5. [IAM ロール] には、ワークロードに付与する権限を持つ IAM ロールを選択します。

> **Note**
>
> リストには、EKS Pod Identity による使用を許可する以下の信頼ポリシーを持つロールのみが含まれています。

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "AllowEksAuthToAssumeRoleForPodIdentity",
            "Effect": "Allow",
            "Principal": {
                "Service": "pods.eks.amazonaws.com"
            },
            "Action": [
                "sts:AssumeRole",
                "sts:TagSession"
            ]
        }
    ]
}
```

`sts:AssumeRole` - EKS Pod Identity は一時的な認証情報をポッドに渡す前に、`AssumeRole` を使用して IAM ロールを引き継ぎます。

`sts:TagSession` - EKS Pod Identity は、AWS STS へのリクエストに セッションタグ を含めるために `TagSession` を使用します。

これらのタグを信頼ポリシーの 条件キー で使用して、このロールを使用できるサービスアカウント、名前空間、およびクラスターを制限できます。

Amazon EKS 条件キーのリストについては「サービス認可リファレンス」の「Amazon エラスティックKubernetesサービス によって定義された条件」を参照してください。条件キーを使用できるアクションとリソースについては「Amazon エラスティックKubernetesサービス で定義されるアクション」を参照してください。

6. [Kubernetes 名前空間] では、サービスアカウントとワークロードが含まれている Kubernetes 名前空間を選択します。オプションで、クラスターに存在しない名前で名前空間名で指定できます。
7. Kubernetes サービスアカウント では、使用する Kubernetes サービスアカウントを選択します。Kubernetes ワークロードのマニフェストには、このサービスアカウントを指定する必要があります。オプションで、クラスターに存在しないサービスアカウントを名前で指定できます。
8. (オプション) [セッションタグを無効にする] を選択することで、Pod Identity がロールを引き受けるときに自動的に追加するデフォルトのセッションタグを無効にできます。
9. (オプション) [セッションポリシーの設定] を切り替えることで、IAM ロールにアタッチされた IAM ポリシーで定義されているアクセス許可以外の追加制限を、この Pod Identity 関連付けに適用するように IAM ポリシーを設定できます。

> **Note**
>
> セッションポリシーは、[セッションタグを無効にする] の設定がチェックされている場合にのみ適用できます。

10. (オプション) [タグ] で [タグを追加] を選択し、キーと値のペアにメタデータを追加します。これらのタグは関連付けに適用され、IAM ポリシーで使用できます。

    このステップでは、複数のリージョンを追加できます。
11. [作成] を選択します。

Pod Identity の関連付けを作成する (AWS CLI)

1. 既存の IAM ポリシーを IAM ロールに関連付ける場合は、次のステップにスキップします。

   IAM ポリシーを作成します。ポリシーを自作することも、必要となるアクセス権限のいくつかが既に付与されている AWS マネージドポリシーをコピーし、特定の要件に応じてカスタマイズすることもできます。詳細については、「IAM ユーザーガイド」の「IAM ポリシーの作成」を参照してください。

   a. Pod にアクセスさせる AWS サービスの権限を含むファイルを作成します。すべての AWS サービスに対するアクションの全リストについては、「サービス認可リファレンス」を参照してください。

      次のコマンドを実行して、Amazon S3 バケットへの読み取り専用アクセスを許可するサンプルポリシーファイルを作成できます。必要に応じて、このバケットに設定情報またはブートストラップスクリプトを格納すると、Pod 内のコンテナがバケットからファイルを読み取り、アプリケーションにロードできます。このサンプルポリシーを作成する場合は、次のコンテンツをデバイスにコピーします。*my-pod-secrets-bucket* をバケット名に置き換え、コマンドを実行します。

      ```
      {
          "Version":"2012-10-17",
          "Statement": [
              {
                  "Effect": "Allow",
                  "Action": "s3:GetObject",
                  "Resource": "arn:aws:s3:::my-pod-secrets-bucket"
              }
          ]
      }
      ```

   b. IAM ポリシーを作成します。

      ```
      aws iam create-policy --policy-name my-policy --policy-document file://my-
      policy.json
      ```

2. IAM ロールを作成し、Kubernetes サービスアカウントに関連付けます。

   a. IAM ロールを引き受ける既存の Kubernetes サービスアカウントがある場合は、この手順を省略できます。

Kubernetes サービスアカウントを作成します。次のコンテンツをデバイスにコピーします。*my-service-account* を目的の名前に置き換え、必要に応じて *default* を別の名前空間に置き換えます。*default* を変更する場合、名前空間は既に存在している必要があります。

```
cat >my-service-account.yaml <<EOF
apiVersion: v1
kind: ServiceAccount
metadata:
  name: my-service-account
  namespace: default
EOF
kubectl apply -f my-service-account.yaml
```

以下のコマンドを実行してください。

```
kubectl apply -f my-service-account.yaml
```

b. IAM ロール用の信頼ポリシーファイルを作成するには、次のコマンドを実行します。

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "AllowEksAuthToAssumeRoleForPodIdentity",
            "Effect": "Allow",
            "Principal": {
                "Service": "pods.eks.amazonaws.com"
            },
            "Action": [
                "sts:AssumeRole",
                "sts:TagSession"
            ]
        }
    ]
}
```

c. ロールを作成します。*my-role* を IAM ロールの名前に置き換え、*my-role-description* をロールの説明に置き換えます。

```
aws iam create-role --role-name my-role --assume-role-policy-document file://
trust-relationship.json --description "my-role-description"
```

d. IAM ポリシーをロールにアタッチします。*my-role* を IAM ロールの名前に置き換え、*my-policy* を、作成した既存のポリシーの名前に置き換えます。

```
aws iam attach-role-policy --role-name my-role --policy-
arn=arn:aws:iam::111122223333:policy/my-policy
```

> **Note**
>
> サービスアカウントの IAM ロールとは異なり、EKS Pod Identity はサービスアカウントの注釈を使用しません。

e. 次のコマンドを実行して、関連付けを作成します。my-cluster をクラスターの名前で置き換え、*my-service-account* を目的の名前に置き換え、必要に応じて *default* を別の名前空間に置き換えます。

```
aws eks create-pod-identity-association --cluster-name my-cluster --role-arn
 arn:aws:iam::111122223333:role/my-role --namespace default --service-account my-
service-account
```

出力例は次のとおりです。

```
{
    "association": {
        "clusterName": "my-cluster",
        "namespace": "default",
        "serviceAccount": "my-service-account",
        "roleArn": "arn:aws:iam::111122223333:role/my-role",
        "associationArn": "arn:aws::111122223333:podidentityassociation/my-
cluster/a-abcdefghijklmnop1",
        "associationId": "a-abcdefghijklmnop1",
        "tags": {},
        "createdAt": 1700862734.922,
        "modifiedAt": 1700862734.922
    }
}
```

> **Note**
>
> 名前空間とサービスアカウントは、クラスターには存在しない名前で指定できます。EKS Pod Identity の関連付けを機能させるには、名前空間、サービスアカウント、およびサービスアカウントを使用するワークロードを作成する必要があります。

設定の確認

1. IAM ロールの信頼ポリシーが正しく設定されていることを確認します。

```
aws iam get-role --role-name my-role --query Role.AssumeRolePolicyDocument
```

出力例は次のとおりです。

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "Service": "pods.eks.amazonaws.com"
            },
            "Action": [
                "sts:AssumeRole",
                "sts:TagSession"
            ]
        }
    ]
}
```

2. 前の手順でロールにアタッチしたポリシーが、そのロールにアタッチされていることを確認します。

```
aws iam list-attached-role-policies --role-name my-role --query
 'AttachedPolicies[].PolicyArn' --output text
```

出力例は次のとおりです。

```
                arn:aws:iam::111122223333:policy/my-policy
```

3. 使用するポリシーの Amazon リソースネーム (ARN) を保存する変数を設定します。*my-policy* を、アクセス許可を確認するポリシーの名前に置き換えます。

```
export policy_arn=arn:aws:iam::111122223333:policy/my-policy
```

4. ポリシーのデフォルトバージョンを確認します。

```
aws iam get-policy --policy-arn $policy_arn
```

出力例は次のとおりです。

```
{
    "Policy": {
        "PolicyName": "my-policy",
        "PolicyId": "EXAMPLEBIOWGLDEXAMPLE",
        "Arn": "arn:aws:iam::111122223333:policy/my-policy",
        "Path": "/",
        "DefaultVersionId": "v1",
        [...]
    }
}
```

5. ポリシーの内容を表示して、Pod で必要な権限がすべて含まれていることを確認します。必要であれば、次のコマンドの *1* を、前の出力で返されたバージョンに置き換えます。

```
aws iam get-policy-version --policy-arn $policy_arn --version-id v1
```

出力例は次のとおりです。

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "s3:GetObject",
            "Resource": "arn:aws:s3:::my-pod-secrets-bucket"
        }
```

```
    ]
}
```

前の手順でサンプルポリシーを作成した場合、出力は同じになります。別のポリシーを作成した場合、#### の内容は異なります。

次のステップ

the section called “Pod サービスアカウント”

EKS Pod Identity のターゲット IAM ロールを使用して AWS リソースにアクセスする

Amazon Elastic Kubernetes Service (Amazon EKS) でアプリケーションを実行する場合、異なる AWS アカウントに存在する AWS リソースへのアクセスが必要になることがあります。このガイドでは、EKS Pod Identity を使用してクロスアカウントアクセスを設定する方法を示します。これにより、Kubernetes ポッドからターゲットロールを使用して他の AWS リソースにアクセスできるようになります。

前提条件

作業を開始する前に、次の手順が完了していることを確認してください。

- Amazon EKS Pod Identity エージェントのセットアップ
- EKS Pod Identity ロールの作成

仕組み

Pod Identity を使用すると、EKS クラスター内のアプリケーションから、ロールチェーンと呼ばれるプロセスを通じてアカウント間の AWS リソースにアクセスできます。

Pod Identity の関連付けを作成する際は、EKS クラスターと同じアカウント内にある EKS Pod Identity ロールと、アクセスしたいユーザーの AWS リソース (S3 バケットや RDS データベースなど) が含まれたアカウントのターゲット IAM ロールという 2 つの IAM ロールを指定できます。IAM PassRole の要件により、EKS Pod Identity ロールは EKS クラスターのアカウント内に存在する必要がありますが、ターゲット IAM ロールは任意の AWS アカウントに存在できます。PassRole を使用すると、AWS エンティティはロールの引き受けを別のサービスに委任できます。EKS Pod Identity は PassRole を使用してロールを Kubernetes サービスアカウントに接続し、ロールとそれを渡す ID の両方が EKS クラスターと同じ AWS アカウントに存在する必要があります。アプリケーションポッドから AWS リソースにアクセスする必要がある場合、ポッドで Pod Identity の認証情報が要求

されます。次に、Pod Identity は 2 つのロールの引き受けを順番に自動実行します。つまり、最初に EKS Pod Identity ロールを引き受け、次にその認証情報を使用してターゲット IAM ロールを引き受けます。このプロセスにより、ターゲットロールで定義されたアクセス許可を含む一時的な認証情報がポッドに提供され、他の AWS アカウントのリソースへの安全なアクセスが実現します。

キャッシュに関する考慮事項

キャッシュメカニズムにより、既存の Pod Identity の関連付けで IAM ロールを更新しても、EKS クラスターで実行されているポッドですぐに有効にならない場合があります。Pod Identity エージェントは、認証情報の取得時に関連付けの設定に基づいて IAM 認証情報をキャッシュします。関連付けに EKS Pod Identity ロールのみが含まれ、ターゲット IAM ロールが含まれていない場合、キャッシュされた認証情報は 6 時間有効となります。関連付けに EKS Pod Identity ロール ARN とターゲット IAM ロールの両方が含まれている場合、キャッシュされた認証情報は 59 分間有効です。EKS Pod Identity ロール ARN の更新、ターゲット IAM ロールの追加、関連付けのセッションポリシーの更新など、既存の関連付けを変更しても、既存のキャッシュはリセットされません。その結果、エージェントはキャッシュされた認証情報が更新されるまで更新を認識しません。変更をより迅速に適用するには、既存のポッドを再作成できます。再作成しない場合は、キャッシュの有効期限が切れるまで待機する必要があります。

ステップ 1: ターゲット IAM ロールを作成して関連付ける

このステップでは、ターゲット IAM ロールを作成して設定することで、安全な信頼チェーンを確立します。デモンストレーションでは、2 つの AWS アカウント間で信頼チェーンを確立するための新しいターゲット IAM ロールを作成します。EKS クラスターの AWS アカウント内にある EKS Pod Identity ロール (例: `eks-pod-identity-primary-role`) で、ターゲットアカウントでターゲット IAM ロール (例 : `eks-pod-identity-aws-resources`) を引き受ける権限を取得し、Amazon S3 バケットなどの AWS リソースへのアクセスを有効にします。

ターゲット IAM ロールを作成する

1. Amazon IAM コンソールを開きます。
2. 上部のナビゲーションバーで、ターゲット IAM ロールの AWS リソース (S3 バケットや DynamoDB テーブルなど) を含むアカウントにサインインしていることを確認します。
3. 左のナビゲーションペインで、[ロール] を選択してください。
4. [ロールの作成] ボタンを選択し、「信頼できるエンティティタイプ」の下にある AWS アカウントを選択します。
5. 別の AWS アカウントを選択し、AWS のアカウント番号 (EKS Pod Identity ロールが存在するアカウント) を入力し、[次へ] を選択します。

6. ロールに関連付けるアクセス許可ポリシー (AmazonS3FullAccess など) を追加し、[次へ] を選択します。
7. 「MyCustomIAMTargetRole」などのロール名を入力し、[ロールの作成] を選択します。

ターゲット IAM ロールの信頼ポリシーを更新する

1. ロールを作成すると、[ロール] リストに戻ります。前の手順で作成した新規ロールを検索し、選択します (例: MyCustomIAMTargetRole)。
2. [信頼関係] タブを選択します。
3. [Edit trust policy] (信頼ポリシーを編集) を選択します。
4. ポリシーエディタで、デフォルトの JSON を信頼ポリシーに置き換えます。IAM ロール ARN のロール名と 111122223333 のプレースホルダー値を、EKS クラスターをホストする AWS アカウント ID に置き換えます。オプションで、ロール信頼ポリシーの PrincipalTags を使用して、特定のクラスターと名前空間の特定のサービスアカウントのみがターゲットロールを引き受けることを許可することもできます。例えば、次のようになります。

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "AWS": "arn:aws:iam::111122223333:root"
      },
      "Action": [
        "sts:AssumeRole",
        "sts:TagSession"
      ],
      "Condition": {
        "StringEquals": {
          "aws:RequestTag/eks-cluster-arn": "arn:aws:eks:us-
east-1:111122223333:cluster/example-cluster",
          "aws:RequestTag/kubernetes-namespace": "ExampleNameSpace",
          "aws:RequestTag/kubernetes-service-account": "ExampleServiceAccountName"
        },
        "ArnEquals": {
          "aws:PrincipalARN": "arn:aws:iam::111122223333:role/eks-pod-identity-primary-
role"
        }
```

```
      }
    }
  ]
}
```

上記のポリシーでは、関連する EKS Pod Identity セッションタグを持つ AWS アカウント 111122223333 のロール `eks-pod-identity-primary-role` がこのロールを引き受けることを許可します。

EKS Pod Identity でセッションタグを無効にした場合、EKS Pod Identity は、ターゲットロールを引き受けるときに、ポッドのクラスター、名前空間、およびサービスアカウントに関する情報を `sts:ExternalId` に設定します。

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "AWS": "arn:aws:iam::111122223333:root"
      },
      "Action": "sts:AssumeRole",
      "Condition": {
        "StringEquals": {
          "sts:ExternalId": "region/111122223333/cluster-name/namespace/service-
account-name"
        },
        "ArnEquals": {
          "aws:PrincipalARN": "arn:aws:iam::111122223333:role/eks-pod-identity-primary-
role"
        }
      }
    },
    {
      "Effect": "Allow",
      "Principal": {
        "AWS": "arn:aws:iam::111122223333:root"
      },
      "Action": "sts:TagSession"
    }
  ]
}
```

上記のポリシーは、予想されるクラスター、名前空間、サービスアカウントのみがターゲットロールを引き受けることができるようにするのに役立ちます。

EKS Pod Identity ロールのアクセス許可ポリシーを更新する

このステップでは、ターゲット IAM ロール ARN をリソースとして追加し、Amazon EKS クラスターに関連付けられた EKS Pod Identity ロールのアクセス許可ポリシーを更新します。

1. Amazon EKS コンソール を開きます。
2. 左のナビゲーションペインで [クラスター] を選択し、次に EKS クラスターの名前を選択します。
3. アクセス タブを選択します。
4. [Pod Identity の関連付け] で、[EKS Pod Identity ロール] を選択します。
5. [アクセス許可]、[許可を追加] を選択し、次に [インラインポリシーを作成] をクリックします。
6. JSON を選択します。
7. ポリシーエディタで、デフォルトの JSON をアクセス許可ポリシーに置き換えます。IAM ロール ARN のロール名と 222233334444 のプレースホルダー値をターゲット IAM ロールに置き換えます。例えば、次のようになります。

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "sts:AssumeRole",
                "sts:TagSession"
            ],
            "Resource": "arn:aws:iam::222233334444:role/eks-pod-identity-aws-resources"
        }
    ]
}
```

ステップ 2: ターゲット IAM ロールを Kubernetes サービスアカウントに関連付ける

このステップでは、ターゲット IAM ロールと EKS クラスター内の Kubernetes サービスアカウントとの関連付けを作成します。

1. Amazon EKS コンソール を開きます。

2. 左のナビゲーションペインで [クラスター] を選択し、次に関連付けを追加するクラスター名を選択します。
3. [アクセス] タブを選択します。
4. [Pod Identity の関連付け] で [作成] を選択します。
5. ワークロードが引き受ける IAM ロールの EKS Pod Identity ロールを選択します。
6. EKS Pod Identity ロールが引き受けるターゲット IAM ロールを [ターゲット IAM ロール] で選択します。
7. [Kubernetes 名前空間] フィールドに、関連付けを作成する名前空間の名前を入力します (例: `my-app-namespace`)。これにより、サービスアカウントが存在する場所が定義されます。
8. [Kubernetes サービスアカウント] フィールドに、IAM 認証情報を使用するサービスアカウントの名前 (例: `my-service-account`) を入力します。これにより、IAM ロールがサービスアカウントにリンクされます。
9. (オプション) [セッションタグを無効にする] を選択することで、Pod Identity がロールを引き受けるときに自動的に追加するデフォルトのセッションタグを無効にできます。
10. (オプション) [セッションポリシーの設定] を切り替えることで、ターゲット IAM ロールにアタッチされた IAM ポリシーで定義されているアクセス許可以外の追加制限を、この Pod Identity 関連付けに適用するように IAM ポリシーを設定できます。

> **Note**
>
> 1. セッションポリシーは、[セッションタグを無効にする] の設定がチェックされている場合にのみ適用できます。2. セッションポリシーを指定すると、ポリシーの制限は、この Pod Identity の関連付けに関連付けられた IAM ロールではなく、ターゲット IAM ロールのアクセス許可に適用されます。

11. [作成] を選択して関連付けを作成します。

サービスアカウントを使用して AWS サービスにアクセスするように Pod を設定する

Pod が AWS サービスにアクセスする必要がある場合は、Kubernetes サービスアカウントを使用するように設定する必要があります。サービスアカウントは、AWS サービスにアクセスする権限がある AWS Identity and Access Management (IAM) ロールに関連付ける必要があります。

- 既存のクラスター。まだ所有していない場合は、はじめに でのガイドのいずれかを参照しながら作成できます。

- 既存の Kubernetes サービスアカウントと、そのサービスアカウントを IAM ロールに関連付ける EKS Pod Identity 関連付け。ロールには、Pod が AWS サービスを使用するアクセス許可を含む IAM ポリシーが関連付けられている必要があります。サービスアカウントとロールの作成および設定方法については、the section called “IAM ロールを割り当てる” を参照してください。
- ご使用のデバイスまたは AWS CloCloudShell に最新バージョンの AWS CLI がインストールおよび設定されていること。現在のバージョンは `aws --version | cut -d / -f2 | cut -d ' ' -f1` で確認できます。yum、apt-get、macOS 用の Homebrew などのパッケージマネージャーは、多くの場合 AWS CLI の最新バージョンより数バージョン古くなっています。最新バージョンをインストールするには、「AWS コマンドラインインターフェイスユーザーガイド」の「インストール」および「aws configure を使用したクイック設定」を参照してください。AWS CloudShell にインストールされている AWS CLI バージョンは、最新バージョンより数バージョン遅れている可能性もあります。更新するには、「AWS CloudShell ユーザーガイド」の「ホームディレクトリへの AWS CLI のインストール」を参照してください。
- デバイスまたは AWS CloudShell に、kubectl コマンドラインツールがインストールされていること。バージョンはご使用のクラスターの Kubernetes バージョンと同じか、1 つ前のマイナーバージョン以前、あるいはそれより新しいバージョンが使用できます。例えば、クラスターのバージョンが 1.29 である場合、kubectl のバージョン 1.28、1.29、または 1.30 が使用できます。kubectl をインストールまたはアップグレードする方法については、the section called “kubectl と eksctl のセットアップ” を参照してください。
- クラスター構成を含む既存の kubectl config ファイル。kubectl config ファイルの作成については、the section called “kubectl を使用してクラスターにアクセスする” を参照してください。

  1. 次コマンドを使用して、Pod をデプロイして設定を確認できるデプロイマニフェストを作成します。example の値は独自の値に置き換えます。

```
cat >my-deployment.yaml <<EOF
apiVersion: apps/v1
kind: Deployment
metadata:
  name: my-app
spec:
  selector:
    matchLabels:
      app: my-app
  template:
    metadata:
      labels:
        app: my-app
    spec:
```

```
      serviceAccountName: my-service-account
      containers:
      - name: my-app
        image: public.ecr.aws/nginx/nginx:X.XX
EOF
```

2. マニフェストをクラスターにデプロイします。

```
kubectl apply -f my-deployment.yaml
```

3. Pod に必要な環境変数が存在することを確認してください。

   a. 前の手順のデプロイ時にデプロイされた Pod を確認します。

   ```
   kubectl get pods | grep my-app
   ```

   出力例は次のとおりです。

   ```
   my-app-6f4dfff6cb-76cv9   1/1     Running   0          3m28s
   ```

   b. Pod にサービスアカウントトークンのファイルがマウントされていることを確認します。

   ```
   kubectl describe pod my-app-6f4dfff6cb-76cv9 | grep
    AWS_CONTAINER_AUTHORIZATION_TOKEN_FILE:
   ```

   出力例は次のとおりです。

   ```
   AWS_CONTAINER_AUTHORIZATION_TOKEN_FILE:  /var/run/secrets/
   pods.eks.amazonaws.com/serviceaccount/eks-pod-identity-token
   ```

4. ロールにアタッチされた IAM ポリシーで割り当てたアクセス許可を使用して、Pod が AWS サービスとやり取りできることを確認します。

> **Note**
>
> サービスアカウントに関連付けられた IAM ロールの AWS 認証情報を Pod が使用する場合、AWS CLI またはその Pod のコンテナ内にある他の SDK は、そのロールから提供される認証情報を使用します。Amazon EKS ノードの IAM ロールに提供された認証情報へのアクセスを制限しない場合、ポッドは引き続きこれらの認証情報にアクセスで

きます。詳細については「ワーカーノードに割り当てられたインスタンスプロファイルへのアクセスを制限する」を参照してください。

Pod が想定通りにサービスとやり取りできない場合は、次の手順を実行して、すべてが正しく設定されていることを確認してください。

a. EKS Pod Identity 関連付けを介した IAM ロールの引き受けをサポートする AWS SDK バージョンを Pod が使用していることを確認します。詳細については、the section called “サポートされている SDK” を参照してください。

b. デプロイがサービスアカウントを使用していることを確認します。

```
kubectl describe deployment my-app | grep "Service Account"
```

出力例は次のとおりです。

```
Service Account:  my-service-account
```

タグに基づいてポッドに AWS リソースへのアクセス権を付与する

属性ベースのアクセス制御 (ABAC) は、複数の属性をまとめたポリシーを利用して、ユーザーに権限を付与するものです。EKS Pod Identity は、クラスター名、名前空間、サービスアカウント名などの属性が含まれたタグを各 Pod の一時認証情報にアタッチします。これらのロールセッションタグを使用すると、管理者は一致するタグに基づいて AWS リソースへのアクセスを許可することで、複数のサービスアカウントで機能する単一のロールを作成できます。ロールセッションタグのサポートを追加することで、同じ IAM ロールと IAM ポリシーを再利用しながら、クラスター間およびクラスター内のワークロード間のセキュリティ境界を厳しくすることができます。

タグ付きポリシーの例

以下は、対応するオブジェクトが EKS クラスター名でタグ付けされたときに `s3:GetObject` アクセス許可を付与する IAM ポリシーの例です。

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
```

```
                "s3:ListBucket"
            ],
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "s3:GetObject",
                "s3:GetObjectTagging"
            ],
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "s3:ExistingObjectTag/eks-cluster-name": "${aws:PrincipalTag/eks-
cluster-name}"
                }
            }
        }
    ]
}
```

セッションタグを有効または無効にする

EKS Pod Identity で、ロールを引き受けるときに事前定義されたセッションタグのセットを追加します。これらのセッションタグを使用すると、管理者は一致するタグに基づいて AWS リソースへのアクセスを許可することで、リソース間で機能する単一のロールを作成できます。

セッションタグを有効にする

セッションタグは EKS Pod Identity で自動的に有効になります。ユーザー側のアクションは必要ありません。デフォルトでは、EKS Pod Identity によって事前定義されたタグのセットがセッションにアタッチされます。ポリシーでこれらのタグを参照するには、`${aws:PrincipalTag/` 構文の後にタグキーを使用します。例えば、`${aws:PrincipalTag/kubernetes-namespace}`。

- `eks-cluster-arn`
- `eks-cluster-name`
- `kubernetes-namespace`
- `kubernetes-service-account`
- `kubernetes-pod-name`
- `kubernetes-pod-uid`

セッションタグを無効にする

AWS では、インラインセッションポリシー、マネージドポリシー ARN、およびセッションタグが、個別の制限を持つひとまとめのバイナリ形式に圧縮されます。ひとまとめのバイナリ形式がサイズ制限を超過したことを示す `PackedPolicyTooLarge` エラーが表示された場合は、EKS Pod Identity によって追加されたセッションタグを無効にすることでサイズを縮小できます。これらのセッションタグを無効にするには、次の手順に従います。

1. Amazon EKS コンソール を開きます。
2. 左のナビゲーションペインで [クラスター] を選択し、次に修正するクラスターの名前を選択します。
3. [アクセス] タブを選択します。
4. [Pod Identity の関連付け] で、[関連付け ID] で変更する関連付け ID を選択し、[編集] を選択します。
5. [セッションタグ] で、[セッションタグを無効にする] を選択します。
6. [Save changes] (変更の保存) をクリックします。

アカウント間のコピー

EKS Pod Identity によって追加されるセッションタグはすべて推移的です。タグのキーと値は、ワークロードがロールを別のアカウントに切り替えるために使用するすべての `AssumeRole` アクションに渡されます。これらのタグを他のアカウントのポリシーで使用して、クロスアカウントシナリオでのアクセスを制限できます。詳細については、「IAM ユーザーガイド」の「セッションタグを使用したロールの連鎖」を参照してください。

カスタムタグ

EKS Pod Identity は、実行する `AssumeRole` アクションにカスタムタグを追加できません。ただし、IAM ロールに適用するタグは、常に同じ形式 (`${aws:PrincipalTag/MyCustomTag}` のように、`${aws:PrincipalTag/` の後にキーが続く) で使用できます。

> **Note**
>
> `sts:AssumeRole` リクエストによってセッションに追加されたタグは、競合が発生した場合に優先されます。例えば、次のように言います。
>
> - EKS が顧客ロールを引き継ぐときに Amazon EKS がセッションにキー `eks-cluster-name` と値 `my-cluster` を追加します。

- また、値 `my-own-cluster` を含む IAM ロールに `eks-cluster-name` タグも追加します。

この場合は前者が優先され、`eks-cluster-name` タグの値は `my-cluster` になります。

AWS SDK で Pod Identity を使用する

EKS Pod Identity 認証情報の使用

EKS Pod Identity の関連付けからの認証情報を使用するには、コードで任意の AWS SDK を使用して SDK を使用する AWS サービスのクライアントを作成できます。デフォルトでは、SDK は使用する AWS Identity and Access Management 認証情報を一連の場所で検索します。クライアントの作成時や SDK の初期化時に認証情報プロバイダーを指定しなかった場合は、EKS Pod Identity 認証情報が使用されます。

これがうまくいくのは、EKS Pod Identity がデフォルトの認証情報チェーンのステップで検索されるコンテナ認証情報プロバイダーに追加されているからです。現在、ワークロードが認証情報チェーンの前にある認証情報を使用している場合は、同じワークロードに EKS Pod Identity の関連付けを設定しても、その認証情報は引き続き使用されます。

EKS Pod Identity の仕組みの詳細については、「the section called “仕組み”」を参照してください。

「Learn how EKS Pod Identity grants pods access to AWS services」に従うと、Pod 内のコンテナでは EKS Pod Identity エージェントからの IAM ロールの引き受けをサポートする AWS SDK バージョンを使用する必要があります。お使いの AWS SDK には、必ず次のバージョン以降を使用してください。

- Java (バージョン 2) – 2.21.30
- Java – 1.12.746
- バージョン1に移動 – v1.47.11
- バージョン 2 に移動 – 2023-11-14 リリース
- Python (Boto3) – 1.34.41
- Python (botocore) – 1.34.41
- AWS CLI – 1.30.0

  AWS CLI – 2.15.0

- JavaScript v2 – 2.1550.0
- JavaScript v3 – v3.458.0
- Kotlin – v1.0.1
- Ruby – 3.188.0
- Rust – release-2024-03-13
- C++ – 1.11.263
- .NET – 3.7.734.0
- PowerShell – 4.1.502
- PHP – 3.289.0

サポートされている SDK を使用していることを確認するには、コンテナを構築する際に、「AWS での構築ツール」で、希望する SDK のインストール手順に従ってください。

EKS Pod Identity をサポートするアドオンのリストについては、「the section called “Pod Identity サポートリファレンス”」を参照してください。

EKS Pod Identity エージェントで **IPv6** を無効にする

AWS マネジメントコンソール

1. EKS Pod Identity エージェントで IPv6 を無効にするには、EKS アドオンの [オプション設定] に次の設定を追加します。
   a. Amazon EKS コンソールを開きます。
   b. 左のナビゲーションペインで、[クラスター] を選択し、次にアドオンを設定するクラスター名を選択してください。
   c. [アドオン] タブを選択してください。
   d. EKS Pod Identity エージェントのアドオンボックスの右上にあるボックスを選択し、[編集] を選択します。
   e. EKS Pod Identity エージェントの設定ページの場合:
      i. 使用する [バージョン] を選択します。前のステップと同じバージョンを保持し、別のアクションでバージョンと設定を更新することをお勧めします。
      ii. [オプションの構成設定] を展開します。
      iii. [設定値] で JSON キー `"agent":` とにネストした JSON オブジェクトの値に、キー `"additionalArgs":` を指定します。結果のテキストは有効な JSON オブジェクトでなけ

ればなりません。このキーと値だけがテキストボックス内のデータである場合はキーと値を中括弧 { } で囲みます。次の例はネットワークポリシーが有効になっていることを示しています：

```
{
    "agent": {
        "additionalArgs": {
            "-b": "169.254.170.23"
        }
    }
}
```

この設定では、エージェントが使用する唯一のアドレスに IPv4 アドレスを設定します。

f. EKS Pod Identity エージェントのポッドを置き換えて新しい設定を適用するには、[変更を保存] を選択します。

Amazon EKS は、EKS Pod Identity エージェントの Kubernetes DaemonSet のロールアウトを使用して EKS アドオンに変更を適用します。ロールアウトのステータスはAWS マネジメントコンソール のアドオン [更新履歴] と kubectl rollout status daemonset/eks-pod-identity-agent --namespace kube-system で追跡できます。

kubectl rollout は以下のコマンドを実行してください：

```
$ kubectl rollout

history  -- View rollout history
pause    -- Mark the provided resource as paused
restart  -- Restart a resource
resume   -- Resume a paused resource
status   -- Show the status of the rollout
undo     -- Undo a previous rollout
```

ロールアウトに時間がかかりすぎる場合、Amazon EKS はロールアウトを取り消し、[アドオン更新] のタイプと [失敗] のステータスのメッセージがアドオンの [更新履歴] に追加されます。問題を調査するには、ロールアウトの履歴から開始し、EKS Pod Identity エージェントのポッドで kubectl logs を実行して EKS Pod Identity エージェントのログを確認します。

2. [更新履歴] の新しいエントリのステータスが [成功] の場合、ロールアウトが完了し、アドオンはすべての EKS Pod Identity エージェントのポッドで新しい設定を使用していることを意味します。

AWS CLI

1. EKS Pod Identity エージェントで IPv6 を無効にするには、EKS アドオンの [オプション設定] に次の設定を追加します。

   次の AWS CLI コマンドを実行します。my-cluster をクラスターの名前に置き換え、IAM ロール ARN を使用するロールに置き換えます。

   ```
   aws eks update-addon --cluster-name my-cluster --addon-name eks-pod-identity-agent \
       --resolve-conflicts PRESERVE --configuration-values '{"agent":{"additionalArgs":
    { "-b": "169.254.170.23"}}}'
   ```

   この設定では、エージェントが使用する唯一のアドレスに IPv4 アドレスを設定します。

   Amazon EKS は、EKS Pod Identity エージェントの Kubernetes DaemonSet のロールアウトを使用して EKS アドオンに変更を適用します。ロールアウトのステータスはAWS マネジメントコンソール のアドオン [更新履歴] と kubectl rollout status daemonset/eks-pod-identity-agent --namespace kube-system で追跡できます。

   kubectl rollout は以下のコマンドを実行してください：

   ```
   kubectl rollout

   history  -- View rollout history
   pause    -- Mark the provided resource as paused
   restart  -- Restart a resource
   resume   -- Resume a paused resource
   status   -- Show the status of the rollout
   undo     -- Undo a previous rollout
   ```

   ロールアウトに時間がかかりすぎる場合、Amazon EKS はロールアウトを取り消し、[アドオン更新] のタイプと [失敗] のステータスのメッセージがアドオンの [更新履歴] に追加されます。問題を調査するには、ロールアウトの履歴から開始し、EKS Pod Identity エージェントのポッドで kubectl logs を実行して EKS Pod Identity エージェントのログを確認します。

EKS Pod Identity で必須の信頼ポリシーを使用して IAM ロールを作成する

```
{
    "Version":"2012-10-17",
```

```
    "Statement": [
        {
            "Sid": "AllowEksAuthToAssumeRoleForPodIdentity",
            "Effect": "Allow",
            "Principal": {
                "Service": "pods.eks.amazonaws.com"
            },
            "Action": [
                "sts:AssumeRole",
                "sts:TagSession"
            ]
        }
    ]
}
```

`sts:AssumeRole`

EKS Pod Identity は一時的な認証情報をポッドに渡す前に、`AssumeRole` を使用して IAM ロールを引き継ぎます。

`sts:TagSession`

EKS Pod Identity は、AWS STS へのリクエストにセッションタグを含めるために `TagSession` を使用します。

条件の設定

これらのタグを信頼ポリシーの 条件キー で使用して、このロールを使用できるサービスアカウント、名前空間、およびクラスターを制限できます。Pod Identity が追加するリクエストタグのリストについては、「the section called “セッションタグを有効または無効にする”」を参照してください。

例えば、追加した `Condition` に以下の信頼ポリシーを指定して、Pod Identity IAM ロールを引き受けることができるポッドを特定の `ServiceAccount` および `Namespace` に限定できます。

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "AllowEksAuthToAssumeRoleForPodIdentity",
            "Effect": "Allow",
            "Principal": {
```

```
                "Service": "pods.eks.amazonaws.com"
            },
            "Action": [
                "sts:AssumeRole",
                "sts:TagSession"
            ],
            "Condition": {
                "StringEquals": {
                    "aws:RequestTag/kubernetes-namespace": [
                        "Namespace"
                    ],
                    "aws:RequestTag/kubernetes-service-account": [
                        "ServiceAccount"
                    ]
                }
            }
        }
    ]
}
```

Amazon EKS 条件キーのリストについては「サービス認可リファレンス」の「Amazon エラスティックKubernetesサービス によって定義された条件」を参照してください。条件キーを使用できるアクションとリソースについては「Amazon エラスティックKubernetesサービス で定義されるアクション」を参照してください。

ノードを使用してコンピューティングリソースを管理する

Kubernetes ノードはコンテナ化されたアプリケーションを実行するマシンです。各ノードには以下のコンポーネントがあります：

- コンテナランタイム - コンテナの実行を担当するソフトウェア。
- kubelet - コンテナが正常で、関連する Pod 内で実行されていることを確認します。
- kube-proxy - Pod との通信を許可するネットワークルールを維持します。

詳細については、Kubernetes ドキュメントの「ノード」を参照してください。

Amazon EKS クラスターは、任意の EKS 自動モードマネージド型ノード、セルフマネージド型ノード、Amazon EKS マネージド型ノードグループ、AWS Fargate、Amazon EKS Hybrid Nodes を組み合わせて、Pod をスケジュールできます。クラスターにデプロイされているノードの詳細についてはthe section called “クラスターリソースにアクセスする”を参照してください。

> **Note**
>
> ハイブリッドノードを除くノードはクラスターの作成時に選択したサブネットと同じ VPC 内にある必要があります。ただし、ノードは同じサブネット内にある必要はありません。

コンピューティングオプションの比較

次の表に、要件を満たすための最適なオプションを決定する際に、評価すべき基準をいくつか示します。セルフマネージド型ノードはリストされているすべての条件をサポートするもう 1 つのオプションですが、より多くの手動メンテナンスが必要です。詳細については、the section called “セルフマネージド型ノード” を参照してください。

> **Note**
>
> Bottlerocket には、この表の一般的な情報と特に異なる点がいくつかあります。詳しくは、GitHub 上で Bottlerocket のドキュメントを参照してください。

条件	EKS マネージド型ノードグループ	EKS 自動モード	Amazon EKS Hybrid Nodes
AWS アウトポスト にデプロイ可能	いいえ	いいえ	いいえ
AWS ローカルゾーン にデプロイ可能	はい	いいえ	いいえ
Windows を必要とするコンテナを実行できる	はい	いいえ	いいえ
Linux を必要とするコンテナを実行可能	はい	はい	はい
インフェクエンティア チップを必要とするワークロードを実行可能	はい – Amazonリナックス ノードのみ	はい	いいえ
GPU を必要とするワークロードを実行可能	はい – Amazonリナックス ノードのみ	はい	はい
Arm プロセッサを必要とするワークロードを実行可能	はい	はい	はい
AWS Bottlerocket を実行可能	はい	はい	いいえ
ポッドはCPU、メモリ、ストレージ、およびネットワークリソースを他のポッドと共有します。	はい	はい	はい

条件	EKS マネージド型ノードグループ	EKS 自動モード	Amazon EKS Hybrid Nodes
Amazon EC2 インスタンスをデプロイおよび管理する必要がある	はい	いいえ - EC2 マネージドインスタンスについて説明します	はい - オンプレミスの物理マシンまたは仮想マシンは選択したツールで管理されます。
Amazon EC2 インスタンスのオペレーティングシステムの保護、保守、パッチ適用を行う必要がある	はい	いいえ	はい – 物理マシンまたは仮想マシンで実行されているオペレーティングシステムは選択したツールで管理されます。
ノードをデプロイする時に、追加の kubelet 引数 などのブートストラップ引数を提供できます。	はい – eksctl、またはカスタム AMI を含む起動テンプレートを使用します。	いいえ - NodeClass を使用してノードを設定します	はい - nodeadm を使用してブートストラップ引数をカスタマイズできます。the section called “ハイブリッドノード用 nodeadm” を参照してください。
ノードに割り当てられた IP アドレスとは異なる CIDR ブロックから、IP アドレスを Pod に割り当てることができます。	はい - カスタム AMI の起動テンプレートを使用します。詳細については、the section called “起動テンプレート” を参照してください。	いいえ	はい - 「the section called “CNI を設定する”」を参照してください。
ノードに SSH 接続可能	はい	いいえ — ノードのトラブルシューティング方法について説明します	はい

条件	EKS マネージド型ノードグループ	EKS 自動モード	Amazon EKS Hybrid Nodes
独自のカスタム AMI をノードにデプロイ可能	はい – 起動テンプレートを使用します	いいえ	はい
独自のカスタム CNI をノードにデプロイ可能	はい – カスタム AMI の起動テンプレートを使用します	いいえ	はい
ノード AMI を自分で更新する必要がある	はい - Amazon EKS 最適化 AMI をデプロイした場合、更新が利用可能になると Amazon EKS コンソールで通知されます。コンソールではワンクリックで更新を実行できます。カスタム AMI をデプロイした場合、更新が利用可能になっても Amazon EKS コンソールに通知されません。更新は自分で実行する必要があります。	いいえ	はい - 物理マシンまたは仮想マシンで実行されているオペレーティングシステムは選択したツールで管理されます。the section called “オペレーティングシステムを準備する” を参照してください。

条件	EKS マネージド型ノードグループ	EKS 自動モード	Amazon EKS Hybrid Nodes
ノードの Kubernetes バージョンを自分で更新する必要があります。	はい - Amazon EKS 最適化 AMI をデプロイした場合、更新が利用可能になると Amazon EKS コンソールで通知されます。コンソールではワンクリックで更新を実行できます。カスタム AMI をデプロイした場合、更新が利用可能になっても Amazon EKS コンソールに通知されません。更新は自分で実行する必要があります。	いいえ	はい - 独自のツールまたは nodeadm を使用して、ハイブリッドノードのアップグレードを管理します。the section called “ハイブリッドノードをアップグレードする” を参照してください。
Pod で Amazon EBS ストレージを使用可能	はい	はい。統合された機能として使用できます。ストレージクラスを作成する方法について説明します。	いいえ
Pod で Amazon EFS ストレージを使用可能	はい	はい	いいえ
Pod で Amazon S3 ファイルストレージを使用可能	はい	はい	いいえ

条件	EKS マネージド型ノードグループ	EKS 自動モード	Amazon EKS Hybrid Nodes
Pod で Amazon FSx for Lustre ストレージを使用可能	はい	はい	いいえ
Network Load Balancer をサービスに使用可能	はい	はい	はい - ターゲットタイプ `ip` を使用する必要があります。
ポッドはパブリックサブネットで実行可能	はい	はい	いいえ - ポッドはオンプレミス環境で実行されます。
それぞれの Pod に、異なる VPC セキュリティグループを割り当て可能	はい: Linux ノードのみ	いいえ	いいえ
Kubernetes DaemonSets を実行可能	はい	はい	はい
Pod マニフェストで `HostPort` および `HostNetwork` をサポートします	はい	はい	はい
AWS が利用可能なリージョン	すべての Amazon EKS 対応リージョン	すべての Amazon EKS 対応リージョン	AWS Govクラウド (米国) リージョンと中国リージョンを除くAmazon EKS がサポートするすべてのリージョン。

条件	EKS マネージド型ノードグループ	EKS 自動モード	Amazon EKS Hybrid Nodes
Amazon EC2 専有ホストでコンテナを実行できます	はい	なし	いいえ
料金	複数の Pod を実行する Amazon EC2 インスタンスのコスト。詳細については、Amazon EC2 料金表 を参照してください。	クラスターで EKS 自動モードが有効になっている場合は自動モードのコンピューティング機能を使用して起動されたインスタンスに対しては標準の EC2 インスタンス料金に加えて個別の料金がかかります。この金額は起動されたインスタンスタイプとクラスターが配置されている AWS リージョンによって異なります。詳細については「Amazon EKS の料金」を参照してください。	ハイブリッドノードの vCPU の 1 時間あたりのコスト。詳細については「Amazon EKS の料金」を参照してください。

マネージドノードグループを使用してノードライフサイクルを簡素化する

Amazon EKS マネージド型ノードグループは、Amazon EKS Kubernetes クラスターのノード (Amazon EC2 インスタンス) のプロビジョニングとライフサイクル管理を自動化します。

Amazon EKS マネージド型ノードグループでは、Kubernetes アプリケーションを実行するためのコンピューティング性能を提供する Amazon EC2 インスタンスを個別にプロビジョニングまたは登録

する必要はありません。1 回の操作で、クラスターのノードを作成、自動的に更新、または終了できます。ノードを更新し、終了させることで、ノードを自動的にドレーンし、アプリケーションを利用できる状態にしておきます。

すべてのマネージド型ノードは、Amazon EKS によって管理される Amazon EC2 Auto Scaling グループの一部としてプロビジョニングされます。インスタンスやAuto Scaling グループを含むすべてのリソースは、AWS アカウント内で実行されます。各ノードグループは、定義された複数のアベイラビリティーゾーンで実行できます。

マネージド型ノードグループは、ノードのヘルスを継続的にモニタリングするノード自動修復をオプションで利用することもできます。検出された問題に自動的に対応し、可能な場合はノードを置き換えます。これにより、最小限の手動操作でクラスターの全体的な可用性が向上します。詳細については、the section called “ノードのヘルスと修復” を参照してください。

Amazon EKS コンソール、`eksctl`、AWS CLI、AWS API、または AWS CloudFormation を含む Infrastructure as Code ツールを使用し、新規または既存のクラスターにマネージド型ノードグループを追加できます。マネージド型ノードグループの一部として起動されたノードは、自動的に Kubernetes Cluster Autoscaler によって自動検出用にタグ付けされます。ノードグループを使用して、Kubernetes ラベルをノードに適用し、いつでも更新できます。

Amazon EKS マネージド型ノードグループの使用に追加料金はかかりません。お支払いいただくのはプロビジョニングした AWS リソースの分だけです。これには、Amazon EC2 インスタンス、Amazon EBS ボリューム、Amazon EKS クラスター時間、その他の AWS インフラストラクチャが含まれます。最低料金や前払いの義務は発生しません。

新しい Amazon EKS クラスターおよびマネージド型ノードグループの使用を開始するには、「the section called “クラスターを作成する (コンソールと CLI)”」を参照してください。

既存のクラスターにマネージド型ノードグループを追加するには、「the section called “作成”」を参照してください。

マネージド型ノードグループの概念

- Amazon EKS マネージド型ノードグループは、Amazon EC2 インスタンスを作成し、管理します。
- すべてのマネージド型ノードは、Amazon EKS によって管理される Amazon EC2 Auto Scaling グループの一部としてプロビジョニングされます。さらに、Amazon EC2 インスタンスやAuto Scaling グループを含むすべてのリソースは、AWS アカウント内で実行されます。

- Amazon EKS は、マネージド型ノードグループのスケーリング設定を定期的に同期して、実際のAuto Scaling グループ値と一致させます。Cluster Autoscaler などの外部アクターがAuto Scaling グループのサイズを変更すると、`DescribeNodegroup` は最終的にそれらの変更を反映します。スケーリング設定を明示的に変更せずにノードグループの更新またはアップグレードを開始すると、ワークフローはノードグループの保存されたスケーリング設定ではなく現在のAuto Scaling グループ値を使用します。ストアドスケーリング設定は、明示的に `UpdateNodegroupConfig` リクエストに含める場合にのみ優先されます。
- マネージド型ノードグループのAuto Scaling グループは、グループの作成時に指定するすべてのサブネットにまたがります。
- Amazon EKS は、Kubernetes Cluster Autoscaler の使用を設定するようにマネージド型ノードグループリソースをタグ付けします。

> **Important**
>
> Amazon EBS ボリュームによってバックアップされ、Kubernetes Cluster Autoscaler を使用する複数のアベイラビリティーゾーンにわたってステートフルアプリケーションを実行している場合、それぞれが単一のアベイラビリティーゾーンにスコープされる複数のノードグループを設定する必要があります。また、`--balance-similar-node-groups` 機能を有効にする必要があります。

- カスタム起動テンプレートを使用すると、マネージド型ノードをデプロイするときに、柔軟性とカスタマイズのレベルを向上させることができます。例えば、追加の `kubelet` 引数を指定して、カスタム AMI を使用できます。詳細については、the section called “起動テンプレート” を参照してください。マネージド型ノードグループを最初に作成するときにカスタム起動テンプレートを使用しない場合は、自動生成された起動テンプレートを使用できます。自動生成されたテンプレートを手動で変更しないでください。エラーが発生します。
- Amazon EKS は、マネージド型ノードグループで CVE およびセキュリティパッチの責任共有モデルに従います。マネージド型ノードが Amazon EKS 最適化 AMI を実行する場合、バグや問題が報告されると、Amazon EKS が AMI のパッチ適用バージョンの構築を担当します。ただし、これらのパッチが適用された AMI バージョンのマネージド型ノードグループへのデプロイはユーザーが担当します。マネージド型ノードでカスタム AMI を実行する場合、バグや問題が報告され、AMI をデプロイする場合は、ユーザーがパッチ適用バージョンの構築を担当します。詳細については、the section called “更新” を参照してください。
- Amazon EKS マネージド型ノードグループは、パブリックサブネットとプライベートサブネットの両方で起動できます。2020 年 4月 22 日 以降にパブリックサブネットでマネージド型ノードグループを起動した場合、インスタンスがクラスターに正常に参加するには、サブネットで

MapPublicIpOnLaunch を true に設定する必要があります。パブリックサブネットが eksctl、または 2020 年 3 月 26 日以降に Amazon EKS が販売した AWS CloudFormation テンプレートを使用して作成された場合、この設定はすでに true に設定されています。パブリックサブネットが 2020 年 3 月 26 日 より前に作成されている場合は、設定を手動で変更する必要があります。詳細については「サブネットの IPv4 アドレス指定属性の変更」を参照してください。

- マネージド型ノードグループをプライベートサブネットにデプロイする場合、Amazon ECR にアクセスしてコンテナイメージをプルできることを確認します。これを行うには、NAT ゲートウェイをサブネットのルートテーブルに接続するか、次の AWS PrivateLink VPC エンドポイントを追加します。
  - Amazon ECR API のエンドポイントインターフェイス — com.amazonaws.*region-code*.ecr.api
  - Amazon ECR Docker レジストリ API のエンドポイントインターフェイス — com.amazonaws.*region-code*.ecr.dkr
  - Amazon S3 ゲートウェイのエンドポイント - com.amazonaws.*region-code*.s3

  その他の一般的に使用されるサービスとエンドポイントについては、「the section called “プライベートクラスター”」を参照してください。
- マネージドノードグループは、AWS Outposts または AWS Wavelength にはデプロイできません。マネージドノードグループは、AWS Local Zones で作成できます。詳細については、the section called “ AWS ローカルゾーン” を参照してください。
- 1 つのクラスター内に複数のマネージド型ノードグループを作成できます。例えば、一部のワークロード用に、標準 Amazon EKS 最適化 Amazon Linux AMI を使用するノードグループを作成し、GPU サポートを必要とするワークロード用に GPU バリアントを使用する別のノードグループを作成できます。
- マネージド型ノードグループで Amazon EC2 インスタンスのステータスチェックに障害が発生した場合、Amazon EKS は問題の診断に役立つエラーコードを返します。詳細については、the section called “マネージド型ノードグループのエラー” を参照してください。
- Amazon EKS は、マネージド型ノードグループインスタンスに Kubernetes ラベルを追加します。これらの Amazon EKS によって提供されたラベルには、プレフィックス eks.amazonaws.com が付与されます。
- Amazon EKS は、終了または更新時に Kubernetes API を使用してノードを自動的にドレーンします。
- AZRebalance を使用してノードを終了、または目的のノード数を減らす際には、ポッドの停止状態の予算は考慮されません。これらのアクションはノード内にある Pod を削除しようとします。ただし、15 分が経過すると、ノード内のすべての Pod が終了しているかどうかにかかわらず、

ノードは終了します。ノードが終了するまでの期間を延長するには、Auto Scaling グループにライフサイクルフックを追加します。詳細については、「Amazon EC2 Auto Scaling ユーザーガイド」の「ライフサイクルフックの追加」を参照してください。

- スポット中断通知またはキャパシティリバランス通知を受け取った後、ドレインプロセスを正しく実行するには、`CapacityRebalance` を `true` に設定する必要があります。
- マネージドノードグループを更新すると、Pod に設定した Pod 中断予算が考慮されます。詳細については、the section called “動作の詳細を更新する” を参照してください。
- Amazon EKS マネージド型ノードグループの使用に、追加料金はかかりません。プロビジョニングした AWS リソースに対してのみ料金が発生します。
- ノードの Amazon EBS ボリュームを暗号化する場合は、起動テンプレートを使用してノードをデプロイできます。起動テンプレートを使用せずに、暗号化された Amazon EBS ボリュームを持つマネージド型ノードをデプロイするには、アカウントに作成された新しい Amazon EBS ボリュームをすべて暗号化してください。詳細については、「Amazon EC2 ユーザーガイド」の「デフォルトで暗号化」を参照してください。

マネージド型ノードグループのキャパシティータイプ

マネージド型ノードグループを作成する場合、キャパシティータイプをオンデマンドまたはスポットのいずれかから選択できます。Amazon EKS は、Amazon EC2 Auto Scaling グループを使用して、マネージド型ノードグループをデプロイします。これにはオンデマンドのみか、Amazon EC2 スポットインスタンスのみが含まれます。単一の Kubernetes クラスター内で、耐障害性のあるアプリケーションの Pod をスポットのマネージド型ノードグループにスケジュールし、非フォールトトレラントなアプリケーションの Pod をオンデマンドのノードグループにスケジュールできます。デフォルトでは、マネージド型ノードグループはオンデマンド Amazon EC2 インスタンスをデプロイします。

オンデマンド

オンデマンドインスタンスでは、長期契約なしで、コンピューティング性能に対して秒単位で支払います。

デフォルトでは、キャパシティータイプを指定しない場合、マネージド型ノードグループはオンデマンドインスタンスでプロビジョニングされます。マネージド型ノードグループは、ユーザーの代わりに Amazon EC2 Auto Scaling グループを次のように設定します。

- オンデマンドキャパシティーをプロビジョニングするための配分戦略は、`prioritized` に設定されます。マネージド型ノードグループは、API 内部で渡されたインスタンスタイプの優先度を

使用して、オンデマンドキャパシティーを満たすときに最初に使用するインスタンスタイプを決定します。例えば、3 つのインスタンスタイプを `c5.large`、`c4.large`、`c3.large` の順序で指定するとします。オンデマンドインスタンスが起動されると、マネージド型ノードグループは `c5.large`、`c4.large`、`c3.large` の順でオンデマンドキャパシティーを満たします。詳しくは、Amazon EC2 Auto Scaling ユーザーガイドの Amazon EC2 Auto Scaling グループを参照してください。

- Amazon EKS は、キャパシティータイプを `eks.amazonaws.com/capacityType: ON_DEMAND` に指定しているマネージド型ノードグループ内のすべてのノードに、次の Kubernetes ラベルを追加します。このラベルを使用すると、オンデマンドノードで、ステートフルまたは非フォールトトレラントなアプリケーションをスケジュールできます。

Spot

Amazon EC2 スポットインスタンスは、オンデマンドの料金から大きな割引で提供される、Amazon EC2 の予備容量です。Amazon EC2 スポットインスタンスは、EC2 から容量の返却を求められると 2 分間の中断通知をもって中断されることがあります。詳細については、「Amazon EC2 ユーザーガイド」の「スポットインスタンス」を参照してください。Amazon EC2 スポットインスタンスを使用してマネージド型ノードグループを構成し、Amazon EKS クラスターで実行されているコンピューティングノードのコストを最適化できます。

マネージド型ノードグループ内でスポットインスタンスを使用するには、キャパシティータイプを `spot` に設定してマネージド型ノードグループを作成してください。マネージド型ノードグループは、ユーザーの代わりに Amazon EC2 Auto Scaling グループを設定し、次のようなスポットベストプラクティスを適用します。

- Spot ノードが最適な Spot キャパシティプールにプロビジョニングされるように、割り当て戦略を以下のいずれかに設定します。
  - `price-capacity-optimized` (PCO) – Kubernetes バージョン `1.28` 以降のクラスターに新しいノードグループを作成すると、割り当てストラテジーは `price-capacity-optimized` に設定されます。ただし、Amazon EKS マネージドノードグループが PCO をサポートし始める前に `capacity-optimized` で作成済みのノードグループの割り当て戦略は変更されません。
  - `capacity-optimized` (CO) – Kubernetes バージョン `1.27` 以前のクラスターに新しいノードグループを作成すると、割り当て戦略は `capacity-optimized` に設定されます。

  容量を割り当てるために利用可能なスポットキャパシティープールの数を増やすには、複数のインスタンスタイプを使用するようにマネージド型ノードグループを設定します。

- スポットノードが中断するリスクが高い場合に、Amazon EKS がスポットノードを適切にドレーンおよび再調整して、アプリケーションの中断を最小限に抑えられるように、Amazon EC2 Spot Capacity Rebalancing が有効化されています。詳細については、Amazon EC2 Auto Scaling ユーザーガイド の Amazon EC2 Auto Scaling 容量の再調整を参照してください。
  - スポットノードが再調整の推奨事項を受け取ると、Amazon EKS は自動的に新しい代替スポットノードの起動を試みます。
  - 代替スポットノードが Ready 状態になる前にスポットの 2 分間の中断通知が到着すると、Amazon EKS は再調整に関する推奨事項を受け取ったスポットノードのドレーンを開始します。Amazon EKS はベストエフォートベースでノードをドレインします。そのため、Amazon EKS が既存のノードをドレインする前に、代替ノードがクラスターに参加するのを待機するという保証はありません。
  - 代替スポットノードがブートストラップされ、Kubernetes 上で Ready 状態になると、Amazon EKS は再調整に関する推奨事項を受信したスポットノードを遮断し、ドレインします。スポットノードを遮断すると、サービスコントローラーがこのスポットノードに新しいリクエストを送信しないようにします。正常でアクティブなスポットノードのリストからも削除されます。スポットノードをドレインすると、実行中の Pod が適切に削除されます。
- Amazon EKS は、キャパシティータイプを `eks.amazonaws.com/capacityType: SPOT` に指定しているマネージド型ノードグループ内のすべてのノードに、次の Kubernetes ラベルを追加します。このラベルを使用して、スポットノードで耐障害性のあるアプリケーションをスケジュールできます。

> **Important**
>
> EC2 は、スポットインスタンスを終了する 2 分前にスポットの中断通知を発行します。ただし、スポットノード上のポッドは、正常なシャットダウンのために 2 分間すべてを確保できない場合があります。EC2 が通知を発行すると、Amazon EKS がポッドのエビクションを開始するまでに遅延が発生します。エビクションは Kubernetes API サーバーを保護するために順次実行されるため、複数のスポット回収が同時に発生した場合、一部のポッドでは エビクション通知の受信が遅れることがあります。特にポッド密度が高いノード上、同時回収中、または長い終了猶予期間を使用している場合、ポッドは終了シグナルを受信せずに強制終了される場合があります。スポットワークロードでは、中断耐性のあるアプリケーションを設計し、30 秒以下の終了猶予期間を使用し、長時間実行される preStop フックを回避し、ポッド エビクションメトリクスをモニタリングして、クラスターで実際に適用される猶予期間を把握することをお勧めします。正常な終了が保証され

る必要があるワークロードについては、代わりにオンデマンドキャパシティーを使用することをお勧めします。

オンデマンドキャパシティーとスポットキャパシティーのどちらでノードグループをデプロイするかを決定するときは、次の条件を考慮する必要があります。

- スポットインスタンスは、ステートレスかつフォールトトレラントで、柔軟性の高いアプリケーションに適しています。これには、バッチトレーニングワークロード、機械学習トレーニングワークロード、Apache Spark などのビッグデータ ETL、キュー処理アプリケーション、ステートレス API エンドポイントなどがあります。スポットは Amazon EC2 の予備容量であり、時間の経過とともに変化する可能性があるため、中断されても問題ないワークロードには、スポットキャパシティーを使用することをお勧めします。具体的には、スポット容量は、必要な容量が使用できない期間を許容できるワークロードに適しています。
- 非フォールトトレラントなアプリケーションには、オンデマンドを使用することをお勧めします。これには、モニタリングツールやオペレーションツールなどのクラスター管理ツール、`StatefulSets` を必要とするデプロイ、データベースなどのステートフルなアプリケーションなどが含まれます。
- スポットインスタンスの使用中にアプリケーションの可用性を最大化するには、スポットのマネージド型ノードグループが複数のインスタンスタイプを使用するように構成することをお勧めします。複数のインスタンスタイプを使用する場合は、次のルールを適用することをお勧めします。
  - マネージド型ノードグループ内で Cluster Autoscaler を使用している場合、同じサイズの vCPU とメモリリソースを持つインスタンスタイプを柔軟に組み合わせて使用することをお勧めします。これは、クラスター内のノードが期待どおりにスケールされるようにするためです。例えば、4 つの vCPU と 8 GiB のメモリが必要な場合は、`c3.xlarge`、`c4.xlarge`、`c5.xlarge`、`c5d.xlarge`、`c5a.xlarge`、`c5n.xlarge`、またはその他の同等なインスタンスタイプを使用してください。
  - アプリケーションの可用性を高めるために、複数のスポットマネージド型ノードグループをデプロイすることをお勧めします。これを行うには、各グループが、同じ vCPU とメモリリソースを持つ、インスタンスタイプの柔軟な組み合わせを使用する必要があります。例えば、4 つの vCPU および 8 GiB のメモリが必要な場合は、`c3.xlarge`、`c4.xlarge`、`c5.xlarge`、`c5d.xlarge`、`c5a.xlarge`、`c5n.xlarge`、または他の同等なインスタンスタイプのマネージド型ノードグループを 1 つ作成し、`m3.xlarge`、`m4.xlarge`、`m5.xlarge`、`m5d.xlarge`、`m5a.xlarge`、`m5n.xlarge`、または他の同等なインスタンスタイプで 2 つ目のマネージド型ノードグループを作成することをお勧めします。

- カスタム起動テンプレートを使用しているスポットキャパシティータイプでノードグループをデプロイする場合、API を使用して複数のインスタンスタイプを渡します。起動テンプレートを使って 1 つのインスタンスタイプを渡さないでください。起動テンプレートを使用したノードグループのデプロイについての詳細は、「the section called “起動テンプレート”」をご覧ください。

クラスターのマネージドノードグループを作成する

このトピックでは、Amazon EKS クラスターに登録しているノードの、Amazon EKS マネージド型ノードグループを起動する方法を説明します。ノードがクラスターに参加したら、それらのノードに Kubernetes アプリケーションをデプロイ可能になります。

Amazon EKS マネージド型ノードグループを初めて起動する場合は、はじめに のガイドのいずれかに従うことをお勧めします。このガイドでは、ノードを使用して Amazon EKS クラスターを作成するためのウォークスルーを説明します。

> **Important**
>
> - Amazon EKS ノードは、標準の Amazon EC2 インスタンスです。通常の Amazon EC2 料金に基づいて請求されます。詳細については、Amazon EC2 の料金表 を参照してください。
> - AWS Outposts または AWS Wavelength が有効になっている AWS リージョンでは、マネージドノードを作成できません。ユーザーは代わりに、セルフマネージド型ノードを作成できます。詳細については、the section called “Amazon Linux”、the section called “Windows”、および the section called “Bottlerocket” を参照してください。また、Outpost に自己管理型 Amazon Linux ノードグループを作成することもできます。詳細については、the section called “ノード” を参照してください。
> - Amazon EKS 最適化 Linux または Bottlerocket に含まれる `bootstrap.sh` ファイル用に AMI ID を指定しない場合、マネージドノードグループは `maxPods` の値に最大数を適用します。vCPU が 30 未満のインスタンスの場合、最大数は `110` です。30 を超える vCPU を持つインスタンスの場合、最大数は `250` に跳ね上がります。この適用は、`maxPodsExpression` を含む他の `maxPods` 設定を上書きします。`maxPods` の決定方法とカスタマイズ方法の詳細については、「the section called “maxPods の決定方法”」を参照してください。

- 既存の Amazon EKS クラスター。デプロイするには、the section called “クラスターを作成する” を参照してください。
- ノードが使用する既存の IAM ロール。作成する場合は、the section called “ノード の IAM ロール” を参照してください。この役割に VPC CNI のポリシーがどちらも含まれていない場合、VPC CNI ポッドには以下の別の役割が必要です。
- (オプションですが推奨) 必要な IAM ポリシーがアタッチされた独自の IAM ロールで構成された Amazon VPC CNI plugin for Kubernetes アドオン。詳細については、the section called “IRSA 用の設定” を参照してください。
- 「最適な Amazon EC2 ノードインスタンスタイプを選択する」に記載されている考慮事項に精通していること。選択したインスタンスタイプによってはクラスターと VPC に関する追加の前提条件がある場合もあります。
- Windows マネージドノードグループを追加するには、まずクラスターの Windows サポートを有効にする必要があります。詳細については、the section called “Windows サポートを有効にする” を参照してください。

マネージド型ノードグループを作成するには、以下のいずれかを使用します。

- the section called “eksctl”
- the section called “AWS マネジメントコンソール”

eksctl

eksctl を使用してマネージド型ノードグループを作成する

この手順には、eksctl バージョン 0.215.0 以降が必要です。お使いのバージョンは、以下のコマンドを使用して確認できます。

```
eksctl version
```

eksctl のインストールまたはアップグレードの手順については、eksctl ドキュメントの「インストール」を参照してください。

1. (オプション) [AmazonEKS_CNI_Policy] マネージド IAM ポリシーが Amazon EKS ノードの IAM ロールにアタッチされている場合は、代わりに Kubernetes aws-node サービスアカウントに関連付けた IAM ロールに割り当てることをお勧めします。詳細については、the section called “IRSA 用の設定” を参照してください。

2. カスタム起動テンプレートの有無にかかわらず、マネージド型ノードグループを作成します。起動テンプレートを手動で指定すると、ノードグループをより詳細にカスタマイズできます。たとえば、カスタム AMI をデプロイしたり、Amazon EKS 最適化 AMI の bootstrap.sh スクリプトの引数を指定したりできます。すべての使用可能なオプションとデフォルト値の一覧を表示するには、次のコマンドを入力します。

```
eksctl create nodegroup --help
```

次のコマンドで、*my-cluster* をクラスターの名前に置き換え、*my-mng* をノードグループの名前に置き換えます。ノードグループ名は 63 文字以下である必要があります。先頭は文字または数字でなければなりませんが、残りの文字にはハイフンおよびアンダースコアを含めることもできます。

> **Important**
>
> マネージド型ノードグループを最初に作成する際にカスタム起動テンプレートを使用しない場合は、後でノードグループに使用しないでください。カスタム起動テンプレートを指定しなかった場合、システムにより起動テンプレートが自動生成されます。これを手動で変更することはお勧めしません。自動生成された起動テンプレートを手動で変更すると、エラーが発生する場合があります。

起動テンプレートなし

eksctl は、デフォルトの Amazon EC2 起動テンプレートをアカウント内に作成し、指定したオプションに基づいて作成した起動テンプレートを使用してノードグループをデプロイします。--node-type の値を指定する前に、the section called “Amazon EC2 インスタンスタイプ” を参照してください。

ami-family を許可されているキーワードに置き換えます。詳細については、「eksctl ドキュメント」の「Setting the node AMI Family」(ノード AMI ファミリーの設定) を参照してください。*my-key* を Amazon EC2 キーペアまたはパブリックキーの名前に置き換えます。このキーは起動後のノードに SSH 接続するために使用されます。

> **Note**
>
> Windows の場合、このコマンドは SSH を有効にしません。代わりに Amazon EC2 キーペアをインスタンスに関連付け、インスタンスに RDP できるようにします。

Amazon EC2 キーペアをまだ持っていない場合は、AWS マネジメントコンソール で作成できます。Linux の詳細については、「Amazon EC2 ユーザーガイド」の「Amazon EC2 key pairs and Linux instances」を参照してください。Windows の詳細については、「Amazon EC2 ユーザーガイド」の「Amazon EC2 key pairs and Windows instances」を参照してください。

次の条件が true の場合、IMDS への Pod アクセスをブロックすることをお勧めします。

- IAM ロールをすべての Kubernetes サービスアカウントに割り当てることによって、必要最小限のアクセス許可のみを Pod に付与しようとしている。
- クラスター内の Pod が、現在の AWS リージョンの取得など、その他の理由で Amazon EC2 インスタンスメタデータサービス (IMDS) へのアクセスを必要としていない。

詳細については、「ワーカーノードに割り当てられたインスタンスプロファイルへのアクセスを制限する」を参照してください。

IMDS への Pod のアクセスをブロックするには、`--disable-pod-imds` オプションを次のコマンドに追加します。

```
eksctl create nodegroup \
  --cluster my-cluster \
  --region region-code \
  --name my-mng \
  --node-ami-family ami-family \
  --node-type m5.large \
  --nodes 3 \
  --nodes-min 2 \
  --nodes-max 4 \
  --ssh-access \
  --ssh-public-key my-key
```

ご使用のインスタンスでは、オプションで大量の IP アドレスをポッドに割り当てることや、インスタンスではなく CIDR ブロックから Pod に IP アドレスを割り当てることができます。また、そのインスタンスはインターネットにアクセスできないクラスターにデプロイすることも可能です。詳細に

ついては、追加オプションのthe section called “IP アドレスを増やす”、the section called “カスタムネットワーキング” および the section called “プライベートクラスター” を参照して、前のコマンドに追加します。

マネージドノードグループは、インスタンスタイプに基づいて、ノードグループの各ノードで実行できる Pod の最大数に対して 1 つの値を計算して適用します。異なるインスタンスタイプを持つノードグループを作成する場合、すべてのインスタンスタイプで計算された最小値が、ノードグループ内のすべてのインスタンスタイプで実行できる Pod の最大数として適用されます。この値の計算方法の詳細については、the section called “maxPods の決定方法” を参照してください。

起動テンプレートの使用

起動テンプレートがすでに存在している必要があり、また、「起動テンプレート設定の基本」で指定されている要件を満たしている必要があります。次の条件が true の場合、IMDS への Pod アクセスをブロックすることをお勧めします。

- IAM ロールをすべての Kubernetes サービスアカウントに割り当てることによって、必要最小限のアクセス許可のみを Pod に付与しようとしている。
- クラスター内の Pod が、現在の AWS リージョンの取得など、その他の理由で Amazon EC2 インスタンスメタデータサービス (IMDS) へのアクセスを必要としていない。

詳細については、「ワーカーノードに割り当てられたインスタンスプロファイルへのアクセスを制限する」を参照してください。

IMDS への Pod のアクセスをブロックするには、起動テンプレートで必要な設定を行います。

a. 次のコンテンツをデバイスにコピーします。サンプル値を置き換えたら、変更したコマンドを実行して `eks-nodegroup.yaml` ファイルを作成します。起動テンプレートなしでデプロイしたときに指定したいくつかの設定は、起動テンプレートに移動されます。`version` を指定しない場合は、テンプレートのデフォルトバージョンが使用されます。

```
cat >eks-nodegroup.yaml <<EOF
apiVersion: eksctl.io/v1alpha5
kind: ClusterConfig
metadata:
  name: my-cluster
  region: region-code
managedNodeGroups:
- name: my-mng
  launchTemplate:
```

```
    id: lt-id
    version: "1"
EOF
```

eksctl 設定ファイル使用の詳細については、eksctl ドキュメントの「Config file schema」を参照してください。オプションで、インスタンスはポッドに大量の IP アドレスを割り当てたり、インスタンスのブロックとは異なる CIDR ブロックから IP アドレスをポッドに割り当てたり、アウトバウンドのインターネットアクセスのないクラスターにデプロイしたりできます。詳細については、the section called “IP アドレスを増やす”、the section called “カスタムネットワーキング”、および the section called “プライベートクラスター” を参照して、設定ファイルに追加する追加オプションを確認してください。

起動テンプレートで AMI ID を指定しなかった場合、マネージドノードグループは、インスタンスタイプに基づいて、ノードグループの各ノードで実行できる Pod の最大数に対して 1 つの値を計算して適用します。異なるインスタンスタイプを持つノードグループを作成する場合、すべてのインスタンスタイプで計算された最小値が、ノードグループ内のすべてのインスタンスタイプで実行できる Pod の最大数として適用されます。この値の計算方法の詳細については、the section called “maxPods の決定方法” を参照してください。

起動テンプレートで AMI ID を指定した場合、カスタムネットワークを使用しているか、またはインスタンスに割り当てられている IP アドレスの数を増やす場合には、ノードグループの各ノードで実行できる Pod の最大数を指定します。詳細については、the section called “maxPods の決定方法” を参照してください。

b. 次のコマンドでノードグループをデプロイします。

```
eksctl create nodegroup --config-file eks-nodegroup.yaml
```

AWS マネジメントコンソール

AWS マネジメントコンソール を使用してマネージド型ノードグループを作成する

1. クラスターステータスが ACTIVE と表示されるまで待ちます。まだ ACTIVE ではないクラスターにはマネージド型ノードグループを作成できません。
2. Amazon EKS コンソールを開きます。
3. マネージド型ノードグループを作成するクラスターの名前を選択します。
4. [コンピューティング] タブを選択します。

5. [ノードグループを追加] を選択します。
6. [ノードグループの設定] ページで、必要に応じてパラメータを指定し、[次へ] を選択します。
   - [名前] – マネージド型ノードグループの一意の名前を入力します。ノードグループ名は 63 文字以下である必要があります。先頭は文字または数字でなければなりませんが、残りの文字にはハイフンおよびアンダースコアを含めることもできます。
   - [ノード IAM ロール] – ノードグループで使用するノードインスタンスロールを選択します。詳細については、the section called “ノード の IAM ロール” を参照してください。

     > **Important**
     > - クラスターの作成に使用したロールは使用できません。
     > - 新しいセルフマネージドノードグループで使用する予定がない限り、現在どのセルフマネージドノードグループでも使用されていないロールを使用することをお勧めします。詳細については、「the section called “削除”」を参照してください。

   - 起動テンプレートを使用する — (オプション) 既存の起動テンプレートを使用するかどうかを選択します。[起動テンプレート名] を選択します。次に、[起動テンプレートのバージョン] を選択します。バージョンを選択しない場合、Amazon EKS はテンプレートのデフォルトのバージョンを使用します。起動テンプレートを使用すると、ノードグループを詳細にカスタマイズできます。これにより、カスタム AMI のデプロイ、ポッドへの大量の IP アドレスの割り当て、インスタンスのブロックとは異なる CIDR ブロックからのポッドに対する IP アドレスの割り当て、さらにアウトバウンドのインターネットアクセスのないクラスターに対するノードのデプロイが可能になります。詳細については、the section called “IP アドレスを増やす”、the section called “カスタムネットワーキング”、および the section called “プライベートクラスター” を参照してください。

     起動テンプレート は、「起動テンプレートを使用してマネージドノードをカスタマイズする」の要件を満たしている必要があります。独自の起動テンプレートを使用しない場合、Amazon EKS API はデフォルトの Amazon EC2 起動テンプレートをアカウントに作成し、デフォルトの起動テンプレートを使用してノードグループをデプロイします。

     サービスアカウントの IAM ロールを実装する場合は、AWS サービスへのアクセス許可を必要とするすべての Pod に必要なアクセス許可を直接割り当て、クラスター内の Pod が、現在の AWS リージョンを取得するなどの理由で IMDS にアクセスしないようにします。また、起動テンプレートでホストネットワークを使用しないポッドの、IMDS へのアクセスを無効にすること

もできます。詳細については、「ワーカーノードに割り当てられたインスタンスプロファイルへのアクセスを制限する」を参照してください。

- [Kubernetes ラベル]: (オプション) マネージド型ノードグループ内のノードに Kubernetes ラベルを適用するように選択できます。
- Kubernetes テイント - (オプション) 管理対象ノードグループ内のノードに Kubernetes テイントを適用することを選択できます。[効果] メニューでの利用可能なオプションは **`NoSchedule`**、**`NoExecute`**、および **`PreferNoSchedule`** です。詳細については、the section called "特定のノードで Pod がスケジュールされないようにする" を参照してください。
- [タグ] – (オプション) Amazon EKS マネージド型ノードグループにタグを付けることを選択できます。これらのタグは、Auto Scaling グループやインスタンスなど、ノードグループ内の他のリソースには伝達されません。詳細については、the section called "リソースのタグ付け" を参照してください。

7. [コンピューティング構成とスケーリングの設定] ページで、必要に応じてパラメータを指定し、[次へ] を選択します。
   - [AMI タイプ] – AMI タイプを選択します。Arm インスタンスをデプロイする場合は、デプロイする前に「Amazon EKS optimized Arm Amazon Linux AMIs」の考慮事項を確認してください。

     前のページで起動テンプレートを指定し、起動テンプレートで AMI を指定した場合は、値を選択できません。テンプレートの値が表示されます。テンプレートで指定された AMI は、「AMI の指定」の要件を満たしている必要があります。
   - [キャパシティータイプ] – キャパシティタイプを選択します。キャパシティータイプの選択の詳細については、the section called "マネージド型ノードグループのキャパシティータイプ" を参照してください。同じノードグループ内で、異なるキャパシティータイプを混在させることはできません。両方のキャパシティータイプを使用したい場合は、キャパシティータイプとインスタンスタイプをそれぞれに持つ、別々のノードグループを作成します。GPU アクセラレーテッドワーカーノードのプロビジョニングとスケーリングについては、「the section called "マネージドノードグループとセルフマネージドノード"」を参照してください。
   - [インスタンスタイプ] – デフォルトで 1 つまたは複数のインスタンスタイプが指定されています。デフォルトのインスタンスタイプを削除するには、インスタンスタイプの右側にある X を選択します。マネージド型ノードグループで使用するインスタンスタイプを選択します。詳細については、the section called "Amazon EC2 インスタンスタイプ" を参照してください。

     コンソールには、一般的に使用されるインスタンスタイプのセットが表示されます。表示されてないインスタンスタイプを持つマネージド型ノードグループの作成が必要な場合は、`eksctl`、AWS CLI、AWS CloudFormation、または SDK を使用して、ノードグループを作成します。前のページで起動テンプレートを指定した場合、起動テンプレートでインスタン

スタイプを指定する必要があるため、値を選択できません。起動テンプレートの値が表示されます。[キャパシティータイプ] で [スポット] を選択した場合は、可用性を高めるために、複数のインスタンスタイプを指定することをお勧めします。

- [ディスクサイズ] – ノードのルートボリュームに使用するディスクサイズ (GiB 単位) を入力します。

  前のページで起動テンプレートを指定した場合は、値を起動テンプレートで指定する必要があるため、値を選択できません。

- [必要なサイズ] – マネージド型ノードグループが起動時に保持する必要があるノードの現在の数を指定します。

  > **Note**
  >
  > Amazon EKS は、ノードグループを自動的にスケールインまたはスケールアウトしません。ただし、これを行うように Kubernetes Cluster Autoscaler を設定することはできます。詳細については、「AWS の Cluster Autoscaler」を参照してください。

- [最小サイズ] – マネージド型ノードグループがスケールインできるノードの最小数を指定します。
- [最大サイズ] – マネージド型ノードグループがスケールアウトできるノードの最大数を指定します。
- ノードグループの更新設定 – (オプション) 並行して更新するノードの数または割合を選択できます。これらのノードは、更新中は使用できません。[使用できない最大値] で、次のいずれかのオプションを選択し、その [値] を指定します:
  - [数値] – 並行して更新できるノードグループ内のノード数を選択して指定します。
  - [パーセンテージ] – 並行して更新できるノードグループ内のノードの割合を選択して指定します。ノードグループに多数のノードがある場合に便利です。
- ノードの自動修復設定 – (オプション) [ノードの自動修復を有効にする] チェックボックスを有効にすると、Amazon EKS は検出された問題が発生したときにノードを自動的に置き換えます。詳細については、the section called “ノードのヘルスと修復” を参照してください。
- ウォームプール設定 – (オプション) ウォームプール設定を有効にする チェックボックスを有効にすると、Amazon EKS は ASG にウォームプールを作成します。詳細については、the section called “ウォームプール” を参照してください。

8. [ネットワーキングを指定] ページで、必要に応じてパラメータを指定し、[次へ] を選択します。
   - [サブネット]: マネージド型ノードを起動するサブネットを選択します。

> **Important**
>
> Amazon EBS ボリュームによってバックアップされ、Kubernetes Cluster Autoscaler を使用する複数のアベイラビリティーゾーンにわたってステートフルアプリケーションを実行している場合、それぞれが単一のアベイラビリティーゾーンにスコープされる複数のノードグループを設定する必要があります。また、`--balance-similar-node-groups` 機能を有効にする必要があります。

> **Important**
>
> - パブリックサブネットを選択し、クラスターでパブリック API サーバーのエンドポイントのみが有効になっている場合は、サブネットの `MapPublicIPOnLaunch` に `true` をセットして、インスタンスがクラスターに正常に参加できるようにします。サブネットが 2020 年 3 月 26 日以降に `eksctl` または Amazon EKS で発行された AWS CloudFormation テンプレートを使用して作成された場合は、この設定はすでに `true` に設定されています。サブネットが 2020 年 3 月 26 日より前に `eksctl` または AWS CloudFormation テンプレートを使用して作成されている場合は、設定を手動で変更する必要があります。詳細については「サブネットの IPv4 アドレス指定属性の変更」を参照してください。
> - 起動テンプレートを使用しており、複数のネットワークインターフェイスを指定している場合には、たとえ `MapPublicIpOnLaunch` が `true` に設定されていても、Amazon EC2 はパブリック `IPv4` アドレスの自動的な割り当てを行いません。このシナリオでノードがクラスターに参加するには、クラスターのプライベート API サーバーエンドポイントを有効にするか、NAT ゲートウェイなどの別の方法によってアウトバウンドインターネットアクセスを提供する、プライベートサブネットでノードを起動する必要があります。詳細については、「Amazon EC2 ユーザーガイド」の「Amazon EC2 インスタンスの IP アドレス指定」を参照してください。

- ノードへの SSH アクセスの設定 (オプション)。SSH を有効にすることにより、インスタンスに接続し、問題がある場合に診断情報を収集できます。ノードグループを作成するときは、リモートアクセスを有効にすることを強くお勧めします。ノードグループの作成後にリモートアクセスを有効にすることはできません。

  起動テンプレートの使用を選択した場合、このオプションは表示されません。ノードへのリモートアクセスを有効にするには、起動テンプレートでキーペアを指定し、起動テンプレート

で指定したセキュリティグループのノードに対して適切なポートが開いていることを確認します。詳細については、the section called “カスタムセキュリティグループを使用する” を参照してください。

> **Note**
>
> Windows の場合、このコマンドは SSH を有効にしません。代わりに Amazon EC2 キーペアをインスタンスに関連付け、インスタンスに RDP できるようにします。

- [SSH キーペア] (オプション) の場合は、使用する Amazon EC2 SSH キーを選択します。Linux の詳細については、「Amazon EC2 ユーザーガイド」の「Amazon EC2 key pairs and Linux instances」を参照してください。Windows の詳細については、「Amazon EC2 ユーザーガイド」の「Amazon EC2 key pairs and Windows instances」を参照してください。起動テンプレートを使用することを選択した場合、選択することはできません。Bottlerocket AMI を使用するノードグループに Amazon EC2 SSH キーが提供されると、管理コンテナも有効になります。詳細については、GitHub の Admin container を参照してください。
- [次からの SSH リモートアクセスを許可する] の場合、特定のインスタンスへのアクセスを制限するには、それらのインスタンスに関連付けられているセキュリティグループを選択します。特定のセキュリティグループを選択しないと、インターネット上のどの場所 (`0.0.0.0/0`) からでも SSH アクセスが許可されます。

9. [確認と作成] ページで、マネージド型ノードグループの設定を確認し、[作成] を選択してください。

   ノードがクラスターに参加できない場合は、トラブルシューティングの章にある the section called “ノードがクラスターへの参加に失敗します” を参照してください。

10. ノードのステータスを監視し、`Ready` ステータスになるまで待機します。

```
kubectl get nodes --watch
```

11. (GPU ノードのみ) GPU インスタンスタイプと Amazon EKS 最適化高速 AMI を選択した場合は、クラスター上の DaemonSet として Kubernetes 用の NVIDIA デバイスプラグインを適用する必要があります。次のコマンドを実行する前に、*`vX.X.X`* を必要となる NVIDIA/k8s-device-plugin バージョンに置き換えます。

```
kubectl apply -f https://raw.githubusercontent.com/NVIDIA/k8s-device-plugin/vX.X.X/
deployments/static/nvidia-device-plugin.yml
```

Kubernetes アドオンをインストールする

ノードが関連付けられた Amazon EKS クラスターが実行中になったところで、Kubernetes アドオンのインストールとクラスターへのアプリケーションのデプロイを開始できます。以下のトピックはクラスターの機能を拡張するのに役立ちます。

- クラスターを作成した IAM プリンシパルは、kubectl または AWS マネジメントコンソール を使用して Kubernetes API サーバーを呼び出すことができる唯一のプリンシパルです。他の IAM プリンシパルがクラスターにアクセスできるようにする場合はそれらを追加する必要があります。詳細については、the section called “Kubernetes API アクセス” および the section called “必要なアクセス許可” を参照してください。
- 次の条件が true の場合、IMDS への Pod アクセスをブロックすることをお勧めします。
  - IAM ロールをすべての Kubernetes サービスアカウントに割り当てることによって、必要最小限のアクセス許可のみを Pod に付与しようとしている。
  - クラスター内の Pod が、現在の AWS リージョンの取得など、その他の理由で Amazon EC2 インスタンスメタデータサービス (IMDS) へのアクセスを必要としていない。

  詳細については「ワーカーノードに割り当てられたインスタンスプロファイルへのアクセスを制限する」を参照してください。
- ノードグループ内のノード数を自動的に調整するように Kubernetes Cluster Autoscaler を設定します。
- サンプルアプリケーション をクラスターにデプロイします。
- クラスターを管理するための重要なツールを使用して、クラスターリソースを整理およびモニタリングします。

マネージド型ノードグループを備えたウォームプールを使用して、起動時間の長いアプリケーションのレイテンシーを低減する

アプリケーションの初期化または起動時間が長い場合、スケールアウトイベントによって遅延が発生する可能性があります。ポッドをスケジュールする前に、新しいノードが完全に起動してクラスターに参加する必要があります。このレイテンシーは、トラフィックの急増時や急激な規模拡大時に、アプリケーションの可用性に影響を与える可能性があります。ウォームプールは、起動プロセスがすでに完了している初期化済みの EC2 インスタンスのプールを維持することでこの問題を解決します。スケールアウトイベント中、インスタンスはウォームプールから直接クラスターに移動し、時間のかかる初期化手順がスキップされ、新しい容量が利用可能になるまでの時間が大幅に短縮されます。詳

細については、Amazon EC2 Auto Scaling ユーザーガイド」の「ウォームプールを使用して起動時間が長いアプリケーションのレイテンシーを減らす」を参照してください。

Amazon EKS マネージド型ノードグループは、Amazon EC2 Auto Scaling ウォームプールをサポートしています。ウォームプールは、スケールアウトイベント中にクラスターにすばやく参加できる Auto Scaling グループとともに、事前に初期化された EC2 インスタンスを維持します。ウォームプールのインスタンスはすでに起動初期化プロセスを完了しており、`Stopped`、`Running`、または `Hibernated` 状態に保持できます。

Amazon EKS は、`AWSServiceRoleForAmazonEKSNodegroup` サービスにリンクされたロールを使用してノードグループのライフサイクル全体でウォームプールを管理し、ウォームプールリソースを作成、更新、削除します。

仕組み

ウォームプールを設定すると、Amazon EKS はノードグループのAuto Scaling グループにアタッチされた EC2 自動スケーリングウォームプールを作成します。インスタンスはウォームプールで起動し、起動の初期化プロセスを完了し、必要に応じて設定された状態 (`Running`、`Stopped`、または `Hibernated`) を維持します。スケールアウトイベント中、インスタンスはウォームプールからAuto Scaling グループに移動し、Amazon EKS 初期化プロセスを完了してクラスターに参加し、ポッドスケジューリングに使用できるようになります。インスタンスの再利用が有効になっている場合、インスタンスはスケールインイベント中にウォームプールに戻ることができます。

> **Important**
>
> ウォームプールは、必ず `create-nodegroup` または `update-nodegroup-config` を使用して Amazon EKS API を介して設定します。EC2 自動スケーリング API を使用してウォームプール設定を手動で変更しないでください。リソースの Amazon EKS 管理と競合する可能性があります。

考慮事項

> **Important**
>
> ウォームプールを設定する前に、Amazon EC2 Auto Scaling ユーザーガイド」の「Amazon EC2 Auto Scaling のウォームプール」の前提条件と制限を確認してください。すべてのインスタンスタイプ、AMI、または設定がサポートされているわけではありません。

- IAM アクセス許可 – AWSServiceRoleForAmazonEKSNodegroup サービスにリンクされたロール (最初のマネージドノードグループで自動的に作成されます) には、必要なウォームプール管理アクセス許可が含まれています。
- AMI の制限 — ウォームプールはカスタム AMI をサポートしていません。Amazon EKS 最適化 AMI を使用する必要があります。
- Bottlerocket の制限 – Bottlerocket AMI を使用している場合、Hibernated プールの状態はサポートされていません。Stopped または Running プールの状態のみを使用します。さらに、reuseOnScaleIn 機能は Bottlerocket AMI ではサポートされていません。
- 休止のサポート – Hibernated プールの状態は、特定のインスタンスタイプでのみサポートされます。サポートされているインスタンスタイプについては、Amazon EC2 ユーザーガイド の ハイバネーションの前提条件 を参照してください。
- コストへの影響 – 必要のないときにウォームプールを作成すると、不要なコストが発生する可能性があります。
- キャパシティプランニング – コストと可用性のバランスを取るために、スケーリングパターンに基づいてウォームプールの規模を決定します。予想されるピーク容量の 10 ~ 20% から開始します。
- VPC ネットワーク – Auto Scaling グループインスタンスとウォームプールインスタンスの両方に十分な IP アドレスを確保します。

ウォームプールを設定

新しいマネージド型ノードグループを作成するときにウォームプールを設定したり、既存のマネージド型ノードグループを更新してウォームプールのサポートを追加したりできます。

設定パラメータ

- enabled – (ブール値) マネージド型ノードグループにウォームプールをアタッチするインテントを示します。ウォームプールのサポートを有効にするために必要です。
- maxGroupPreparedCapacity – (整数) ウォームプールとAuto Scaling グループの合計最大合計インスタンス数。
- minSize – (整数) ウォームプールに維持するインスタンスの最小数。デフォルト: 0。
- poolState – (文字列) ウォームプールインスタンスの状態。デフォルト: Stopped。
- reuseOnScaleIn – (ブール値) インスタンスがスケールインイベント中に終了する代わりにウォームプールに戻るかどうか。デフォルト: false。Bottlerocket AMI ではサポートされていません。

AWS CLI の使用

マネージド型ノードグループを作成するときにウォームプールを設定することも、既存のノードグループに追加することもできます。

ウォームプールを持つノードグループを作成

```
aws eks create-nodegroup \
  --cluster-name my-cluster \
  --nodegroup-name my-nodegroup \
  --node-role arn:aws:iam::111122223333:role/AmazonEKSNodeRole \
  --subnets subnet-12345678 subnet-87654321 \
  --region us-east-1 \
  --scaling-config minSize=2,maxSize=10,desiredSize=3 \
  --warm-pool-config
 enabled=true,maxGroupPreparedCapacity=8,minSize=2,poolState=Stopped,reuseOnScaleIn=true
```

既存のノードグループにウォームプールを追加

```
aws eks update-nodegroup-config \
  --cluster-name my-cluster \
  --nodegroup-name my-nodegroup \
  --region us-east-1 \
  --warm-pool-config
 enabled=true,maxGroupPreparedCapacity=8,minSize=2,poolState=Stopped,reuseOnScaleIn=true
```

設定の更新

update-nodegroup-config を使用して、いつでもウォームプールの設定を更新します。既存のウォームプールインスタンスはすぐには影響を受けません。新しい設定は、更新後にウォームプールに入るインスタンスに適用されます。

```
aws eks update-nodegroup-config \
  --cluster-name my-cluster \
  --nodegroup-name my-nodegroup \
  --region us-east-1 \
  --warm-pool-config
 enabled=true,maxGroupPreparedCapacity=10,minSize=3,poolState=Running,reuseOnScaleIn=true
```

ノードグループにアタッチされたウォームプールを無効にするには、enabled=false を設定します。

```
aws eks update-nodegroup-config \
  --cluster-name my-cluster \
  --nodegroup-name my-nodegroup \
  --region us-east-1 \
  --warm-pool-config enabled=false
```

その他のリソース

- 詳細については、Amazon EC2 Auto Scaling ユーザーガイド の Warm pools for Amazon EC2 Auto Scaling を参照してください。
- マネージド型ノードグループを使用してノードライフサイクルを簡素化

クラスターのためにマネージドノードグループを更新する

マネージド型ノードグループの更新を開始すると、Amazon EKS はノードを自動的に更新し、「ノードの更新の各フェーズを理解する」でリストされた手順を完了します。Amazon EKS 最適化 AMI を使用している場合、Amazon EKS は最新のセキュリティパッチとオペレーティングシステムの更新を、最新の AMI リリースバージョンの一部として自動的にノードに適用します。

いくつかのシナリオでは、Amazon EKS マネージド型ノードグループのバージョンや設定を更新すると便利です。

- Amazon EKS クラスターの Kubernetes バージョンを更新し、同じ Kubernetes バージョンを使用するようにノードを更新する場合。
- マネージド型ノードグループでは、新しい AMI リリースバージョンを使用できます。AMI バージョンの詳細については、次のセクションを参照してください。
  - the section called “バージョン情報を取得する”
  - the section called “Bottlerocket”
  - the section called “バージョン情報を取得する”
- マネージド型ノードグループ内のインスタンスの最小数、最大数、または必要な数を調整する場合。
- マネージド型ノードグループのインスタンスで Kubernetes ラベルを追加または削除する場合。
- マネージド型ノードグループに AWS タグを追加または削除する場合。
- カスタム AMI の更新などの設定の変更を伴う、新しいバージョンの起動テンプレートをデプロイする必要があります。

- Amazon VPC CNI アドオンのバージョン `1.9.0` 以降をデプロイし、プレフィックス委任用のアドオンを有効にし、ノードグループ内の新しい AWS Nitro System インスタンスで大幅に増加した Pod の数をサポートしたい場合。詳細については、「the section called “IP アドレスを増やす”」を参照してください。
- Windows ノードの IP プレフィックスの委任を有効にし、ノードグループ内の新しい AWS Nitro System インスタンスで、大幅に増加したポッドをサポートした場合。詳細については、「the section called “IP アドレスを増やす”」を参照してください。

マネージドノードグループの Kubernetes バージョンに対して、新しい AMI リリースバージョンがある場合は、ノードグループのバージョンを更新して、新しい AMI バージョンを使用することができます。同様に、クラスターがノードグループより新しい Kubernetes バージョンを実行している場合、クラスターの Kubernetes バージョンに一致する最新の AMI リリースバージョンを使用するようにノードグループを更新できます。

スケーリングオペレーションまたは更新によってマネージドノードグループ内のノードが終了すると、そのノードの Pod が最初にドレインされます。詳細については、「the section called “動作の詳細を更新する”」を参照してください。

ノードグループバージョンの更新

ノードグループのバージョンは、次のいずれかを使用して更新できます。

- the section called “`eksctl`”
- the section called “AWS マネジメントコンソール”

更新するバージョンは、コントロールプレーンバージョンよりも新しいバージョンにすることはできません。

`eksctl`

`eksctl` を使用してマネージドノードグループを更新する

次のコマンドを使用して、マネージドノードグループを、ノードに現在デプロイされているのと同じ Kubernetes バージョンの最新 AMI リリースに更新します。各`#####`は独自の値に置き換えます。

```
eksctl upgrade nodegroup \
  --name=node-group-name \
  --cluster=my-cluster \
```

```
  --region=region-code
```

> **Note**
>
> 起動テンプレートでデプロイされたノードグループを、新しい起動テンプレートのバージョンにアップグレードする場合は、上記のコマンドに `--launch-template-version` *`version-number`* を追加します。起動テンプレート は、「起動テンプレートを使用してマネージドノードをカスタマイズする」に記載されている要件を満たしている必要があります。起動テンプレートにカスタム AMI が含まれている場合、AMI は「Specifying an AMI」(AMI の指定) の要件を満たしている必要があります。ノードグループを新しいバージョンの起動テンプレートにアップグレードすると、指定した起動テンプレートバージョンの新しい設定と一致するように、すべてのノードがリサイクルされます。

起動テンプレートなしでデプロイしたノードグループを、新しい起動テンプレートバージョンに直接アップグレードすることはできません。代わりに、起動テンプレートを使用して新しいノードグループをデプロイし、新しい起動テンプレートバージョンにノードグループを更新する必要があります。

コントロールプレーンの Kubernetes バージョンと同じバージョンにノードグループをアップグレードできます。例えば、Kubernetes `1.37` を実行しているクラスターがある場合、次のコマンドを使用して、現在 Kubernetes `1.36` を実行しているノードをバージョン `1.37` にアップグレードできます。

```
eksctl upgrade nodegroup \
  --name=node-group-name \
  --cluster=my-cluster \
  --region=region-code \
  --kubernetes-version=1.37
```

AWS マネジメントコンソール

を使用してマネージドノードグループを更新するAWS マネジメントコンソール

1. Amazon EKS コンソールを開きます。
2. 更新するノードグループを含むクラスターを選択します。
3. 少なくとも 1 つのノードグループに利用可能な更新がある場合、ページの上部に利用可能な更新について通知するボックスが表示されます。[Compute] (コンピューティング) タブを選択すると、利用可能な更新があるノードグループの [Node Groups] (ノードグループ) 表の、[AMI release

version] (AMI リリースバージョン) 列に、[Update now] (今すぐ更新) が表示されます。ノードグループを更新するには、[Update now] (今すぐ更新) を選択します。

カスタム AMI でデプロイされたノードグループの通知は表示されません。ノードがカスタム AMI でデプロイされている場合は、次の手順を実行して、新しく更新されたカスタム AMI をデプロイします。

a. AMI の新しいバージョンを作成します。

b. 新しい AMI ID を使用して、新しい起動テンプレートのバージョンを作成します。

c. 起動テンプレートの新しいバージョンにノードを更新する

4. [Update node group version] (ノードグループバージョンの更新) ダイアログボックスで、次のオプションを有効または無効にします。

- [Update node group version] (ノードグループのバージョンの更新) - カスタム AMI をデプロイした場合、あるいは Amazon EKS に最適化された AMI が現在のクラスターで最新のバージョンである場合には、このオプションは利用できません。
- [Change launch template version] (起動テンプレートバージョンの変更) - ノードグループがカスタムの起動テンプレートを使用せずにデプロイされている場合、このオプションは利用できません。カスタム起動テンプレートを使用してデプロイされたノードグループの、起動テンプレートのバージョンのみ更新できます。ノードグループを更新する [Launch template version] (起動テンプレートバージョン) を選択します。ノードグループがカスタム AMI で設定されている場合は、選択するバージョンも AMI を指定する必要があります。新しいバージョンの起動テンプレートにアップグレードすると、指定した起動テンプレートバージョンの新しい設定と一致するように、すべてのノードがリサイクルされます。

5. [Update strategy] (更新戦略) で、次のいずれかのオプションを選択します。

- [ローリング更新]: このオプションは、クラスターの Pod の中断予算を尊重します。Pod 中断予算の問題により、Amazon EKS がこのノードグループで実行されている Pod を正常にドレインできない場合、更新が失敗します。
- [強制更新]: このオプションは Pod の中断予算を尊重しません。ノードの再起動を強制的に実行することにより、Pod の中断予算の問題に関係なく更新が行われます。

6. [更新] を選択します。

ノードグループ設定の編集

マネージド型ノードグループの設定の一部を変更できます。

1. Amazon EKS コンソールを開きます。

2. 編集するノードグループを含むクラスターを選択します。
3. [Compute] (コンピューティング) タブを選択します。
4. 編集するノードグループを選択し、次に [Edit] (編集) を選択します。
5. (オプション) [Edit Node Group] (ノードグループの編集) ページで以下を実行します。
   a. ノードグループのスケーリング設定を編集します。
      - [Desired size] (必要なサイズ) - マネージド型ノードグループが保持する必要があるノードの現在の数を指定します。
      - [最小サイズ] – マネージド型ノードグループがスケールインできるノードの最小数を指定します。
      - [最大サイズ]: マネージド型ノードグループがスケールアウトできるノードの最大数を指定します。ノードグループでサポートされるノードの最大数については、「the section called “Service Quotas”」を参照してください。
   b. (オプション) ノードグループ内のノードに [Kubernetes ラベル] を追加または削除します。ここに示すラベルは、Amazon EKS で適用したラベルのみです。ここには表示されていない他のラベルがノードに存在する可能性があります。
   c. (オプション) ノードグループ内のノードに [Kubernetes テイント] を追加または削除します。追加されたテイントは、 **`NoSchedule`** 、 **`NoExecute`** 、または **`PreferNoSchedule`** の影響があります。詳細については、「the section called “特定のノードで Pod がスケジュールされないようにする”」を参照してください。
   d. (オプション) ノードグループリソースに[Tags] (タグ) を追加または削除します。これらのタグは、Amazon EKS ノードグループにのみ適用されます。これらは、Amazon EC2 インスタンスやサブネットなど、ノードグループの他のリソースには伝達されません。
   e. (オプション) ノードグループの更新設定を編集します。[Number] (数値) または [Percentage] (パーセンテージ) のいずれかを選択します。
      - [数値]: 並行して更新できるノードグループ内のノード数を選択して指定します。これらのノードは、更新中は使用できません。
      - [パーセンテージ]: 並行して更新できるノードグループ内のノードの割合を選択して指定します。これらのノードは、更新中は使用できません。ノードグループに多数のノードがある場合に便利です。
   f. 編集が終了したら、[変更の保存] を選択します。

> **Important**
>
> ノードグループ設定を更新する場合、`NodegroupScalingConfig` を変更しても Pod 中断予算 (PDB) は考慮されません。ノードグループの更新プロセス (アップグレードフェーズ中にノードをドレインし、PDB を考慮する) とは異なり、スケーリング設定を更新すると、Auto Scaling グループ (ASG) スケールダウンコールによってノードがすぐに終了します。これは、スケールダウン先のターゲットサイズに関係なく、PDB を考慮せずに発生します。つまり、Amazon EKS マネージドノードグループの `desiredSize` を減らすと、ノードが終了するとすぐに、PDB を尊重せずに Pod が削除されます。

マネージドノードグループバージョンをロールバックする

クラスターコントロールプレーンを以前の Kubernetes バージョンにロールバックする必要がある場合は、まずマネージドノードグループをロールバックして、Kubernetes バージョンスキューポリシーに準拠する必要があります。ワーカーノードは、コントロールプレーンより新しいバージョンを実行できません。

以前の Kubernetes バージョンの `UpdateNodegroupVersion` API を使用して、マネージドノードグループをロールバックできます。

```
aws eks update-nodegroup-version \
  --cluster-name my-cluster \
  --nodegroup-name my-nodegroup \
  --kubernetes-version 1.30 \
  --region us-west-2
```

ノードグループのロールバックは、アップグレードと同様に、構成された更新設定 (`maxUnavailable` または `maxUnavailablePercentage`) と更新戦略 (Rolling または Force) を尊重します。

クラスターの完全なロールバックプロセスの詳細については、the section called “Kubernetes バージョンのロールバック” を参照してください。

ノードの更新の各フェーズを理解する

Amazon EKS マネージド型ワーカーノードのアップグレード戦略には、次のセクションで説明する 4 つの異なるフェーズがあります。

セットアップフェーズ

セットアップフェーズには、次の手順があります。

1. ノードグループに関連付けられた Auto Scaling グループ用に新しい Amazon EC2 起動テンプレートバージョンを作成します。新しい起動テンプレートバージョンでは、ターゲットの AMI またはカスタム起動テンプレートバージョンを更新に使用します。
2. 最新の起動テンプレートバージョンを使用するように、Auto Scaling グループを更新します。
3. ノードグループの `updateConfig` プロパティを使用して、並列でアップグレードするノードの最大数を決定します。使用不可の最大値には、クォータとして 100 ノードがあります。デフォルト値は 1 ノードです。詳細については、Amazon EKS API リファレンス 内の updateConfig プロパティを参照してください。

スケールアップフェーズ

マネージド型ノードグループ内のノードをアップグレードするとき、アップグレードされたノードは、アップグレードされているノードと同じアベイラビリティーゾーンで起動されます。この配置を保証するために、Amazon EC2 のアベイラビリティーゾーンの再調整を使用します。詳細については、Amazon EC2 Auto Scaling ユーザーガイドのアベイラビリティーゾーンの再調整を参照してください。この要件を満たすために、マネージド型ノードグループのアベイラビリティーゾーンごとに最大 2 つのインスタンスを起動できます。

スケールアップフェーズには、次の手順があります。

1. Auto Scaling グループの最大サイズと希望のサイズを、次のうちいずれか大きい方だけ増加させます。
   - Auto Scaling グループがデプロイされているアベイラビリティーゾーン数の最大 2 倍。
   - アップグレードができない最大値。

     たとえば、ノードグループのアベイラビリティーゾーンが 5 つで、`maxUnavailable` が 1 である場合、アップグレードプロセスで最大 10 個のノードを起動できます。しかし、`maxUnavailable` が 20 (または 10 より大きいいずれかの値) である場合、プロセスにより起動される新しいノードは 20 個です。
2. Auto Scaling グループのスケーリング後、最新の設定を使用するノードがノードグループに存在するかどうかをチェックします。この手順は、次の基準を満たす場合にのみ成功します。
   - ノードが存在するすべてのアベイラビリティーゾーンで、少なくとも 1 つの新しいノードが起動されます。

- 新しいノードはすべて、Ready 状態である必要があります。
- 新しいノードには Amazon EKS 適用ラベルが必要です。

  これは、通常のノードグループのワーカーノード上の Amazon EKS 適用ラベルです。

  - `eks.amazonaws.com/nodegroup-image=$amiName`
  - `eks.amazonaws.com/nodegroup=$nodeGroupName`

  これは、カスタム起動テンプレートまたは AMI ノードグループのワーカーノード上の Amazon EKS 適用ラベルです。

  - `eks.amazonaws.com/nodegroup-image=$amiName`
  - `eks.amazonaws.com/nodegroup=$nodeGroupName`
  - `eks.amazonaws.com/sourceLaunchTemplateId=$launchTemplateId`
  - `eks.amazonaws.com/sourceLaunchTemplateVersion=$launchTemplateVersion`

  > **Note**
  >
  > スケーリング設定を変更せずに更新またはアップグレードが開始されると、ワークフローはノードグループの保存されたスケーリング設定ではなく、ライブAuto Scaling グループの値を開始点として使用します。詳細については、the section called “マネージド型ノードグループの概念” を参照してください。

3. 新しい Pod がスケジュールされないように、ノードをスケジュール不可としてマークします。古いノードを終了する前にロードバランサーからノードを削除するため、ノードに `node.kubernetes.io/exclude-from-external-load-balancers=true` のラベルを付けます。

このフェーズで `NodeCreationFailure` エラーが発生する既知の原因を次に示します。

アベイラビリティーゾーンの容量が不十分です

アベイラビリティーゾーンに、リクエストされたインスタンスタイプの容量がない可能性があります。マネージド型ノードグループを作成するときは、複数のインスタンスタイプを設定することをお勧めします。

アカウント内の EC2 インスタンス制限

Service Quotas を使用してアカウントが同時に実行できる Amazon EC2 インスタンスの数を増やす必要がある場合があります。詳細については、Linux インスタンス向け Amazon Elastic Compute Cloud ユーザーガイドの EC2 の Service Quotas を参照してください。

カスタムユーザーデータ

カスタムユーザーデータは、ブートストラッププロセスを中断することがあります。このシナリオでは、ノードで `kubelet` が起動しないか、ノードで想定される Amazon EKS ラベルが取得されない可能性があります。詳細については、the section called “AMI を指定する” を参照してください。

ノードを異常な状態または準備ができていない状態にする変更

ノードのディスク負荷、メモリ負荷、および同様の条件により、ノードが `Ready` 状態にならない可能性があります。

各ノードのブートストラップにかかる時間は 15 分以内

ブートストラップしてクラスターに加わるまでに 15 分以上かかるノードがあると、アップグレードがタイムアウトします。この時間は、新しいノードが必要になってからクラスターに加わるまで、新しいノードのブートストラップに要した合計時間です。マネージドノードグループをアップグレードする場合は、Auto Scaling グループのサイズが大きくなると、すぐに時間の測定が始まります。

アップグレードフェーズ

アップグレードフェーズの動作には 2 通りの方法があり、更新戦略によって異なります。デフォルトと最小という 2 つの更新戦略があります。

通常は、デフォルト戦略をお勧めします。古いノードを終了する前に新しいノードを作成するという戦略で、アップグレードフェーズ中も使用可能な容量が維持されます。最小は、リソースやコスト (例えば、GPU などのハードウェアアクセラレーター) が限られている場合に便利な戦略です。新しいノードを作成する前に古いノードを終了するため、合計容量が設定しておいた数量を超えることはありません。

デフォルト更新戦略のステップは次のとおりです。

1. Auto Scaling グループ内のノード数を (必要な数だけ) 増やして、ノードグループに追加でノードを作成します。

2. ノードグループのために設定されている使用不可の最大数を上限として、アップグレードが必要なノードをランダムに選択します。
3. 新しいノードが準備完了状態になるとすぐに古いノードを遮断し、サービスコントローラーがそのノードに新しいリクエストを送信できなくなります。
4. ノードから Pod をドレインします。Pod が 15 分以内にノードを離れず、強制フラグがない場合は、PodEvictionFailure というエラーが表示されて、アップグレードフェーズは失敗します。この状況になった場合、update-nodegroup-version リクエストで強制フラグを適用して Pod を削除できます。
5. すべての Pod を削除してから 60 秒待機します。次に、Auto Scaling グループに終了リクエストを送信し、アクティブなノードのリストからノードを削除します。
6. 以前のバージョンの起動テンプレートでデプロイされたノードグループ内にノードが存在しなくなるまで、以前のアップグレード手順を繰り返します。

最小更新戦略のステップは次のとおりです。

1. ノードグループのすべてのノードを最初に遮断して、サービスコントローラーがこれらのノードに新しいリクエストを送信しないようにします。
2. ノードグループのために設定されている使用不可の最大数を上限として、アップグレードが必要なノードをランダムに選択します。
3. 選択したノードからポッドをドレインします。Pod が 15 分以内にノードを離れず、強制フラグがない場合は、PodEvictionFailure というエラーが表示されて、アップグレードフェーズは失敗します。この状況になった場合、update-nodegroup-version リクエストで強制フラグを適用して Pod を削除できます。
4. すべてのポッドを削除し 60 秒間待機すると、終了リクエストが、選択したノードのAuto Scaling グループに送信されます。Auto Scaling グループは、不足分の容量を補うために (選択したノードと同数の) 新しいノードを作成します。
5. 以前のバージョンの起動テンプレートでデプロイされたノードグループ内にノードが存在しなくなるまで、以前のアップグレード手順を繰り返します。

アップグレードフェーズ中の **PodEvictionFailure** エラー

このフェーズで PodEvictionFailure エラーが発生する既知の原因を次に示します。

アグレッシブ PDB

Pod にアグレッシブ PDB が定義されているか、同じ Pod を指す複数の PDB が存在します。

すべてのテイントを許容するデプロイ

すべての Pod が削除されたら、前のステップでノードがテイントされているため、ノードは空になるはずです。ただし、デプロイがすべてのテイントを許容する場合、ノードが空でない可能性が高くなり、Pod のエビクションが失敗します。

スケールダウンフェーズ

スケールダウンフェーズでは、Auto Scaling グループの最大サイズと希望するサイズが 1 ずつ減り、更新が開始される前の値に戻ります。

ワークフローのスケールダウンフェーズ中に Cluster Autoscaler がノードグループをスケールアップしているとアップグレードワークフローが判断した場合、ノードグループを元のサイズに戻すことなく、すぐに終了します。

> **Note**
>
> ```
> If your node group has a warm pool enabled, warm pool instances are drained
>  before the scale-up operation begins. This is because warm pool instances have
>  not been updated to the new launch template configuration – during the scale-
> up phase, they would be pulled into the Auto Scaling Group instead of launching
>  new instances with the updated configuration, which would break the upgrade
>  process. Draining the warm pool ensures that only new instances with the
>  updated configuration are launched. Once the scale-down operation completes,
>  the warm pool is restored, and the new instances in the warm pool are launched
>  with the updated launch template configuration.
> ```
>
> ウォームプールの詳細については、「the section called “ウォームプール”」を参照してください。

起動テンプレートを使用してマネージドノードをカスタマイズする

このページの手順に基づいて自分の起動テンプレートを使用することで、マネージド型ノードをデプロイでき、最高レベルのカスタマイズを実現できます。起動テンプレートを使用すると、ノードのデプロイ中にブートストラップ引数 (追加の kubelet 引数など) を指定したり、ノードに割り当てられた IP アドレスとは異なる CIDR ブロックからポッドに IP アドレスを割り当てたり、独自のカスタム AMI をノードにデプロイしたり、独自のカスタム CNI をノードにデプロイしたりできます。

最初にマネージドノードグループを作成するときに独自の起動テンプレートを指定する場合にも、後で高い柔軟性を得ることができます。自分の起動テンプレートを使用してマネージド型ノードグループをデプロイする限り、同じ起動テンプレートの異なるバージョンとして繰り返し更新できます。ノードグループを別のバージョンの起動テンプレートに更新すると、指定した起動テンプレートバージョンの新しい設定と一致するように、グループ内のすべてのノードがリサイクルされます。

マネージド型ノードグループは常に Amazon EC2 自動スケーリング グループで使用する起動テンプレートでデプロイされます。起動テンプレートを指定しない場合、Amazon EKS API はアカウントのデフォルト値を使用して起動テンプレートを自動的に作成します。ただし、自動生成された起動テンプレートを変更することは推奨しません。さらに、カスタム起動テンプレートを使用していない既存のノードグループは直接更新できません。代わりに、カスタム起動テンプレートを使用して、新しいノードグループを作成する必要があります。

起動テンプレート設定の基本

AWS マネジメントコンソール、AWS CLI、または AWS SDK を使用して、Amazon EC2 自動スケーリング 起動テンプレートを作成できます。詳細については「Amazon EC2 自動スケーリング ユーザーガイド」の「自動スケーリング グループの起動テンプレートを作成する」を参照してください。起動テンプレートの一部の設定はマネージド型ノードで使われている設定と似ています。起動テンプレートを使用してノードグループをデプロイまたは更新する場合、一部の設定はノードグループ設定または起動テンプレートのいずれかで指定する必要があります。両方の場所で設定を指定しないでください。存在してはいけない場所に設定が存在する場合、ノードグループの作成や更新などの操作は失敗します。

次の表に、起動テンプレートで禁止されている設定を示します。また、マネージド型ノードグループの設定に必要な同様の設定がある場合はその設定も示します。リスト化されている設定はコンソールに表示される設定です。AWS CLI と SDK では類似するが異なる名前がある場合があります。

起動テンプレート — 禁止	Amazon EKS ノードグループ設定
[ネットワークインターフェイス] ([ネットワークインターフェイスを追加]) の下の [サブネット]	[Specify networking] (ネットワーキングを指定) ページの [Node group network configuration] (ノードグループのネットワーク設定) の下の [Subnets] (サブネット)
[高度な詳細] の下の [IAM インスタンスプロファイル]	[Configure Node group] (ノードグループを設定) ページの [Node group configuration] (ノー

起動テンプレート — 禁止	Amazon EKS ノードグループ設定
	ドグループ設定) の下の [Node IAM role] (ノード IAM 役割)
[高度な詳細] の下の [シャットダウン動作] および [停止 – 休止動作]。起動テンプレートのどちらの設定でも、[起動テンプレート設定に含めない] のデフォルトを保持します。	同等のものはありません。Amazon EKS は自動スケーリング グループではなく、インスタンスのライフサイクルを管理する必要があります。

次の表に、マネージド型ノードグループ構成で禁止される設定を示します。また、起動テンプレートで必要な同様の設定がある場合はその設定も示します。リスト化されている設定はコンソールに表示される設定です。AWS CLI と SDK とで名前が似ている場合があります。

Amazon EKS ノードグループ設定 — 禁止	起動テンプレート
(起動テンプレートでカスタム AMI を指定した場合のみ) [コンピューティングとスケーリングの設定を実行] ページの [ノードグループのコンピューティング設定] の下の [AMI タイプ] — [起動テンプレートで指定] と、指定した AMI ID がコンソールに表示されます。	[Launch template contents] (起動テンプレートのコンテンツ) での [Application and OS Images (Amazon Machine Image)] (アプリケーションと OS のイメージ (Amazon マシンイメージ)) - 次のいずれかの要件がある場合はIDを指定する必要があります。
起動テンプレートで [アプリ and OS Images (Amazon Machine Image)] (アプリケーションと OS のイメージ (Amazon マシンイメージ)) が指定されていない場合はノードグループ設定で AMI を選択できます。	• カスタム AMI の使用。AMI を指定するに記載されている要件を満たさない AMI を指定すると、ノードグループのデプロイが失敗します。 • Amazon EKS に最適化された AMI に含まれる `bootstrap.sh` ファイルに引数を提供するためのユーザーデータを提供する必要がある。インスタンスでは、Pod に多数の IP アドレスを割り当てたり、インスタンスとは異なる CIDR ブロックから Pod に IP アドレスを割り当てたりできます。さらに、アウトバウンドのインターネットアクセスのないプライベートクラスターをデプロイすることも

Amazon EKS ノードグループ設定 — 禁止	起動テンプレート
	できます。詳細については、以下の各トピックを参照してください。 • プレフィックスを使用して Amazon EKS ノードに割り当てる IP アドレスを増やす • カスタムネットワーキングを使用して代替サブネットに Pod をデプロイする • インターネットアクセスが制限されたプライベートクラスターをデプロイする • AMI を指定する
[Set compute and scaling configuration] (コンピューティングとスケーリングの設定を実行) ページの [Node Group compute configuration] (ノードグループのコンピューティング設定) の下の [Disk size] (ディスクサイズ) — [Specified in launch template] (起動テンプレートで指定) がコンソールに表示されます。	[ストレージ (ボリューム)] ([新しいボリュームの追加]) の下の [サイズ]。これを起動テンプレートで指定する必要があります。
[Specify Networking] (ネットワーキングを指定) ページの [Node group configuration] (ノードグループ設定) の下の [SSH key pair] (SSH キーペア) — コンソールに起動テンプレートで指定したキーまたは [Not specified in launch template] (起動テンプレートで指定されていません) が表示されます。	[キーペア (ログイン)] の下の [キーペア名]。
起動テンプレートを使用する場合はリモートアクセスを許可するソースセキュリティグループを指定できません。	インスタンスの場合、[ネットワーク設定] の下の [セキュリティグループ]、または [ネットワークインターフェイス] ([ネットワークインターフェイスを追加])の下の [セキュリティグループ]。両方設定することはできません。詳細については「the section called “カスタムセキュリティグループを使用する”」を参照してください。

Note

- 起動テンプレートを使用してノードグループをデプロイする場合は起動テンプレート内の [起動テンプレートのコンテンツ] で 0 または 1 の [インスタンスタイプ] を指定します。またはコンソールの [コンピューティングとスケーリングの構成を設定する] ページにある [インスタンスタイプ] で 0 から 20 までのインスタンスタイプを指定できます。または Amazon EKS API を使用する他のツールを使用して行うこともできます。起動テンプレートでインスタンスタイプを指定し、その起動テンプレートを使用してノードグループをデプロイする場合、コンソールや、Amazon EKS API を使用する他のツールを使用してインスタンスタイプを指定することはできません。起動テンプレートやコンソール、または Amazon EKS API を使用する他のツールを使用してインスタンスタイプを指定しない場合は `t3.medium` インスタンスタイプが使用されます。ノードグループがスポットキャパシティータイプを使用している場合はコンソールを使用して複数のインスタンスタイプを指定することをお勧めします。詳細については「the section called “マネージド型ノードグループのキャパシティータイプ”」を参照してください。
- ノードグループにデプロイするコンテナが、インスタンスメタデータサービスバージョン 2 を使用している場合は起動テンプレートの [メタデータレスポンスのホップ制限] を 2 に設定してください。詳細については「Amazon EC2 ユーザーガイド」の「Instance metadata and user data」(インスタンスメタデータとユーザーデータ) を参照してください。
- 起動テンプレートでは、インスタンスタイプを柔軟に選択できるようにする `InstanceRequirements` 機能はサポートされていません。

Amazon EC2 インスタンスへのタグ付け

起動テンプレートの `TagSpecification` パラメータを使用して、ノードグループの Amazon EC2 インスタンスに適用するタグを指定します。`CreateNodegroup` または `UpdateNodegroupVersion` API を呼び出す IAM エンティティには`ec2:RunInstances` および `ec2:CreateTags` へのアクセス許可が必要です。また、起動テンプレートにタグが追加される必要があります。

カスタムセキュリティグループを使用する

起動テンプレートでカスタムの Amazon EC2 セキュリティグループを指定し、ノードグループ内のインスタンスに適用できます。これはインスタンスレベルのセキュリティグループのパラメータ内か、ネットワークインターフェイス設定のパラメータの一部として指定できます。しかし、インス

タンスレベルとネットワークインタフェイス、両方のセキュリティグループを指定して、起動テンプレートを作成することはできません。マネージドノード型グループでカスタムセキュリティグループを使用する際に適用される、次の条件を考慮してください。

- AWS マネジメントコンソール を使用する場合、Amazon EKS では単一のネットワークインターフェイス仕様の起動テンプレートのみ使用できます。
- デフォルトではAmazon EKS はクラスターセキュリティグループをノードグループ内のインスタンスに追加して、ノードとコントロールプレーンとの間の通信を容易にします。前述のいずれかのオプションを使用して、起動テンプレートでカスタムセキュリティグループを指定した場合、Amazon EKS はクラスターセキュリティグループを追加しません。したがって、セキュリティグループのインバウンドルールとアウトバウンドルールで、クラスターのエンドポイントとの通信が有効になっていることを確認する必要があります。セキュリティグループのルールが正しくない場合、ワーカーノードはクラスターに参加できません。セキュリティグループルールの詳細についてはthe section called “セキュリティグループの要件”を参照してください。
- ノードグループ内のインスタンスへの SSH アクセスが必要な場合はそのアクセスを許可するセキュリティグループを含めてください。

Amazon EC2 ユーザーデータ

起動テンプレートにはカスタムユーザーデータのセクションが含まれています。このセクションでは、個々のカスタム AMI を手動で作成しなくても、ノードグループの構成設定を指定できます。Bottlerocket で使用できる設定の詳細については、GitHub で Using user data を参照してください。

`cloud-init` を使用すると、インスタンス起動時に起動テンプレート内の Amazon EC2 ユーザーデータを提供できます。詳細については「cloud-init ドキュメント」を参照してください。ユーザーデータを使用すると、一般的な設定操作を実行できます。これには次の操作が含まれます。

- ユーザーまたはグループを含む
- パッケージのインストール

マネージド型ノードグループで使用される起動テンプレートの Amazon EC2 ユーザーデータは、Amazon Linux AMI の場合は、MIME マルチパートアーカイブ、Bottlerocket AMI の場合は TOML の形式であることが必要です。これはユーザーデータが、ノードがクラスターに参加するために必要な Amazon EKS ユーザーデータにマージされるためです。`kubelet` を起動または変更するコマンドをユーザーデータに指定しないでください。これは Amazon EKS によってマージされた

ユーザーデータの一部として実行されます。ノードへのラベル設定などの一部の kubelet パラメータはマネージド型ノードグループ API 経由で直接設定できます。

> **Note**
>
> 手動での起動や、カスタムの設定パラメータの受け渡しなど、高度な kubelet のカスタマイズについてはthe section called “AMI を指定する”を参照してください。起動テンプレートでカスタム AMI ID が指定されている場合、Amazon EKS はユーザーデータをマージしません。

次の詳細はユーザーデータセクションについて説明しています。

Amazon Linux 2 ユーザーデータ

複数のユーザーデータブロックと単一の MIME マルチパートファイルを組み合わせることができます。例えば、カスタムパッケージをインストールするユーザーデータのシェルスクリプトを、Docker デーモンを設定するクラウドブートフックに組み合わせることができます。MIME マルチパートファイルには次のコンポーネントが含まれます。

- コンテンツタイプとパートバウンダリの宣言: `Content-Type: multipart/mixed; boundary="==MYBOUNDARY=="`
- MIME バージョンの宣言: `MIME-Version: 1.0`
- 次のコンポーネントを含む 1 つ以上のユーザーデータブロック:
  - ユーザーデータブロックの始まりを示す開始境界:`--==MYBOUNDARY==`
  - ブロックのコンテンツの種類の宣言: `Content-Type: text/cloud-config; charset="us-ascii"`。コンテンツタイプの詳細については cloud-init のドキュメントを参照してください。
  - ユーザーデータのコンテンツ (例えば、シェルコマンドや cloud-init ディレクティブのリスト)。
  - MIME マルチパートファイルの終わりを示す、終了境界: `--==MYBOUNDARY==--`

自分で MIME マルチパートファイルを作成するときに使用できる例は次の通りです。

```
MIME-Version: 1.0
Content-Type: multipart/mixed; boundary="==MYBOUNDARY=="

--==MYBOUNDARY==
Content-Type: text/x-shellscript; charset="us-ascii"
```

```
#!/bin/bash
echo "Running custom user data script"

--==MYBOUNDARY==--
```

Amazon Linux 2023 ユーザーデータ

Amazon Linux 2023 (AL2023) ではYAML 設定スキーマを使用する新しいノード初期化プロセス nodeadm が導入されています。セルフマネージド型ノードグループまたは起動テンプレートを持つ AMI を使用している場合は新しいノードグループの作成時に追加のクラスターメタデータを明示的に指定する必要があります。最低限必要なパラメータの例を以下に示します。ここで、apiServerEndpoint、certificateAuthority、サービスの cidr が必要になります。

```
---
apiVersion: node.eks.aws/v1alpha1
kind: NodeConfig
spec:
  cluster:
    name: my-cluster
    apiServerEndpoint: https://example.com
    certificateAuthority: Y2VydGlmaWNhdGVBdXRob3JpdHk=
    cidr: 10.100.0.0/16
```

通常、この設定はユーザーデータにそのまま設定するか、MIME マルチパートドキュメントに埋め込まれます。

```
MIME-Version: 1.0
Content-Type: multipart/mixed; boundary="BOUNDARY"

--BOUNDARY
Content-Type: application/node.eks.aws

---
apiVersion: node.eks.aws/v1alpha1
kind: NodeConfig spec: [...]

--BOUNDARY--
```

AL2 ではこれらのパラメータからのメタデータは Amazon EKS DescribeCluster API コールから検出されていました。AL2023 ではノードの大規模なスケールアップ中に API コールによっ

てスロットリングが発生するリスクがあるため、この動作が変更されました。この変更は起動テンプレートのないマネージド型ノードグループを使用している場合や、Karpenter を使用している場合には影響しません。certificateAuthority とサービス cidr の詳細については、「Amazon EKS API リファレンス」の「DescribeCluster」を参照してください。

次に、AL2023 ユーザーデータの詳しい例を示します。ノードをカスタマイズするシェルスクリプト (パッケージのインストールやコンテナイメージの事前キャッシュなど) を必要な nodeadm 設定と結合しています。この例では、よく使用されるカスタマイズを示しています。* 追加でシステムパッケージをインストールする * コンテナイメージを事前キャッシュしてポッドの起動時間を改善する * HTTP プロキシ設定を設定する * ノードのラベル付けに関する kubelet フラグを設定する

```
MIME-Version: 1.0
Content-Type: multipart/mixed; boundary="BOUNDARY"

--BOUNDARY
Content-Type: text/x-shellscript; charset="us-ascii"

#!/bin/bash
set -o errexit
set -o pipefail
set -o nounset

# Install additional packages
yum install -y htop jq iptables-services

# Pre-cache commonly used container images
nohup docker pull public.ecr.aws/eks-distro/kubernetes/pause:3.2 &

# Configure HTTP proxy if needed
cat > /etc/profile.d/http-proxy.sh << 'EOF'
export HTTP_PROXY="http://proxy.example.com:3128"
export HTTPS_PROXY="http://proxy.example.com:3128"
export NO_PROXY="localhost,127.0.0.1,169.254.169.254,.internal"
EOF

--BOUNDARY
Content-Type: application/node.eks.aws

apiVersion: node.eks.aws/v1alpha1
kind: NodeConfig
spec:
```

```
  cluster:
    name: my-cluster
    apiServerEndpoint: https://example.com
    certificateAuthority: Y2VydGlmaWNhdGVBdXRob3JpdHk=
    cidr: 10.100.0.0/16
  kubelet:
    config:
      clusterDNS:
      - 10.100.0.10
    flags:
    - --node-labels=app=my-app,environment=production

--BOUNDARY--
```

Bottlerocket ユーザーデータ

Bottlerocket はユーザーデータを TOML 形式で構造化しています。Amazon EKS が提供するユーザーデータとマージするユーザーデータを提供できます。たとえば、追加の kubelet 設定を指定できます。

```
[settings.kubernetes.system-reserved]
cpu = "10m"
memory = "100Mi"
ephemeral-storage= "1Gi"
```

サポートされる設定について詳しくはBottlerocket のドキュメントを参照してください。ユーザーデータにノードラベルとテイントを設定できます。ただし、代わりにノードグループ内にこれらを設定することをお勧めします。この場合、Amazon EKS によりこれらの設定が適用されます。

ユーザーデータをマージしても、書式設定は保持されませんが、コンテンツは同じままです。ユーザーデータに指定した設定はAmazon EKS によって構成された設定よりも優先されます。したがって、`settings.kubernetes.max-pods` または `settings.kubernetes.cluster-dns-ip` を設定した場合、ユーザーデータのこれらの値がノードに適用されます。

Amazon EKS で、有効な TOML がすべてサポートされるわけではありません。以下はサポートされていない既知の形式の一覧です。

- 引用符で囲まれたキー内の引用符: `'quoted "value"' = "value"`
- 値内のエスケープされた引用符: `str = "I'm a string. \"You can quote me\""`
- 浮動小数点と整数の混在: `numbers = [ 0.1, 0.2, 0.5, 1, 2, 5 ]`

- 配列内の混合型: `contributors = ["foo@example.com", { name = "Baz", email = "baz@example.com" }]`
- 引用符付きキーを含む括弧で囲まれたヘッダー: `[foo."bar.baz"]`

Windows ユーザーデータ

Windows ユーザーデータでは、PowerShell コマンドを使用します。マネージド型ノードグループを作成すると、カスタムユーザーデータが Amazon EKS マネージドユーザーデータと結合されます。まず PowerShell コマンドが表示され、その後にマネージドユーザーデータコマンドが続きます。そのいずれも 1 つの `<powershell></powershell>` タグ内にあります。

> **Important**
>
> Windows ノードグループを作成すると、Amazon EKS は Linux ベースのノードがクラスターに参加できるように `aws-auth ConfigMap` を更新します。このサービスは、Windows AMI に対するアクセス許可を自動的には設定しません。Windows ノードを使用している場合は、アクセスエントリ API を利用するか、`aws-auth ConfigMap` を直接更新することで、アクセスを管理する必要があります。詳細については、「the section called “Windows サポートを有効にする”」を参照してください。

> **Note**
>
> 起動テンプレートに AMI ID が指定されていない場合はユーザーデータで Windows Amazon EKS ブートストラップスクリプトを使用して Amazon EKS を設定しないでください。

ユーザーデータの例は次のとおりです。

```
<powershell>
Write-Host "Running custom user data script"
</powershell>
```

AMI を指定する

以下のいずれかの要件がある場合は起動テンプレートの `ImageId` フィールドで AMI ID を指定します。追加情報については要件を選択してください。

Example Amazon EKS に最適化された Linux/Bottlerocket AMI に含まれる **bootstrap.sh** ファイルに引数を渡すためのユーザーデータを提供する

ブートストラップとはインスタンスの起動時に実行できるコマンドの追加を表す用語です。例えば、ブートストラップでは追加の kubelet 引数を使用できます。起動テンプレートを指定せずに、eksctl を使用して引数を bootstrap.sh スクリプトに渡すことができます。または起動テンプレートのユーザーデータセクションに情報を指定することでこれを行うことができます。

起動テンプレートを指定しない eksctl

my-nodegroup.yaml という名前のファイルを以下の内容で作成します。各*#####*は独自の値に置き換えます。--apiserver-endpoint、--b64-cluster-ca、および --dns-cluster-ip 引数はオプションです。しかし、これらを定義すると、bootstrap.sh スクリプトによる describeCluster 呼び出しを避けることができます。これはプライベートクラスターのセットアップや、ノードを頻繁にスケールインおよびスケールアウトするクラスターで役立ちます。bootstrap.sh スクリプトの詳細については、GitHub で「bootstrap.sh」ファイルを参照してください。

- 必須となる引数はクラスター名 (*#######*) のみです。
- ami-*1234567890abcdef0* に最適化された AMI ID を取得するには、次のセクションを参照してください。
  - 推奨 Amazon Linux AMI ID を取得する
  - 推奨 Bottlerocket AMI ID を取得する
  - 推奨 Microsoft Windows AMI ID を取得する
- クラスターの *####* を取得するには次のコマンドを実行してください。

```
aws eks describe-cluster --query "cluster.certificateAuthority.data" --output text
 --name my-cluster --region region-code
```

- クラスターの *api-server-endpoint* を取得するには次のコマンドを実行してください。

```
aws eks describe-cluster --query "cluster.endpoint" --output text --name my-
cluster --region region-code
```

- 最終的に、--dns-cluster-ip の値はサービス CIDR を示す .10 となります。クラスターの *service-cidr* を取得するには次のコマンドを実行してください。例えば、戻り値が ipv4 10.100.0.0/16 であれば、自分の値は *10.100.0.10* です。

```
aws eks describe-cluster --query "cluster.kubernetesNetworkConfig.serviceIpv4Cidr"
 --output text --name my-cluster --region region-code
```

- この例ではカスタムの max-pods 値を設定するために kubelet 引数を指定します。その際、Amazon EKS 最適化 AMI に含まれている bootstrap.sh スクリプトを使用します。ノードグループ名は 63 文字以下である必要があります。先頭は文字または数字でなければなりませんが、残りの文字にはハイフンおよびアンダースコアを含めることもできます。*my-max-pods-value* を選択する方法については「the section called “maxPods の決定方法”」を参照してください。マネージド型ノードグループを使用する場合の maxPods の決定方法の詳細については、「the section called “maxPods の決定方法”」を参照してください。

```
---
apiVersion: eksctl.io/v1alpha5
kind: ClusterConfig

metadata:
  name: my-cluster
  region: region-code

managedNodeGroups:
  - name: my-nodegroup
    ami: ami-1234567890abcdef0
    instanceType: m5.large
    privateNetworking: true
    disableIMDSv1: true
    labels: { x86-al2-specified-mng }
    overrideBootstrapCommand: |
      #!/bin/bash
      /etc/eks/bootstrap.sh my-cluster \
        --b64-cluster-ca certificate-authority \
        --apiserver-endpoint api-server-endpoint \
        --dns-cluster-ip service-cidr.10 \
        --kubelet-extra-args '--max-pods=my-max-pods-value' \
        --use-max-pods false
```

利用可能な eksctl config ファイルオプションについては「eksctl ドキュメント」の「Config file schema」(Config ファイルスキーマ) を参照してください。eksctl ユーティリティは引き続き起動テンプレートを作成し、config ファイルに提供したデータを使用してユーザーデータを設定します。

次のコマンドを使用して、ノードグループを作成します。

```
eksctl create nodegroup --config-file=my-nodegroup.yaml
```

起動テンプレートのユーザーデータ

起動テンプレートのユーザーデータセクションで次の情報を指定します。各*#####*は独自の値に置き換えます。--apiserver-endpoint、--b64-cluster-ca、および --dns-cluster-ip 引数はオプションです。しかし、これらを定義すると、bootstrap.sh スクリプトによる describeCluster 呼び出しを避けることができます。これはプライベートクラスターのセットアップや、ノードを頻繁にスケールインおよびスケールアウトするクラスターで役立ちます。bootstrap.sh スクリプトの詳細については、GitHub で「bootstrap.sh」ファイルを参照してください。

- 必須となる引数はクラスター名 (*#######*) のみです。
- クラスターの *####* を取得するには次のコマンドを実行してください。

```
aws eks describe-cluster --query "cluster.certificateAuthority.data" --output text
 --name my-cluster --region region-code
```

- クラスターの *api-server-endpoint* を取得するには次のコマンドを実行してください。

```
aws eks describe-cluster --query "cluster.endpoint" --output text --name my-
cluster --region region-code
```

- 最終的に、--dns-cluster-ip の値はサービス CIDR を示す .10 となります。クラスターの *service-cidr* を取得するには次のコマンドを実行してください。例えば、戻り値が ipv4 10.100.0.0/16 であれば、自分の値は *10.100.0.10* です。

```
aws eks describe-cluster --query "cluster.kubernetesNetworkConfig.serviceIpv4Cidr"
 --output text --name my-cluster --region region-code
```

- この例ではカスタムの max-pods 値を設定するために kubelet 引数を指定します。その際、Amazon EKS 最適化 AMI に含まれている bootstrap.sh スクリプトを使用します。*my-max-pods-value* を選択する方法については「the section called “maxPods の決定方法”」を参照してください。マネージド型ノードグループを使用する場合の maxPods の決定方法の詳細については、「the section called “maxPods の決定方法”」を参照してください。

```
MIME-Version: 1.0
Content-Type: multipart/mixed; boundary="==MYBOUNDARY=="
```

```
--==MYBOUNDARY==
Content-Type: text/x-shellscript; charset="us-ascii"

#!/bin/bash
set -ex
/etc/eks/bootstrap.sh my-cluster \
  --b64-cluster-ca certificate-authority \
  --apiserver-endpoint api-server-endpoint \
  --dns-cluster-ip service-cidr.10 \
  --kubelet-extra-args '--max-pods=my-max-pods-value' \
  --use-max-pods false

--==MYBOUNDARY==--
```

Example Amazon EKS に最適化された Windows AMI に含まれる **Start-EKSBootstrap.ps1** ファイルに引数を渡すためのユーザーデータを提供する

ブートストラップとはインスタンスの起動時に実行できるコマンドの追加を表す用語です。起動テンプレートを指定せずに、eksctl を使用して引数を Start-EKSBootstrap.ps1 スクリプトに渡すことができます。または起動テンプレートのユーザーデータセクションに情報を指定することでこれを行うことができます。

カスタム Windows AMI ID を指定する場合は、次の考慮事項に留意してください。

- 起動テンプレートを使用し、必要なブートストラップコマンドをユーザーデータセクションに入力する必要があります。目的の Windows ID を取得するには、「最適化された Windows AMI を使用してノードを作成する」のテーブルを使用できます。
- いくつかの制限と条件があります。例えば、eks:kube-proxy-windows を AWS IAM 認証 設定マップに追加する必要があります。詳細については「the section called “AMI ID を指定する場合の制限と条件”」を参照してください。

起動テンプレートのユーザーデータセクションで次の情報を指定します。各#####は独自の値に置き換えます。-APIServerEndpoint、-Base64ClusterCA、および -DNSClusterIP 引数はオプションです。しかし、これらを定義すると、Start-EKSBootstrap.ps1 スクリプトによる describeCluster 呼び出しを避けることができます。

- 必須となる引数はクラスター名 (#######) のみです。
- クラスターの #### を取得するには次のコマンドを実行してください。

```
aws eks describe-cluster --query "cluster.certificateAuthority.data" --output text --
name my-cluster --region region-code
```

- クラスターの *api-server-endpoint* を取得するには次のコマンドを実行してください。

```
aws eks describe-cluster --query "cluster.endpoint" --output text --name my-cluster
 --region region-code
```

- 最終的に、--dns-cluster-ip の値はサービス CIDR を示す .10 となります。クラスターの *service-cidr* を取得するには次のコマンドを実行してください。例えば、戻り値が ipv4 10.100.0.0/16 であれば、自分の値は *10.100.0.10* です。

```
aws eks describe-cluster --query "cluster.kubernetesNetworkConfig.serviceIpv4Cidr" --
output text --name my-cluster --region region-code
```

- その他の引数については「the section called “ブートストラップスクリプトの設定パラメータ”」を参照してください。

> **Note**
>
> カスタムサービス CIDR を使用している場合は -ServiceCIDR パラメータを使用して指定する必要があります。そうしないと、クラスター内の Pod の DNS 解決が失敗します。

```
<powershell>
[string]$EKSBootstrapScriptFile = "$env:ProgramFiles\Amazon\EKS\Start-EKSBootstrap.ps1"
& $EKSBootstrapScriptFile -EKSClusterName my-cluster `
  -Base64ClusterCA certificate-authority `
  -APIServerEndpoint api-server-endpoint `
  -DNSClusterIP service-cidr.10
</powershell>
```

Example特定のセキュリティ、コンプライアンス、または社内的なポリシー要件のために、カスタム AMI を実行する

詳細については「Amazon EC2 ユーザーガイド」の「Amazon マシンイメージ (AMI)」を参照してください。Amazon EKS AMI ビルド仕様にはAmazon Linux をベースにしたカスタム Amazon EKS AMI を構築するためのリソースと設定スクリプトが含まれています。詳細については、GitHubの「Amazon EKS AMI ビルド仕様」を参照してください。他のオペレーティングシステムがインス

トールされたカスタム AMI を作成するには、GitHubの「Amazon EKS Sample Custom AMIs」を参照してください。

マネージドノードグループで使用される起動テンプレートで AMI ID に動的パラメータ参照を使用することはできません。

> **Important**
>
> - AMI を指定する場合、Amazon EKS はクラスターのコントロールプレーンバージョンに対して AMI に埋め込まれた Kubernetes バージョンを検証しません。ユーザーは、カスタム AMI の Kubernetes バージョンが以下の Kubernetes バージョンスキューポリシーに準拠していることを確認する責任があります。
>   - ノード上の `kubelet` バージョンはクラスターバージョンより新しいものであってはならない
>   - ノード上の `kubelet` バージョンは、お使いのクラスターバージョン (Kubernetes バージョン `1.28` およびそれ以降) から最大 マイナーバージョン 3 つ前まで、またはクラスターバージョン (Kubernetes バージョン `1.27` およびそれ以前) から最大 マイナーバージョン 2 つ前までである必要がある
>
>   バージョンスキューポリシーに違反したマネージド型ノードグループを作成すると、次の結果になる可能性があります。
>   - ノードをクラスターに結合できない
>   - 未定義の動作または API の非互換性
>   - クラスターの不安定性またはワークロード障害
> - AMI を指定する際、Amazon EKS ではユーザーデータをマージしません。代わりに、ノードのクラスターへの参加に必要な `bootstrap` コマンドを、ユーザーが提供する必要があります。ノードがクラスターに参加できない場合、Amazon EKS `CreateNodegroup` および `UpdateNodegroupVersion` アクションも失敗します。

AMI ID を指定する場合の制限と条件

以下に、マネージド型ノードグループで AMI ID を指定する際の制限と条件を示します。

- 起動テンプレートで AMI ID を指定する場合と、AMI ID を指定しない場合は新しいノードグループを作成して切り替える必要があります。

- 新しい AMI バージョンが利用可能になっても、コンソールでは通知を表示しません。ノードグループを新しい AMI バージョンに更新するには更新された AMI ID で新しいバージョンの起動テンプレートを作成する必要があります。次に、新しい起動テンプレートバージョンでノードグループを更新する必要があります。
- AMI ID を指定した場合はAPI の次のフィールドを設定することはできません。
  - `amiType`
  - `releaseVersion`
  - `version`
- AMI ID を指定する場合、API で設定されたすべての `taints` は非同期に適用されます。ノードがクラスターに参加する前にテイントを適用するには`--register-with-taints` コマンドラインフラグを使用してユーザーデータ内の `kubelet` にテイントを渡す必要があります。詳細については、Kubernetes ドキュメントの「kubelet」を参照してください。
- Windows マネージド型ノードグループのカスタム AMI ID を指定するときは、AWS IAM オーセンティケーター設定マップに `eks:kube-proxy-windows` を追加します。これはDNS が正しく機能するために必要です。
  a. AWS IAM 認証 設定マップを編集用に開きます。

  ```
  kubectl edit -n kube-system cm aws-auth
  ```

  b. このエントリを、Windows ノードに関連付けられた各 `rolearn` の下にある `groups` リストに追加します。設定マップは aws-auth-cm-windows.yamlに似ているはずです。

  ```
  - eks:kube-proxy-windows
  ```

  c. ファイルを保存し、テキストエディタを終了します。
- カスタム起動テンプレートを使用する AMI の場合、マネージドノードグループのデフォルト `HttpPutResponseHopLimit` は 2 に設定されます。

クラスターからマネージドノードグループを削除する

このトピックでは、Amazon EKS マネージド型ノードグループを削除する方法について説明します。マネージド型ノードグループを削除すると、まず Amazon EKS が Auto Scaling グループの最小サイズ、最大サイズ、および必要なサイズをゼロに設定します。これにより、ノードグループがスケールダウンされます。

各インスタンスが終了する前に、Amazon EKS はそのノードを排出するシグナルを送信します。排出プロセス中、Kubernetes はノード上の各ポッドに対して、設定されている preStop ライフサイクルフックを実行し、コンテナに SIGTERM シグナルを送信してから、正常なシャットダウンのために terminationGracePeriodSeconds の間待機します。5 分後もノードが排出されない場合、Amazon EKS は自動スケーリングにインスタンスの強制終了を続行させます。すべてのインスタンスが終了すると、Auto Scaling グループは削除されます。

> **Important**
>
> クラスター内の、他のマネージド型ノードグループで使用されていないノード IAM ロールを使用している、マネージド型ノードグループを削除すると、そのロールは aws-auth ConfigMap から削除されます。クラスター内のいずれかのセルフマネージド型ノードグループが同じノード IAM ロールを使用している場合、セルフマネージド型ノードは NotReady ステータスに移行します。さらに、クラスター操作も中断されます。クラスターのプラットフォームバージョンが「EKS アクセスエントリを使用して Kubernetes へのアクセスを IAM ユーザーに許可する」の前提条件セクションに記載されている最低バージョン以上である場合、セルフマネージドノードグループにのみ使用しているロールのマッピングを追加するには、「the section called “アクセスエントリを作成する”」を参照してください。プラットフォームバージョンがアクセスエントリに必要な最小バージョンより前の場合は、エントリを aws-auth ConfigMap に追加し直すことができます。詳細については、ターミナルに eksctl create iamidentitymapping --help を入力してください。

マネージド型ノードグループを削除するには、以下を使用します。

- the section called “eksctl”
- the section called “AWS マネジメントコンソール”
- the section called “AWS CLI”

eksctl

マネージド型ノードグループを **eksctl** を使用して削除する

次のコマンドを入力します。<example value> をすべて自分の値に置き換えてください。

```
eksctl delete nodegroup \
  --cluster <my-cluster> \
  --name <my-mng> \
```

```
  --region <region-code>
```

その他のオプションについては、eksctl ドキュメントの「ノードグループの削除と排出」を参照してください。

AWS マネジメントコンソール

マネージド型ノードグループを AWS マネジメントコンソール を使用して削除する

1. Amazon EKS コンソールを開きます。
2. [クラスター] ページで、削除するノードグループを含むクラスターを選択します。
3. 選択したクラスターページで、[コンピューティング] タブを選択します。
4. [Node groups] (ノードグループ) セクションで、削除するノードグループを選択してください。その後、[削除] をクリックします。
5. [ノードグループの削除] 確認ダイアログボックスで、ノードグループの名前を入力します。その後、[削除] をクリックします。

AWS CLI

マネージド型ノードグループを AWS CLI を使用して削除する

1. 次のコマンドを入力します。<example value> をすべて自分の値に置き換えてください。

```
aws eks delete-nodegroup \
  --cluster-name <my-cluster> \
  --nodegroup-name <my-mng> \
  --region <region-code>
```

2. CLI 設定で cli_pager= が設定されている場合は、キーボードの矢印キーを使用して、応答出力をスクロールします。終了したら q キーを押します。

   その他のオプションについては、「AWS CLI コマンドリファレンス」の delete-nodegroup コマンドを参照してください。

セルフマネージドノードでノードを自分自身で維持する

クラスターには、Pod がスケジュールされている Amazon EC2 ノードが 1 つ以上含まれています。Amazon EKS ノードは AWS アカウントで実行され、クラスター API サーバーエンドポイント

を介してクラスターのコントロールプレーンに接続します。Amazon EC2 料金に基づいて請求されます。詳細については「Amazon EC2 料金」を参照してください。

クラスターには、複数のノードグループを含めることができます。各ノードグループには、Amazon EC2 Auto Scaling グループにデプロイされる 1 つ以上のノードが含まれます。グループ内のノードのインスタンスタイプは、Karpenter による属性ベースのインスタンスタイプの選択を使用するときなどに、異なる場合があります。ノードグループ内のすべてのインスタンスは、Amazon EKS ノードの IAM ロールを使用する必要があります。

Amazon EKS は、Amazon EKS 最適化 AMI と呼ばれる特殊な Amazon マシンイメージ (AMI) を提供します。AMI は Amazon EKS と連携するように設定されています。そのコンポーネントには `containerd`、`kubelet`、および AWS IAM Authenticator が含まれます。また、AMI にはクラスターのコントロールプレーンを自動的に検出して接続を許可する、特別なブートストラップスクリプトも含まれます。

CIDR ブロックを使用してクラスターのパブリックエンドポイントへのアクセスを制限する場合は、プライベートエンドポイントアクセスも有効にすることをお勧めします。これは、ノードがクラスターと通信できるようにするためです。プライベートエンドポイントが有効になっていない場合、パブリックアクセスに指定する CIDR ブロックに、VPC からの出力ソースを含める必要があります。詳細については、「the section called “クラスターエンドポイントのアクセス”」を参照してください。

Amazon EKS クラスターにセルフマネージド型ノードを追加するには、次のトピックを参照してください。セルフマネージドノードを手動で起動する場合は、各ノードに次のタグを追加すると共に、`<cluster-name>` がクラスターに一致することを確認します。詳細については、「個々のリソースでのタグの追加と削除」を参照してください。このガイドの手順に従うと、必要なタグがノードに自動的に追加されます。

キー	値
`kubernetes.io/cluster/<cluster-name>`	`owned`

Important

Amazon EC2 インスタンスメタデータサービス (IMDS) のタグは、EKS ノードと互換性がありません。インスタンスメタデータタグを有効にすると、タグ値でのスラッシュ (「/」) の使用が防止されます。この制限により、特に Karpenter や Cluster Autoscaler などのノード管

理ツールを使用する場合、インスタンスの起動が失敗する可能性があります。これらのサービスは、スラッシュを含むタグに依存して適切に機能しているためです。

Kubernetes の一般的な視点から見たノードの詳細については、Kubernetes のドキュメントの「Nodes」を参照してください。

トピック

- セルフマネージド Amazon Linux ノードを作成する
- セルフマネージド Bottlerocket ノードの作成
- セルフマネージド Microsoft Windows ノードの作成
- セルフマネージドの Ubuntu Linux ノードを作成する
- クラスターのためにセルフマネージドノードを更新する

セルフマネージド Amazon Linux ノードを作成する

このトピックでは、Amazon EKS クラスターに登録されている Linux ノードのAuto Scaling グループを起動する方法を説明します。ノードがクラスターに参加したら、それらのノードに Kubernetes アプリケーションをデプロイ可能になります。`eksctl` または AWS マネジメントコンソールを使用して、セルフマネージド型の Amazon Linux ノードを起動することもできます。AWS Outposts でノードを起動する必要がある場合は「the section called “ノード”」を参照してください。

- 既存の Amazon EKS クラスター。デプロイするには、the section called “クラスターを作成する” を参照してください。AWS Outposts、AWS 波長、または AWS ローカルゾーンが有効になっている AWS リージョンにサブネットがある場合はクラスターの作成時にそれらのサブネットが渡されるべきではありません。
- ノードが使用する既存の IAM ロール。作成する場合は、the section called “ノード の IAM ロール” を参照してください。この役割に VPC CNI のポリシーがどちらも含まれていない場合、VPC CNI ポッドには以下の別の役割が必要です。
- (オプションですが推奨) 必要な IAM ポリシーがアタッチされた独自の IAM ロールで構成された Amazon VPC CNI plugin for Kubernetes アドオン。詳細については、the section called “IRSA 用の設定” を参照してください。
- 「最適な Amazon EC2 ノードインスタンスタイプを選択する」に記載されている考慮事項に精通していること。選択したインスタンスタイプによってはクラスターと VPC に関する追加の前提条件がある場合もあります。

セルフマネージド Linux ノードは次のいずれかを使用して起動できます。

- the section called “eksctl”
- the section called “AWS マネジメントコンソール”

eksctl

eksctl を使用してセルフマネージド型の Linux ノードを起動する

1. デバイスまたは AWS クラウドシェル にインストールされている eksctl コマンドラインツールのバージョン 0.215.0 以降をインストールします。eksctl をインストールまたはアップグレードするにはeksctl ドキュメントの「インストール」を参照してください。
2. (オプション) [AmazonEKS_CNI_Policy] マネージド IAM ポリシーが Amazon EKS ノードの IAM ロールにアタッチされている場合は、代わりに Kubernetes aws-node サービスアカウントに関連付けた IAM ロールに割り当てることをお勧めします。詳細については、the section called “IRSA 用の設定” を参照してください。
3. 次のコマンドは既存のクラスターにノードグループを作成します。*al-nodes* を、ノードグループの名前に置き換えます。ノードグループ名は 63 文字以下である必要があります。先頭は文字または数字でなければなりませんが、残りの文字にはハイフンおよびアンダースコアを含めることもできます。*#######* の部分は自分のクラスター名に置き換えます。この名前には英数字 (大文字と小文字が区別されます) とハイフンのみを使用できます。先頭の文字は英数字である必要があります。また、100 文字より長くすることはできません。名前はクラスターを作成する AWS リージョンおよび AWS アカウント内で一意である必要があります。残りの*#####*は独自の値に置き換えます。デフォルトでは、ノードはコントロールプレーンと同じ Kubernetes バージョンで作成されます。

   --node-type の値を選択する前に、「最適な Amazon EC2 ノードインスタンスタイプを選択する」を確認してください。

   my-key を Amazon EC2 キーペアまたはパブリックキーの名前に置き換えます。このキーは起動後のノードに SSH 接続するために使用されます。Amazon EC2 キーペアをまだ持っていない場合はAWS マネジメントコンソール で作成できます。詳細については「Amazon EC2 ユーザーガイド」の「Amazon EC2 キーペア」を参照してください。

   次のコマンドを使用して、ノードグループを作成します。

> **Important**
>
> ノードグループを AWS Outposts、波長、またはローカルゾーンサブネットにデプロイする場合、追加の考慮事項があります。
>
> - クラスターを作成したときに、サブネットが渡されていない必要があります。
> - ノードグループはサブネットと `volumeType: gp2` を指定する設定ファイルを使用して作成する必要があります。詳細については「eksctl ドキュメント」の「設定ファイルからノードグループを作成する」と「設定ファイルのスキーマ」を参照してください。

```
eksctl create nodegroup \
  --cluster my-cluster \
  --name al-nodes \
  --node-type t3.medium \
  --nodes 3 \
  --nodes-min 1 \
  --nodes-max 4 \
  --ssh-access \
  --managed=false \
  --ssh-public-key my-key
```

次のノードグループをデプロイするには

- デフォルト設定よりもはるかに多くの IP アドレスを Pod に割り当てることができるノードグループについては、「the section called “IP アドレスを増やす”」を参照してください。
- インスタンスのブロックとは異なる CIDR ブロックから IPv4 アドレスを Pod に割り当てることができるノードグループについては、「the section called “カスタムネットワーキング”」を参照してください。
- Pod とサービスに IPv6 アドレスを割り当てることができるノードグループについては、「the section called “IPv6”」を参照してください。
- アウトバウンドインターネットアクセスがない場合は、the section called “プライベートクラスター” を参照してください。

すべての使用可能なオプションとデフォルト値の一覧を表示するには次のコマンドを入力してください。

```
eksctl create nodegroup --help
```

ノードがクラスターに参加できない場合はトラブルシューティングの章にある「the section called “ノードがクラスターへの参加に失敗します”」を参照してください。

出力例は次のとおりです。ノードの作成中に、複数の行が出力されます。出力の最後の行は次のサンプル行が表示されます。

```
[#]  created 1 nodegroup(s) in cluster "my-cluster"
```

4. (オプション) サンプルアプリケーションをデプロイして、クラスターと Linux ノードをテストします。
5. 次の条件が true の場合、IMDS への Pod アクセスをブロックすることをお勧めします。
   - IAM ロールをすべての Kubernetes サービスアカウントに割り当てることによって、必要最小限のアクセス許可のみを Pod に付与しようとしている。
   - クラスター内の Pod が、現在の AWS リージョンの取得など、その他の理由で Amazon EC2 インスタンスメタデータサービス (IMDS) へのアクセスを必要としていない。

   詳細については「ワーカーノードに割り当てられたインスタンスプロファイルへのアクセスを制限する」を参照してください。

AWS マネジメントコンソール

ステップ 1: AWS マネジメントコンソール を使用してセルフマネージド型の Linux ノードを起動する`

1. AWS クラウドフォーメーション テンプレートの最新バージョンをダウンロードします。

```
curl -O https://s3.us-west-2.amazonaws.com/amazon-eks/cloudformation/2025-11-26/
amazon-eks-nodegroup.yaml
```

2. クラスターステータスが ACTIVE と表示されるまで待ちます。クラスターがアクティブになる前にノードを起動した場合、ノードはクラスターへの登録に失敗し、再起動する必要があります。
3. AWS クラウドフォーメーション コンソールを開きます。
4. [スタックの作成] を選択し、[新しいリソースを使用 (標準)] を選択してください。

5. [テンプレートの指定] で、[テンプレートファイルのアップロード] を選択し、[ファイルを選択] を選択してください。
6. ダウンロードした amazon-eks-nodegroup.yaml ファイルを選択してください。
7. [次へ] を選択してください。
8. [スタックの詳細の指定] ページで、必要に応じて次のパラメータを入力し、[次へ] を選択してください。
   - スタック名: AWS クラウドフォーメーション スタックのスタック名を選択してください。例えば、*#######-nodes* という名前にすることができます。この名前には英数字 (大文字と小文字が区別されます) とハイフンのみを使用できます。先頭の文字は英数字である必要があります。また、100 文字より長くすることはできません。名前はクラスターを作成する AWS リージョンおよび AWS アカウント内で一意である必要があります。
   - [クラスター名]: Amazon EKS クラスターの作成時に使用した名前を入力してください。この名前はクラスター名と同じにする必要があります。同じでない場合、ノードはクラスターに参加できません。
   - [クラスター制御プレーンセキュリティグループ]: VPC の作成時に生成した AWS クラウドフォーメーション 出力の [セキュリティグループs] 値を選択してください。

     次のステップでは該当するグループを取得する 1 つのオペレーションを説明します。

     a. Amazon EKS コンソール を開きます。

     b. クラスターの名前を選択してください。

     c. [ネットワーキング] タブを選択してください。

     d. [クラスター制御プレーンセキュリティグループ] ドロップダウンリストから選択する場合は[追加のセキュリティグループ] の値をリファレンスとして使用します。
   - [ApiServerEndpoint]: EKS クラスターの API サーバーエンドポイントを入力します。これは EKS クラスターコンソールの [詳細] セクションにあります。
   - [CertificateAuthorityData]: base64 でエンコードされた認証機関データを入力します。これは、EKS クラスターコンソールの [詳細] セクションにもあります。
   - [ServiceCidr]: クラスター内の Kubernetes サービスに IP アドレスを割り当てるために使用される CIDR 範囲を入力します。これは、EKS クラスターコンソールの [ネットワーク] タブにあります。
   - [AuthenticationMode]: EKS クラスターコンソール内の [アクセス] タブを確認して、EKS クラスターで使用されている認証モードを選択します。
   - [ノードグループ名]: ノードグループの名前を入力してください。この名前はノードに対して作成される自動スケーリングノードグループを識別するために後で使用できます。ノードグルー

プ名は 63 文字以下である必要があります。先頭は文字または数字でなければなりませんが、残りの文字にはハイフンおよびアンダースコアを含めることもできます。

- [ノードオートスケーリンググループ最小サイズ]: ノードのAuto Scaling グループがスケールインできる最小ノード数を入力してください。
- ノード自動スケーリンググループ希望容量: スタック作成時にスケーリングする必要のあるノード数を入力してください。
- [NodeAutoScalingGroupMaxSize]: ノードのAuto Scaling グループがスケールアウトできる最大ノード数を入力してください。
- [ノードインスタンス型]: ノードのインスタンスタイプを選択してください。詳細については、the section called “Amazon EC2 インスタンスタイプ” を参照してください。
- [NodeImageIdSSMParam]: 最新の Amazon EKS 最適化 Amazon Linux 2023 AMI の Amazon EC2 Systems Manager のパラメータが、可変 Kubernetes バージョン用に事前設定されています。Amazon EKS でサポートされている別の Kubernetes マイナーバージョンを使用するには、*1.XX* を別のサポートされているバージョンに置き換えます。クラスターと同じ Kubernetes バージョンを指定することをお勧めします。

  AmazonLinux-2023 を別の AMI タイプに置き換えることもできます。詳細については、the section called “最新の ID を取得する” を参照してください。

  > **Note**
  >
  > Amazon EKS ノード AMI は Amazon Linux をベースとしています。Amazon Linux セキュリティセンターで Amazon Linux 2023 のセキュリティもしくはプライバシーに関するイベントを追跡したり、関連する RSS フィードをサブスクライブできます。セキュリティおよびプライバシーイベントには問題の概要、影響を受けるパッケージ、および問題を修正するためにインスタンスを更新する方法などがあります。

- ノードイメージId: (オプション) (Amazon EKS 最適化 AMI の代わりに) 独自のカスタム AMI を使用している場合はAWS リージョンのノード AMI ID を入力してください。ここで値を指定すると、[ノードイメージIdSSMParam] フィールドの値はすべて上書きされます。
- [ノードボリュームサイズ]: ノードのルートボリュームのサイズを GiB 単位で指定します。
- [ノードボリューム型]: ノードのルートボリュームタイプを指定します。
- [キー名]: 起動後に、SSH を使用してノードに接続するときに使用できる Amazon EC2 SSH キーペアの名前を入力してください。Amazon EC2 キーペアをまだ持っていない場合はAWS マネジメントコンソール で作成できます。詳細については「Amazon EC2 ユーザーガイド」の「Amazon EC2 キーペア」を参照してください。

- [VpcId]: 作成した VPC の ID を入力してください。
- [Subnets]: VPC 用に作成したサブネットを選択してください。「Amazon EKS クラスターの Amazon VPC を作成する」で説明されている手順で VPC を作成した場合は起動するノードの VPC 内のプライベートサブネットのみを指定します。クラスターの [ネットワーキング] タブから、各サブネットリンクを開き、プライベートのサブネットを確認できます。

> **Important**
>
> - いずれかのサブネットがパブリックサブネットである場合はパブリック IP アドレスの自動割り当て設定を有効にする必要があります。この設定がパブリックサブネットに対して有効になっていない場合、そのパブリックサブネットにデプロイするノードにはパブリック IP アドレスが割り当てられず、クラスターやその他の AWS のサービスと通信できなくなります。2020 年 3 月 26 日以前に、Amazon EKS AWS クラウドフォーメーション VPC テンプレートを使用して、または `eksctl` を使用してサブネットがデプロイされた場合、パブリックサブネットではパブリック IP アドレスの自動割り当てが無効になります。サブネットのパブリック IP アドレス割り当てを有効にする方法については「サブネットのパブリック IPv4 アドレス属性の変更」を参照してください。ノードがプライベートサブネットにデプロイされている場合、NAT ゲートウェイを介してクラスターや他の AWS のサービスと通信できます。
> - サブネットにインターネットアクセスがない場合は「インターネットアクセスが制限されたプライベートクラスターをデプロイする」の考慮事項と追加の手順を確認してください。
> - AWS Outposts、波長、または ローカルゾーンサブネット を選択する場合はクラスターの作成時にサブネットを渡さないようにします。

9. [スタックオプションの設定] ページで、希望する設定を選択し、[次へ] を選択してください。
10. [AWS クラウドフォーメーション が IAM リソースを作成する可能性を認識しています] の左にあるチェックボックスを選択して、[スタックの作成] を選択してください。
11. スタックの作成が完了したら、コンソールで選択し、[出力] を選択してください。`EKS API` または `EKS API and ConfigMap` 認証モードを使用している場合は、これが最後のステップです。
12. `ConfigMap` 認証モードを使用している場合は、作成されたノードグループの NodeInstanceRole を記録します。

ステップ 2: ノードを有効にしてクラスターに参加する

> **Note**
>
> 次の 2 つのステップは、EKS クラスター内で Configmap 認証モードを使用している場合にのみ必要です。さらに、アウトバウンドのインターネットアクセスのないプライベート VPC でノードを起動する場合はノードが VPC 内からクラスターに参加できるようにしてください。

1. aws-auth ConfigMap がすでにあるかどうかを確認します。

```
kubectl describe configmap -n kube-system aws-auth
```

2. aws-auth ConfigMap が表示されている場合は必要に応じて更新してください。

   a. 編集する ConfigMap を開きます。

```
kubectl edit -n kube-system configmap/aws-auth
```

   b. 必要に応じて新しい mapRoles エントリを追加します。rolearn 値を、前の手順で記録した [ノードインスタンス役割] 値に設定します。

```
[...]
data:
  mapRoles: |
    - rolearn: <ARN of instance role (not instance profile)>
      username: system:node:{{EC2PrivateDNSName}}
      groups:
        - system:bootstrappers
        - system:nodes
[...]
```

   c. ファイルを保存し、テキストエディタを終了します。

3. 「Error from server (NotFound): configmaps "aws-auth" not found」というエラーが表示されたら、スタック ConfigMap を適用してください。

   a. 設定マップをダウンロードします。

```
curl -O https://s3.us-west-2.amazonaws.com/amazon-eks/cloudformation/2020-10-29/
aws-auth-cm.yaml
```

b. aws-auth-cm.yaml ファイルで、rolearn 値を前の手順で記録した [ノードインスタンス役割] 値に設定します。これを行うにはテキストエディタを使用するか、*############* を置き換えて次のコマンドを実行してください。

```
sed -i.bak -e 's|<ARN of instance role (not instance profile)>|my-node-instance-role|' aws-auth-cm.yaml
```

c. 設定を適用します。このコマンドが完了するまで数分かかることがあります。

```
kubectl apply -f aws-auth-cm.yaml
```

4. ノードのステータスを監視し、Ready ステータスになるまで待機します。

```
kubectl get nodes --watch
```

Ctrl+C を入力して、シェルプロンプトに戻ります。

> **Note**
>
> 認可またはリソースタイプのエラーが発生した場合はトラブルシューティングトピックの「the section called “許可されていないか、アクセスが拒否されました (kubectl)”」を参照してください。

ノードがクラスターに参加できない場合はトラブルシューティングの章にある「the section called “ノードがクラスターへの参加に失敗します”」を参照してください。

5. (GPU ノードのみ) GPU インスタンスタイプと Amazon EKS 最適化アクセラレーション AMI を選択した場合はクラスター上の DaemonSet として Kubernetes 用の NVIDIA デバイスプラグインを適用する必要があります。次のコマンドを実行する前に、*vX.X.X* を必要となる NVIDIA/k8s-device-plugin バージョンに置き換えます。

```
kubectl apply -f https://raw.githubusercontent.com/NVIDIA/k8s-device-plugin/vX.X.X/deployments/static/nvidia-device-plugin.yml
```

ステップ 3: その他のアクション

1. (オプション) サンプルアプリケーションをデプロイして、クラスターと Linux ノードをテストします。
2. (オプション) AmazonEKS_CNI_Policy マネージド IAM ポリシー (IPv4 クラスターがある場合) または *AmazonEKS_CNI_IPv6_Policy* (IPv6 クラスターがある場合にはユーザー自身が作成したもの) が Amazon EKS ノードの IAM ロールにアタッチされている場合、代わりに Kubernetes aws-node サービスアカウントに関連付けた IAM ロールに割り当てることをお勧めします。詳細については、the section called “IRSA 用の設定” を参照してください。
3. 次の条件が true の場合、IMDS への Pod アクセスをブロックすることをお勧めします。
   - IAM ロールをすべての Kubernetes サービスアカウントに割り当てることによって、必要最小限のアクセス許可のみを Pod に付与しようとしている。
   - クラスター内の Pod が、現在の AWS リージョンの取得など、その他の理由で Amazon EC2 インスタンスメタデータサービス (IMDS) へのアクセスを必要としていない。

   詳細については「ワーカーノードに割り当てられたインスタンスプロファイルへのアクセスを制限する」を参照してください。

セルフマネージド Bottlerocket ノードの作成

> **Note**
>
> マネージド型ノードグループでは、ユースケースにいくつかの利点がある場合があります。詳細については、「the section called “マネージド型ノードグループ”」を参照してください。

このトピックでは、Amazon EKS クラスターに登録する Bottlerocket ノードの Auto Scaling グループを起動する方法について説明します。Bottlerocket は AWS が提供する Linux ベースのオープンソースのオペレーティングシステムで、仮想マシンやベアメタルホスト上でコンテナを実行するために使用できます。ノードがクラスターに参加したら、それらのノードに Kubernetes アプリケーションをデプロイ可能になります。Botttlerocket の詳細については、GitHub の Using a Bottlerocket AMI with Amazon EKS と `eksctl` ドキュメント内の Custom AMI support を参照してください。

インプレースアップグレードについて詳しくは、GitHub の Bottlerocket Update Operator を参照してください。

Important

- Amazon EKS ノードは標準の Amazon EC2 インスタンスであり、通常の Amazon EC2 インスタンス価格に基づいて請求されます。詳細については「Amazon EC2 料金」を参照してください。
- AWS Outposts 上の Amazon EKS 拡張クラスターで Bottlerocket ノードを起動できますが、AWS Outposts 上のローカルクラスターでは起動できません。詳細については、「Amazon EKS on AWS Outposts」を参照してください。
- x86 または Arm プロセッサを使用して Amazon EC2 インスタンスにデプロイできます。ただし、Inferentia チップがあるインスタンスにデプロイすることはできません。
- Bottlerocket は、AWS CloudFormation と互換性があります。ただし、Amazon EKS の Bottlerocket ノードをデプロイするためにコピーできる公式の CloudFormation テンプレートはありません。
- Bottlerocket のイメージには、SSH サーバーやシェルは付属していません。SSH で管理者コンテナを有効にし、ユーザーデータとともにブートストラップの設定ステップを渡すために、帯域外のアクセス方式を使用できます。詳細については、GitHub の「bottlerocket readme.md」のセクションを参照してください。
  - 探査
  - 管理者コンテナ
  - Kubernetes 設定

この手順には、eksctl バージョン 0.215.0 以降が必要です。お使いのバージョンは、以下のコマンドを使用して確認できます。

```
eksctl version
```

eksctl のインストールまたはアップグレードの手順については、eksctl ドキュメントの「インストール」を参照してください。

Note

この手順はeksctl で作成されたクラスターに対してのみ機能します。

1. 次のコンテンツをデバイスにコピーします。*#######* の部分は自分のクラスター名に置き換えます。この名前には英数字 (大文字と小文字が区別されます) とハイフンのみを使用できます。先頭の文字は英数字である必要があります。また、100 文字より長くすることはできません。名前はクラスターを作成する AWS リージョンおよび AWS アカウント内で一意である必要があります。*ng-bottlerocket* をノードグループの名前に置き換えます。ノードグループ名は 63 文字以下である必要があります。先頭は文字または数字でなければなりませんが、残りの文字にはハイフンおよびアンダースコアを含めることもできます。Arm インスタンスにデプロイするには、*m5.large* を Arm インスタンスタイプに置き換えます。*my-ec2-keypair-name* を、起動後に、SSH を使用してノードに接続するときに使用できる Amazon EC2 SSH キーペアの名前に置き換えます。Amazon EC2 キーペアをまだ持っていない場合は、AWS マネジメントコンソールで作成できます。詳細については「Amazon EC2 ユーザーガイド」の「Amazon EC2 キーペア」を参照してください。残りのサンプル値はすべて独自の値に置き換えます。置き換えが完了したら、変更したコマンドを実行して `bottlerocket.yaml` ファイルを作成します。

   Arm Amazon EC2 インスタンスタイプを指定する場合は、デプロイする前に「Amazon EKS 最適化 Arm Amazon Linux AMI」の考慮事項を確認してください。カスタム AMI を使用したデプロイの手順については、GitHub の Building Bottlerocket と、`eksctl` ドキュメントの Custom AMI support を参照してください。マネージド型ノードグループをデプロイするには、起動テンプレートを使用してカスタム AMI をデプロイします。詳細については、「the section called “起動テンプレート”」を参照してください。

   > **Important**
   >
   > ノードグループを AWS Outposts、AWS Wavelength、または AWS Local Zone サブネットにデプロイする場合、クラスターの作成時に AWS Outposts、AWS Wavelength、または AWS Local Zone サブネットは渡さないでください。次の例では、サブネットを指定する必要があります。詳細については、`eksctl` ドキュメントの「設定ファイルからノードグループを作成する」と「Config ファイルのスキーマ」を参照してください。*region-code* を、クラスターのある AWS リージョンに置き換えます。

```
cat >bottlerocket.yaml <<EOF
---
apiVersion: eksctl.io/v1alpha5
kind: ClusterConfig

metadata:
  name: my-cluster
```

```
  region: region-code
  version: '1.37'

iam:
  withOIDC: true

nodeGroups:
  - name: ng-bottlerocket
    instanceType: m5.large
    desiredCapacity: 3
    amiFamily: Bottlerocket
    ami: auto-ssm
    iam:
       attachPolicyARNs:
          - arn:aws:iam::aws:policy/AmazonEKSWorkerNodePolicy
          - arn:aws:iam::aws:policy/AmazonEC2ContainerRegistryReadOnly
          - arn:aws:iam::aws:policy/AmazonSSMManagedInstanceCore
          - arn:aws:iam::aws:policy/AmazonEKS_CNI_Policy
    ssh:
        allow: true
        publicKeyName: my-ec2-keypair-name
EOF
```

2. 次のコマンドでノードをデプロイします。

```
eksctl create nodegroup --config-file=bottlerocket.yaml
```

出力例は次のとおりです。

ノードの作成中に、複数の行が出力されます。出力の最後の行は次のサンプル行が表示されます。

```
[#]  created 1 nodegroup(s) in cluster "my-cluster"
```

3. (オプション) Amazon EBS CSI プラグインを使用して、Bottlerocket ノード上に Kubernetes 永続ボリュームを作成します。デフォルトの Amazon EBS ドライバーは、Bottlerocket に含まれていないファイルシステムツールを利用します。ドライバーを使用してストレージクラスを作成する方法の詳細については、「the section called “Amazon EBS”」を参照してください。
4. (オプション) デフォルトでは、kube-proxy は nf_conntrack_max カーネルパラメータを、Bottlerocket がブート時に設定するものとは異なるデフォルト値に設定します。Bottlerocket のデフォルト設定を保持するには、次のコマンドを使用して kube-proxy 設定を編集します。

```
kubectl edit -n kube-system daemonset kube-proxy
```

次の例にある kube-proxy 引数に、--conntrack-max-per-core と --conntrack-min を追加します。0 の設定は、変更がないことを意味します。

```
      containers:
      - command:
        - kube-proxy
        - --v=2
        - --config=/var/lib/kube-proxy-config/config
        - --conntrack-max-per-core=0
        - --conntrack-min=0
```

5. (オプション) サンプルアプリケーションをデプロイして、Bottlerocket ノードをテストします。
6. 次の条件が true の場合、IMDS への Pod アクセスをブロックすることをお勧めします。
   - IAM ロールをすべての Kubernetes サービスアカウントに割り当てることによって、必要最小限のアクセス許可のみを Pod に付与しようとしている。
   - クラスター内の Pod が、現在の AWS リージョンの取得など、その他の理由で Amazon EC2 インスタンスメタデータサービス (IMDS) へのアクセスを必要としていない。

   詳細については「ワーカーノードに割り当てられたインスタンスプロファイルへのアクセスを制限する」を参照してください。

セルフマネージド Microsoft Windows ノードの作成

このトピックでは、Amazon EKS クラスターに登録されている Windows ノードの Auto Scaling グループの起動方法について説明します。ノードがクラスターに参加したら、それらのノードに Kubernetes アプリケーションをデプロイ可能になります。

> **Important**
>
> - Amazon EKS ノードは標準の Amazon EC2 インスタンスであり、通常の Amazon EC2 インスタンス価格に基づいて請求されます。詳細については「Amazon EC2 料金」を参照してください。

- AWS Outposts 上の Amazon EKS 拡張クラスターで Windows ノードを起動できますが、AWS Outposts 上のローカルクラスターでは起動できません。詳細については、「Amazon EKS on AWS Outposts」を参照してください。

クラスターに対して Windows サポートを有効にします。Windows ノードグループを起動する前に、重要な考慮事項を確認することをお勧めします。詳細については、「the section called “Windows サポートを有効にする”」を参照してください。

次のいずれかを使用して、セルフマネージド型の Windows ノードを起動できます。

- the section called “eksctl”
- the section called “AWS マネジメントコンソール”

eksctl

eksctl を使用して自己管理型 Windows ノードを起動します。

この手順ではeksctl をインストール済みで、お使いの eksctl バージョンが 0.215.0 以上であることが必要です。お使いのバージョンは以下のコマンドを使用して確認できます。

```
eksctl version
```

eksctl のインストールまたはアップグレードの手順についてはeksctl ドキュメントの「インストール」を参照してください。

> **Note**
>
> この手順はeksctl で作成されたクラスターに対してのみ機能します。

1. (オプション) AmazonEKS_CNI_Policy マネージド IAM ポリシー (IPv4 クラスターがある場合) または *AmazonEKS_CNI_IPv6_Policy* (IPv6 クラスターがある場合にはユーザー自身が作成したもの) が Amazon EKS ノードの IAM ロールにアタッチされている場合、代わりに Kubernetes aws-node サービスアカウントに関連付けた IAM ロールに割り当てることをお勧めします。詳細については、「the section called “IRSA 用の設定”」を参照してください。
2. この手順では既存のクラスターがあることを前提としています。Windows ノードグループの追加先となる Amazon EKS クラスターと Amazon Linux ノードグループがまだ存在しない場合は、the

section called “クラスターを作成する (eksctl)” に従うことをお勧めします。このガイドはAmazon Linux ノードで Amazon EKS クラスターを作成する方法についての完全なチュートリアルを提供します。

次のコマンドを使用して、ノードグループを作成します。*#####* を、クラスターのある AWS リージョンに置き換えます。*#######* の部分は自分のクラスター名に置き換えます。この名前には英数字 (大文字と小文字が区別されます) とハイフンのみを使用できます。先頭の文字は英数字である必要があります。また、100 文字より長くすることはできません。名前はクラスターを作成する AWS リージョンおよび AWS アカウント内で一意である必要があります。*ng-windows* をノードグループの名前に置き換えます。ノードグループ名は 63 文字以下である必要があります。先頭は文字または数字でなければなりませんが、残りの文字にはハイフンおよびアンダースコアを含めることもできます。*2019* を 2022 に置き換えて Windows Server 2022 を使用するか、もしくは 2025 に置き換えて Windows Server 2025 を使用できます。残りのサンプル値は独自の値に置き換えます。

> **Important**
>
> ノードグループを AWS Outposts、AWS 波長、または AWS ローカルゾーン サブネットにデプロイする場合、クラスターの作成時に AWS Outposts、波長、または ローカルゾーン サブネットは渡さないでください。AWS Outposts、波長、または ローカルゾーンサブネットを指定した設定ファイルを使用して、ノードグループを作成します。詳細についてはeksctl ドキュメントの「設定ファイルからノードグループを作成する」と「設定ファイルのスキーマ」を参照してください。

```
eksctl create nodegroup \
    --region region-code \
    --cluster my-cluster \
    --name ng-windows \
    --node-type t2.large \
    --nodes 3 \
    --nodes-min 1 \
    --nodes-max 4 \
    --managed=false \
    --node-ami-family WindowsServer2019FullContainer
```

> **Note**
>
> - ノードがクラスターに参加できない場合はトラブルシューティングガイドの「the section called “ノードがクラスターへの参加に失敗します”」を参照してください。
> - `eksctl` コマンドで使用可能なオプションを表示するには次のコマンドを入力してください。
>
> ```
> eksctl command -help
> ```

出力例は次のとおりです。ノードの作成中に、複数の行が出力されます。出力の最後の行は次のサンプル行が表示されます。

```
[#]  created 1 nodegroup(s) in cluster "my-cluster"
```

3. (オプション) サンプルアプリケーションをデプロイして、クラスターと Windows ノードをテストします。
4. 次の条件が true の場合、IMDS への Pod アクセスをブロックすることをお勧めします。
   - IAM ロールをすべての Kubernetes サービスアカウントに割り当てることによって、必要最小限のアクセス許可のみを Pod に付与しようとしている。
   - クラスター内の Pod が、現在の AWS リージョンの取得など、その他の理由で Amazon EC2 インスタンスメタデータサービス (IMDS) へのアクセスを必要としていない。

   詳細については、「ワーカーノードに割り当てられたインスタンスプロファイルへのアクセスを制限する」を参照してください。

AWS マネジメントコンソール

前提条件

- 既存の Amazon EKS クラスターと Linux ノードグループ。これらのリソースがない場合ははじめに のガイドのいずれかに従って作成することをお勧めします。これらのガイドでは、Linux ノードで Amazon EKS クラスターを作成する方法について説明しています。
- Amazon EKS クラスターの要件を満たす、既存の VPC およびセキュリティグループ。詳細についてはthe section called “VPC およびサブネットの要件”およびthe section called “セキュリティグループの要件”を参照してください。はじめに のガイドでは要件を満たす VPC を作成します。ま

たは「Amazon EKS クラスターの Amazon VPC を作成する」に従って手動で作成することもできます。

- Amazon EKS クラスターの要件を満たす VPC およびセキュリティグループを使用している、既存の Amazon EKS クラスター。詳細については「the section called “クラスターを作成する”」を参照してください。AWS Outposts、AWS 波長、または AWS ローカルゾーンが有効になっている AWS リージョンにサブネットがある場合はクラスターの作成時にそれらのサブネットが渡されるべきではありません。

ステップ 1: AWS マネジメントコンソール を使用してセルフマネージド型の Windows ノードを起動する

1. クラスターステータスが ACTIVE と表示されるまで待ちます。クラスターがアクティブになる前にノードを起動した場合、ノードはクラスターへの登録に失敗し、再起動が必要になります。
2. AWS クラウドフォーメーション コンソールを開きます
3. [スタックの作成] を選択してください。
4. [テンプレートを指定] で、[Amazon S3 URL] を選択してください。
5. 次の URL をコピーして、[Amazon S3 URL] に貼り付けます。

```
https://s3.us-west-2.amazonaws.com/amazon-eks/cloudformation/2023-02-09/amazon-eks-windows-nodegroup.yaml
```

6. [次へ] を 2 回選択してください。
7. [スタックのクイック作成] ページで、必要に応じて以下のパラメータを入力してください。
   - スタック名: AWS クラウドフォーメーション スタックのスタック名を選択してください。例えば、my-cluster-nodes という名前にすることができます。
   - [クラスター名]: Amazon EKS クラスターの作成時に使用した名前を入力してください。

     > **Important**
     >
     > この名前は「ステップ 1: Amazon EKS クラスターを作成する」で使用した名前と完全に一致する必要があります。そうしないと、ノードはクラスターに参加できません。

   - クラスター制御プレーンセキュリティグループ: VPC を作成する際に生成した AWS クラウドフォーメーション 出力からセキュリティグループを選択してください。次のステップでは該当するグループを取得する 1 つの方法を説明します。

     a. Amazon EKS コンソールを開きます。

b. クラスターの名前を選択してください。

c. [ネットワーキング] タブを選択してください。

d. [クラスター制御プレーンセキュリティグループ] ドロップダウンリストから選択する場合は[追加のセキュリティグループ] の値をリファレンスとして使用します。

- [ノードグループ名]: ノードグループの名前を入力してください。この名前はノードに対して作成される自動スケーリングノードグループを識別するために後で使用できます。ノードグループ名は 63 文字以下である必要があります。先頭は文字または数字でなければなりませんが、残りの文字にはハイフンおよびアンダースコアを含めることもできます。
- [ノードオートスケーリンググループ最小サイズ]: ノードのAuto Scaling グループがスケールインできる最小ノード数を入力してください。
- ノード自動スケーリンググループ希望容量: スタック作成時にスケーリングする必要のあるノード数を入力してください。
- [NodeAutoScalingGroupMaxSize]: ノードのAuto Scaling グループがスケールアウトできる最大ノード数を入力してください。
- [ノードインスタンス型]: ノードのインスタンスタイプを選択してください。詳細については、「the section called “Amazon EC2 インスタンスタイプ”」を参照してください。

> **Note**
>
> 最新バージョンの Amazon VPC CNI plugin for Kubernetes でサポートされているインスタンスタイプは、GitHub 上の vpc_ip_resource_limit.go にリストされています。サポートされている最新のインスタンスタイプを利用するには、CNI のバージョンを更新することが必要になる場合があります。詳細については、「the section called “Amazon VPC CNI”」を参照してください。

- [NodeImageIdSSMParam]: 現在推奨されている Amazon EKS 最適化 Windows Core AMI ID の Amazon EC2 Systems Manager パラメータが事前設定されています。Windows の通常版を使用する場合は、*Core* を `Full` に置き換えます。
- ノードイメージId: (オプション) (Amazon EKS 最適化 AMI の代わりに) 独自のカスタム AMI を使用している場合はAWS リージョンのノード AMI ID を入力してください。このフィールドに値を指定すると、[ノードイメージIdSSMParam] フィールドの値はすべて上書きされます。
- [ノードボリュームサイズ]: ノードのルートボリュームのサイズを GiB 単位で指定します。
- [キー名]: 起動後に、SSH を使用してノードに接続するときに使用できる Amazon EC2 SSH キーペアの名前を入力してください。Amazon EC2 キーペアをまだ持っていない場合はAWS マ

ネジメントコンソール で作成できます。詳細については「Amazon EC2 ユーザーガイド」の「Amazon EC2 キーペア」を参照してください。

> **Note**
>
> ここでキーペアを指定しないと、AWS クラウドフォーメーション スタックの作成は失敗します。

- [ブートストラップ引数]: `kubelet` を使用して、`-KubeletExtraArgs` の追加引数など、ノードブートストラップスクリプトに渡すオプションの引数を指定します。
- [無効IMDSv1]: 各ノードはデフォルトでインスタンスメタデータサービスバージョン 1 (IMDSv1) および IMDSv2 をサポートします。IMDSv1 は無効にできます。ノードグループ内の将来のノードおよび Pod が MDSv1 を使用しないようにするには DisableIMDSv1 を true に設定します。IDMS の詳細については「インスタンスメタデータサービスの設定」を参照してください。
- [VpcId]: 作成した VPC の ID を選択します。
- [NodeSecurityGroups]: VPC の作成時に Linux ノードグループ用に作成したセキュリティグループを選択します。Linux ノードに複数のセキュリティグループがアタッチされている場合、それらのすべてを指定します。これは、例えば、Linux ノードグループが `eksctl` を使用して作成された場合に当てはまります。
- [Subnets]: 作成したサブネットを選択してください。「Amazon EKS クラスターの Amazon VPC を作成する」で説明されている手順で VPC を作成した場合は起動するノードの VPC 内のプライベートサブネットのみを指定します。

> **Important**
>
> - いずれかのサブネットがパブリックサブネットである場合はパブリック IP アドレスの自動割り当て設定を有効にする必要があります。この設定がパブリックサブネットに対して有効になっていない場合、そのパブリックサブネットにデプロイするノードにはパブリック IP アドレスが割り当てられず、クラスターやその他の AWS のサービスと通信できなくなります。2020 年 3 月 26 日以前に、Amazon EKS AWS クラウドフォーメーション VPC テンプレートを使用して、または `eksctl` を使用してサブネットがデプロイされた場合、パブリックサブネットではパブリック IP アドレスの自動割り当てが無効になります。サブネットのパブリック IP アドレス割り当てを有効にする方法については「サブネットのパブリック IPv4 アドレス属性の変更」を参

照してください。ノードがプライベートサブネットにデプロイされている場合、NAT ゲートウェイを介してクラスターや他の AWS のサービスと通信できます。

- サブネットにインターネットアクセスがない場合は「インターネットアクセスが制限されたプライベートクラスターをデプロイする」の考慮事項と追加の手順を確認してください
- AWS Outposts、波長、またはローカルゾーンサブネット を選択する場合はクラスターの作成時にサブネットを渡さないようにします。

8. スタックが IAM リソースを作成する可能性があることを確認し、[スタックの作成] を選択してください。
9. スタックの作成が完了したら、コンソールで選択し、[出力] を選択してください。
10. 作成されたノードグループの [NodeInstanceRole] を記録します。これは、Amazon EKS Windows ノードを設定するときに必要になります。

ステップ 2: ノードを有効にしてクラスターに参加する

1. aws-auth ConfigMap がすでにあるかどうかを確認します。

```
kubectl describe configmap -n kube-system aws-auth
```

2. aws-auth ConfigMap が表示されている場合は必要に応じて更新してください。
   a. 編集する ConfigMap を開きます。

```
kubectl edit -n kube-system configmap/aws-auth
```

   b. 必要に応じて新しい mapRoles エントリを追加します。rolearn 値を、前の手順で記録した [ノードインスタンス役割] の値に設定します。

```
[...]
data:
  mapRoles: |
- rolearn: <ARN of linux instance role (not instance profile)>
      username: system:node:{{EC2PrivateDNSName}}
      groups:
        - system:bootstrappers
        - system:nodes
    - rolearn: <ARN of windows instance role (not instance profile)>
      username: system:node:{{EC2PrivateDNSName}}
```

```
      groups:
        - system:bootstrappers
        - system:nodes
        - eks:kube-proxy-windows
[...]
```

c. ファイルを保存し、テキストエディタを終了します。

3. 「Error from server (NotFound): configmaps "aws-auth" not found」というエラーが表示されたら、ストック ConfigMap を適用してください。

a. 設定マップをダウンロードします。

```
curl -O https://s3.us-west-2.amazonaws.com/amazon-eks/cloudformation/2020-10-29/
aws-auth-cm-windows.yaml
```

b. aws-auth-cm-windows.yaml ファイルで、rolearn 値を、前の手順で記録した該当する [ノードインスタンス役割] 値に設定します。これを行うにはテキストエディタを使用するか、このサンプル値を置き換えて次のコマンドを実行してください。

```
sed -i.bak -e 's|<ARN of linux instance role (not instance profile)>|my-node-
linux-instance-role|' \
    -e 's|<ARN of windows instance role (not instance profile)>|my-node-windows-
instance-role|' aws-auth-cm-windows.yaml
```

> **Important**
>
> - このファイルの他の行は変更しないでください。
> - Windows ノードと Linux ノードの両方に同じ IAM ロールを使用しないでください。

c. 設定を適用します。このコマンドが完了するまで数分かかることがあります。

```
kubectl apply -f aws-auth-cm-windows.yaml
```

4. ノードのステータスを監視し、Ready ステータスになるまで待機します。

```
kubectl get nodes --watch
```

Ctrl+C を入力して、シェルプロンプトに戻ります。

> **Note**
>
> 認可またはリソースタイプのエラーが発生した場合はトラブルシューティングトピックの「the section called “許可されていないか、アクセスが拒否されました (kubectl)”」を参照してください。

ノードがクラスターに参加できない場合はトラブルシューティングの章にある「the section called “ノードがクラスターへの参加に失敗します”」を参照してください。

ステップ 3: その他のアクション

1. (オプション) サンプルアプリケーションをデプロイして、クラスターと Windows ノードをテストします。
2. (オプション) AmazonEKS_CNI_Policy マネージド IAM ポリシー (IPv4 クラスターがある場合) または *AmazonEKS_CNI_IPv6_Policy* (IPv6 クラスターがある場合にはユーザー自身が作成したもの) が Amazon EKS ノードの IAM ロールにアタッチされている場合、代わりに Kubernetes aws-node サービスアカウントに関連付けた IAM ロールに割り当てることをお勧めします。詳細については、the section called “IRSA 用の設定” を参照してください。
3. 次の条件が true の場合、IMDS への Pod アクセスをブロックすることをお勧めします。
   - IAM ロールをすべての Kubernetes サービスアカウントに割り当てることによって、必要最小限のアクセス許可のみを Pod に付与しようとしている。
   - クラスター内の Pod が、現在の AWS リージョンの取得など、その他の理由で Amazon EC2 インスタンスメタデータサービス (IMDS) へのアクセスを必要としていない。

   詳細については「ワーカーノードに割り当てられたインスタンスプロファイルへのアクセスを制限する」を参照してください。

セルフマネージドの Ubuntu Linux ノードを作成する

> **Note**
>
> マネージド型ノードグループでは、ユースケースにいくつかの利点がある場合があります。詳細については、「the section called “マネージド型ノードグループ”」を参照してください。

このトピックでは、Amazon Elastic Kubernetes Service (EKS) ノード上で Ubuntu の Auto Scaling グループを起動するか、あるいは Amazon EKS クラスターに登録される Amazon Elastic Kubernetes Service (EKS) ノード上で Ubuntu Pro の Auto Scaling グループを起動する方法について説明します。EKS で動作する Ubuntu と Ubuntu Pro は、公式の Ubuntu Minimal LTS に基づいており、AWS と共同で開発されているカスタム AWS カーネルを搭載し、これまで特に EKS 向けに構築されています。Ubuntu Pro は、EKS 延長サポート期間、カーネルライブパッチ、FIPS の遵守、および無制限の Pro コンテナを実行する機能をサポートすることで、さらなるセキュリティの強化も図っています。

ノードがクラスターに参加したら、それらのノードにコンテナ化したアプリケーションをデプロイできるようになります。詳細については、「Ubuntu on AWS」のドキュメントおよび `eksctl` ドキュメントの「Custom AMI support」を参照してください。

> **Important**
>
> - Amazon EKS ノードは標準の Amazon EC2 インスタンスであり、通常の Amazon EC2 インスタンス価格に基づいて請求されます。詳細については「Amazon EC2 料金」を参照してください。
> - AWS Outposts 上の Amazon EKS 拡張クラスターで Ubuntu ノードを起動できますが、AWS Outposts 上のローカルクラスターでは起動できません。詳細については、「Amazon EKS on AWS Outposts」を参照してください。
> - x86 または Arm プロセッサを使用して Amazon EC2 インスタンスにデプロイできます。ただし、インスタンスに Inferentia チップがある場合は、先に Neuron SDK をインストールしておく必要があります。

この手順には、`eksctl` バージョン `0.215.0` 以降が必要です。お使いのバージョンは、以下のコマンドを使用して確認できます。

```
eksctl version
```

eksctl のインストールまたはアップグレードの手順については、eksctl ドキュメントの「インストール」を参照してください。

> **Note**
>
> この手順はeksctl で作成されたクラスターに対してのみ機能します。

1. 次のコンテンツをデバイスにコピーします。my-cluster を自分のクラスター名に置き換えます。この名前には英数字 (大文字と小文字が区別されます) とハイフンのみを使用できます。先頭の文字はアルファベット文字である必要があります。また、100 文字より長くすることはできません。ng-ubuntu をノードグループの名前に置き換えます。ノードグループ名は 63 文字以下である必要があります。先頭は文字または数字でなければなりませんが、残りの文字にはハイフンおよびアンダースコアを含めることもできます。Arm インスタンスにデプロイするには、m5.large を Arm インスタンスタイプに置き換えます。my-ec2-keypair-name を、起動後に、SSH を使用してノードに接続するときに使用できる Amazon EC2 SSH キーペアの名前に置き換えます。Amazon EC2 キーペアをまだ持っていない場合は、AWS マネジメントコンソール で作成できます。詳細については、「Amazon EC2 ユーザーガイド」の「Amazon EC2 キーペア」を参照してください。残りのサンプル値はすべて独自の値に置き換えます。置き換えが完了したら、変更したコマンドを実行して ubuntu.yaml ファイルを作成します。

   > **Important**
   >
   > ノードグループを AWS Outposts、AWS Wavelength、または AWS Local Zone サブネットにデプロイする場合、クラスターの作成時に AWS Outposts、AWS Wavelength、または AWS Local Zone サブネットは渡さないでください。次の例では、サブネットを指定する必要があります。詳細については、eksctl ドキュメントの「設定ファイルからノードグループを作成する」と「Config ファイルのスキーマ」を参照してください。*region-code* を、クラスターのある AWS リージョンに置き換えます。

   ```
   cat >ubuntu.yaml <<EOF
   ---
   apiVersion: eksctl.io/v1alpha5
   kind: ClusterConfig
   ```

```
metadata:
  name: my-cluster
  region: region-code
  version: '1.37'

iam:
  withOIDC: true

nodeGroups:
  - name: ng-ubuntu
    instanceType: m5.large
    desiredCapacity: 3
    amiFamily: Ubuntu2204
    iam:
       attachPolicyARNs:
          - arn:aws:iam::aws:policy/AmazonEKSWorkerNodePolicy
          - arn:aws:iam::aws:policy/AmazonEC2ContainerRegistryReadOnly
          - arn:aws:iam::aws:policy/AmazonSSMManagedInstanceCore
          - arn:aws:iam::aws:policy/AmazonEKS_CNI_Policy
    ssh:
        allow: true
        publicKeyName: my-ec2-keypair-name
EOF
```

Ubuntu Pro ノードグループを作成するには、amiFamily 値を UbuntuPro2204 に変更します。

2. 次のコマンドでノードをデプロイします。

```
eksctl create nodegroup --config-file=ubuntu.yaml
```

出力例は次のとおりです。

ノードの作成中に、複数の行が出力されます。出力の最後の行は次のサンプル行が表示されます。

```
[#]  created 1 nodegroup(s) in cluster "my-cluster"
```

3. (オプション) サンプルアプリケーションをデプロイして Ubuntu ノードをテストします。
4. 次の条件が true の場合、IMDS への Pod アクセスをブロックすることをお勧めします。
   - IAM ロールをすべての Kubernetes サービスアカウントに割り当てることによって、必要最小限のアクセス許可のみを Pod に付与しようとしている。

- クラスター内の Pod が、現在の AWS リージョンの取得など、その他の理由で Amazon EC2 インスタンスメタデータサービス (IMDS) へのアクセスを必要としていない。

詳細については「ワーカーノードに割り当てられたインスタンスプロファイルへのアクセスを制限する」を参照してください。

クラスターのためにセルフマネージドノードを更新する

Amazon EKS に最適化された最新の AMI がリリースされたら、セフルマネージド型ノードグループのノードを新しい AMI に置き換えることを検討してください。同様に、Amazon EKS クラスターの Kubernetes バージョンを更新した場合は、ノードを更新して、同じ Kubernetes バージョンのノードを使用します。

> **Important**
>
> このトピックでは、セルフマネージド型ノードのノードの更新について説明します。マネージドノードグループを使用している場合は、「the section called “更新”」を参照してください。

クラスター内のセルフマネージド型ノードグループを更新して新しい AMI を使用するには、2 つの基本的な方法があります。

アプリケーションを新しいノードグループに移行する

新しいノードグループを作成し、Pod をそのグループに移行します。新しいノードグループへの移行は、既存の AWS CloudFormation スタックで AMI ID を単に更新するよりも適切です。これは、移行プロセスが古いノードグループを `NoSchedule` としてテイントに設定し、新しいスタックが既存の Pod ワークロードを受け入れる準備ができた後にそのノードをドレインするためです。

AWS CloudFormation ノードスタックを更新する

既存のノードグループの AWS CloudFormation スタックを更新して、新しい AMI が使用されるようにします。この方法は、`eksctl` を使用して作成されたノードグループではサポートされていません。

アプリケーションを新しいノードグループに移行する

このトピックは、新しいノードグループを作成し、既存のアプリケーションを新しいグループに適切に移行し、クラスターから古いノードグループを削除する方法について説明します。新しいノードグループに移行するには、eksctl または AWS マネジメントコンソール を使用します。

- the section called “eksctl”
- the section called “AWS マネジメントコンソール および AWS CLI”

eksctl

eksctl を使用してアプリケーションを新しいノードグループに移行する

eksctl を移行に使用する方法の詳細については、eksctl ドキュメントの「管理対象外のノードグループ」を参照してください。

この手順には、eksctl バージョン 0.215.0 以降が必要です。お使いのバージョンは、以下のコマンドを使用して確認できます。

```
eksctl version
```

eksctl のインストールまたはアップグレードの手順については、eksctl ドキュメントの「インストール」を参照してください。

> **Note**
>
> この手順は、eksctl で作成されたクラスターとノードグループに対してのみ機能します。

1. *my-cluster* をクラスター名に置き換えて、既存のノードグループの名前を取得します。

   ```
   eksctl get nodegroups --cluster=my-cluster
   ```

   出力例は次のとおりです。

   ```
   CLUSTER      NODEGROUP          CREATED               MIN SIZE      MAX SIZE
    DESIRED CAPACITY     INSTANCE TYPE     IMAGE ID
   default      standard-nodes   2019-05-01T22:26:58Z  1             4             3
                  t3.medium         ami-05a71d034119ffc12
   ```

2. 次のコマンドを使用して、eksctl で新しいノードグループを起動します。コマンドのすべての#####を独自の値に置き換えます。バージョン番号をお使いのコントロールプレーンの Kubernetes バージョンよりも新しいものにすることはできません。また、コントロールプレーンの Kubernetes バージョンより 3 つ以上前のマイナーなバージョンにすることもできません。コントロールプレーンと同じバージョンを使用することをお勧めします。

   次の条件が true の場合、IMDS への Pod アクセスをブロックすることをお勧めします。

   - IAM ロールをすべての Kubernetes サービスアカウントに割り当てることによって、必要最小限のアクセス許可のみを Pod に付与しようとしている。
   - クラスター内の Pod が、現在の AWS リージョンの取得など、その他の理由で Amazon EC2 インスタンスメタデータサービス (IMDS) へのアクセスを必要としていない。

     詳細については「ワーカーノードに割り当てられたインスタンスプロファイルへのアクセスを制限する」を参照してください。

     Pod から IMDS へのアクセスをブロックするには、次のコマンドに --disable-pod-imds オプションを追加します。

     > **Note**
     >
     > 使用可能なフラグとその説明については、「https://eksctl.io/」を参照してください。

   ```
   eksctl create nodegroup \
     --cluster my-cluster \
     --version 1.37 \
     --name standard-nodes-new \
     --node-type t3.medium \
     --nodes 3 \
     --nodes-min 1 \
     --nodes-max 4 \
     --managed=false
   ```

3. 前のコマンドが完了したら、次のコマンドを使用して、すべてのノードが Ready 状態になったことを確認します。

   ```
   kubectl get nodes
   ```

4. 次のコマンドを使用して、元のノードグループを削除します。このコマンドで、すべての#####を自分のクラスター名とノードグループ名に置き換えます。

```
eksctl delete nodegroup --cluster my-cluster --name standard-nodes-old
```

AWS マネジメントコンソール および AWS CLI

AWS マネジメントコンソール と AWS CLI を使用してアプリケーションを新しいノードグループに移行する

1. 新しいノードグループを起動するには、「セルフマネージド Amazon Linux ノードを作成する」で説明されている手順に従います。
2. スタックの作成が完了したら、コンソールで選択し、[出力] を選択してください。
3. 作成されたノードグループの NodeInstanceRole を記録します。Amazon EKS ノードをクラスターに追加するには、これが必要です。

> **Note**
>
> 追加の IAM ポリシーを古いノードグループの IAM ロールにアタッチした場合は、この同じポリシーを新しいノードグループの IAM ロールにアタッチして、新しいグループでその機能を管理します。これは、たとえば、Kubernetes Cluster Autoscaler のアクセス許可を追加した場合に適用されます。

4. 相互通信できるように、両方のノードグループのセキュリティグループを更新します。詳細については、「the section called “セキュリティグループの要件”」を参照してください。
   a. 両方のノードグループのセキュリティグループ ID を記録します。これは、AWS CloudFormation スタックからの出力に、NodeSecurityGroup の値として表示されます。

      次の AWS CLI コマンドを使用して、スタック名からセキュリティグループ ID を取得します。これらのコマンドで、oldNodes は、古いノードスタックの AWS CloudFormation スタック名、newNodes は、移行先のスタックの名前です。各#####は独自の値に置き換えます。

```
oldNodes="old_node_CFN_stack_name"
newNodes="new_node_CFN_stack_name"

oldSecGroup=$(aws cloudformation describe-stack-resources --stack-name $oldNodes \
--query 'StackResources[?
ResourceType==`AWS::EC2::SecurityGroup`].PhysicalResourceId' \
--output text)
newSecGroup=$(aws cloudformation describe-stack-resources --stack-name $newNodes \
```

```
--query 'StackResources[?
ResourceType==`AWS::EC2::SecurityGroup`].PhysicalResourceId' \
--output text)
```

b. 互いにトラフィックを受け入れるように、各ノードのセキュリティグループに進入ルールを追加します。

以下の AWS CLI コマンドでは、他のセキュリティグループからのすべてのプロトコルのトラフィックをすべて許可するインバウンドルールを各セキュリティグループに追加します。この設定により、ワークロードを新しいグループに移行している際に、各ノードグループ内の Pod が相互に通信できるようになります。

```
aws ec2 authorize-security-group-ingress --group-id $oldSecGroup \
--source-group $newSecGroup --protocol -1
aws ec2 authorize-security-group-ingress --group-id $newSecGroup \
--source-group $oldSecGroup --protocol -1
```

5. aws-auth configmap を編集して、新しいノードインスタンスロールを RBAC にマッピングします。

```
kubectl edit configmap -n kube-system aws-auth
```

新しいノードグループの新しい mapRoles エントリを追加します。

```
apiVersion: v1
data:
  mapRoles: |
    - rolearn: ARN of instance role (not instance profile)
      username: system:node:{{EC2PrivateDNSName}}
      groups:
        - system:bootstrappers
        - system:nodes>
    - rolearn: arn:aws:iam::111122223333:role/nodes-1-16-NodeInstanceRole-
U11V27W93CX5
      username: system:node:{{EC2PrivateDNSName}}
      groups:
        - system:bootstrappers
        - system:nodes
```

########## ARN (################) スニペットを、前の手順で記録した NodeInstanceRole の値に置き換えます。次に、更新された configmap を適用するには、ファイルを保存して閉じます。

6. ノードのステータスを監視し、新しいノードがクラスターに参加して、Ready ステータスになるまで待機します。

```
kubectl get nodes --watch
```

7. (オプション) Kubernetes Cluster Autoscaler を使用している場合は、スケーリングアクションの競合を回避するために、デプロイをゼロ (0) レプリカにスケールダウンします。

```
kubectl scale deployments/cluster-autoscaler --replicas=0 -n kube-system
```

8. 以下のコマンドを使用して、NoSchedule で削除する各ノードをテイントに設定します。その目的は、ノードを置き換えたときにそのノードで新しい Pod がスケジュールまたは再スケジュールされないようにすることです。詳細については、Kubernetes ドキュメントの「テイントと容認」を参照してください。

```
kubectl taint nodes node_name key=value:NoSchedule
```

ノードを新しい Kubernetes バージョンにアップグレードする場合は、次のコードスニペットを使用して、特定の Kubernetes バージョン (この場合は 1.35) のすべてのノードを識別してテイントに設定できます。バージョン番号をお使いのコントロールプレーンの Kubernetes バージョンよりも新しいものにすることはできません。また、コントロールプレーンの Kubernetes バージョンより 3 つ以上前のマイナーなバージョンにすることもできません。コントロールプレーンと同じバージョンを使用することをお勧めします。

```
K8S_VERSION=1.35
nodes=$(kubectl get nodes -o jsonpath="{.items[?(@.status.nodeInfo.kubeletVersion==
\"v$K8S_VERSION\")].metadata.name}")
for node in ${nodes[@]}
do
    echo "Tainting $node"
    kubectl taint nodes $node key=value:NoSchedule
done
```

9. クラスターの DNS プロバイダーを決定します。

```
kubectl get deployments -l k8s-app=kube-dns -n kube-system
```

出力例は次のとおりです。このクラスターでは、DNS 解決に CoreDNS を使用していますが、クラスターからは kube-dns が返される場合もあります。

```
NAME      DESIRED   CURRENT   UP-TO-DATE   AVAILABLE   AGE
coredns   1         1         1            1           31m
```

10現在のデプロイメントで実行しているレプリカが 2 つ未満の場合は、そのデプロイメントを 2 つのレプリカにスケールアウトします。前のコマンドの出力が kubedns を返していた場合は、*coredns* を kubedns に置き換えます。

```
kubectl scale deployments/coredns --replicas=2 -n kube-system
```

11次のコマンドを使用して、クラスターから削除する各ノードをドレーンします。

```
kubectl drain node_name --ignore-daemonsets --delete-emptydir-data
```

ノードを新しい Kubernetes バージョンにアップグレードする場合は、次のコードスニペットを使用して、特定の Kubernetes バージョン (この場合は *1.35*) のすべてのノードを識別してドレインします。

```
K8S_VERSION=1.35
nodes=$(kubectl get nodes -o jsonpath="{.items[?(@.status.nodeInfo.kubeletVersion==
\"v$K8S_VERSION\")].metadata.name}")
for node in ${nodes[@]}
do
    echo "Draining $node"
    kubectl drain $node --ignore-daemonsets --delete-emptydir-data
done
```

12.古いノードのドレーンが完了したら、以前承認したセキュリティグループのインバウンドルールを取り消します。次に、AWS CloudFormation スタックを削除してインスタンスを終了してください。

> **Note**
>
> さらに IAM ポリシーを古いノードグループの IAM ロールにアタッチした場合 (Kubernetes Cluster Autoscaler に対するアクセス許可を追加する場合など) は、ロールか

らその追加ポリシーをデタッチしてから、AWS CloudFormation スタックを削除できます。

a. 先ほどノードのセキュリティグループ用に作成したインバウンドルールを取り消します。これらのコマンドで、oldNodes は、古いノードスタックの AWS CloudFormation スタック名、newNodes は、移行先のスタックの名前です。

```
oldNodes="old_node_CFN_stack_name"
newNodes="new_node_CFN_stack_name"

oldSecGroup=$(aws cloudformation describe-stack-resources --stack-name $oldNodes \
--query 'StackResources[?
ResourceType==`AWS::EC2::SecurityGroup`].PhysicalResourceId' \
--output text)
newSecGroup=$(aws cloudformation describe-stack-resources --stack-name $newNodes \
--query 'StackResources[?
ResourceType==`AWS::EC2::SecurityGroup`].PhysicalResourceId' \
--output text)
aws ec2 revoke-security-group-ingress --group-id $oldSecGroup \
--source-group $newSecGroup --protocol -1
aws ec2 revoke-security-group-ingress --group-id $newSecGroup \
--source-group $oldSecGroup --protocol -1
```

b. AWS CloudFormation コンソールを開きます。

c. 古いノードスタックを選択します。

d. [削除] を選択します。

e. [スタックの削除] 確認ダイアログボックスで、[スタックを削除] をクリックします。

13aws-auth configmap を編集して、RBAC から古いノードインスタンスロールを削除します。

```
kubectl edit configmap -n kube-system aws-auth
```

古いノードグループの mapRoles エントリを削除します。

```
apiVersion: v1
data:
  mapRoles: |
    - rolearn: arn:aws:iam::111122223333:role/nodes-1-16-NodeInstanceRole-
W70725MZQFF8
```

```
      username: system:node:{{EC2PrivateDNSName}}
      groups:
        - system:bootstrappers
        - system:nodes
    - rolearn: arn:aws:iam::111122223333:role/nodes-1-15-NodeInstanceRole-
U11V27W93CX5
      username: system:node:{{EC2PrivateDNSName}}
      groups:
        - system:bootstrappers
        - system:nodes>
```

更新された configmap を適用するには、ファイルを保存して閉じます。

14.(オプション) Kubernetes Cluster Autoscaler を使用している場合は、デプロイのスケーリングを 1 レプリカに戻します。

> **Note**
>
> 必要に応じて、新しい Auto Scaling グループにタグ (`k8s.io/cluster-autoscaler/enabled,k8s.io/cluster-autoscaler/my-cluster` など) を付け、新しくタグ付けした Auto Scaling グループを参照するように、Cluster Autoscaler デプロイのコマンドを更新する必要があります。詳細については、「AWS の Cluster Autoscaler」を参照してください。

```
kubectl scale deployments/cluster-autoscaler --replicas=1 -n kube-system
```

15.(オプション) 最新バージョンの Amazon VPC CNI Plugin for Kubernetes を使用していることを確認します。サポートされている最新のインスタンスタイプを利用するにはCNI のバージョンを更新することが必要になる場合があります。詳細については、「the section called “Amazon VPC CNI”」を参照してください。

16.クラスターが DNS 解決 ([migrate-determine-dns-step] を参照) に `kube-dns` を使用している場合は、`kube-dns` デプロイを 1 レプリカにスケールインします。

```
kubectl scale deployments/kube-dns --replicas=1 -n kube-system
```

AWS CloudFormation ノードスタックを更新する

このトピックは、新しい AMI を使用して既存の AWS CloudFormation セルフマネージド型ノードスタックを更新する方法について説明します。この手順を使用して、クラスターの更新後にノードを新しいバージョンの Kubernetes に更新することができます。それ以外の場合は、Kubernetes の既存バージョン用の Amazon EKS に最適化された最新の AMI に更新することができます。

> **Important**
>
> このトピックでは、セルフマネージド型ノードのノードの更新について説明します。「マネージドノードグループを使用してノードライフサイクルを簡素化する」方法の詳細については、「the section called “更新”」を参照してください。

最新のデフォルトの Amazon EKS ノード AWS CloudFormation テンプレートは、古い AMI を一度に 1 つずつ削除する前に、新しい AMI を使用してインスタンスをクラスターに起動するように設定されています。この設定により、ローリング更新中にクラスター内で Auto Scaling グループのアクティブなインスタンスが常に必要数確保されるようになります。

> **Note**
>
> この方法は、`eksctl` を使用して作成されたノードグループではサポートされていません。`eksctl` を使用してクラスターまたはノードグループを作成した場合は、「the section called “移行”」を参照してください。

1. クラスターの DNS プロバイダーを決定します。

```
kubectl get deployments -l k8s-app=kube-dns -n kube-system
```

出力例は次のとおりです。このクラスターでは、DNS 解決に CoreDNS を使用していますが、クラスターからは `kube-dns` が返される場合があります。使用しているバージョンの `kubectl` によって、出力が異なる場合があります。

```
NAME      DESIRED   CURRENT   UP-TO-DATE   AVAILABLE   AGE
coredns   1         1         1            1           31m
```

2. 現在のデプロイメントで実行しているレプリカが 2 つ未満の場合は、そのデプロイメントを 2 つのレプリカにスケールアウトします。前のコマンドの出力が kube-dns を返していた場合は、*coredns* を kube-dns に置き換えます。

```
kubectl scale deployments/coredns --replicas=2 -n kube-system
```

3. (オプション) Kubernetes Cluster Autoscaler を使用している場合は、スケーリングアクションの競合を回避するために、デプロイをゼロ (0) レプリカにスケールダウンします。

```
kubectl scale deployments/cluster-autoscaler --replicas=0 -n kube-system
```

4. 現在のノードグループのインスタンスタイプと必要なインスタンス数を決定します。グループの AWS CloudFormation テンプレートを更新する場合は、後でこれらの値を入力します。
   a. Amazon EC2 コンソールの https://console.aws.amazon.com/ec2/ を開いてください。
   b. 左ナビゲーションペインの [Launch Configurations] (起動設定) を選択し、既存のノードの起動設定のインスタンスタイプを書き留めます。
   c. 左ナビゲーションペインの [Auto Scaling] (グループ) を選択し、既存の Auto Scaling グループのノードの、インスタンスの [Desired] (必要) 数を書き留めます。
5. AWS CloudFormation コンソールを開きます。
6. ノードグループスタックを選択し、[更新] を選択します。
7. [Replace current template] (現在のテンプレートを置き換える) を選択し、Amazon S3 URL を選択します。
8. Amazon S3 URL で、ノードの AWS CloudFormation テンプレートの最新バージョンを使用していることを確認するために以下の URL をテキストエリアに貼り付けます。続いて、[次へ] を選択します。

```
https://s3.us-west-2.amazonaws.com/amazon-eks/cloudformation/2022-12-23/amazon-eks-nodegroup.yaml
```

9. [Specify stack details] (スタックの詳細の指定) ページで、以下のパラメータを指定し、[Next] (次へ) を選択します。
   - [NodeAutoScalingGroupDesiredCapacity] – 前のステップで記録した必要なインスタンス数を入力します。または、スタック更新時にスケーリングする必要のある新しいノード数を入力します。
   - NodeAutoScalingGroupMaxSize – ノードの Auto Scaling グループがスケールアウトする最大ノード数を入力します。この値は、必要な容量よりも少なくとも 1 ノード多い必要がありま

す。これは、更新時にノード数を減らさずにノードのローリング更新を実行できるようにするためです。

- NodeInstanceType – 前の手順で記録したインスタンスタイプを選択します。または、ノードに別のインスタンスタイプを選択します。異なるインスタンスタイプを選択する前に、「最適な Amazon EC2 ノードインスタンスタイプを選択する」を確認してください。各 Amazon EC2 インスタンスタイプは最大数の Elastic Network Interface (ネットワークインターフェイス) をサポートし、各ネットワークインターフェイスは最大数の IP アドレスをサポートします。ワーカーノードと Pod には、それぞれ IP アドレスが割り当てられています。そのため、各 Amazon EC2 ノードで実行する Pod の最大数をサポートするインスタンスタイプを選択することが重要です。インスタンスタイプごとにサポートされるネットワークインターフェイスと IP アドレスのリストについては、「各インスタンスタイプのネットワークインターフェイスごとの IP アドレス」を参照してください。例えば、`m5.large` インスタンスタイプは、ワーカーノードと Pod に対して最大 30 の IP アドレスをサポートします。

> **Note**
>
> 最新バージョンの Amazon VPC CNI plugin for Kubernetes でサポートされているインスタンスタイプは、GitHub 上の vpc_ip_resource_limit.go に示されています。サポートされている最新のインスタンスタイプを利用するには、Amazon VPC CNI Plugin for Kubernetes のバージョンを更新することが必要になる場合があります。詳細については、「the section called “Amazon VPC CNI”」を参照してください。

> **Important**
>
> AWS リージョンによっては利用できないインスタンスタイプがあります。

- [NodeImageIdSSMParam]: 更新する AMI ID の Amazon EC2 Systems Manager パラメータです。次の値では、Kubernetes バージョン `1.37` 用の最新の Amazon EKS 最適化 AMI が使用されています。

```
/aws/service/eks/optimized-ami/1.37/amazon-linux-2/recommended/image_id
```

`1.37` は、同一の platform-version に置き換えることができます。または、コントロールプレーンで実行されている Kubernetes バージョンより 1 つ前までのバージョンである必要があります。ノードは、コントロールプレーンと同じバージョンにしておくことをお勧めしま

す。*amazon-linux-2* を別の AMI タイプに置き換えることもできます。詳細については、「the section called “最新の ID を取得する”」を参照してください。

> **Note**
>
> Amazon EC2 Systems Manager パラメータを使用すると、AMI ID を検索して指定することなく、後でノードを更新できます。AWS CloudFormation スタックがこの値を使用している場合、スタックの更新によって常に、指定された Kubernetes バージョンで推奨される最新の Amazon EKS 最適化 AMI が起動されます。これは、テンプレートの値を変更しない場合も同様です。

- NodeImageId – 独自のカスタム AMI を使用するには、使用する AMI の ID を入力します。

> **Important**
>
> この値は、NodeImageIdSSMParam に指定されたすべての値を上書きします。NodeImageIdSSMParam 値を使用する場合は、 NodeImageId の値が空白であることを確認します。

- [DisableIMDSv1] – 各ノードは、デフォルトでインスタンスメタデータサービスバージョン 1 (IMDSv1) および IMDSv2 をサポートします。ただし、IMDSv1 を無効にできます。ノードグループでスケジュールされているノードまたは Pod で IMDSv1 を使用しない場合は、[true] を選択します。IDMS の詳細については、「インスタンスメタデータサービスの設定」を参照してください。サービスアカウントの IAM ロールを実装した場合は、AWS のサービスへのアクセスを必要とするすべての Pod に対し、必要なアクセス許可を直接割り当てます。これにより、クラスター内の Pod は、現在の AWS リージョンの取得など、その他の理由で IMDS へのアクセスを必要としません。次に、ホストネットワークを使用しない Pod の IMDSv2 へのアクセスを無効にすることもできます。詳細については、ワーカーノードに割り当てられたインスタンスプロファイルへのアクセスを制限する を参照してください。

10. (オプション) オプション ページで、スタックリソースをタグ付けします。[次へ] を選択します。
11. [確認] ページで情報を確認し、スタックで IAM リソースを作成することを承認して、[スタックの更新] を選択します。

> **Note**
>
> クラスター内の各ノードの更新には数分かかります。すべてのノードの更新が完了するのを待ってから、次の手順を実行します。

12.クラスターの DNS プロバイダーが kube-dns の場合は、kube-dns デプロイを 1 レプリカにスケールインします。

```
kubectl scale deployments/kube-dns --replicas=1 -n kube-system
```

13.(オプション) Kubernetes Cluster Autoscaler を使用している場合は、デプロイのスケーリングを目的のレプリカ数に戻します。

```
kubectl scale deployments/cluster-autoscaler --replicas=1 -n kube-system
```

14.(オプション) 最新バージョンの Amazon VPC CNI Plugin for Kubernetes を使用していることを確認します。サポートされている最新のインスタンスタイプを利用するには、Amazon VPC CNI Plugin for Kubernetes のバージョンを更新することが必要になる場合があります。詳細については、「the section called “Amazon VPC CNI”」を参照してください。

AWS Fargate を使用してコンピューティング管理を簡素化する

このトピックでは、Amazon EKS を使用して AWS Fargate で Kubernetes Pod を実行する方法について説明します。Fargate は、コンテナに適切なサイズのコンピューティング能力をオンデマンドで提供するテクノロジーです。Fargate を使用すると、コンテナを実行するために仮想マシンのグループをプロビジョニング、設定、スケールする必要がありません。これにより、サーバータイプの選択、ノードグループをスケールするタイミングの決定、クラスターのパッキングの最適化を行う必要がなくなります。

Fargate で起動する Pod と、Fargate プロファイルでの実行方法を制御できます。Fargate プロファイルは Amazon EKS クラスターの一部として定義されています。Amazon EKS は、Kubernetes が提供するアップストリームの拡張可能なモデルを使用して AWS により構築されたコントローラーを使用して、Kubernetes と Fargate を統合します。これらのコントローラーは、Amazon EKS マネージド Kubernetes コントロールプレーンの一部として実行され、ネイティブ Kubernetes Pod を Fargate にスケジューリングします。Fargate コントローラーには、いくつかの変更と検証アドミッションコントローラーに加えて、デフォルトの Kubernetes スケジューラーとともに実行される新しいスケジューラが含まれています。Fargate で実行する条件を満たす Pod を起動すると、クラスターで実行されている Fargate コントローラーは Pod を認識し、更新し、Fargate にスケジューリングします。

このトピックでは、Fargate で実行されている Pod のさまざまなコンポーネントについて説明し、Amazon EKS で Fargate を使用する際の特別な考慮事項を示します。

AWS Fargate 考慮事項

Amazon EKS での Fargate の使用についての考慮事項を以下に示します。

- Fargate で実行される各 Pod には、独自のコンピューティング境界があります。基本となるカーネル、CPU リソース、メモリリソース、または Elastic Network Interface を別のPod と共有しません。
- Network Load Balancer と Application Load Balancer (ALBs) は、IPターゲットでのみ Fargate で使用できます。詳細については、「the section called “ネットワークロードバランサーを作成する”」および「the section called “アプリケーション負荷分散”」を参照してください。
- Fargate 公開サービスは、ターゲットタイプの IP モードでのみ実行され、ノード IP モードでは実行されません。マネージド型ノードで実行されているサービスと Fargate で実行されているサービスからの接続を確認するためのおすすめの方法は、サービス名を使用して接続することです。
- Fargate で実行するには、ポッドがスケジューリングされた時点で Fargate プロファイルと一致する必要があります。Fargate プロファイルに一致しない Pod は `Pending` としてスタックする可能性があります。一致する Fargate プロファイルが存在する場合、Fargate に再スケジューリングするために作成した保留中の Pod を削除できます。
- デーモンセットは Fargate ではサポートされていません。アプリケーションにデーモンが必要な場合は、そのデーモンを Pod のサイドカーコンテナとして実行するように再設定します。
- 特権を持つコンテナは Fargate ではサポートされていません。
- Fargate で実行されている Pod は、`HostPort` または `HostNetwork` を Pod マニフェストに指定できません。
- Fargate Pod の場合、デフォルトの `nofile` および `nproc` のソフトリミットは 1024、ハードリミットは 65535 です。
- GPU は現在、Fargate では使用できません。
- Fargate で実行されているポッドは、プライベートサブネットでのみサポートされています (AWS のサービスへの NAT ゲートウェイアクセスができますが、インターネットゲートウェイへの直接ルーティングはできません)。そのため、クラスターの VPC ではプライベートサブネットを使用できるようにする必要があります。アウトバウンドインターネットアクセスのないクラスターについては、「」を参照してくださいthe section called “プライベートクラスター”
- 「Vertical Pod Autoscaler でポッドリソースを調整する」を使用して Fargate Pod の CPU とメモリの適切な初期サイズを設定し、「Horizontal Pod Autoscaler を使用してポッドデプロイをスケールする」を使用してそれらの Pod をスケールできます。Vertical Pod Autoscaler で、より大きい CPU とメモリの組み合わせを持つ Fargate に Pod を自動的に再デプロイする場合は、正しい機能

を確保するため、Vertical Pod Autoscaler のモードを `Auto` または `Recreate` に設定します。詳細については、GitHub で Vertical Pod Autoscaler のドキュメントを参照してください。

- VPC で DNS 解決と DNS ホスト名を有効にする必要があります。詳細については、「VPC の DNS サポートを更新する」を参照してください。
- Amazon EKS Fargate は、仮想マシン (VM) 内の各ポッドを分離することで、Kubernetes アプリケーションの多重防御を追加します。この VM 境界は、コンテナエスケープが発生した場合に他のポッドが使用するホストベースのリソースへのアクセスを防ぎます。これは、コンテナ化されたアプリケーションを攻撃し、コンテナ外のリソースにアクセスする一般的な方法です。

  Amazon EKS を使用しても、責任共有モデルに基づくお客様の責任は変わりません。クラスターのセキュリティとガバナンスのコントロールの設定について、慎重に検討する必要があります。アプリケーションを分離する最も安全な方法は、常に別のクラスターで実行することです。
- Fargate プロファイルでは、VPC セカンダリ CIDR ブロックからのサブネットの指定がサポートされています。セカンダリ CIDR ブロックを指定することもできます。サブネットの使用できる IP アドレスの数は限られています。結果的に、クラスター内に作成できる Pod の数も制限されます。Pod に別のサブネットを使用することで、使用可能な IP アドレスの数を増やすことができます。詳細については、「IPv4 CIDR ブロックの VPC への追加」を参照してください。
- Amazon EC2 インスタンスメタデータサービス (IMDS) は、Fargate ノードにデプロイされた Pod では使用できません。IAM 認証情報を必要とする Fargate にデプロイされた Pod がある場合は、サービスアカウントの IAM ロールを使用して Pod に割り当てます。Pod が IMDS を通じて利用可能な他の情報にアクセスする必要がある場合は、この情報を Pod 仕様にハードコーディングする必要があります。これには、Pod がデプロイされる AWS リージョンまたはアベイラビリティーゾーンが含まれます。
- AWS Outposts、AWS Wavelength、または AWS Local Zones に Fargate Pod をデプロイすることはできません。
- Amazon EKS は定期的に Fargate Pod にパッチを適用して安全に保つ必要があります。影響を軽減する方法で更新を試みますが、Pod のエビクションが正常に行われなかった場合、Pod を削除しなければならない場合があります。中断を最小限に抑えるために行えるアクションがいくつかあります。詳細については、「the section called “OS パッチ適用イベント”」を参照してください。
- Amazon VPC CNI plugin for Amazon EKS は、Fargate ノードにインストールされています。Fargate ノードでは Amazon EKS クラスターの代替 CNI プラグインを使用することはできません。
- Fargate 上で実行される Pod は、ドライバーの手動インストールステップなしで、Amazon EFS ファイルシステムを自動的にマウントします。Fargate ノードでは動的永続ボリュームプロビジョニングを使用できませんが、静的プロビジョニングを使用することはできます。

- Amazon EKS は Fargate Spot をサポートしていません。
- Amazon EBS ボリュームを Fargate Pod にマウントすることはできません。
- Amazon EBS CSI コントローラーは Fargate ノードで実行できますが、Amazon EBS CSI ノード DaemonSet は Amazon EC2 インスタンスでのみ実行できます。
- バージョンロールバック: バージョンロールバックは Fargate ワーカーノードではサポートされていません。Fargate を使用するクラスターのコントロールプレーンはロールバックできますが、コントロールプレーンと同じ Kubernetes バージョンを実行している Fargate ポッドは、ERROR ステータスの kubelet バージョンスキューインサイトをトリガーします。コントロールプレーンのバージョンと一致するマイナーバージョンを持つ Fargate ポッドがある場合は、ロールバックする前にそれらのポッドを削除します。次に、コントロールプレーンをロールバックします。残りのポッドは、再デプロイ時にロールバックバージョンで起動します。または、`--force` を使用してインサイトチェックをバイパスできますが、kubelet バージョンスキュー違反を続行すると、予期しない動作が発生する可能性があります。詳細については、「the section called “Kubernetes バージョンのロールバック”」を参照してください。
- Kubernetes Job が `Completed` または `Failed` とマークされた後も、Job が作成した Pod は通常存在し続けます。この動作によりログと結果を表示できますが、Fargate を使用する場合、その後 Job をクリーンアップしないとコストが発生します。

  Job が完了または失敗した後に、関連する Pod を自動的に削除するには、有効期限 (TTL) コントローラーを使用して期間を指定できます。次の例では、Job マニフェストで `.spec.ttlSecondsAfterFinished` を指定する方法を示しています。

```
apiVersion: batch/v1
kind: Job
metadata:
  name: busybox
spec:
  template:
    spec:
      containers:
      - name: busybox
        image: busybox
        command: ["/bin/sh", "-c", "sleep 10"]
      restartPolicy: Never
  ttlSecondsAfterFinished: 60 # <-- TTL controller
```

Fargate 比較テーブル

条件	AWS Fargate
AWS Outposts にデプロイ可能	いいえ
AWS Local Zone にデプロイ可能	いいえ
Windows を必要とするコンテナを実行できる	いいえ
Linux を必要とするコンテナを実行可能	はい
インフェクエンティア チップを必要とするワークロードを実行可能	いいえ
GPU を必要とするワークロードを実行可能	いいえ
Arm プロセッサを必要とするワークロードを実行可能	いいえ
AWS Bottlerocket を実行可能	いいえ
Pod は、カーネルランタイム環境を他の Pod と共有します。	いいえ: 各 Pod には専用のカーネルがあります。
ポッドはCPU、メモリ、ストレージ、およびネットワークリソースを他のポッドと共有します。	いいえ — 各 Pod には専用のリソースがあり、リソース使用率を最大化するために個別にサイズ設定できます。
Pod 仕様で要求されているよりも多くのハードウェアとメモリを、Pod が使用できます。	いいえ: より大きな vCPU およびメモリー設定を使用して、Pod を再デプロイできます。
Amazon EC2 インスタンスをデプロイおよび管理する必要がある	いいえ
Amazon EC2 インスタンスのオペレーティングシステムの保護、保守、パッチ適用を行う必要がある	いいえ

条件	AWS Fargate
ノードをデプロイする時に、追加の kubelet 引数 などのブートストラップ引数を提供できます。	いいえ
ノードに割り当てられた IP アドレスとは異なる CIDR ブロックから、IP アドレスを Pod に割り当てることができます。	いいえ
ノードに SSH 接続可能	いいえ – SSH 接続を行うノードホストオペレーティングシステムはありません。
独自のカスタム AMI をノードにデプロイ可能	いいえ
独自のカスタム CNI をノードにデプロイ可能	いいえ
ノード AMI を自分で更新する必要がある	いいえ
ノードの Kubernetes バージョンを自分で更新する必要があります。	いいえ: ノードを管理しません。
Pod で Amazon EBS ストレージを使用可能	いいえ
Pod で Amazon EFS ストレージを使用可能	はい
Pod で Amazon FSx for Lustre ストレージを使用可能	いいえ
Network Load Balancer をサービスに使用可能	はい、「Network Load Balancer を作成する」を使用する場合
ポッドはパブリックサブネットで実行可能	いいえ
それぞれの Pod に、異なる VPC セキュリティグループを割り当て可能	はい
Kubernetes DaemonSets を実行可能	いいえ

条件	AWS Fargate
Pod マニフェストで HostPort および HostNetwork をサポートします	いいえ
AWS が利用可能なリージョン	一部の Amazon EKS 対応リージョン
Amazon EC2 専有ホストでコンテナを実行できます	いいえ
料金	個々のFargate メモリとCPU構成のコスト。各 Pod には、独自のコストがあります。詳細については、「AWS Fargate の料金」を参照してください。

クラスターの AWS Fargate の使用を開始する

このトピックでは、Amazon EKS クラスターを使用して AWS Fargate で Pod の実行を開始する方法について説明します。

CIDR ブロックを使用してクラスターのパブリックエンドポイントへのアクセスを制限する場合は、プライベートエンドポイントアクセスも有効にすることをお勧めします。こうすることで、Fargate Pod がクラスターと通信できるようになります。プライベートエンドポイントが有効になっていない場合、パブリックアクセスに指定する CIDR ブロックに、VPC からのアウトバウンドソースを含める必要があります。詳細については、「the section called “クラスターエンドポイントのアクセス”」を参照してください。

前提条件

既存のクラスター。既存の Amazon EKS クラスターがなければ、はじめに を参照してください。

ステップ 1: 既存のノードが Fargate Pod と通信できることを確認する

ノードのない新しいクラスター、またはマネージド型ノードグループ (the section called “マネージド型ノードグループ” を参照) のみを持つクラスターを使用している場合は、「the section called “ステップ 2: Fargate Pod 実行ロールを作成する”」に進みます。

既にそれに関連付けられているノードがある既存のクラスターで作業していると仮定します。これらのノードの Pod が Fargate で実行されている Pod と自由に通信できることを確認する必要がありま

す。Fargate で実行されている Pod は、関連付けられているクラスターのクラスターセキュリティグループを使用するように自動的に設定されます。クラスター内の既存のノードが、クラスターセキュリティグループとの間でトラフィックを送受信できることを確認します。マネージドノードグループもクラスターセキュリティグループを使用するように自動的に設定されるため、この互換性を変更または確認する必要はありません (「the section called “マネージド型ノードグループ”」を参照)。

`eksctl` または Amazon EKS マネージド型 AWS CloudFormation テンプレートで作成された既存のノードグループの場合、クラスターセキュリティグループをノードに手動で追加できます。または、ノードグループの Auto Scaling グループ起動テンプレートを変更して、クラスターセキュリティグループをインスタンスにアタッチすることもできます。詳細については、Amazon VPC ユーザーガイドのインスタンスのセキュリティグループを変更するを参照してください。

AWS マネジメントコンソール で、クラスターの [Networking] (ネットワーク) セクションでクラスターのクラスターセキュリティグループを確認できます。これを行うには、次の AWS CLI コマンドを実行します。このコマンドを使用する場合は、`<my-cluster>` を自分のクラスター名に置き換えます。

```
aws eks describe-cluster --name <my-cluster> --query
 cluster.resourcesVpcConfig.clusterSecurityGroupId
```

ステップ 2: Fargate Pod 実行ロールを作成する

クラスターが AWS Fargate で Pod を作成する場合、Fargate インフラストラクチャで実行されるコンポーネントは、ユーザーに代わって AWS API を呼び出す必要があります。Amazon EKS の Pod 実行ロールにより、これらを行うための IAM アクセス許可が付与されます。AWS Fargate Pod 実行ロールを作成するには、「the section called “ポッド実行IAMロール”」を参照してください。

> **Note**
>
> `--fargate` オプションを使用して `eksctl` でクラスターを作成した場合は、クラスターに Pod 実行ロールが既にあり、これは `eksctl-my-cluster-FargatePodExecutionRole-ABCDEFGHIJKL` というパターンで IAM コンソールに表示されます。同様に、`eksctl` を使用して Fargate プロファイルを作成する場合、`eksctl` では Pod 実行ロールを作成します (まだ存在しない場合)。

ステップ 3: クラスターの Fargate プロファイルを作成する

クラスターの Fargate で実行されている Pod をスケジューリングする前に、起動時に Fargate を使用する Pod を指定する Fargate プロファイルを定義する必要があります。詳細については、「the section called “プロファイルを定義する”」を参照してください。

> **Note**
>
> `--fargate` オプションを使用して `eksctl` でクラスターを作成した場合、クラスターの Fargate プロファイルは、`kube-system` と `default` の名前空間のすべての Pod のセレクターを使用して既に作成されています。Fargate で使用するその他の名前空間の Fargate プロファイルを作成するには、以下の手順に従います。

Fargate プロファイルを作成するには、次のツールのいずれかを使用します。

- the section called “`eksctl`”
- the section called “AWS マネジメントコンソール”

`eksctl`

この手順には、`eksctl` バージョン `0.215.0` 以降が必要です。お使いのバージョンは、以下のコマンドを使用して確認できます。

```
eksctl version
```

`eksctl` のインストールまたはアップグレードの手順については、`eksctl` ドキュメントの「インストール」を参照してください。

で、Fargate プロファイルを作成するには**`eksctl`**

以下の `eksctl` コマンドで Fargate プロファイルを作成し、すべての `<example value>` を自分の値に置き換えます。名前空間を指定する必要があります。ただし、`--labels` オプションは必須ではありません。

```
eksctl create fargateprofile \
    --cluster <my-cluster> \
    --name <my-fargate-profile> \
```

```
    --namespace <my-kubernetes-namespace> \
    --labels <key=value>
```

<my-kubernetes-namespace> および <key=value> ラベルには、特定のワイルドカードを使用できます。詳細については、「the section called “Fargate プロファイルのワイルドカード”」を参照してください。

AWS マネジメントコンソール

で、Fargate プロファイルを作成するにはAWS マネジメントコンソール

1. Amazon EKS コンソールを開きます。
2. Fargate プロファイルを作成するクラスターを選択します。
3. [コンピューティング] タブを開きます。
4. [Fargate プロファイル] で、[Fargate プロファイルを追加] を選択します。
5. [Fargate プロファイルを設定] ページで、次の操作を行います。
   a. [名前] に Fargate プロファイルの名前を入力します。名前は一意である必要があります。
   b. [Pod 実行ロール] で、Fargate プロファイルで使用する Pod 実行ロールを選択します。eks-fargate-pods.amazonaws.com サービスプリンシパルを持つ IAM ロールのみが表示されます。ここにロールが表示されない場合は、ロールを作成する必要があります。詳細については、「the section called “ポッド実行IAMロール”」を参照してください。
   c. 選択した [サブネット] を必要に応じて変更します。

      > **Note**
      > Fargate で実行される Pod では、プライベートサブネットのみがサポートされます。

   d. [Tags] (タグ) では、オプションで Fargate プロファイルにタグを付けることができます。これらのタグは、Pod など、プロファイルに関連付けられた他のリソースには伝達されません。
   e. [次へ] を選択します。
6. [Pod の選択を設定] ページで、次の操作を行います。
   a. [名前空間] に、Pod と照合する名前空間を入力します。
      - kube-system または default など、特定の名前空間を使用して照合できます。
      - 特定のワイルドカード (例: prod-*) を使用して、複数の名前空間 (例: prod-deployment および prod-test) と照合することができます。詳細については、「the section called “Fargate プロファイルのワイルドカード”」を参照してください。

b. (オプション) セレクタに Kubernetes ラベルを追加します。特に、指定された名前空間内の Pod が一致する必要があるものにそれらを追加します。

- ラベル `infrastructure: fargate` をセレクターに追加して、`infrastructure: fargate` Kubernetes ラベルも持つ指定された名前空間の Pod のみがセレクターと一致するようにすることができます。
- 特定のワイルドカード (例: `key?: value?`) を使用して、複数の名前空間 (例: `keya: valuea` および `keyb: valueb`) と照合することができます。詳細については、「the section called “Fargate プロファイルのワイルドカード”」を参照してください。

c. [Next] (次へ) を選択します。

7. [確認と作成] ページで、Fargate プロファイルの情報を確認し、[作成] を選択します。

ステップ 4: CoreDNS を更新する

デフォルトでは、CoreDNS は Amazon EKS クラスターの Amazon EC2 インフラストラクチャで実行するように設定されています。クラスター内で Fargate でのみ Pod を実行する場合は、次の手順を実行します。

> **Note**
>
> `--fargate` オプションを使用して `eksctl` でクラスターを作成した場合は、the section called “次のステップ” にスキップできます。

1. 次のコマンドを使用して、CoreDNS の Fargate プロファイルを作成します。`<my-cluster>` をクラスター名、`<111122223333>` をアカウント ID、`<AmazonEKSFargatePodExecutionRole>` をポッド実行ロール名に置き換え、また、`<000000000000000a>`、`<000000000000000b>`、`<000000000000000c>` をプライベートサブネットの ID に置き換えます。Pod 実行ロールがない場合は、最初に作成する必要があります (「the section called “ステップ 2: Fargate Pod 実行ロールを作成する”」を参照)。

> **Important**
>
> ロール ARN に / 以外のパスを含めることはできません。例えば、ロールの名前が `development/apps/AmazonEKSFargatePodExecutionRole` の場合、ロールの ARN を指定するときに `AmazonEKSFargatePodExecutionRole` に変更する必

要があります。ロール ARN の形式は arn:aws:iam::<111122223333>:role/<AmazonEKSFargatePodExecutionRole> であることが必要です。

```
aws eks create-fargate-profile \
    --fargate-profile-name coredns \
    --cluster-name <my-cluster> \
    --pod-execution-role-arn arn:aws:iam::<111122223333>:role/
<AmazonEKSFargatePodExecutionRole> \
    --selectors namespace=kube-system,labels={k8s-app=kube-dns} \
    --subnets subnet-<000000000000000a> subnet-<000000000000000b> subnet-
<000000000000000c>
```

2. coredns デプロイのロールアウトをトリガーします。

```
kubectl rollout restart -n kube-system deployment coredns
```

次のステップ

- 以下のワークフローを使用すると、Fargate で実行するために既存のアプリケーションの移行を開始できます。
  a. アプリケーションの Kubernetes 名前空間と Kubernetes ラベルに一致する the section called “Fargate プロファイルの作成”。
  b. 既存のいずれかの Pod を削除および再作成し、Fargate でスケジューリングされるようにします。<namespace> と <deployment-type> を変更して、特定のポッドを更新します。

  ```
  kubectl rollout restart -n <namespace> deployment <deployment-type>
  ```

- the section called “アプリケーション負荷分散” をデプロイして、Fargate で実行されている Pod が Ingress オブジェクトを使用できるようにします。
- the section called “Vertical Pod Autoscaler” を使用して Fargate Pod の CPU とメモリの適切な初期サイズを設定し、the section called “Horizontal Pod Autoscaler” を使用してそれらの Pod をスケールできます。Vertical Pod Autoscaler で、より上位の CPU とメモリの組み合わせを持つ Fargate に Pod を自動的に再デプロイする場合は、Vertical Pod Autoscaler のモードを Auto または Recreate に設定します。これは、正しい機能を保証するためです。詳細については、GitHub で Vertical Pod Autoscaler のドキュメントを参照してください。

- これらの手順に従って、アプリケーションをモニタリングするための AWS Distro for OpenTelemetry (ADOT) コレクターをセットアップします。

起動時にどの Pod が AWS Fargate を使用するのかを定義する

クラスターの Fargate で Pod をスケジューリングする前に、起動時に Fargate を使用するPod を指定する Fargate プロファイルを少なくとも 1 つ定義する必要があります。

管理者は、Fargate プロファイルを使用してどの Pod が Fargate で実行されるかを宣言できます。これはプロファイルのセレクターを介して行うことができます。1 つのプロファイルに最大 5 つのセレクターを追加できます。各セレクターには名前空間が含まれている必要があります。セレクターにはラベルを含めることもできます。ラベルフィールドは、複数のオプションのキーと値のペアで構成されます。セレクターに一致するポッドは Fargate でスケジュールされます。ポッドは、セレクターで指定された名前空間とラベルを使用して照合されます。名前空間セレクターがラベルなしで定義されている場合、Amazon EKS は、プロファイルを使用して、その名前空間で実行されるすべての Pod を Fargate にスケジュールしようとします。スケジューリングされる Pod が Fargate プロファイル内のいずれかのセレクターと一致する場合、その Pod は Fargate でスケジューリングされます。

Pod が複数の Fargate プロファイルと一致する場合、次の Kubernetes ラベルを Pod 仕様に追加することで、Pod がどのプロファイルを使用するかを指定することができます: `eks.amazonaws.com/fargate-profile: my-fargate-profile`。Pod を Fargate にスケジューリングするには、そのプロファイル内のセレクターと一致する必要があります。Kubernetes アフィニティ/アンチアフィニティルールは適用されないため、Amazon EKS Fargate Pod では必要ありません。

Fargate プロファイルを作成する際は、Pod 実行ロールを指定する必要があります。この実行ロールは、プロファイルを使用して Fargate インフラストラクチャで実行される Amazon EKS コンポーネントのためのものです。承認のためにクラスターの Kubernetes ロールベースのアクセスコントロール (RBAC) に追加されます。これにより、Fargate インフラストラクチャで実行されている `kubelet` が Amazon EKS クラスターに登録され、クラスター内でノードとして表示されるようになります。Pod 実行ロールは、Amazon ECR イメージリポジトリへの読み取りアクセスを許可するために、Fargate インフラストラクチャへの IAM アクセス許可も提供します。詳細については、「the section called “ポッド実行IAMロール”」を参照してください。

Fargate プロファイルは変更できません。ただし、新しい更新されたプロファイルを作成して既存のプロファイルを置き換え、その後元のプロファイルを削除することはできます。

> **Note**
>
> Fargate プロファイルを使用して実行中の Pod は、プロファイルが削除されると停止され、保留状態になります。

クラスターのいずれかの Fargate プロファイルが `DELETING` ステータスである場合は、Fargate プロファイルの削除が完了するのを待ってから、そのクラスターに他のプロファイルを作成する必要があります。

> **Note**
>
> Fargate は現在 Kubernetes topologySpreadConstraints をサポートしていません。

Amazon EKS と Fargate は、Fargate プロファイルで定義されている各サブネットに Pod を分散させます。ただし、分散が偏ってしまう場合があります。均等な分散が必要な場合は、2 つの Fargate プロファイルを使用してください。2 つのレプリカをデプロイし、ダウンタイムを発生させたくないシナリオでは、均等な分散が重要になります。各プロファイルにはサブネットを 1 つだけ含めることをお勧めします。

Fargate プロファイルのコンポーネント

Fargate プロファイルには、次のコンポーネントが含まれています。

ポッド実行ロール

クラスターが AWS Fargate で Pod を作成するとき、Fargate インフラストラクチャ上で実行されている `kubelet` は、ユーザーに代わって AWS API を呼び出す必要があります。例えば、Amazon ECR からコンテナイメージを取得するためのコールを行う必要があります。Amazon EKS の Pod 実行ロールにより、これらを行うための IAM アクセス許可が付与されます。

Fargate プロファイルを作成する際は、Pod で使用する Pod 実行ロールを指定する必要があります。このロールは、承認のためにクラスターの Kubernetes ロールベースのアクセスコントロール (RBAC) に追加されます。これにより、Fargate インフラストラクチャで実行されている `kubelet` が Amazon EKS クラスターに登録され、クラスター内でノードとして表示されるようになります。詳細については、「the section called “ポッド実行IAMロール”」を参照してください。

サブネット

このプロファイルを使用する Pod を起動するサブネットの ID。現時点では、Fargate で実行されている Pod にはパブリック IP アドレスが割り当てられていません。したがって、このパラメータには、プライベートサブネット (インターネットゲートウェイへの直接ルートなし) のみが受け入れられます。

セレクター

この Fargate プロファイルを使用するために Pod に一致するセレクター。1 つの Fargate プロファイルに最大 5 つのセレクターを指定できます。セレクターには次のコンポーネントがあります:

- 名前空間 – セレクターの名前空間を指定する必要があります。セレクターはこの名前空間で作成された Pod のみを照合します。ただし、複数のセレクターを作成して複数の名前空間をターゲットにすることはできます。
- ラベル – オプションで、セレクターに一致する Kubernetes ラベルを指定できます。セレクターは、セレクターで指定されているすべてのラベルを持つ Pod のみに一致します。

Fargate プロファイルのワイルドカード

名前空間、ラベルキー、およびラベル値のセレクター基準では、Kubernetes で許可されている文字に加えて、* および ? の使用が許可されています。

- * は、文字なし、1 文字、または複数の文字を表します。例: `prod*` は `prod` および `prod-metrics` を表します。
- ? は 1 文字を表します (例: `value?` は `valuea` を表します)。しかし、`value` および `value-a` を表すことはできません。? は厳密に 1 文字だけを表すためです。

これらのワイルドカード文字は、任意の位置や組み合わせで使用できます (例: `prod*`、`*dev`、および `frontend*?`)。正規表現など、他のワイルドカードやパターンマッチング形式はサポートされていません。

Pod 仕様内の名前空間とラベルに一致するプロファイルが複数ある場合、Fargate はプロファイル名による英数字ソートに基づいてプロファイルを選択します。例えば、(`beta-workload` という名前の) プロファイル A と (`prod-workload` という名前の) プロファイル B の両方に、起動する Pod に一致するセレクターがある場合、Fargate は Pod にプロファイル A (`beta-workload`) を選択します。Pod には、プロファイル A のラベルが付いています (例: `eks.amazonaws.com/fargate-profile=beta-workload`)。

ワイルドカードを使用する新しいプロファイルに既存の Fargate Pod を移行するには、次の 2 つの方法があります。

- 一致するセレクターを持つ新しいプロファイルを作成し、古いプロファイルを削除します。古いプロファイルでラベル付けされたポッドは、一致する新しいプロファイルに再スケジュールされます。
- ワークロードを移行したいものの、各 Fargate Pod にどの Fargate ラベルが付いているかわからない場合は、次の方法を使用できます。新しいプロファイルを作成し、同じクラスター上のプロファイルの中でアルファベット順で最初に並べられる名前を付けます。次に、新しいプロファイルに移行する必要がある Fargate Pod をリサイクルします。

Fargate プロファイルの作成

このセクションでは、Fargate プロファイルを作成する方法について説明します。また、Fargate プロファイルに使用する Pod 実行ロールを作成しておく必要があります。詳細については、「the section called “ポッド実行IAMロール”」を参照してください。Fargate で実行されている Pod は、プライベートサブネットでのみサポートされています (AWS サービスへの NAT ゲートウェイアクセスはできますが、インターネットゲートウェイへの直接ルーティングはできません)。これは、クラスターの VPC がプライベートサブネットを使用できるようにするためです。

プロファイルは以下を使用して作成できます。

- the section called “eksctl”
- the section called “AWS マネジメントコンソール”

eksctl

eksctl で、Fargate プロファイルを作成するには

以下の eksctl コマンドで Fargate プロファイルを作成し、すべてのサンプル値を自分の値に置き換えます。名前空間を指定する必要があります。ただし、--labels オプションは必須ではありません。

```
eksctl create fargateprofile \
    --cluster my-cluster \
    --name my-fargate-profile \
    --namespace my-kubernetes-namespace \
    --labels key=value
```

my-kubernetes-namespace および key=value ラベルには、特定のワイルドカードを使用できます。詳細については、「the section called “Fargate プロファイルのワイルドカード”」を参照してください。

AWS マネジメントコンソール

AWS マネジメントコンソール で、Fargate プロファイルを作成するには

1. アマゾン EKS コンソールを開きます。
2. Fargate プロファイルを作成するクラスターを選択します。
3. [コンピューティング] タブを開きます。
4. [Fargate プロファイル] で、[Fargate プロファイルを追加] を選択します。
5. [Fargate プロファイルを設定] ページで、次の操作を行います。
   a. [名前] に、Fargate プロファイルの一意の名前 (my-profile など) を入力します。
   b. [Pod 実行ロール] で、Fargate プロファイルで使用する Pod 実行ロールを選択します。eks-fargate-pods.amazonaws.com サービスプリンシパルを持つ IAM ロールのみが表示されます。ここにロールが表示されない場合は、ロールを作成する必要があります。詳細については、「the section called “ポッド実行IAMロール”」を参照してください。
   c. 選択した [サブネット] を必要に応じて変更します。

      > **Note**
      >
      > Fargate で実行される Pod では、プライベートサブネットのみがサポートされます。

   d. [Tags] (タグ) では、オプションで Fargate プロファイルにタグを付けることができます。これらのタグは、Pod など、プロファイルに関連付けられた他のリソースには伝達されません。
   e. [Next] を選択します。
6. [Pod の選択を設定] ページで、次の操作を行います。
   a. [名前空間] に、Pod と照合する名前空間を入力します。
      - kube-system または default など、特定の名前空間を使用して照合できます。
      - 特定のワイルドカード (例: prod-*) を使用して、複数の名前空間 (例: prod-deployment および prod-test) と照合することができます。詳細については、「the section called “Fargate プロファイルのワイルドカード”」を参照してください。
   b. (オプション) セレクタに Kubernetes ラベルを追加します。特に、指定された名前空間内の Pod が一致する必要があるものにそれらを追加します。

- ラベル `infrastructure: fargate` をセレクターに追加して、`infrastructure: fargate` Kubernetes ラベルも持つ指定された名前空間の Pod のみがセレクターと一致するようにすることができます。
- 特定のワイルドカード (例: `key?: value?`) を使用して、複数の名前空間 (例: `keya: valuea` および `keyb: valueb`) と照合することができます。詳細については、「the section called “Fargate プロファイルのワイルドカード”」を参照してください。

c. [Next] を選択します。

7. [確認と作成] ページで、Fargate プロファイルの情報を確認し、[作成] を選択します。

Fargate プロファイルを削除する

このトピックでは、Fargate プロファイルを削除する方法について説明します。Fargate プロファイルを削除すると、そのプロファイルで Fargate にスケジューリングされていた Pod はすべて削除されます。これらの Pod が別の Fargate プロファイルと一致する場合、それらの Pod はそのプロファイルで Fargate にスケジューリングされます。Fargate プロファイルがどのプロファイルとも一致しなくなった場合は、Fargate へのスケジューリングは行われず、保留中のままになる可能性があります。

一度にクラスター内の 1 つの Fargate プロファイルのみが、`DELETING` ステータスになることができます。そのクラスター内の他のプロファイルを削除する前に、Fargate プロファイルの削除が完了するまでお待ちください。

プロファイルは、次のいずれかのツールを使用して削除できます。

- the section called “`eksctl`”
- the section called “AWS マネジメントコンソール”
- the section called “AWS CLI”

`eksctl`

`eksctl` で、Fargate プロファイルを削除する

クラスターからプロファイルを削除するには、次のコマンドを使用します。各`#####`は独自の値に置き換えます。

```
eksctl delete fargateprofile  --name my-profile --cluster my-cluster
```

AWS マネジメントコンソール

AWS マネジメントコンソール で、Fargate プロファイルを削除する

1. アマゾン EKS コンソールを開きます。
2. 左のナビゲーションペインで [Clusters] (クラスター) を選択してください。クラスターの一覧で Fargate プロファイルを削除するクラスターを選択します。
3. [コンピューティング] タブを開きます。
4. 削除する Fargate プロファイルを選択し、[削除] を選択します。
5. [Fargate プロファイルの削除] ページで、プロファイルの名前を入力し、[削除] を選択します。

AWS CLI

AWS CLI で、Fargate プロファイルを削除する

クラスターからプロファイルを削除するには、次のコマンドを使用します。各#####は独自の値に置き換えます。

```
aws eks delete-fargate-profile --fargate-profile-name my-profile --cluster-name my-cluster
```

Fargate Pod 設定の詳細を理解する

このセクションでは、AWS Fargate で Kubernetes Pod を実行するための固有の Pod 設定の詳細について説明します。

ポッド CPU とメモリ

Kubernetes では、Pod 内の各コンテナに割り当てられるリクエスト、最小量の vCPU およびメモリリソースを定義できます。Pod は Kubernetes によってスケジュールされ、少なくとも各 Pod に対してリクエストされたリソースがコンピューティングリソース上で利用可能になるようにします。詳細については、Kubernetes のドキュメントの「Managing Compute Resources for Containers」を参照してください。

> **Note**
>
> Amazon EKS Fargate はノードごとに Pod を 1 つだけ実行するため、リソースが少ない場合に Pod を削除するシナリオは発生しません。すべての Amazon EKS Fargate Pod は保証さ

れた優先度で実行されるため、リクエストされた CPU とメモリは、すべてのコンテナの制限に等しくする必要があります。詳細については、Kubernetes ドキュメントの「Configure Quality of Service for Pods (ポッド用にサービスの品質を設定する)」を参照してください。

Pod が Fargate にスケジュールされている場合、Pod 仕様内の vCPU とメモリの予約によって、Pod にプロビジョニングする CPU とメモリの量が決まります。

- Init コンテナからの最大リクエストは、Init リクエスト vCPU およびメモリ要件を決定するために使用されます。
- 実行時間の長いコンテナのリクエストは、実行時間の長いリクエスト vCPU とメモリ要件を決定するために合計されます。
- 前記の 2 つの値のうち大きい方が、Pod 用に使用する vCPU とメモリリクエストに対して選択されます。
- Fargate は、必要な Kubernetes コンポーネント (`kubelet`、`kube-proxy`、および `containerd`) の各 Pod のメモリ予約に 256 MB を追加します。

Fargate では、Pod の実行に必要なリソースを常に確保するために、vCPU とメモリのリクエストの合計に最も近い、次に示すコンピューティング設定に切り上げられます。

vCPU とメモリの組み合わせを指定しない場合は、使用可能な最小の組み合わせ (.25 vCPU と 0.5 GB のメモリ) が使用されます。

次の表は、Fargate で実行されている Pod で使用可能な vCPU とメモリの組み合わせを示しています。

vCPU 値	メモリの値
.25 vCPU	0.5 GB、1 GB、2 GB
.5 vCPU	1 GB、2 GB、3 GB、4 GB
1 vCPU	2 GB、3 GB、4 GB、5 GB、6 GB、7 GB、8 GB
2 vCPU	4 GB ~ 16 GB (1 GB のインクリメント)
4 vCPU	8 GB ~ 30 GB (1 GB のインクリメント)

vCPU 値	メモリの値
8 vCPU	16 GB ~ 60 GB (4 GB のインクリメント)
16 vCPU	32 GB ~ 120 GB (8 GB のインクリメント)

Kubernetes コンポーネント用に予約されている追加メモリにより、要求よりも多くの vCPU による Fargate タスクがプロビジョニングされる可能性があります。例えば、1 つの vCPU と 8 GB のメモリを要求すると、メモリ要求に 256 MB が追加され、2 つの vCPU と 9 GB のメモリの Fargate タスクがプロビジョニングされます。これは、1 つの vCPU と 9 GB のメモリのタスクがないためです。

Fargate で実行されている Pod のサイズと、Kubernetes が `kubectl get nodes` でレポートするノードサイズに相関はありません。レポートされるノードサイズは、多くの場合 Pod のキャパシティーよりも大きくなります。次のコマンドを使用して、Pod キャパシティーを確認できます。*`default`* を Pod の名前空間に、*`pod-name`* を Pod の名前に置き換えます。

```
kubectl describe pod --namespace default pod-name
```

出力例は次のとおりです。

```
[...]
annotations:
    CapacityProvisioned: 0.25vCPU 0.5GB
[...]
```

`CapacityProvisioned` アノテーションは、実際の Pod キャパシティーを表し、これで Fargate で実行されている Pod のコストが決まります。コンピューティング設定の料金情報については、「AWS Fargate の料金」を参照してください。

Fargate ストレージ

Fargate 上で実行される Pod は、ドライバーの手動インストールステップなしで、Amazon EFS ファイルシステムを自動的にマウントします。Fargate ノードでは動的永続ボリュームプロビジョニングを使用できませんが、静的プロビジョニングを使用することはできます。詳細については、GitHub の「Amazon EFS CSI ドライバー」を参照してください。

プロビジョニングすると、Fargate で実行されている各 Pod に対して、デフォルトで 20 GiB のエフェメラルストレージが割り当てられます。このタイプのストレージは、Pod の停止後に削除されます。Fargate に起動される新しい Pod では、エフェメラルストレージボリュームの暗号化がデ

フォルトで有効になっています。エフェメラル Pod ストレージは、AWS Fargate で管理されるキーを使用して AES-256 暗号化アルゴリズムで暗号化されます。

> **Note**
>
> Fargate で実行される Amazon EKS Pod のデフォルトで使用可能なストレージは、20 GiB 未満です。これは、一部のスペースが `kubelet` と Pod 内に読み込まれるその他の Kubernetes モジュールによって使用されるためです。

エフェメラルストレージの総量は、最大 175 GiB まで増やすことができます。Kubernetes でサイズを設定するには、Pod 内の各コンテナに対して、`ephemeral-storage` リソースのリクエストを指定します。Kubernetes が Pod をスケジュールすると、各 Pod に対するリソースリクエストの合計が Fargate タスクの容量を下回ることが保証されます。詳細については、Kubernetes ドキュメントの「Resource Management for Pods and Containers」を参照してください。

Amazon EKS Fargate は、システム使用目的でリクエストされた数よりも大きなエフェメラルストレージをプロビジョニングします。たとえば、100 GiB のリクエストでは、115 GiB のエフェメラルストレージの Fargate タスクがプロビジョニングされます。

AWS Fargate OS パッチ適用イベントのアクションを設定する

Amazon EKS は定期的に AWS Fargate ノードの OS にパッチを適用し、ノードを安全に保ちます。パッチ適用プロセスの一環として、ノードをリサイクルして OS パッチをインストールします。更新は、サービスへの影響を最小限に抑える方法で試行されます。ただし、Pod のエビクションが正常に行われない場合は、Pod を削除する必要がある場合があります。以下に示しているのは、中断の可能性を最小限に抑えるために実行できるアクションです。

- 適切な Pod の中断予算 (PDB) を設定して、同時にダウンする Pod の数を制御します。
- Pod が削除される前に、失敗したエビクションに対応する Amazon EventBridge ルールを作成します。
- 受信した通知に記載されているエビクション日よりも前に、影響を受けるポッドを手動で再起動します。
- AWS User Notifications で通知設定を作成します。

Amazon EKS は、Kubernetes コミュニティと緊密に協力して、できるだけ早く、バグ修正とセキュリティパッチが入手可能になるようにしています。すべての Fargate Pod は最新の Kubernetes パッ

チバージョンで起動されます。このバージョンは、Amazon EKS からクラスターの Kubernetes バージョンとして入手できます。以前のパッチバージョンの Pod をお持ちの場合、Amazon EKS はそれをリサイクルして最新バージョンに更新することがあります。これにより、Pod に最新のセキュリティアップデートが確実に装備されます。そのため、クリティカルな共通脆弱性識別子 (CVE) 問題があっても、最新の状態に保たれてセキュリティリスクが軽減されています。

AWS Fargate OS が更新されると、Amazon EKS は影響を受けるリソースと今後のポッドのエビクション日を含む通知を送信します。指定されたエビクション日が不都合な場合は、通知に記載されたエビクション日よりも前に、影響を受けるポッドを手動で再起動するオプションを利用できます。通知を受け取る前に作成されたポッドは、エビクションの対象となります。ポッドを手動で再起動する方法の詳細については、Kubernetes ドキュメントを参照してください。

Pod がリサイクルされるときに一度にダウンする Pod の数を制限するために、Pod の中断予算 (PDB) を設定できます。PDB を使用して、更新の実行を許可しながら、各アプリケーションの要件に基づいて最小限使用可能なポッドを定義できます。PDB の最小可用性は 100% 未満である必要があります。詳細については、Kubernetes ドキュメントの「アプリケーションに Disruption Budget 指定する」を参照してください。

Amazon EKS は Eviction API を使用して、アプリケーションに設定した PDB を尊重しながら Pod を安全に排出します。ポッドのエビクションは、影響を最小限に抑えるために、アベイラビリティーゾーンごとに行われます。エビクションが成功すると、新しい Pod には最新のパッチが適用され、それ以上のアクションは必要ありません。

Pod のエビクションが失敗すると、Amazon EKS はエビクションに失敗した Pod の詳細を含むイベントをアカウントに送信します。スケジュールされた終了時刻より前にメッセージに対応できます。具体的な時刻は、パッチの緊急性によって異なります。その時刻になると、Amazon EKS は Pod のエビクションを再び試行します。ただし、今回は、エビクションが失敗しても新しいイベントは送信されません。エビクションが再び失敗すると、新しい Pod に最新のパッチが適用できるように、既存の Pod が定期的に削除されます。

Pod のエビクションが失敗したときに受信されるイベントのサンプルを次に示します。これには、クラスター、Pod 名、Pod 名前空間、Fargate プロファイル、およびスケジュールされた終了時刻に関する詳細が含まれます。

```
{
    "version": "0",
    "id": "12345678-90ab-cdef-0123-4567890abcde",
    "detail-type": "EKS Fargate Pod Scheduled Termination",
    "source": "aws.eks",
```

```
    "account": "111122223333",
    "time": "2021-06-27T12:52:44Z",
    "region": "region-code",
    "resources": [
        "default/my-database-deployment"
    ],
    "detail": {
        "clusterName": "my-cluster",
        "fargateProfileName": "my-fargate-profile",
        "podName": "my-pod-name",
        "podNamespace": "default",
        "evictErrorMessage": "Cannot evict pod as it would violate the pod's disruption
 budget",
        "scheduledTerminationTime": "2021-06-30T12:52:44.832Z[UTC]"
    }
}
```

さらに、Pod に複数の PDB が関連付けられていると、エビクション失敗イベントが発生する可能性があります。このイベントは次のエラーメッセージを返します。

```
"evictErrorMessage": "This pod has multiple PodDisruptionBudget, which the eviction
 subresource does not support",
```

このイベントに基づいて、目的のアクションを作成できます。例えば、Pod の中断予算 (PDB) を調整して、Pod のエビクションの方法を制御できます。具体的には、利用可能な Pod の目標パーセンテージを指定する PDB から始めたとします。アップグレード中に Pod が強制終了される前に、PDB を異なる割合の Pod に調整できます。このイベントを受信するには、クラスターが属する AWS アカウントおよび AWS リージョンで Amazon EventBridge ルールを作成する必要があります。ルールでは、次のカスタムパターンを使用する必要があります。EventBridge ルールの作成の詳細については、Amazon EventBridge ユーザーガイドの「イベントに反応する Amazon EventBridge ルールの作成」を参照してください 。

```
{
  "source": ["aws.eks"],
  "detail-type": ["EKS Fargate Pod Scheduled Termination"]
}
```

キャプチャするためのイベントに適切なターゲットを設定できます。詳細については、「Amazon EventBridge ユーザーガイド」の「Amazon EventBridge ターゲット」を参照してください。AWS User Notifications で通知設定を作成することもできます。AWS マネジメントコンソー

ル を使用して通知を作成する場合は、[イベントルール] で、 [AWS 名] に [Elastic Kubernetes Service (EKS)] を選択し、[イベントタイプ] に [EKS Fargate Pod のスケジュールされた終了時刻] を選択します。詳細については、「AWS User Notifications ユーザーガイド」の「AWS User Notifications の開始方法」を参照してください。

EKS Pod エビクションに関するよくある質問については、AWS re:Post の「よくある質問: Fargate Pod のエビクション通知」を参照してください。

AWS Fargate アプリケーションと使用状況のメトリクスを収集する

AWS Fargate のシステムメトリクスおよび CloudWatch 使用状況メトリクスを収集できます。

アプリケーションメトリクス

Amazon EKS や AWS Fargate で動作するアプリケーションには、AWS Distro for OpenTelemetry (ADOT) を使用できます。ADOT では、システムメトリクスを収集し、CloudWatch コンテナインサイトダッシュボードに送信できます。Fargate で実行されているアプリケーションで ADOT の使用を開始するには、「ADOT ドキュメント」の「Using CloudWatch Container Insights with AWS Distro for OpenTelemetry」(Distro for OpenTelemetry を活用した CloudWatch Container Insights の使用) を参照してください。

使用状況メトリクス

CloudWatch 使用状況メトリクスを使用して、アカウントのリソースの使用状況を把握できます。これらのメトリクスを使用して、CloudWatch グラフやダッシュボードで現在のサービスの使用状況を可視化できます。

AWS Fargate 使用状況メトリクスは、AWS Service Quotas に対応しています。使用量がサービスクォータに近づいたときに警告するアラームを設定することもできます。Fargate のサービスのクォータの詳細については、「the section called “Service Quotas”」を参照してください。

AWS Fargate は、 `AWS/Usage` 名前空間に以下のメトリクスを公開します。

メトリクス	説明
`ResourceCount`	アカウントで実行されている指定されたリソースの合計数。リソースは、メトリクスに関連付けられたディメンションによって定義されます。

次のディメンションは、AWS Fargate によって公開される使用状況メトリクスを絞り込むために使用されます。

ディメンション	説明
`Service`	リソースを含む AWS のサービスの名前。AWS Fargate 使用状況メトリクスの場合、このディメンションの値は `Fargate` です。
`Type`	報告されるエンティティのタイプ。現在、AWS Fargate 使用状況メトリクスの有効な値は `Resource` のみです。
`Resource`	実行中のリソースのタイプ。 現在、AWS Fargate は Fargate のオンデマンド 使用状況に関する情報を返します。Fargate のオンデマンド使用状況のリソース値は `OnDemand` です。 [注意] ==== Fargate のオンデマンド使用状況とは、Fargate を使用した Amazon EKS Pod、Fargate 起動タイプを使用した Amazon ECS タスク、`FARGATE` キャパシティープロバイダーを使用した Amazon ECS タスクを組み合わせたものです。 ====
`Class`	追跡されているリソースのクラス。現在、AWS Fargate はクラスディメンションを使用していません。

Fargate のリソース使用状況メトリクスをモニタリングするための CloudWatch アラームの作成

AWS Fargate は、Fargate オンデマンドリソース使用状況の AWS Service Quotas に対応する CloudWatch 使用状況メトリクスを提供します。Service Quotas コンソールでは、使用状況をグラフ

で可視化できます。また、使用量が Service Quotas に近づいたときに警告するアラームも設定できます。詳細については、「the section called “メトリクスを収集する”」を参照してください。

以下の手順を使用して、Fargate リソース使用状況メトリクスに基づく CloudWatch アラームを作成します。

1. Service Quotas コンソールを開きますhttps://console.aws.amazon.com/servicequotas/
2. 左のナビゲーションペインで [AWS サービス] を選択します。
3. [AWS サービス] リストから、[AWS Fargate] を探して選択します。
4. [Service quotas] (サービスクォータ) リストで、アラームを作成する Fargate 使用量クォータを選択します。
5. Amazon CloudWatch アラームのセクションで [Create] (作成) を選択します。
6. [アラームのしきい値] で、適用されたクォータ値からアラーム値として設定する値の割合を選択します。
7. [アラーム名] にアラームの名前を入力し、[Create (作成)] を選択します。

クラスターの AWS Fargate ログ記録を開始する

Fargate の Amazon EKS では、Fluent Bit をベースにした組み込みのログルーターが利用できます。Fluent Bit コンテナをサイドカーとして明示的に実行する必要はなく、この実行は Amazon によって行われます。必要となるのは、ログルーターの設定だけです。設定は専用の `ConfigMap` を介して行い、その際は以下の基準を満たす必要があります。

- 名前のついた `aws-logging`
- `aws-observability` と呼ばれる専用の名前空間での作成
- 5,300 文字以内にしてください。

`ConfigMap` を作成すると、Fargate の Amazon EKS は自動的にそれを検出しログルーターの設定を行います。Fargate は、Fluent Bit の AWS のバージョンを使用しています。これは、AWS によって管理される Fluent Bit の上流対応のディストリビューションです。詳細については、GitHub の「AWS for Fluent Bit」を参照してください。

ログルーターを使用すると、AWS のさまざまなサービスをログの分析と保管に使用できます。Fargate からは、Amazon CloudWatch および Amazon OpenSearch Service にログを直接ストリーミングできます。また、Amazon Data Firehose 経由で Amazon S3、Amazon Kinesis Data Streams、およびパートナーツールなどの送信先にログをストリーミングすることも可能です。

- Fargate Pod をデプロイする既存の Kubernetes 名前空間を指定する既存の Fargate プロファイル。詳細については、「the section called “ステップ 3: クラスターの Fargate プロファイルを作成する”」を参照してください。
- 既存の Fargate Pod 実行ロール。詳細については、「the section called “ステップ 2: Fargate Pod 実行ロールを作成する”」を参照してください。

ログルーターの設定

> **Important**
>
> ログを正常に公開するには、そのクラスターが属している VPC からログの送信先までの、ネットワークアクセスが必要になります。これには、ユーザーによる VPC の出力ルールの、カスタマイズが関係します。詳細については、Amazon CloudWatch Logs ユーザーガイドの「Using CloudWatch Logs with interface VPC endpoints」を参照してください。

以下のステップでは、すべての`#####`を独自の値に置き換えます。

1. `aws-observability` という名前で、専用の Kubernetes 名前空間を作成します。
   a. 次の内容をコンピュータ上の `aws-observability-namespace.yaml` という名前のファイルに保存します。`name` の値は `aws-observability` である必要があり、`aws-observability: enabled` ラベルが必須です。

   ```
   kind: Namespace
   apiVersion: v1
   metadata:
     name: aws-observability
     labels:
       aws-observability: enabled
   ```

   b. 名前空間を作成します。

   ```
   kubectl apply -f aws-observability-namespace.yaml
   ```

2. データ値 `Fluent Conf` を使用して `ConfigMap` を作成し、コンテナログを送信先に送ります。Fluent Conf とは Fluent Bit であり、コンテナログを任意のログ送信先にルーティングするために使用される、高速で軽量なログプロセッサ構成言語です。詳細については、Fluent Bit ドキュメントの「Configuration File」を参照してください。

> **Important**
>
> 典型的な Fluent Conf に含まれている主なセクションは、ServiceInput、Filter および Output です。ただし、Fargate のログルータでは、以下だけを受け入れます。
>
> - Filter および Output セクション。
> - Parser セクション。
>
> 他のセクションを提供した場合、拒否されます。

Fargate ログルーターは Service および Input セクションを管理します。次の Input セクションがありますが、これらは変更できず、ConfigMap には必要ありません。ただし、メモリバッファー制限やログに適用されるタグなどの洞察は得られます。

```
[INPUT]
    Name tail
    Buffer_Max_Size 66KB
    DB /var/log/flb_kube.db
    Mem_Buf_Limit 45MB
    Path /var/log/containers/*.log
    Read_From_Head On
    Refresh_Interval 10
    Rotate_Wait 30
    Skip_Long_Lines On
    Tag kube.*
```

ConfigMap の作成時は、以下の (Fargate がフィールドの検証に使用する) ルールを考慮に入れます。

- [FILTER]、[OUTPUT]および [PARSER] は、それぞれが対応するキーにより指定する必要があります。例: [FILTER] が filters.conf の下にある必要があります。filters.conf には、複数の [FILTER] を含められます。また、[OUTPUT] および [PARSER] セクションは、それぞれと対応するキーの下に置く必要があります。複数の [OUTPUT] セクションを指定することで、ログを異なる送信先に同時にルーティングできます。
- Fargate は各セクションに必要なキーを検証します。Name および match がそれぞれの [FILTER] および [OUTPUT] に必要です。Name および format がそれぞれの [PARSER] に必要です。キーの大文字と小文字は区別されません。
- ${ENV_VAR} などの環境変数は ConfigMap では許可されていません。

- インデントは、それぞれの `filters.conf`、`output.conf`、および `parsers.conf` の中で、ディレクティブまたはキーと値のペアで同じである必要があります。キーと値のペアは、ディレクティブよりも深いインデントにする必要があります。
- Fargate は、サポートされている次のフィルターに対して検証します。`grep`、`parser`、`record_modifier`、`rewrite_tag`、`throttle`、`nest`、`modify`、および `kubernetes`。
- Fargate は、サポートされている次の出力に対して検証します。`es`、`firehose`、`kinesis_firehose`、`cloudwatch`、`cloudwatch_logs`、および `kinesis`。
- ログ記録を有効にするには、サポートされている `Output` プラグインが少なくとも 1 つ `ConfigMap` にあることが必要です。`Filter` および `Parser` は、ログ記録を有効にするために必要ありません。

また、希望の設定を使用して Amazon EC2 で Fluent Bit を実行し、検証によって発生する問題をトラブルシューティングすることもできます。以下のいずれかの例に従って、`ConfigMap` を作成します。

> **Important**
>
> Amazon EKS Fargate のログ記録では、`ConfigMap` での動的設定をサポートしていません。`ConfigMap` に対する任意の変更は、新しい Pod に対してのみ適用されます。既存の Pod には、これらの変更は適用されません。

例を使用して、必要なログ送信先用に `ConfigMap` を作成します。

> **Note**
>
> また、Amazon Kinesis Data Streams を ログの宛先として使用できます。Kinesis Data Streams を使用する場合は、ポッド実行ロールに `kinesis:PutRecords` 権限が付与されていることを確認してください。詳細については、Fluent Bit: 公式マニュアルの Amazon Kinesis Data Streams の「許可」を参照してください。

Example

CloudWatch

CloudWatch を使用する場合、次の 2 つの出力オプションがあります。

- C で記述された出力プラグイン
- Golang で記述された出力プラグイン

次の例は、cloudwatch_logs プラグインを使用して CloudWatch にログを送信する方法を示しています。

a. 次の内容を aws-logging-cloudwatch-configmap.yaml という名前のファイルに保存します。*region-code* を、クラスターのある AWS リージョンに置き換えます。[OUTPUT] のパラメータは必須です。

```
kind: ConfigMap
apiVersion: v1
metadata:
  name: aws-logging
  namespace: aws-observability
data:
  flb_log_cw: "false"  # Set to true to ship Fluent Bit process logs to
 CloudWatch.
  filters.conf: |
    [FILTER]
        Name parser
        Match *
        Key_name log
        Parser crio
    [FILTER]
        Name kubernetes
        Match kube.*
        Merge_Log On
        Keep_Log Off
        Buffer_Size 0
        Kube_Meta_Cache_TTL 300s
  output.conf: |
    [OUTPUT]
        Name cloudwatch_logs
        Match   kube.*
        region region-code
        log_group_name my-logs
```

```
        log_stream_prefix from-fluent-bit-
        log_retention_days 60
        auto_create_group true
  parsers.conf: |
    [PARSER]
        Name crio
        Format Regex
        Regex ^(?<time>[^ ]+) (?<stream>stdout|stderr) (?<logtag>P|F) (?
<log>.*)$
        Time_Key    time
        Time_Format %Y-%m-%dT%H:%M:%S.%L%z
```

b. マニフェストをクラスターに適用します。

```
kubectl apply -f aws-logging-cloudwatch-configmap.yaml
```

Amazon OpenSearch Service

ログを Amazon OpenSearch Service に送信したい場合、C で書かれたプラグインである、es 出力を使用することができます。以下の例では、OpenSearch にログを送信するためにプラグインを使用する方法について説明します。

a. 次の内容を `aws-logging-opensearch-configmap.yaml` という名前のファイルに保存します。各*#####*は独自の値に置き換えます。

```
kind: ConfigMap
apiVersion: v1
metadata:
  name: aws-logging
  namespace: aws-observability
data:
  output.conf: |
    [OUTPUT]
      Name  es
      Match *
      Host  search-example-gjxdcilagiprbglqn42jsty66y.region-
code.es.amazonaws.com
      Port  443
      Index example
      Type  example_type
      AWS_Auth On
      AWS_Region region-code
      tls   On
```

b. マニフェストをクラスターに適用します。

```
kubectl apply -f aws-logging-opensearch-configmap.yaml
```

Firehose

Firehose にログを送信する場合、次の二つの出力オプションがあります。

- kinesis_firehose – C 言語で記述された出力プラグイン。
- firehose – Go 言語で記述された出力プラグイン。

次の例は、Firehose にログを送信するために kinesis_firehose プラグインを使用する方法を示しています。

a. 次の内容を aws-logging-firehose-configmap.yaml という名前のファイルに保存します。*region-code* を、クラスターのある AWS リージョンに置き換えます。

```
kind: ConfigMap
apiVersion: v1
metadata:
  name: aws-logging
  namespace: aws-observability
data:
  output.conf: |
    [OUTPUT]
     Name  kinesis_firehose
     Match *
     region region-code
     delivery_stream my-stream-firehose
```

b. マニフェストをクラスターに適用します。

```
kubectl apply -f aws-logging-firehose-configmap.yaml
```

3. 目的の宛先にログを送信できるように Fargate Pod 実行ロールに対するアクセス許可を設定します。

a. 送信先の IAM ポリシーをコンピュータにダウンロードします。

Example

CloudWatch

CloudWatch IAM ポリシーをコンピュータにダウンロードします。GitHub でポリシーの表示をすることもできます。

```
curl -O https://raw.githubusercontent.com/aws-samples/amazon-eks-fluent-
logging-examples/mainline/examples/fargate/cloudwatchlogs/permissions.json
```

Amazon OpenSearch Service

OpenSearch IAM ポリシーをコンピュータにダウンロードします。GitHub でポリシーの表示をすることもできます。

```
curl -O https://raw.githubusercontent.com/aws-samples/amazon-eks-
fluent-logging-examples/mainline/examples/fargate/amazon-elasticsearch/
permissions.json
```

OpenSearch Dashboards のアクセスコントロールが適切に設定されていることを確認します。OpenSearch Dashboards の `all_access role` には、Fargate Pod 実行ロールと IAM ロールがマッピングされている必要があります。同様のマッピングが、`security_manager` ロールに対しても必要です。以前のマッピングを追加するには、`Menu`、`Security`、`Roles` の順にクリックした後、それぞれに対応するロールを選択します。詳細については、「CloudWatch Logs が Amazon ES ドメインにストリーミングされるようにトラブルシューティングする方法を教えてください。」を参照してください。

Firehose

Firehose IAM ポリシーをコンピュータにダウンロードします。GitHub でポリシーの表示をすることもできます。

```
curl -O https://raw.githubusercontent.com/aws-samples/amazon-eks-fluent-
logging-examples/mainline/examples/fargate/kinesis-firehose/permissions.json
```

b. ダウンロードしたポリシーファイルから IAM ポリシーを作成します。

```
aws iam create-policy --policy-name eks-fargate-logging-policy --policy-document
 file://permissions.json
```

c. 次のコマンドを使用して、Fargate プロファイルに指定されたポッド実行ロールに IAM ポリシーを添付します。*111122223333* は、ご自分のアカウント ID に置き換えます。*AmazonEKSFargatePodExecutionRole* を Pod 実行ロールに置き換えます (詳細については、「the section called “ステップ 2: Fargate Pod 実行ロールを作成する”」を参照してください)。

```
aws iam attach-role-policy \
  --policy-arn arn:aws:iam::111122223333:policy/eks-fargate-logging-policy \
  --role-name AmazonEKSFargatePodExecutionRole
```

Kubernetes フィルターのサポート

Fluent Bit Kubernetes フィルターを使用すると、Kubernetes メタデータをログファイルに追加できます。フィルターの詳細については、Fluent Bit ドキュメントの Kubernetes を参照してください。API サーバーエンドポイントを使用して、フィルターを適用できます。

```
filters.conf: |
    [FILTER]
        Name                kubernetes
        Match               kube.*
        Merge_Log           On
        Buffer_Size         0
        Kube_Meta_Cache_TTL 300s
```

Important

- `Kube_URL`、`Kube_CA_File`、`Kube_Token_Command`、および `Kube_Token_File` はサービス所有の設定パラメータであるため、指定しないでください。Amazon EKS Fargate がこれらの値を設定します。
- `Kube_Meta_Cache_TTL` は、Fluent Bit が最新のメタデータを取得するために API サーバーと通信するまで待機する時間です。`Kube_Meta_Cache_TTL` が指定されていない場合、Amazon EKS Fargate は API サーバーの負荷を軽減するためにデフォルト値である 30 分を追加します。

Fluent Bit プロセスログをアカウントに送付するには

必要に応じて、Fluent Bit プロセスログは、以下の ConfigMap を使用して Amazon CloudWatch に送付できます。Fluent Bit プロセスログを CloudWatch に送信するには、追加のログの取り込みおよびストレージのコストがかかります。*region-code* を、クラスターのある AWS リージョンに置き換えます。

```
kind: ConfigMap
apiVersion: v1
metadata:
  name: aws-logging
  namespace: aws-observability
  labels:
data:
  # Configuration files: server, input, filters and output
  # ======================================================
  flb_log_cw: "true"  # Ships Fluent Bit process logs to CloudWatch.

  output.conf: |
    [OUTPUT]
        Name cloudwatch
        Match kube.*
        region region-code
        log_group_name fluent-bit-cloudwatch
        log_stream_prefix from-fluent-bit-
        auto_create_group true
```

ログは、CloudWatch 内のクラスターと同じ AWS リージョンにあります。ロググループ名は *my-cluster*-fluent-bit-logs で、Fluent Bit のログストリーム名は fluent-bit-*podname*-*pod-namespace* です。

> **Note**
>
> - プロセスログは、Fluent Bit プロセスが正常に開始された場合にのみ送付されます。Fluent Bit の起動中に障害が発生すると、プロセスログが失われます。プロセスログは CloudWatch にのみ送付できます。
> - 送付プロセスログをアカウントにデバッグするには、前述の ConfigMap を適用してプロセスログを取得します。Fluent Bit が起動に失敗するのは、通常、起動中に ConfigMap が Fluent Bit によって解析または受け入れられていないためです。

Fluent Bit プロセスログの送信を停止するには

Fluent Bit プロセスログを CloudWatch に送信するには、追加のログの取り込みおよびストレージのコストがかかります。既存の ConfigMap 設定でプロセスログを除外するには、次のステップを実行します。

1. Fargate ログ記録を有効にした後、Amazon EKS クラスターの Fluent Bit プロセスログ用に自動的に作成された CloudWatch ロググループを見つけます。これはフォーマット *my-cluster*-fluent-bit-logs に従います。
2. CloudWatch ロググループ内の各 Pod のプロセスログ用に作成された既存の CloudWatch ログストリームを削除します。
3. ConfigMap を編集して flb_log_cw: "false" を設定します。
4. クラスター内の既存の Pod を再起動します。

アプリケーションをテストする

1. サンプル Pod をデプロイします。
   a. 次の内容をコンピュータ上の sample-app.yaml という名前のファイルに保存します。

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: sample-app
  namespace: same-namespace-as-your-fargate-profile
spec:
  replicas: 3
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
        - name: nginx
          image: nginx:latest
          ports:
            - name: http
```

```
                containerPort: 80
```

b. マニフェストをクラスターに適用します。

```
kubectl apply -f sample-app.yaml
```

2. ConfigMap で設定した送信先を使用して、NGINX ログを表示します。

サイズに関する考慮事項

ログルーター用のメモリは、最大 50 MBに収まるようにすることをお勧めします。アプリケーションで非常に高いスループットでログが生成されることが予想される場合は、最大 100 MB を想定して計画する必要があります。

トラブルシューティング

ConfigMap が無効になっているなど、何らかの理由でログ機能が有効または無効になっているかどうか、および無効になっている理由を確認するには、Pod イベントを kubectl describe pod *pod-name* でチェックしてください。出力には、次の出力例のように、ロギングが有効かどうかを明確にする Pod イベントが含まれる場合があります。

```
[...]
Annotations:          CapacityProvisioned: 0.25vCPU 0.5GB
                      Logging: LoggingDisabled: LOGGING_CONFIGMAP_NOT_FOUND
[...]
Events:
  Type     Reason           Age        From
              Message
  ----     ------           ----       ----
              -------
  Warning  LoggingDisabled  <unknown>  fargate-scheduler
               Disabled logging because aws-logging configmap was not found. configmap
 "aws-logging" not found
```

Pod イベントは一時的なもので、その期間は設定によります。kubectl describe pod *pod-name* を使用して Pod のアノテーションを表示することもできます。Pod アノテーションには、ロギング機能が有効か無効かとその理由に関する情報があります。

最適な Amazon EC2 ノードインスタンスタイプを選択する

Amazon EC2 では、ワーカーノード用のインスタンスタイプが幅広く用意されています。インスタンスタイプごとに、コンピューティング、メモリ、ストレージの異なる機能が提供されます。また、各インスタンスファミリーは、これらの機能に基づきグループ化されています。リストについては、「Amazon EC2 ユーザーガイド」の「利用可能なインスタンスタイプ」を参照してください。Amazon EKS は、(特定の) サポートを有効にするために、Amazon EC2 AMI のいくつかのバリエーションをリリースしています。選択したインスタンスタイプに Amazon EKS との互換性があることを確認する際は、次の基準を考慮してください。

- `mac` ファミリは、現在、すべての Amazon EKS AMI でサポートされていません。
- Arm およびアクセラレータ付きではない Amazon EKS AMI では、`g3`、`g4`、`inf`、および `p` ファミリはサポートされません。
- アクセラレータ付き Amazon EKS AMI は、`a`、`c`、`hpc`、`m`、および `t` ファミリをサポートしていません。
- Arm ベースのインスタンスの場合、Amazon Linux 2023 (AL2023) は Graviton2 以降のプロセッサを使用するインスタンスタイプのみをサポートします。AL2023 は `A1` インスタンスをサポートしていません。

Amazon EKS でサポートされているインスタンスタイプを選択する場合は、各タイプで次の機能を考慮してください。

ノードグループ内のインスタンス数

特に多くの DaemonSet が存在する場合などは、一般的に、数少ない大型のインスタンスの使用が適しています。各インスタンスには API サーバーへの API コールが必要です。したがって、インスタンス数が多いほど、API サーバーのロードが高くなります。

オペレーティングシステム

Linux、Windows、および Bottlerocket に対応しているインスタンスタイプを確認します。Windows インスタンスを作成する前に、「Deploy Windows nodes on EKS clusters」を確認してください。

ハードウェアアーキテクチャ

x86 または Arm のどちらが必要か検討します。Arm インスタンスをデプロイする前に、「Amazon EKS optimized Arm Amazon Linux AMIs」を確認してください。Nitro System (Linux

または Windows) 上に構築されたインスタンス、またはアクセラレータ付き機能が必要かを検討します。アクセラレータ付き機能は、Linux の Amazon EKS でのみ使用できます。

Pod の最大数

各 Pod には独自の IP アドレスが割り当てられているため、インスタンスタイプでサポートされている IP アドレスの数が、インスタンスで実行できる Pod の数を決定するための要因になります。インスタンスタイプでポッドの最大数を決定する方法については、the section called “maxPods の決定方法” を参照してください。

AWS Nitro システム のインスタンスタイプには、サポートできる IP アドレスの数を、非 Nitro のシステムのインスタンスタイプよりも大幅に多くできるオプションがあります。ただし、インスタンスに割り当てられたすべての IP アドレスが Pod で使用できるわけではありません。インスタンスに割り当てる IP アドレスの数を大幅に増やすには、クラスターにバージョン `1.9.0` 以降の Amazon VPC CNI アドオンをインストールし、適切に設定する必要があります。詳細については、the section called “IP アドレスを増やす” を参照してください。インスタンスに最大数の IP アドレスを割り当てるには、バージョン `1.10.1` 以降の Amazon VPC CNI アドオンをクラスターにインストールし、`IPv6` ファミリーを使用してそのクラスターをデプロイする必要があります。

IP ファミリー

クラスターで `IPv4` ファミリーを使用している場合、サポートされているすべてのインスタンスタイプが使用可能になります。これによりクラスターは、プライベート `IPv4` アドレスを Pod とサービスに割り当てることができます。ただし、クラスターに `IPv6` ファミリを使用する場合は、AWS Nitro システム インスタンスタイプ、またはベアメタルインスタンスタイプを使用する必要があります。Windows インスタンスでは、`IPv4` のみがサポートされています。クラスターでは、バージョン `1.10.1` 以降の Amazon VPC CNI アドオンを実行している必要があります。`IPv6` の使用の詳細については、the section called “IPv6” を参照してください。

実行している Amazon VPC CNI アドオンのバージョン

最新バージョンの Amazon VPC CNI Plugin for Kubernetes は、こちらのインスタンスタイプ をサポートしています。サポートされている最新のインスタンスタイプを利用するには、Amazon VPC CNI アドオンのバージョンを更新する必要があります。詳細については、the section called “Amazon VPC CNI” を参照してください。最新バージョンでは、Amazon EKS で使用できる最新の機能をサポートしています。以前のバージョンでは、すべての機能がサポートされているわけではありません。さまざまなバージョンでサポートされている機能は、GitHub の Changelog で確認できます。

ノードを作成している AWS リージョン

AWS リージョンによっては使用できないインスタンスタイプがあります。

Pod にセキュリティグループを使用しているかどうか

Pod にセキュリティグループを使用している場合は、特定のインスタンスタイプのみがサポートされます。詳細については、the section called “ポッドのセキュリティグループ” を参照してください。

maxPods の決定方法

ノードに適用される最終 maxPods 値は、特定の優先順位で相互作用する複数のコンポーネントによって異なります。この優先順位を理解することで、maxPods をカスタマイズする際に予期しない動作を回避できます。

優先順位 (最高から最低):

1. マネージド型ノードグループの適用 – カスタム AMI なしでマネージド型ノードグループを使用する場合、Amazon EKS はノードのユーザーデータの maxPods に上限を適用します。vCPU が 30 未満のインスタンスの場合、上限は 110 です。vCPU が 30 を超えるインスタンスの場合、上限は 250 です。この値は、maxPodsExpression を含む他の maxPods 設定よりも優先されます。
2. kubelet **maxPods** 設定 – kubelet 設定で直接 maxPods を設定する場合 (カスタム AMI を使用した起動テンプレートなど)、この値は maxPodsExpression よりも優先されます。
3. nodeadm **maxPodsExpression** – NodeConfig で maxPodsExpression を使用する場合、nodeadm は式を評価して maxPods を計算します。これは、値がより優先順位の高いソースによってすでに設定されていない場合にのみ有効です。
4. デフォルトの ENI ベースの計算 – 他の値が設定されていない場合、AMI はそのインスタンスタイプでサポートされている Elastic Network Interface と IP アドレスの数に基づいて maxPods を計算します。これは式 `(number of ENIs × (IPs per ENI − 1)) + 2` と同等です。`+ 2` は、ポッド IP アドレスを消費しないすべてのノードで実行される Amazon VPC CNI と kube-proxy を表します。

> **Important**
>
> マネージド型ノードグループを使用して NodeConfig で maxPodsExpression を設定すると、マネージド型ノードグループの適用によって式が上書きされます。マネージド型ノードグループでカスタム maxPods 値を使用するには、起動テンプレートでカスタム AMI を指定

し、maxPods を直接設定する必要があります。詳細については、the section called “起動テンプレート” を参照してください。

マネージド型ノードグループとセルフマネージド型ノードの比較

マネージド型ノードグループ (カスタム AMI なし) では、Amazon EKS はノードのブートストラップユーザーデータに maxPods 値を挿入します。つまり、次のようになります。

- maxPods 値は常にインスタンスサイズに応じて 110 または 250 に制限されます。
- 設定するすべての maxPodsExpression は、この挿入された値によって上書きされます。
- 別の maxPods 値を使用するには、起動テンプレートでカスタム AMI を指定し、--use-max-pods false を --kubelet-extra-args '--max-pods=*my-value*' と一緒に bootstrap.sh スクリプトに渡します。例については、the section called “起動テンプレート” を参照してください。

セルフマネージド型ノードを使用すると、ブートストラッププロセスを完全に制御できます。NodeConfig で maxPodsExpression を使用するか、もしくは --max-pods を bootstrap.sh に直接渡すことができます。

EKS Auto Mode に関する考慮事項

EKS Auto Mode では、ノード上のポッドの数が以下のいずれか低い方に制限されます。

- ハード上限の 110 ポッド
- number of ENIs × (IPs per ENI - 1)

事前構築済みの最適化されたイメージを使用してノードを作成する

マネージドノードグループまたはセルフマネージド型ノードを使用する場合、事前に構築された Amazon EKS 最適化 Amazon マシンイメージ (AMI)、または独自のカスタム AMI のいずれかを使用してノードをデプロイできます。ハイブリッドノードを実行している場合は、「the section called “オペレーティングシステムを準備する”」を参照してください。Amazon EKS 最適化 AMI の各タイプの詳細については、以下のいずれかのトピックを参照してください。独自のカスタム AMI を作成する手順については、「the section called “EKS に最適化されたカスタム Amazon Linux AMI の構築”」を参照してください。

Amazon EKS Auto Mode では、EKS は AMI の選択と更新を含め EC2 インスタンスを管理します。

トピック

- EKS AL2 AMI と AL2 高速化 AMI の移行機能ガイド
- 最適化された Amazon Linux AMI を使用してノードを作成する
- 最適化された Bottlerocket AMI を使用してノードを作成する
- 最適化された Ubuntu Linux AMI を使用してノードを作成する
- 最適化された Windows AMI を使用してノードを作成する

EKS AL2 AMI と AL2 高速化 AMI の移行機能ガイド

> **Warning**
>
> Amazon EKS は、2025 年 11 月 26 日に、EKS 最適化 Amazon Linux 2 (AL2) AMI の公開を取り止めました。Amazon EKS 用の AL2023 および Bottlerocket ベースの AMI は、バージョン 1.33 以降をはじめサポート対象のすべての Kubernetes で使用できます。

AWS は、EKS AL2 最適化 AMI と AL2 高速化 AMI に対するサポートを、2025 年 11 月 26 日に終了します。お客様は、こちらのサポート終了 (EOS) の日付 (2025 年 11 月 26 日) 以降も EKS AL2 AMI をご使用いただけますが、EKS はこの日以降、新しい Kubernetes バージョンや AL2 AMI の更新 (マイナーリリース、パッチ、バグ修正を含む) をリリースすることはありません。当社はお客様に、Amazon Linux 2023 (AL2023) または Bottlerocket AMI へのアップグレードを推奨しています。

- AL2023 では、事前設定済みのセキュリティポリシー、SELinux の Permissive モード、IMDSv2 専用モードのデフォルトでの有効化、起動時間の最適化、セキュリティとパフォーマンスを向上させるパッケージ管理の改良によって、OS レベルの直接アクセスや広範なノード変更など大幅なカスタマイズを必要とするインフラストラクチャに最適な、セキュリティの機能や設定をデフォルトで組み込んだ「セキュアバイデフォルト」のアプローチを実現しています。詳細については、「Amazon Linux 2023 に関するよくある質問」を参照するか、「the section called “AL2023 にアップグレードする”」にある詳細な移行ガイダンスを参照してください。
- Bottlerocket は、ノードのカスタマイズを最小限に抑えたコンテナネイティブアプローチに最適な、専用のコンテナ最適化設計により、セキュリティの強化や起動時間の短縮、アタックサーフェスの縮小を実現して、効率の向上を可能にします。詳細については、「Bottlerocket に関するよくある質問」を参照するか、「the section called “Bottlerocket”」にある詳細な移行ガイダンスを参照してください。

または、EOS 日 (2025 年 11 月 26 日) までは、the section called “EKS に最適化されたカスタム Amazon Linux AMI の構築”が可能です。また、Amazon Linux 2 の EOS 日 (2026 年 6 月 30 日) までは、Amazon Linux 2 ベースインスタンスを使用してカスタムの AMI を構築することができます。

移行とサポートに関するよくある質問

AL2 から AL2023 AMI に移行するにはどうすればよいですか?

詳細なアプリケーションワークロードテストと文書化されたロールバック手順を含む移行計画を作成、実装し、EKS の公式ドキュメントの「Amazon Linux 2 から Amazon Linux 2023 にアップグレードする」の手順に従うことが推奨されます。

EKS 最適化 AL2 AMI の EKS サポート終了 (EOS) 日以降に、カスタム AL2 AMI を構築することは可能ですか?

当社は、AL2023 または Bottlerocket 用として正式にサポートされ公開されている EKS 最適化 AMI に移行されることを推奨しますが、EKS AL2 最適化 AMI と AL2 高速化 AMI は、AL2 AMI の EOS 日 (2025 年 11 月 26 日) まで構築することが可能です。また、Amazon Linux 2 の EOS 日 (2026 年 6 月 30 日) までは、Amazon Linux 2 ベースインスタンスを使用してカスタムの AMI を構築することができます。カスタム EKS AL2 最適化 AMI および AL2 高速化 AMI の構築手順については、EKS 公式ドキュメント内の「カスタム Amazon Linux AMI を構築する」を参照してください。

EKS Kubernetes のバージョンサポートポリシーは、Amazon Linux ディストリビューションに適用されますか?

適用されません。EKS AL2 最適化 AMI と AL2 高速化 AMI の EOS 日は、EKS による Kubernetes バージョンの標準サポートおよび延長サポートのスケジュールとは無関係です。EKS の延長サポートをご利用いただいている場合でも、AL2023 または Bottlerocket への移行が必要になります。

cgroupv1 から cgroupv2 に変更すると、自分の移行にどのような影響が及びますか?

Kubernetes コミュニティは、`cgroupv1` サポート (AL2 で使用) をメンテナンスモードに移行しました。つまり、今後は新機能は追加されず、重要なセキュリティ修正およびバグ修正のみが行われます。`cgroupv2` を Kubernetes に導入するには、OS、カーネル、コンテナランタイム、Kubernetes コンポーネント間で互換性を確保する必要があります。そのためには、AL2023、Bottlerocket、Red Hat Enterprise Linux (RHEL) 9+、Ubuntu 22.04+、Debian 11+ など、`cgroupv2` をデフォルトで有効にする Linux ディストリビューションが必要になります。これらのディストリビューションにはカーネルバージョン 5.8 以降が付属しています。こちらは Kubernetes で `cgroupv2` をサポートするための最小要件です。詳細については「About cgroup v2」を参照してください。

カスタム AL2 AMI で Neuron が必要な場合、どうすればよいですか?

AL2 ベースの AMI では、完全に Neuron をベースとするアプリケーションをネイティブに実行することはできません。AL2 AMI で AWS Neuron を使用する場合は、Neuron がサポートしているコンテナを使って、非 AL2 Linux ディストリビューション (Ubuntu 22.04 や Amazon Linux 2023 など) でアプリケーションをコンテナ化し、そのコンテナを Neuron ドライバー (`aws-neuronx-dkms`) がインストールされた AL2 ベースの AMI にデプロイする必要があります。

EKS AL2 AMI の EOS 日 (2025 年 11 月 26 日) 以降に、ベア Amazon Linux 2 ベースインスタンスに切り替える必要がありますか。

ベア Amazon Linux 2 ベースインスタンスに切り替えると、公式の EKS AL2 最適化 AMI および AL2 高速 AMI が提供する、特定の最適化、コンテナランタイム設定、カスタマイズが利用できません。そのため、AL2 ベースのソリューションを引き続き使用する必要がある場合は、「the section called “EKS に最適化されたカスタム Amazon Linux AMI の構築”」または「Amazon EKS AMI Build Specification」にある EKS AMI レシピを使用して、カスタム AMI を構築することをお勧めします。これにより、既存のワークロードとの互換性が確保され、Amazon Linux 2 の EOS 日 (2026 年 6 月 30 日) まで AL2 カーネルの更新が含まれます。

EKS AL2 AMI の EOS 日 (2025 年 11 月 26 日) 以降に EKS AMI GitHub リポジトリを使用してカスタム AL2 AMI を構築する場合、amzn2-core や amzn2extra-docker などのリポジトリからのパッケージには、どのようなサポートが提供されますか。

Amazon EKS AMI Build Specification にある EKS AMI レシピは、amzn2-core や amzn2extra-docker などの標準的な Amazon Linux 2 ソフトウェアから、YUM 経由でパッケージを取得します。EKS AL2 AMI の EOS 日 (2025 年 11 月 26 日) 以降も、これらのソフトウェアは、Amazon Linux 2 の EOS 日 (2026 年 6 月 30 日) まで引き続きサポートされます。この期間中のサポートはカーネル更新に限定されるため、セキュリティと互換性を維持するには、その他のパッケージ更新、セキュリティパッチ、およびカーネル以外の依存関係を手動で管理し、適用する必要があります。

Amazon EKS で AL2023 を使用している環境において、JDK8 の古いバージョンを使用する Java アプリケーションが Out of Memory (OOM) 例外やポッドの再起動を経験するのはなぜですか。また、どのように解決できますか。

AL2023 を使用する Amazon EKS ノード上で実行する場合、`jdk8u372` より前の JDK 8 バージョンに依存する Java アプリケーションでは、JVM が `cgroupv2` と互換性がないため、OOM 例外やポッドの再起動が発生する可能性があります。この問題は、Amazon Linux 2023 のデフォルトである `cgroupv2` を使用したコンテナのメモリ制限を、JVM が検出できないことに起因します。その結果、JVM は、ポッドで定義された制限ではなく、ノードの合計メモリに基づいてヒープ割り当て

を行います。これは、cgroupv2 によってメモリ制限データの保存場所が変更されたことで、古い Java バージョンが使用可能なメモリを誤って読み取り、ノードレベルのリソースを前提としてしまうためです。考えられる対応策として、次のようなものがあります。

- JDK バージョンのアップグレード: jdk8u372 以降、または cgroupv2 を完全にサポートする新しい JDK バージョンにアップグレードすることで、この問題を解決できます。cgroupv2 を完全にサポートする互換性のある Java バージョンのリストについては、「cgroup v2 について」を参照してください。
- カスタム AMI の構築: AL2 ベースのソリューションを引き続き使用する必要がある場合は、「the section called “EKS に最適化されたカスタム Amazon Linux AMI の構築”」または「Amazon EKS AMI Build Specification」を使用して、AL2 ベースのカスタム (2025 年 11 月 26 日まで) を構築できます。例えば、AL2 ベースの v1.33 AMI (2025 年 11 月 26 日まで) を構築できます。Amazon EKS は、EKS AL2 の EOS 日 (2025 年 11 月 26 日) まで AL2 ベースの AMI を提供します。EOS 日 (2025 年 11 月 26 日) 以降は、独自の AMI を構築する必要があります。
- cgroupv1 の有効化: cgroupv1 を引き続き使用する必要がある場合は、EKS AL2023 AMI で cgroupv1 を有効にできます。有効にするには、sudo grubby --update-kernel=ALL --args="systemd.unified_cgroup_hierarchy=0" を実行し、システム (例えば、Amazon Linux 2023 を実行している EC2 インスタンスまたはノード) を再起動します。これにより、システムのブートパラメータが変更され (例えば、GRUB 設定にカーネルパラメータ「systemd.unified_cgroup_hierarchy=0」を追加して、systemd にレガシーの cgroupv1 階層を使用するように指示します)、cgroupv1 が有効になります。この grubby コマンドの実行により、カーネルは cgroupv1 を有効化し、cgroupv2 を無効化した状態で起動するよう再構成される点に注意してください。これらの cgroup バージョンのうち、ノードでアクティブなリソース管理に使用されるのはいずれか一方のみです。これは、cgroupv1 API に対する下位互換性を備えた cgroupv2 を実行することとは異なります。

> **Warning**
>
> 引き続き cgroupv1 を使用することはお勧めしません。代わりに、cgroupv2 に移行することをお勧めします。Kubernetes コミュニティは、cgroupv1 サポート (AL2 で使用) をメンテナンスモードに移行しました。つまり、今後は新機能や更新は追加されず、重要なセキュリティ修正およびバグ修正のみが行われます。cgroupv1 サポートの完全な削除は今後のリリースで予定されていますが、この完全削除の具体的な日付はまだ発表されていません。cgroupv1 で問題が発生した場合、AWS はサポートを提供できないため、cgroupv2 にアップグレードすることをお勧めします。

互換性とバージョン

AL2 AMI でサポートされている Kubernetes バージョン

Kubernetes バージョン 1.32 は Amazon EKS が Amazon Linux 2 (AL2) AMI をリリースする最後のバージョンです。サポートされている 1.32 までの Kubernetes バージョンに対して、EKS は、2025 年 11 月 26 日まで AL2 AMI (AL2_ARM_64、AL2_x86_64) と AL2 高速化 AMI (AL2_x86_64_GPU) をリリースします。この日を過ぎると、EKS はすべての Kubernetes バージョンで AL2 最適化 AMI と AL2 高速化 AMI のリリースを停止します。EKS AL2 最適化 AMI と AL2 高速化 AMI の EOS 日は、EKS による Kubernetes バージョンの標準サポートおよび延長サポートのスケジュールとは無関係ですのでご注意ください。

AL2、AL2023、Bottlerocket AMI でサポートされているドライバーと Linux カーネルバージョンの比較

コンポーネント	EKS AL2 AMI	EKS AL2023 AMI	EKS Bottlerocket AMI
基本 OS の互換性	RHEL7/CentOS 7	Fedora/CentOS 9	該当なし
CUDA ユーザーモードドライバー	12.x	12.x、13.x	12.x、13.x
NVIDIA GPU ドライバー	R570	R580	R570、R580
AWS Neuron ドライバー	2.20 以降	2.20 以降	2.20 以降
Linux Kernel	5.10	6.1、6.12	6.1、6.12

NVIDIA ドライバーと CUDA の互換性の詳細については、NVIDIA ドキュメントを参照してください。

AL2 AMI と AWS Neuron の互換性

AWS Neuron の 2.20 のリリース以降は、EKS AL ベースの AMI で使用される Neuron ランタイム (`aws-neuronx-runtime-lib`) では、Amazon Linux 2 (AL2) がサポートされなくなります。現在、Amazon Linux 2 をサポートしている AWS Neuron パッケージは、Neuron ドライバー (`aws-neuronx-dkms`) のみです。したがって、AL2 ベースの AMI では、Neuron をベースとするアプリ

ケーションをネイティブに実行することはできません。AL2023 AMI で Neuron をセットアップする方法については、「AWS Neuron Setup」のガイドを参照してください。

Kubernetes と AL2 AMI の互換性

Kubernetes コミュニティは、cgroupv1 サポート (AL2 で使用) をメンテナンスモードに移行しました。つまり、今後は新機能は追加されず、重要なセキュリティ修正およびバグ修正のみが行われます。cgroupv2 に依存する Kubernetes の機能 (MemoryQoS や拡張リソース分離など) は、AL2 では使用することができません。さらに、Amazon EKS Kubernetes バージョン 1.32 は、AL2 AMI をサポートする最後のバージョンとなります。最新の Kubernetes バージョンとの互換性を維持するため、当社は AL2023 または Bottlerocket への移行を推奨しています。これらは cgroupv2 がデフォルトで有効になります。

Linux バージョンと AL2 AMI の互換性

Amazon Linux 2 (AL2) は、2026 年 6 月 30 日のサポート終了 (EOS) 日まで AWS でサポートされます。ただし、AL2 はリリースから長い時間が経過しているため、新しいアプリケーションや機能に対する、より広範な Linux コミュニティによるサポートは、ますます制限されています。AL2 AMI は Linux カーネル 5.10 をベースにしており、一方の AL2023 は Linux カーネル 6.1 を使用しています。AL2023 と異なり、AL2 はより広範な Linux コミュニティからのサポートが、制限されています。つまり、AL2 の古いカーネルバージョンと連携するには、アップストリームの多くの Linux パッケージやツールをバックポートする必要があります。最新の Linux 機能やセキュリティ強化機能の中には、カーネルが古いために使用できないものがあり、オープンソースのプロジェクトの多くが、5.10 などの古いカーネルバージョンのサポートを廃止するか制限しています。

AL2023 に含まれていない非推奨パッケージ

AL2023 に含まれていない、または AL2023 で変更された最も一般的なパッケージには、次のようなものがあります。

- Amazon Linux 2 の一部のソースバイナリパッケージは、Amazon Linux 2023 で利用できなくなりました。
- Amazon Linux による、AL2023 のさまざまなバージョンのパッケージ (amazon-linux-extras システムなど) のサポート方法が変更されています。
- Enterprise Linux (EPEL) の追加パッケージは AL2023 ではサポートされていません。
- 32-bit アプリケーションは AL2023 ではサポートされていません。

詳細については、「AL2 と AL2023 の比較」を参照してください。

AL2、AL2023、Bottlerocket 間の FIPS 検証の比較

Amazon Linux 2 (AL2)、Amazon Linux 2023 (AL2023)、Bottlerocket は、連邦情報処理標準 (FIPS) 適合性をサポートしています。

- AL2 は FIPS 140-2 に基づいて認定され、AL2023 は FIPS 140-3 に基づいて認定されています。AL2023 で FIPS モードを有効にするには、Amazon EC2 インスタンスに必要なパッケージをインストールし、「Enable FIPS Mode on AL2023」に記載された手順に従って設定を行います。詳細については、「Amazon Linux 2023 に関するよくある質問」を参照してください。
- Bottlerocket は、カーネルとユーザースペースのコンポーネントを、FIPS 140-3 Cryptographic Module Validation Program (暗号モジュール試験及び認証制度) に送信された暗号モジュールの使用に制限する、FIPS 専用のバリアントを提供します。

EKS AMI ドライバーとバージョンの変更ログ

EKS AMI のコンポーネントとそのバージョンのリストの完全版は、GitHub の「Amazon EKS AMI Release Notes」を参照してください。

最適化された Amazon Linux AMI を使用してノードを作成する

Amazon Elastic Kubernetes Service (Amazon EKS) は、Kubernetes ワーカーノードの実行に最適化された専用の Amazon マシンイメージ (AMI) を提供します。これらの EKS に最適化された Amazon Linux (AL) AMI は、クラスター内でのシームレスな統合とセキュリティを確保するために、`kubelet`、AWS IAM Authenticator、`containerd` などの重要なコンポーネントで事前設定されています。このガイドでは、使用可能な AMI バージョンの詳細を示すと共に、高速コンピューティングと Arm ベースのアーキテクチャ向けの専用オプションについて説明します。

考慮事項

- Amazon Linux のセキュリティまたはプライバシーに関するイベントを、Amazon Linux セキュリティセンターで追跡できます。そのためには目的のバージョンのタブを選択してください。該当する RSS フィードをサブスクライブすることもできます。セキュリティおよびプライバシーイベントには問題の概要、影響を受けるパッケージ、および問題を修正するためにインスタンスを更新する方法などがあります。
- 高速または Arm AMI をデプロイする前に、Amazon EKS 最適化高速 Amazon Linux AMI および the section called “Amazon EKS 最適化 Arm Amazon Linux AMI” の情報を確認してください。
- Amazon EC2 P2 インスタンスは、`NVIDIA` ドライバーバージョン 470 以前を必要とするため、Amazon EKS ではサポートされていません。

- バージョン 1.30 以降のクラスターで新しく作成されたマネージド型ノードグループは、ノードオペレーティングシステムとして自動的に AL2023 を使用するようデフォルトで設定されます。

Amazon EKS 最適化高速 Amazon Linux AMI

Amazon EKS 最適化高速 Amazon Linux (AL) AMI は、標準的な EKS 最適化 Amazon Linux AMI 上に構築されています。これは、Amazon EKS ノードのオプションのイメージとして機能し、GPU、Inferentia、Trainium ベースのワークロードをサポートするよう設定されています。

詳細については、「the section called “EKS Linux GPU AMI を使用する”」を参照してください。

Amazon EKS 最適化 Arm Amazon Linux AMI

Arm インスタンスは、ウェブサーバー、コンテナ化されたマイクロサービス、キャッシュフリート、および分散データストアといったスケールアウト型の Arm ベースアプリケーションにおいて、コストを大幅に削減します。クラスターに Arm ノードを追加する際には、次の考慮事項を確認してください。

- クラスターが 2020 年 8 月 17 日より前にデプロイされている場合、クラスターの重要なアドオンマニフェストを 1 回だけアップグレードする必要があります。これは、クラスター内で使用中の各ハードウェアアーキテクチャのイメージを、Kubernetes が正しく取得できるようにするためです。クラスターのアドオン更新の詳細については、「the section called “ステップ 1: アップグレードの準備をする”」を参照してください。2020 年 8 月 17 日以降にクラスターをデプロイしている場合、ご使用の CoreDNS、kube-proxy、および Amazon VPC CNI Plugin for Kubernetes アドオンは、既にマルチアーキテクチャに対応しています。
- Arm ノードにデプロイされたアプリケーションは、Arm 用にコンパイルする必要があります。
- 既存のクラスターでデプロイ済みの DaemonSet がある場合、または新しいクラスターで Arm ノードと共にこれをデプロイする場合は、クラスター内のすべてのハードウェアアーキテクチャで DaemonSet が実行可能であることを確認します。
- 同じクラスタ内で、Arm ノードグループと x86 ノードグループを実行することができます。その場合、Pod をデプロイできるハードウェアアーキテクチャを Kubernetes が認識できるようにするために、マルチアーキテクチャのコンテナイメージを Amazon Elastic Container Registry などのコンテナリポジトリにデプロイした上で、ノードセレクターをマニフェストに追加する作業も考慮に入れてください。詳細については、Amazon ECR ユーザーガイドのマルチアーキテクチャイメージのプッシュと、ブログ記事 Introducing multi-architecture container images for Amazon ECR を参照してください。

詳細情報

Amazon EKS 最適化 Amazon Linux AMI の詳細については、以下のセクションを参照してください。

- Amazon Linux をマネージド型ノードグループと使用するには、「the section called “マネージド型ノードグループ”」を参照してください。
- セルフマネージド型の Amazon Linux ノードの起動には、「the section called “最新の ID を取得する”」を参照してください。
- バージョンについては、「the section called “バージョン情報を取得する”」を参照してください。
- Amazon EKS 最適化 Amazon Linux AMI の最新の ID を取得するには、「the section called “最新の ID を取得する”」を参照してください。
- Amazon EKS 最適化 AMI の構築に使用されているオープンソーススクリプトについては、「the section called “EKS に最適化されたカスタム Amazon Linux AMI の構築”」を参照してください。

Amazon Linux 2 から Amazon Linux 2023 にアップグレードする

> **Warning**
>
> Amazon EKS は、2025 年 11 月 26 日に、EKS 最適化 Amazon Linux 2 (AL2) AMI の公開を取り止めました。Amazon EKS 用の AL2023 および Bottlerocket ベースの AMI は、バージョン 1.33 以降をはじめサポート対象のすべての Kubernetes で使用できます。

AL2023 は、クラウドアプリケーションに安全かつ安定した高パフォーマンス環境を提供するように設計された、Linux ベースのオペレーティングシステムです。これは Amazon Web Services の次世代 Amazon Linux であり、サポートされているすべての Amazon EKS バージョンで利用できます。

AL2023 には、AL2 に比べていくつかの改善点があります。詳細については、「Amazon Linux 2023 ユーザーガイド」の「AL2 と AL2023 の比較」を参照してください。AL2 からいくつかのパッケージが追加、アップグレード、削除されています。アップグレードする前に、AL2023 でアプリケーションをテストすることを強くお勧めします。AL2023 のパッケージに関するすべての変更の一覧については、「Amazon Linux 2023 リリースノート」の「Amazon Linux 2023 でのパッケージ変更」を参照してください。

これらの変更に加えて、以下の点に注意してください。

- AL2023 では、YAML 設定スキーマを使用する新しいノード初期化プロセス nodeadm が導入されています。セルフマネージドノードグループまたはカスタム AMI ID を指定する起動テンプレートを使用する場合は、追加のクラスターメタデータを明示的に指定する必要があります。このメタデータは、新しいノードグループを作成するときに指定する必要があります。nodeadm の最低限必要なパラメータの例を以下に示します。ここで、apiServerEndpoint、certificateAuthority、およびサービス cidr が必要になります。

```
---
apiVersion: node.eks.aws/v1alpha1
kind: NodeConfig
spec:
  cluster:
    name: my-cluster
    apiServerEndpoint: https://example.com
    certificateAuthority: Y2VydGlmaWNhdGVBdXRob3JpdHk=
    cidr: 10.100.0.0/16
```

  AL2 ではこれらのパラメータからのメタデータは Amazon EKS DescribeCluster API コールから検出されていました。AL2023 ではノードの大規模なスケールアップ中に API コールによってスロットリングが発生するリスクがあるため、この動作が変更されました。この変更は起動テンプレートのないマネージド型ノードグループを使用している場合や、Karpenter を使用している場合には影響しません。certificateAuthority とサービス cidr の詳細については、「Amazon EKS API リファレンス」の「DescribeCluster」を参照してください。

- AL2023 の場合、nodeadm は、NodeConfigSpec を使用して各ノードの kubelet にパラメータを適用する形式も変更します。AL2 では、これは --kubelet-extra-args パラメータを使用して行われました。これは一般的に、ノードにラベルとテイントを追加するために使用されます。以下の例は、ノードへの maxPods と --node-labels の適用を示しています。

```
---
apiVersion: node.eks.aws/v1alpha1
kind: NodeConfig
spec:
  cluster:
    name: test-cluster
    apiServerEndpoint: https://example.com
    certificateAuthority: Y2VydGlmaWNhdGVBdXRob3JpdHk=
    cidr: 10.100.0.0/16
  kubelet:
```

```
    config:
      maxPods: 110
    flags:
      - --node-labels=karpenter.sh/capacity-type=on-demand,karpenter.sh/nodepool=test
```

- AL2023 には Amazon VPC CNI バージョン 1.16.2 以降が必要です。
- AL2023 にはデフォルトで IMDSv2 が必要です。IMDSv2 には、セキュリティ体制の改善に役立ついくつかの利点があります。セッション指向の認証方式を使用しており、セッションを開始するためにシンプルな HTTP PUT リクエストでシークレットトークンを作成する必要があります。セッショントークンの有効期間は 1 秒 ~ 6 時間です。IMDSv1 から IMDSv2 への移行方法の詳細については、「インスタンスメタデータサービスバージョン 2 の使用への移行」および「Get the full benefits of IMDSv2 and disable IMDSv1 across your AWS infrastructure」を参照してください。IMDSv1 を使用する場合は、インスタンスのメタデータオプションの起動プロパティで設定を手動で上書きすることで使用可能になります。

  > **Note**
  >
  > AL2023 を使用する IMDSv2 の場合、マネージドノードグループのデフォルトのホップ数は異なる場合があります。
  >
  > - 起動テンプレートを使用しない場合、デフォルトは 1 に設定されます。つまり、コンテナが IMDS を使用してノードの認証情報にアクセスすることはできません。ノードの認証情報へのコンテナアクセスが必要な場合は、カスタム Amazon EC2 起動テンプレートを使用するとアクセスが可能になります。
  > - 起動テンプレートでカスタム AMI を使用する場合、デフォルト HttpPutResponseHopLimit は 2 に設定されます。起動テンプレートの HttpPutResponseHopLimit を手動で上書きできます。
  >
  > または、IMDSv2 の代わりに Amazon EKS Pod Identity を使用して、認証情報を提供することもできます。

- AL2023 は、次世代の統合コントロールグループ階層 (cgroupv2) を備えています。cgroupv2 は、コンテナランタイムを実装するために systemd によって使用されます。AL2023 には、cgroupv1 を使用してシステムを実行できるコードが引き続き含まれていますが、この設定は推奨されません。またサポート対象でもありません。この設定は、今後の Amazon Linux のメジャーリリースで完全に削除される予定です。
- AL2023 をサポートするには、eksctl に eksctl のバージョン 0.176.0 以降が必要です。

既存のマネージド型ノードグループの場合は、起動テンプレートの使用方法に応じて、インプレースアップグレードまたは Blue/Green アップグレードを実行できます。

- マネージド型ノードグループでカスタム AMI を使用している場合は、起動テンプレートの AMI ID を入れ替えることで、インプレースアップグレードを実行できます。このアップグレード戦略を実行する前に、まずアプリケーションとユーザーデータが AL2023 に転送されていることを必ず確認してください。
- 標準の起動テンプレートまたは AMI ID を指定しないカスタム起動テンプレートでマネージド型ノードグループを使用している場合は、Blue/Green 戦略を使用してアップグレードする必要があります。通常、Blue/Green アップグレードはより複雑で、AMI タイプとして AL2023 を指定する完全に新しいノードグループを作成する必要があります。新しいノードグループの設定は、AL2 ノードグループのすべてのカスタムデータが新しい OS と互換性があることを確認して、慎重に行う必要があります。新しいノードグループがアプリケーションでテストおよび検証されると、Pod を古いノードグループから新しいノードグループに移行できます。移行が完了したら、古いノードグループを削除できます。

Karpenter を利用していて、AL2023 を使用する場合は、`EC2NodeClass` `amiFamily` フィールドを AL2023 に変更する必要があります。デフォルトでは、Karpenter ではドリフトが有効になっています。つまり、`amiFamily` フィールドが変更されると、Karpenter はワーカーノードを使用可能な最新の AMI に自動的に更新します。

nodeadm に関する追加情報

EKS 最適化 Amazon Linux 2023 AMI を利用する場合、または公式の amazon-eks-ami GitHub リポジトリで提供されている Packer スクリプトを使用してカスタム EKS Amazon Linux 2023 AMI を構築する場合は、EC2 ユーザーデータ内またはカスタム AMI の一部として nodeadm init を明示的に実行しないようにする必要があります。

ユーザーデータで動的 NodeConfig を生成する場合は、その設定を `/etc/eks/nodeadm.d` のドロップイン yaml または json ファイルに書き込むことができます。これらの設定ファイルは、nodeadm init が起動プロセスの後半で自動的に開始されると、マージされてノードに適用されます。例えば、次のようになります。

```
cat > /etc/eks/nodeadm.d/additional-node-labels.yaml << EOF
apiVersion: node.eks.aws/v1alpha1
kind: NodeConfig
spec:
  kubelet:
    flags:
```

```
      - --node-labels=foo=bar
EOF
```

EKS 最適化 Amazon Linux 2023 AMI は、個別の systemd サービスを通じて nodeadm init を 2 つのフェーズで自動的に実行します。nodeadm-config はユーザーデータの実行前に実行され、nodeadm-run は実行後に実行されます。nodeadm-config サービスは、ユーザーデータを実行する前に containerd と kubelet のベースライン設定を確立します。nodeadm-run サービスは、選択したシステムデーモンを実行し、ユーザーデータの実行後に最終的な設定を完了します。こうした自動実行以外にユーザーデータまたはカスタム AMI を介して nodeadm init コマンドが実行されると、実行順序に関する仮定が破られ、ENI の設定ミスなどの予期しない結果につながる可能性があります。

Amazon Linux AMI のバージョン情報を取得する

Amazon EKS 最適化 Amazon Linux AMI は、Kubernetes バージョンと AMI のリリース日によって次の形式でバージョニングされています。

```
k8s_major_version.k8s_minor_version.k8s_patch_version-release_date
```

各 AMI リリースには、kubelet、Linux カーネル、containerd のさまざまなバージョンが含まれます。また高速 AMI には、さまざまなバージョンの NVIDIA ドライバーも含まれます。このバージョンに関する情報は、GitHub の「Releases」で検索できます。

推奨 Amazon Linux AMI ID を取得する

ノードをデプロイする際に、事前構築済みの Amazon EKS 最適化 Amazon マシンイメージ (AMI) の ID を指定できます。希望する設定に合った AMI ID を取得するには、AWS Systems Manager Parameter Store API をクエリします。この API を使用すると、Amazon EKS 最適化 AMI ID を手動で検索する必要がなくなります。詳細については、「GetParameter」を参照してください。使用する IAM プリンシパルには、Amazon EKS 最適化 AMI メタデータを取得するための `ssm:GetParameter` IAM アクセス許可が必要です。

サブパラメータ `image_id` を指定する次のコマンドを使用することで、推奨される最新の Amazon EKS 最適化 Amazon Linux AMI のイメージ ID を取得できます。必要に応じてコマンドに次の変更を加え、変更したコマンドを実行してください：

- `<kubernetes-version>` を Amazon EKS がサポートする任意のバージョンに置き換えます。
- *`ami-type`* は、以下のいずれかのオプションに置き換えます。Amazon EC2 インスタンスタイプの詳細については、「Amazon EC2 インスタンスタイプ」を参照してください。

- Amazon Linux 2023 (AL2023) x86 ベースのインスタンスには *amazon-linux-2023/x86_64/standard* を使用します。
- AWS Graviton ベースのインスタンスなどの AL2023 ARM インスタンスには *amazon-linux-2023/arm64/standard* を使用します。
- 最新の承認済みの AL2023 NVIDIA x86 ベースのインスタンスには、*amazon-linux-2023/x86_64/nvidia* を使用します。
- 最新の承認済みの AL2023 NVIDIA arm64 ベースのインスタンスには、*amazon-linux-2023/arm64/nvidia* を使用します。
- 最新の AL2023 AWS Neuron インスタンスには、*amazon-linux-2023/x86_64/neuron* を使用します。

- <region-code> を、AMI ID を必要とする Amazon EKS がサポートされている AWS リージョンで置き換えます。

```
aws ssm get-parameter --name /aws/service/eks/optimized-ami/<kubernetes-version>/<ami-type>/recommended/image_id \
    --region <region-code> --query "Parameter.Value" --output text
```

プレースホルダーの置換が行われた後のコマンドの例を以下に示します。

```
aws ssm get-parameter --name /aws/service/eks/optimized-ami/1.31/amazon-linux-2023/x86_64/standard/recommended/image_id \
    --region us-west-2 --query "Parameter.Value" --output text
```

出力例は次のとおりです。

```
ami-1234567890abcdef0
```

EKS に最適化されたカスタム Amazon Linux AMI の構築

> **Warning**
>
> Amazon EKS は、2025 年 11 月 26 日に、EKS 最適化 Amazon Linux 2 (AL2) AMI の公開を取り止めました。Amazon EKS 用の AL2023 および Bottlerocket ベースの AMI は、バージョン 1.33 以降をはじめサポート対象のすべての Kubernetes で使用できます。

Amazon EKS では、「Amazon EKS AMI Build Specification」リポジトリにオープンソースのビルドスクリプトが用意されており、これらを使用してkubelet、ランタイム環境、AWS IAM Authenticator for Kubernetes の設定を確認し、AL ベースの AMI を一から構築できます。

このリポジトリには、ブート時に実行される特殊な AL2 用 bootstrap スクリプトと AL2023 用 nodeadm ツールが保存されています。これらのスクリプトによって、インスタンスの証明書データ、コントロールプレーンエンドポイント、クラスター名などを設定します。ここに置かれているものは、Amazon EKS 最適化 AMI のビルド用として最も信頼できるスクリプトと見なされるため、GitHub リポジトリの状態をフォローすることで、AMI への変更をモニタリングできます。

EKS 最適化 AMI をベースとしてカスタム AMI を構築する場合、コンポーネントの互換性が損なわれる可能性があるため、オペレーティングシステムのアップグレード (dnf upgrade など) を実行したり、EKS 最適化 AMI に含まれる Kubernetes または GPU パッケージをアップグレードしたりすることは、推奨もサポートもされていません。EKS 最適化 AMI に含まれるオペレーティングシステムまたはパッケージをアップグレードする場合は、本番環境にデプロイする前に、開発環境またはステージング環境で徹底的にテストすることをお勧めします。

GPU インスタンス用のカスタム AMI を構築する場合は、実行するインスタンスタイプの世代およびファミリーごとに、個別のカスタム AMI を構築することをお勧めします。EKS 最適化高速 AMI は、基盤となるインスタンスタイプの世代およびファミリーに基づいて、ランタイム時にドライバーとパッケージを選択的にインストールします。詳細については、インストールとランタイムの EKS AMI スクリプトを参照してください。

前提条件

- AWS CLI をインストールする
- HashiCorp Packer v1.9.4+ をインストールする
- GNU Make をインストールする

必要な Amazon プラグインをインストールします。

```
packer plugins install github.com/hashicorp/amazon
```

Packer がインストールされていることを確認します。

```
packer --version
```

公式の Amazon EKS AMI リポジトリをクローンまたはフォークします。

```
git clone https://github.com/awslabs/amazon-eks-ami.git
cd amazon-eks-ami
```

デフォルト値と追加オプションを表示するには、次のコマンドを実行します。AMI のカスタマイズに使用できる設定の詳細については、「Amazon Linux 2023」ページのテンプレート変数を参照してください。

```
make help
```

カスタム AMI を作成する

さまざまなカスタム AMI のコマンド例を以下に示します。

EKS NVIDIA AL2 AMI

```
make k8s=1.32 os_distro=al2 \
  enable_accelerator=nvidia \
  nvidia_driver_major_version=580 \
  enable_efa=true
```

EKS NVIDIA AL2023 AMI

```
make k8s=1.36 os_distro=al2023 \
  enable_accelerator=nvidia \
  nvidia_driver_major_version=580 \
  enable_efa=true
```

カスタムドライバーバージョンを搭載した EKS NVIDIA AL2023 AMI

EKS 最適化 AL2023 NVIDIA AMI は、2 つの NVIDIA ドライバーバージョンを 1 つのイメージにバンドルします。各ノードは、起動時に GPU に適したドライバーを自動的に選択します。EKS AL2023 AMI バージョン v20260917 の時点で、LTS (長期サポート) バージョンは 580、本番ブランチバージョンは 595 であり、すべての EC2 GPU インスタンスタイプをサポートしています。デフォルトバージョンを上書きするには、nvidia_driver_lts_version と nvidia_driver_pb_version を一緒に設定します。

```
make k8s=1.36 os_distro=al2023 \
  enable_accelerator=nvidia \
  nvidia_driver_lts_version=580 \
```

```
  nvidia_driver_pb_version=610 \
  enable_efa=true
```

> **Note**
>
> EKS AL2023 AMI の NVIDIA ドライバー選択ロジックは実行時に動作しますが、AMI が高速 EC2 インスタンス上に構築されている場合は早期にアクティブ化されます。その結果、NVIDIA ドライバー 580 をデフォルトとする高速 EC2 インスタンス (g6 や g7e など) にカスタム AMI を構築する場合、そのカスタム AMI を、新しい NVIDIA ドライバーバージョン (ドライバー 595 を必要とする g7 など) を必要とする高速 EC2 インスタンスにデプロイすることはできません。これを回避するには、非高速 EC2 インスタンス (m6 など) にカスタム AMI を構築し、任意の高速 EC2 インスタンスにデプロイします。このアプローチにより、EKS AL2023 AMI の NVIDIA ドライバー選択ロジックが維持されます。

STIG 準拠 Neuron AL2023 AMI

```
make k8s=1.31 os_distro=al2023 \
  enable_accelerator=neuron \
  enable_fips=true \
  source_ami_id=ami-0abcd1234efgh5678 \
  kms_key_id=alias/aws-stig
```

これらのコマンドを実行すると、Packer は以下を実行します:

- 一時的な Amazon EC2 インスタンスを起動します。
- Kubernetes コンポーネント、ドライバー、設定をインストールします。
- AWS アカウントで AMI を作成します。

正常な出力は次の例のようになります。

```
==> Wait completed after 8 minutes 42 seconds

==> Builds finished. The artifacts of successful builds are:
--> amazon-ebs: AMIs were created:
us-west-2: ami-0e139a4b1a7a9a3e9

--> amazon-ebs: AMIs were created:
us-west-2: ami-0e139a4b1a7a9a3e9
```

```
--> amazon-ebs: AMIs were created:
us-west-2: ami-0e139a4b1a7a9a3e9
```

最適化された Bottlerocket AMI を使用してノードを作成する

Bottlerocket は、AWS がスポンサーおよびサポートするオープンソースの Linux ディストリビューションです。Bottlerocket は、コンテナワークロードをホストするために専用に構築されています。Bottlerocket を使用すると、コンテナインフラストラクチャの更新を自動化することで、コンテナ化されたデプロイの可用性を向上させ、運用コストを削減できます。Bottlerocket には、コンテナの実行に不可欠なソフトウェアのみが含まれており、リソース使用率の向上、セキュリティ上の脅威の軽減、管理オーバーヘッドの軽減が実現されます。Bottlerocket AMI には、`containerd`、`kubelet`、および AWS IAM Authenticator が含まれます。Bottlerocket は、マネージド型ノードグループとセルフマネージド型ノードに加え、Karpenter でもサポートされています。

利点

Amazon EKS クラスターと Bottlerocket を使用すると次のようなメリットがあります。

- 運用コストが低く、管理の複雑さが軽減されたことによるアップタイムの向上 – Bottlerocket は、他の Linux ディストリビューションよりもリソースフットプリントが小さく、ブート時間が短く、セキュリティ脅威に対する脆弱性が低くなります。Bottlerocket のフットプリントが小さいため、ストレージ、コンピューティング、ネットワークリソースの使用量が減り、コストを削減できます。
- OS の自動更新によるセキュリティの向上 — Bottlerocket への更新プログラムは単一のユニットとして適用され、必要に応じてロールバックできます。これにより、システムが使用不能な状態にさらす更新プログラムの破損または失敗のリスクをなくします。Bottlerocket を使用すると、セキュリティ更新プログラムが利用可能になり次第、中断を最小限に抑えながら自動的に適用され、障害が発生した場合はロールバックできます。
- プレミアムサポート — Amazon EC2 で AWS が提供する Bottlerocket のビルドは、Amazon EC2、Amazon EKS、Amazon ECR などの AWS サービスも対象とする同じ AWS サポートプランの対象となります。

考慮事項

Bottlerocket を AMI タイプに使用する際、次の事項を考慮してください。

- Bottlerocket は、x86_64 と arm64 プロセッサを搭載した Amazon EC2 インスタンスをサポートします。
- Bottlerocket は、GPU を搭載した Amazon EC2 インスタンスをサポートします。詳細については、「the section called “EKS Linux GPU AMI を使用する”」を参照してください。
- Bottlerocket イメージには、SSH サーバーまたはシェルは含まれません。帯域外のアクセス方法を使用して SSH を許可できます。これらの手法は、管理者コンテナを有効にし、ユーザーデータでブートストラップの設定ステップの渡しを可能にします。詳細については、GitHub の「Bottlerocket OS」内にある次のセクションを参照してください。
  - 探査
  - 管理者コンテナ
  - Kubernetes 設定
- Bottlerocket は以下のさまざまなコンテナタイプを使用します。
  - デフォルトで、「コントロールコンテナ」は有効になっています。このコンテナは、Amazon EC2 Bottlerocket インスタンス上でのコマンドの実行や、シェルセッションの開始に使用できる AWS Systems Manager エージェントを実行します。詳細については、AWS Systems Manager ユーザーガイドの「Session Manager のセットアップ」を参照してください。
  - ノードグループの作成時に SSH キーが与えられる場合、管理者コンテナが有効になります。admin container は、開発とテストのシナリオにのみ使用することをお勧めします。本番環境で使用することはお勧めしません。詳細については、GitHub の Admin container を参照してください。

詳細情報

Amazon EKS 最適化 Bottlerocket AMI の使用に関する詳細については、以下のセクションを参照してください。

- Bottlerocket の詳細については、Bottlerocket のドキュメントを参照してください。
- バージョン情報のリソースについては、「the section called “バージョン情報を取得する”」を参照してください。
- Bottlerocket をマネージド型ノードグループと使用するには、「the section called “マネージド型ノードグループ”」を参照してください。
- セルフマネージド型の Bottlerocket ノードを起動するには、「the section called “Bottlerocket”」を参照してください。

- Amazon EKS 最適化 Bottlerocket AMI の最新の ID を取得するには、「the section called “最新の ID を取得する”」を参照してください。
- コンプライアンスサポートの詳細については、「the section called “コンプライアンスサポート”」を参照してください。
- 特定の NVIDIA ドライバーを使用してカスタム Bottlerocket バリアントを構築するには、the section called “カスタム Bottlerocket AMI を構築する” を参照してください。

Bottlerocket AMI のバージョン情報を取得する

各 Bottlerocket AMI リリースには、kubelet、Bottlerocket カーネル、containerd のさまざまなバージョンが含まれます。また高速 AMI バリアントには、さまざまなバージョンの NVIDIA ドライバーも含まれます。このバージョン情報は、Bottlerocket のドキュメントの「OS」のトピックで確認できます。このページから、該当する「Version Information」のサブトピックに移動します。

Bottlerocket のドキュメントは、GitHub で利用可能なバージョンよりも遅れることがあります。最新バージョンの変更のリストは、GitHub の「Releases」で確認できます。

推奨 Bottlerocket AMI ID を取得する

ノードをデプロイする際に、事前構築済みの Amazon EKS 最適化 Amazon マシンイメージ (AMI) の ID を指定できます。希望する設定に合った AMI ID を取得するには、AWS Systems Manager Parameter Store API をクエリします。この API を使用すると、Amazon EKS 最適化 AMI ID を手動で検索する必要がなくなります。詳細については、「GetParameter」を参照してください。使用する IAM プリンシパルには、Amazon EKS 最適化 AMI メタデータを取得するための `ssm:GetParameter` IAM アクセス許可が必要です。

サブパラメータ `image_id` を使用する次の AWS CLI コマンドを使用することで、推奨される最新の Amazon EKS 最適化 Bottlerocket AMI のイメージ ID を取得できます。必要に応じてコマンドに次の変更を加え、変更したコマンドを実行してください：

- *`kubernetes-version`* を、サポートされている platform-version に置き換えます。
- *`-flavor`* は、以下のいずれかのオプションに置き換えます。
  - GPU のないバリアントの場合は *`-flavor`* を削除します。
  - GPU 対応バリアントには *`-nvidia`* を使用します。
  - FIPS 対応バリアントには *`-fips`* を使用します。
- *`architecture`* は、以下のいずれかのオプションに置き換えます。

  - x86 ベースのインスタンスには *x86_64* を使用します。
  - ARM インスタンスには *arm64* を使用します。
- *region-code* を、AMI ID が必要な Amazon EKS がサポートしている AWS リージョンに置き換えます。

```
aws ssm get-parameter --name /aws/service/bottlerocket/aws-k8s-kubernetes-version-
flavor/architecture/latest/image_id \
    --region region-code --query "Parameter.Value" --output text
```

プレースホルダーの置換が行われた後のコマンドの例を以下に示します。

```
aws ssm get-parameter --name /aws/service/bottlerocket/aws-k8s-1.31/x86_64/latest/
image_id \
    --region us-west-2 --query "Parameter.Value" --output text
```

出力例は次のとおりです。

```
ami-1234567890abcdef0
```

Bottlerocket を使用してコンプライアンス要件に準拠する

Bottlerocket はさまざまな組織によって定義された推奨事項に準拠しています。

- CIS ベンチマークは Bottlerocket に定義されています。デフォルト設定では、Bottlerocket イメージには CIS レベル 1 構成プロファイルに必要なほとんどのコントロールが含まれています。CIS レベル 2 構成プロファイルに必要なコントロールを実装できます。詳細については、「AWS ブログ」の「Amazon EKS に最適化された Bottlerocket AMI を CIS ベンチマークと照合して検証」を参照してください。
- 最適化された機能セットおよび最小化したアタックサーフェスは、攻撃対象領域が小さくなるため、Bottlerocket インスタンスが PCI DSS 要件を満たすために必要な構成が少なくて済みます。Bottlerocket の CIS Benchmark は、ガイダンスを強化するために優れたリソースであり、PCI DSS 要件 2.2 に基づく安全な構成標準の要件をサポートします。Fluent Bit を活用して、PCI DSS 要件 10.2 に基づくオペレーティングシステムレベルの監査ログ記録の要件をサポートすることもできます。AWS は、PCI DSS 要件 6.2 (v3.2.1 用) および要件 6.3.3 (v4.0 用) を満たすために役立つ新しい (パッチが適用された) Bottlerocket インスタンスを定期的に公開します。

- Bottlerocket は、Amazon EC2 および Amazon EKS の両方に規制対象のワークロードで使用が承認された HIPAA 対象の機能です。詳細については、「HIPAA 対応サービスのリファレンス」を参照してください。
- Bottlerocket AMI はFIPS 140-3 検証済み暗号化モジュールを使用するように事前設定されています。これには Amazon Linux 2023 Kernel Crypto API 暗号化モジュールや AWS-LC 暗号化モジュールなどがあります。詳細については、「the section called “Bottlerocket FIPS AMI”」を参照してください。

Bottlerocket FIPS AMI を使用してワーカーノードを FIPS 対応にする

連邦情報処理規格 (Federal Information Processing Standards/FIPS) 公開情報 140-3 は、機密情報を保護する暗号モジュールのセキュリティ要件を規定する、米国およびカナダ政府の基準です。Bottlerocket からは、FIPS カーネルを備えた AMI が提供されており、これによって、FIPS への準拠が容易になります。

これらの AMI は、FIPS 140-3 検証済み暗号化モジュールを使用するように事前構成されています。これには Amazon Linux 2023 Kernel Crypto API 暗号化モジュールや Go 暗号化モジュールなどがあります。

Bottlerocket FIPS AMI を使用すると、ワーカーノードは「FIPS 対応」になりますが、自動的に「FIPS に準拠」するわけではありません。詳細については、「連邦情報処理規格 (FIPS) 140-3」を参照してください。

考慮事項

- クラスターで分離サブネットを使用している場合、Amazon ECR FIPS エンドポイントにアクセスできない場合があります。これによって、ノードのブートストラップが失敗する可能性があります。ネットワーク設定で、必要な FIPS エンドポイントへのアクセスが許可されていることを確認してください。詳細については、「AWS PrivateLink ガイド」の「Access a resource through a resource VPC endpoint」を参照してください。
- クラスターが PrivateLink でサブネットを使用している場合は、Amazon ECR FIPS PrivateLink エンドポイントが設定されていることを確認してください。Amazon ECR は PrivateLink を通じて FIPS エンドポイントをサポートします。詳細については、「Amazon ECR インターフェイス VPC エンドポイント」を参照してください。

Bottlerocket FIPS AMI を使用してマネージド型ノードグループを作成する

Bottlerocket FIPS AMI には、ワークロードをサポートする 4 つのバリアントがあります。

- BOTTLEROCKET_x86_64_FIPS
- BOTTLEROCKET_ARM_64_FIPS
- BOTTLEROCKET_x86_64_NVIDIA_FIPS
- BOTTLEROCKET_ARM_64_NVIDIA_FIPS

Bottlerocket FIPS AMI を使用してマネージド型ノードグループを作成するには、作成プロセス中に、該当する AMI タイプを選択します。詳細については、the section called “作成” を参照してください。

FIPS 対応バリアントの選択の詳細については、the section called “最新の ID を取得する” を参照してください。

サポートされていない AWS リージョンの FIPS エンドポイントを無効にする

Bottlerocket FIPS AMI は、AWS GovCloud (米国) リージョンを含め、米国では直接サポートされています。AMI が使用可能でも、直接サポートされていない AWS リージョンでは、起動テンプレートでマネージド型ノードグループを作成すると、AMI を引き続き使用できます。

Bottlerocket FIPS AMI は、ブートストラップ中に Amazon ECR FIPS エンドポイントに依存しますが、このエンドポイントは、米国以外では一般には利用できません。Amazon ECR FIPS エンドポイントが利用できない AWS リージョンで FIPS カーネルを目的に AMI を使用するには、次のステップを実行して FIPS エンドポイントを無効にします。

1. 以下を記述した新しい設定ファイルを作成するか、既存の設定ファイルに以下を記述します。

```
[default]
use_fips_endpoint=false
```

1. ファイルの記述を Base64 形式でエンコードします。
2. 起動テンプレートの UserData で、エンコードした次の文字列を TOML 形式で追加します。

```
[settings.aws]
config = "<your-base64-encoded-string>"
```

その他の設定については、GitHub にある、Bottlerocket の「Description of settings」を参照してください。

起動テンプレートに記述した UserData の例を次に示します。

```
[settings]
motd = "Hello from eksctl!"
[settings.aws]
config = "W2RlZmF1bHRdCnVzZV9maXBzX2VuZHBvaW50PWZhbHNlCg==" # Base64-encoded string.
[settings.kubernetes]
api-server = "<api-server-endpoint>"
cluster-certificate = "<cluster-certificate-authority>"
cluster-name = "<cluster-name>"
...<other-settings>
```

起動テンプレート作成の詳細については、「the section called “起動テンプレート”」を参照してください。

Amazon EKS 用のカスタム Bottlerocket AMI バリアントを構築する

Amazon Elastic Kubernetes Service (Amazon EKS) で GPU ワークロードを実行する場合は、Bottlerocket バリアントを選択します。バリアントは、Kubernetes バージョンとアクセラレータータイプと一致する必要があります。Bottlerocket は、一般的な設定用に検証済みバリアントを提供します。ただし、次のいずれかの理由により、組織で別のバリアントが必要になる可能性があります:

- 新しい NVIDIA ドライバーブランチ
- 規制コンプライアンスのためのピン留めされたドライバーバージョン
- モニタリング用の追加パッケージ
- セキュリティチームが必要とする強化されたベースライン

GitHub ウェブサイトの Bottlerocket は完全なオープンソースであるため、これらのニーズを満たすカスタムバリアントを作成できます。このトピックでは、既存のバリアントのクローンを作成し、NVIDIA ドライバーを R580 ブランチから R595 (バージョン 595.71.05) にスワップする方法について説明します。次に、イメージを構築し、それをプライベート AMI として登録します。

> **Important**
>
> g7 インスタンスタイプには、EKS Bottlerocket NVIDIA 1.37 AMI でサポートされている NVIDIA ドライバーバージョン 595 以降が必要です。EKS Bottlerocket NVIDIA AMI 1.36 以前には現在、g7 インスタンスをサポートしていない NVIDIA ドライバーバージョン 580 が含まれています。

NVIDIA ドライバーバージョン 595 でバリアントを構築する手順については、GitHub ウェブサイトの「Bottlerocket リポジトリ」を参照してください。このトピックでは、ステップ 1 から始まる完全なプロセスについて説明します。

ビルドシステムの仕組み

`cargo make -e BUILDSYS_VARIANT=aws-k8s-1.36-nvidia` を実行すると、次の 3 つのことが発生します:

- 依存関係を取得する。Twoliter (Bottlerocket ビルドオーケストレーター) は `Twoliter.toml` を読み取り、`public.ecr.aws/bottlerocket` から 3 つの Open Container Initiative (OCI) アーティファクトをプルします:
  - bottlerocket-sdk – 完全なクロスコンパイルツールチェーン (GCC、Rust、Go、RPM マクロ) を持つコンテナイメージ。
  - bottlerocket-kernel-kit – カーネル、カーネルモジュール (NVIDIA kmod パッケージを含む)、およびファームウェア用の構築済み RPM。
  - bottlerocket-core-kit – ユーザースペース用の構築済み RPM: kubelet、containerd、NVIDIA デバイスプラグインとコンテナツールキット、設定プラグイン、システムサービス。

    `Twoliter.toml` はバージョンをピン留めし、`Twoliter.lock` はダイジェストをロックします。いずれかを変更するには、`./tools/twoliter/twoliter update` を実行して再解決します。
- バリアントを構築する。Twoliter は SDK コンテナ内で Docker ビルドを起動します。バリアントの設定デフォルトクレートをコンパイルし、キットから RPM 依存関係ツリーを解決し、すべてをディスクイメージにアセンブルします。
- 出力する。Twoliter は最終 `.img.lz4` ファイルを `build/images/` に書き込みます。ビルドは決定論的な出力を生成します: ホストに関係なく、同じ `Twoliter.toml` ピンとバリアント `Cargo.toml` は常に同じイメージを生成します。

リポジトリレイアウト

次のディレクトリはバリアント作業に関連しています:

```
bottlerocket/
### Cargo.toml                      # workspace: lists every variant
### Twoliter.toml                   # pins SDK + kit versions
### Twoliter.lock                   # locked digests for the above
```

```
### Licenses.toml                    # you create this (NVIDIA license acknowledgement)
### Infra.toml                       # you create this (AMI publish regions)
#
### variants/
#   ### aws-k8s-1.36-nvidia/         # example variant you'll copy
#   #   ### Cargo.toml               #   package list + kernel params
#   #   ### amispec.toml             #   symlink # ../shared/amispec-split.toml
#   ### shared/                      # shared AMI spec templates
#
### sources/
#   ### Cargo.toml                   # workspace: lists every settings-defaults crate
#   ### shared-defaults/             # the actual defaults (symlink targets)
#   ### settings-defaults/
#       ### aws-k8s-1.36-nvidia/
#           ### Cargo.toml
#           ### defaults.d/          # 30+ symlinks into shared-defaults/
#
### packages/
    ### settings-defaults/
    #   ### settings-defaults.spec   # RPM: declares which variants exist
    ### settings-plugins/
        ### settings-plugins.spec    # RPM: maps variants to settings plugins
```

リポジトリ構造に関する次の注意事項をレビューします:

- バリアントは主にメタデータです。外部キットは、カーネル、ドライバー、ユーザースペースを提供します。
- Settings-defaults ファイルは、コピーではなくシンボリックリンクです。`cp -R` (macOS の `cp -r` ではなく) を使用してそれらを保持します。
- バリアントを追加するには、次の 5 つの場所で編集する必要があります: 2 つのワークスペース `Cargo.toml` ファイル、2 つの `.spec` ファイル、および `README.md`。

前提条件

このチュートリアルを完了するには、以下が必要になります:

- EC2 インスタンスを起動して AMI を登録するアクセス許可を持つ AWS アカウント
- 少なくとも 8 コア、16 GiB メモリ、および 150 GB ディスクを持つ EC2 インスタンス (または同等の Linux x86_64 ホスト)
- Ubuntu 24.04 LTS (または Fedora。macOS はビルドホストとしてサポートされていません)

- Docker 20.10 以降
- Rust (安定ツールチェーン、rustup 経由でインストールされている)
- cargo-make (最新バージョン)
- Git、Rust の Cargo、RPM パッケージングの概念に精通していること

> **Note**
>
> このチュートリアルを完了したら、EC2 インスタンスを終了し、継続的な料金が発生しないよう不要になった AMI を登録解除します。クリーンアップ手順については、the section called “クリーンアップ” を参照してください。

ステップ 1: ビルドホストを準備する

EC2 インスタンスを起動します – 例えば、`c7i.8xlarge` (32 vCPU、64 GiB メモリ)。150 GB の gp3 ルートボリュームを使用し、SSM アクセス用の `AmazonSSMManagedInstanceCore` マネージドポリシーをアタッチします。

AWS Systems Manager (SSM) Session Manager を使用してインスタンスに接続します:

```
aws ssm start-session --target <instance-id>
cd ~
```

必要なオペレーティングシステムパッケージをインストールします:

```
apt-get update
apt-get install -y build-essential openssl libssl-dev pkg-config lz4 \
                   git ca-certificates curl gnupg
```

> **Note**
>
> 公式の `BUILDING.md` は `liblz4-tool` を参照します。最近の Ubuntu バージョンでは、パッケージは `lz4` という名前です。

次のコマンドを使用して Docker をインストールします:

```
install -m 0755 -d /etc/apt/keyrings
curl -fsSL https://download.docker.com/linux/ubuntu/gpg \
  -o /etc/apt/keyrings/docker.asc
chmod a+r /etc/apt/keyrings/docker.asc
echo "deb [arch=$(dpkg --print-architecture) signed-by=/etc/apt/keyrings/docker.asc] \
  https://download.docker.com/linux/ubuntu noble stable" \
  > /etc/apt/sources.list.d/docker.list
apt-get update
apt-get install -y docker-ce docker-ce-cli containerd.io docker-buildx-plugin
systemctl enable --now docker
```

次のコマンドを使用して Rust と cargo-make をインストールします:

```
curl --proto '=https' --tlsv1.2 -sSf https://sh.rustup.rs | sh -s -- -y
. "$HOME/.cargo/env"
cargo install cargo-make
```

ステップ 2: リポジトリのクローンを作成する

GitHub ウェブサイトの Bottlerocket リポジトリのクローンを作成し、ディレクトリに移動します:

```
cd ~/bottlerocket
```

再現可能なビルドを作成するには、タグ付けされたリリース (`git checkout v1.62.1` など) をチェックアウトします。最新のパッケージを使用するには、`develop` ブランチに留まります。

ステップ 3: キットのバージョンを確認する

`Twoliter.toml` を開き、キットのバージョンを確認します。R595 をサポートするには、`bottlerocket-kernel-kit` バージョン 6.2.2 以降が必要です。

```
[[kit]]
name = "bottlerocket-kernel-kit"
version = "6.2.2"
vendor = "bottlerocket"
```

バージョンが古い場合は、更新してロックを再生成します:

```
./tools/twoliter/twoliter update
```

ステップ 4: cargo-make パスの問題を回避する

Twoliter は CARGO_HOME を ~/bottlerocket/.cargo に設定します。これにより、内部の Cargo プロセスが、グローバルにインストールされた cargo-make を検索できなくなります。シンボリックリンクを作成します。

```
mkdir -p ~/bottlerocket/.cargo/bin
ln -sf /root/.cargo/bin/cargo-make ~/bottlerocket/.cargo/bin/cargo-make
```

このステップを行わないと、ビルドは error: no such command: make で失敗します。

ステップ 5: 使用可能なドライバーブランチを検索する

NVIDIA kmod パッケージは bottlerocket-kernel-kit 内に含まれています。使用可能なドライバーパッケージの詳細については、GitHub ウェブサイトの「kernel-kit パッケージディレクトリ」、または GitHub ウェブサイトの「リリースノート」を参照してください。

カーネル 6.18 (aws-k8s-1.36 バリアントで使用) の場合:

```
kmod-6.18-nvidia-r580   # driver 580.159.03 (current default)
kmod-6.18-nvidia-r595   # driver 595.71.05
```

カーネル 6.12 (aws-k8s-1.33、1.34、1.35 で使用) の場合:

```
kmod-6.12-nvidia-r580
kmod-6.12-nvidia-r595
```

> **Note**
>
> 各 kmod パッケージにはサブパッケージ (-tesla、-open-gpu、-grid、-fabricmanager、-imex) が含まれています。公式 Bottlerocket NVIDIA バリアントは -tesla を参照します。これは、RPM 依存関係を介して必要なすべてのサブパッケージをプルします。起動時に、Bottlerocket はインスタンスタイプに基づいて適切なドライバーフレーバーを自動的に選択します。

ステップ 6: 新しいバリアントを作成する

既存のバリアントをコピーします。シンボリックリンクを保持するには、cp -R を使用します:

```
cp -R variants/aws-k8s-1.36-nvidia variants/aws-k8s-1.36-nvidia-595
cp -R sources/settings-defaults/aws-k8s-1.36-nvidia \
      sources/settings-defaults/aws-k8s-1.36-nvidia-595
```

`variants/aws-k8s-1.36-nvidia-595/Cargo.toml` を編集します:

```
- name = "aws-k8s-1_36-nvidia"
+ name = "aws-k8s-1_36-nvidia-595"

- "kmod-6.18-nvidia-r580-tesla",
+ "kmod-6.18-nvidia-r595-tesla",
```

`sources/settings-defaults/aws-k8s-1.36-nvidia-595/Cargo.toml` を編集します:

```
- name = "settings-defaults-aws-k8s-1_36-nvidia"
+ name = "settings-defaults-aws-k8s-1_36-nvidia-595"
```

ステップ 7: バリアントを登録する

新しいバリアントを 5 つのファイルに登録します:

1. **`Cargo.toml`** – ワークスペースメンバーを追加します:

```
 "variants/aws-k8s-1.36-nvidia",
+    "variants/aws-k8s-1.36-nvidia-595",
 "variants/aws-k8s-1.36-nvidia-fips",
```

2. **`sources/Cargo.toml`** – settings-defaults メンバーを追加します:

```
 "settings-defaults/aws-k8s-1.36-nvidia",
+    "settings-defaults/aws-k8s-1.36-nvidia-595",
```

3. **`packages/settings-defaults/settings-defaults.spec`** – %package ブロック、両方のビルドループのエントリ、および %files セクションを追加します:

```
%package aws-k8s-1.36-nvidia-595
Summary: Settings defaults for the aws-k8s 1.36 nvidia-595 variant
Requires: %{_cross_os}variant(aws-k8s-1.36-nvidia-595)
Provides: %{_cross_os}settings-defaults(any)
```

```
Provides: %{_cross_os}settings-defaults(aws-k8s-1.36-nvidia-595)
Conflicts: %{_cross_os}settings-defaults(any)

%description aws-k8s-1.36-nvidia-595
%{summary}.
```

両方の `for defaults in` ループに追加します:

```
    aws-k8s-1.36-nvidia \
+   aws-k8s-1.36-nvidia-595 \
    metal-dev \
```

`%files` セクションを追加します:

```
%files aws-k8s-1.36-nvidia-595
%{_cross_defaultsdir}/aws-k8s-1.36-nvidia-595.toml
%{_cross_tmpfilesdir}/storewolf-defaults-aws-k8s-1.36-nvidia-595.conf
```

4. **`packages/settings-plugins/settings-plugins.spec`** – `%package aws-k8s-nvidia` の下に `Provides:` 行を追加します:

```
 Provides: %{_cross_os}settings-plugin(aws-k8s-1.36-nvidia)
+Provides: %{_cross_os}settings-plugin(aws-k8s-1.36-nvidia-595)
 Conflicts: %{_cross_os}settings-plugin(any)
```

ステップ 8: ロックファイルを更新する

ワークスペースメンバーを追加すると、`sources/Cargo.lock` が無効になります。それを更新します:

```
cd ~/bottlerocket/sources
cargo update --workspace
```

> **Important**
>
> `cargo generate-lockfile` を使用しないでください。これにより、ロックファイル全体が書き換えられ、推移的な依存関係がバンプされ、`cargo-deny` バージョンが重複するエラーが発生します。

ステップ 9: NVIDIA ライセンスファイルを作成する

NVIDIA はドライバーソースの再配布を制限します。以下を構築する前に、明示的なライセンス確認を追加する必要があります:

```
cat > ~/bottlerocket/Licenses.toml <<'EOF'
[nvidia]
spdx-id = "LicensesRef-NVIDIA-Customer-Use"
licenses = [
  { path = "LICENSE", license-url = "https://www.nvidia.com/en-us/drivers/nvidia-
license/" }
]
EOF
```

ステップ 10: AMI を構築して発行する

ターゲットリージョンで `Infra.toml` を作成します:

```
cat > ~/bottlerocket/Infra.toml <<'EOF'
[aws]
regions = ["us-west-2", "us-east-1", "us-east-2"]
EOF
```

イメージを構築する:

```
cd ~/bottlerocket
cargo make \
  -e BUILDSYS_VARIANT=aws-k8s-1.36-nvidia-595 \
  -e BUILDSYS_UPSTREAM_SOURCE_FALLBACK=true \
  -e BUILDSYS_UPSTREAM_LICENSE_FETCH=true \
  -e BUILDSYS_JOBS=32
```

最初のビルドでは、SDK イメージとキットイメージ (合計約 2 GB) をプルします。それ以降のビルドは、32 コアホストで 3～5 分かかります。

AMI を発行します:

```
cargo make \
  -e BUILDSYS_VARIANT=aws-k8s-1.36-nvidia-595 \
  -e PUBLISH_REGIONS=us-west-2,us-east-1,us-east-2 \
  ami
```

ビルドは AMI ID を `build/images/x86_64-aws-k8s-1.36-nvidia-595/latest/*-amis.json` に書き込みます。EKS マネージドノードグループ、Karpenter `EC2NodeClass`、または起動テンプレートでこれらを使用します。

構築できるその他の組み合わせ

このチュートリアルでは NVIDIA ドライバーブランチをスワップしますが、アップストリームキットで利用可能なパッケージでも同じアプローチを使用できます。次の例は、キットをフォークせずにアセンブルできる内容を示しています:

必要なもの	バリアント **`Cargo.toml`** の変更点
異なる NVIDIA ドライバーブランチ	`kmod-6.18-nvidia-r580-tesla` → `kmod-6.18-nvidia-r595-tesla`
異なるカーネルバージョン	`kernel-6.18` → `kernel-6.12` (それに応じて kmod を調整する)
NVIDIA を完全に削除する	`included-packages` から 3 つの `nvidia-*` 行を削除する
EFA サポートを追加する	`included-packages` に `kmod-6.18-efa` を追加する
コンテナランタイムバージョンを切り替える	`containerd-2.2` → `containerd-2.1`

利用可能なパッケージの詳細については、GitHub ウェブサイトの「kernel-kit」と GitHub ウェブサイトの「core-kit」を参照してください。

> **Important**
>
> Bottlerocket ルートファイルシステムはイミュータブルです。ランタイム時にパッケージをインストールすることはできません。キット内のすべてのパッケージは、Bottlerocket 専用にクロスコンパイルされています。標準のアップストリーム RPM は機能しません。キットにまだ含まれていないソフトウェアが必要な場合は、ランタイムの代替手段として、GitHub ウェブサイトの「ブートストラップコンテナ」または GitHub ウェブサイトの「ホストコンテナ」を検討してください。

クリーンアップ

ビルドホストが不要になった場合は、継続的な料金が発生しないように EC2 インスタンスを終了してください。AMI はアカウント内で個別に保持されます。不要になった場合は EC2 コンソールまたは CLI で登録解除します。

概要

このトピックでは、別の NVIDIA ドライバーブランチを使用してカスタム Bottlerocket バリアントを作成する方法を示しました。このプロセスには、既存のバリアントのコピー、1 つのパッケージ参照の変更、ワークスペースと RPM 仕様への新しいバリアントの登録、およびビルドの実行が含まれます。カーネルバージョンの切り替え、パッケージの追加、新しい Kubernetes リリース向けのバリアントの作成など、あらゆるカスタマイズに同じアプローチが適用されます。

最適化された Ubuntu Linux AMI を使用してノードを作成する

Canonical は Amazon EKS と提携し、クラスターで使用できるノード AMI を作成しました。

Canonical 社は、特定用途向けの Kubernetes ノード OS イメージ用のビルドを提供します。この最小化された Ubuntu イメージは Amazon EKS に最適化されたものであり、AWS が共同開発を行ったカスタム AWS カーネルが含まれます。詳細については、「Ubuntu on Amazon Elastic Kubernetes Service (EKS)」と「the section called “Ubuntu Linux”」を参照してください。サポートの詳細については、「AWS プレミアムサポートのよくある質問」の「サードパーティソフトウェア」セクションを参照してください。

最適化された Windows AMI を使用してノードを作成する

Windows Amazon EKS 最適化 AMI は Windows Server 2019、Windows Server 2022、Windows Server 2025 上に構築されています。Amazon EKS ノードのベースイメージとして機能するように構成されています。デフォルトでは、AMI には次のコンポーネントが含まれています。

- kubelet
- kube-proxy
- AWS IAM Authenticator for Kubernetes
- csi-proxy
- containerd

> **Note**
>
> Microsoft Security Update Guide を使用して、Windows Server のセキュリティイベントやプライバシーイベントを追跡できます。

Amazon EKS は、次のバリアントで Windows コンテナに最適化された AMI を提供しています。

- Amazon EKS 最適化 Windows Server 2019 Core AMI
- Amazon EKS 最適化 Windows Server 2019 Full AMI
- Amazon EKS 最適化 Windows Server 2022 Core AMI
- Amazon EKS 最適化 Windows Server 2022 Full AMI
- Amazon EKS 最適化 Windows Server 2025 Core AMI
- Amazon EKS 最適化 Windows Server 2025 Full AMI

> **Important**
>
> - Amazon EKS 最適化 Windows Server 20H2 Core AMI は非推奨です。この AMI の新しいバージョンはリリースされません。
> - 最新のセキュリティ更新プログラムがデフォルトで適用されるように、Amazon EKS は直近 4 か月間の最適化 Windows AMI を維持します。新しい AMI はそれぞれ、初回リリースから 4 か月間使用できます。この期間が過ぎると、古い AMI は非公開になり、アクセス不可になります。セキュリティの脆弱性を回避し、サポート期間が終了した古い AMI にアクセスできなくなることを防ぐために、最新の AMI を使用することをお勧めします。非公開になった AMI へのアクセスを可能にできるかは保証できませんが、AWS サポートにチケットを提出することでアクセスをリクエストできます。

リリースカレンダー

以下の表に、Amazon EKS 用の Windows 各バージョンについて、そのリリース日およびサポート終了日を示します。終了日が空白の場合、そのバージョンのサポートは継続されています。

Windows のバージョン	Amazon EKS リリース	Amazon EKS のサポート終了
Windows Server 2025 Core	01/27/2026	
Windows Server 2025 Full	01/27/2026	
Windows Server 2022 Core	10/17/2022	
Windows Server 2022 Full	10/17/2022	
Windows Server 20H2 Core	8/12/2021	8/9/2022
Windows Server 2004 Core	8/19/2020	12/14/2021
Windows Server 2019 Core	10/7/2019	
Windows Server 2019 Full	10/7/2019	
Windows Server 1909 Core	10/7/2019	12/8/2020

ブートストラップスクリプトの設定パラメータ

Windows ノードを作成すると、ノード上にさまざまなパラメータが設定できるスクリプトができます。設定によりますが、このスクリプトはノード上の次のような場所で確認できます。`C:\Program Files\Amazon\EKS\Start-EKSBootstrap.ps1`。カスタムパラメータ値は、ブートストラップスクリプトの引数として指定できます。例えば、起動テンプレートのユーザーデータを更新できます。詳細については、「the section called “Amazon EC2 ユーザーデータ”」を参照してください。

このスクリプトには次のコマンドラインパラメータが含まれます。

- `-EKSClusterName` - このワーカーノードを参加させるための Amazon EKS クラスター名を指定します。
- `-KubeletExtraArgs` - `kubelet` の追加引数を指定します (オプション)。
- `-KubeProxyExtraArgs` - `kube-proxy` の追加引数を指定します (オプション)。
- `-APIServerEndpoint` — Amazon EKS クラスター API サーバーエンドポイントを指定します (オプション)。`-Base64ClusterCA` と共に使用した場合のみ有効となります。`Get-EKSCluster` の呼び出しをバイパスします。

- -Base64ClusterCA — Base64 でエンコードされたクラスターの CA の内容を指定します (オプション)。-APIServerEndpoint と共に使用した場合のみ有効となります。Get-EKSCluster の呼び出しをバイパスします。
- -DNSClusterIP — クラスター内の DNS クエリに使用する IP アドレスを上書きします (オプション)。プライマリインターフェイスの IP アドレスに基づく 10.100.0.10 または 172.20.0.10 のデフォルトです。
- -ServiceCIDR — クラスターサービスの宛先となる Kubernetes サービスの IP アドレス範囲をオーバーライドします。プライマリインターフェイスの IP アドレスに基づく 172.20.0.0/16 または 10.100.0.0/16 のデフォルトです。
- -ExcludedSnatCIDRs — 送信元ネットワークアドレス変換 (SNAT) から除外する IPv4 CIDR のリストです。つまり、VPC アドレスで呼び出し可能なポッドプライベート IP は、アウトバウンドトラフィック用のインスタンス ENI のプライマリ IPv4 アドレスの IP アドレスに変換されません。デフォルトでは、Amazon EKS Windows ノードの VPC の IPv4 CIDR が追加されます。このパラメータに CIDR を指定すると、指定した CIDR も除外されます。詳細については、「the section called “アウトバウンドトラフィック”」を参照してください。
- -EnableCimFS – (オプション) Composite Image File System (CimFS) containerd スナップショッターをオンにします。これは、Amazon EKS 最適化 Windows Server 2025 AMI の Kubernetes バージョン 1.35 以降でのみ使用できるオプトイン機能です。詳細については、「the section called “CimFS スナップショッター”」を参照してください。

コマンドラインパラメータに加えて、いくつかの環境変数パラメータを指定することもできます。コマンドラインパラメータを指定する場合、そのパラメータはそれぞれの環境変数よりも優先されます。ブートストラップスクリプトはマシンスコープの変数のみを読み取るため、環境変数はマシン (またはシステム) スコープとして定義する必要があります。

このスクリプトでは、次の環境変数が考慮されます。

- SERVICE_IPV4_CIDR — 定義については、ServiceCIDR コマンドラインパラメータを参照してください。
- EXCLUDED_SNAT_CIDRS — コンマ区切りの文字列にする必要があります。定義については ExcludedSnatCIDRs コマンドラインパラメータを参照してください。
- EKS_ENABLE_CIMFS – CimFS containerd スナップショッターをオンにするには、true または 1 に設定します。定義については -EnableCimFS コマンドラインパラメータを参照してください。

gMSA 認証サポート

Amazon EKS Windows Pod では、さまざまなタイプのグループ管理サービスアカウント (gMSA) 認証が可能です。

- Amazon EKS は認証用の Active Directory ドメインアイデンティティをサポートしています。ドメイン参加型 gMSA の詳細については、AWS ブログの「Windows Authentication on Amazon EKS Windowspods」を参照してください。
- Amazon EKS では、非ドメイン参加型 Windows ノードがポータブルなユーザーアイデンティティを使用して gMSA 認証情報を取得できるようにするプラグインが提供されています。ドメインレス gMSA の詳細については、AWS ブログの「Domainless Windows Authentication for Amazon EKS Windowspods」を参照してください。

キャッシュされたコンテナイメージ

Amazon EKS Windows 最適化 AMI には、`containerd` のランタイムのためにキャッシュされたコンテナイメージが含まれます。コンテナイメージは、Amazon 管理ビルドコンポーネントを使用してカスタム AMI を構築するときにキャッシュされます。詳細については、「the section called “Amazon 管理ビルドコンポーネントを使用する”」を参照してください。

`containerd` ランタイムのキャッシュされたコンテナイメージは以下のとおりです。

- `amazonaws.com/eks/pause-windows`
- `mcr.microsoft.com/windows/nanoserver`
- `mcr.microsoft.com/windows/servercore`

CimFS スナップショッター

Amazon EKS に最適化された Windows Server 2025 AMI には、Microsoft Learn ウェブサイトで説明されているように、Composite Image File System (CimFS) `containerd` スナップショッターのサポートが含まれています。CimFS はコンテナイメージレイヤーを `.cim` アーカイブとして保存するため、大規模な Windows コンテナイメージのコールドイメージプル時間を短縮できます。

CimFS はオプトイン機能です。Kubernetes バージョン `1.35` 以降の Amazon EKS 最適化 Windows Server 2025 AMI (Core および Full) でのみ使用できます。デフォルトでは、CimFS は無効化されています。

> **Windows Server のバージョン要件**
>
> CimFS は Windows Server 2025 (ビルド 26100) 以降でのみサポートされています。有効にしても、以前の Windows Server バージョンには影響しません。これらの AMI には CimFS containerd 設定が存在しないためです。

AMI には、デフォルト設定とともに、事前構築済みの CimFS containerd 設定 (config-cimfs.toml) が付属しています。CimFS を有効にすると、ブートストラップスクリプトは CimFS 設定にスワップし、containerd を再起動し、CimFS スナップショッターを使用して一時停止およびベースコンテナイメージを再キャッシュします。

ノードの CimFS を有効にするには、次のいずれかの方法を使用します (the section called “ブートストラップスクリプトの設定パラメータ” を参照)。

- -EnableCimFS スイッチを Start-EKSBootstrap.ps1 ブートストラップスクリプトに渡します。
- ブートストラップスクリプトを実行する前に、EKS_ENABLE_CIMFS マシンスコープの環境変数を true または 1 に設定します。

次の例は、起動テンプレートのユーザーデータから CimFS を有効にする方法を示しています。

```
<powershell>
[string]$EKSBootstrapScriptFile = "$env:ProgramFiles\Amazon\EKS\Start-EKSBootstrap.ps1"
& $EKSBootstrapScriptFile -EKSClusterName "my-cluster" -EnableCimFS 3>&1 4>&1 5>&1 6>&1
</powershell>
```

ストリームリダイレクト (3>&1 4>&1 5>&1 6>&1) を使用すると、トラブルシューティングのために、ブートストラップスクリプトの警告、詳細、デバッグ、および情報出力をインスタンスのユーザーデータログで確認できます。

詳細情報

Amazon EKS 最適化 Windows AMI の使用に関する詳細については、以下のセクションを参照してください。

- Amazon EKS 最適化アクセラレーテッド Windows AMI でのワークロードの実行に関する詳細については、「the section called “Windows GPU AMI のセットアップ”」を参照してください。

- Windows をマネージド型ノードグループと使用するには、「the section called “マネージド型ノードグループ”」を参照してください。
- セルフマネージド型 Windows ノードを起動するには、「the section called “Windows”」を参照してください。
- バージョンについては、「the section called “バージョン情報を取得する”」を参照してください。
- Amazon EKS 最適化 Windows AMI の最新の ID を取得するには、「the section called “最新の ID を取得する”」を参照してください。
- Amazon EC2 Image Builder を使用して、カスタムの Amazon EKS 最適化 Windows AMI を作成するには、「the section called “カスタムビルド”」を参照してください。
- ベストプラクティスについては、「EKS ベストプラクティスガイド」の「Amazon EKS 最適化 Windows AMI の管理」を参照してください。

`eksctl` を使用してセルフマネージド Windows Server 2022 ノードを作成する

次の `test-windows-2022.yaml` は、セルフマネージド Windows Server 2022 ノードを作成するためのリファレンスとして使用できます。各`#####`は独自の値に置き換えます。

> **Note**
>
> セルフマネージド Windows Server 2022 ノードを実行するには、`eksctl` バージョン 0.116.0 以降を使用する必要があります。

```
apiVersion: eksctl.io/v1alpha5
kind: ClusterConfig

metadata:
  name: windows-2022-cluster
  region: region-code
  version: '1.37'

nodeGroups:
  - name: windows-ng
    instanceType: m5.2xlarge
    amiFamily: WindowsServer2022FullContainer
    volumeSize: 100
    minSize: 2
    maxSize: 3
```

```
  - name: linux-ng
    amiFamily: AmazonLinux2
    minSize: 2
    maxSize: 3
```

ノードグループは、次のコマンドを使用して作成できます。

```
eksctl create cluster -f test-windows-2022.yaml
```

Windows AMI バージョンに関する情報を取得する

このトピックでは、Amazon EKS 最適化 Windows AMI のバージョンと、それに対応する kubelet、containerd、および csi-proxy のバージョンを示します。

AMI ID を含む、Amazon EKS 最適化 AMI のメタデータでは、各バリアントをプログラム的に取得することができます。詳細については、「the section called “最新の ID を取得する”」を参照してください。

各 AMI は次の形式により、Kubernetes のバージョンならびに AMI のリリース日によってバージョン付けされています。

```
k8s_major_version.k8s_minor_version-release_date
```

> **Note**
>
> Amazon EKS マネージド型ノードグループは、Windows AMI の 2022 年 11 月以降のリリースをサポートしています。

この特定のドキュメントページへのすべてのソースファイルの変更通知を受け取るには、RSS リーダーを使用して次の URL をサブスクライブできます。

```
https://github.com/awsdocs/amazon-eks-user-guide/commits/mainline/latest/ug/nodes/eks-ami-versions-windows.adoc.atom
```

Amazon EKS 最適化 Windows Server 2025 Core AMI

次の表は、Amazon EKS 最適化 Windows Server 2025 Core AMI の現在および以前のバージョンを示しています。

Example

Kubernetes version 1.36

AMI のバージョン	kubelet バージョン	containerd バージョン	csi-proxy バージョン	リリースノート
1.36-2026.09.14	1.36.3	2.2.3	1.2.1	CVE-2026-76654 のパッチを含む
1.36-2026.08.15	1.36.2	2.2.3	1.2.1	
1.36-2026.07.18	1.36.2	2.2.3	1.2.1	
1.36-2026.06.18	1.36.2	2.2.3	1.2.1	
1.36-2026.05.19	1.36.1	2.2.3	1.2.1	

Kubernetes version 1.35

AMI のバージョン	kubelet バージョン	containerd バージョン	csi-proxy バージョン	リリースノート
1.35-2026.09.14	1.35.7	2.1.7	1.2.1	CVE-2026-76654 のパッチを含む
1.35-2026.08.15	1.35.6	2.1.7	1.2.1	
1.35-2026.07.18	1.35.6	2.1.7	1.2.1	

AMI のバージョン	kubelet バージョン	containerd バージョン	csi-proxy バージョン	リリースノート
1.35-2026.06.18	1.35.6	2.1.7	1.2.1	
1.35-2026.05.19	1.35.5	2.1.7	1.2.1	
1.35-2026.04.18	1.35.3	2.1.6	1.2.1	
1.35-2026.02.16	1.35.0	2.1.6	1.2.1	
1.35-2026-01-22	1.35.0	2.1.6	1.2.1	

Amazon EKS 最適化 Windows Server 2025 Full AMI

次の表は、Amazon EKS 最適化 Windows Server 2025 Full AMI の現在および以前のバージョンを示しています。

Example

Kubernetes version 1.36

AMI のバージョン	kubelet バージョン	containerd バージョン	csi-proxy バージョン	リリースノート
1.36-2026.09.14	1.36.3	2.2.3	1.2.1	CVE-2026-76654 のパッチを含む
1.36-2026.08.15	1.36.2	2.2.3	1.2.1	
1.36-2026.07.18	1.36.2	2.2.3	1.2.1	

AMI のバージョン	kubelet バージョン	containerd バージョン	csi-proxy バージョン	リリースノート
1.36-2026.06.18	1.36.2	2.2.3	1.2.1	
1.36-2026.05.19	1.36.1	2.2.3	1.2.1	

Kubernetes version 1.35

AMI のバージョン	kubelet バージョン	containerd バージョン	csi-proxy バージョン	リリースノート
1.35-2026.09.14	1.35.7	2.1.7	1.2.1	CVE-2026-76654 のパッチを含む
1.35-2026.08.15	1.35.6	2.1.7	1.2.1	
1.35-2026.07.21	1.35.6	2.1.7	1.2.1	
1.35-2026.06.18	1.35.6	2.1.7	1.2.1	
1.35-2026.05.19	1.35.5	2.1.7	1.2.1	
1.35-2026.04.18	1.35.3	2.1.6	1.2.1	
1.35-2026.02.16	1.35.0	2.1.6	1.2.1	
1.35-2026-01-22	1.35.0	2.1.6	1.2.1	

Amazon EKS 最適化 Windows Server 2022 Core AMI

次の表は、Amazon EKS 最適化 Windows Server 2022 Core AMI の現在および以前のバージョンを示しています。

Example

Kubernetes version 1.36

AMI のバージョン	kubelet バージョン	containerd バージョン	csi-proxy バージョン	リリースノート
1.36-2026.09.14	1.36.3	2.2.3	1.2.1	CVE-2026-76654 のパッチを含む
1.36-2026.08.15	1.36.2	2.2.3	1.2.1	
1.36-2026.07.18	1.36.2	2.2.3	1.2.1	
1.36-2026.06.18	1.36.2	2.2.3	1.2.1	
1.36-2026.05.19	1.36.1	2.2.3	1.2.1	

Kubernetes version 1.35

AMI のバージョン	kubelet バージョン	containerd バージョン	csi-proxy バージョン	リリースノート
1.35-2026.09.14	1.35.7	2.1.7	1.2.1	CVE-2026-76654 のパッチを含む

AMI のバージョン	kubelet バージョン	containerd バージョン	csi-proxy バージョン	リリースノート
1.35-2026.08.15	1.35.6	2.1.7	1.2.1	
1.35-2026.07.21	1.35.6	2.1.7	1.2.1	
1.35-2026.06.18	1.35.6	2.1.7	1.2.1	
1.35-2026.05.19	1.35.5	2.1.7	1.2.1	
1.35-2026.04.18	1.35.3	2.1.6	1.2.1	
1.35-2026.02.16	1.35.0	2.1.6	1.2.1	
1.35-2026-01-22	1.35.0	2.1.6	1.2.1	

Kubernetes version 1.34

AMI のバージョン	kubelet バージョン	containerd バージョン	csi-proxy バージョン	リリースノート
1.34-2026.09.14	1.34.10	2.1.7	1.2.1	CVE-2026-76654 のパッチを含む
1.34-2026.08.15	1.34.9	2.1.7	1.2.1	
1.34-2026.07.18	1.34.9	2.1.7	1.2.1	

AMI のバージョン	kubelet バージョン	containerd バージョン	csi-proxy バージョン	リリースノート
1.34-2026.06.18	1.34.9	2.1.6	1.2.1	
1.34-2026.05.19	1.34.8	2.1.6	1.2.1	
1.34-2026.04.18	1.34.6	2.1.6	1.2.1	
1.34-2026.02.13	1.34.3	2.1.6	1.2.1	
1.34-2026.01.22	1.34.2	2.1.6	1.2.1	containerd を 2.1.6 にアップグレードしました。
1.34-2025.12.15	1.34.2	2.1.4	1.2.1	
1.34-2025.11.14	1.34.1	2.1.4	1.2.1	
1.34-2025.10.18	1.34.1	2.1.4	1.2.1	
1.34-2025.09.13	1.34.0	2.1.4	1.2.1	

Kubernetes version 1.33

AMI のバージョン	kubelet バージョン	containerd バージョン	csi-proxy バージョン	リリースノート
1.33-2026.09.14	1.33.13	1.7.30	1.2.1	CVE-2026-76654 のパッチを含む
1.33-2026.08.15	1.33.13	1.7.30	1.2.1	
1.33-2026.07.18	1.33.13	1.7.30	1.2.1	
1.33-2026.06.18	1.33.13	1.7.30	1.2.1	
1.33-2026.05.19	1.33.11	1.7.30	1.2.1	
1.33-2026.04.18	1.33.10	1.7.30	1.2.1	
1.33-2026.02.13	1.33.7	1.7.30	1.2.1	
1.33-2026.01.22	1.33.5	1.7.30	1.2.1	containerd を 1.7.30 にアップグレードしました。
1.33-2025.12.15	1.33.5	1.7.28	1.2.1	
1.33-2025.11.14	1.33.5	1.7.28	1.2.1	

AMI のバージョン	kubelet バージョン	containerd バージョン	csi-proxy バージョン	リリースノート
1.33-2025.10.18	1.33.5	1.7.28	1.2.1	
1.33-2025.09.13	1.33.4	1.7.28	1.2.1	GMSA プラグインログを Windows イベントに変更
1.33-2025.08.18	1.33.3	1.7.27	1.2.1	
1.33-2025.07.16	1.33.1	1.7.27	1.2.1	
1.33-2025.06.13	1.33.1	1.7.27	1.2.1	
1.33-2025.05.17	1.33.1	1.7.27	1.2.1	

Kubernetes version 1.32

AMI のバージョン	kubelet バージョン	containerd バージョン	csi-proxy バージョン	リリースノート
1.32-2026.09.14	1.32.13	1.7.30	1.2.1	CVE-2026-76654 のパッチを含む
1.32-2026.08.15	1.32.13	1.7.30	1.2.1	
1.32-2026.07.18	1.32.13	1.7.30	1.2.1	

AMI のバージョン	kubelet バージョン	containerd バージョン	csi-proxy バージョン	リリースノート
1.32-2026.06.18	1.32.13	1.7.30	1.2.1	
1.32-2026.05.19	1.32.13	1.7.30	1.2.1	
1.32-2026.04.18	1.32.13	1.7.30	1.2.1	
1.32-2026.02.13	1.32.11	1.7.30	1.2.1	
1.32-2026.01.22	1.32.9	1.7.30	1.2.1	containerd を 1.7.30 にアップグレードしました。
1.32-2025.12.15	1.32.9	1.7.28	1.2.1	
1.32-2025.11.14	1.32.9	1.7.28	1.2.1	
1.32-2025.10.18	1.32.9	1.7.28	1.2.1	
1.32-2025.09.13	1.32.8	1.7.28	1.2.1	GMSA プラグインログを Windows イベントに変更
1.32-2025.08.18	1.32.7	1.7.27	1.2.1	
1.32-2025.07.16	1.32.5	1.7.27	1.2.1	

AMI のバージョン	kubelet バージョン	containerd バージョン	csi-proxy バージョン	リリースノート
1.32-2025.06.13	1.32.5	1.7.27	1.2.1	containerd を 1.7.27 にアップグレードしました。
1.32-2025.05.17	1.32.5	1.7.20	1.2.1	
1.32-2025.04.14	1.32.1	1.7.20	1.1.3	
1.32-2025.03.14	1.32.1	1.7.20	1.1.3	
1.32-2025.02.18	1.32.1	1.7.20	1.1.3	
1.32-2025.01.15	1.32.0	1.7.20	1.1.3	

Kubernetes version 1.31

AMI のバージョン	kubelet バージョン	containerd バージョン	csi-proxy バージョン	リリースノート
1.31-2026.09.14	1.31.14	1.7.30	1.2.1	CVE-2026-76654 のパッチを含む
1.31-2026.08.15	1.31.14	1.7.30	1.2.1	
1.31-2026.07.18	1.31.14	1.7.30	1.2.1	

AMI のバージョン	kubelet バージョン	containerd バージョン	csi-proxy バージョン	リリースノート
1.31-2026.06.18	1.31.14	1.7.30	1.2.1	
1.31-2026.05.19	1.31.14	1.7.30	1.2.1	
1.31-2026.04.18	1.31.14	1.7.30	1.2.1	
1.31-2026.02.13	1.31.14	1.7.30	1.2.1	
1.31-2026.01.22	1.31.13	1.7.30	1.2.1	containerd を 1.7.30 にアップグレードしました。
1.31-2025.12.15	1.31.13	1.7.28	1.2.1	
1.31-2025.11.14	1.31.13	1.7.28	1.2.1	
1.31-2025.10.18	1.31.13	1.7.28	1.2.1	
1.31-2025.09.13	1.31.12	1.7.28	1.2.1	GMSA プラグインログを Windows イベントに変更
1.31-2025.08.18	1.31.11	1.7.27	1.2.1	
1.31-2025.07.16	1.31.9	1.7.27	1.2.1	

AMI のバージョン	kubelet バージョン	containerd バージョン	csi-proxy バージョン	リリースノート
1.31-2025.06.13	1.31.9	1.7.27	1.2.1	containerd を 1.7.27 にアップグレードしました。
1.31-2025.05.17	1.31.9	1.7.20	1.2.1	
1.31-2025.04.14	1.31.5	1.7.20	1.1.3	
1.31-2025.03.14	1.31.5	1.7.20	1.1.3	
1.31-2025.02.15	1.31.5	1.7.20	1.1.3	
1.31-2025.01.15	1.31.4	1.7.20	1.1.3	
1.31-2025.01.01	1.31.4	1.7.20	1.1.3	CVE-2024-9042 のパッチが含まれます。
1.31-2024.12.13	1.31.3	1.7.20	1.1.3	
1.31-2024.11.12	1.31.1	1.7.20	1.1.3	
1.31-2024.10.08	1.31.1	1.7.20	1.1.3	
1.31-2024.10.01	1.31.1	1.7.20	1.1.3	

AMI のバージョン	kubelet バージョン	containerd バージョン	csi-proxy バージョン	リリースノート
1.31-2024.09.10	1.31.0	1.7.20	1.1.3	

Kubernetes version 1.30

AMI のバージョン	kubelet バージョン	containerd バージョン	csi-proxy バージョン	リリースノート
1.30-2026.07.18	1.30.14	1.7.30	1.2.1	
1.30-2026.06.18	1.30.14	1.7.30	1.2.1	
1.30-2026.05.19	1.30.14	1.7.30	1.2.1	
1.30-2026.04.18	1.30.14	1.7.30	1.2.1	
1.30-2026.02.13	1.30.14	1.7.30	1.2.1	
1.30-2026.01.22	1.30.14	1.7.30	1.2.1	containerd を 1.7.30 にアップグレードしました。
1.30-2025.12.15	1.30.14	1.7.28	1.2.1	
1.30-2025.11.21	1.30.14	1.7.28	1.2.1	

AMI のバージョン	kubelet バージョン	containerd バージョン	csi-proxy バージョン	リリースノート
1.30-2025.10.18	1.30.14	1.7.28	1.2.1	
1.30-2025.09.13	1.30.14	1.7.28	1.2.1	GMSA プラグインログを Windows イベントに変更
1.30-2025.08.18	1.30.14	1.7.27	1.2.1	
1.30-2025.07.16	1.30.13	1.7.27	1.2.1	
1.30-2025.06.13	1.30.13	1.7.27	1.2.1	containerd を 1.7.27 にアップグレードしました。
1.30-2025.05.17	1.30.13	1.7.20	1.2.1	
1.30-2025.04.14	1.30.9	1.7.20	1.1.3	
1.30-2025.03.14	1.30.9	1.7.20	1.1.3	containerd を 1.7.20 にアップグレードしました。
1.30-2025.02.15	1.30.9	1.7.14	1.1.3	
1.30-2025.01.15	1.30.8	1.7.14	1.1.3	

AMI のバージョン	kubelet バージョン	containerd バージョン	csi-proxy バージョン	リリースノート
1.30-2025.01.01	1.30.8	1.7.14	1.1.3	CVE-2024-9042 のパッチが含まれます。
1.30-2024.12.11	1.30.7	1.7.14	1.1.3	
1.30-2024.11.12	1.30.4	1.7.14	1.1.3	
1.30-2024.10.08	1.30.4	1.7.14	1.1.3	
1.30-2024.09.10	1.30.2	1.7.14	1.1.3	
1.30-2024.08.13	1.30.2	1.7.14	1.1.3	
1.30-2024.07.10	1.30.2	1.7.14	1.1.2	CVE-2024-5321 のパッチが含まれます。
1.30-2024.06.17	1.30.0	1.7.14	1.1.2	containerd を 1.7.14 にアップグレードしました。
1.30-2024.05.15	1.30.0	1.6.28	1.1.2	

Kubernetes version 1.29

AMI のバージョン	kubelet バージョン	containerd バージョン	csi-proxy バージョン	リリースノート
1.29-2026.02.13	1.29.15	1.7.30	1.2.1	
1.29-2026.01.22	1.29.15	1.7.30	1.2.1	containerd を 1.7.30 にアップグレードしました。
1.29-2025.12.15	1.29.15	1.7.28	1.2.1	
1.29-2025.11.14	1.29.15	1.7.28	1.2.1	
1.29-2025.10.18	1.29.15	1.7.28	1.2.1	
1.29-2025.09.13	1.29.15	1.7.28	1.2.1	GMSA プラグインログを Windows イベントに変更
1.29-2025.08.18	1.29.15	1.7.27	1.2.1	
1.29-2025.07.16	1.29.15	1.7.27	1.2.1	
1.29-2025.06.13	1.29.15	1.7.27	1.2.1	containerd を 1.7.27 にアップグレードしました。

AMI のバージョン	kubelet バージョン	containerd バージョン	csi-proxy バージョン	リリースノート
1.29-2025.05.17	1.29.15	1.7.20	1.2.1	
1.29-2025.04.14	1.29.13	1.7.20	1.1.3	
1.29-2025.03.14	1.29.13	1.7.20	1.1.3	containerd を 1.7.20 にアップグレードしました。
1.29-2025.02.15	1.29.13	1.7.14	1.1.3	
1.29-2025.01.15	1.29.12	1.7.14	1.1.3	
1.29-2025.01.01	1.29.12	1.7.14	1.1.3	CVE-2024-9042 のパッチが含まれます。
1.29-2024.12.11	1.29.10	1.7.14	1.1.3	
1.29-2024.11.12	1.29.8	1.7.14	1.1.3	
1.29-2024.10.08	1.29.8	1.7.14	1.1.3	
1.29-2024.09.10	1.29.6	1.7.14	1.1.3	
1.29-2024.08.13	1.29.6	1.7.14	1.1.3	

AMI のバージョン	kubelet バージョン	containerd バージョン	csi-proxy バージョン	リリースノート
1.29-2024.07.10	1.29.6	1.7.11	1.1.2	CVE-2024-5321 のパッチが含まれます。
1.29-2024.06.17	1.29.3	1.7.11	1.1.2	
1.29-2024.05.15	1.29.3	1.7.11	1.1.2	containerd を 1.7.11 にアップグレードしました。kubelet を 1.29.3 アップグレードしました。
1.29-2024.04.09	1.29.0	1.6.28	1.1.2	containerd を 1.6.28 にアップグレードしました。golang 1.22.1 を使用して CNI と csi-proxy が再構築されました。
1.29-2024.03.12	1.29.0	1.6.25	1.1.2	
1.29-2024.02.13	1.29.0	1.6.25	1.1.2	

AMI のバージョン	kubelet バージョン	containerd バージョン	csi-proxy バージョン	リリースノート
1.29-2024.02.06	1.29.0	1.6.25	1.1.2	一時停止画像が kubelet ガベージコレクション処理によって誤って削除されてしまうバグを修正しました。
1.29-2024.01.11	1.29.0	1.6.18	1.1.2	Windows Server 2022 Core AMI のスタンドアロン Windows Update KB5034439 は除外しました。KB は、Amazon EKS 最適化 Windows AMI に含まれていない、個別の WinRE パーティションのある Windows インストールにのみ適用されます。

Kubernetes version 1.28

AMI のバージョン	kubelet バージョン	containerd バージョン	csi-proxy バージョン	リリースノート
1.28-2025.11.14	1.28.15	1.7.28	1.2.1	
1.28-2025.10.18	1.28.15	1.7.28	1.2.1	
1.28-2025.09.13	1.28.15	1.7.28	1.2.1	GMSA プラグインログを Windows イベントに変更
1.28-2025.08.18	1.28.15	1.7.27	1.2.1	
1.28-2025.07.16	1.28.15	1.7.27	1.2.1	
1.28-2025.06.13	1.28.15	1.7.27	1.2.1	containerd を 1.7.27 にアップグレードしました。
1.28-2025.05.17	1.28.15	1.7.20	1.2.1	
1.28-2025.04.14	1.28.15	1.7.20	1.1.3	
1.28-2025.03.14	1.28.15	1.7.20	1.1.3	containerd を 1.7.20 にアップグレードしました。

AMI のバージョン	kubelet バージョン	containerd バージョン	csi-proxy バージョン	リリースノート
1.28-2025.02.15	1.28.15	1.7.14	1.1.3	
1.28-2025.01.15	1.28.15	1.7.14	1.1.3	
1.28-2025-01-01	1.28.15	1.7.14	1.1.3	CVE-2024-9042 のパッチが含まれます。
1.28-2024.12.11	1.28.15	1.7.14	1.1.3	
1.28-2024.11.12	1.28.13	1.7.14	1.1.3	
1.28-2024.10.08	1.28.13	1.7.14	1.1.3	
1.28-2024.09.10	1.28.11	1.7.14	1.1.3	
1.28-2024.08.13	1.28.11	1.7.14	1.1.3	
1.28-2024.07.10	1.28.11	1.7.11	1.1.2	CVE-2024-5321 のパッチが含まれます。
1.28-2024.06.17	1.28.8	1.7.11	1.1.2	containerd を 1.7.11 にアップグレードしました。

AMI のバージョン	kubelet バージョン	containerd バージョン	csi-proxy バージョン	リリースノート
1.28-2024.05.14	1.28.8	1.6.28	1.1.2	containerd を 1.6.28 にアップグレードしました。kubelet を 1.28.8 にアップグレードしました。
1.28-2024.04.09	1.28.5	1.6.25	1.1.2	containerd を 1.6.25 にアップグレードしました。golang 1.22.1 を使用して CNI と csi-proxy が再構築されました。
1.28-2024.03.12	1.28.5	1.6.18	1.1.2	
1.28-2024.02.13	1.28.5	1.6.18	1.1.2	

AMI のバージョン	kubelet バージョン	containerd バージョン	csi-proxy バージョン	リリースノート
1.28-2024.01.11	1.28.5	1.6.18	1.1.2	Windows Server 2022 Core AMI のスタンドアロン Windows Update KB5034439 は除外しました。KB は、Amazon EKS 最適化 Windows AMI に含まれていない、個別の WinRE パーティションのある Windows インストールにのみ適用されます。
1.28-2023.12.12	1.28.3	1.6.18	1.1.2	
1.28-2023.11.14	1.28.3	1.6.18	1.1.2	CVE-2023-5528 のパッチが含まれます。

AMI のバージョン	kubelet バージョン	containerd バージョン	csi-proxy バージョン	リリースノート
1.28-2023.10.19	1.28.2	1.6.18	1.1.2	containerd を 1.6.18 にアップグレードしました。新しいブートストラップスクリプト環境変数 (SERVICE_IPV4_CIDR と EXCLUDED_SNAT_CIDRS) を追加しました。
1.28-2023-09.27	1.28.2	1.6.6	1.1.2	kubelet のセキュリティ勧告を修正しました。
1.28-2023.09.12	1.28.1	1.6.6	1.1.2	

Amazon EKS 最適化 Windows Server 2022 Full AMI

次の表は、Amazon EKS 最適化 Windows Server 2022 Full AMI の現在および以前のバージョンを示しています。

Example

Kubernetes version 1.36

AMI のバージョン	kubelet バージョン	containerd バージョン	csi-proxy バージョン	リリースノート
1.36-2026.09.14	1.36.3	2.2.3	1.2.1	CVE-2026-76654 のパッチを含む
1.36-2026.08.15	1.36.2	2.2.3	1.2.1	
1.36-2026.07.18	1.36.2	2.2.3	1.2.1	
1.36-2026.06.18	1.36.2	2.2.3	1.2.1	
1.36-2026.05.19	1.36.1	2.2.3	1.2.1	

Kubernetes version 1.35

AMI のバージョン	kubelet バージョン	containerd バージョン	csi-proxy バージョン	リリースノート
1.35-2026.09.14	1.35.7	2.1.7	1.2.1	CVE-2026-76654 のパッチを含む
1.35-2026.08.15	1.35.6	2.1.7	1.2.1	
1.35-2026.07.18	1.35.6	2.1.7	1.2.1	

AMI のバージョン	kubelet バージョン	containerd バージョン	csi-proxy バージョン	リリースノート
1.35-2026.06.18	1.35.6	2.1.7	1.2.1	
1.35-2026.05.19	1.35.5	2.1.7	1.2.1	
1.35-2026.04.18	1.35.3	2.1.6	1.2.1	
1.35-2026.02.16	1.35.0	2.1.6	1.2.1	
1.35-2026-01-22	1.35.0	2.1.6	1.2.1	

Kubernetes version 1.34

AMI のバージョン	kubelet バージョン	containerd バージョン	csi-proxy バージョン	リリースノート
1.34-2026.09.14	1.34.10	2.1.7	1.2.1	CVE-2026-76654 のパッチを含む
1.34-2026.08.15	1.34.9	2.1.7	1.2.1	
1.34-2026.07.18	1.34.9	2.1.7	1.2.1	
1.34-2026.06.18	1.34.9	2.1.6	1.2.1	
1.34-2026.05.19	1.34.8	2.1.6	1.2.1	

AMI のバージョン	kubelet バージョン	containerd バージョン	csi-proxy バージョン	リリースノート
1.34-2026.04.18	1.34.6	2.1.6	1.2.1	
1.34-2026.02.13	1.34.3	2.1.6	1.2.1	
1.34-2026.01.22	1.34.2	2.1.6	1.2.1	containerd を 2.1.6 にアップグレードしました。
1.34-2025.12.15	1.34.2	2.1.4	1.2.1	
1.34-2025.11.14	1.34.1	2.1.4	1.2.1	
1.34-2025.10.18	1.34.1	2.1.4	1.2.1	
1.34-2025.09.13	1.34.0	2.1.4	1.2.1	

Kubernetes version 1.33

AMI のバージョン	kubelet バージョン	containerd バージョン	csi-proxy バージョン	リリースノート
1.33-2026.09.14	1.33.13	1.7.30	1.2.1	CVE-2026-76654 のパッチを含む
1.33-2026.08.15	1.33.13	1.7.30	1.2.1	

AMI のバージョン	kubelet バージョン	containerd バージョン	csi-proxy バージョン	リリースノート
1.33-2026.07.21	1.33.13	1.7.30	1.2.1	
1.33-2026.06.18	1.33.13	1.7.30	1.2.1	
1.33-2026.05.19	1.33.11	1.7.30	1.2.1	
1.33-2026.04.18	1.33.10	1.7.30	1.2.1	
1.33-2026.02.13	1.33.7	1.7.30	1.2.1	
1.33-2026.01.22	1.33.5	1.7.30	1.2.1	containerd を 1.7.30 にアップグレードしました。
1.33-2025.12.15	1.33.5	1.7.28	1.2.1	
1.33-2025.11.14	1.33.5	1.7.28	1.2.1	
1.33-2025.10.18	1.33.5	1.7.28	1.2.1	
1.33-2025.09.13	1.33.4	1.7.28	1.2.1	GMSA プラグインログを Windows イベントに変更
1.33-2025.08.18	1.33.3	1.7.27	1.2.1	

AMI のバージョン	kubelet バージョン	containerd バージョン	csi-proxy バージョン	リリースノート
1.33-2025.07.16	1.33.1	1.7.27	1.2.1	
1.33-2025.06.13	1.33.1	1.7.27	1.2.1	
1.33-2025.05.17	1.33.1	1.7.27	1.2.1	

Kubernetes version 1.32

AMI のバージョン	kubelet バージョン	containerd バージョン	csi-proxy バージョン	リリースノート
1.32-2026.09.14	1.32.13	1.7.30	1.2.1	CVE-2026-76654 のパッチを含む
1.32-2026.08.15	1.32.13	1.7.30	1.2.1	
1.32-2026.07.18	1.32.13	1.7.30	1.2.1	
1.32-2026.06.18	1.32.13	1.7.30	1.2.1	
1.32-2026.05.19	1.32.13	1.7.30	1.2.1	
1.32-2026.04.18	1.32.13	1.7.30	1.2.1	
1.32-2026.02.13	1.32.11	1.7.30	1.2.1	

AMI のバージョン	kubelet バージョン	containerd バージョン	csi-proxy バージョン	リリースノート
1.32-2026.01.22	1.32.9	1.7.30	1.2.1	containerd を 1.7.30 にアップグレードしました。
1.32-2025.12.15	1.32.9	1.7.28	1.2.1	
1.32-2025.11.14	1.32.9	1.7.28	1.2.1	
1.32-2025.10.18	1.32.9	1.7.28	1.2.1	
1.32-2025.09.13	1.32.8	1.7.28	1.2.1	GMSA プラグインログを Windows イベントに変更
1.32-2025.08.18	1.32.7	1.7.27	1.2.1	
1.32-2025.07.16	1.32.5	1.7.27	1.2.1	
1.32-2025.06.13	1.32.5	1.7.27	1.2.1	containerd が 1.7.27 にアップグレードされました
1.32-2025.05.17	1.32.5	1.7.20	1.2.1	
1.32-2025.04.14	1.32.1	1.7.20	1.1.3	

AMI のバージョン	kubelet バージョン	containerd バージョン	csi-proxy バージョン	リリースノート
1.32-2025.03.14	1.32.1	1.7.20	1.1.3	
1.32-2025.02.18	1.32.1	1.7.20	1.1.3	
1.32-2025.01.01	1.32.0	1.7.20	1.1.3	

Kubernetes version 1.31

AMI のバージョン	kubelet バージョン	containerd バージョン	csi-proxy バージョン	リリースノート
1.31-2026.09.14	1.31.14	1.7.30	1.2.1	CVE-2026-76654 のパッチを含む
1.31-2026.08.15	1.31.14	1.7.30	1.2.1	
1.31-2026.07.18	1.31.14	1.7.30	1.2.1	
1.31-2026.06.18	1.31.14	1.7.30	1.2.1	
1.31-2026.05.19	1.31.14	1.7.30	1.2.1	
1.31-2026.04.18	1.31.14	1.7.30	1.2.1	
1.31-2026.02.13	1.31.14	1.7.30	1.2.1	

AMI のバージョン	kubelet バージョン	containerd バージョン	csi-proxy バージョン	リリースノート
1.31-2026.01.22	1.31.13	1.7.30	1.2.1	containerd を 1.7.30 にアップグレードしました。
1.31-2025.12.15	1.31.13	1.7.28	1.2.1	
1.31-2025.11.14	1.31.13	1.7.28	1.2.1	
1.31-2025.10.18	1.31.13	1.7.28	1.2.1	
1.31-2025.09.13	1.31.12	1.7.28	1.2.1	GMSA プラグインログを Windows イベントに変更
1.31-2025.08.18	1.31.11	1.7.27	1.2.1	
1.31-2025.07.16	1.31.9	1.7.27	1.2.1	
1.31-2025.06.13	1.31.9	1.7.27	1.2.1	containerd を 1.7.27 にアップグレードしました。
1.31-2025.05.17	1.31.9	1.7.20	1.2.1	
1.31-2025.04.14	1.31.5	1.7.20	1.1.3	

AMI のバージョン	kubelet バージョン	containerd バージョン	csi-proxy バージョン	リリースノート
1.31-2025.03.14	1.31.5	1.7.20	1.1.3	
1.31-2025.02.15	1.31.5	1.7.20	1.1.3	
1.31-2025.01.15	1.31.4	1.7.20	1.1.3	
1.31-2025.01.01	1.31.4	1.7.20	1.1.3	CVE-2024-9042 のパッチが含まれます。
1.31-2024.12.13	1.31.3	1.7.20	1.1.3	
1.31-2024.11.12	1.31.1	1.7.20	1.1.3	
1.31-2024.10.08	1.31.1	1.7.20	1.1.3	
1.31-2024.10.01	1.31.1	1.7.20	1.1.3	
1.31-2024.09.10	1.31.0	1.7.20	1.1.3	

Kubernetes version 1.30

AMI のバージョン	kubelet バージョン	containerd バージョン	csi-proxy バージョン	リリースノート
1.30-2026.07.18	1.30.14	1.7.30	1.2.1	

AMI のバージョン	kubelet バージョン	containerd バージョン	csi-proxy バージョン	リリースノート
1.30-2026.06.18	1.30.14	1.7.30	1.2.1	
1.30-2026.05.19	1.30.14	1.7.30	1.2.1	
1.30-2026.04.18	1.30.14	1.7.30	1.2.1	
1.30-2026.02.13	1.30.14	1.7.30	1.2.1	
1.30-2026.01.22	1.30.14	1.7.30	1.2.1	containerd を 1.7.30 にアップグレードしました。
1.30-2025.12.15	1.30.14	1.7.28	1.2.1	
1.30-2025.11.21	1.30.14	1.7.28	1.2.1	
1.30-2025.10.18	1.30.14	1.7.28	1.2.1	
1.30-2025.09.13	1.30.14	1.7.28	1.2.1	GMSA プラグインログを Windows イベントに変更
1.30-2025.08.18	1.30.14	1.7.27	1.2.1	
1.30-2025.07.16	1.30.13	1.7.27	1.2.1	

AMI のバージョン	kubelet バージョン	containerd バージョン	csi-proxy バージョン	リリースノート
1.30-2025.06.13	1.30.13	1.7.27	1.2.1	containerd を 1.7.27 にアップグレードしました。
1.30-2025.05.17	1.30.13	1.7.20	1.2.1	
1.30-2025.04.14	1.30.9	1.7.20	1.1.3	
1.30-2025.03.14	1.30.9	1.7.20	1.1.3	containerd を 1.7.20 にアップグレードしました。
1.30-2025.02.15	1.30.9	1.7.14	1.1.3	
1.30-2025.01.15	1.30.8	1.7.14	1.1.3	
1.30-2025.01.01	1.30.8	1.7.14	1.1.3	CVE-2024-9042 のパッチが含まれます。
1.30-2024.12.11	1.30.7	1.7.14	1.1.3	
1.30-2024.11.12	1.30.4	1.7.14	1.1.3	
1.30-2024.10.08	1.30.4	1.7.14	1.1.3	

AMI のバージョン	kubelet バージョン	containerd バージョン	csi-proxy バージョン	リリースノート
1.30-2024.09.10	1.30.2	1.7.14	1.1.3	
1.30-2024.08.13	1.30.2	1.7.14	1.1.3	
1.30-2024.07.10	1.30.2	1.7.14	1.1.2	CVE-2024-5321 のパッチが含まれます。
1.30-2024.06.17	1.30.0	1.7.14	1.1.2	containerd を 1.7.14 にアップグレードしました。
1.30-2024.05.15	1.30.0	1.6.28	1.1.2	

Kubernetes version 1.29

AMI のバージョン	kubelet バージョン	containerd バージョン	csi-proxy バージョン	リリースノート
1.29-2026.02.13	1.29.15	1.7.30	1.2.1	
1.29-2026.01.22	1.29.15	1.7.30	1.2.1	containerd を 1.7.30 にアップグレードしました。
1.29-2025.12.15	1.29.15	1.7.28	1.2.1	

AMI のバージョン	kubelet バージョン	containerd バージョン	csi-proxy バージョン	リリースノート
1.29-2025.11.14	1.29.15	1.7.28	1.2.1	
1.29-2025.10.18	1.29.15	1.7.28	1.2.1	
1.29-2025.09.13	1.29.15	1.7.28	1.2.1	GMSA プラグインログを Windows イベントに変更
1.29-2025.08.18	1.29.15	1.7.27	1.2.1	
1.29-2025.07.16	1.29.15	1.7.27	1.2.1	
1.29-2025.06.13	1.29.15	1.7.27	1.2.1	containerd を 1.7.27 にアップグレードしました。
1.29-2025.05.17	1.29.15	1.7.20	1.2.1	
1.29-2025.04.14	1.29.13	1.7.20	1.1.3	
1.29-2025.03.14	1.29.13	1.7.20	1.1.3	containerd を 1.7.20 にアップグレードしました。
1.29-2025.02.15	1.29.13	1.7.14	1.1.3	

AMI のバージョン	kubelet バージョン	containerd バージョン	csi-proxy バージョン	リリースノート
1.29-2025.01.15	1.29.12	1.7.14	1.1.3	
1.29-2025.01.01	1.29.12	1.7.14	1.1.3	CVE-2024-9042 のパッチが含まれます。
1.29-2024.12.11	1.29.10	1.7.14	1.1.3	
1.29-2024.11.12	1.29.8	1.7.14	1.1.3	
1.29-2024.10.08	1.29.8	1.7.14	1.1.3	
1.29-2024.09.10	1.29.6	1.7.14	1.1.3	
1.29-2024.08.13	1.29.6	1.7.14	1.1.3	
1.29-2024.07.10	1.29.6	1.7.11	1.1.2	CVE-2024-5321 のパッチが含まれます。
1.29-2024.06.17	1.29.3	1.7.11	1.1.2	
1.29-2024.05.15	1.29.3	1.7.11	1.1.2	containerd を 1.7.11 にアップグレードしました。kubelet を 1.29.3 にアップグレードしました。

AMI のバージョン	kubelet バージョン	containerd バージョン	csi-proxy バージョン	リリースノート
1.29-2024.04.09	1.29.0	1.6.28	1.1.2	containerd を 1.6.28 にアップグレードしました。golang 1.22.1 を使用して CNI と csi-proxy が再構築されました。
1.29-2024.03.12	1.29.0	1.6.25	1.1.2	
1.29-2024.02.13	1.29.0	1.6.25	1.1.2	
1.29-2024.02.06	1.29.0	1.6.25	1.1.2	一時停止画像が kubelet ガベージコレクション処理によって誤って削除されてしまうバグを修正しました。
1.29-2024.01.09	1.29.0	1.6.18	1.1.2	

Kubernetes version 1.28

AMI のバージョン	kubelet バージョン	containerd バージョン	csi-proxy バージョン	リリースノート
1.28-2025.11.14	1.28.15	1.7.28	1.2.1	
1.28-2025.10.18	1.28.15	1.7.28	1.2.1	
1.28-2025.09.13	1.28.15	1.7.28	1.2.1	GMSA プラグインログを Windows イベントに変更
1.28-2025.08.18	1.28.15	1.7.27	1.2.1	
1.28-2025.07.16	1.28.15	1.7.27	1.2.1	
1.28-2025.06.13	1.28.15	1.7.27	1.2.1	containerd を 1.7.27 にアップグレードしました。
1.28-2025.05.17	1.28.15	1.7.20	1.2.1	
1.28-2025.04.14	1.28.15	1.7.20	1.1.3	
1.28-2025.03.14	1.28.15	1.7.20	1.1.3	containerd を 1.7.20 にアップグレードしました。

AMI のバージョン	kubelet バージョン	containerd バージョン	csi-proxy バージョン	リリースノート
1.28-2025.02.15	1.28.15	1.7.14	1.1.3	
1.28-2025.01.15	1.28.15	1.7.14	1.1.3	
1.28-2025.01.01	1.28.15	1.7.14	1.1.3	CVE-2024-9042 のパッチが含まれます。
1.28-2024.12.11	1.28.15	1.7.14	1.1.3	
1.28-2024.11.12	1.28.13	1.7.14	1.1.3	
1.28-2024.10.08	1.28.13	1.7.14	1.1.3	
1.28-2024.09.10	1.28.11	1.7.14	1.1.3	
1.28-2024.08.13	1.28.11	1.7.14	1.1.3	
1.28-2024.07.10	1.28.11	1.7.11	1.1.2	CVE-2024-5321 のパッチが含まれます。
1.28-2024.06.17	1.28.8	1.7.11	1.1.2	containerd を 1.7.11 にアップグレードしました。

AMI のバージョン	kubelet バージョン	containerd バージョン	csi-proxy バージョン	リリースノート
1.28-2024.05.14	1.28.8	1.6.28	1.1.2	containerd を 1.6.28 にアップグレードしました。kubelet を 1.28.8 にアップグレードしました。
1.28-2024.04.09	1.28.5	1.6.25	1.1.2	containerd を 1.6.25 にアップグレードしました。golang 1.22.1 を使用して CNI と csi-proxy が再構築されました。
1.28-2024.03.12	1.28.5	1.6.18	1.1.2	
1.28-2024.02.13	1.28.5	1.6.18	1.1.2	
1.28-2024.01.09	1.28.5	1.6.18	1.1.2	
1.28-2023.12.12	1.28.3	1.6.18	1.1.2	
1.28-2023.11.14	1.28.3	1.6.18	1.1.2	CVE-2023-5528 のパッチが含まれます。

AMI のバージョン	kubelet バージョン	containerd バージョン	csi-proxy バージョン	リリースノート
1.28-2023.10.19	1.28.2	1.6.18	1.1.2	containerd を 1.6.18 にアップグレードしました。新しいブートストラップスクリプト環境変数 (SERVICE_IPV4_CIDR と EXCLUDED_SNAT_CIDRS) を追加しました。
1.28-2023-09.27	1.28.2	1.6.6	1.1.2	kubelet のセキュリティ勧告を修正しました。
1.28-2023.09.12	1.28.1	1.6.6	1.1.2	

Amazon EKS 最適化 Windows Server 2019 Core AMI

次の表は、Amazon EKS 最適化 Windows Server 2019 Core AMI の現在および以前のバージョンを示しています。

Example

Kubernetes version 1.36

AMI のバージョン	kubelet バージョン	containerd バージョン	csi-proxy バージョン	リリースノート
1.36-2026.09.14	1.36.3	2.2.3	1.2.1	CVE-2026-76654 のパッチを含む
1.36-2026.08.15	1.36.2	2.2.3	1.2.1	
1.36-2026.07.18	1.36.2	2.2.3	1.2.1	
1.36-2026.06.18	1.36.2	2.2.3	1.2.1	
1.36-2026.05.19	1.36.1	2.2.3	1.2.1	

Kubernetes version 1.35

AMI のバージョン	kubelet バージョン	containerd バージョン	csi-proxy バージョン	リリースノート
1.35-2026.09.14	1.35.7	2.1.7	1.2.1	CVE-2026-76654 のパッチを含む
1.35-2026.08.15	1.35.6	2.1.7	1.2.1	
1.35-2026.07.18	1.35.6	2.1.7	1.2.1	

AMI のバージョン	kubelet バージョン	containerd バージョン	csi-proxy バージョン	リリースノート
1.35-2026.06.18	1.35.6	2.1.7	1.2.1	
1.35-2026.05.19	1.35.5	2.1.7	1.2.1	
1.35-2026.04.18	1.35.3	2.1.6	1.2.1	
1.35-2026.02.16	1.35.0	2.1.6	1.2.1	
1.35-2026-01-22	1.35.0	2.1.6	1.2.1	

Kubernetes version 1.34

AMI のバージョン	kubelet バージョン	containerd バージョン	csi-proxy バージョン	リリースノート
1.34-2026.09.14	1.34.10	2.1.7	1.2.1	CVE-2026-76654 のパッチを含む
1.34-2026.08.15	1.34.9	2.1.7	1.2.1	
1.34-2026.07.18	1.34.9	2.1.7	1.2.1	
1.34-2026.06.18	1.34.9	2.1.6	1.2.1	
1.34-2026.05.19	1.34.8	2.1.6	1.2.1	

AMI のバージョン	kubelet バージョン	containerd バージョン	csi-proxy バージョン	リリースノート
1.34-2026.04.18	1.34.6	2.1.6	1.2.1	
1.34-2026.02.13	1.34.3	2.1.6	1.2.1	
1.34-2026.01.22	1.34.2	2.1.6	1.2.1	containerd を 2.1.6 にアップグレードしました。
1.34-2025.12.15	1.34.2	2.1.4	1.2.1	
1.34-2025.11.14	1.34.1	2.1.4	1.2.1	
1.34-2025.10.18	1.34.1	2.1.4	1.2.1	
1.34-2025.09.13	1.34.0	2.1.4	1.2.1	

Kubernetes version 1.33

AMI のバージョン	kubelet バージョン	containerd バージョン	csi-proxy バージョン	リリースノート
1.33-2026.09.14	1.33.13	1.7.30	1.2.1	CVE-2026-76654 のパッチを含む
1.33-2026.08.15	1.33.13	1.7.30	1.2.1	

AMI のバージョン	kubelet バージョン	containerd バージョン	csi-proxy バージョン	リリースノート
1.33-2026.07.18	1.33.13	1.7.30	1.2.1	
1.33-2026.06.18	1.33.13	1.7.30	1.2.1	
1.33-2026.05.19	1.33.11	1.7.30	1.2.1	
1.33-2026.04.18	1.33.10	1.7.30	1.2.1	
1.33-2026.02.13	1.33.7	1.7.30	1.2.1	
1.33-2026.01.22	1.33.5	1.7.30	1.2.1	containerd を 1.7.30 にアップグレードしました。
1.33-2025.12.15	1.33.5	1.7.28	1.2.1	
1.33-2025.11.14	1.33.5	1.7.28	1.2.1	
1.33-2025.10.18	1.33.5	1.7.28	1.2.1	
1.33-2025.09.13	1.33.4	1.7.28	1.2.1	GMSA プラグインログを Windows イベントに変更
1.33-2025.08.18	1.33.3	1.7.27	1.2.1	

AMI のバージョン	kubelet バージョン	containerd バージョン	csi-proxy バージョン	リリースノート
1.33-2025.07.16	1.33.1	1.7.27	1.2.1	
1.33-2025.06.13	1.33.1	1.7.27	1.2.1	
1.33-2025.05.17	1.33.1	1.7.27	1.2.1	

Kubernetes version 1.32

AMI のバージョン	kubelet バージョン	containerd バージョン	csi-proxy バージョン	リリースノート
1.32-2026.09.14	1.32.13	1.7.30	1.2.1	CVE-2026-76654 のパッチを含む
1.32-2026.08.15	1.32.13	1.7.30	1.2.1	
1.32-2026.07.18	1.32.13	1.7.30	1.2.1	
1.32-2026.06.18	1.32.13	1.7.30	1.2.1	
1.32-2026.05.19	1.32.13	1.7.30	1.2.1	
1.32-2026.04.18	1.32.13	1.7.30	1.2.1	
1.32-2026.02.13	1.32.11	1.7.30	1.2.1	

AMI のバージョン	kubelet バージョン	containerd バージョン	csi-proxy バージョン	リリースノート
1.32-2026.01.22	1.32.9	1.7.30	1.2.1	containerd を 1.7.30 にアップグレードしました。
1.32-2025.12.15	1.32.9	1.7.28	1.2.1	
1.32-2025.11.14	1.32.9	1.7.28	1.2.1	
1.32-2025.10.18	1.32.9	1.7.28	1.2.1	
1.32-2025.09.13	1.32.8	1.7.28	1.2.1	GMSA プラグインログを Windows イベントに変更
1.32-2025.08.18	1.32.7	1.7.27	1.2.1	
1.32-2025.07.16	1.32.5	1.7.27	1.2.1	
1.32-2025.06.13	1.32.5	1.7.27	1.2.1	containerd を 1.7.27 にアップグレードしました。
1.32-2025.05.17	1.32.5	1.7.20	1.2.1	
1.32-2025.04.14	1.32.1	1.7.20	1.1.3	

AMI のバージョン	kubelet バージョン	containerd バージョン	csi-proxy バージョン	リリースノート
1.32-2025.03.14	1.32.1	1.7.20	1.1.3	
1.32-2025.02.18	1.32.1	1.7.20	1.1.3	
1.32-2025.01.15	1.32.4	1.7.20	1.1.3	

Kubernetes version 1.31

AMI のバージョン	kubelet バージョン	containerd バージョン	csi-proxy バージョン	リリースノート
1.31-2026.09.14	1.31.14	1.7.30	1.2.1	CVE-2026-76654 のパッチを含む
1.31-2026.08.15	1.31.14	1.7.30	1.2.1	
1.31-2026.07.18	1.31.14	1.7.30	1.2.1	
1.31-2026.06.18	1.31.14	1.7.30	1.2.1	
1.31-2026.05.19	1.31.14	1.7.30	1.2.1	
1.31-2026.04.18	1.31.14	1.7.30	1.2.1	
1.31-2026.02.13	1.31.14	1.7.30	1.2.1	

AMI のバージョン	kubelet バージョン	containerd バージョン	csi-proxy バージョン	リリースノート
1.31-2026.01.22	1.31.13	1.7.30	1.2.1	containerd を 1.7.30 にアップグレードしました。
1.31-2025.12.15	1.31.13	1.7.28	1.2.1	
1.31-2025.11.14	1.31.13	1.7.28	1.2.1	
1.31-2025.10.18	1.31.13	1.7.28	1.2.1	
1.31-2025.09.13	1.31.12	1.7.28	1.2.1	GMSA プラグインログを Windows イベントに変更
1.31-2025.08.18	1.31.11	1.7.27	1.2.1	
1.31-2025.07.16	1.31.9	1.7.27	1.2.1	
1.31-2025.06.13	1.31.9	1.7.27	1.2.1	containerd を 1.7.27 にアップグレードしました。
1.31-2025.05.17	1.31.9	1.7.20	1.2.1	
1.31-2025.04.14	1.31.5	1.7.20	1.1.3	

AMI のバージョン	kubelet バージョン	containerd バージョン	csi-proxy バージョン	リリースノート
1.31-2025.03.14	1.31.5	1.7.20	1.1.3	
1.31-2025.02.15	1.31.5	1.7.20	1.1.3	
1.31-2025.01.15	1.31.4	1.7.20	1.1.3	
1.31-2025.01.01	1.31.4	1.7.20	1.1.3	CVE-2024-9042 のパッチが含まれます。
1.31-2024.12.13	1.31.3	1.7.20	1.1.3	
1.31-2024.11.12	1.31.1	1.7.20	1.1.3	
1.31-2024.10.08	1.31.1	1.7.20	1.1.3	
1.31-2024.10.01	1.31.1	1.7.20	1.1.3	
1.31-2024.09.10	1.31.0	1.7.20	1.1.3	

Kubernetes version 1.30

AMI のバージョン	kubelet バージョン	containerd バージョン	csi-proxy バージョン	リリースノート
1.30-2026.07.18	1.30.14	1.7.30	1.2.1	

AMI のバージョン	kubelet バージョン	containerd バージョン	csi-proxy バージョン	リリースノート
1.30-2026.06.18	1.30.14	1.7.30	1.2.1	
1.30-2026.05.19	1.30.14	1.7.30	1.2.1	
1.30-2026.04.18	1.30.14	1.7.30	1.2.1	
1.30-2026.02.13	1.30.14	1.7.30	1.2.1	
1.30-2026.01.22	1.30.14	1.7.30	1.2.1	containerd を 1.7.30 にアップグレードしました。
1.30-2025.12.15	1.30.14	1.7.28	1.2.1	
1.30-2025.11.21	1.30.14	1.7.28	1.2.1	
1.30-2025.10.18	1.30.14	1.7.28	1.2.1	
1.30-2025.09.13	1.30.14	1.7.28	1.2.1	GMSA プラグインログを Windows イベントに変更
1.30-2025.08.18	1.30.14	1.7.27	1.2.1	
1.30-2025.07.16	1.30.13	1.7.27	1.2.1	

AMI のバージョン	kubelet バージョン	containerd バージョン	csi-proxy バージョン	リリースノート
1.30-2025.06.13	1.30.13	1.7.27	1.2.1	containerd を 1.7.27 にアップグレードしました。
1.30-2025.05.17	1.30.13	1.7.20	1.2.1	
1.30-2025.04.14	1.30.9	1.7.20	1.1.3	
1.30-2025.03.14	1.30.9	1.7.20	1.1.3	containerd を 1.7.20 にアップグレードしました。
1.30-2025-02-15	1.30.9	1.7.14	1.1.3	
1.30-2025.01.15	1.30.8	1.7.14	1.1.3	
1.30-2025.01.01	1.30.8	1.7.14	1.1.3	CVE-2024-9042 のパッチが含まれます。
1.30-2024.12.11	1.30.7	1.7.14	1.1.3	
1.30-2024.11.12	1.30.4	1.7.14	1.1.3	
1.30-2024.10.08	1.30.4	1.7.14	1.1.3	

AMI のバージョン	kubelet バージョン	containerd バージョン	csi-proxy バージョン	リリースノート
1.30-2024.09.10	1.30.2	1.7.14	1.1.3	
1.30-2024.08.13	1.30.2	1.7.14	1.1.3	
1.30-2024.07.10	1.30.2	1.7.14	1.1.2	CVE-2024-5321 のパッチが含まれます。
1.30-2024.06.17	1.30.0	1.7.14	1.1.2	containerd を 1.7.14 にアップグレードしました。
1.30-2024.05.15	1.30.0	1.6.28	1.1.2	

Kubernetes version 1.29

AMI のバージョン	kubelet バージョン	containerd バージョン	csi-proxy バージョン	リリースノート
1.29-2026.02.13	1.29.15	1.7.30	1.2.1	
1.29-2026.01.22	1.29.15	1.7.30	1.2.1	containerd を 1.7.30 にアップグレードしました。
1.29-2025.12.15	1.29.15	1.7.28	1.2.1	

AMI のバージョン	kubelet バージョン	containerd バージョン	csi-proxy バージョン	リリースノート
1.29-2025.11.14	1.29.15	1.7.28	1.2.1	
1.29-2025.10.18	1.29.15	1.7.28	1.2.1	
1.29-2025.09.13	1.29.15	1.7.28	1.2.1	GMSA プラグインログを Windows イベントに変更
1.29-2025.08.18	1.29.15	1.7.27	1.2.1	
1.29-2025.07.16	1.29.15	1.7.27	1.2.1	
1.29-2025.06.13	1.29.15	1.7.27	1.2.1	containerd を 1.7.27 にアップグレードしました。
1.29-2025.05.17	1.29.15	1.7.20	1.2.1	
1.29-2025.04.14	1.29.13	1.7.20	1.1.3	
1.29-2025.03.14	1.29.13	1.7.20	1.1.3	containerd を 1.7.20 にアップグレードしました。
1.29-2025.02.15	1.29.13	1.7.14	1.1.3	

AMI のバージョン	kubelet バージョン	containerd バージョン	csi-proxy バージョン	リリースノート
1.29-2025.01.15	1.29.12	1.7.14	1.1.3	
1.29-2025.01.01	1.29.12	1.7.14	1.1.3	CVE-2024-9042 のパッチが含まれます。
1.29-2024.12.11	1.29.10	1.7.14	1.1.3	
1.29-2024.11.12	1.29.8	1.7.14	1.1.3	
1.29-2024.10.08	1.29.8	1.7.14	1.1.3	
1.29-2024.09.10	1.29.6	1.7.14	1.1.3	
1.29-2024.08.13	1.29.6	1.7.14	1.1.3	
1.29-2024.07.10	1.29.6	1.7.11	1.1.2	CVE-2024-5321 のパッチが含まれます。
1.29-2024.06.17	1.29.3	1.7.11	1.1.2	
1.29-2024.05.15	1.29.3	1.7.11	1.1.2	containerd を 1.7.11 にアップグレードしました。kubelet を 1.29.3 にアップグレードしました。

AMI のバージョン	kubelet バージョン	containerd バージョン	csi-proxy バージョン	リリースノート
1.29-2024.04.09	1.29.0	1.6.28	1.1.2	containerd を 1.6.28 にアップグレードしました。golang 1.22.1 を使用して CNI と csi-proxy が再構築されました。
1.29-2024.03.13	1.29.0	1.6.25	1.1.2	
1.29-2024.02.13	1.29.0	1.6.25	1.1.2	
1.29-2024.02.06	1.29.0	1.6.25	1.1.2	一時停止画像が kubelet ガベージコレクション処理によって誤って削除されてしまうバグを修正しました。
1.29-2024.01.09	1.29.0	1.6.18	1.1.2	

Kubernetes version 1.28

AMI のバージョン	kubelet バージョン	containerd バージョン	csi-proxy バージョン	リリースノート
1.28-2025.11.14	1.28.15	1.7.28	1.2.1	
1.28-2025.10.18	1.28.15	1.7.28	1.2.1	
1.28-2025.09.13	1.28.15	1.7.28	1.2.1	GMSA プラグインログを Windows イベントに変更
1.28-2025.08.18	1.28.15	1.7.27	1.2.1	
1.28-2025.07.16	1.28.15	1.7.27	1.2.1	
1.28-2025.06.13	1.28.15	1.7.20	1.2.1	containerd を 1.7.27 にアップグレードしました。
1.28-2025.05.17	1.28.15	1.7.20	1.2.1	
1.28-2025.04.14	1.28.15	1.7.20	1.1.3	
1.28-2025.03.14	1.28.15	1.7.20	1.1.3	containerd を 1.7.20 にアップグレードしました。

AMI のバージョン	kubelet バージョン	containerd バージョン	csi-proxy バージョン	リリースノート
1.28-2025.02.15	1.28.15	1.7.14	1.1.3	
1.28-2025-01-15	1.28.15	1.7.14	1.1.3	
1.28-2025-01-01	1.28.15	1.7.14	1.1.3	CVE-2024-9042 のパッチが含まれます。
1.28-2024.12.11	1.28.15	1.7.14	1.1.3	
1.28-2024.11.12	1.28.13	1.7.14	1.1.3	
1.28-2024.10.08	1.28.13	1.7.14	1.1.3	
1.28-2024.09.10	1.28.11	1.7.14	1.1.3	
1.28-2024.08.13	1.28.11	1.7.14	1.1.3	
1.28-2024.07.10	1.28.11	1.7.11	1.1.2	CVE-2024-5321 のパッチが含まれます。
1.28-2024.06.17	1.28.8	1.7.11	1.1.2	containerd を 1.7.11 にアップグレードしました。

AMI のバージョン	kubelet バージョン	containerd バージョン	csi-proxy バージョン	リリースノート
1.28-2024.05.14	1.28.8	1.6.28	1.1.2	containerd を 1.6.28 にアップグレードしました。kubelet を 1.28.8 にアップグレードしました。
1.28-2024.04.09	1.28.5	1.6.25	1.1.2	containerd を 1.6.25 にアップグレードしました。golang 1.22.1 を使用して CNI と csi-proxy が再構築されました。
1.28-2024.03.13	1.28.5	1.6.18	1.1.2	
1.28-2024.02.13	1.28.5	1.6.18	1.1.2	
1.28-2024.01.09	1.28.5	1.6.18	1.1.2	
1.28-2023.12.12	1.28.3	1.6.18	1.1.2	
1.28-2023.11.14	1.28.3	1.6.18	1.1.2	CVE-2023-5528 のパッチが含まれます。

AMI のバージョン	kubelet バージョン	containerd バージョン	csi-proxy バージョン	リリースノート
1.28-2023.10.19	1.28.2	1.6.18	1.1.2	containerd を 1.6.18 にアップグレードしました。新しいブートストラップスクリプト環境変数 (SERVICE_IPV4_CIDR と EXCLUDED_SNAT_CIDRS) を追加しました。
1.28-2023-09.27	1.28.2	1.6.6	1.1.2	kubelet のセキュリティ勧告を修正しました。
1.28-2023.09.12	1.28.1	1.6.6	1.1.2	

Amazon EKS 最適化 Windows Server 2019 Full AMI

次の表は、Amazon EKS 最適化 Windows Server 2019 Full AMI の現在および以前のバージョンを示しています。

Example

Kubernetes version 1.36

AMI のバージョン	kubelet バージョン	containerd バージョン	csi-proxy バージョン	リリースノート
1.36-2026.09.14	1.36.3	2.2.3	1.2.1	CVE-2026-76654 のパッチを含む
1.36-2026.08.15	1.36.2	2.2.3	1.2.1	
1.36-2026.07.18	1.36.2	2.2.3	1.2.1	
1.36-2026.06.18	1.36.2	2.2.3	1.2.1	
1.36-2026.05.19	1.36.1	2.2.3	1.2.1	

Kubernetes version 1.35

AMI のバージョン	kubelet バージョン	containerd バージョン	csi-proxy バージョン	リリースノート
1.35-2026.09.14	1.35.7	2.1.7	1.2.1	CVE-2026-76654 のパッチを含む
1.35-2026.08.15	1.35.6	2.1.7	1.2.1	
1.35-2026.07.18	1.35.6	2.1.7	1.2.1	

AMI のバージョン	kubelet バージョン	containerd バージョン	csi-proxy バージョン	リリースノート
1.35-2026.06.18	1.35.6	2.1.7	1.2.1	
1.35-2026.05.19	1.35.5	2.1.7	1.2.1	
1.35-2026.04.18	1.35.3	2.1.6	1.2.1	
1.35-2026.02.16	1.35.0	2.1.6	1.2.1	
1.35-2026-01-22	1.35.0	2.1.6	1.2.1	

Kubernetes version 1.34

AMI のバージョン	kubelet バージョン	containerd バージョン	csi-proxy バージョン	リリースノート
1.34-2026.09.14	1.34.10	2.1.7	1.2.1	CVE-2026-76654 のパッチを含む
1.34-2026.08.15	1.34.9	2.1.7	1.2.1	
1.34-2026.07.21	1.34.9	2.1.7	1.2.1	
1.34-2026.06.18	1.34.9	2.1.6	1.2.1	
1.34-2026.05.19	1.34.8	2.1.6	1.2.1	

AMI のバージョン	kubelet バージョン	containerd バージョン	csi-proxy バージョン	リリースノート
1.34-2026.04.18	1.34.6	2.1.6	1.2.1	
1.34-2026.02.13	1.34.3	2.1.6	1.2.1	
1.34-2026.01.22	1.34.2	2.1.6	1.2.1	containerd を 2.1.6 にアップグレードしました。
1.34-2025.12.15	1.34.2	2.1.4	1.2.1	
1.34-2025.11.14	1.34.1	2.1.4	1.2.1	
1.34-2025.10.18	1.34.1	2.1.4	1.2.1	
1.34-2025.09.13	1.34.0	2.1.4	1.2.1	

Kubernetes version 1.33

AMI のバージョン	kubelet バージョン	containerd バージョン	csi-proxy バージョン	リリースノート
1.33-2026.09.14	1.33.13	1.7.30	1.2.1	CVE-2026-76654 のパッチを含む
1.33-2026.08.15	1.33.13	1.7.30	1.2.1	

AMI のバージョン	kubelet バージョン	containerd バージョン	csi-proxy バージョン	リリースノート
1.33-2026.07.18	1.33.13	1.7.30	1.2.1	
1.33-2026.06.18	1.33.13	1.7.30	1.2.1	
1.33-2026.05.19	1.33.11	1.7.30	1.2.1	
1.33-2026.04.18	1.33.10	1.7.30	1.2.1	
1.33-2026.02.13	1.33.7	1.7.30	1.2.1	
1.33-2026.01.22	1.33.5	1.7.30	1.2.1	containerd を 1.7.30 にアップグレードしました。
1.33-2025.12.15	1.33.5	1.7.28	1.2.1	
1.33-2025.11.14	1.33.5	1.7.28	1.2.1	
1.33-2025.10.18	1.33.5	1.7.28	1.2.1	
1.33-2025.09.13	1.33.4	1.7.28	1.2.1	GMSA プラグインログを Windows イベントに変更
1.33-2025.08.18	1.33.3	1.7.27	1.2.1	

AMI のバージョン	kubelet バージョン	containerd バージョン	csi-proxy バージョン	リリースノート
1.33-2025.07.16	1.33.1	1.7.27	1.2.1	
1.33-2025.06.13	1.33.1	1.7.27	1.2.1	
1.33-2025.05.17	1.33.1	1.7.27	1.2.1	

Kubernetes version 1.32

AMI のバージョン	kubelet バージョン	containerd バージョン	csi-proxy バージョン	リリースノート
1.32-2026.09.14	1.32.13	1.7.30	1.2.1	CVE-2026-76654 のパッチを含む
1.32-2026.08.15	1.32.13	1.7.30	1.2.1	
1.32-2026.07.18	1.32.13	1.7.30	1.2.1	
1.32-2026.06.18	1.32.13	1.7.30	1.2.1	
1.32-2026.05.19	1.32.13	1.7.30	1.2.1	
1.32-2026.04.18	1.32.13	1.7.30	1.2.1	
1.32-2026.02.13	1.32.11	1.7.30	1.2.1	

AMI のバージョン	kubelet バージョン	containerd バージョン	csi-proxy バージョン	リリースノート
1.32-2026.01.22	1.32.9	1.7.30	1.2.1	containerd を 1.7.30 にアップグレードしました。
1.32-2025.12.15	1.32.9	1.7.28	1.2.1	
1.32-2025.11.14	1.32.9	1.7.28	1.2.1	
1.32-2025.10.18	1.32.9	1.7.28	1.2.1	
1.32-2025.09.13	1.32.8	1.7.28	1.2.1	GMSA プラグインログを Windows イベントに変更
1.32-2025.08.18	1.32.7	1.7.27	1.2.1	
1.32-2025.07.16	1.32.5	1.7.27	1.2.1	
1.32-2025.06.13	1.32.5	1.7.27	1.2.1	containerd を 1.7.27 にアップグレードしました。
1.32-2025.05.17	1.32.5	1.7.20	1.2.1	
1.32-2025.04.14	1.32.1	1.7.20	1.1.3	

AMI のバージョン	kubelet バージョン	containerd バージョン	csi-proxy バージョン	リリースノート
1.32-2025.03.14	1.32.1	1.7.20	1.1.3	
1.32-2025.02.18	1.32.1	1.7.20	1.1.3	
1.32-2025.01.15	1.32.0	1.7.20	1.1.3	

Kubernetes version 1.31

AMI のバージョン	kubelet バージョン	containerd バージョン	csi-proxy バージョン	リリースノート
1.31-2026.09.14	1.31.14	1.7.30	1.2.1	CVE-2026-76654 のパッチを含む
1.31-2026.08.15	1.31.14	1.7.30	1.2.1	
1.31-2026.07.18	1.31.14	1.7.30	1.2.1	
1.31-2026.06.18	1.31.14	1.7.30	1.2.1	
1.31-2026.05.19	1.31.14	1.7.30	1.2.1	
1.31-2026.04.18	1.31.14	1.7.30	1.2.1	
1.31-2026.02.13	1.31.14	1.7.30	1.2.1	

AMI のバージョン	kubelet バージョン	containerd バージョン	csi-proxy バージョン	リリースノート
1.31-2026.01.22	1.31.13	1.7.30	1.2.1	containerd を 1.7.30 にアップグレードしました。
1.31-2025.12.15	1.31.13	1.7.28	1.2.1	
1.31-2025.11.14	1.31.13	1.7.28	1.2.1	
1.31-2025.10.18	1.31.13	1.7.28	1.2.1	
1.31-2025.09.13	1.31.12	1.7.28	1.2.1	GMSA プラグインログを Windows イベントに変更
1.31-2025.08.18	1.31.11	1.7.27	1.2.1	
1.31-2025.07.16	1.31.9	1.7.27	1.2.1	
1.31-2025.06.13	1.31.9	1.7.27	1.2.1	containerd を 1.7.27 にアップグレードしました。
1.31-2025.05.17	1.31.9	1.7.20	1.2.1	
1.31-2025.04.14	1.31.5	1.7.20	1.1.3	

AMI のバージョン	kubelet バージョン	containerd バージョン	csi-proxy バージョン	リリースノート
1.31-2025.03.14	1.31.5	1.7.20	1.1.3	
1.31-2025.02.15	1.31.5	1.7.20	1.1.3	
1.31-2025.01.15	1.31.4	1.7.20	1.1.3	
1.31-2025.01.01	1.31.4	1.7.20	1.1.3	CVE-2024-9042 のパッチが含まれます。
1.31-2024.12.13	1.31.3	1.7.20	1.1.3	
1.31-2024.11.12	1.31.1	1.7.20	1.1.3	
1.31-2024.10.08	1.31.1	1.7.20	1.1.3	
1.31-2024.10.01	1.31.1	1.7.20	1.1.3	
1.31-2024.09.10	1.31.0	1.7.20	1.1.3	

Kubernetes version 1.30

AMI のバージョン	kubelet バージョン	containerd バージョン	csi-proxy バージョン	リリースノート
1.30-2026.07.18	1.30.14	1.7.30	1.2.1	

AMI のバージョン	kubelet バージョン	containerd バージョン	csi-proxy バージョン	リリースノート
1.30-2026.06.18	1.30.14	1.7.30	1.2.1	
1.30-2026.05.19	1.30.14	1.7.30	1.2.1	
1.30-2026.04.18	1.30.14	1.7.30	1.2.1	
1.30-2026.02.13	1.30.14	1.7.30	1.2.1	
1.30-2026.01.22	1.30.14	1.7.30	1.2.1	containerd を 1.7.30 にアップグレードしました。
1.30-2025.12.15	1.30.14	1.7.28	1.2.1	
1.30-2025.11.21	1.30.14	1.7.28	1.2.1	
1.30-2025.10.18	1.30.14	1.7.28	1.2.1	
1.30-2025.09.13	1.30.14	1.7.28	1.2.1	GMSA プラグインログを Windows イベントに変更
1.30-2025.08.18	1.30.14	1.7.27	1.2.1	
1.30-2025.07.16	1.30.13	1.7.27	1.2.1	

AMI のバージョン	kubelet バージョン	containerd バージョン	csi-proxy バージョン	リリースノート
1.30-2025.06.13	1.30.13	1.7.27	1.2.1	containerd を 1.7.27 にアップグレードしました。
1.30-2025.05.17	1.30.13	1.7.20	1.2.1	
1.30-2025.04.14	1.30.9	1.7.20	1.1.3	
1.30-2025.03.14	1.30.9	1.7.20	1.1.3	containerd を 1.7.20 にアップグレードしました。
1.30-2025.02.15	1.30.9	1.7.14	1.1.3	
1.30-2025.01.15	1.30.8	1.7.14	1.1.3	
1.30-2025.01.01	1.30.8	1.7.14	1.1.3	CVE-2024-9042 のパッチが含まれます。
1.30-2024.12.11	1.30.7	1.7.14	1.1.3	
1.30-2024.11.12	1.30.4	1.7.14	1.1.3	
1.30-2024.10.08	1.30.4	1.7.14	1.1.3	

AMI のバージョン	kubelet バージョン	containerd バージョン	csi-proxy バージョン	リリースノート
1.30-2024.09.10	1.30.2	1.7.14	1.1.3	
1.30-2024.08.13	1.30.2	1.7.14	1.1.3	
1.30-2024.07.10	1.30.2	1.7.14	1.1.2	CVE-2024-5321 のパッチが含まれます。
1.30-2024.06.17	1.30.0	1.7.14	1.1.2	containerd を 1.7.14 にアップグレードしました。
1.30-2024.05.15	1.30.0	1.6.28	1.1.2	

Kubernetes version 1.29

AMI のバージョン	kubelet バージョン	containerd バージョン	csi-proxy バージョン	リリースノート
1.29-2026.02.13	1.29.15	1.7.30	1.2.1	
1.29-2026.01.22	1.29.15	1.7.30	1.2.1	containerd を 1.7.30 にアップグレードしました。
1.29-2025.12.15	1.29.15	1.7.28	1.2.1	

AMI のバージョン	kubelet バージョン	containerd バージョン	csi-proxy バージョン	リリースノート
1.29-2025.11.14	1.29.15	1.7.28	1.2.1	
1.29-2025.10.18	1.29.15	1.7.28	1.2.1	
1.29-2025.09.13	1.29.15	1.7.28	1.2.1	GMSA プラグインログを Windows イベントに変更
1.29-2025.08.18	1.29.15	1.7.27	1.2.1	
1.29-2025.07.16	1.29.15	1.7.27	1.2.1	
1.29-2025.06.13	1.29.15	1.7.27	1.2.1	containerd を 1.7.27 にアップグレードしました。
1.29-2025.05.17	1.29.15	1.7.20	1.2.1	
1.29-2025.04.14	1.29.13	1.7.20	1.1.3	
1.29-2025.03.14	1.29.13	1.7.20	1.1.3	containerd を 1.7.20 にアップグレードしました。
1.29-2025.02.15	1.29.13	1.7.14	1.1.3	

AMI のバージョン	kubelet バージョン	containerd バージョン	csi-proxy バージョン	リリースノート
1.29-2025.01.15	1.29.12	1.7.14	1.1.3	
1.29-2025.01.01	1.29.12	1.7.14	1.1.3	CVE-2024-9042 のパッチが含まれます。
1.29-2024.12.11	1.29.10	1.7.14	1.1.3	
1.29-2024.11.12	1.29.8	1.7.14	1.1.3	
1.29-2024.10.08	1.29.8	1.7.14	1.1.3	
1.29-2024.09.10	1.29.6	1.7.14	1.1.3	
1.29-2024.08.13	1.29.6	1.7.14	1.1.3	
1.29-2024.07.10	1.29.6	1.7.11	1.1.2	CVE-2024-5321 のパッチが含まれます。
1.29-2024.06.17	1.29.3	1.7.11	1.1.2	
1.29-2024.05.15	1.29.3	1.7.11	1.1.2	containerd を 1.7.11 にアップグレードしました。kubelet を 1.29.3 にアップグレードしました。

AMI のバージョン	kubelet バージョン	containerd バージョン	csi-proxy バージョン	リリースノート
1.29-2024.04.09	1.29.0	1.6.28	1.1.2	containerd を 1.6.28 にアップグレードしました。golang 1.22.1 を使用して CNI と csi-proxy が再構築されました。
1.29-2024.03.13	1.29.0	1.6.25	1.1.2	
1.29-2024.02.13	1.29.0	1.6.25	1.1.2	
1.29-2024.02.06	1.29.0	1.6.25	1.1.2	一時停止画像が kubelet ガベージコレクション処理によって誤って削除されてしまうバグを修正しました。
1.29-2024.01.09	1.29.0	1.6.18	1.1.2	

Kubernetes version 1.28

AMI のバージョン	kubelet バージョン	containerd バージョン	csi-proxy バージョン	リリースノート
1.28-2025.11.14	1.28.15	1.7.28	1.2.1	
1.28-2025.10.18	1.28.15	1.7.28	1.2.1	
1.28-2025.09.13	1.28.15	1.7.28	1.2.1	GMSA プラグインログを Windows イベントに変更
1.28-2025.08.18	1.28.15	1.7.27	1.2.1	
1.28-2025.07.16	1.28.15	1.7.27	1.2.1	
1.28-2025.06.13	1.28.15	1.7.27	1.2.1	containerd を 1.7.27 にアップグレードしました。
1.28-2025.05.17	1.28.15	1.7.20	1.2.1	
1.28-2025.04.14	1.28.15	1.7.20	1.1.3	
1.28-2025.03.14	1.28.15	1.7.20	1.1.3	containerd を 1.7.20 にアップグレードしました。

AMI のバージョン	kubelet バージョン	containerd バージョン	csi-proxy バージョン	リリースノート
1.28-2025.02.15	1.28.15	1.7.14	1.1.3	
1.28-2025-01-15	1.28.15	1.7.14	1.1.3	
1.28-2025-01-01	1.28.15	1.7.14	1.1.3	CVE-2024-9042 のパッチが含まれます。
1.28-2024.12.11	1.28.15	1.7.14	1.1.3	
1.28-2024.11.12	1.28.13	1.7.14	1.1.3	
1.28-2024.10.08	1.28.13	1.7.14	1.1.3	
1.28-2024.09.10	1.28.11	1.7.14	1.1.3	
1.28-2024.08.13	1.28.11	1.7.14	1.1.3	
1.28-2024.07.10	1.28.11	1.7.11	1.1.2	CVE-2024-5321 のパッチが含まれます。
1.28-2024.06.17	1.28.8	1.7.11	1.1.2	containerd を 1.7.11 にアップグレードしました。

AMI のバージョン	kubelet バージョン	containerd バージョン	csi-proxy バージョン	リリースノート
1.28-2024.05.14	1.28.8	1.6.28	1.1.2	containerd を 1.6.28 にアップグレードしました。kubelet を 1.28.8 にアップグレードしました。
1.28-2024.04.09	1.28.5	1.6.25	1.1.2	containerd を 1.6.25 にアップグレードしました。golang 1.22.1 を使用して CNI と csi-proxy が再構築されました。
1.28-2024.03.13	1.28.5	1.6.18	1.1.2	
1.28-2024.02.13	1.28.5	1.6.18	1.1.2	
1.28-2024.01.09	1.28.5	1.6.18	1.1.2	
1.28-2023.12.12	1.28.3	1.6.18	1.1.2	
1.28-2023.11.14	1.28.3	1.6.18	1.1.2	CVE-2023-5528 のパッチが含まれます。

AMI のバージョン	kubelet バージョン	containerd バージョン	csi-proxy バージョン	リリースノート
1.28-2023.10.19	1.28.2	1.6.18	1.1.2	containerd を 1.6.18 にアップグレードしました。新しいブートストラップスクリプト環境変数 (SERVICE_IPV4_CIDR と EXCLUDED_SNAT_CIDRS) を追加しました。
1.28-2023-09.27	1.28.2	1.6.6	1.1.2	kubelet のセキュリティ勧告を修正しました。
1.28-2023.09.12	1.28.1	1.6.6	1.1.2	

推奨 Microsoft Windows AMI ID を取得する

ノードをデプロイする際に、事前構築済みの Amazon EKS 最適化 Amazon マシンイメージ (AMI) の ID を指定できます。希望する設定に合った AMI ID を取得するには、AWS Systems Manager Parameter Store API をクエリします。この API を使用すると、Amazon EKS 最適化 AMI ID を手動で検索する必要がなくなります。詳細については、「GetParameter」を参照してください。使用する IAM プリンシパルには、Amazon EKS 最適化 AMI メタデータを取得するための `ssm:GetParameter` IAM アクセス許可が必要です。

サブパラメータ `image_id` を使用する次のコマンドで、推奨される最新の Amazon EKS 最適化 Windows AMI のイメージ ID を取得できます。必要に応じてコマンドに次の変更を加え、変更したコマンドを実行してください：

- *release* は、以下のいずれかのオプションに置き換えます。
  - Windows Server 2025 には *2025* を使用します。
  - Windows Server 2022 には *2022* を使用します。
  - Windows Server 2019 には *2019* を使用します。
- *installation-option* は、以下のいずれかのオプションに置き換えます。詳細については、「What is the Server Core installation option in Windows Server」を参照してください。
  - アタックサーフェスを小さくした最小限のインストールには *Core* を使用します。
  - Windows デスクトップエクスペリエンスを含めるには *Full* を使用します。
- *kubernetes-version* を、サポートされている platform-version に置き換えます。
- *region-code* を、AMI ID が必要な Amazon EKS がサポートしている AWS リージョンに置き換えます。

```
aws ssm get-parameter --name /aws/service/ami-windows-latest/Windows_Server-release-
English-installation-option-EKS_Optimized-kubernetes-version/image_id \
    --region region-code --query "Parameter.Value" --output text
```

プレースホルダーの置換が行われた後のコマンドの例を以下に示します。

```
aws ssm get-parameter --name /aws/service/ami-windows-latest/Windows_Server-2022-
English-Core-EKS_Optimized-k8s-n-2/image_id \
    --region us-west-2 --query "Parameter.Value" --output text
```

出力例は次のとおりです。

```
ami-1234567890abcdef0
```

Image Builder を使用してカスタム Windows AMI を構築する

EC2 Image Builder を使用し、次のオプションのいずれかを使用して、カスタムの Amazon EKS 最適化 Windows AMI を作成できます。

- Amazon EKS 最適化 Windows AMI をベースとして使用する
- Amazon 管理ビルドコンポーネントを使用する

どちらの方法でも、独自の Image Builder レシピを作成する必要があります。詳細については、「Image Builder ユーザーガイド」の「イメージレシピの新しいバージョンの作成」を参照してください。

> **Important**
>
> 以下の eks の [Amazon 管理] コンポーネントには、CVE-2024-5321 のパッチが含まれます。
>
> - 1.28.2 以上
> - 1.29.2 以上
> - 1.30.1 以上
> - Kubernetes 1.31 以降のすべてのバージョン

Amazon EKS 最適化 Windows AMI をベースとして使用する

このオプションは、カスタム Windows AMI を構築するのに推奨される方法です。当社が提供する Amazon EKS 最適化 Windows AMI は、Amazon 管理のビルドコンポーネントよりも頻繁に更新されます。

1. 新しい Image Builder レシピを始める
   a. https://console.aws.amazon.com/imagebuilder で、EC2 Image Builder コンソールを開きます。
   b. 左側のナビゲーションペインで、[イメージレシピ] を選択します。
   c. [イメージレシピの作成] を選択します。
2. [レシピの詳細] セクションで、[名前] と [バージョン] を入力します。
3. [ベースイメージ] セクションで Amazon EKS 最適化 Windows AMI の ID を指定します。
   a. [カスタム AMI ID を入力] を選択します。
   b. 必要な Windows OS バージョンの AMI ID を取得します。詳細については、「the section called “最新の ID を取得する”」を参照してください。
   c. カスタム [AMI ID] を入力します。AMI ID が見つからない場合は、AMI ID の AWS リージョンがコンソールの右上に表示されている AWS リージョンと一致しているか確認してください。
4. (オプション) 最新のセキュリティアップデートを入手するには、update-windows コンポーネントを [ビルドコンポーネント -] セクションに追加します。
   a. [名前でコンポーネントを検索する] 検索ボックスの右側にあるドロップダウンリストから、[Amazon 管理] を選択します。

b. [名前でコンポーネントを検索する] 検索ボックスに update-windows と入力します。

c. **update-windows** の検索結果のチェックボックスを選択します。このコンポーネントは、オペレーティングシステムの最新 Windows パッチを含みます。

5. 必要な設定で残りのイメージレシピの入力を完了します。詳細については、「Image Builder ユーザーガイド」の「イメージレシピの新しいバージョンの作成 (コンソール)」を参照してください。

6. [レシピを作成する] を選択します。

7. 新しいイメージレシピを新規または既存のイメージパイプラインで使用します。イメージパイプラインが正常に実行されると、カスタム AMI が出力イメージとして一覧表示され、使用できるようになります。詳細については、「EC2 Image Builder コンソールウィザードを使用してイメージパイプラインを作成する」を参照してください。

Amazon 管理ビルドコンポーネントを使用する

Amazon EKS 最適化 Windows AMI をベースとして使用できない場合は、代わりに Amazon 管理ビルドコンポーネントを使用できます。このオプションは、サポートされている最新の Kubernetes バージョンよりも遅れる可能性があります。

1. 新しい Image Builder レシピを始める

a. https://console.aws.amazon.com/imagebuilder で、EC2 Image Builder コンソールを開きます。

b. 左側のナビゲーションペインで、[イメージレシピ] を選択します。

c. [イメージレシピの作成] を選択します。

2. [レシピの詳細] セクションで、[名前] と [バージョン] を入力します。

3. [ベースイメージ] のセクションで、カスタム AMI の作成に使用するオプションを決定します。

- [管理イメージの選択] — [イメージオペレーティングシステム (OS)] で [Windows] を選択します。次に、[イメージのオリジン] について、以下のいずれかのオプションを選択します。
  - [クイックスタート (Amazon 管理)] – [イメージ名] ドロップダウンから、Amazon EKS でサポートされている Windows Server のバージョンを選択します。詳細については、「the section called “Windows”」を参照してください。
  - [ユーザー所有のイメージ] – [イメージ名] で、独自ライセンスを持つ独自のイメージの ARN を選択します。Amazon EKS コンポーネントをインストール済みのイメージを指定することはできません。

- [カスタム AMI ID を入力] – [AMI ID] に、独自ライセンスを持つ AMI の ID を入力します。Amazon EKS コンポーネントをインストール済みのイメージを指定することはできません。

4. [ビルドコンポーネント - Windows] セクションで、次の操作を行います。
   a. [名前でコンポーネントを検索する] 検索ボックスの右側にあるドロップダウンリストから、[Amazon 管理] を選択します。
   b. [名前でコンポーネントを検索する] 検索ボックスに `eks` と入力します。
   c. たとえ目的のバージョンではない場合でも、 **`eks-optimized-ami-windows`** 検索結果のチェックボックスを選択します。
   d. [名前でコンポーネントを検索する] 検索ボックスに `update-windows` と入力します。
   e. update-windows の検索結果のチェックボックスを選択します。このコンポーネントは、オペレーティングシステムの最新 Windows パッチを含みます。
5. [選択したコンポーネント] セクションで、次の手順を実行します。
   a. [**`eks-optimized-ami-windows`**] の [バージョニングオプション] を選択してください。
   b. [コンポーネントバージョンを指定する] を選択します。
   c. [コンポーネントバージョン] フィールドに *`version.x`* と入力し、*`#####`*をサポートされている Kubernetes バージョンに置き換えます。バージョン番号の一部に *`x`* を入力すると、明示的に定義したバージョンの一部と一致する最新のコンポーネントバージョンが使用されます。コンソールの出力に注目してください。目的のバージョンが管理コンポーネントとして利用可能かどうかが示されます。ビルドコンポーネントでは、最新の Kubernetes バージョンを使用できない場合があります。使用できるバージョンの詳細については、「the section called “eks-optimized-ami-windows コンポーネントのバージョンに関する情報を取得する”」を参照してください。
6. 必要な設定で残りのイメージレシピの入力を完了します。詳細については、「Image Builder ユーザーガイド」の「イメージレシピの新しいバージョンの作成 (コンソール)」を参照してください。
7. [レシピを作成する] を選択します。
8. 新しいイメージレシピを新規または既存のイメージパイプラインで使用します。イメージパイプラインが正常に実行されると、カスタム AMI が出力イメージとして一覧表示され、使用できるようになります。詳細については、「EC2 Image Builder コンソールウィザードを使用してイメージパイプラインを作成する」を参照してください。

eks-optimized-ami-windows コンポーネントのバージョンに関する情報を取得する

各コンポーネントにインストールされているものに関する特定の情報を取得できます。例えば、どの kubelet バージョンがインストールされているかを確認できます。コンポーネントには、Amazon EKS がサポートする Windows オペレーティングシステムのバージョンで機能テストが実施されます。詳細については、「the section called “リリースカレンダー”」を参照してください。その他の Windows OS バージョンで、サポートされていない、またはサポート終了としてリストにあるものは、このコンポーネントと互換性がない可能性があります。

1. https://console.aws.amazon.com/imagebuilder で、EC2 Image Builder コンソールを開きます。
2. 左にあるナビゲーションペインで、[コンポーネント] をクリックします。
3. [名前でコンポーネントを検索する] 検索ボックスの右側にあるドロップダウンリストで、[自己所有] を [クイックスタート (Amazon 管理)] に変更します。
4. [名前でコンポーネントを検索] ボックスに eks と入力します。
5. (オプション) 最新バージョンを使用している場合は、[バージョン] 列を 2 回選択して降順にソートします。
6. 目的のバージョンの [**eks-optimized-ami-windows**] リンクを選択します。

結果のページの [説明] には、特定の情報が表示されます。

ノードのヘルス問題を検出し、自動ノード修復を有効にする

ノードのヘルスとは、ワークロードを効果的に実行するための Kubernetes ノードの運用ステータスと機能を指します。正常なノードは、期待されるネットワーク接続を維持し、十分なコンピューティングリソースとストレージリソースを持ち、中断することなくワークロードを正常に実行できます。

EKS クラスター内のノードを正常な状態に維持するために、EKS にはノードモニタリングエージェントと自動ノード修復の機能があります。これらの機能は、EKS Auto Mode コンピューティングで自動的に有効になります。EKS マネージド型ノードグループと Karpenter で自動ノード修復を使用でき、AWS Fargate を除く任意の EKS コンピューティングタイプで EKS ノードモニタリングエージェントを使用することもできます。EKS ノードモニタリングエージェントと自動ノード修復は、一緒に使用すると最も効果的ですが、EKS クラスターで個別に使用することもできます。

> **Important**
>
> ノードモニタリングエージェントとノード自動修復を使用できるのは、Linux のみです。Windows では使用できません。

ノードモニタリングエージェント

EKS ノードモニタリングエージェントはノードログを読み取り、ヘルスの問題を検出します。ログを解析して障害を検出し、ノードのヘルスステータスに関するステータス情報を表示します。検出された問題のカテゴリごとに、エージェントはワーカーノード専用の `NodeCondition` を適用します。EKS ノードモニタリングエージェントによって検出されたノードのヘルス問題の詳細については、「the section called “ノードのヘルス検出”」を参照してください。

EKS Auto Mode コンピューティングには、ノードモニタリングエージェントが備わっています。他の EKS コンピューティングタイプでは、ノードモニタリングエージェントを EKS アドオンとして追加することも、Helm などの Kubernetes ツールで管理することもできます。詳細については、「the section called “ノードモニタリングエージェントを設定する”」を参照してください。

EKS ノードモニタリングエージェントでは、以下のカテゴリのノードのヘルス問題がノード条件として表示されます。`Ready`、`DiskPressure`、`MemoryPressure` は、EKS ノードモニタリングエージェントがない場合でも表示される標準の Kubernetes ノード条件です。

ノードの状態	説明
AcceleratedHardwareReady	AcceleratedHardwareReady は、ノードの高速ハードウェア (GPU、Neuron) が正しく機能しているかどうかを示します。
ContainerRuntimeReady	ContainerRuntimeReady は、コンテナランタイム (containerd など) が正しく機能し、コンテナを実行できるかどうかを示します。
DiskPressure	DiskPressure は、ノードにディスクプレッシャー (ディスク容量が少ない、または I/O が高い) が発生していることを示す標準の Kubernetes 条件です。

ノードの状態	説明
KernelReady	KernelReady は、重大なエラー、パニック、またはリソースの枯渇が発生することなく、カーネルが正しく機能しているかどうかを示します。
MemoryPressure	MemoryPressure は、ノードにメモリプレッシャー (使用可能なメモリが少ない) が発生していることを示す標準の Kubernetes 条件です。
NetworkingReady	NetworkingReady は、ノードのネットワークスタック (インターフェイス、ルーティング、接続) が正しく機能しているかどうかを示します。
StorageReady	StorageReady は、ノードのストレージサブシステム (ディスク、ファイルシステム、I/O) が正しく機能しているかどうかを示します。
Ready	Ready は、ノードが正常でポッドを受け入れる準備ができていることを示す標準の Kubernetes 条件です。

自動ノード修復

EKS 自動ノード修復は、ノードのヘルスを継続的にモニタリングし、検出された問題に対応し、可能な場合はノードを置き換えるか再起動します。これにより、手動介入を最小限に抑えながらクラスターの信頼性を向上させ、アプリケーションのダウンタイムを短縮できます。

単体では、EKS 自動ノード修復は、kubelet の Ready 条件、手動で削除されたノードオブジェクト、クラスターの参加に失敗した EKS マネージド型ノードグループインスタンスに対応します。ノードモニタリングエージェントをインストールした状態で EKS 自動ノード修復を有効にすると、EKS 自動ノード修復は次の追加のノード条件に対応します: AcceleratedHardwareReady、ContainerRuntimeReady、KernelReady、NetworkingReady、Sto

EKS 自動ノード修復は、標準の Kubernetes `DiskPressure`、`MemoryPressure`、または `PIDPressure` ノード条件には対応しません。これらの条件は、多くの場合、ノードレベルの障害ではなく、アプリケーションの動作、ワークロード設定、またはリソース制限の問題を示しているため、適切なデフォルトの修復アクションを決定することは困難です。これらのシナリオでは、ワークロードは Kubernetes ノードのプレッシャーエビクション動作の影響を受けます。

EKS 自動ノード修復の詳細については、「the section called “自動ノード修復”」を参照してください。

EKS ノードモニタリングエージェントのノードのヘルス問題を検出する

このトピックでは、EKS ノードモニタリングエージェントによって検出されたノードのヘルス問題、それらの問題がノード条件またはイベントとしてどのように表示されるか、ノードモニタリングエージェントを設定する方法について説明します。

EKS ノードモニタリングエージェントは、EKS 自動ノード修復の有無にかかわらず使用できます。EKS 自動ノード修復の詳細については、「the section called “自動ノード修復”」を参照してください。

EKS ノードモニタリングエージェントのソースコードは、GitHub の aws/eks-node-monitoring-agent リポジトリに公開されています。

ノードのヘルスの問題

次の表は、ノードモニタリングエージェントが検出できるノードのヘルス問題を示しています。次の 2 種類の問題があります。

- 症状 – インスタンスの置き換えや再起動などの修復アクションを必要とするターミナルの問題。自動修復が有効になっている場合、Amazon EKS はノードの置換または再起動として修復アクションを実行します。詳細については、「the section called “ノードの状態”」を参照してください。
- イベント – 一時的な問題または最適ではないノード設定。自動修復アクションは行われません。詳細については、「the section called “ノードイベント”」を参照してください。

AcceleratedHardware ノードのヘルス問題

以下の表に記載されている、緊急度が「条件」である問題について、モニタリング条件は `AcceleratedHardwareReady` です。以下の表のイベントと条件は、NVIDIA および Neuron 関連

のノードのヘルス問題に関するものです。再起動による修復アクションは、EKS マネージド型ノードグループにのみ適用されます。EKS Auto Mode または Karpenter を使用する場合、修復アクションは再起動のみとなります。

名前	緊急度	説明	修復アクション
DCGMDiagnosticFailure	条件	DCGM アクティブ診断テストスイートからのテストケースが失敗しました。	なし
DCGMError	条件	DCGM ホストプロセスへの接続を失ったか、確立できませんでした。	なし
DCGMFieldError[Code]	イベント	DCGM が、フィールド識別子を介して GPU の劣化を検出しました。	なし
DCGMHealthCode[Code]	イベント	DCGM ヘルスチェックが、致命的ではない方法で失敗しました。	なし
DCGMHealthCode[Code]	条件	DCGM ヘルスチェックが、致命的な方法で失敗しました。	なし
NeuronDMAError	条件	DMA エンジンで回復不可能なエラーが発生しました。	置換
NeuronHBMUncorrectableError	条件	HBM で修正不可能なエラーが発生し、誤った結果が生成されました。	置換

名前	緊急度	説明	修復アクション
NeuronNCUncorrectableError	条件	Neuron Core の修正不可能なメモリエラーが検出されました。	置換
NeuronSRAMUncorrectableError	条件	オンチップ SRAM でパリティエラーが発生し、誤った結果が生成されました。	置換
NvidiaDeviceCountMismatch	イベント	NVML で認識される GPU の数が、ファイルシステムの NVIDIA デバイス数と一致していません。	なし
NvidiaDoubleBitError	条件	GPU ドライバーによってダブルビットエラーが生成されました。	置換
NvidiaNCCLError	イベント	NVIDIA Collective Communications ライブラリ (`libnccl`) でセグメンテーション違反が発生しました。	なし
NvidiaNVLinkError	条件	NVLink エラーが GPU ドライバーによって報告されました。	置換
NvidiaPCIeError	イベント	送信エラーから回復するために PCIe リプレイがトリガーされました。	なし

名前	緊急度	説明	修復アクション
NvidiaPageRetirement	イベント	GPU ドライバーがメモリページを廃止としてマークしました。これは、1 つのダブルビットエラーが起こった場合、または同じアドレスで 2 つのシングルビットエラーが起こった場合に発生する可能性があります。	なし
NvidiaPowerError	イベント	GPU の電力使用率が許容しきい値を超えました。	なし
NvidiaThermalError	イベント	GPU の熱ステータスが許容しきい値を超えました。	なし
NvidiaXID[Code]Error	条件	重大な GPU エラーが発生しました。	置換または再起動
NvidiaXID[Code]Warning	イベント	重要ではない GPU エラーが発生しました。	なし

NVIDIA XID エラーコード

ノードモニタリングエージェントは、GPU カーネルログから NVIDIA XID エラーを検出します。XID エラーは 2 つのカテゴリに分類されます。

- 既知の XID コード – ノード条件 (`AcceleratedHardwareReady=False`) を設定し、有効になっている場合に自動修復をトリガーする重大なエラー。理由コードの形式は `NvidiaXID[Code]Error` です。EKS ノードモニタリングエージェントが検出する既知の XID

コードは、修復アクションを必要とする NVIDIA XID コードの完全なリストを表していない場合があります。

- 不明な XID コード – Kubernetes イベントとしてのみログに記録されます。これらは自動修復をトリガーしません。理由コードの形式は `NvidiaXID[Code]Warning` です。不明な XID エラーを調査するには、`dmesg | grep -i nvrm` でカーネルログを確認します。

XID エラーの詳細については、「NVIDIA GPU のデプロイおよび管理ドキュメント」の「Xid エラー」を参照してください。個々の XID メッセージの詳細については、「NVIDIA GPU Deployment and Management Documentation」の「Understanding Xid Messages」を参照してください。

以下の表に、既知の XID コード、その意味、および有効になっている場合のデフォルトのノード修復アクションを示します。再起動による修復アクションは、EKS マネージド型ノードグループにのみ適用されます。EKS Auto Mode または Karpenter を使用する場合、修復アクションは再起動のみとなります。

XID コード	説明	修復アクション
46	GPU が処理を停止 – GPU は内部タイムアウトのために処理を停止しました。復旧するには GPU のリセットが必要です。	リブート
48	ダブルビット ECC エラー – GPU メモリで修正不可能なダブルビットエラーが発生しました。これはハードウェアの劣化の可能性を示しています。	リブート
54	補助電源が接続されていない — 補助電源が GPU ボードに接続されていません。通常、電源コネクタが正しく配置されていないことを示しています。	リブート
62	内部マイクロコントローラーの停止 – GPU の内部マイクロコントローラーが停止しました。これはファームウェアまたはハードウェアの障害を示しており、GPU のリセットが必要です。	リブート
63	GPU メモリの再マッピングイベント – 検出されたエラーにより、GPU ドライバーが GPU メモリの一部を再マッピングしました。これは多くの場合、回復可能です。	リブート

XID コード	説明	修復アクション
64	GPU メモリの再マッピングの失敗 – GPU が欠陥メモリを再マッピングできませんでした。これはハードウェアの問題を示しています。	置換
74	NVLink エラー – GPU 間の高速 NVLink 相互接続でエラーが発生しました。	置換
79	GPU がバスから切断された – GPU は PCIe 経由でアクセスできなくなりました。これは通常、ハードウェアの障害や電源の問題を示しています。	置換
95	非限定的なメモリエラー – メモリエラーが発生し、他のアプリケーションやシステムメモリに影響を与えた可能性があります。	リブート
109	コンテキストスイッチのタイムアウト – GPU コンテキストスイッチが時間内に完了しませんでした。GPU がハングしており、リセットが必要であることを示しています。	リブート
110	セキュリティ障害エラー – GPU でセキュリティ障害が検出されました。GPU のリセットが必要であり、問題が解決しない場合はハードウェアの調査が必要です。	リブート
119	GSP RPC タイムアウト – GPU システムプロセッサとの通信がタイムアウトしました。これは、ファームウェアの問題が原因である可能性があります。	置換
120	GSP エラー – GPU システムプロセッサでエラーが発生しました。	置換
136	リンクトレーニングに失敗 – GPU はリンクトレーニング中に正常な NVLink 接続を確立できませんでした。	リブート
140	ECC 未回復エラー – ECC エラーが検出されず、データが破損した可能性があります。	リブート

XID コード	説明	修復アクション
142	NVENC3 エラー – NVENC3 ビデオエンコーダーハードウェアが失敗し、自動復旧が利用できません。GB200 に適用されます。	置換
143	GPU 初期化エラー – GPU が正しく初期化されませんでした。	リブート
151	キーローテーションエラー – 機密コンピューティングキーのローテーションに失敗しました。H100、B100、GB200 に適用されます。	置換
155	NVLink ソフトウェア定義エラー – ソフトウェア定義エラーが NVLink で報告されました。	リブート
156	リソースリタイアイベント – GPU は、検出された障害のためにハードウェアリソースをリタイアしました。	リブート
158	GPU の致命的なタイムアウト – GPU で致命的なタイムアウトが発生しました。	リブート

GPU ヘルスに関連する現在のノード条件を表示するには、以下のコマンドを実行します。

```
kubectl get nodes -o custom-
columns='NAME:.metadata.name,ACCELERATOR_READY:.status.conditions[?
(@.type=="AcceleratedHardwareReady")].status,REASON:.status.conditions[?
(@.type=="AcceleratedHardwareReady")].reason'
```

クラスターで XID 関連のイベントを表示するには、以下のいずれかのコマンドを実行します。

```
kubectl get events | grep -i "NvidiaXID"
```

ContainerRuntime ノードのヘルスの問題

以下の表に記載されている、緊急度が「条件」である問題について、モニタリング条件は `ContainerRuntimeReady` です。

名前	緊急度	説明	修復アクション
ContainerRuntimeFailed	イベント	コンテナランタイムはコンテナの作成に失敗しました。繰り返し発生する場合は、報告された問題に関連している可能性があります。	なし
DeprecatedContainerdConfiguration	イベント	廃止されたイメージマニフェストバージョン 2、スキーマ 1 を使用するコンテナイメージが、最近 containerd を介してノードにプルされました。	なし
KubeletFailed	イベント	kubelet が失敗状態になりました。	なし
LivenessProbeFailures	イベント	ライブネスプローブの障害が検出されました。アプリケーションコードの問題や、繰り返し発生する場合はタイムアウト値が不十分である可能性を示しています。	なし
PodStuckTerminating	条件	ポッドが終了しているか、または長時間停止していました。これは、ポッドの状態の進行を妨げる	置換

名前	緊急度	説明	修復アクション
		CRI エラーが原因で発生する可能性があります。	
ReadinessProbeFailures	イベント	レディネスプローブの障害が検出されました。アプリケーションコードの問題や、繰り返し発生する場合はタイムアウト値が不十分である可能性を示しています。	なし
[Name]RepeatedRestart	イベント	systemd ユニットが頻繁に再起動しています。	なし
ServiceFailedToStart	イベント	systemd ユニットの起動に失敗しました。	なし

カーネルノードのヘルスの問題

以下の表に記載されている、緊急度が「条件」である問題について、モニタリング条件は `KernelReady` です。

名前	緊急度	説明	修復アクション
AppBlocked	イベント	タスクが長時間スケジュールからブロックされており、通常は入力または出力がブロックされている	なし

名前	緊急度	説明	修復アクション
		ことが原因で発生します。	
AppCrash	イベント	ノード上のアプリケーションがクラッシュしました。	なし
ApproachingKernelPidMax	イベント	プロセスの数が、現在の kernel.pid_max 設定で使用可能な PID の最大数に近づいています。最大数に到達すると、これ以上プロセスを起動することはできません。	なし
ApproachingMaxOpenFiles	イベント	開いているファイルの数は、現在のカーネル設定で可能な開いているファイルの最大数に近づいています。最大数に到達すると、新しいファイルを開くことができなくなります。	なし
ConntrackExceededKernel	イベント	接続追跡がカーネルの最大数を超え、新しい接続を確立できなかったため、パケットロスが発生する可能性があります。	なし

名前	緊急度	説明	修復アクション
ExcessiveZombiePro cesses	イベント	完全に再要求できないプロセスが大量に蓄積されています。これはアプリケーションの問題を示しており、システムプロセスの制限に達する可能性があります。	なし
ForkFailedOutOfPIDs	条件	フォークまたは実行の呼び出しが失敗したのは、システムがプロセス ID またはメモリ不足であるためです。これは、ゾンビプロセスまたは物理メモリの枯渇が原因である可能性があります。	置換
KernelBug	イベント	CPU またはメモリの使用率が高いノードが原因でイベント処理が遅れることがありますが、カーネルのバグが Linux カーネル自体によって検出および報告されました。	なし

名前	緊急度	説明	修復アクション
LargeEnvironment	イベント	このプロセスの環境変数の数は予想よりも多く、enableServiceLinks が true に設定されている多くのサービスが原因で発生する可能性があります。このため、パフォーマンスの問題が発生する可能性があります。	なし
RapidCron	イベント	このノードでは、cron ジョブが 5 分間隔よりも速く実行されています。このため、ジョブが大量のリソースを消費すると、パフォーマンスに影響が出る可能性があります。	なし
SoftLockup	イベント	CPU が一定時間停止しました。	なし

ネットワークノードのヘルスの問題

以下の表に記載されている、緊急度が「条件」である問題について、モニタリング条件は NetworkingReady です。

名前	緊急度	説明	修復アクション
BandwidthInExceeded	イベント	インバウンド集計帯域幅がインスタンスの最大値を超えたた	なし

名前	緊急度	説明	修復アクション
		め、パケットはキューに入れられたか、ドロップされました。	
BandwidthOutExceeded	イベント	アウトバウンド集計帯域幅がインスタンスの最大値を超えたため、パケットはキューに入れられたか、ドロップされました。	なし
ConntrackExceeded	イベント	接続追跡がインスタンスの最大数を超え、新しい接続を確立できなかったため、パケットロスが発生する可能性があります。	なし
EFAErrorMetric	イベント	EFA ドライバーメトリクスは、パフォーマンスが低下しているインターフェイスがあることを示しています。	なし
IPAMDInconsistentState	イベント	ディスク上の IPAMD チェックポイントの状態が、コンテナランタイム内の IP を反映していません。	なし
IPAMDNoIPs	イベント	IPAMD が IP アドレスから外れています。	なし

名前	緊急度	説明	修復アクション
IPAMDNotReady	条件	IPAMD が API サーバーに接続できません。	置換
IPAMDNotRunning	条件	Amazon VPC CNI プロセスは実行中であることが確認されませんでした。	置換
IPAMDRepeatedlyRestart	イベント	IPAMD サービスで複数の再起動が発生しました。	なし
InterfaceNotRunning	条件	このインターフェイスは実行されていないか、ネットワークの問題があります。	置換
InterfaceNotUp	条件	このインターフェイスは起動していないか、ネットワークの問題があります。	置換
KubeProxyNotReady	イベント	Kube-proxy がリソースの監視または一覧表示に失敗しました。	なし
LinkLocalExceeded	イベント	ローカルプロキシサービスへのトラフィックの PPS がネットワークインターフェイスの最大値を超えたため、パケットがドロップされました。	なし

名前	緊急度	説明	修復アクション
MACAddressPolicyMisconfigured	イベント	systemd-networkd のリンク設定で、`MACAddressPolicy` の値が正しくありません。	なし
MissingDefaultRoutes	イベント	デフォルトのルートルールが見つかりません。	なし
MissingIPRoutes	イベント	Pod IP のルートがありません。	なし
MissingIPRules	イベント	Pod IP のルールがありません。	なし
MissingLoopbackInterface	条件	このインスタンスには、ループバックインターフェイスがないため、ローカル接続によってはサービスの不具合が発生します。	置換
NetworkSysctl	イベント	このノードのネットワーク `sysctl` 設定が正しくない可能性があります。	なし
PPSExceeded	イベント	双方向 PPS がインスタンスの最大値を超えたため、パケットはキューに入れられたか、ドロップされました。	なし

名前	緊急度	説明	修復アクション
PortConflict	イベント	ポッドが hostPort を使用する場合、ホストの既にバインドされているポートを上書きするように iptables ルールを書き込みできるため、API サーバーが kubelet にアクセスできなくなる可能性があります。	なし
UnexpectedRejectRule	イベント	iptables で予期しない REJECT または DROP ルールが検出されたため、予想されたトラフィックがブロックされる可能性があります。	なし

ストレージノードのヘルスの問題

以下の表に記載されている、緊急度が「条件」である問題について、モニタリング条件は StorageReady です。

名前	緊急度	説明	修復アクション
EBSInstanceIOPSExceeded	イベント	インスタンスの最大 IOPS を超えました。	なし
EBSInstanceThroughputExceeded	イベント	インスタンスの最大スループットを超えました。	なし

名前	緊急度	説明	修復アクション
EBSVolume IOPSExceeded	イベント	特定の EBS ボリュームに対する最大 IOPS を超えました。	なし
EBSVolume ThroughputExceeded	イベント	特定の Amazon EBS ボリュームに対する最大スループットを超えました。	なし
EtcHostsMountFailed	イベント	kubelet-container オペレーション中にユーザーデータを /var/lib/kubelet/pods に再マウントしたため、kubelet が生成した /etc/hosts のマウントに失敗しました。	なし
IODelays	イベント	入力または出力の遅延がプロセスで検出されました。過剰な場合は入力/出力プロビジョニングが不十分である可能性があります。	なし

名前	緊急度	説明	修復アクション
KubeletDiskUsageSlow	イベント	kubelet は、ファイルシステムにアクセスしようとしているときにディスクの使用が遅いことを報告しています。これは、ディスクの入出力が不十分である、またはファイルシステムに問題がある可能性を示しています。	なし
XFSSmallAverageClusterSize	イベント	XFS 平均クラスターサイズが小さいことは、空き領域が過度に断片化されていることを示しています。このため、inode や空き領域が使用可能であっても、ファイルを作成できない場合があります。	なし

ノードモニタリングエージェントを設定する

EKS ノードモニタリングエージェントは DaemonSet としてデプロイされます。EKS アドオンとしてデプロイする場合、以下の設定値を使用してインストールをカスタマイズできます。デフォルト設定については、EKS ノードモニタリングエージェントの Helm チャートを参照してください。

設定オプション	説明
monitoringAgent.resources.requests.cpu	モニタリングエージェントの CPU リソースリクエスト。

設定オプション	説明
monitoringAgent.resources.requests.memory	モニタリングエージェントのメモリリソースリクエスト。
monitoringAgent.resources.limits.cpu	モニタリングエージェントの CPU リソース制限。
monitoringAgent.resources.limits.memory	モニタリングエージェントのメモリリソース制限。
monitoringAgent.tolerations	テイントが適用されたノードでモニタリングエージェントをスケジュールするための許容範囲。
monitoringAgent.additionalArgs	モニタリングエージェントに渡す追加のコマンドライン引数。

> **Note**
>
> EKS アドオンまたは Helm インストールを使用して、hostname-override と verbosity を monitoringAgent.additionalArgs として設定できます。現在、EKS アドオンまたは Helm インストールでは、追加の引数を使用してノードモニタリングエージェントの probe-address (8002) または metrics-address (8003) をカスタマイズすることはできません。

ノードモニタリングエージェントには、NVIDIA GPU をモニタリングするための NVIDIA DCGM (Data Center GPU Manager) サーバーコンポーネント (nv-hostengine) が含まれています。このコンポーネントは、エージェントの Helm チャートの nodeAffinity で示されるように、NVIDIA GPU インスタンスタイプのノードでのみ実行されます。EKS ノードモニタリングエージェントで既存の NVIDIA DCGM インストールを使用することはできません。この機能が必要な場合は、EKS ロードマップ GitHub 問題 #2763 に関するフィードバックを提供してください。

EKS ノードモニタリングエージェントを EKS アドオンとしてデプロイする場合、以下の設定値を使用して NVIDIA DCGM のインストールをカスタマイズできます。

設定オプション	説明
dcgmAgent.resources.requests.cpu	DCGM エージェントの CPU リソースリクエスト。
dcgmAgent.resources.requests.memory	DCGM エージェントのメモリリソースリクエスト。
dcgmAgent.resources.limits.cpu	DCGM エージェントの CPU リソース制限。
dcgmAgent.resources.limits.memory	DCGM エージェントのメモリリソース制限。
dcgmAgent.tolerations	テイントが適用されたノードで DCGM エージェントをスケジュールするための許容範囲。

以下の AWS CLI コマンドを使用して、EKS ノードモニタリングエージェントの EKS アドオンのバージョンとスキーマに関する有用な情報を取得できます。

Kubernetes バージョンの最新のエージェントアドオンバージョンを取得します。1.35 を Kubernetes バージョンに置き換えます。

```
aws eks describe-addon-versions \
  --addon-name eks-node-monitoring-agent \
  --kubernetes-version 1.35 \
  --query='addons[].addonVersions[].addonVersion'
```

EKS アドオンでサポートされているエージェントアドオンスキーマを取得します。v1.5.1-eksbuild.1 をエージェントバージョンに置き換えます。

```
aws eks describe-addon-configuration \
  --addon-name eks-node-monitoring-agent \
  --addon-version v1.5.1-eksbuild.1
```

EKS クラスター内のノードを自動的に修復する

このトピックでは、EKS 自動ノード修復の動作と、要件を満たすように設定する方法について詳しく説明します。EKS 自動ノード修復は EKS Auto Mode でデフォルトで有効になっており、EKS マネージド型ノードグループと Karpenter で使用できます。

デフォルトの EKS 自動ノード修復アクションは、以下の表にまとめられており、EKS Auto Mode、EKS マネージド型ノードグループ、Karpenter の動作に適用されます。EKS Auto Mode または Karpenter を使用する場合、すべての `AcceleratedHardwareReady` 修復アクションは `Replace` であり、EKS マネージド型ノードグループのみが修復アクションとして `Reboot` をサポートします。

EKS ノードモニタリングエージェントによって検出されたノードのヘルス問題とその対応するノード修復アクションの詳細なリストについては、「the section called “ノードのヘルス検出”」を参照してください。

ノードの状態	説明	修復までの時間	修復アクション
AcceleratedHardwareReady	AcceleratedHardwareReady は、ノードの高速ハードウェア (GPU、Neuron) が正しく機能しているかどうかを示します。	10m	置換または再起動
ContainerRuntimeReady	ContainerRuntimeReady は、コンテナランタイム (containerd など) が正しく機能し、コンテナを実行できるかどうかを示します。	30m	置換
DiskPressure	DiskPressure は、ノードにディスクプレッシャー (ディスク容量が少ない、または I/O が高い) が発生	該当なし	なし

ノードの状態	説明	修復までの時間	修復アクション
	していることを示す標準の Kubernetes 条件です。		
KernelReady	KernelReady は、重大なエラー、パニック、またはリソースの枯渇が発生することなく、カーネルが正しく機能しているかどうかを示します。	30m	置換
MemoryPressure	MemoryPressure は、ノードにメモリプレッシャー (使用可能なメモリが少ない) が発生していることを示す標準の Kubernetes 条件です。	該当なし	なし
NetworkingReady	NetworkingReady は、ノードのネットワークスタック (インターフェイス、ルーティング、接続) が正しく機能しているかどうかを示します。	30m	置換
StorageReady	StorageReady は、ノードのストレージサブシステム (ディスク、ファイルシステム、I/O) が正しく機能しているかどうかを示します。	30m	置換

ノードの状態	説明	修復までの時間	修復アクション
Ready	Ready は、ノードが正常でポッドを受け入れる準備ができていることを示す標準の Kubernetes 条件です。	30m	置換

EKS 自動ノード修復アクションは、以下のシナリオではデフォルトで無効になっています。進行中のノード修復アクションは、各シナリオで続行されます。これらのデフォルト設定を上書きする方法については、「the section called “自動ノード修復を設定する”」を参照してください。

EKS マネージド型ノードグループ

- ノードグループに 5 つを超えるノードがあり、そのうち 20% を超えるノードに異常があります。
- クラスターのゾーンシフトは、Application Recovery Controller (ARC) を介してトリガーされます。

EKS Auto Mode と Karpenter

- NodePool 内のノードのうち、20% を超えるノードに異常があります。
- スタンドアロン NodeClaims の場合、クラスター内のノードの 20% に異常があります。

自動ノード修復を設定する

EKS Auto Mode を使用する場合、自動ノード修復を設定することはできません。自動ノード修復は常に Karpenter と同じデフォルト設定で有効になります。

Karpenter

Karpenter で自動ノード修復を使用するには、機能ゲート `NodeRepair=true` を有効にします。機能ゲートは、Karpenter デプロイの `--feature-gates` CLI オプションまたは `FEATURE_GATES` 環境変数を使用して有効にできます。詳細については、Karpenter のドキュメントを参照してください。

マネージドノードグループ

新しい EKS マネージド型ノードグループを作成するとき、または既存の EKS マネージド型ノードグループを更新することで、自動ノード修復を有効にできます。

- Amazon EKS コンソール – マネージド型ノードグループの [ノード自動修復を有効にする] チェックボックスを選択します。詳細については、「the section called “作成”」を参照してください。
- AWS CLI – `--node-repair-config enabled=true` を `eks create-nodegroup` または `eks update-nodegroup-config` コマンドに追加します。
- eksctl – `managedNodeGroups.nodeRepairConfig.enabled: true` を設定します。eksctl GitHub の例を参照してください。

EKS マネージド型ノードグループを使用する場合、以下の設定でノード自動修復の動作を制御できます。

ノード自動修復がいつアクションを停止するかを制御するには、ノードグループ内の異常のあるノードの数に基づいてしきい値を設定します。絶対数またはパーセンテージのいずれかを設定します。両方を設定することはできません。

設定	説明
`maxUnhealthyNodeThresholdCount`	ノード自動修復が停止する異常ノードの絶対数。これを使用して、修復の範囲を制限します。
`maxUnhealthyNodeThresholdPercentage`	ノード自動修復が停止する異常ノードの割合 (0 ~ 100)。

同時に修復するノードの数を制御するには、修復並列処理を設定できます。異常のあるノードしきい値と同様に、絶対数またはパーセンテージのいずれかを設定します。両方を設定することはできません。

設定	説明
`maxParallelNodesRepairedCount`	同時に修復するノードの最大数。

設定	説明
maxParallelNodesRepairedPercentage	同時に修復する異常ノードの最大パーセンテージ (0～100)。

nodeRepairConfigOverrides では、特定の条件に合わせて修復動作をカスタマイズできます。これは、さまざまな問題タイプに対して異なる修復アクションまたは待機時間が必要な場合に使用します。

上書きにはそれぞれ、以下のすべてのフィールドが必要です。

フィールド	説明
nodeMonitoringCondition	ノードモニタリングエージェントによって報告されたノードの状態タイプ。例えば、AcceleratedHardwareReady 、NetworkingReady 、StorageReady 、KernelReady などです。
nodeUnhealthyReason	異常状態の特定の理由コード。例: NvidiaXID31Error 、IPAMDNotRunning 。
minRepairWaitTimeMins	ノードが修復対象となるまでに、その状態が継続している必要がある最小時間 (分単位)。これは、一時的な問題のためにノードを修復しないようにする場合に使用します。
repairAction	条件が満たされたときに実行するアクション。有効な値: Replace (ノードを終了して置き換える)、Reboot (ノードを再起動する)、または NoAction (修復アクションなし)。

以下の AWS CLI の例では、カスタム修復設定を使用してノードグループを作成します。

```
aws eks create-nodegroup \
  --cluster-name my-cluster \
  --nodegroup-name my-nodegroup \
  --node-role arn:aws:iam::111122223333:role/NodeRole \
```

```
  --subnets subnet-0123456789abcdef0 \
  --node-repair-config '{
    "enabled": true,
    "maxUnhealthyNodeThresholdPercentage": 10,
    "maxParallelNodesRepairedCount": 3,
    "nodeRepairConfigOverrides": [
      {
        "nodeMonitoringCondition": "AcceleratedHardwareReady",
        "nodeUnhealthyReason": "NvidiaXID64Error",
        "minRepairWaitTimeMins": 5,
        "repairAction": "Replace"
      },
      {
        "nodeMonitoringCondition": "AcceleratedHardwareReady",
        "nodeUnhealthyReason": "NvidiaXID31Error",
        "minRepairWaitTimeMins": 15,
        "repairAction": "NoAction"
      }
    ]
  }'
```

この設定では、以下のことが行われます。

- ノード自動修復を有効にする
- 10% を超えるノードに異常がある場合に修復アクションを停止する
- 一度に最大 3 つのノードを修復する
- XID 64 エラー (GPU メモリの再マッピングの失敗) を上書きして、5 分後にノードを置き換える。デフォルトでは 10 分後に再起動します。
- XID 31 エラー (GPU メモリページの障害) を上書きして、アクションを実行しない。デフォルトでは 10 分後に再起動します。

ノードのヘルスステータスを表示する

このトピックでは、Amazon EKS クラスターのノードのヘルスステータスをモニタリングするために使用可能なツールと方法について説明します。この情報には、ノードレベルの問題の特定と診断に役立つノードの状態、イベント、検出ケースが取り上げられます。ここで説明するコマンドとパターンを使用して、ノードのヘルスリソースを検査し、ステータス条件を解明し、ノードイベントを分析して運用上のトラブルシューティングを行います。

すべてのノードに対して Kubernetes コマンドを使用して、ノードのヘルス情報を取得できます。また、Amazon EKS Auto Mode または Amazon EKS マネージドアドオンを介してノードモニタリングエージェントを使用すると、トラブルシューティングに役立つさまざまなノードシグナルが得られます。ノードモニタリングエージェントによって検出されたヘルス問題の説明は、オブザーバビリティダッシュボードでも確認できます。詳細については、「the section called “ノードのヘルス検出”」を参照してください。

ノードの状態

ノードの状態は、インスタンスの置き換えや再起動などの修復アクションを必要とするターミナルの問題を表します。

すべてのノードの状態を取得するには:

```
kubectl get nodes -o 'custom-
columns=NAME:.metadata.name,CONDITIONS:.status.conditions[*].type,STATUS:.status.conditions[*].
```

特定のノードの詳細な状態を取得するには

```
kubectl describe node node-name
```

正常なノードの状態の出力の例:

```
  - lastHeartbeatTime: "2024-11-21T19:07:40Z"
    lastTransitionTime: "2024-11-08T03:57:40Z"
    message: Monitoring for the Networking system is active
    reason: NetworkingIsReady
    status: "True"
    type: NetworkingReady
```

ネットワーク形成の問題がある異常なノードの状態の例:

```
  - lastHeartbeatTime: "2024-11-21T19:12:29Z"
    lastTransitionTime: "2024-11-08T17:04:17Z"
    message: IPAM-D has failed to connect to API Server which could be an issue with
      IPTable rules or any other network configuration.
    reason: IPAMDNotReady
    status: "False"
    type: NetworkingReady
```

ノードイベント

ノードイベントは、一時的な問題または最適ではない設定を示します。

ノードモニタリングエージェントによって報告されたすべてのイベントを取得するには

ノードモニタリングエージェントが使用可能になったら、次のコマンドを実行できます。

```
kubectl get events --field-selector=reportingComponent=eks-node-monitoring-agent
```

サンプル出力:

```
LAST SEEN   TYPE      REASON       OBJECT
 MESSAGE
4s          Warning   SoftLockup   node/ip-192-168-71-251.us-west-2.compute.internal
 CPU stuck for 23s
```

すべてのノードのイベントを取得するには

```
kubectl get events --field-selector involvedObject.kind=Node
```

特定のノードのイベントを取得するには

```
kubectl get events --field-selector involvedObject.kind=Node,involvedObject.name=node-
name
```

イベントをリアルタイムで監視するには

```
kubectl get events -w --field-selector involvedObject.kind=Node
```

イベント出力の例:

```
LAST SEEN   TYPE     REASON           OBJECT         MESSAGE
2m          Warning  MemoryPressure   Node/node-1    Node experiencing memory pressure
5m          Normal   NodeReady        Node/node-1    Node became ready
```

一般的なトラブルシューティングのコマンド

```
# Get comprehensive node status
```

```
kubectl get node node-name -o yaml

# Watch node status changes
kubectl get nodes -w

# Get node metrics
kubectl top node
```

kubectl と S3 を使用してマネージドノードのノードログを取得する

ノードモニタリングエージェントを備えた Amazon EKS マネージドノードのノードログを取得する方法について説明します。

前提条件

以下があることを確認します。

- ノードモニタリングエージェントを備えた既存の Amazon EKS クラスター。詳細については、「the section called “ノードのヘルスと修復”」を参照してください。
- クラスターと通信するようにインストールおよび設定された kubectl コマンドラインツール。
- S3 バケットとオブジェクトを作成するのに十分なアクセス許可が付与された状態でインストールおよびログインした AWS CLI。
- インストール済みの Python 3 最新バージョン
- インストール済みの AWS SDK for Python 3、Boto 3。

ステップ 1: S3 バケットの宛先を作成する (任意)

ログを格納する S3 バケットがまだない場合は、作成します。次の AWS CLI コマンドを使用します。バケットのデフォルトは private アクセスコントロールリストです。*bucket-name* はユーザーが選択した一意のバケット名に置き換えます。

```
aws s3api create-bucket --bucket <bucket-name>
```

ステップ 2: HTTP Put の事前署名された S3 URL を作成する

Amazon EKS は、指定した URL に対して HTTP PUT オペレーションを実行してノードログを返します。このチュートリアルでは、事前署名された S3 HTTP PUT URL を生成します。

ログは gzip tarball として `.tar.gz` 拡張子で返されます。

> **Note**
>
> AWS API または SDK を使用して、EKS がログファイルをアップロードするための事前署名された S3 アップロード URL を作成する必要があります。AWS CLI を使用して事前署名された S3 アップロード URL を作成することはできません。

1. ログを格納するバケット内の場所を決定します。例えば、キーとして *`2024-11-12/logs1.tar.gz`* を使用できます。
2. *`presign-upload.py`* ファイルに次の Python コードを保存します。*`<bucket-name>`* と *`<key>`* を置き換えます。キーは `.tar.gz` で終わる必要があります。

```
import boto3; print(boto3.client('s3').generate_presigned_url(
   ClientMethod='put_object',
   Params={'Bucket': '<bucket-name>', 'Key': '<key>'},
   ExpiresIn=1000
))
```

3. 次を使用してスクリプトを実行します

```
python presign-upload.py
```

4. URL 出力をメモします。この値は、次のステップで *`http-put-destination`* として使用します。

詳細については、AWS Boto3 SDK for Python ドキュメントの「Generate a presigned URL to upload a file」を参照してください。

ステップ 3: NodeDiagnostic リソースを作成する

ログを収集するノードの名前を特定します。

リソースの名前としてノードの名前を使用する `NodeDiagnostic` マニフェストを作成し、HTTP PUT URL の宛先を指定します。

```
apiVersion: eks.amazonaws.com/v1alpha1
kind: NodeDiagnostic
```

```
metadata:
    name: <node-name>
spec:
    logCapture:
        destination: http-put-destination
```

マニフェストをクラスターに適用します。

```
kubectl apply -f nodediagnostic.yaml
```

NodeDiagnostic リソースを記述することで、コレクションのステータスを確認できます。

- ステータスが Success または SuccessWithErrors の場合は、タスクが完了し、ログが指定された宛先にアップロードされたことを示します (SuccessWithErrors は、一部のログが欠落している可能性があることを示します)。
- ステータスが Failure の場合は、アップロード URL が正しい形式であり、有効期限が切れていないことを確認してください。

```
kubectl describe nodediagnostics.eks.amazonaws.com/<node-name>
```

ステップ 4: S3 からログをダウンロードする

ログをダウンロードする前に約 1 分待ちます。そして、S3 CLI を使用してログをダウンロードします。

```
# Once NodeDiagnostic shows Success status, download the logs
aws s3 cp s3://<bucket-name>/key ./<path-to-node-logs>.tar.gz
```

ステップ 5: NodeDiagnostic リソースをクリーンアップする

- NodeDiagnostic リソースは自動的には削除されません。ログアーティファクトを取得したら、自分でクリーンアップする必要があります。

```
# Delete the NodeDiagnostic resource
kubectl delete nodediagnostics.eks.amazonaws.com/<node-name>
```

NodeDiagnostic **node** 宛先

ノードモニタリングエージェントのバージョン v1.6.1-eksbuild.1 以降では、ログ収集の宛先を node に設定するオプションがあります。この宛先を使用すると、ログは後で収集できるようにノード上に一時的に保持されます。この機能のほか、ノードモニタリングエージェントの GitHub リポジトリ内にインタラクションとログ収集を容易にする kubectl プラグインもあり、インストールして利用できます。詳細については、kubectl ekslogs プラグインのドキュメントを参照してください。

使用例

```
# Collect NodeDiagnostic logs from a single node
kubectl ekslogs <node-name>

# Collect NodeDiagnostic logs from multiple nodes
kubectl ekslogs <node-name-1> <node-name-2> <node-name-3>

# Collect NodeDiagnostic logs from all nodes with a specific label
kubectl ekslogs -l <key>=<value>
```

kubectl と S3 を使用して、マネージドノード上のネットワークトラフィックをキャプチャする

ノードモニタリングエージェントを備えた Amazon EKS マネージドノードで、ネットワークトラフィックをキャプチャする方法について説明します。エージェントはノードで tcpdump を実行し、キャプチャファイルを圧縮して、S3 バケットにアップロードします。

前提条件

以下があることを確認します。

- ノードモニタリングエージェントを備えた既存の Amazon EKS 自動モードクラスター。詳細については、the section called “ノードのヘルスと修復” を参照してください。
- クラスターと通信するようにインストールおよび設定された kubectl コマンドラインツール。
- S3 バケットとオブジェクトを作成するのに十分なアクセス許可が付与された状態でインストールおよびログインした AWS CLI。
- インストール済みの Python 3 最新バージョン

- インストール済みの AWS SDK for Python 3、Boto 3。
- PyYAML ライブラリがインストールされました (`pip install pyyaml`)。

ステップ 1: S3 バケットの宛先を作成する (任意)

キャプチャファイルを保存するための S3 バケットがまだない場合は、作成します。*bucket-name* と *region* をご自身の値に置き換えます。

```
aws s3api create-bucket --bucket <bucket-name> \
    --region <region> \
    --create-bucket-configuration LocationConstraint=<region>
```

> **Note**
>
> `--create-bucket-configuration` パラメータは、`us-east-1` を除くすべてのリージョンで必要です。

ステップ 2: パケットキャプチャを開始する

ノードモニタリングエージェントリポジトリ (`tools/start-capture.py`) の `start-capture.py` スクリプトを使用して、署名付き S3 認証情報を生成し、`NodeDiagnostic` リソースを作成してクラスターに適用します。

1. トラフィックをキャプチャしたいノードを特定します。

   ```
   kubectl get nodes
   ```

2. ノードモニタリングエージェントリポジトリからローカルマシンに start-capture.py スクリプトを保存して実行します。*<bucket-name>* と *<node-name>* をご自身の値に置き換えます。

   ```
   python3 start-capture.py --bucket <bucket-name> --node <node-name>
   ```

   一般的なオプション:

   ```
   # Capture for 5 minutes on eth0 with a filter
   ```

```
python3 start-capture.py --bucket <bucket-name> --node <node-name> \
    --duration 5m --interface eth0 --filter "tcp port 443"

# Preview the YAML without applying
python3 start-capture.py --bucket <bucket-name> --node <node-name> --dry-run
```

スクリプトには、 boto3 と pyyaml がインストールされ、クラスター用に kubectl 設定された Python 3 が必要です。

スクリプトは、次のような NodeDiagnostic リソースを生成します。この例は参照用です。upload フィールドには、スクリプトによってプログラムで生成された署名付き S3 POST 認証情報が必要であることに注意してください。

```
apiVersion: eks.amazonaws.com/v1alpha1
kind: NodeDiagnostic
metadata:
  name: <node-name>                    # Required: node instance ID
spec:
  packetCapture:
    duration: "30s"                      # Required: capture duration (max 1h)
    # interface: "eth0"                  # Optional: default is primary ENI. Use
 "any" for all interfaces
    # filter: "tcp port 443"             # Optional: tcpdump filter expression
    # chunkSizeMB: 10                    # Optional: file rotation size in MB (1-100,
 default: 10)
    upload:                              # Required: pre-signed S3 POST credentials
      url: "https://<bucket>.s3.amazonaws.com/"
      fields:
        key: "captures/<node-name>/${filename}"
        # ... other pre-signed POST fields (generated by the script)
```

ステップ 3: キャプチャの進行状況を監視する

キャプチャのステータスを確認します。

```
kubectl describe nodediagnostic <node-name>
```

ステータスには以下が表示されます。

- Running はキャプチャ処理中に動作します。

- キャプチャが完了し、すべてのファイルがアップロードされた時点で、理由 Success を伴う Completed。
- キャプチャ中にエラーが発生した場合は、理由 Failure とともに Completed と表示されます。

captureID (S3 パス識別に使用される) を含む完全なステータスを表示するには、以下を参照してください。

```
kubectl get nodediagnostic <node-name> -o jsonpath='{.status.captureStatuses}'
```

ステップ 4: S3 からキャプチャファイルをダウンロードする

ステータスが Success になったら、S3 からキャプチャファイルをダウンロードします。

```
aws s3 cp s3://<bucket-name>/captures/ ./captures/ --recursive
```

ファイルは、gzip 圧縮された pcap 形式です。tcpdump または Wireshark を使用して解凍および分析します。

```
gunzip captures/*.gz
tcpdump -r captures/capture.pcap0000 -n
```

ステップ 5: クリーンアップ

NodeDiagnostic リソースは、自動的には削除されません。キャプチャファイルを取得したら、クリーンアップします。キャプチャ実行中にリソースを削除すると、キャプチャは直ちに停止します。

```
kubectl delete nodediagnostic <node-name>
```

設定オプションと動作

完全な packetCapture 仕様リファレンス、設定オプション、動作の詳細については、ノードモニタリングエージェントリポジトリの パケットキャプチャドキュメント を参照してください。

Amazon EKS Hybrid Nodes の概要

Amazon EKS Hybrid Nodes では、オンプレミスおよびエッジインフラストラクチャを Amazon EKS クラスターのノードとして使用できます。AWS は、Amazon EKS クラスターの AWS ホスト型

Kubernetes コントロールプレーンを管理し、オンプレミス環境またはエッジ環境で実行されるハイブリッドノードを管理します。これにより、Kubernetes 管理が環境全体で統合され、Kubernetes コントロールプレーン管理がオンプレミスおよびエッジアプリケーションの AWS にオフロードされます。

Amazon EKS Hybrid Nodes は、オンプレミスのハードウェアまたは仮想マシンと連携し、アプリケーションを実行する必要がある場所に Amazon EKS の効率、スケーラビリティ、可用性をもたらします。Amazon EKS Hybrid Nodes では、Amazon EKS アドオン、Amazon EKS Pod Identity、クラスターアクセスエントリ、クラスターインサイト、Kubernetes バージョンの延長サポートといったさまざまな Amazon EKS 機能を使用できます。Amazon EKS Hybrid Nodes は AWS Systems Manager、AWS IAM Roles Anywhere、Amazon Managed Service for Prometheus、Amazon CloudWatch などの AWS サービスとネイティブに統合し、一元化したモニタリング、ログ記録、ID 管理を実現します。

Amazon EKS Hybrid Nodes では、前払いの義務や最低料金はなく、ハイブリッドノードの vCPU リソースが Amazon EKS クラスターにアタッチされると、1 時間ごとに課金されます。料金の詳細については、「Amazon EKS の料金表」を参照してください。

機能

EKS Hybrid Nodes には、次の高度な機能があります。

- マネージド型 Kubernetes のコントロールプレーン: AWS がホストする、EKS クラスターの Kubernetes コントロールプレーンは AWS が管理し、オンプレミス環境またはエッジ環境で実行されるハイブリッドノードはユーザーが管理します。これにより、Kubernetes 管理が環境全体で統合され、Kubernetes コントロールプレーン管理がオンプレミスおよびエッジアプリケーションの AWS にオフロードされます。Kubernetes コントロールプレーンを AWS に移動させることで、ユーザーはアプリケーションのオンプレミス容量を節約し、ワークロードのスケーリングは Kubernetes コントロールプレーンに任せることができます。
- 一貫性のある EKS エクスペリエンス: ほとんどの EKS 機能は EKS Hybrid Nodes でサポートされており、EKS アドオン、EKS Pod Identity、クラスターアクセスエントリ、クラスターインサイト、Kubernetes バージョンの拡張サポートその他の、オンプレミス環境とクラウド環境を横断する、一貫性のある EKS エクスペリエンスを実現しています。EKS Hybrid Nodes でサポートされている EKS アドオンの詳細については、「the section called “CNI アドオンの設定”」を参照してください。
- オブザーバビリティと ID 管理を一元化: EKS Hybrid Nodes は、AWS Systems Manager、AWS IAM Roles Anywhere、Amazon Managed Service for Prometheus、Amazon CloudWatch などの AWS サービスとネイティブに統合されており、モニタリング、ログ記録、ID 管理を一元的に行えます。
- Burst-to-cloud またはオンプレミス容量の追加: EKS クラスターを 1 つ使うことで、AWS リージョン、AWS Local Zones、AWS Outposts のいずれかでハイブリッドノードおよびノードを実行し、EKS クラスターに Burst-to-cloud する、またはオンプレミス容量を追加することができます。詳細については、「Considerations for mixed mode clusters」を参照してください。
- 柔軟なインフラストラクチャ: EKS Hybrid Nodes は Bring Your Own Infrastructure のアプローチに従い、ユーザーがハイブリッドノードに使用しているインフラストラクチャに依存しません。ハイブリッドノードを物理マシンまたは仮想マシン、および x86 と ARM のアーキテクチャで実行できるため、ハイブリッドノードで実行されているオンプレミスワークロードを、さまざまなインフラストラクチャタイプに移行することができます。
- 柔軟なネットワーキング: EKS Hybrid Nodes を使用すると、EKS コントロールプレーンとハイブリッドノード間の通信は、クラスターの作成時にユーザーが渡す VPC とサブネットを介してルーティングされます。これは、コントロールプレーンからノードへのネットワーキング向けに既存のメカニズムの上に構築されます。これにより、ユーザーが希望する、オンプレミスネットワークを AWS の VPC に接続する方法に、柔軟に対応できます。AWS Site-to-Site VPN、AWS Direct Connect、あるいはユーザー独自の VPN ソリューションなど、文書化されたオプションを複数利用でき、ユースケースに最適な方法を選択できます。

制限

- クラスターごとにリモートノードネットワークには最大 15 CIDR がサポートされ、リモートポッドネットワークには最大 15 CIDR がサポートされています。

考慮事項

- EKS Hybrid Nodes は、新規の EKS クラスターでも既存の EKS クラスターでも使用できます。
- EKS Hybrid Nodes は、AWS GovCloud (米国) リージョンと AWS 中国リージョンを除くすべての AWS リージョンで使用できます。
- EKS Hybrid Nodes には、オンプレミス環境と AWS の間に信頼性の高い接続が必要です。EKS Hybrid Nodes は、切断された環境、中断しているまたは一時的に停止している環境、制限 (DDIL) されている環境には適していません。DDIL 環境で実行している場合は、Amazon EKS Anywhere を検討してください。ネットワーク切断シナリオでハイブリッドノードの動作に関する詳細については、「EKS Hybrid Nodes のベストプラクティス」を参照してください。
- EKS Hybrid Nodes をクラウドインフラストラクチャ (AWS リージョン、AWS Local Zones、AWS Outposts) やその他のクラウドで実行することは、サポートされていません。Amazon EC2 インスタンスでハイブリッドノードを実行すると、ハイブリッドノード料金が課金されます。
- ハイブリッドノードの請求は、ノードが EKS クラスターに追加されたとに開始し、ノードがクラスターから削除されたときに停止します。ハイブリッドノードを使用しないときは、必ず EKS クラスターからこれを削除してください。

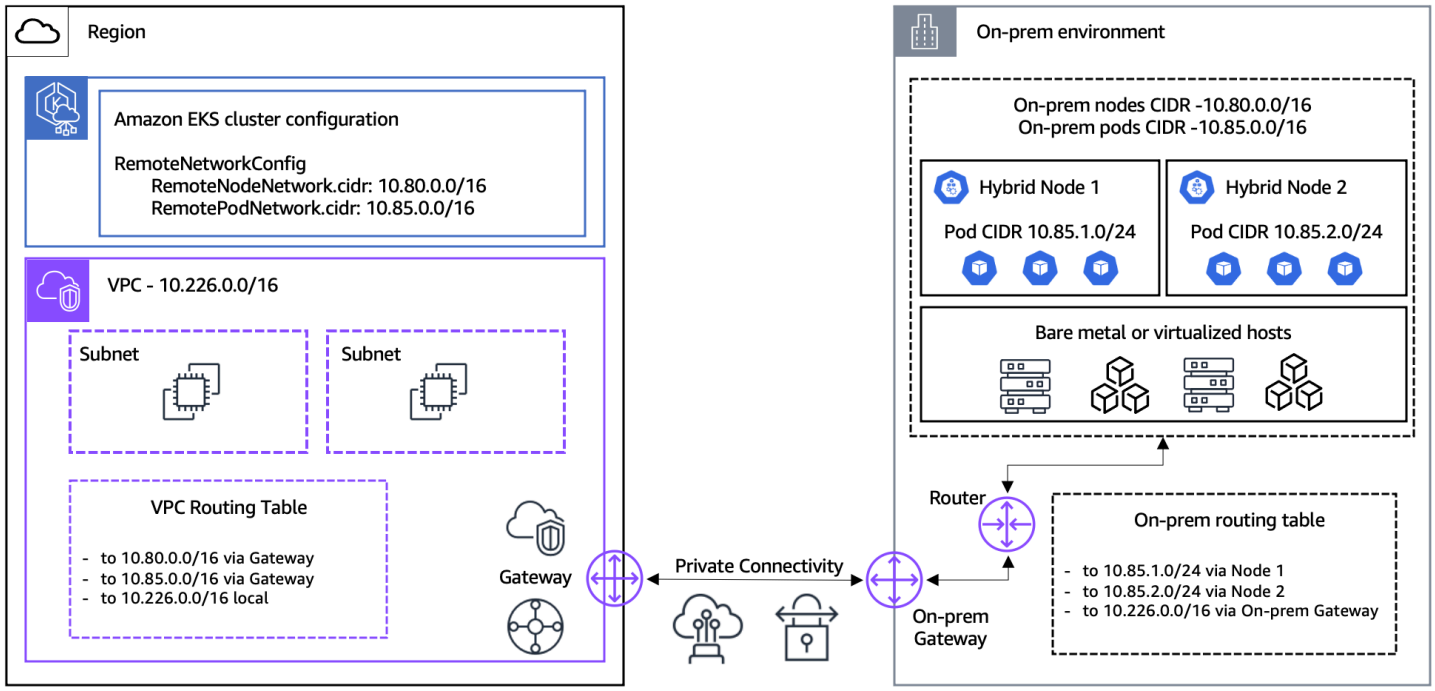
その他のリソース

- EKS Hybrid Nodes ワークショップ: EKS Hybrid Nodes をデモ環境にデプロイするためのステップバイステップ手順。
- AWS re:Invent: EKS Hybrid Nodes: EKS Hybrid Nodes の顧客環境でのローンチを紹介し、顧客が独自の環境で EKS Hybrid Nodes どのように使用しているかを示す AWS re:Invent セッション。
- AWS re:Post: EKS Hybrid Nodes のクラスターネットワーク: EKS Hybrid Nodes のネットワークを設定するためのさまざまな方法について説明する記事。
- AWS ブログ: EKS Hybrid Nodes を使用した複数環境間での生成 AI 推論の実行: EKS Hybrid Nodes を使用した複数環境間で生成 AI 推論を実行する方法について示すブログ投稿。

ハイブリッドノードの前提条件のセットアップ

Amazon EKS Hybrid Nodes を使用するには、オンプレミス環境と AWS 間のプライベート接続、サポートされているオペレーティングシステムを持つベアメタルサーバーまたは仮想マシン、AWS IAM Roles Anywhere または AWS Systems Manager (SSM) ハイブリッドアクティベーションを設定する必要があります。ハイブリッドノードのライフサイクル全体を通じてこれらの前提条件を管理するのはお客様の責任となります。

- オンプレミス環境と AWS 間のハイブリッドネットワーク接続
- 物理マシンまたは仮想マシンの形式のインフラストラクチャ
- ハイブリッドノードと互換性のあるオペレーティングシステム
- オンプレミスの IAM 認証情報プロバイダーが設定されました



ハイブリッドネットワーク接続

Amazon EKS コントロールプレーンとハイブリッドノード間の通信は、クラスターの作成時に渡される VPC とサブネットを介してルーティングされます。これは、コントロールプレーンからノードへのネットワーク形成のために Amazon EKS の既存のメカニズム上に構築されます。オンプレミス環境と VPC の接続には、AWS Site-to-Site VPN、AWS Direct Connect、独自の VPN 接続など、文書化されたオプションがいくつか利用可能です。ハイブリッドネットワーク接続にこれらのソリュー

ションを使用する方法の詳細については、「AWS Site-to-Site VPN ユーザーガイド」および「AWS Direct Connect ユーザーガイド」を参照してください。

最適なエクスペリエンスを実現するには、ハイブリッドノードから AWS リージョンへの接続には、100 Mbps 以上の通信速度と 200 ミリ秒以下のラウンドトリップレイテンシーを持つ、信頼性の高いネットワーク接続が推奨されます。これはほとんどのユースケースに対応する一般的なガイダンスですが、厳格な要件ではありません。帯域幅とレイテンシーの要件は、アプリケーションのイメージサイズ、アプリケーションの伸縮性、モニタリングとログ記録の設定、他の AWS サービスに保存されているデータへのアクセスに関するアプリケーションの依存関係など、ハイブリッドノードの数とワークロードの特性によって異なります。本番環境にデプロイする前に、独自のアプリケーションと環境でテストして、ネットワーク設定がワークロードの要件を満たしているか確認することをお勧めします。

オンプレミスのネットワーク設定

Amazon EKS コントロールプレーンからオンプレミス環境へのインバウンドネットワークアクセスを有効にして、Amazon EKS コントロールプレーンが、ハイブリッドノードで実行されている `kubelet` と通信できるようにし、オプションでハイブリッドノードで実行されているウェブフックと通信できるようにする必要があります。さらに、Amazon EKS コントロールプレーンと通信するには、ハイブリッドノードとハイブリッドノードで実行されているコンポーネントのアウトバウンドネットワークアクセスを有効にする必要があります。この通信は、AWS Direct Connect、AWS Site-to-Site VPN、または独自の VPN 接続に対して完全にプライベートなままになるように設定できます。

オンプレミスノードおよびポッドネットワークに使用する Classless Inter-Domain Routing (CIDR) 範囲では、IPv4 RFC1918 アドレスまたは CGNAT アドレスの範囲を使用する必要があります。オンプレミスルーターは、オンプレミスノードとオプションでポッドへのルートで設定する必要があります。ファイアウォールおよびオンプレミス環境で有効にする必要があるポートおよびプロトコルの完全なリストなど、オンプレミスのネットワーク要件の詳細については、「the section called “オンプレミスネットワーク設定”」を参照してください。

EKS クラスター設定

レイテンシーを最小限に抑えるには、オンプレミス環境またはエッジ環境に最も近い AWS リージョンで Amazon EKS クラスターを作成することが推奨されます。Amazon EKS クラスターの作成時に、2 つの API フィールド、`RemoteNodeNetwork` および `RemotePodNetwork` を介してオンプレミスのノードとポッドの CIDR を渡します。オンプレミスのノードとポッドの CIDR を特定するために、オンプレミスのネットワークチームに相談する必要がある場合があります。ノード CIDR はオ

ンプレミスネットワークから割り当てられ、ポッド CIDR は、CNI のオーバーレイネットワークを使用している場合に使用する Container Network Interface (CNI) から割り当てられます。Cilium および Calico はデフォルトでオーバーレイネットワークを使用します。

`RemoteNodeNetwork` および `RemotePodNetwork` フィールドを介して設定するオンプレミスノードおよびポッド CIDR は、VPC 経由でハイブリッドノードで実行されている `kubelet` およびポッドにトラフィックをルーティングするように Amazon EKS コントロールプレーンを設定するために使用されます。クラスター作成時に渡す VPC CIDR、または Amazon EKS クラスターのサービス IPv4 構成と互いに重複することはできません。また、オンプレミスのルーターがトラフィックをルーティングできるように、ポッド CIDR を各 EKS クラスターに対して一意にする必要があります。

Amazon EKS Kubernetes API サーバーエンドポイントには、パブリックエンドポイントまたはプライベートエンドポイントのいずれかのアクセスを使用することが推奨されます。「Public and Private (パブリックとプライベート)」を選択した場合、Amazon EKS Kubernetes API サーバーエンドポイントは、常に VPC の外部で稼働しているハイブリッドノードのパブリック IP に解決されるため、ハイブリッドノードがクラスターに参加できなくなる可能性があります。パブリックエンドポイントアクセスを使用すると、Kubernetes API サーバーエンドポイントはパブリック IP に解決され、ハイブリッドノードから Amazon EKS コントロールプレーンへの通信はインターネット経由でルーティングされます。プライベートエンドポイントアクセスを選択すると、Kubernetes API サーバーエンドポイントはプライベート IP に解決され、ハイブリッドノードから Amazon EKS コントロールプレーンへの通信は、プライベート接続リンク (ほとんどの場合 AWS Direct Connect または AWS Site-to-Site VPN) を介してルーティングされます。

VPC の構成

Amazon EKS クラスター作成時に渡す VPC のルーティングテーブルに、オンプレミスノード用のルートを設定し、オプションで仮想プライベートゲートウェイ (VGW) またはトランジットゲートウェイ (TGW) をターゲットとするポッドネットワークを設定する必要があります。次に例を示します。`REMOTE_NODE_CIDR` と `REMOTE_POD_CIDR` をオンプレミスのネットワークの値に置き換えます。

ルーティング先	ターゲット	説明
10.226.0.0/16	ローカル	VPC 内の VPC ルートへのローカルトラフィック

ルーティング先	ターゲット	説明
REMOTE_NODE_CIDR	tgw-abcdef123456	オンプレミスノード CIDR、トラフィックを TGW にルーティングする
REMOTE_POD_CIDR	tgw-abcdef123456	オンプレミスポッド CIDR、トラフィックを TGW にルーティングする

セキュリティグループの構成

クラスターを作成すると、Amazon EKS により `eks-cluster-sg-<cluster-name>-<uniqueID>` という名前のセキュリティグループが作成されます。このクラスターセキュリティグループのインバウンドルールは変更できませんが、アウトバウンドルールは制限できます。クラスターに追加のセキュリティグループを追加して、kubelet と、オプションでハイブリッドノードで実行されているウェブフックが Amazon EKS コントロールプレーンに接続できるようにする必要があります。この追加のセキュリティグループに必要なインバウンドルールを以下に示します。`REMOTE_NODE_CIDR` と `REMOTE_POD_CIDR` をオンプレミスのネットワークの値に置き換えます。

名前	セキュリティグループのルール ID	IP バージョン	タイプ	プロトコル	ポート範囲	ソース
オンプレミスノードのインバウンド	sgr-abcdef123456	IPv4	HTTPS	TCP	443	REMOTE_NODE_CIDR
オンプレミスポッドのインバウンド	sgr-abcdef654321	IPv4	HTTPS	TCP	443	REMOTE_POD_CIDR

インフラストラクチャ

ハイブリッドノードとして使用するには、ベアメタルサーバーまたは仮想マシンが必要です。ハイブリッドノードは基盤となるインフラストラクチャに依存せず、x86 および ARM アーキテクチャに対応しています。Amazon EKS Hybrid Nodes は「独自のインフラストラクチャを導入する」アプローチに従います。ここでは、ハイブリッドノードに使用するベアメタルサーバーまたは仮想マシンのプロビジョニングと管理はお客様の責任となります。厳格な最小リソースの要件はありませんが、ハイブリッドノードには少なくとも 1 vCPU および 1GiB RAM のホストを使用することが推奨されます。

オペレーティングシステム

Bottlerocket、Amazon Linux 2023 (AL2023)、Ubuntu、RHEL は、ハイブリッドノードのノードオペレーティングシステムとして使用されるために継続的に検証されます。Bottlerocket は、VMware vSphere 環境でのみ AWS でサポートされています。AL2023 は、Amazon EC2 の外部で実行される場合、AWS サポートプランの対象外です。AL2023 はオンプレミスの仮想化環境でのみ使用できます。詳細については、「Amazon Linux 2023 ユーザーガイド」を参照してください。AWS は Ubuntu および RHEL オペレーティングシステムとのハイブリッドノード統合をサポートしていますが、オペレーティングシステム自体はサポートしていません。

オペレーティングシステムのプロビジョニングと管理はお客様の責任となります。ハイブリッドノードを初めてテストする場合、プロビジョニング済みのホストで Amazon EKS Hybrid Nodes CLI (`nodeadm`) を実行するのが最も簡単です。本稼働のデプロイでは、ホスト起動時にホストを Amazon EKS クラスターに自動的に参加させる systemd サービスとして実行されるように設定された `nodeadm` を、オペレーティングシステムのゴールデンイメージに含めることが推奨されます。

オンプレミスの IAM 認証情報プロバイダー

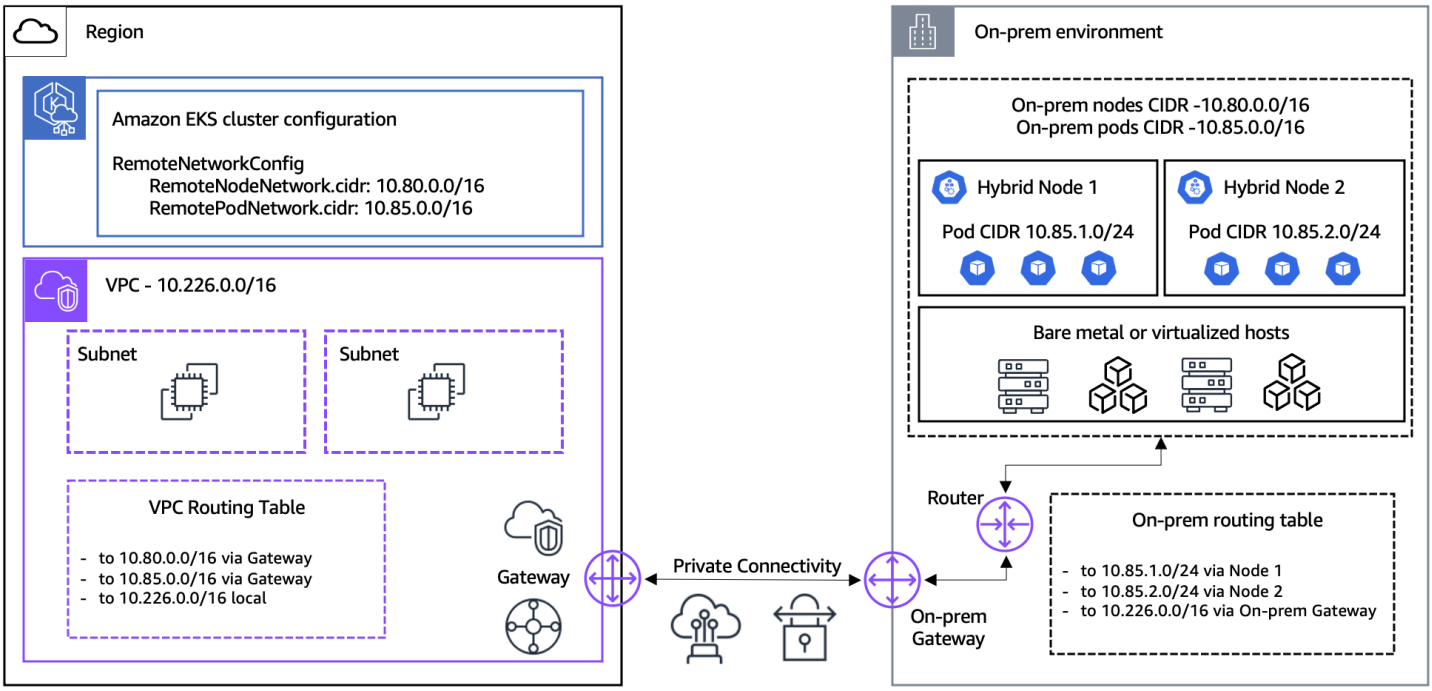
Amazon EKS Hybrid Nodes は、AWS SSM ハイブリッドアクティベーションまたは AWS IAM Roles Anywhere によってプロビジョニングされた一時的な IAM 認証情報を使用して、Amazon EKS クラスターで認証します。Amazon EKS Hybrid Nodes CLI (`nodeadm`) を用いて、AWS SSM ハイブリッドアクティベーションまたは AWS IAM Roles Anywhere を使用する必要があります。認証局 (CA) およびオンプレミス環境の証明書を持つ既存の公開鍵基盤 (PKI) がない場合、AWS SSM ハイブリッドアクティベーションを使用することをお勧めします。既存の PKI と証明書がオンプレミスにある場合は、AWS IAM Roles Anywhere を使用します。

Amazon EC2 で実行されているノードの the section called “ノード の IAM ロール” と同様に、ハイブリッドノードを Amazon EKS クラスターに参加させるために必要なアクセス許可を持つハイブリッ

ドノード IAM ロールを作成します。AWS IAM Roles Anywhere を使用している場合は、AWS IAM Roles Anywhere が Hybrid Nodes IAM Role を引き受けることを許可する信頼ポリシーを設定し、ハイブリッドノード IAM ロールを引き受け可能なロールとして AWS IAM Roles Anywhere プロファイルを設定します。AWS SSM を使用している場合は、AWS SSM がハイブリッドノード IAM ロールを引き受け、ハイブリッドノード IAM ロールを使用してハイブリッドアクティベーションを作成できるようにする信頼ポリシーを設定します。必要なアクセス許可を持つハイブリッドノード IAM ロールを作成する方法については、「the section called “認証情報を準備する”」を参照してください。

ハイブリッドノード用のネットワークを準備する

このトピックでは、Amazon EKS クラスターを作成してハイブリッドノードをアタッチする前に設定する必要があるネットワーク設定の概要を説明します。このガイドでは、AWS Site-to-Site VPN、AWS Direct Connect、または独自の VPN ソリューションを使用したハイブリッドネットワーク接続の前提条件の要件を満たしていることを前提としています。



オンプレミスネットワーク設定

最小ネットワーク要件

最適なエクスペリエンスを実現するには、ハイブリッドノードから AWS リージョンへの接続には、100 Mbps 以上の通信速度と 200 ミリ秒以下のラウンドトリップレイテンシーを持つ、信頼性の高いネットワーク接続が推奨されます。これはほとんどのユースケースに対応する一般的なガイダン

スですが、厳格な要件ではありません。帯域幅とレイテンシーの要件は、アプリケーションのイメージサイズ、アプリケーションの伸縮性、モニタリングとログ記録の設定、他の AWS サービスに保存されているデータへのアクセスに関するアプリケーションの依存関係など、ハイブリッドノードの数とワークロードの特性によって異なります。本番環境にデプロイする前に、独自のアプリケーションと環境でテストして、ネットワーク設定がワークロードの要件を満たしているか確認することをお勧めします。

オンプレミスノードとポッド CIDR

ハイブリッドノードに使用するノードとポッドの CIDR と、それらで実行されているワークロードを特定します。CNI にオーバーレイネットワークを使用している場合、ノード CIDR はオンプレミスネットワークから割り当てられ、ポッド CIDR は Container Network Interface (CNI) から割り当てられます。`RemoteNodeNetwork` および `RemotePodNetwork` フィールドで EKS クラスターを作成する際、オンプレミスノード CIDR およびポッド CIDR を入力として渡します。オンプレミスノード CIDR は、オンプレミスネットワークでルーティング可能である必要があります。オンプレミスポッド CIDR のルーティング可能性の詳細については、次のセクションを参照してください。

オンプレミスノードとポッド CIDR ブロックは、以下の要件を満たしている必要があります。

1. IPv4 RFC-1918 のいずれかの範囲内にある (`10.0.0.0/8`、`172.16.0.0/12`、または `192.168.0.0/16`) か、RFC 6598 で定義される CGNAT 範囲内にある (`100.64.0.0/10`) こと。
2. EKS クラスターの VPC CIDR や Kubernetes サービスの IPv4 CIDR と相互に重複しないこと。

> **Important**
>
> クラスターの作成中に Kubernetes サービス IPv4 CIDR を明示的に指定しない場合、Amazon EKS は設定されたリモートノードおよびポッドネットワークと競合しないサービス CIDR を自動的に選択します。つまり、割り当てられたサービス CIDR は、標準のデフォルト (`10.100.0.0/16` または `172.20.0.0/16`) と異なる場合があります。例えば、リモートネットワークが `172.20.0.0/16` と重複している場合、Amazon EKS は代わりに `172.16.0.0/16` を割り当てます。予期しないサービス CIDR の割り当てを回避するには、クラスターの作成時にサービス IPv4 CIDR を明示的に指定することをお勧めします。クラスターに割り当てられたサービス CIDR を検索する必要がある場合は、`aws eks describe-cluster --name CLUSTER_NAME --query "cluster.kubernetesNetworkConfig.serviceIpv4Cidr"` を実行します。

オンプレミスポッドのネットワークルーティング

EKS Hybrid Node を使用する際、クラウド環境とオンプレミス環境間で完全なクラスター通信および機能を有効にするため、オンプレミスネットワークでオンプレミスポッド CIDR をルーティング可能にすることが一般的に推奨されています。

ルーティング可能なポッドネットワーク

オンプレミスネットワークでポッドネットワークをルーティング可能にできる場合、以下のガイダンスに従ってください。

1. オンプレミスポッド CIDR で EKS クラスターの `RemotePodNetwork` フィールドを設定し、オンプレミスポッド CIDR で VPC ルートテーブルを設定し、オンプレミスポッド CIDR で EKS クラスターセキュリティグループを設定します。
2. オンプレミスネットワークでオンプレミスポッド CIDR をルーティング可能にするために使用できる手法がいくつかあります。具体的には、ボーダーゲートウェイプロトコル (BGP)、静的ルート、その他のカスタムルーティングソリューションなどです。BGP はカスタムまたは手動でルート設定する必要がある代替ソリューションよりもスケーラブルで管理しやすいため、推奨されるソリューションです。AWS は、ポッド CIDR をアドバタイズするための Cilium と Calico の BGP 機能をサポートしています。詳細については、「the section called “CNI を設定する”」および「the section called “ルーティング可能なリモートポッド CIDR”」を参照してください。
3. EKS コントロールプレーンはウェブフックに割り当てられたポッド IP アドレスと通信できるため、ウェブフックはハイブリッドノードで実行できます。
4. クラウドノードで実行されているワークロードは、同じ EKS クラスター内のハイブリッドノードで実行されているワークロードと直接通信することができます。
5. 他の AWS サービス (AWS Application Load Balancer や Amazon Managed Service for Prometheus など) はハイブリッドノードで実行されているワークロードと通信し、ネットワークトラフィックのバランスを取ってポッドメトリクスをスクレイピングすることができます。

ルーティング不可能なポッドネットワーク

オンプレミスネットワークでポッドネットワークをルーティングできない場合、以下のガイダンスに従ってください。

1. ウェブフックは EKS コントロールプレーンからウェブフックに割り当てられたポッド IP アドレスへの接続が必要なため、ウェブフックはハイブリッドノードで実行することはできません。この場合、ハイブリッドノードと同じ EKS クラスター内のクラウドノードでウェブフックを実行す

ることが推奨されます。詳細については、「the section called “ウェブフックの設定”」を参照してください。

2. クラウドノードで実行されているワークロードは、クラウドノードに VPC CNI を使用する際、またはハイブリッドノードに Cilium または Calico を使用する際、ハイブリッドノードで実行されているワークロードと直接通信することはできません。
3. サービストラフィック分散を使用し、トラフィックを発信元のゾーン内に留めます。サービストラフィック分散の詳細については、「the section called “サービストラフィック分散の設定”」を参照してください。
4. ポッドトラフィックがオンプレミスホストから出る際、出力マスカレードまたはネットワークアドレス変換 (NAT) を使用するように CNI を設定します。Cilium ではデフォルトで有効になっています。Calico では `natOutgoing` を `true` に設定する必要があります。
5. AWS Application Load Balancer や Amazon Managed Service for Prometheus などの他の AWS サービスは、ハイブリッドノードで実行されているワークロードと通信できません。

ハイブリッドノードのインストールとアップグレードに必要なアクセス

ホストにハイブリッドノードの依存関係をインストールするインストールプロセス中は、以下のドメインにアクセスできる必要があります。このプロセスは、オペレーティングシステムイメージを構築するときに 1 回実行することも、ランタイム時に各ホストで実行することもできます。これには、初期インストールと、ハイブリッドノードの Kubernetes バージョンをアップグレードするときが含まれます。

パッケージの中には、OS のデフォルトパッケージマネージャーを使用してインストールされるものがあります。AL2023 および RHEL の場合、`yum` コマンドを使用して `containerd`、`ca-certificates`、`iptables`、`amazon-ssm-agent` をインストールします。Ubuntu の場合、`apt` を使用して `containerd`、`ca-certificates`、`iptables` をインストールし、`snap` を使用して `amazon-ssm-agent` をインストールします。

コンポーネント	[URL]	プロトコル	ポート
EKS ノードアーティファクト (S3)	https://hybrid-assets.eks.amazonaws.com	HTTPS	443

コンポーネント	[URL]	プロトコル	ポート
VPC サービスエンドポイント	https://eks.*region*.amazonaws.com	HTTPS	443
ECR サービスエンドポイント	https://api.ecr.*region*.amazonaws.com	HTTPS	443
EKS ECR エンドポイント	リージョンのエンドポイントについては、「the section called “Amazon イメージレジストリを表示する”」を参照してください。	HTTPS	443
SSM バイナリエンドポイント [1]	https://amazon-ssm-*region*.s3.*region*.amazonaws.com	HTTPS	443
SSM サービスエンドポイント [1]	https://ssm.*region*.amazonaws.com	HTTPS	443
IAM Anywhere バイナリエンドポイント [2]	https://rolesanywhere.amazonaws.com	HTTPS	443
IAM Anywhere サービスエンドポイント [2]	https://rolesanywhere.*region*.amazonaws.com	HTTPS	443

コンポーネント	[URL]	プロトコル	ポート
オペレーティングシステムパッケージマネージャーのエンドポイント	パッケージリポジトリのエンドポイントは OS に固有であり、地理的リージョンによって異なる場合があります。	HTTPS	443

> **Note**
>
> [1] AWS SSM エンドポイントへのアクセスは、オンプレミスの IAM 認証情報プロバイダーに AWS SSM ハイブリッドアクティベーションを使用している場合のみ必要です。
> [2] AWS IAM エンドポイントへのアクセスは、オンプレミスの IAM 認証情報プロバイダーに AWS IAM Roles Anywhere を使用している場合のみ必要です。

継続的なクラスター運用に必要なアクセス

クラスターの継続的な運用には、オンプレミスのファイアウォールに以下のネットワークアクセスが必要です。

> **Important**
>
> CNI の選択に応じて、CNI ポートに追加のネットワークアクセスルールを設定する必要があります。詳細については、「Cilium のドキュメント」および「Calico のドキュメント」を参照してください。

タイプ	プロトコル	方向	ポート	送信元	目的地	使用方法
HTTPS	TCP	アウトバウンド	443	リモートノード CIDR	EKS クラスター IP [1]	Kubelet から Kubernetes API サーバーへ

タイプ	プロトコル	方向	ポート	送信元	目的地	使用方法
HTTPS	TCP	アウトバウンド	443	リモートポッド CIDR	EKS クラスター IP [1]	Pod から Kubernetes API サーバーへ
HTTPS	TCP	アウトバウンド	443	リモートノード CIDR	SSM サービスエンドポイント	SSM ハイブリッドアクティベーション認証情報の更新と 5 分ごとの SSM ハートビート
HTTPS	TCP	アウトバウンド	443	リモートノード CIDR	IAM Anywhere サービスエンドポイント	IAM Roles Anywhere 認証情報の更新
HTTPS	TCP	アウトバウンド	443	リモートポッド CIDR	STS リージョナルエンドポイント	Pod から STS エンドポイントへ、IRSA のみ必要

タイプ	プロトコル	方向	ポート	送信元	目的地	使用方法
HTTPS	TCP	アウトバウンド	443	リモートノード CIDR	Amazon EKS Auth サービスエンドポイント	Amazon EKS Pod Identity のみに必要な Amazon EKS 認証エンドポイントへのノード
HTTPS	TCP	インバウンド	10250	EKS クラスター IP [1]	リモートノード CIDR	Kubernetes API サーバーから kubelet へ
HTTPS	TCP	インバウンド	ウェブフックポート	EKS クラスター IP [1]	リモートポッド CIDR	Kubernetes API サーバーからウェブフックへ

タイプ	プロトコル	方向	ポート	送信元	目的地	使用方法
HTTPS	TCP、UDP	インバウンド、アウトバウンド	53	リモートポッド CIDR	リモートポッド CIDR	Pod から CoreDNS へ。クラウドで CoreDNS のレプリカを少なくとも 1 つ実行する場合は、CoreDNS が実行されている VPC への DNS トラフィックを許可する必要があります。
ユーザー定義	ユーザー定義	インバウンド、アウトバウンド	アプリポート	リモートポッド CIDR	リモートポッド CIDR	Pod から Pod へ

Note

[1] EKS クラスターの IP。Amazon EKS Elastic Network Interface については、次のセクションを参照してください。

Amazon EKS ネットワークインターフェイス

Amazon EKS は、クラスターの作成時に渡す VPC 内のサブネットにネットワークインターフェイスをアタッチして、EKS コントロールプレーンと VPC 間の通信を有効にします。Amazon EKS が作成するネットワークインターフェイスは、クラスターの作成後に、Amazon EC2 コンソールまたは

AWS CLI で確認できます。Kubernetes バージョンのアップグレードなど、EKS クラスターに変更が適用されると、元のネットワークインターフェイスが削除され、新しいネットワークインターフェイスが作成されます。クラスターの作成時に渡すサブネットに制約付きサブネットサイズを使用することで、Amazon EKS ネットワークインターフェイスの IP 範囲を制限できます。これにより、この既知の制約付き IP セットへのインバウンド/アウトバウンド接続を許可するようにオンプレミスファイアウォールを設定することが容易になります。ネットワークインターフェイスが作成されるサブネットを制御するためには、クラスター作成時に指定するサブネットの数を制限するか、クラスター作成後にサブネットを更新することができます。

Amazon EKS によってプロビジョニングされるネットワークインターフェイスには、`Amazon EKS your-cluster-name` の形式の説明が付けられています。Amazon EKS がプロビジョニングするネットワークインターフェイスの IP アドレスを見つけるために使用できる AWS CLI コマンドについては、以下の例を参照してください。`VPC_ID` を、クラスターの作成時に渡す VPC の ID に置き換えます。

```
aws ec2 describe-network-interfaces \
--query 'NetworkInterfaces[?(VpcId == VPC_ID && contains(Description,Amazon
 EKS))].PrivateIpAddress'
```

AWS VPC とサブネットの設定

ハイブリッドノードを持つクラスターには、Amazon EKS の既存の VPC とサブネットの要件が適用されます。さらに、VPC CIDR はオンプレミスノードおよびポッド CIDR と重複することはできません。オンプレミスノードの VPC ルートテーブルにルートを設定し、オプションでポッド CIDR を設定する必要があります。これらのルートは、ハイブリッドネットワーク接続に使用しているゲートウェイにトラフィックをルーティングするように設定する必要があります。通常、これは仮想プライベートゲートウェイ (VGW) またはトランジットゲートウェイ (TGW) です。TGW または VGW を使用して VPC をオンプレミス環境に接続する場合は、VPC の TGW または VGW アタッチメントを作成する必要があります。VPC は、DNS ホスト名と DNS 解決がサポートされている必要があります。

AWS CLI を使用する手順は以下のとおりです。これらのリソースは AWS マネジメントコンソールで、または AWS CloudFormation、AWS CDK、Terraform などの他のインターフェイスでも作成することができます。

ステップ 1: VPC を作成する

1. 次のコマンドを実行して VPC を作成します。VPC_CIDR を、RFC 1918 (プライベート)、CGNAT (RFC 6598)、または非 RFC 1918/非 CGNAT (パブリック) のいずれかの IPv4 CIDR

範囲に置き換えます (例えば、10.0.0.0/16 など)。注: EKS の要件である DNS 解決は、VPC に対しデフォルトで有効になっています。

```
aws ec2 create-vpc --cidr-block VPC_CIDR
```

2. VPC の DNS ホスト名を有効にします。DNS 解決は、VPC に対しデフォルトで有効になっています。VPC_ID を、前のステップで作成した VPC の ID に置き換えます。

```
aws ec2 modify-vpc-attribute --vpc-id VPC_ID --enable-dns-hostnames
```

ステップ 2: サブネットを作成する

少なくとも 2 つのサブネットを作成します。Amazon EKS は、これらのサブネットをクラスターのネットワークインターフェイスに使用します。詳細については、「サブネットの要件と考慮事項」を参照してください。

1. AWS リージョンのアベイラビリティーゾーンは、以下のコマンドで確認できます。us-west-2 を実際のリージョンに置き換えます。

```
aws ec2 describe-availability-zones \
     --query 'AvailabilityZones[?(RegionName == us-west-2)].ZoneName'
```

2. サブネットを作成します。VPC_ID を VPC の ID に置き換えます。SUBNET_CIDR を、サブネットの CIDR ブロック (10.0.1.0/24 など) に置き換えます。AZ を、サブネットが作成されるアベイラビリティーゾーン (us-west-2a など) に置き換えます。作成するサブネットは、少なくとも 2 つの異なるアベイラビリティーゾーンに存在している必要があります。

```
aws ec2 create-subnet \
    --vpc-id VPC_ID \
    --cidr-block SUBNET_CIDR \
    --availability-zone AZ
```

(オプション) ステップ 3: Amazon VPC Transit Gateway (TGW) または AWS Direct Connect 仮想プライベートゲートウェイ (VGW) を使用して VPC をアタッチする

TGW または VGW を使用している場合は、VPC を TGW または VGW にアタッチします。詳細については、「Amazon VPC attachments in Amazon VPC Transit Gateways」または「AWS Direct Connect virtual private gateway associations」を参照してください。

Transit Gateway

以下のコマンドを実行して、トランジットゲートウェイをアタッチします。VPC_ID を VPC の ID に置き換えます。SUBNET_ID1 と SUBNET_ID2 を、前のステップで作成したサブネットの ID に置き換えます。TGW_ID を TGW の ID に置き換えます。

```
aws ec2 create-transit-gateway-vpc-attachment \
    --vpc-id VPC_ID \
    --subnet-ids SUBNET_ID1 SUBNET_ID2 \
    --transit-gateway-id TGW_ID
```

仮想プライベートゲートウェイ

以下のコマンドを実行して、トランジットゲートウェイをアタッチします。VPN_ID をVGW の ID に置き換えます。VPC_ID を VPC の ID に置き換えます。

```
aws ec2 attach-vpn-gateway \
    --vpn-gateway-id VPN_ID \
    --vpc-id VPC_ID
```

(オプション) ステップ 4: ルートテーブルを作成する

VPC のメインルートテーブルを変更するか、カスタムルートテーブルを作成できます。以下の手順では、オンプレミスノードとポッド CIDR へのルートを含むカスタムルートテーブルを作成します。詳細については、「サブネットルートテーブル」を参照してください。VPC_ID を VPC の ID に置き換えます。

```
aws ec2 create-route-table --vpc-id VPC_ID
```

ステップ 5: オンプレミスノードとポッドのルートを作成する

オンプレミスの各リモートノードのルートテーブルにルートを作成します。VPC のメインルートテーブルを変更するか、前のステップで作成したカスタムルートテーブルを使用できます。

以下の例は、オンプレミスノードとポッド CIDR のルートを作成する方法を示します。この例では、トランジットゲートウェイ (TGW) を使用して VPC をオンプレミス環境に接続します。複数のオンプレミスノードとポッド CIDR がある場合は、CIDR ごとにステップを繰り返します。

- インターネットゲートウェイまたは仮想プライベートゲートウェイ (VGW) を使用している場合は、`--transit-gateway-id` を `--gateway-id` に置き換えます。
- `RT_ID` を、前のステップで作成したルートテーブルの ID に置き換えます。
- `REMOTE_NODE_CIDR` を、ハイブリッドノードに使用する CIDR 範囲に置き換えます。
- `REMOTE_POD_CIDR` を、ハイブリッドノードで稼働しているポッドに使用する CIDR 範囲に置き換えます。ポッド CIDR 範囲は、オンプレミスのオーバーレイネットワークを最も一般的に使用するコンテナネットワークインターフェイス (CNI) 設定に対応しています。詳細については、「the section called “CNI を設定する”」を参照してください。
- `TGW_ID` を TGW の ID に置き換えます。

リモートノードネットワーク

```
aws ec2 create-route \
    --route-table-id RT_ID \
    --destination-cidr-block REMOTE_NODE_CIDR \
    --transit-gateway-id TGW_ID
```

リモートポッドネットワーク

```
aws ec2 create-route \
    --route-table-id RT_ID \
    --destination-cidr-block REMOTE_POD_CIDR \
    --transit-gateway-id TGW_ID
```

(オプション) ステップ 6: サブネットをルートテーブルに関連付ける

前のステップでカスタムルートテーブルを作成した場合は、前のステップで作成した各サブネットをカスタムルートテーブルに関連付けます。VPC メインルートテーブルを変更する場合、サブネッ

トは VPC のメインルートテーブルに自動的に関連付けられるため、このステップをスキップできます。

前のステップで作成したサブネットごとに、以下のコマンドを実行します。RT_ID を、前のステップで作成したルートテーブルに置き換えます。SUBNET_ID をサブネットの ID に置き換えます。

```
aws ec2 associate-route-table --route-table-id RT_ID --subnet-id SUBNET_ID
```

クラスターセキュリティグループの設定

クラスターの継続的な運用には、EKS クラスターセキュリティグループの以下のアクセスが必要です。Amazon EKS は、リモートノードとポッドネットワークが設定されたクラスターを作成または更新するときに、ハイブリッドノードに必要なインバウンドセキュリティグループルールを自動的に作成します。セキュリティグループがデフォルトですべてのアウトバウンドトラフィックを許可するため、Amazon EKS はハイブリッドノードのクラスターセキュリティグループのアウトバウンドルールを自動的には変更しません。クラスターセキュリティグループをカスタマイズする場合は、以下の表に示したルールにトラフィックを制限できます。

タイプ	プロトコル	方向	ポート	送信元	目的地	使用方法
HTTPS	TCP	インバウンド	443	リモートノード CIDR	該当なし	Kubelet から Kubernetes API サーバーへ
HTTPS	TCP	インバウンド	443	リモートポッド CIDR	該当なし	CNI がポッドトラフィックに NAT を使用していない場合で、ポッドが K8s API サーバーへのアクセス

タイプ	プロトコル	方向	ポート	送信元	目的地	使用方法
						を必要とする場合。
HTTPS	TCP	アウトバウンド	10250	該当なし	リモートノード CIDR	Kubernetes API サーバーから Kubelet へ
HTTPS	TCP	アウトバウンド	ウェブフックポート	該当なし	リモートポッド CIDR	Kubernetes API サーバーからウェブフックへ (ハイブリッドノードでウェブフックを実行している場合)

Important

セキュリティグループルールの制限: Amazon EC2 セキュリティグループに保持できるインバウンドルールはデフォルトで最大 60 個です。クラスターセキュリティグループがこの制限に近づいた場合、セキュリティグループのインバウンドルールが適用されないことがあります。この場合、欠落しているインバウンドルールを手動で追加しなければならないことがあります。
CIDR クリーンアップの責任: EKS クラスターからリモートノードまたはポッドネットワークを削除した場合、EKS は対応するセキュリティグループルールを自動的に削除しません。セキュリティグループルールから未使用のリモートノードまたはポッドネットワークを手動で削除する責任はお客様にあります。

Amazon EKS が作成するクラスターセキュリティグループの詳細については「the section called “セキュリティグループの要件”」を参照してください。

(オプション) セキュリティグループの手動設定

追加のセキュリティグループを作成したり、自動的に作成されたルールを変更したりする必要がある場合は、次のコマンドを参照として使用できます。デフォルトでは、以下のコマンドは、すべてのアウトバウンドアクセスを許可するセキュリティグループを作成します。アウトバウンドアクセスを上記のルールのみに制限することができます。アウトバウンドルールの制限を検討している場合、変更したルールを本番稼働用のクラスターに適用する前に、すべてのアプリケーションとポッドの接続を徹底的にテストすることをお勧めします。

- 最初のコマンドで、SG_NAME をセキュリティグループの名前に置き換えます
- 最初のコマンドで、VPC_ID を前のステップで作成した VPC の ID に置き換えます
- 2 番目のコマンドで、SG_ID を最初のコマンドで作成したセキュリティグループの ID に置き換えます
- 2 番目のコマンドで、REMOTE_NODE_CIDR と REMOTE_POD_CIDR をハイブリッドノードとオンプレミスネットワークの値にそれぞれ置き換えます。

```
aws ec2 create-security-group \
    --group-name SG_NAME \
    --description "security group for hybrid nodes" \
    --vpc-id VPC_ID
```

```
aws ec2 authorize-security-group-ingress \
    --group-id SG_ID \
    --ip-permissions '[{"IpProtocol": "tcp", "FromPort": 443, "ToPort": 443,
 "IpRanges": [{"CidrIp": "REMOTE_NODE_CIDR"}, {"CidrIp": "REMOTE_POD_CIDR"}]}]'
```

ハイブリッドノード用のオペレーティングシステムを準備する

Bottlerocket、Amazon Linux 2023 (AL2023)、Ubuntu、RHEL は、ハイブリッドノードのノードオペレーティングシステムとして使用されるために継続的に検証されます。Bottlerocket は、VMware vSphere 環境でのみ AWS でサポートされています。AL2023 は、Amazon EC2 の外部で実行される場合、AWS サポートプランの対象外です。AL2023 はオンプレミスの仮想化環境でのみ使用できます。詳細については、「Amazon Linux 2023 ユーザーガイド」を参照してください。AWS は Ubuntu および RHEL オペレーティングシステムとのハイブリッドノード統合をサポートしていますが、オペレーティングシステム自体はサポートしていません。

オペレーティングシステムのプロビジョニングと管理はお客様の責任となります。ハイブリッドノードを初めてテストする場合、プロビジョニング済みのホストで Amazon EKS Hybrid Nodes CLI (nodeadm) を実行するのが最も簡単です。本稼働デプロイでは、ホスト起動時にホストを Amazon EKS クラスターに自動的に結合する systemd サービスとして実行するように設定された nodeadm を、オペレーティングシステムイメージに含めることが推奨されます。Bottlerocket を vSphere のノードオペレーティングシステムとして使用している場合は nodeadm を使用する必要はありません。Bottlerocket にはハイブリッドノードに必要な依存関係が既に含まれ、ホストのスタートアップ時に設定するクラスターに自動的に接続されるからです。

⚠ SSM 認証情報プロバイダーに必要な nodeadm バージョン

ハイブリッドノードの認証情報プロバイダーとして AWS Systems Manager (SSM) を使用する場合は、新しいインストールとアップグレードに nodeadm バージョン `1.0.19` 以降を使用する必要があります。以前のバージョンの nodeadm には古い SSM 署名キーが含まれており、`nodeadm install` と `nodeadm upgrade` の実行中に失敗し、次の署名検証エラーが発生します:

```
"msg":"Command failed","error":"failed to install ssm installer: validating ssm-setup-cli signature: Signature Verification Error: No matching signature"
```

このエラーを解決するには、`nodeadm install` または `nodeadm upgrade` を実行する前に nodeadm の最新バージョンをダウンロードします。

バージョン互換性

以下の表は、ハイブリッドノードのノードオペレーティングシステムとしての使用に互換性があり、検証済みとなっているオペレーティングシステムのバージョンを示しています。オペレーティングシステムのこの表に記載されていないバリアントやバージョンを使用している場合、そのオペレーティングシステムのバリアントまたはバージョンとのハイブリッドノードの互換性は、AWS サポートの対象外となります。ハイブリッドノードは基盤となるインフラストラクチャに依存せず、x86 および ARM アーキテクチャに対応しています。

オペレーティングシステム	バージョン
Amazon Linux	Amazon Linux 2023 (AL2023)

オペレーティングシステム	バージョン
Bottlerocket	v1.37.0 以降、Kubernetes v1.28 以降を実行している VMware のバリアント
Ubuntu	Ubuntu 20.04、Ubuntu 22.04、Ubuntu 24.04
Red Hat Enterprise Linux	RHEL 8、RHEL 9

オペレーティングシステムに関する考慮事項

General

- Amazon EKS Hybrid Nodes CLI (nodeadm) を使用すると、ハイブリッドノードのコンポーネントおよび依存関係のインストールと設定を簡素化できます。`nodeadm install` プロセスは、オペレーティングシステムイメージのビルドパイプライン中、または各オンプレミスホストの実行時に実行できます。nodeadm がインストールするコンポーネントの詳細については、「the section called “ハイブリッドノード用 nodeadm”」を参照してください。
- オンプレミス環境でプロキシを使用してインターネットにアクセスしている場合、インストールおよびアップグレードのプロセスでパッケージマネージャーがプロキシを使用するよう設定するために、追加のオペレーティングシステム設定が必要です。手順については「the section called “プロキシを設定する”」を参照してください。

Bottlerocket

- Bottlerocket ノードを接続するときのステップとツールは他のオペレーティングシステムのステップとは異なるため、「the section called “ハイブリッドノードを接続する”」の手順ではなく「the section called “Bottlerocket を実行しているハイブリッドノードの接続”」に記載しています。
- Bottlerocket のステップでは、ハイブリッドノード CLI のツールである `nodeadm` は使用しません。
- EKS Hybrid Nodes でサポートされているのは、Bottlerocket のバージョンが v1.37.0 以降である VMware のバリアントのみです。Bottlerocket の VMware のバリアントは、Kubernetes のバージョン v1.28 以降で使用できます。その他の Bottlerocket のバリアントは、ハイブリッドノードのオペレーティングシステムとしてサポートされていません。注: Bottlerocket の VMware のバリアントを使用できるのは、x86_64 のアーキテクチャのみです。

Containerd

- Containerd は標準の Kubernetes コンテナランタイムであり、ハイブリッドノードのみならず、あらゆるコンピューティングタイプの Amazon EKS ノードの依存関係の 1 つです。Amazon EKS Hybrid Nodes CLI (nodeadm) は、`nodeadm install` プロセス中に containerd のインストールを試みます。`--containerd-source` コマンドラインオプションを使用して、`nodeadm install` の実行時に containerd のインストールを設定できます。有効なオプションは、`none`、`distro`、`docker` です。RHEL を使用している場合、`distro` は有効なオプションではなく、Docker のリポジトリから containerd ビルドをインストールするよう `nodeadm` を設定するか、containerd を手動でインストールできます。AL2023 または Ubuntu を使用する場合、`nodeadm` はオペレーティングシステムのディストリビューションから containerd をインストールするのがデフォルトになります。nodeadm で containerd をインストールしない場合は、`--containerd-source none` オプションを使用します。

Ubuntu

- Ubuntu 24.04 を使用している場合、ポッドを適切に終了させるための修正を適用するために、containerd のバージョンを更新するか、AppArmor の設定を変更する必要がある場合があります。「Ubuntu #2065423」を参照してください。AppArmor プロファイルに変更を適用するには、再起動が必要です。Ubuntu 24.04 の最新バージョンには、修正が含まれた更新版の containerd バージョン (containerd バージョン 1.7.19 以降) がパッケージマネージャーに含まれています。

ARM

- ARM ハードウェアを使用している場合、EKS kube-proxy アドオンのバージョン 1.31 以降を実行するためには、Cryptography Extension (ARMv8.2+crypto) を搭載した ARMv8.2 準拠のプロセッサが必要です。Cortex-A72 ベースのプロセッサのほか、Raspberry Pi 5 より前のすべての Raspberry Pi システムもこの要件を満たしていません。回避策として、2026 年 7 月に延長サポートが終了するまで、EKS kube-proxy アドオンのバージョン 1.30 をこれまで通り使用できます。「Kubernetes release calendar」を参照するか、アップストリームからカスタム kube-proxy イメージを使用してください。
- kube-proxy ログに以下のエラーメッセージが記録されていれば、この非互換性が生じていることになります。

```
Fatal glibc error: This version of Amazon Linux requires a newer ARM64 processor
 compliant with at least ARM architecture 8.2-a with Cryptographic extensions. On EC2
 this is Graviton 2 or later.
```

オペレーティングシステムイメージの構築

Amazon EKS は、nodeadm を含み、ホスト起動時に実行するように設定されたオペレーティングシステムイメージを作成するために使用できる Packer テンプレートの例を提供しています。このプロセスは、各ホストでハイブリッドノードの依存関係を個別にプルすることを避け、ハイブリッドノードのブートストラッププロセスを自動化するために推奨されています。Packer テンプレートの例は、Ubuntu 22.04、Ubuntu 24.04、RHEL 8、または RHEL 9 ISO イメージで使用でき、イメージの出力は OVA、Qcow2、または raw の形式で行えます。

前提条件

Packer テンプレートの例を使用する前に、Packer を実行しているマシンに以下がインストールされている必要があります。

- Packer バージョン 1.11.0 以降。Packer のインストール手順については、「Packer documentation」の「nstall Packer」を参照してください。
- OVA を構築する場合、VMware vSphere プラグイン 1.4.0 以降
- Qcow2 または raw イメージを構築する場合、QEMU プラグインバージョン 1.x

環境変数の設定

Packer ビルドを実行する前に、Packer を実行しているマシンで次の環境変数を設定します。

全般

いずれのオペレーティングシステムおよび出力形式においても、イメージを構築する際には、以下の環境変数を設定する必要があります。

環境変数	タイプ	説明
PKR_SSH_PASSWORD	String	Packer は、プロビジョニング時に作成されたマシンに SSH 接続する際は、ssh_username および ssh_password 変数を使用します。こ

環境変数	タイプ	説明
		れは、各 OS のキックスタートファイルまたは user-data ファイル内に初期ユーザーを作成する際に使用されるパスワードと一致する必要があります。デフォルトは、OS に応じて「builder」または「ubuntu」に設定されます。パスワードを設定する際は、必ず対応する `ks.cfg` または `user-data` ファイル内のパスワードも一致するように変更してください。
ISO_URL	String	使用する ISO の URL。サーバーからダウンロードするウェブリンクでも、ローカルファイルへの絶対パスでもかまいません
ISO_CHECKSUM	String	指定された ISO に関連するチェックサム。
CREDENTIAL_PROVIDER	String	ハイブリッドノードの認証情報プロバイダー。有効な値は `ssm` (SSM ハイブリッドアクティベーション用、デフォルト) と `iam` (IAM Roles Anywhere 用) です

環境変数	タイプ	説明
K8S_VERSION	String	ハイブリッドノードの Kubernetes バージョン (`1.31` など)。サポートされている Kubernetes のバージョンについては、「Amazon EKS supported versions」を参照してください。
NODEADM_ARCH	String	`nodeadm install` 用アーキテクチャ。`amd` または `arm` を選択します。

RHEL

RHEL を使用している場合は、以下の環境変数を設定する必要があります。

環境変数	タイプ	説明
RH_USERNAME	String	RHEL サブスクリプションマネージャーのユーザー名
RH_PASSWORD	String	RHEL サブスクリプションマネージャーのパスワード
RHEL_VERSION	String	使用する RHEL の ISO のバージョン。有効な値は 8 または 9 です。

Ubuntu

Ubuntu 固有の環境変数は必要ありません。

vSphere

VMware vSphere OVA を構築する場合は、以下の環境変数を設定する必要があります。

環境変数	タイプ	説明
VSPHERE_SERVER	String	vSphere のサーバーアドレス
VSPHERE_USER	String	vSphere のユーザー名
VSPHERE_PASSWORD	String	vSphere のパスワード
VSPHERE_DATACENTER	String	vSphere のデータセンター名
VSPHERE_CLUSTER	String	vSphere のクラスター名
VSPHERE_DATASTORE	String	vSphere のデータストア名
VSPHERE_NETWORK	String	vSphere のネットワーク名
VSPHERE_OUTPUT_FOLDER	String	テンプレート用の vSphere の出力フォルダ

QEMU

環境変数	タイプ	説明
PACKER_OUTPUT_FORMAT	String	QEMU ビルダーの出力形式。有効な値は、`qcow2` および `raw` です。

テンプレートを検証する

環境変数を設定した後は、ビルドを実行する前に以下のコマンドでテンプレートを検証します。テンプレートに異なる名前を使用している場合は、`template.pkr.hcl` を置き換えてください。

```
packer validate template.pkr.hcl
```

イメージを構築する

以下のコマンドを使用してイメージを構築し、`-only` フラグを使用してイメージのターゲットとオペレーティングシステムを指定します。テンプレートに異なる名前を使用している場合は、`template.pkr.hcl` を置き換えてください。

vSphere OVA

> **Note**
>
> vSphere で RHEL を使用している場合は、キックスタートファイルを OEMDRV イメージに変換し、起動元の ISO として渡す必要があります。詳細については、EKS ハイブリッドノードの GitHub リポジトリにある「Packer Readme」を参照してください。

Ubuntu 22.04 OVA

```
packer build -only=general-build.vsphere-iso.ubuntu22 template.pkr.hcl
```

Ubuntu 24.04 OVA

```
packer build -only=general-build.vsphere-iso.ubuntu24 template.pkr.hcl
```

RHEL 8 OVA

```
packer build -only=general-build.vsphere-iso.rhel8 template.pkr.hcl
```

RHEL 9 OVA

```
packer build -only=general-build.vsphere-iso.rhel9 template.pkr.hcl
```

QEMU

> **Note**
>
> ビルダーのホストと一致しない特定のホストの CPU 用にイメージを構築する場合は、ホストの CPU に対応する名前を「QEMU」のドキュメントで確認し、以下のコマンドを実行する際にホストの CPU 名を指定した `-cpu` フラグを使用します。

Ubuntu 22.04 Qcow2/Raw

```
packer build -only=general-build.qemu.ubuntu22 template.pkr.hcl
```

Ubuntu 24.04 Qcow2/Raw

```
packer build -only=general-build.qemu.ubuntu24 template.pkr.hcl
```

RHEL 8 Qcow2/Raw

```
packer build -only=general-build.qemu.rhel8 template.pkr.hcl
```

RHEL 9 Qcow2/Raw

```
packer build -only=general-build.qemu.rhel9 template.pkr.hcl
```

nodeadm 設定を user-data を通して渡す

cloud-init を介して nodeadm の設定を user-data で渡すことで、ホストの起動時にハイブリッドノードを設定し、自動的に EKS クラスターに接続することができます。以下は、ハイブリッドノードのインフラストラクチャとして VMware vSphere を使用する場合の実装例です。

1. GitHub の govc readme の手順に従って govc CLI をインストールします。
2. 前のセクションで Packer ビルドを実行してテンプレートをプロビジョニングした後は、以下を使用してテンプレートをクローンし、複数の異なるノードを作成することができます。ハイブリッドノードに使用する新規 VM を作成するたびに、テンプレートをクローンする必要があります。以下のコマンド内の変数は、お使いの環境に応じた値に置き換えます。以下のコマンドの VM_NAME は、metadata.yaml ファイルを介して VM の名前を注入する際に NODE_NAME として使用されます。

   ```
   govc vm.clone -vm "/PATH/TO/TEMPLATE" -ds="YOUR_DATASTORE" \
       -on=false -template=false -folder=/FOLDER/TO/SAVE/VM "VM_NAME"
   ```

3. 各新規 VM のテンプレートをクローンした後は、VM の metadata.yaml と userdata.yaml を作成します。VM は同じ userdata.yaml と metadata.yaml を共有でき、以下のステップで VM ごとにこれらを入力していきます。nodeadm の設定は、userdata.yaml の write_files セクションで作成および定義されます。以下の例では、ハイブリッドノードのオンプレミス認証情報プロバイダーとして AWS SSM ハイブリッドアクティベーションを使用していま

す。nodeadm 設定の詳細については、「the section called “ハイブリッドノード用 nodeadm”」を参照してください。

userdata.yaml:

```
#cloud-config
users:
  - name: # username for login. Use 'builder' for RHEL or 'ubuntu' for Ubuntu.
    passwd: # password to login. Default is 'builder' for RHEL.
    groups: [adm, cdrom, dip, plugdev, lxd, sudo]
    lock-passwd: false
    sudo: ALL=(ALL) NOPASSWD:ALL
    shell: /bin/bash

write_files:
  - path: /usr/local/bin/nodeConfig.yaml
    permissions: '0644'
    content: |
      apiVersion: node.eks.aws/v1alpha1
      kind: NodeConfig
      spec:
          cluster:
              name: # Cluster Name
              region: # AWS region
          hybrid:
              ssm:
                  activationCode: # Your ssm activation code
                  activationId: # Your ssm activation id

runcmd:
  - /usr/local/bin/nodeadm init -c file:///usr/local/bin/nodeConfig.yaml >> /var/log/
nodeadm-init.log 2>&1
```

metadata.yaml:

お使いの環境に応じた metadata.yaml を作成します。"$NODE_NAME" 変数の形式は、後続のステップで値が入力されるため、ファイル内にそのまま保持してください。

```
instance-id: "$NODE_NAME"
local-hostname: "$NODE_NAME"
network:
  version: 2
```

```
  ethernets:
    nics:
      match:
        name: ens*
      dhcp4: yes
```

4. 以下のコマンドを使用して、userdata.yaml および metadata.yaml ファイルを gzip +base64 文字列として追加します。以下のコマンドは、作成する VM ごとに実行する必要があります。VM_NAME は、更新する VM の名前に置き換えてください。

```
export NODE_NAME="VM_NAME"
export USER_DATA=$(gzip -c9 <userdata.yaml | base64)

govc vm.change -dc="YOUR_DATASTORE" -vm "$NODE_NAME" -e
 guestinfo.userdata="${USER_DATA}"
govc vm.change -dc="YOUR_DATASTORE" -vm "$NODE_NAME" -e
 guestinfo.userdata.encoding=gzip+base64

envsubst '$NODE_NAME' < metadata.yaml > metadata.yaml.tmp
export METADATA=$(gzip -c9 <metadata.yaml.tmp | base64)

govc vm.change -dc="YOUR_DATASTORE" -vm "$NODE_NAME" -e
 guestinfo.metadata="${METADATA}"
govc vm.change -dc="YOUR_DATASTORE" -vm "$NODE_NAME" -e
 guestinfo.metadata.encoding=gzip+base64
```

5. 新しい VM の電源をオンにします。これにより、設定した EKS クラスターに自動的に接続されます。

```
govc vm.power -on "${NODE_NAME}"
```

ハイブリッドノードの認証情報を準備する

Amazon EKS Hybrid Nodes は、AWS SSM ハイブリッドアクティベーションまたは AWS IAM Roles Anywhere によってプロビジョニングされた一時的な IAM 認証情報を使用して、Amazon EKS クラスターで認証します。Amazon EKS Hybrid Nodes CLI (nodeadm) を用いて、AWS SSM ハイブリッドアクティベーションまたは AWS IAM Roles Anywhere を使用する必要があります。AWS SSM ハイブリッドアクティベーションと AWS IAM Roles Anywhere の両方を使用しないでください。認証局 (CA) およびオンプレミス環境の証明書を持つ既存の公開鍵基盤 (PKI) がない場合、AWS SSM ハ

イブリッドアクティベーションを使用することをお勧めします。既存の PKI と証明書がオンプレミスにある場合は、AWS IAM Roles Anywhere を使用します。

ハイブリッドノードの IAM ロール

ハイブリッドノードを Amazon EKS クラスターに接続する前に、SSM ハイブリッドアクティベーションで使用する AWS IAM ロールまたはハイブリッドノードの認証情報用の AWS IAM Roles Anywhere を作成する必要があります。クラスターの作成後、Amazon EKS アクセスエントリまたは `aws-auth` ConfigMap エントリでこのロールを使用して、IAM ロールを Kubernetes ロールベースのアクセスコントロール (RBAC) にマッピングします。ハイブリッドノードの IAM ロールと Kubernetes RBAC の関連付けの詳細については、「the section called “クラスターアクセスを準備する”」を参照してください。

ハイブリッドノードの IAM ロールには、以下のアクセス許可が必要です。

- ハイブリッドノードを接続するクラスターに関する情報を収集するための `eks:DescribeCluster` アクションを、`nodeadm` が使用するためのアクセス許可。`eks:DescribeCluster` アクションを有効にしない場合は、`nodeadm init` コマンドに渡すノード設定で、Kubernetes API エンドポイント、クラスター CA バンドル、およびサービス IPv4 CIDR を渡す必要があります。
- ハイブリッドノードを接続するクラスターのアクセスエントリを一覧表示するための `eks:ListAccessEntries` アクションを、`nodeadm` が使用するためのアクセス許可。`eks:ListAccessEntries` アクションを有効にしない場合は、`nodeadm init` コマンドの実行時に `--skip cluster-access-validation` フラグを渡す必要があります。
- kubelet が「AmazonEC2ContainerRegistryPullOnly」 ポリシーで定義されているように、Amazon Elastic Container Registry (Amazon ECR) のコンテナイメージを使用するためのアクセス許可。
- AWS SSM を使用している場合、https://docs.aws.amazon.com/aws-managed-policy/latest/reference/AmazonSSMManagedInstanceCore.html ポリシーで定義されている AWS SSM ハイブリッドアクティベーションを使用するための `nodeadm init` のアクセス許可。
- AWS SSM を使用している場合、`nodeadm uninstall` がインスタンスを登録解除するために `ssm:DeregisterManagedInstance` アクションと `ssm:DescribeInstanceInformation` アクションを使用するためのアクセス許可。
- (オプション) Amazon EKS Pod Identity エージェントが `eks-auth:AssumeRoleForPodIdentity` アクションを使用してポッドの認証情報を取得するためのアクセス許可。

AWS SSM ハイブリッドアクティベーションのセットアップ

SSM 認証情報プロバイダーに必要な nodeadm バージョン

ハイブリッドノードの認証情報プロバイダーとして AWS Systems Manager (SSM) を使用する場合は、新しいインストールとアップグレードに nodeadm バージョン `1.0.19` 以降を使用する必要があります。以前のバージョンの nodeadm には古い SSM 署名キーが含まれており、`nodeadm install` と `nodeadm upgrade` の実行中に失敗し、次の署名検証エラーが発生します:

```
"msg":"Command failed","error":"failed to install ssm installer: validating ssm-setup-cli signature: Signature Verification Error: No matching signature"
```

このエラーを解決するには、`nodeadm install` または `nodeadm upgrade` を実行する前に nodeadm の最新バージョンをダウンロードします。

Important

アクティベーションの作成方法に応じて、Systems Manager はすぐにアクティベーションコードと ID をコンソールまたはコマンドウィンドウに返します。この情報をコピーして、安全な場所に保管します。コンソールから離れたり、コマンドウィンドウを閉じたりすると、この情報は失われる可能性があります。紛失した場合は、新しいアクティベーションを作成する必要があります。

AWS SSM ハイブリッドアクティベーションを設定する前に、ハイブリッドノードの IAM ロールを作成して設定する必要があります。詳細については、「the section called “ハイブリッドノードの IAM ロールを作成する”」を参照してください。「AWS Systems Manager ユーザーガイド」の「Systems Manager にハイブリッドノードを登録するためのハイブリッドアクティベーションの作成」の指示に従って、ハイブリッドノード用の AWS SSM ハイブリッドアクティベーションを作成します。ホストを Amazon EKS クラスターのハイブリッドノードとして登録する際に、nodeadm でアクティベーションコードと ID を使用します。ハイブリッドノード用の Amazon EKS クラスターを作成し準備した後は、このステップに戻ることができます。

デフォルトでは、AWS SSM ハイブリッドアクティベーションは 24 時間アクティブになります。または、ハイブリッドアクティベーションを作成する際に、`2024-08-01T00:00:00` のようなタイムスタンプ形式で `--expiration-date` を指定することもできます。認証情報プロバイダーとして

AWS SSM を使用する場合、ハイブリッドノードのノード名は設定できず、AWS SSM によって自動生成されます。AWS SSM マネージドインスタンスは、Fleet Manager の AWS Systems Manager コンソールで表示および管理できます。追加コストなしで、アカウントにつき AWS リージョンごとに最大 1,000 のスタンダードハイブリッドアクティベーションノードを登録できます。ただし、1,000 を超えるハイブリッドノードを登録するには、アドバンストインスタンス層のアクティブ化が必要です。これには、Amazon EKS Hybrid Nodes の料金に含まれていない、アドバンストインスタンス層の利用料金が発生します。詳細については、「AWS Systems Manager の料金」を参照してください。

ハイブリッドノードの IAM ロールを使用して AWS SSM ハイブリッドアクティベーションを作成する方法については、以下の例を参照してください。ハイブリッドノードの認証情報に AWS SSM ハイブリッドアクティベーションを使用する場合、ハイブリッドノードの名前は `mi-012345678abcdefgh` 形式になり、AWS SSM によってプロビジョニングされた一時的な認証情報は 1 時間有効です。AWS SSM を認証情報プロバイダーとして使用する場合、ノード名または認証情報の期間を変更することはできません。一時的な認証情報は AWS SSM によって自動的にローテーションされ、ローテーションはノードやアプリケーションのステータスには影響しません。

ハイブリッドノードの IAM ロールの AWS SSM の `ssm:DeregisterManagedInstance` アクセス許可の範囲を、お使いの AWS SSM ハイブリッドアクティベーションに関連付けられたインスタンスしか登録解除できないよう制限するために、EKS クラスターごとに 1 つの AWS SSM ハイブリッドアクティベーションを使用することをお勧めします。このページの例では、EKS クラスター ARN を含むタグが使用され、AWS SSM ハイブリッドアクティベーションを EKS クラスターにマッピングするために使用できます。または、任意のタグと方法を使用して、アクセス許可の境界と要件に基づいて AWS SSM アクセス許可の範囲を絞り込むこともできます。以下のコマンドの `REGISTRATION_LIMIT` オプションは、AWS SSM ハイブリッドアクティベーションを使用できるマシンの数を制限するために使用される整数です (`10` など)。

```
aws ssm create-activation \
     --region AWS_REGION \
     --default-instance-name eks-hybrid-nodes \
     --description "Activation for EKS hybrid nodes" \
     --iam-role AmazonEKSHybridNodesRole \
     --tags Key=EKSClusterARN,Value=arn:aws:eks:AWS_REGION:AWS_ACCOUNT_ID:cluster/
CLUSTER_NAME \
     --registration-limit REGISTRATION_LIMIT
```

AWS SSM ハイブリッドアクティベーションで使用可能な構成設定の詳細については、「ハイブリッドアクティベーションを作成して Systems Manager にノードを登録する」の手順を確認してください。

AWS IAM Roles Anywhere の設定

「IAM Roles Anywhere ユーザーガイド」の「IAM Roles Anywhere の開始方法」の手順に従って、ハイブリッドノードの IAM ロールの一時的な IAM 認証情報に使用するトラストアンカーとプロファイルを設定します。プロファイルを作成する際は、ロールを追加せずにプロファイルを作成できます。プロファイルを作成してから、ハイブリッドノードの IAM ロールを作成するステップに戻り、作成したプロファイルにロールを追加することもできます。または、このページの後半にある AWS CloudFormation ステップを使用して、ハイブリッドノードの IAM Roles Anywhere セットアップを完了することもできます。

ハイブリッドノードの IAM ロールをプロファイルに追加するときは、AWS IAM Roles Anywhere コンソールの [プロファイルの編集] ページの下部にある [カスタムロール] セッション名パネルで [カスタムロールセッション名を受け入れる] を選択します。これは `CreateProfile` API の acceptRoleSessionName フィールドに対応します。これにより、ブートストラッププロセス中に `nodeadm` に渡す設定内で、ハイブリッドノードのカスタムノード名を指定できます。`nodeadm init` プロセス中にカスタムノード名を渡す必要があります。プロファイルの作成後、カスタムロールセッション名を受け入れるようにプロファイルを更新できます。

AWS IAM Roles Anywhere では、AWS IAM Roles Anywhere プロファイルの durationSeconds フィールドを通して認証情報の有効期間を設定できます。デフォルトの期間は 1 時間で、最大は 12 時間です。ハイブリッドノードの IAM ロールでの `MaxSessionDuration` 設定は、AWS IAM Roles Anywhere プロファイルの `durationSeconds` 設定よりも大きくする必要があります。`MaxSessionDuration` の詳細については、「UpdateRole API documentation」を参照してください。

認証機関 (CA) から生成するマシンごとの証明書とキーは、ファイル名を証明書は `server.pem`、キーは `server.key` として、各ハイブリッドノードの `/etc/iam/pki` ディレクトリに配置する必要があります。

ハイブリッドノードの IAM ロールを作成する

このセクションのステップを実行するには、AWS コンソールまたは AWS CLI を使用する IAM プリンシパルに以下のアクセス許可が必要です。

- `iam:CreatePolicy`
- `iam:CreateRole`
- `iam:AttachRolePolicy`
- AWS IAM Roles Anywhere を使用している場合

- rolesanywhere:CreateTrustAnchor
- rolesanywhere:CreateProfile
- iam:PassRole

AWS CloudFormation

まだである場合は、AWS CLI をインストールして設定します。「AWS CLI の最新バージョンのインストールまたは更新」を参照してください。

AWS SSM ハイブリッドアクティベーションのステップ

CloudFormation スタックは、上記のアクセス許可を持つハイブリッドノードの IAM ロールを作成します。CloudFormation テンプレートは、AWS SSM ハイブリッドアクティベーションを作成しません。

1. ハイブリッドノード用の AWS SSM CloudFormation テンプレートをダウンロードします。

```
curl -OL 'https://raw.githubusercontent.com/aws/eks-hybrid/refs/heads/main/example/
hybrid-ssm-cfn.yaml'
```

2. 以下のオプションを使用して cfn-ssm-parameters.json を作成します。
   a. ROLE_NAME をハイブリッドノードの IAM ロールの名前に置き換えます。デフォルトでは、名前を指定しない場合、CloudFormation テンプレートは作成するロールの名前として AmazonEKSHybridNodesRole を使用します。
   b. TAG_KEY を、AWS SSM ハイブリッドアクティベーションの作成時に使用した AWS SSM リソースのタグキーに置き換えます。タグキーとタグ値の組み合わせは、ssm:DeregisterManagedInstance の条件で使用されます。これにより、ハイブリッドノードの IAM ロールは、ユーザーの AWS SSM ハイブリッドアクティベーションに関連付けられている AWS SSM マネージドインスタンスの登録解除のみ許可されるようになります。CloudFormation テンプレートでは、TAG_KEY はデフォルトで EKSClusterARN になります。
   c. TAG_VALUE を、AWS SSM ハイブリッドアクティベーションの作成時に使用した SSM AWS リソースタグ値に置き換えます。タグキーとタグ値の組み合わせは、ssm:DeregisterManagedInstance の条件で使用されます。これにより、ハイブリッドノードの IAM ロールは、ユーザーの AWS SSM ハイブリッドアクティベーションに関連付けられている AWS SSM マネージドインスタンスの登録解除のみ許可されるようになります。EKSClusterARN のデフォルトの TAG_KEY を使用している場合

は、EKS クラスター ARN を TAG_VALUE として渡します。EKS クラスター ARN の形式は arn:aws:eks:AWS_REGION:AWS_ACCOUNT_ID:cluster/CLUSTER_NAME になります。

```
{
  "Parameters": {
    "RoleName": "ROLE_NAME",
    "SSMDeregisterConditionTagKey": "TAG_KEY",
    "SSMDeregisterConditionTagValue": "TAG_VALUE"
  }
}
```

3. CloudFormation スタックをデプロイします。STACK_NAME は、その CloudFormation スタックに付けた任意の名前に置き換えてください。

```
aws cloudformation deploy \
    --stack-name STACK_NAME \
    --template-file hybrid-ssm-cfn.yaml \
    --parameter-overrides file://cfn-ssm-parameters.json \
    --capabilities CAPABILITY_NAMED_IAM
```

AWS IAM Roles Anywhere のステップ

CloudFormation スタックは、設定した認証機関 (CA) を使用して AWS IAM Roles Anywhere のトラストアンカーを作成し、AWS IAM Roles Anywhere プロファイルを作成し、前述のアクセス許可を使用して、ハイブリッドノードの IAM ロールを作成します。

1. 認証機関 (CA) のセットアップ
   a. AWS Private CA リソースを使用するには、AWS Private Certificate Authority コンソールを開きます。「AWS Private CA ユーザーガイド」の指示に従ってください。
   b. 外部 CA を使用するには、CA が提供する手順に従ってください。証明書本文は後のステップで提出します。
   c. パブリック CA から発行された証明書をトラストアンカーとして使用することはできません。
2. ハイブリッドノード用の AWS IAM Roles Anywhere CloudFormation テンプレートをダウンロードします

```
curl -OL 'https://raw.githubusercontent.com/aws/eks-hybrid/refs/heads/main/example/
hybrid-ira-cfn.yaml'
```

3. 以下のオプションを使用して cfn-iamra-parameters.json を作成します。

a. ROLE_NAME をハイブリッドノードの IAM ロールの名前に置き換えます。デフォルトでは、名前を指定しない場合、CloudFormation テンプレートは作成するロールの名前として AmazonEKSHybridNodesRole を使用します。

b. CERT_ATTRIBUTE を、ホストを一意に識別するマシンごとの証明書属性に置き換えます。使用する証明書属性は、ハイブリッドノードをクラスターに接続するときに nodeadm 設定に使用する nodeName と一致する必要があります。詳細については、「the section called "ハイブリッドノード用 nodeadm"」を参照してください。デフォルトでは、CloudFormation テンプレートは ${aws:PrincipalTag/x509Subject/CN} を CERT_ATTRIBUTE として使用します。これは、マシンごとの証明書の CN フィールドに対応します。または、$(aws:PrincipalTag/x509SAN/Name/CN} を CERT_ATTRIBUTE として渡すこともできます。

c. CA_CERT_BODY を、CA の証明書本文から改行を取り除いたものに置き換えます。CA_CERT_BODY は、プライバシー強化メール (PEM) 形式である必要がありす。PEM 形式の CA 証明書がある場合は、CA 証明書本文を cfn-iamra-parameters.json ファイルに配置する前に、改行と BEGIN CERTIFICATE 行と END CERTIFICATE 行を削除します。

```
{
  "Parameters": {
    "RoleName": "ROLE_NAME",
    "CertAttributeTrustPolicy": "CERT_ATTRIBUTE",
    "CABundleCert": "CA_CERT_BODY"
  }
}
```

4. CloudFormation テンプレートをデプロイする。STACK_NAME は、その CloudFormation スタックに付けた任意の名前に置き換えてください。

```
aws cloudformation deploy \
    --stack-name STACK_NAME \
    --template-file hybrid-ira-cfn.yaml \
    --parameter-overrides file://cfn-iamra-parameters.json
    --capabilities CAPABILITY_NAMED_IAM
```

AWS CLI

まだである場合は、AWS CLI をインストールして設定します。「AWS CLI の最新バージョンのインストールまたは更新」を参照してください。

EKS DescribeCluster ポリシーを作成する

1. 次の内容で、eks-describe-cluster-policy.json という名前のファイルを作成します。

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "eks:DescribeCluster"
            ],
            "Resource": "*"
        }
    ]
}
```

2. 次のコマンドを使用してポリシーを作成します。

```
aws iam create-policy \
    --policy-name EKSDescribeClusterPolicy \
    --policy-document file://eks-describe-cluster-policy.json
```

AWS SSM ハイブリッドアクティベーションのステップ

1. 次の内容で、eks-hybrid-ssm-policy.json という名前のファイルを作成します。このポリシーは、ssm:DescribeInstanceInformation と ssm:DeregisterManagedInstance の 2 つのアクションのアクセス許可を付与します。このポリシーは、ssm:DeregisterManagedInstance へのアクセス許可の範囲を、信頼ポリシーで指定したリソースタグに基づき、AWS SSM ハイブリッドアクティベーションに関連付けられた AWS SSM マネージドインスタンスに制限します。
   a. AWS_REGION を、AWS SSM ハイブリッドアクティベーションの AWS リージョンに置き換えます。
   b. AWS_ACCOUNT_ID を AWS アカウント ID に置き換えます。
   c. TAG_KEY を、AWS SSM ハイブリッドアクティベーションの作成時に使用した AWS SSM リソースのタグキーに置き換えます。タグキーとタグ値の組み合わせは、ssm:DeregisterManagedInstance の条件で使用されます。これにより、ハイブリッドノードの IAM ロールは、ユーザーの AWS SSM ハイブリッドアクティベーションに関連

付けられている AWS SSM マネージドインスタンスの登録解除のみ許可されるようになります。CloudFormation テンプレートでは、TAG_KEY はデフォルトで EKSClusterARN になります。

d. TAG_VALUE を、AWS SSM ハイブリッドアクティベーションの作成時に使用した SSM AWS リソースタグ値に置き換えます。タグキーとタグ値の組み合わせは、ssm:DeregisterManagedInstance の条件で使用されます。これにより、ハイブリッドノードの IAM ロールは、ユーザーの AWS SSM ハイブリッドアクティベーションに関連付けられている AWS SSM マネージドインスタンスの登録解除のみ許可されるようになります。EKSClusterARN のデフォルトの TAG_KEY を使用している場合は、EKS クラスター ARN を TAG_VALUE として渡します。EKS クラスター ARN の形式は arn:aws:eks:AWS_REGION:AWS_ACCOUNT_ID:cluster/CLUSTER_NAME になります。

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "ssm:DescribeInstanceInformation",
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": "ssm:DeregisterManagedInstance",
            "Resource": "arn:aws:ssm:us-east-1:123456789012:managed-instance/*",
            "Condition": {
                "StringEquals": {
                    "ssm:resourceTag/TAG_KEY": "TAG_VALUE"
                }
            }
        }
    ]
}
```

2. 次のコマンドを使用してポリシーを作成します。

```
aws iam create-policy \
    --policy-name EKSHybridSSMPolicy \
    --policy-document file://eks-hybrid-ssm-policy.json
```

3. eks-hybrid-ssm-trust.json という名前のファイルを作成します。AWS_REGION を AWS SSM ハイブリッドアクティベーションの AWS リージョンに、AWS_ACCOUNT_ID をお使いの AWS アカウント ID に置き換えます。

```
{
   "Version":"2012-10-17",
   "Statement":[
      {
         "Sid":"",
         "Effect":"Allow",
         "Principal":{
            "Service":"ssm.amazonaws.com"
         },
         "Action":"sts:AssumeRole",
         "Condition":{
            "StringEquals":{
               "aws:SourceAccount":"123456789012"
            },
            "ArnEquals":{
               "aws:SourceArn":"arn:aws:ssm:us-east-1:123456789012:*"
            }
         }
      }
   ]
}
```

4. 次のコマンドを使用してロールを作成します。

```
aws iam create-role \
    --role-name AmazonEKSHybridNodesRole \
    --assume-role-policy-document file://eks-hybrid-ssm-trust.json
```

5. 前のステップで作成した EKSDescribeClusterPolicy と EKSHybridSSMPolicy をアタッチします。AWS_ACCOUNT_ID を AWS アカウント ID に置き換えます。

```
aws iam attach-role-policy \
    --role-name AmazonEKSHybridNodesRole \
    --policy-arn arn:aws:iam::AWS_ACCOUNT_ID:policy/EKSDescribeClusterPolicy
```

```
aws iam attach-role-policy \
    --role-name AmazonEKSHybridNodesRole \
```

```
    --policy-arn arn:aws:iam::AWS_ACCOUNT_ID:policy/EKSHybridSSMPolicy
```

6. AmazonEC2ContainerRegistryPullOnly と AmazonSSMManagedInstanceCore AWS マネージドポリシーをアタッチします。

```
aws iam attach-role-policy \
    --role-name AmazonEKSHybridNodesRole \
    --policy-arn arn:aws:iam::aws:policy/AmazonEC2ContainerRegistryPullOnly
```

```
aws iam attach-role-policy \
    --role-name AmazonEKSHybridNodesRole \
    --policy-arn arn:aws:iam::aws:policy/AmazonSSMManagedInstanceCore
```

AWS IAM Roles Anywhere のステップ

AWS IAM Roles Anywhere を使用するには、ハイブリッドノードの IAM ロールを作成する前に、AWS IAM Roles Anywhere のトラストアンカーを設定する必要があります。手順については「the section called “AWS IAM Roles Anywhere の設定”」を参照してください。

1. eks-hybrid-iamra-trust.json という名前のファイルを作成します。TRUST_ANCHOR ARN を、the section called “AWS IAM Roles Anywhere の設定” のステップで作成したトラストアンカーの ARN に置き換えます。この信頼ポリシーの条件は、AWS IAM Roles Anywhere がハイブリッドノードの IAM ロールを引き受けて一時的な IAM 認証情報を交換する機能を、ロールセッション名がハイブリッドノードにインストールされている x509 証明書の CN と一致する場合のみに制限します。別の方法として、他の証明書属性を使用してノードを一意に識別することもできます。信頼ポリシーで使用する証明書属性は、nodeadm 設定で設定した nodeName に対応している必要があります。詳細については、「the section called “ハイブリッドノード用 nodeadm”」を参照してください。

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "Service": "rolesanywhere.amazonaws.com"
            },
            "Action": [
                "sts:TagSession",
```

```
                "sts:SetSourceIdentity"
            ],
            "Condition": {
                "StringEquals": {
                    "aws:PrincipalTag/x509Subject/CN": "${aws:PrincipalTag/
x509Subject/CN}"
                },
                "ArnEquals": {
                    "aws:SourceArn": "arn:aws:rolesanywhere:us-
east-1:123456789012:trust-anchor/TA_ID"
                }
            }
        },
        {
            "Effect": "Allow",
            "Principal": {
                "Service": "rolesanywhere.amazonaws.com"
            },
            "Action": "sts:AssumeRole",
            "Condition": {
                "StringEquals": {
                    "sts:RoleSessionName": "${aws:PrincipalTag/x509Subject/CN}",
                    "aws:PrincipalTag/x509Subject/CN": "${aws:PrincipalTag/
x509Subject/CN}"
                },
                "ArnEquals": {
                    "aws:SourceArn": "arn:aws:rolesanywhere:us-
east-1:123456789012:trust-anchor/TA_ID"
                }
            }
        }
    ]
}
```

2. 次のコマンドを使用してロールを作成します。

```
aws iam create-role \
    --role-name AmazonEKSHybridNodesRole \
    --assume-role-policy-document file://eks-hybrid-iamra-trust.json
```

3. 前のステップで作成した EKSDescribeClusterPolicy をアタッチします。AWS_ACCOUNT_ID を AWS アカウント ID に置き換えます。

```
aws iam attach-role-policy \
    --role-name AmazonEKSHybridNodesRole \
    --policy-arn arn:aws:iam::AWS_ACCOUNT_ID:policy/EKSDescribeClusterPolicy
```

4. AmazonEC2ContainerRegistryPullOnly AWS マネージドポリシーをアタッチします。

```
aws iam attach-role-policy \
    --role-name AmazonEKSHybridNodesRole \
    --policy-arn arn:aws:iam::aws:policy/AmazonEC2ContainerRegistryPullOnly
```

AWS マネジメントコンソール

EKS DescribeCluster ポリシーを作成する

1. Amazon IAM コンソールを開きます。
2. 左のナビゲーションペインで、 [ポリシー] を選択してください。
3. [ポリシー] ページで、[ポリシーの作成] を選択します。
4. [アクセス許可を指定] ページの [サービスの選択] パネルで、[EKS] を選択します。
   a. DescribeCluster でアクションをフィルタリングし、DescribeCluster の Read アクションを選択します。
   b. [次へ] を選択します。
5. [確認と作成] ページで以下を行います。
   a. ポリシーのポリシー名を、たとえば EKSDescribeClusterPolicy のように入力します。
   b. [Create policy] (ポリシーの作成) を選択します。

AWS SSM ハイブリッドアクティベーションのステップ

1. Amazon IAM コンソールを開きます。
2. 左のナビゲーションペインで、 [ポリシー] を選択してください。
3. [ポリシー] ページで、[ポリシーの作成] を選択します。
4. [アクセス許可を指定] ページの [ポリシーエディタ] の右上のナビゲーションで、[JSON] を選択します。以下のスニペットを貼り付けます。AWS_REGION を AWS SSM ハイブリッドアクティベーションの AWS リージョンに置き換え、AWS_ACCOUNT_ID をお使いの AWS アカウント ID に置き換えます。TAG_KEY と TAG_VALUE を、AWS SSM ハイブリッドアクティベーションの作成時に使用した AWS SSM リソースのタグキーに置き換えます。

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "ssm:DescribeInstanceInformation",
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": "ssm:DeregisterManagedInstance",
            "Resource": "arn:aws:ssm:us-east-1:123456789012:managed-instance/*",
            "Condition": {
                "StringEquals": {
                    "ssm:resourceTag/TAG_KEY": "TAG_VALUE"
                }
            }
        }
    ]
}
```

a. [次へ] を選択します。

5. [確認と作成] ページで以下を行います。

a. ポリシーのポリシー名を、たとえば EKSHybridSSMPolicy のように入力します。

b. [ポリシーの作成] を選択します。

6. 左のナビゲーションペインで、[ロール] を選択してください。

7. [ロール] ページで、[ロールの作成] を選択してください。

8. [信頼されたエンティティを選択] ページで、以下の操作を実行します。

a. [信頼されたエンティティのタイプ] セクションで [カスタム信頼ポリシー] を選択します。以下をカスタム信頼ポリシーエディタに貼り付けます。AWS_REGION を AWS SSM ハイブリッドアクティベーションの AWS リージョンに、AWS_ACCOUNT_ID をお使いの AWS アカウント ID に置き換えます。

```
{
   "Version":"2012-10-17",
   "Statement":[
      {
         "Sid":"",
```

```
            "Effect":"Allow",
            "Principal":{
               "Service":"ssm.amazonaws.com"
            },
            "Action":"sts:AssumeRole",
            "Condition":{
               "StringEquals":{
                  "aws:SourceAccount":"123456789012"
               },
               "ArnEquals":{
                  "aws:SourceArn":"arn:aws:ssm:us-east-1:123456789012:*"
               }
            }
         }
      ]
}
```

b. [次へ] を選択します。

9. [アクセス許可を追加] ページで、カスタムポリシーをアタッチするか、以下の操作を行います。

   a. [フィルターポリシー] ボックスに、EKSDescribeClusterPolicy または上記で作成したポリシーの名前を入力します。検索結果でポリシー名の左にあるチェックボックスを選択します。

   b. [フィルターポリシー] ボックスに、EKSHybridSSMPolicy または上記で作成したポリシーの名前を入力します。検索結果でポリシー名の左にあるチェックボックスを選択します。

   c. [フィルタポリシー] ボックスに AmazonEC2ContainerRegistryPullOnly と入力します。検索結果の AmazonEC2ContainerRegistryPullOnly の左にあるチェックボックスを選択します。

   d. [フィルタポリシー] ボックスに AmazonSSMManagedInstanceCore と入力します。検索結果の AmazonSSMManagedInstanceCore の左にあるチェックボックスを選択します。

   e. [次へ] を選択します。

10. [名前を付けて、レビューし、作成する] ページで、以下の操作を実行します。

    a. [ロール名] に、AmazonEKSHybridNodesRole などのロールの一意の名前を入力します。

    b. [Description] (説明) では、現在のテキストを「Amazon EKS - Hybrid Nodes role」などの説明文に置き換えます。

    c. [ロールの作成] を選択してください。

AWS IAM Roles Anywhere のステップ

AWS IAM Roles Anywhere を使用するには、ハイブリッドノードの IAM ロールを作成する前に、AWS IAM Roles Anywhere のトラストアンカーを設定する必要があります。手順については「the section called “AWS IAM Roles Anywhere の設定”」を参照してください。

1. Amazon IAM コンソールを開きます。
2. 左のナビゲーションペインで、[ロール] を選択してください。
3. [ロール] ページで、[ロールの作成] を選択してください。
4. [信頼されたエンティティを選択] ページで、以下の操作を実行します。
   a. [信頼されたエンティティのタイプ] セクションで [カスタム信頼ポリシー] を選択します。以下をカスタム信頼ポリシーエディタに貼り付けます。TRUST_ANCHOR ARN を、the section called “AWS IAM Roles Anywhere の設定” のステップで作成したトラストアンカーの ARN に置き換えます。この信頼ポリシーの条件は、AWS IAM Roles Anywhere がハイブリッドノードの IAM ロールを引き受けて一時的な IAM 認証情報を交換する機能を、ロールセッション名がハイブリッドノードにインストールされている x509 証明書の CN と一致する場合のみに制限します。別の方法として、他の証明書属性を使用してノードを一意に識別することもできます。信頼ポリシーで使用する証明書属性は、nodeadm 設定で設定した nodeName に対応している必要があります。詳細については、「the section called “ハイブリッドノード用 nodeadm”」を参照してください。

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "Service": "rolesanywhere.amazonaws.com"
            },
            "Action": [
                "sts:TagSession",
                "sts:SetSourceIdentity"
            ],
            "Condition": {
                "StringEquals": {
                    "aws:PrincipalTag/x509Subject/CN": "${aws:PrincipalTag/
x509Subject/CN}"
                },
                "ArnEquals": {
                    "aws:SourceArn": "arn:aws:rolesanywhere:us-
east-1:123456789012:trust-anchor/TA_ID"
```

```
                }
            }
        },
        {
            "Effect": "Allow",
            "Principal": {
                "Service": "rolesanywhere.amazonaws.com"
            },
            "Action": "sts:AssumeRole",
            "Condition": {
                "StringEquals": {
                    "sts:RoleSessionName": "${aws:PrincipalTag/x509Subject/CN}",
                    "aws:PrincipalTag/x509Subject/CN": "${aws:PrincipalTag/
x509Subject/CN}"
                },
                "ArnEquals": {
                    "aws:SourceArn": "arn:aws:rolesanywhere:us-
east-1:123456789012:trust-anchor/TA_ID"
                }
            }
        }
    ]
}
```

b. [次へ] を選択します。

5. [アクセス許可を追加] ページで、カスタムポリシーをアタッチするか、以下の操作を行います。

   a. [フィルターポリシー] ボックスに、EKSDescribeClusterPolicy または上記で作成したポリシーの名前を入力します。検索結果でポリシー名の左にあるチェックボックスを選択します。

   b. [フィルタポリシー] ボックスに AmazonEC2ContainerRegistryPullOnly と入力します。検索結果の AmazonEC2ContainerRegistryPullOnly の左にあるチェックボックスを選択します。

   c. [次へ] を選択します。

6. [名前を付けて、レビューし、作成する] ページで、以下の操作を実行します。

   a. [ロール名] に、AmazonEKSHybridNodesRole などのロールの一意の名前を入力します。

   b. [Description] (説明) では、現在のテキストを「Amazon EKS - Hybrid Nodes role」などの説明文に置き換えます。

   c. [ロールの作成] を選択してください。

ハイブリッドノードを使用して Amazon EKS クラスターを作成する

このトピックでは、使用可能なオプションの概要と、ハイブリッドノード対応の Amazon EKS クラスターの作成時に考慮すべき点を説明します。EKS Hybrid Nodes には、クラウドノードを使用する Amazon EKS クラスターと同じ Kubernetes バージョンのサポートがあります (標準サポートや拡張サポートなど)。

EKS Hybrid Nodes を使用する予定がない場合は、「the section called “クラスターを作成する”」にある Amazon EKS でのクラスター作成に関する基本ドキュメントを参照してください。

前提条件

- the section called “前提条件” が完了していること。ハイブリッドノード対応クラスターを作成する前に、オンプレミスノードとオプションでポッド CIDR を特定し、EKS の要件とハイブリッドノードの要件に従って VPC とサブネットを作成し、オンプレミスおよびオプションでポッド CIDR のインバウンドルールを持つセキュリティグループを指定する必要があります。前提条件の詳細については、the section called “ネットワークを準備する” を参照してください。
- AWS コマンドラインインターフェイス (AWS CLI) の最新バージョンがデバイスにインストールおよび設定されていること。現在のバージョンを確認するには、`aws --version` を参照してください。yum、apt-get、macOS 用の Homebrew などのパッケージマネージャーは、多くの場合 AWS CLI の最新バージョンより数バージョン古くなっています。最新バージョンをインストールするには、「AWS Command Line Interface ユーザーガイド」の「AWS CLI の最新バージョンのインストールまたは更新」および「AWS CLI の設定を構成する」を参照してください。
- IAM ロールを作成してポリシーをアタッチし、EKS クラスターを作成および記述するアクセス許可を持つ IAM プリンシパルがあること

考慮事項

- クラスターは、クラスター認証モードに `API` または `API_AND_CONFIG_MAP` を使用する必要があります。
- クラスターは IPv4 アドレスファミリーを使用する必要があります。
- クラスターは、パブリッククラスターエンドポイント接続またはプライベートクラスターエンドポイント接続を使用する必要があります。ハイブリッドノードが VPC 外で実行される場合、Amazon EKS Kubernetes API サーバーエンドポイントがパブリック IP に解決されるため、クラスターは「パブリックおよびプライベート」のクラスターエンドポイント接続を使用できません。
- OIDC 認証は、ハイブリッドノードが有効な EKS クラスターでサポートされています。

- 既存のクラスターのハイブリッドノード設定を追加、変更、削除できます。詳細については、「the section called “既存のクラスター”」を参照してください。
- ハイブリッドノード対応クラスターの作成時に Kubernetes サービス IPv4 CIDR を明示的に指定しない場合、Amazon EKS は設定されたリモートノードおよびポッドネットワークと重複しないサービス CIDR を自動的に選択します。割り当てられた CIDR は、標準のデフォルト (10.100.0.0/16 または 172.20.0.0/16) とは異なる場合があります。例えば、リモートノードまたはポッドネットワークがこれらのデフォルト範囲と重複する場合、Amazon EKS は 172.16.0.0/16 などのプライベート IP 範囲から代替 CIDR を選択します。予測可能な設定を確保するために、クラスターの作成中にサービス IPv4 CIDR を明示的に指定することをお勧めします。クラスターの作成後に aws eks describe-cluster --name CLUSTER_NAME --query “cluster.kubernetesNetworkConfig.serviceIpv4Cidr” を実行して、割り当てられたサービス CIDR を検索できます。

ステップ 1: クラスター IAM ロールを作成する

既にクラスター IAM ロールがある場合や、eksctl または AWS CloudFormation を使用してクラスターを作成する場合は、このステップはスキップできます。デフォルトでは、eksctl および AWS CloudFormation テンプレートによってクラスター IAM ロールが作成されます。

1. IAM 信頼ポリシー用の JSON ファイルを作成するには次のコマンドを実行してください。

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "eks.amazonaws.com"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
```

2. Amazon EKS クラスターの IAM ロールを作成します。必要であれば、前のステップでファイルを書き込んだコンピュータ上のパスを eks-cluster-role-trust-policy.json の前につけます。このコマンドは前のステップで作成した信頼ポリシーをロールに関連付けます。IAM ロールを作成するには、ロールを作成する IAM プリンシパル に iam:CreateRole アクション (許可) を割り当てる必要があります。

```
aws iam create-role \
    --role-name myAmazonEKSClusterRole \
    --assume-role-policy-document file://"eks-cluster-role-trust-policy.json"
```

3. Amazon EKS 管理のポリシーを割り当てるか、独自のカスタムポリシーを作成できます。カスタムポリシーで使用する必要がある最小限の許可については、「the section called “ノード の IAM ロール”」を参照してください。このロールに、Amazon EKS 管理の IAM ポリシー (AmazonEKSClusterPolicy) をアタッチします。IAM ポリシーを IAM プリンシパルにアタッチするには、ポリシーのアタッチを行っているプリンシパルに、次のいずれかの IAM アクション (許可) を割り当てる必要があります: iam:AttachUserPolicy または iam:AttachRolePolicy。

```
aws iam attach-role-policy \
    --policy-arn arn:aws:iam::aws:policy/AmazonEKSClusterPolicy \
    --role-name myAmazonEKSClusterRole
```

ステップ 2: ハイブリッドノード対応クラスターを作成する

以下を使用してクラスターを作成できます :

- eksctl
- AWS CloudFormation
- AWS CLI
- AWS マネジメントコンソール

ハイブリッドノード対応クラスターの作成 - eksctl

eksctl コマンドラインツールの最新バージョンをインストールする必要があります。eksctl をインストールまたはアップグレードするには、eksctl ドキュメントの「インストール」を参照してください。

1. cluster-config.yaml を作成して、ハイブリッドノード対応の Amazon EKS IPv4 クラスターを定義します。cluster-config.yaml で以下の置換を実行します。設定の完全なリストについては、「eksctl documentation」を参照してください。
   a. CLUSTER_NAME をクラスターの名前に置き換えます。この名前には英数字 (大文字と小文字が区別されます) とハイフンのみを使用できます。先頭の文字は英数字である必要があります。

また、100 文字より長くすることはできません。名前はクラスターを作成する AWS リージョンおよび AWS アカウント内で一意である必要があります。

b. AWS_REGION を、クラスターを作成する AWS リージョンに置き換えます。

c. K8S_VERSION を Amazon EKS がサポートする任意のバージョンに置き換えます。

d. the section called “認証情報を準備する” のステップで設定した認証情報プロバイダーに基づいて、CREDS_PROVIDER を ssm または ira に置き換えます。

e. 認証情報プロバイダーが ira に設定されている場合 (AWS IAM Roles Anywhere を認証情報プロバイダーとして使用)、CA_BUNDLE_CERT を置き換えます。CA_BUNDLE_CERT は認証機関 (CA) 証明書本文であり、CA の選択によって異なります。証明書はプライバシー強化メール (PEM) 形式である必要があります。

f. GATEWAY_ID を、VPC にアタッチする仮想プライベートゲートウェイまたはトランジットゲートウェイの ID に置き換えます。

g. REMOTE_NODE_CIDRS を、ハイブリッドノードのオンプレミスノード CIDR に置き換えます。

h. REMOTE_POD_CIDRS をハイブリッドノードで実行されているワークロードのオンプレミスポッド CIDR に置き換えるか、ハイブリッドノードでウェブフックを実行していない場合は設定から行を削除します。CNI がネットワークアドレス変換 (NAT) を使用しない場合や、ポッドトラフィックがオンプレミスホストから出るときにポッド IP アドレスをマスクする場合は、REMOTE_POD_CIDRS を設定する必要があります。ハイブリッドノードでウェブフックを実行している場合は、REMOTE_POD_CIDRS を設定する必要があります。詳細については、「the section called “ウェブフックの設定”」を参照してください。

i. オンプレミスノードとポッド CIDR ブロックは、以下の要件を満たしている必要があります。

   i. IPv4 RFC-1918 のいずれかの範囲内にある (10.0.0.0/8、172.16.0.0/12、または 192.168.0.0/16) か、RFC 6598 で定義される CGNAT 範囲内にある (100.64.0.0/10) こと。

   ii. 相互に、クラスターの VPC CIDR、または Kubernetes サービスの IPv4 CIDR と重複しないこと

```
apiVersion: eksctl.io/v1alpha5
kind: ClusterConfig

metadata:
  name: CLUSTER_NAME
  region: AWS_REGION
  version: "K8S_VERSION"

remoteNetworkConfig:
```

```
  iam:
    provider: CREDS_PROVIDER # default SSM, can also be set to IRA
    # caBundleCert: CA_BUNDLE_CERT
  vpcGatewayID: GATEWAY_ID
  remoteNodeNetworks:
  - cidrs: ["REMOTE_NODE_CIDRS"]
  remotePodNetworks:
  - cidrs: ["REMOTE_POD_CIDRS"]
```

2. 次のコマンドを実行してください：

```
eksctl create cluster -f cluster-config.yaml
```

クラスターのプロビジョニングには数分かかります。クラスターの作成中は数行の出力が表示されます。出力の最後の行は次のサンプル行のようになります。

```
[#]  EKS cluster "CLUSTER_NAME" in "REGION" region is ready
```

3. 「the section called “ステップ 3: kubeconfig を更新する”」に進みます。

ハイブリッドノード対応クラスターの作成 - AWS CloudFormation

CloudFormation スタックは、指定した RemotePodNetwork と RemoteNodeNetwork を使用して、EKS クラスター IAM ロールと EKS クラスターを作成します。CloudFormation テンプレートで公開されていない EKS クラスターの設定をカスタマイズする必要がある場合は、CloudFormation テンプレートを変更します。

1. CloudFormation テンプレートをダウンロードします。

```
curl -OL 'https://raw.githubusercontent.com/aws/eks-hybrid/refs/heads/main/example/
hybrid-eks-cfn.yaml'
```

2. cfn-eks-parameters.json を作成し、各値の設定を指定します。
   a. CLUSTER_NAME: 作成する EKS クラスターの名前。
   b. CLUSTER_ROLE_NAME: 作成する EKS クラスター IAM ロールの名前。テンプレートのデフォルトは「EKSClusterRole」です。
   c. SUBNET1_ID: 前提条件のステップで作成した最初のサブネットの ID
   d. SUBNET2_ID: 前提条件のステップで作成した 2 番目のサブネットの ID
   e. SG_ID: 前提条件のステップで作成したセキュリティグループの ID

f. REMOTE_NODE_CIDRS: ハイブリッドノードのオンプレミスノード CIDR

g. REMOTE_POD_CIDRS: ハイブリッドノードで実行されているワークロードのオンプレミスポッド CIDR CNI がネットワークアドレス変換 (NAT) を使用しない場合や、ポッドトラフィックがオンプレミスホストから出るときにポッド IP アドレスをマスクする場合は、REMOTE_POD_CIDRS を設定する必要があります。ハイブリッドノードでウェブフックを実行している場合は、REMOTE_POD_CIDRS を設定する必要があります。詳細については、「the section called “ウェブフックの設定”」を参照してください。

h. オンプレミスノードとポッド CIDR ブロックは、以下の要件を満たしている必要があります。

   i. IPv4 RFC-1918 のいずれかの範囲内にある (10.0.0.0/8、172.16.0.0/12、または 192.168.0.0/16) か、RFC 6598 で定義される CGNAT 範囲内にある (100.64.0.0/10) こと。

   ii. 相互に、クラスターの VPC CIDR、または Kubernetes サービスの IPv4 CIDR と重複しないこと。

i. CLUSTER_AUTH: クラスターのクラスター認証モード。有効な値は、API および API_AND_CONFIG_MAP です。テンプレートのデフォルトは API_AND_CONFIG_MAP です。

j. CLUSTER_ENDPOINT: クラスターのクラスターエンドポイント接続。有効な値は「Public」および「Private」です。テンプレートのデフォルトは Private です。つまり、Kubernetes API エンドポイントには VPC 内からしか接続できません。

k. K8S_VERSION: クラスターに使用する Kubernetes のバージョン。「サポートされている Amazon EKS バージョン」を参照してください。

```
{
  "Parameters": {
    "ClusterName": "CLUSTER_NAME",
    "ClusterRoleName": "CLUSTER_ROLE_NAME",
    "SubnetId1": "SUBNET1_ID",
    "SubnetId2": "SUBNET2_ID",
    "SecurityGroupId" "SG_ID",
    "RemoteNodeCIDR": "REMOTE_NODE_CIDRS",
    "RemotePodCIDR": "REMOTE_POD_CIDRS",
    "ClusterAuthMode": "CLUSTER_AUTH",
    "ClusterEndpointConnectivity": "CLUSTER_ENDPOINT",
    "K8sVersion": "K8S_VERSION"
  }
 }
```

3. CloudFormation スタックをデプロイします。STACK_NAME を CloudFormation スタックの名前に置き換え、AWS_REGION をクラスターを作成する AWS リージョンに置き換えます。

```
aws cloudformation deploy \
    --stack-name STACK_NAME \
    --region AWS_REGION \
    --template-file hybrid-eks-cfn.yaml \
    --parameter-overrides file://cfn-eks-parameters.json \
    --capabilities CAPABILITY_NAMED_IAM
```

クラスターのプロビジョニングには数分かかります。以下のコマンドを使用してスタックの状態を確認できます。STACK_NAME を CloudFormation スタックの名前に置き換え、AWS_REGION をクラスターを作成する AWS リージョンに置き換えます。

```
aws cloudformation describe-stacks \
    --stack-name STACK_NAME \
    --region AWS_REGION \
    --query 'Stacks[].StackStatus'
```

4. 「the section called “ステップ 3: kubeconfig を更新する”」に進みます。

ハイブリッドノード対応クラスターの作成 - AWS CLI

1. 以下のコマンドを実行して、ハイブリッドノード対応の EKS クラスターを作成します。コマンドを実行する前に、以下を自分の設定に置き換えます。設定の完全なリストについては、the section called “クラスターを作成する” のドキュメントを参照してください。
   a. CLUSTER_NAME: 作成する EKS クラスターの名前。
   b. AWS_REGION: クラスターを作成する AWS リージョン。
   c. K8S_VERSION: クラスターに使用する Kubernetes のバージョン。「サポートされている Amazon EKS バージョン」を参照してください。
   d. ROLE_ARN: クラスター用に設定した Amazon EKS クラスターロール。詳細については、「Amazon EKS クラスターの IAM ロール」を参照してください。
   e. SUBNET1_ID: 前提条件のステップで作成した最初のサブネットの ID
   f. SUBNET2_ID: 前提条件のステップで作成した 2 番目のサブネットの ID
   g. SG_ID: 前提条件のステップで作成したセキュリティグループの ID
   h. クラスターアクセス認証モードには API と API_AND_CONFIG_MAP を使用できます。以下のコマンドでは、クラスターアクセス認証モードは API_AND_CONFIG_MAP に設定されています。

i. endpointPrivateAccess パラメータ と endpointPublicAccess パラメータを使用して、クラスターの Kubernetes API サーバーエンドポイントへのパブリックとプライベートアクセスを有効または無効にすることができます。以下のコマンドでは、endpointPublicAccess は false に設定され、endpointPrivateAccess は true に設定されています。

j. REMOTE_NODE_CIDRS: ハイブリッドノードのオンプレミスノード CIDR。

k. REMOTE_POD_CIDRS (オプション): ハイブリッドノードで実行されているワークロードのオンプレミスポッド CIDR

l. オンプレミスノードとポッド CIDR ブロックは、以下の要件を満たしている必要があります。

i. IPv4 RFC-1918 のいずれかの範囲内にある (10.0.0.0/8、172.16.0.0/12、または 192.168.0.0/16) か、RFC 6598 で定義される CGNAT 範囲内にある (100.64.0.0/10) こと。

ii. 相互に、Amazon EKS クラスターの VPC CIDR、または Kubernetes サービスの IPv4 CIDR と重複しないこと。

```
aws eks create-cluster \
    --name CLUSTER_NAME \
    --region AWS_REGION \
    --kubernetes-version K8S_VERSION \
    --role-arn ROLE_ARN \
    --resources-vpc-config
 subnetIds=SUBNET1_ID,SUBNET2_ID,securityGroupIds=SG_ID,endpointPrivateAccess=true,endpo
 \
    --access-config authenticationMode=API_AND_CONFIG_MAP \
    --remote-network-config '{"remoteNodeNetworks":[{"cidrs":
["REMOTE_NODE_CIDRS"]}],"remotePodNetworks":[{"cidrs":["REMOTE_POD_CIDRS"]}]}'
```

2. クラスターがプロビジョニングされるまでに数分かかります。クラスターのステータスのクエリを実行するには次のコマンドを使用します。CLUSTER_NAME を、作成しているクラスターの名前に置き換え、AWS_REGION を、クラスターを作成している AWS リージョンに置き換えます。出力が「ACTIVE」を返すまで、次のステップに進まないでください。

```
aws eks describe-cluster \
    --name CLUSTER_NAME \
    --region AWS_REGION \
    --query "cluster.status"
```

3. 「the section called “ステップ 3: kubeconfig を更新する”」に進みます。

ハイブリッドノード対応クラスターの作成 - AWS マネジメントコンソール

1. Amazon EKS コンソールで Amazon EKS コンソールを開きます。
2. [クラスターを追加]、[作成] の順にクリックします。
3. [クラスターの設定] ページで、次のフィールドに入力します。
   a. [名前] - クラスターの名前。この名前には英数字 (大文字と小文字が区別されます)、ハイフン、下線のみを使用できます。先頭の文字は英数字である必要があります。また、100 文字より長くすることはできません。名前はクラスターを作成する AWS リージョンおよび AWS アカウント内で一意である必要があります。
   b. [クラスター IAM ロール] – ユーザーに代わって AWS リソースを管理することを Kubernetes コントロールプレーンに許可するために作成した Amazon EKS クラスター IAM ロールを選択します。
   c. [Kubernetes バージョン] – クラスターで使用する Kubernetes のバージョン。以前のバージョンが必要でない限り、最新バージョンを選択することをお勧めします。
   d. [アップグレードポリシー] — 拡張または標準を選択します。
      i. 拡張: このオプションは、リリース日から 26 か月間 Kubernetes バージョンをサポートします。延長サポート期間には、標準サポート期間終了後に開始される追加の時間単位のコストがかかります。延長サポートが終了すると、クラスターは次のバージョンに自動的にアップグレードされます。
      ii. 標準: このオプションは、リリース日から 14 か月間 Kubernetes バージョンをサポートします。追加コストはかかりません。標準サポートが終了すると、クラスターは次のバージョンに自動的にアップグレードされます。
   e. クラスターアクセス - クラスター管理者アクセスを許可または禁止し、認証モードを選択します。ハイブリッドノードが有効なクラスターでは、以下の認証モードがサポートされます。
      i. EKS API: クラスターは、認証された IAM プリンシパルを EKS アクセスエントリ API からのみ取得します。
      ii. EKS API と ConfigMap: クラスターは、EKS アクセスエントリ API と `aws-auth` ConfigMap の両方から認証された IAM プリンシパルを取得します。
   f. [Secret encryption] (シークレット暗号化) – (オプション) KMS キーを使用して Kubernetes シークレットのシークレット暗号化を有効にするよう選択します。クラスターを作成した後で、これを有効にすることもできます。この機能を有効にする前に、the section called “シークレット暗号化を有効にする” の情報をよく理解していることを確認してください。

g. ARC ゾーンシフト - 有効にすると、EKS はクラスターを ARC ゾーンシフトに登録し、ゾーンシフトを使用してアプリケーショントラフィックを AZ から遠ざけることができるようになります。

h. [タグ] - (オプション) クラスターに任意のタグを追加します。詳細については、the section called “リソースのタグ付け” を参照してください。

i. このページを読み終えたら、[次へ] を選択してください。

4. [ネットワーキングの指定] ページで、次のフィールドの値を選択してください：

a. VPC – the section called “VPC およびサブネットの要件” および Amazon EKS Hybrid Nodes の要件を満たす既存の VPC を選択します。VPC を選択する前に、「VPC、サブネット、ハイブリッドノードの Amazon EKS ネットワーキング要件を表示する」の要件と考慮事項をすべて理解しておくことをお勧めします。クラスターの作成後は使用する VPC を変更できません。VPC が表示されていない場合はまず作成する必要があります。詳細については、「the section called “「VPC を作成する」”」および「Amazon EKS Hybrid Nodes のネットワーク要件」を参照してください。

b. [サブネット] - デフォルトで、前のフィールドで指定した VPC 内の利用可能なすべてのサブネットがあらかじめ選択されています。少なくとも 2 つ選択する必要があります。

c. [セキュリティグループ] - (オプション Amazon EKS が作成するネットワークインターフェイスに関連付ける 1 つまたは複数のセキュリティグループを指定します。指定するセキュリティグループの少なくとも 1 つには、オンプレミスのノードとオプションで ポッド CIDR に対するインバウンドルールが必要です。詳細については、「Amazon EKS Hybrid Nodes のネットワーク要件」を参照してください。セキュリティグループを選択するかどうかにかかわらず、Amazon EKS はクラスターと VPC 間の通信を可能にするセキュリティグループを作成します。Amazon EKS はこのセキュリティグループおよびユーザーが選択したセキュリティグループを、作成するネットワークインターフェイスに関連付けます。Amazon EKS が作成するクラスターセキュリティグループの詳細については、「the section called “セキュリティグループの要件”」を参照してください。Amazon EKS が作成するクラスターセキュリティグループのルールは変更できます。

d. クラスター IP アドレスファミリーの選択 — ハイブリッドノード対応クラスターには IPv4 を選択する必要があります。

e. (オプション) [Kubernetes サービス IP アドレスの範囲を設定する] を選択し、[サービスの IPv4 範囲] を指定します。ハイブリッドノードが有効なクラスターには、明示的なサービス CIDR を指定することをお勧めします。指定しない場合、Amazon EKS は、リモートノードおよびポッドネットワークとの競合を回避する CIDR を選択しますが、これは標準のデフォルト (`10.100.0.0/16` または `172.20.0.0/16`) とは異なる場合があります。

f. [リモートネットワークを設定してハイブリッドノードを有効にする] を選択し、ハイブリッドノードのオンプレミスノードとポッド CIDR を指定します。

g. ポッドトラフィックがオンプレミスホストを離れる際、CNI がポッド IP アドレスに対してネットワークアドレス変換 (NAT) またはマスカレードを使用しない場合は、リモートポッド CIDR を設定する必要があります。ハイブリッドノードでウェブフックを実行している場合は、リモートポッド CIDR を設定する必要があります。

h. オンプレミスノードとポッド CIDR ブロックは、以下の要件を満たしている必要があります。

i. IPv4 RFC-1918 のいずれかの範囲内にある (`10.0.0.0/8`、`172.16.0.0/12`、または `192.168.0.0/16`) か、RFC 6598 で定義される CGNAT 範囲内にある (`100.64.0.0/10`) こと。

ii. 相互に、クラスターの `VPC CIDR`、または Kubernetes サービスの IPv4 CIDR と重複しないこと

i. [クラスターエンドポイントのアクセス] で、オプションを選択してください。クラスターを作成した後で、このオプションを変更できます。ハイブリッドノードが有効なクラスターの場合は、パブリックまたはプライベートのいずれかを選択する必要があります。デフォルト以外のオプションを選択する前に、オプションとその意味を理解しておいてください。詳細については、the section called “クラスターエンドポイントのアクセス” を参照してください。

j. このページを読み終えたら、[次へ] を選択してください。

5. (オプション) [オブザーバビリティの設定] ページで、有効にする [メトリクス] と [コントロールプレーンのロギング] オプションを選択します。デフォルトではそれぞれのログタイプは無効化されています。

a. Prometheus メトリクスの詳細については、「the section called “Prometheus メトリクス”」を参照してください。

b. EKS コントロールのログ記録のオプションの詳細については、「the section called “コントロールプレーンのログ”」を参照してください。

c. このページを読み終えたら、[次へ] を選択してください。

6. [アドオンの選択] ページで、クラスターに追加するアドオンを選択してください。

a. [Amazon EKS アドオン] と [AWS マーケットプレイス アドオン] は必要な数だけ選択できます。ハイブリッドノードと互換性のない Amazon EKS アドオンは「ハイブリッドノードと互換性がない」とマークされ、これらのアドオンには、ハイブリッドノードで実行されないようにするためのアンチアフィニティルールが設定されます。詳細については、「ハイブリッドノードのアドオンの設定」を参照してください。インストールする [AWS Marketplace アドオン] が一覧にない場合は、検索ボックスにテキストを入力して、利用可能な [AWS Marketplace アド

オン] を検索できます。[カテゴリ]、[ベンダー]、または [価格モデル] で検索し、検索結果からアドオンを選択することもできます。

b. CoreDNS や kube-proxy などの一部のアドオンは、デフォルトでインストールされます。デフォルトのアドオンのいずれかを無効にすると、Kubernetes アプリケーションを実行する機能に影響する場合があります。

c. このページを読み終えたら、[Next] を選択してください。

7. [選択したアドオン設定の構成] ページで、インストールするバージョンを選択し、[次へ] を選択してください。

a. クラスターを作成した後はいつでも新しいバージョンに更新できます。クラスターの作成後に、各アドオンの設定を更新できます。アドオンの設定の詳細については「the section called “アドオンを更新する”」を参照してください。ハイブリッドノードと互換性のあるアドオンのバージョンについては、「the section called “CNI アドオンの設定”」を参照してください。

b. このページを読み終えたら、[次へ] を選択してください。

8. [確認と作成] ページで、前のページで入力または選択した情報を確認します。変更する必要がある場合は[編集] を選択してください。そのままでよければ、[作成] を選択してください。クラスターがプロビジョニングされている間、[ステータス] フィールドに [作成中] と表示されます。クラスターのプロビジョニングには数分かかります。

9. 「the section called “ステップ 3: kubeconfig を更新する”」に進みます。

ステップ 3: kubeconfig を更新する

eksctl を使用してクラスターを作成した場合、このステップはスキップできます。eksctl によってこのステップは既に完了しているからです。新しいコンテキストを kubectl 設定ファイルに追加して、kubectl がクラスターと通信できるようにします。ファイルを作成および更新する方法の詳細については「the section called “kubectl を使用してクラスターにアクセスする”」を参照してください。

```
aws eks update-kubeconfig --name CLUSTER_NAME --region AWS_REGION
```

出力例は次のとおりです。

```
Added new context arn:aws:eks:AWS_REGION:111122223333:cluster/CLUSTER_NAME to /home/
username/.kube/config
```

次のコマンドを実行して、クラスターとの通信を確認します。

```
kubectl get svc
```

出力例は次のとおりです。

```
NAME         TYPE        CLUSTER-IP   EXTERNAL-IP   PORT(S)   AGE
kubernetes   ClusterIP   10.100.0.1   <none>        443/TCP   28h
```

ステップ 4: クラスターのセットアップ

次のステップとして、ハイブリッドノードをクラスターに参加させるためにアクセスを有効にする手順は、「the section called “クラスターアクセスを準備する”」を参照してください。

既存の Amazon EKS クラスターでハイブリッドノードを有効にするか、設定を変更する

このトピックでは、使用可能なオプションの概要と、Amazon EKS クラスターのハイブリッドノード設定を追加、変更、削除する際の考慮事項について説明します。

Amazon EKS クラスターでハイブリッドノードを使用できるようにするには、オンプレミスノードとポッドネットワーク (オプション) の IP アドレス CIDR 範囲を `RemoteNetworkConfig` 設定に追加します。EKS では、この CIDR リストを使用して、クラスターとオンプレミスネットワーク間の接続を有効にします。クラスター設定を更新する際のオプションについては、「Amazon EKS API リファレンス」の「UpdateClusterConfig」でリスト全体を確認できます。

クラスター内にある EKS Hybrid Nodes のネットワーク設定では、次のいずれかのアクションを実行できます。

- リモートネットワーク設定を追加して、既存のクラスターで EKS Hybrid Nodes を有効にする。
- 既存のクラスター内のリモートノードネットワークまたはリモートポッドネットワークを追加、変更、削除する。
- すべてのリモートノードネットワーク CIDR 範囲を削除して、既存のクラスターの EKS Hybrid Nodes を無効にする。

前提条件

- ハイブリッドノードの Amazon EKS クラスターを有効にする前に、環境が次の要件を満たしていることを確認してください: 「the section called “前提条件”」で概説されている要件に加え、「the

section called “ネットワークを準備する”」、「the section called “オペレーティングシステムを準備する”」、「the section called “認証情報を準備する”」で詳述されている要件

- クラスターは IPv4 アドレスファミリーを使用する必要があります。
- クラスターは、クラスター認証モードに `API` または `API_AND_CONFIG_MAP` を使用する必要があります。クラスター認証モードを変更するプロセスについては、「the section called “[Authentication mode] (認証モード)”」を参照してください。
- Amazon EKS Kubernetes API サーバーエンドポイントには、パブリックまたはプライベートいずれか (両方ではなく) のエンドポイントアクセスを使用することが推奨されます。「Public and Private (パブリックとプライベート)」を選択した場合、Amazon EKS Kubernetes API サーバーエンドポイントは、常に VPC の外部で稼働しているハイブリッドノードのパブリック IP に解決されるため、ハイブリッドノードがクラスターに参加できなくなる可能性があります。クラスターへのネットワークアクセスを変更するプロセスについては、「the section called “クラスターエンドポイントのアクセス”」を参照してください。
- AWS コマンドラインインターフェイス (AWS CLI) の最新バージョンがデバイスにインストールおよび設定されていること。現在のバージョンを確認するには、`aws --version` を参照してください。yum、apt-get、macOS 用の Homebrew などのパッケージマネージャーは、多くの場合 AWS CLI の最新バージョンより数バージョン古くなっています。最新バージョンをインストールするには、「AWS Command Line Interface ユーザーガイド」の「AWS CLI の最新バージョンのインストールまたは更新」および「AWS CLI の設定を構成する」を参照してください。
- Amazon EKS クラスターで UpdateClusterConfig を呼び出す権限を持つ IAM プリンシパル。
- ハイブリッドノードと互換性のあるバージョンにアドオンを更新します。ハイブリッドノードと互換性のあるアドオンのバージョンについては、「the section called “CNI アドオンの設定”」を参照してください。
- ハイブリッドノードと互換性のないアドオンを実行している場合は、ハイブリッドノードへのデプロイを防ぐために、アドオンの DaemonSet または Deployment に、次のアフィニティルールが設定されていることを確認します。次のアフィニティルールが存在しない場合は追加してください。

```
affinity:
  nodeAffinity:
    requiredDuringSchedulingIgnoredDuringExecution:
      nodeSelectorTerms:
      - matchExpressions:
        - key: eks.amazonaws.com/compute-type
          operator: NotIn
          values:
          - hybrid
```

考慮事項

`remoteNetworkConfig` JSON オブジェクトは、更新中に次のように動作します。

- 指定していない既存の設定部分は変更されません。`remoteNodeNetworks` または `remotePodNetworks` のいずれかを指定していない場合、該当箇所への変更は行われません。
- `remoteNodeNetworks` または `remotePodNetworks` の CIDR リストを変更する場合は、最終設定に必要な CIDR の完全なリストを指定しなければなりません。`remoteNodeNetworks` または `remotePodNetworks` の CIDR リストのいずれかに変更を指定すると、更新中に元のリストが置き換えられます。
- オンプレミスノードとポッド CIDR ブロックは、以下の要件を満たしている必要があります。
  1. IPv4 RFC-1918 のいずれかの範囲内にある (10.0.0.0/8、172.16.0.0/12、または 192.168.0.0/16) か、RFC 6598 で定義される CGNAT 範囲内にある (`100.64.0.0/10`) こと。
  2. Amazon EKS クラスターの VPC の全 CIDR、または Kubernetes サービスの IPv4 CIDR と相互に重複していないこと。

既存のクラスターでハイブリッドノードを有効にする

既存のクラスターで EKS Hybrid Nodes を有効にするには、以下を使用します。

- AWS CloudFormation
- AWS CLI
- AWS マネジメントコンソール

既存のクラスターで EKS Hybrid Nodes を有効にする - AWS CloudFormation

1. クラスターで EKS Hybrid Nodes を有効にするには、`RemoteNodeNetwork` と `RemotePodNetwork` (オプション) を CloudFormation テンプレートに追加して、スタックを更新します。`RemoteNodeNetwork` は最大 1 つの `Cidrs` 項目が含まれるリストであり、`Cidrs` は複数の IP CIDR 範囲が含まれるリストであることに注意してください。

```
RemoteNetworkConfig:
  RemoteNodeNetworks:
    - Cidrs: [RemoteNodeCIDR]
  RemotePodNetworks:
    - Cidrs: [RemotePodCIDR]
```

2. 「the section called “クラスターアクセスを準備する”」に進みます。

既存のクラスターで EKS Hybrid Nodes を有効にする - AWS CLI

1. EKS クラスターの EKS Hybrid Nodes に使用する RemoteNetworkConfig を有効にするには、以下のコマンドを実行します。コマンドを実行する前に、以下を自分の設定に置き換えます。全設定のリストについては、「Amazon EKS API リファレンス」の「UpdateClusterConfig」を参照してください。
   a. CLUSTER_NAME: 更新する EKS クラスターの名前。
   b. AWS_REGION: EKS クラスターが実行されている AWS リージョン。
   c. REMOTE_NODE_CIDRS: ハイブリッドノードのオンプレミスノード CIDR。
   d. REMOTE_POD_CIDRS (オプション): ハイブリッドノードで実行されているワークロードのオンプレミスポッド CIDR

```
aws eks update-cluster-config \
    --name CLUSTER_NAME \
    --region AWS_REGION \
    --remote-network-config '{"remoteNodeNetworks":[{"cidrs":
["REMOTE_NODE_CIDRS"]}],"remotePodNetworks":[{"cidrs":["REMOTE_POD_CIDRS"]}]}'
```

2. クラスターの更新には数分かかります。クラスターのステータスのクエリを実行するには次のコマンドを使用します。CLUSTER_NAME は変更するクラスターの名前に、AWS_REGION はクラスターが稼働している AWS リージョンに置き換えてください。出力が「ACTIVE」を返すまで、次のステップに進まないでください。

```
aws eks describe-cluster \
    --name CLUSTER_NAME \
    --region AWS_REGION \
    --query "cluster.status"
```

3. 「the section called “クラスターアクセスを準備する”」に進みます。

既存のクラスターで EKS Hybrid Nodes を有効にする - AWS マネジメントコンソール

1. Amazon EKS コンソールで Amazon EKS コンソールを開きます。
2. クラスター名を選択すると、そのクラスターの情報を表示されます。
3. [ネットワーキング] タブを開き、[管理] 選択します。
4. ドロップダウンで、[リモートネットワーク] を選択します。

5. [リモートネットワークを設定してハイブリッドノードを有効にする] を選択し、ハイブリッドノードのオンプレミスノードとポッド CIDR を指定します。
6. [変更を保存] を選択して終了します。クラスターのステータスが [アクティブ] に戻るまで待ちます。
7. 「the section called “クラスターアクセスを準備する”」に進みます。

既存のクラスターでハイブリッドノード設定を更新する

既存のハイブリッドクラスターで `remoteNetworkConfig` を変更するには、以下のいずれかを使用します。

- AWS CloudFormation
- AWS CLI
- AWS マネジメントコンソール

既存のクラスターでハイブリッド設定を更新する - AWS CloudFormation

1. CloudFormation テンプレートを新しいネットワーク CIDR 値で更新します。

```
RemoteNetworkConfig:
  RemoteNodeNetworks:
    - Cidrs: [NEW_REMOTE_NODE_CIDRS]
  RemotePodNetworks:
    - Cidrs: [NEW_REMOTE_POD_CIDRS]
```

> **Note**
>
> `RemoteNodeNetworks` または `RemotePodNetworks` の CIDR リストを更新するときは、すべての CIDR (新規と既存) を含めます。更新中には、リスト全体が置き換えられます。更新リクエストでこれらのフィールドを省略すると、既存の設定が保持されます。

2. 変更したテンプレートで CloudFormation スタックを更新し、スタックの更新が完了するまで待ちます。

既存のクラスターでハイブリッド設定を更新する - AWS CLI

1. リモートネットワークの CIDR を変更するには、次のコマンドを実行します。値をご自身の設定に置き換えます。

```
aws eks update-cluster-config
--name CLUSTER_NAME
--region AWS_REGION
--remote-network-config '{"remoteNodeNetworks":[{"cidrs":
["NEW_REMOTE_NODE_CIDRS"]}],"remotePodNetworks":[{"cidrs":
["NEW_REMOTE_POD_CIDRS"]}]}'
```

> **Note**
>
> `remoteNodeNetworks` または `remotePodNetworks` の CIDR リストを更新するときは、すべての CIDR (新規と既存) を含めます。更新中には、リスト全体が置き換えられます。更新リクエストでこれらのフィールドを省略すると、既存の設定が保持されます。

2. クラスターのステータスが [アクティブ] に戻るまで待ち、次に進みます。

既存のクラスターでハイブリッド設定を更新する - AWS マネジメントコンソール

1. Amazon EKS コンソールで Amazon EKS コンソールを開きます。
2. クラスター名を選択すると、そのクラスターの情報を表示されます。
3. [ネットワーキング] タブを開き、[管理] 選択します。
4. ドロップダウンで、[リモートネットワーク] を選択します。
5. 必要に応じて、`Remote node networks` と `Remote pod networks - Optional` の CIDR を更新します。
6. [変更の保存] を選択し、クラスターのステータスが [アクティブ] に戻るのを待ちます。

既存のクラスターで Hybrid Nodes を無効にする

既存のクラスターで EKS Hybrid Nodes を無効にするには、以下を使用します。

- AWS CloudFormation
- AWS CLI
- AWS マネジメントコンソール

既存のクラスターで EKS Hybrid Nodes を無効にする - AWS CloudFormation

1. クラスターで EKS Hybrid Nodes を無効にするには、RemoteNodeNetworks と RemotePodNetworks を CloudFormation テンプレートの空の配列に設定して、スタックを更新します。

```
RemoteNetworkConfig:
  RemoteNodeNetworks: []
  RemotePodNetworks: []
```

既存のクラスターで EKS Hybrid Nodes を無効にする - AWS CLI

1. クラスターから RemoteNetworkConfig を削除するには、以下のコマンドを実行します。コマンドを実行する前に、以下を自分の設定に置き換えます。全設定のリストについては、「Amazon EKS API リファレンス」の「UpdateClusterConfig」を参照してください。
   a. CLUSTER_NAME: 更新する EKS クラスターの名前。
   b. AWS_REGION: EKS クラスターが実行されている AWS リージョン。

```
aws eks update-cluster-config \
    --name CLUSTER_NAME \
    --region AWS_REGION \
    --remote-network-config '{"remoteNodeNetworks":[],"remotePodNetworks":[]}'
```

2. クラスターの更新には数分かかります。クラスターのステータスのクエリを実行するには次のコマンドを使用します。CLUSTER_NAME は変更するクラスターの名前に、AWS_REGION はクラスターが稼働している AWS リージョンに置き換えてください。出力が「ACTIVE」を返すまで、次のステップに進まないでください。

```
aws eks describe-cluster \
    --name CLUSTER_NAME \
    --region AWS_REGION \
    --query "cluster.status"
```

既存のクラスターで EKS Hybrid Nodes を無効にする - AWS マネジメントコンソール

1. Amazon EKS コンソールで Amazon EKS コンソールを開きます。
2. クラスター名を選択すると、そのクラスターの情報を表示されます。

3. [ネットワーキング] タブを開き、[管理] 選択します。
4. ドロップダウンで、[リモートネットワーク] を選択します。
5. [リモートネットワークを設定してハイブリッドノードを有効にする] を選択して、`Remote node networks` と `Remote pod networks - Optional` のすべての CIDR を削除します。
6. [変更を保存] を選択して終了します。クラスターのステータスが [アクティブ] に戻るまで待ちます。

ハイブリッドノードのクラスターアクセスを準備する

ハイブリッドノードを Amazon EKS クラスターに接続するには、クラスターに参加するために Kubernetes のアクセス許可を持つハイブリッドノードの IAM ロールを有効にする必要があります。ハイブリッドノードの IAM ロールの作成方法については、「the section called “認証情報を準備する”」を参照してください。Amazon EKS は、IAM プリンシパルを Kubernetes ロールベースアクセスコントロール (RBAC) に関連付ける方法として、Amazon EKS アクセスエントリと `aws-auth` ConfigMap の 2 つをサポートしています。Amazon EKS アクセス管理の詳細については、「the section called “Kubernetes API アクセス”」を参照してください。

以下の手順を使用して、ハイブリッドノードの IAM ロールを Kubernetes のアクセス許可に関連付けます。Amazon EKS のアクセスエントリを使用するには、クラスターが `API` または `API_AND_CONFIG_MAP` 認証モードで作成されている必要があります。`aws-auth` ConfigMap を使用するには、クラスターが `API_AND_CONFIG_MAP` 認証モードで作成されている必要があります。`CONFIG_MAP` のみの認証モードは、ハイブリッドノードが有効な Amazon EKS クラスターではサポートされていません。

ハイブリッドノードの IAM ロールに Amazon EKS のアクセスエントリを使用する

IAM ロールと組み合わせて使用できる、HYBRID_LINUX というハイブリッドノード用 Amazon EKS アクセスエントリタイプがあります。このアクセスエントリタイプでは、ユーザー名は自動的に system:node:{{SessionName}} に設定されます。アクセスエントリの作成の詳細については、「the section called “アクセスエントリを作成する”」を参照してください。

AWS CLI

1. デバイスに最新バージョンの AWS CLI をインストールし、設定しておく必要があります。現在のバージョンを確認するには、`aws --version` を参照してください。yum、apt-get、macOS 用の Homebrew などのパッケージマネージャーは、多くの場合 AWS CLI の最新バージョンより数バージョン古くなっています。最新バージョンをインストールするには、「AWS コマンドラインイン

ターフェイスユーザーガイド」の「インストール」および「aws configure を使用したクイック設定」を参照してください。

2. 以下のコマンドを使用してアクセスエントリーを作成します。CLUSTER_NAME をお使いのクラスターの名前に、HYBRID_NODES_ROLE_ARN を the section called “認証情報を準備する” のステップで作成したロールの ARN に置き換えます。

```
aws eks create-access-entry --cluster-name CLUSTER_NAME \
    --principal-arn HYBRID_NODES_ROLE_ARN \
    --type HYBRID_LINUX
```

AWS マネジメントコンソール

1. Amazon EKS コンソールで Amazon EKS コンソールを開きます。
2. ハイブリッドノードが有効なクラスターの名前を選択します。
3. [リモートアクセス] タブを選択してください。
4. [アクセスエントリの作成] を選択してください。
5. [IAM プリンシパル] には、「the section called “認証情報を準備する”」のステップで作成したハイブリッドノードの IAM ロールを選択します。
6. [タイプ] には Hybrid Linux を選択します。
7. (オプション) [タグ] ではアクセスエントリにラベルを割り当てます。例えば、同じタグを持つすべてのリソースを検索しやすくするためです。
8. [レビューと作成にスキップ] を選択します。Hybrid Linux のアクセスエントリにポリシーを追加することや、アクセス範囲を変更することはできません。
9. アクセスエントリの設定を確認してください。何かが正しくないと思われる場合は[戻る] を選択してステップに戻り、エラーを修正します。設定が正しければ、[作成] を選択してください。

ハイブリッドノードの IAM ロールに aws-auth ConfigMap を使用する

以下のステップでは、the section called “認証情報を準備する” のステップで作成したハイブリッドノード IAM ロールの ARN を使用して、`aws-auth` ConfigMap を作成または更新します。

1. クラスターに既存の `aws-auth` ConfigMap があるかどうかを確認します。特定の `kubeconfig` ファイルを使用している場合は、`--kubeconfig` フラグを使用してください。

```
kubectl describe configmap -n kube-system aws-auth
```

2. aws-auth ConfigMap が表示されている場合は、必要に応じて更新してください。

   a. ConfigMap を編集用に開きます。

   ```
   kubectl edit -n kube-system configmap/aws-auth
   ```

   b. 必要に応じて新しい mapRoles エントリを追加します。HYBRID_NODES_ROLE_ARN をハイブリッドノード IAM ロールの ARN に置き換えます。なお、{{SessionName}} は ConfigMap に保存するための正しいテンプレート形式です。他の値には置き換えないでください。

   ```
   data:
     mapRoles: |
     - groups:
       - system:bootstrappers
       - system:nodes
       rolearn: HYBRID_NODES_ROLE_ARN
       username: system:node:{{SessionName}}
   ```

   c. ファイルを保存し、テキストエディタを終了します。

3. クラスターに既存の aws-auth ConfigMap がない場合は、以下のコマンドを使用して作成します。HYBRID_NODES_ROLE_ARN をハイブリッドノード IAM ロールの ARN に置き換えます。なお、{{SessionName}} は ConfigMap に保存するための正しいテンプレート形式です。他の値には置き換えないでください。

   ```
   kubectl apply -f=/dev/stdin <<-EOF
   apiVersion: v1
   kind: ConfigMap
   metadata:
     name: aws-auth
     namespace: kube-system
   data:
     mapRoles: |
     - groups:
       - system:bootstrappers
       - system:nodes
       rolearn: HYBRID_NODES_ROLE_ARN
       username: system:node:{{SessionName}}
   EOF
   ```

ハイブリッドノードでオンプレミスのワークロードを実行する

ハイブリッドノードが有効になっている EKS クラスターではAWS クラウドで使用するのと同じ Amazon EKS クラスター、機能、ツールを使用して、独自のインフラストラクチャでオンプレミスおよびエッジアプリケーションを実行できます。

以下のセクションではハイブリッドノードを使用するためのステップバイステップの手順について説明します。

トピック

- ハイブリッドノードを接続する
- Bottlerocket を実行しているハイブリッドノードの接続
- クラスターのハイブリッドノードをアップグレードする
- ハイブリッドノード用のセキュリティ更新プログラムにパッチを適用する
- ハイブリッドノードを削除する

ハイブリッドノードを接続する

> **Note**
>
> 次の手順は、互換性のあるオペレーティングシステム (Bottlerocket を除く) を実行しているハイブリッドノードに適用されます。Bottlerocket を実行するハイブリッドノードを接続する手順については、「the section called “Bottlerocket を実行しているハイブリッドノードの接続”」を参照してください。

このトピックでは、ハイブリッドノードを Amazon EKS クラスターに接続する方法について説明します。ハイブリッドノードがクラスターに参加すると、Amazon EKS コンソールと kubectl などの Kubernetes 互換ツールに準備中ステータスが表示されます。このページのステップを完了したら、「the section called “CNI を設定する”」に進み、ハイブリッドノードでアプリケーションを実行する準備をします。

前提条件

ハイブリッドノードを Amazon EKS クラスターに接続する前に、前提条件の手順が完了していることを確認してください。

- オンプレミス環境から Amazon EKS クラスターをホストする AWS リージョンへのネットワーク接続があること。詳細については「the section called “ネットワークを準備する”」を参照してください。
- ハイブリッドノード用の互換性のあるオペレーティングシステムがオンプレミスホストにインストールされていること。詳細については「the section called “オペレーティングシステムを準備する”」を参照してください。
- ハイブリッドノードの IAM ロールを作成し、オンプレミス認証情報プロバイダー (AWS Systems Manager ハイブリッドアクティベーションまたは AWS IAM Roles Anywhere) をセットアップしていること。詳細については「the section called “認証情報を準備する”」を参照してください。
- ハイブリッドノード対応の Amazon EKS クラスターを作成していること。詳細については「the section called “クラスターを作成する”」を参照してください。
- ハイブリッドノードの IAM ロールを、Kubernetes ロールベースのアクセスコントロール (RBAC) のアクセス許可に関連付けていること。詳細については「the section called “クラスターアクセスを準備する”」を参照してください。

⚠ SSM 認証情報プロバイダーに必要な nodeadm バージョン

ハイブリッドノードの認証情報プロバイダーとして AWS Systems Manager (SSM) を使用する場合は、新しいインストールとアップグレードに nodeadm バージョン `1.0.19` 以降を使用する必要があります。以前のバージョンの nodeadm には古い SSM 署名キーが含まれており、`nodeadm install` と `nodeadm upgrade` の実行中に失敗し、次の署名検証エラーが発生します:

```
"msg":"Command failed","error":"failed to install ssm installer: validating ssm-setup-cli signature: Signature Verification Error: No matching signature"
```

このエラーを解決するには、`nodeadm install` または `nodeadm upgrade` を実行する前に nodeadm の最新バージョンをダウンロードします。

ステップ 1: ハイブリッドノード CLI (**nodeadm**) を各オンプレミスホストにインストールする

構築済みのオペレーティングシステムイメージに Amazon EKS Hybrid Nodes CLI (nodeadm) を含める場合は、このステップをスキップできます。nodeadm のハイブリッドノード版の詳細については、「the section called “ハイブリッドノード用 nodeadm”」を参照してください。

nodeadm のハイブリッドノード版は、Amazon CloudFront をフロントに置いた Amazon S3 でホストされます。各オンプレミスホストに nodeadm をインストールするにはオンプレミスホストから以下のコマンドを実行してください。

x86_64 ホストの場合:

```
curl -OL 'https://hybrid-assets.eks.amazonaws.com/releases/latest/bin/linux/amd64/
nodeadm'
```

ARM ホストの場合

```
curl -OL 'https://hybrid-assets.eks.amazonaws.com/releases/latest/bin/linux/arm64/
nodeadm'
```

各ホストでダウンロードしたバイナリに実行可能ファイルのアクセス許可を追加します。

```
chmod +x nodeadm
```

ステップ 2: **nodeadm** を使用してハイブリッドノードの依存関係をインストールする

構築済みのオペレーティングシステムイメージにハイブリッドノードの依存関係をインストールする場合は、このステップをスキップできます。nodeadm install コマンドを使用して、ハイブリッドノードに必要なすべての依存関係をインストールできます。ハイブリッドノードの依存関係には、containerd、kubelet、kubectl、AWS SSM または AWS IAM Roles Anywhere コンポーネントが含まれます。nodeadm install によってインストールされるコンポーネントとファイルの場所の詳細については、「the section called “ハイブリッドノード用 nodeadm”」を参照してください。オンプレミスファイアウォールで nodeadm install プロセスのために許可する必要があるドメインの詳細については、「the section called “ネットワークを準備する”」を参照してください。

以下のコマンドを実行して、ハイブリッドノードの依存関係をオンプレミスホストにインストールします。以下のコマンドは、ホストで sudo/root アクセスを持つユーザーで実行する必要があります。

> **Important**
>
> ハイブリッドノードの CLI (nodeadm) は、ホストで sudo/root アクセスを持つユーザーで実行する必要があります。

- K8S_VERSION を、1.31 などの Amazon EKS クラスターの Kubernetes マイナーバージョンに置き換えます。サポートされている Kubernetes のバージョンのリストについては、「Amazon EKS supported versions」を参照してください。
- CREDS_PROVIDER を、使用しているオンプレミスの認証情報プロバイダーに置き換えます。有効な値は、AWS SSM の場合は ssm、AWS IAM Roles Anywhere の場合は iam-ra です。

```
nodeadm install K8S_VERSION --credential-provider CREDS_PROVIDER
```

ステップ 3: ハイブリッドノードをクラスターに接続する

ハイブリッドノードをクラスターに接続する前に、Amazon EKS コントロールプレーンとハイブリッドノード間の通信のために、オンプレミスファイアウォールとクラスターのセキュリティグループで必要なアクセスを許可していることを確認してください。このステップのほとんどの問題は、ファイアウォール設定、セキュリティグループ設定、またはハイブリッドノードの IAM ロール設定に関連しています。

> **Important**
>
> ハイブリッドノードの CLI (nodeadm) は、ホストで sudo/root アクセスを持つユーザーで実行する必要があります。

1. デプロイの値を使用して、各ホストに nodeConfig.yaml ファイルを作成します。使用可能な構成設定の詳細については、「the section called “ハイブリッドノード用 nodeadm”」を参照してください。ハイブリッドノードの IAM ロールに eks:DescribeCluster アクションのアクセス許可がない場合は、nodeConfig.yaml のクラスターセクションで Kubernetes API エンドポイント、クラスター CA バンドル、および Kubernetes サービス IPv4 CIDR を渡す必要があります。
   a. オンプレミス認証情報プロバイダーに AWS SSM ハイブリッドアクティベーションを使用している場合は、以下の nodeConfig.yaml の例を使用します。
      i. CLUSTER_NAME を自分のクラスター名に置き換えます。
      ii. AWS_REGION を、クラスターがホストされている AWS リージョンに置き換えます。例えば、us-west-2。
      iii. ACTIVATION_CODE を、AWS SSM ハイブリッドアクティベーションの作成時に受け取ったアクティベーションコードに置き換えます。詳細については「the section called “認証情報を準備する”」を参照してください。

iv. ACTIVATION_ID を、AWS SSM ハイブリッドアクティベーションの作成時に受け取ったアクティベーション ID に置き換えます。この情報は、AWS Systems Manager コンソールまたは AWS CLI `aws ssm describe-activations` コマンドから取得できます。

```
apiVersion: node.eks.aws/v1alpha1
kind: NodeConfig
spec:
  cluster:
    name: CLUSTER_NAME
    region: AWS_REGION
  hybrid:
    ssm:
      activationCode: ACTIVATION_CODE
      activationId: ACTIVATION_ID
```

b. オンプレミス認証情報プロバイダーに AWS IAM Roles Anywhere を使用している場合は、以下の `nodeConfig.yaml` の例を使用します。

i. CLUSTER_NAME を自分のクラスター名に置き換えます。

ii. AWS_REGION を、クラスターがホストされている AWS リージョンに置き換えます。例えば、`us-west-2`。

iii. NODE_NAME をノードの名前に置き換えます。ハイブリッドノードの IAM ロールの信頼ポリシーを `"sts:RoleSessionName": "${aws:PrincipalTag/x509Subject/CN}"` リソース条件で設定した場合、ノード名はホスト上の証明書の CN と一致する必要があります。使用する nodeName は 64 文字以下にする必要があります。

iv. TRUST_ANCHOR_ARN を、ハイブリッドノードの認証情報を準備する手順で設定したトラストアンカーの ARN に置き換えます。

v. PROFILE_ARN を、the section called “認証情報を準備する” のステップで設定したトラストアンカーの ARN に置き換えます。

vi. ROLE_ARN をハイブリッドノード IAM ロールの ARN に置き換えます。

vii. CERTIFICATE_PATH をノード証明書へのディスク内のパスに置き換えます。指定しなかった場合、デフォルトは `/etc/iam/pki/server.pem` です。

viii. KEY_PATH を、証明書のプライベートキーへのディスク内のパスに置き換えます。指定しなかった場合、デフォルトは `/etc/iam/pki/server.key` です。

```
apiVersion: node.eks.aws/v1alpha1
kind: NodeConfig
spec:
```

```
  cluster:
    name: CLUSTER_NAME
    region: AWS_REGION
  hybrid:
    iamRolesAnywhere:
      nodeName: NODE_NAME
      trustAnchorArn: TRUST_ANCHOR_ARN
      profileArn: PROFILE_ARN
      roleArn: ROLE_ARN
      certificatePath: CERTIFICATE_PATH
      privateKeyPath: KEY_PATH
```

2. nodeConfig.yaml で nodeadm init コマンドを実行して、ハイブリッドノードを Amazon EKS クラスターに接続します。

```
nodeadm init -c file://nodeConfig.yaml
```

上記のコマンドが正常に完了すれば、ハイブリッドノードは Amazon EKS クラスターに参加したことになります。これは、Amazon EKS コンソールでクラスターの [コンピューティング] タブに移動する (IAM プリンシパルに表示権限があることを確認してください) か、または kubectl get nodes を使用して確認できます。

> **Important**
>
> ノードのステータスは Not Ready になります。これは、ハイブリッドノードで実行されている CNI がないことが原因であり、予想通りのことです。ノードがクラスターに参加しなかった場合は、「the section called “トラブルシューティング”」を参照してください。

ステップ 4: ハイブリッドノードの CNI を設定する

ハイブリッドノードでアプリケーションを実行する準備を整えるには、「the section called “CNI を設定する”」の手順に進みます。

Bottlerocket を実行しているハイブリッドノードの接続

このトピックでは、Bottlerocket を実行しているハイブリッドノードを Amazon EKS クラスターに接続する方法について解説します。Bottlerocket は、AWS が支援およびサポートしているオープンソースの Linux ディストリビューションです。Bottlerocket は、コンテナワークロードをホストするために専用に構築されています。Bottlerocket を使用すると、コンテナインフラストラクチャの更

新を自動化することで、コンテナ化されたデプロイの可用性を向上させ、運用コストを削減できます。Bottlerocket には、コンテナの実行に不可欠なソフトウェアのみが含まれており、リソース使用率の向上、セキュリティ上の脅威の軽減、管理オーバーヘッドの軽減が実現されます。

EKS Hybrid Nodes でサポートされているのは、Bottlerocket のバージョンが v1.37.0 以降である VMware のバリアントのみです。Bottlerocket の VMware のバリアントは、Kubernetes のバージョン v1.28 以降で使用できます。これらのバリアント OS イメージには、kubelet、containerd、aws-iam-authenticator、ならびに EKS Hybrid Nodes のその他のソフトウェアの前提条件、が含まれています。これらのコンポーネントは、Bottlerocket のブートストラップと管理コンテナ向けの、base64 でエンコードされたユーザーデータを含む、Bottlerocket の設定ファイルを使用して設定できます。これらを設定することで、Bottlerocket は、ハイブリッドノードの認証情報プロバイダーを使用して、クラスターのハイブリッドノードを認証できるようになります。ハイブリッドノードをクラスターに追加すると、Amazon EKS コンソールや Kubernetes 互換ツール (`kubectl` など) にそれらのステータスが `Not Ready` と表示されます。このページのステップを完了したら、「the section called “CNI を設定する”」に進み、ハイブリッドノードでアプリケーションを実行する準備をします。

前提条件

ハイブリッドノードを Amazon EKS クラスターに接続する前に、前提条件の手順が完了していることを確認してください。

- オンプレミス環境から Amazon EKS クラスターをホストする AWS リージョンへのネットワーク接続があること。詳細については「the section called “ネットワークを準備する”」を参照してください。
- ハイブリッドノードの IAM ロールを作成し、オンプレミス認証情報プロバイダー (AWS Systems Manager ハイブリッドアクティベーションまたは AWS IAM Roles Anywhere) をセットアップしていること。詳細については「the section called “認証情報を準備する”」を参照してください。
- ハイブリッドノード対応の Amazon EKS クラスターを作成していること。詳細については「the section called “クラスターを作成する”」を参照してください。
- ハイブリッドノードの IAM ロールを、Kubernetes ロールベースのアクセスコントロール (RBAC) のアクセス許可に関連付けていること。詳細については「the section called “クラスターアクセスを準備する”」を参照してください。

ステップ 1: Bottlerocket 設定の TOML ファイルを作成する

ハイブリッドノード用に Bottlerocket を設定するには、`settings.toml` ファイルを必要とする設定で作成する必要があります。TOML ファイルのコンテンツは、使用している認証情報プロバイダー

(SSM または IAM Roles Anywhere) に応じて異なります。このファイルは、Bottlerocket インスタンスをプロビジョニングするときに、ユーザーデータとして渡されます。

> **Note**
>
> 以下に示す TOML ファイルは、Bottlerocket VMWare マシンを EKS クラスターのノードとして初期化するために必要な最小設定のみを示しています。Bottlerocket にはさまざまなユースケースに対応する幅広い設定が用意されているため、ハイブリッドノードの初期化以外の設定オプションについては、使用している Bottlerocket バージョンに関して文書化されているすべての設定の包括的なリストが記載されている Bottlerocket ドキュメントを参照してください (例: Bottlerocket 1.51.x で使用できるすべての設定は次のとおりです)。

SSM

AWS Systems Manager を認証情報プロバイダーとして使用している場合は、コンテンツを次のようにして `settings.toml` ファイルを作成します。

```
[settings.kubernetes]
cluster-name = "<cluster-name>"
api-server = "<api-server-endpoint>"
cluster-certificate = "<cluster-certificate-authority>"
hostname-override = "<hostname>"
provider-id = "eks-hybrid:///<region>/<cluster-name>/<hostname>"
authentication-mode = "aws"
cloud-provider = ""
server-tls-bootstrap = true

[settings.network]
hostname = "<hostname>"

[settings.aws]
region = "<region>"

[settings.kubernetes.credential-providers.ecr-credential-provider]
enabled = true
cache-duration = "12h"
image-patterns = [
    "*.dkr.ecr.*.amazonaws.com",
    "*.dkr.ecr.*.amazonaws.com.cn",
    "*.dkr.ecr.*.amazonaws.eu",
```

```
    "*.dkr.ecr-fips.*.amazonaws.com",
    "*.dkr.ecr-fips.*.amazonaws.eu",
    "public.ecr.aws"
]

[settings.kubernetes.node-labels]
"eks.amazonaws.com/compute-type" = "hybrid"
"eks.amazonaws.com/hybrid-credential-provider" = "ssm"

[settings.host-containers.admin]
enabled = true
user-data = "<base64-encoded-admin-container-userdata>"

[settings.bootstrap-containers.eks-hybrid-setup]
mode = "always"
user-data = "<base64-encoded-bootstrap-container-userdata>"

[settings.host-containers.control]
enabled = true
```

プレースホルダを次の値に置き換えます。

- <cluster-name>: Amazon EKS クラスターの名前
- <api-server-endpoint>: クラスターの API サーバーエンドポイント
- <cluster-certificate-authority>: クラスターの base64 エンコードされた CA バンドル
- <region>: クラスターをホストしている AWS リージョン。例: "us-east-1"
- <hostname>: Bottlerocket インスタンスのホスト名。これはノード名としても設定されます。こちらは任意の一意の値にすることができますが、Kubernetes オブジェクトの命名規則に従う必要があります。さらに、使用するホスト名は 64 文字以下にする必要があります。注: SSM プロバイダーを使用する場合、このホスト名とノード名は、インスタンスが SSM に登録された後、マネージドインスタンス ID (mi-* ID など) に置き換えられます。
- <base64-encoded-admin-container-userdata>: Bottlerocket 管理コンテナの設定の、base64 でエンコードされたコンテンツ。管理コンテナを有効にすると、SSH を使用して Bottlerocket インスタンスに接続し、システムの探索やデバッグを行えるようになります。これは必須の設定ではありませんが、トラブルシューティングを容易にするため有効にしておくことが推奨されます。管理コンテナを使用した認証の詳細については Bottlerocket 管理コンテナのドキュメントを参照してください。管理者コンテナは、以下の例に示すように、SSH ユーザーとキーの入力を JSON 形式で受け取ります。

```
{
  "user": "<ssh-user>",
  "ssh": {
    "authorized-keys": [
      "<ssh-authorized-key>"
    ]
  }
}
```

- <base64-encoded-bootstrap-container-userdata>: Bottlerocket ブートストラップコンテナの設定の、base64 でエンコードされたコンテンツ。設定の詳細については Bottlerocket ブートストラップコンテナのドキュメントを参照してください。ブートストラップコンテナは、インスタンスを AWS SSM マネージドインスタンスとして登録し、これを Amazon EKS クラスターの Kubernetes ノードとして結合する役割を果たします。ブートストラップコンテナに渡されるユーザーデータは、以前に作成した SSM ハイブリッドアクティベーションコードと ID を入力として受け入れる、コマンド呼び出しの形式をとります。

```
eks-hybrid-ssm-setup --activation-id=<activation-id> --activation-code=<activation-
code> --region=<region>
```

IAM Roles Anywhere

AWS IAM Roles Anywhere を認証情報プロバイダーとして使用している場合は、コンテンツを次のようにして settings.toml ファイルを作成します。

```
[settings.kubernetes]
cluster-name = "<cluster-name>"
api-server = "<api-server-endpoint>"
cluster-certificate = "<cluster-certificate-authority>"
hostname-override = "<hostname>"
provider-id = "eks-hybrid:///<region>/<cluster-name>/<hostname>"
authentication-mode = "aws"
cloud-provider = ""
server-tls-bootstrap = true

[settings.network]
hostname = "<hostname>"

[settings.aws]
region = "<region>"
```

```
config = "<base64-encoded-aws-config-file>"

[settings.kubernetes.credential-providers.ecr-credential-provider]
enabled = true
cache-duration = "12h"
image-patterns = [
    "*.dkr.ecr.*.amazonaws.com",
    "*.dkr.ecr.*.amazonaws.com.cn",
    "*.dkr.ecr.*.amazonaws.eu",
    "*.dkr.ecr-fips.*.amazonaws.com",
    "*.dkr.ecr-fips.*.amazonaws.eu",
    "public.ecr.aws"
]

[settings.kubernetes.node-labels]
"eks.amazonaws.com/compute-type" = "hybrid"
"eks.amazonaws.com/hybrid-credential-provider" = "iam-ra"

[settings.host-containers.admin]
enabled = true
user-data = "<base64-encoded-admin-container-userdata>"

[settings.bootstrap-containers.eks-hybrid-setup]
mode = "always"
user-data = "<base64-encoded-bootstrap-container-userdata>"
```

プレースホルダを次の値に置き換えます。

- <cluster-name>: Amazon EKS クラスターの名前
- <api-server-endpoint>: クラスターの API サーバーエンドポイント
- <cluster-certificate-authority>: クラスターの base64 エンコードされた CA バンドル
- <region>: クラスターをホストしている AWS リージョン。例: "us-east-1"
- <hostname>: Bottlerocket インスタンスのホスト名。これはノード名としても設定されます。こちらは任意の一意の値にすることができますが、Kubernetes オブジェクトの命名規則に従う必要があります。さらに、使用するホスト名は 64 文字以下にする必要があります。注: IAM-RA プロバイダーを使用しているときは、ノード名は、ハイブリッドノードの IAM ロールの信頼ポリシーを "sts:RoleSessionName": "${aws:PrincipalTag/x509Subject/CN}" リソース条件で設定した場合にホストの証明書の CN と一致している必要があります。
- <base64-encoded-aws-config-file>: AWS 設定ファイルの base64 でエンコードされたコンテンツ。ファイルのコンテンツは以下のようになるはずです。

```
[default]
credential_process = aws_signing_helper credential-process --certificate /root/.aws/
node.crt --private-key /root/.aws/node.key --profile-arn <profile-arn> --role-arn
 <role-arn> --trust-anchor-arn <trust-anchor-arn> --role-session-name <role-session-
name>
```

- <base64-encoded-admin-container-userdata>: Bottlerocket 管理コンテナの設定の、base64 でエンコードされたコンテンツ。管理コンテナを有効にすると、SSH を使用して Bottlerocket インスタンスに接続し、システムの探索やデバッグを行えるようになります。これは必須の設定ではありませんが、トラブルシューティングを容易にするため有効にしておくことが推奨されます。管理コンテナを使用した認証の詳細については Bottlerocket 管理コンテナのドキュメントを参照してください。管理者コンテナは、以下の例に示すように、SSH ユーザーとキーの入力を JSON 形式で受け取ります。

```
{
  "user": "<ssh-user>",
  "ssh": {
    "authorized-keys": [
      "<ssh-authorized-key>"
    ]
  }
}
```

- <base64-encoded-bootstrap-container-userdata>: Bottlerocket ブートストラップコンテナの設定の、base64 でエンコードされたコンテンツ。設定の詳細については Bottlerocket ブートストラップコンテナのドキュメントを参照してください。ブートストラップコンテナは、インスタンスに IAM Roles Anywhere ホスト証明書と証明書プライベートキーファイルを作成する役割を果たします。これらはその後、Amazon EKS クラスターで認証を行うときに、一時的な認証情報を取得するために aws_signing_helper が使用します。ブートストラップコンテナに渡されるユーザーデータは、以前に作成した証明書とプライベートキーのコンテンツを入力として受け入れる、コマンド呼び出しの形式をとります。

```
eks-hybrid-iam-ra-setup --certificate=<certificate> --key=<private-key>
```

ステップ 2: ユーザーデータを使用して Bottlerocket vSphere VM をプロビジョニングする

TOML ファイルを作成したら、vSphere VM の作成時にこれをユーザーデータとして渡します。ユーザーデータは VM の電源を初めて入れる前に設定しておく必要があることを忘れないでください。そのため、インスタンスの作成時にこれを指定しておく必要があります。または、VM を事前に作成する場合は、ユーザーデータを設定するまで VM のステータスを poweredOff にしておく必要があります。govc CLI を使用する場合の例を以下に示します。

VM を初めて作成する

```
govc vm.create \
  -on=true \
  -c=2 \
  -m=4096 \
  -net.adapter=<network-adapter> \
  -net=<network-name> \
  -e guestinfo.userdata.encoding="base64" \
  -e guestinfo.userdata="$(base64 -w0 settings.toml)" \
  -template=<template-name> \
  <vm-name>
```

既存の VM のユーザーデータを更新する

```
govc vm.create \
    -on=false \
    -c=2 \
    -m=4096 \
    -net.adapter=<network-adapter> \
    -net=<network-name> \
    -template=<template-name> \
    <vm-name>

govc vm.change
    -vm <vm-name> \
    -e guestinfo.userdata="$(base64 -w0 settings.toml)" \
    -e guestinfo.userdata.encoding="base64"

govc vm.power -on <vm-name>
```

上記のセクションでは、`-e guestinfo.userdata.encoding="base64"` オプションでユーザーデータが base64 でエンコードされていることを指定します。`-e guestinfo.userdata` オプションは、`settings.toml` ファイルの base64 でエンコードされたコンテンツを、ユーザーデータと

して Bottlerocket インスタンスに渡します。プレースホルダーを Bottlerocket OVA テンプレートやネットワークの詳細など特定の値に置き換えます。

ステップ 3: ハイブリッドノードの接続を検証する

Bottlerocket インスタンスは、起動すると、Amazon EKS クラスターに結合しようとします。Amazon EKS コンソールでこの接続を検証するには、クラスターの [コンピューティング] タブに移動するか、次のコマンドを実行します。

```
kubectl get nodes
```

> **Important**
>
> ノードのステータスは `Not Ready` になります。これは、ハイブリッドノードで実行されている CNI がないことが原因であり、予想通りのことです。ノードがクラスターに参加しなかった場合は、「the section called “トラブルシューティング”」を参照してください。

ステップ 4: ハイブリッドノードの CNI を設定する

ハイブリッドノードでアプリケーションを実行する準備を整えるには、「the section called “CNI を設定する”」の手順に進みます。

クラスターのハイブリッドノードをアップグレードする

ハイブリッドノードのアップグレードに関するガイダンスは、Amazon EC2 で実行されるセルフマネージド Amazon EKS ノードと類似しています。ターゲットの Kubernetes バージョンで新しいハイブリッドノードを作成し、既存のアプリケーションを新しい Kubernetes バージョンのハイブリッドノードに円滑に移行し、古い Kubernetes バージョンのハイブリッドノードをクラスターから削除することが推奨されます。アップグレードを開始する前に、アップグレードに関する「Amazon EKS ベストプラクティス」を確認してください。Amazon EKS Hybrid Nodes には、標準サポートや拡張サポートなどの、クラウドノードを持つ Amazon EKS クラスターに対して同じ Kubernetes バージョンのサポートがあります。

Amazon EKS Hybrid Nodes は、アップストリーム Kubernetes と同じノードのバージョンスキューポリシーに従います。Amazon EKS Hybrid Nodes は、Amazon EKS コントロールプレーンよりも新しいバージョンにすることはできません。また、ハイブリッドノードは、Amazon EKS のコントロールプレーンのマイナーバージョンより、最大 3 つ前まで古い Kubernetes のマイナーバージョンである可能性があります。

カットオーバー移行のアップグレード戦略のためにターゲットの Kubernetes バージョンに新しいハイブリッドノードを作成する空き容量がない場合は、代わりに Amazon EKS Hybrid Nodes CLI (nodeadm) を使用してハイブリッドノードの Kubernetes バージョンをインプレースでアップグレードできます。

Important

nodeadm を使用してインプレースでハイブリッドノードをアップグレードしている場合、古いバージョンの Kubernetes コンポーネントがシャットダウンされ、新しい Kubernetes バージョンコンポーネントがインストールされて起動されるプロセス中に、ノードのダウンタイムが発生します。

SSM 認証情報プロバイダーに必要な nodeadm バージョン

ハイブリッドノードの認証情報プロバイダーとして AWS Systems Manager (SSM) を使用する場合は、新しいインストールとアップグレードに nodeadm バージョン `1.0.19` 以降を使用する必要があります。以前のバージョンの nodeadm には古い SSM 署名キーが含まれており、`nodeadm install` と `nodeadm upgrade` の実行中に失敗し、次の署名検証エラーが発生します:

```
"msg":"Command failed","error":"failed to install ssm installer: validating ssm-setup-cli signature: Signature Verification Error: No matching signature"
```

このエラーを解決するには、`nodeadm install` または `nodeadm upgrade` を実行する前に nodeadm の最新バージョンをダウンロードします。

前提条件

アップグレードする前に、以下の前提条件を満たしていることを確認してください。

- ハイブリッドノードのアップグレードのターゲット Kubernetes バージョンは、Amazon EKS コントロールプレーンのバージョン以下である必要があります。
- カットオーバー移行のアップグレード戦略に従う場合、ターゲット Kubernetes バージョンにインストールする新しいハイブリッドノードは、the section called “前提条件” 要件を満たしている必要があります。この要件には、Amazon EKS クラスターの作成時に渡した Remote Node Network CIDR 内に IP アドレスがあることが含まれます。

- カットオーバー移行とインプレースアップグレードの両方について、ハイブリッドノードは、ハイブリッドノードの依存関係の新しいバージョンを取得するために必要なドメインにアクセスできる必要があります。
- Amazon EKS Kubernetes API エンドポイントとのやり取りに使用するローカルマシンまたはインスタンスに kubectl をインストールしてる必要があります。
- CNI のバージョンは、アップグレード先の Kubernetes バージョンをサポートしている必要があります。そうでない場合は、ハイブリッドノードをアップグレードする前に CNI バージョンをアップグレードします。詳細については「the section called “CNI を設定する”」を参照してください。

カットオーバー移行 (Blue/Green) アップグレード

カットオーバー移行のアップグレードとは、ターゲット Kubernetes バージョンを使用して新しいホストに新しいハイブリッドノードを作成し、既存のアプリケーションをターゲット Kubernetes バージョンの新しいハイブリッドノードに円滑に移行し、クラスターから古い Kubernetes のバージョンのハイブリッドノードを削除するプロセスを指します。この戦略は Blue/Green 移行とも呼ばれます。

1. the section called “ハイブリッドノードを接続する” 手順に従って、新しいホストをハイブリッドノードとして接続します。`nodeadm install` コマンドを実行するときは、ターゲット Kubernetes バージョンを使用します。
2. ターゲット Kubernetes バージョンの新しいハイブリッドノードと古い Kubernetes バージョンのハイブリッドノード間の通信を有効にします。この設定により、ターゲット Kubernetes バージョンのハイブリッドノードにワークロードを移行している間に、ポッドが相互に通信できるようになります。
3. ターゲット Kubernetes バージョンのハイブリッドノードがクラスターに正常に参加しており、ステータスが Ready (準備完了) であることを確認します。
4. 以下のコマンドを使用して、削除する各ノードをスケジュール不可としてマークします。これは、置き換えるノード上で新しいポッドがスケジュールまたは再スケジュールされないようにするためです。詳細については、Kubernetes のドキュメントの「kubectl cordon」を参照してください。`NODE_NAME` を古い Kubernetes バージョンのハイブリッドノードの名前に置き換えます。

   ```
   kubectl cordon NODE_NAME
   ```

   次のコードスニペットを使用して、特定の Kubernetes バージョン (この場合は `1.28`) のすべてのノードを識別して隔離できます。

```
K8S_VERSION=1.28
for node in $(kubectl get nodes -o json | jq --arg K8S_VERSION
 "$K8S_VERSION" -r '.items[] | select(.status.nodeInfo.kubeletVersion |
 match("\($K8S_VERSION)")).metadata.name')
do
    echo "Cordoning $node"
    kubectl cordon $node
done
```

5. 現在のデプロイメントで、ハイブリッドノードで実行している CoreDNS レプリカが 2 つ未満の場合は、そのデプロイメントを少なくとも 2 つのレプリカにスケールアウトします。通常のオペレーションでは、回復力を高めるためにハイブリッドノードで少なくとも 2 つの CoreDNS レプリカを実行することが推奨されます。

```
kubectl scale deployments/coredns --replicas=2 -n kube-system
```

6. 次のコマンドを使用して、クラスターから削除する古い Kubernetes バージョンのハイブリッドノードをそれぞれドレインします。ノードドレインのドレイン処理に関する詳細については、Kubernetes ドキュメントの「Safely Drain a Node」を参照してください。NODE_NAME を古い Kubernetes バージョンのハイブリッドノードの名前に置き換えます。

```
kubectl drain NODE_NAME --ignore-daemonsets --delete-emptydir-data
```

   以下のコードスニペットを使用して、特定の Kubernetes バージョン (この場合は 1.28) のすべてのノードを識別してドレインすることができます。

```
K8S_VERSION=1.28
for node in $(kubectl get nodes -o json | jq --arg K8S_VERSION
 "$K8S_VERSION" -r '.items[] | select(.status.nodeInfo.kubeletVersion |
 match("\($K8S_VERSION)")).metadata.name')
do
    echo "Draining $node"
    kubectl drain $node --ignore-daemonsets --delete-emptydir-data
done
```

7. nodeadm を使用して、ホストからハイブリッドノードアーティファクトを停止および削除できます。root/sudo 権限を持つユーザーで nodeadm を実行する必要があります。デフォルトでは、ノードにポッドが残っている場合、nodeadm uninstall は続行されません。詳細については、「the section called “ハイブリッドノード用 nodeadm”」を参照してください。

```
nodeadm uninstall
```

8. ハイブリッドノードのアーティファクトを停止およびアンインストールしたら、クラスターからノードリソースを削除します。

```
kubectl delete node node-name
```

次のコードスニペットを使用して、特定の Kubernetes バージョン (この場合は 1.28) のすべてのノードを識別および削除できます。

```
K8S_VERSION=1.28
for node in $(kubectl get nodes -o json | jq --arg K8S_VERSION
 "$K8S_VERSION" -r '.items[] | select(.status.nodeInfo.kubeletVersion |
 match("\($K8S_VERSION)")).metadata.name')
do
    echo "Deleting $node"
    kubectl delete node $node
done
```

9. CNI の選択によっては、上記のステップを実行した後にハイブリッドノードにアーティファクトが残っている場合があります。詳細については「the section called “CNI を設定する”」を参照してください。

インプレースアップグレード

インプレースアップグレードプロセスとは、nodeadm upgrade を使用して、新しい物理ホストまたは仮想ホストやカットオーバー移行戦略を使用せずに、ハイブリッドノードの Kubernetes バージョンをアップグレードすることを指します。この nodeadm upgrade プロセスでは、ハイブリッドノードで実行されている既存の古い Kubernetes コンポーネントのシャットダウン、既存の古い Kubernetes コンポーネントのアンインストール、新しいターゲット Kubernetes コンポーネントのインストールを行い、新しいターゲット Kubernetes コンポーネントを起動します。ハイブリッドノードで実行されているアプリケーションへの影響を最小限に抑えるために、一度にアップグレードするノードは 1 つにすることを強くお勧めします。このプロセスの期間は、ネットワーク帯域幅とレイテンシーによって異なります。

1. アップグレードするノードをスケジュール不可としてマークするには、次のコマンドを使用します。これは、アップグレードするノード上で新しいポッドがスケジュールまたは再スケ

ジュールされないようにするためです。詳細については、Kubernetes のドキュメントの「kubectl cordon」を参照してください。NODE_NAME をアップグレードするハイブリッドノードの名前に置き換える

```
kubectl cordon NODE_NAME
```

2. 次のコマンドを使用して、アップグレードするノードをドレインします。ノードドレインのドレイン処理に関する詳細については、Kubernetes ドキュメントの「Safely Drain a Node」を参照してください。NODE_NAME をアップグレードするハイブリッドノードの名前に置き換えます。

```
kubectl drain NODE_NAME --ignore-daemonsets --delete-emptydir-data
```

3. アップグレードするハイブリッドノードで nodeadm upgrade を実行します。root/sudo 権限を持つユーザーで nodeadm を実行する必要があります。ノードの名前は、AWS SSM と AWS IAM Roles Anywhere の両方の認証情報プロバイダーのアップグレードによって保持されます。アップグレードプロセス中に認証情報プロバイダーを変更することはできません。nodeConfig.yaml の設定値については、「the section called “ハイブリッドノード用 nodeadm”」を参照してください。K8S_VERSION を、アップグレード先のターゲット Kubernetes バージョンに置き換えます。

```
nodeadm upgrade K8S_VERSION -c file://nodeConfig.yaml
```

4. アップグレード後にノード上で Pod をスケジュールできるように、以下を入力します。NODE_NAME をノードの名前に置き換えます。

```
kubectl uncordon NODE_NAME
```

5. ハイブリッドノードのステータスを監視し、ノードがシャットダウンするのを待ち、Ready (準備完了) ステータスになったら新しい Kubernetes バージョンで再起動します。

```
kubectl get nodes -o wide -w
```

ハイブリッドノード用のセキュリティ更新プログラムにパッチを適用する

このトピックでは、ハイブリッドノードで実行されている特定のパッケージと依存関係を対象としたセキュリティ更新プログラムにインプレースでパッチを適用する手順について説明します。ベストプラクティスとして、ハイブリッドノードを定期的に更新して CVE とセキュリティパッチを受け取ることをお勧めします。

Kubernetes バージョンをアップグレードする手順については、「the section called “ハイブリッドノードをアップグレードする”」を参照してください。

セキュリティパッチの適用が必要になるソフトウェアの一例が containerd です。

Containerd

containerd は、標準の Kubernetes コンテナランタイムであり、EKS ハイブリッドノードとはコアな依存関係にあります。その用途は、イメージのプルやコンテナ実行の管理などコンテナのライフサイクルを管理することです。ハイブリッドノードでは、nodeadm CLI を利用するか手動で containerd をインストールできます。ノードのオペレーティングシステムに応じて、nodeadm は OS 配布のパッケージまたは Docker パッケージから containerd をインストールします。

containerd の CVE が公開されたら、以下のオプションを利用して、ハイブリッドノードで containerd をパッチ適用済みのバージョンにアップグレードできます。

ステップ 1: パッチがパッケージマネージャーに公開されているかどうかを確認する

containerd CVE パッチがそれぞれの OS パッケージマネージャーに公開されているかどうか、対応するセキュリティ速報を参照することで確認できます。

- Amazon Linux 2023
- RHEL
- Ubuntu 20.04
- Ubuntu 22.04
- Ubuntu 24.04

Docker リポジトリを containerd のソースとして使用している場合は、Docker セキュリティのお知らせをチェックして、Docker リポジトリからパッチ適用済みのバージョンを入手できるかどうかを確認できます。

ステップ 2: パッチのインストール方法を選択する

ノードにインプレースでパッチを適用し、セキュリティアップグレードをインストールする方法が 3 つあります。どの方法を使用できるかは、パッチをパッケージマネージャーでオペレーティングシステムから入手できるかどうかによって異なります。

1. nodeadm upgrade を使用してパッケージマネージャーに公開されているパッチをインストールします。「ステップ 2 a」を参照してください。

2. パッケージマネージャーで直接パッチをインストールします。「ステップ 2 b」を参照してください。
3. パッケージマネージャーに公開されていないカスタムパッチをインストールします。containerd に対するカスタムパッチについては特別な考慮事項があるので注意してください。「ステップ 2 c」を参照してください。

ステップ 2 a: **nodeadm upgrade** でパッチを適用する

containerd CVE パッチが OS や Docker リポジトリ (Apt または RPM) に公開されたことを確認したら、nodeadm upgrade コマンドを使用して containerd を最新バージョンにアップグレードできます。これは Kubernetes バージョンのアップグレードではないため、現在の Kubernetes バージョンを nodeadm アップグレードコマンドに渡す必要があります。

```
nodeadm upgrade K8S_VERSION --config-source file:///root/nodeConfig.yaml
```

ステップ 2 b: オペレーティングシステムのパッケージマネージャーでパッチを適用する

あるいは、それぞれのパッケージマネージャーから更新することも可能であり、それを使用して以下のように containerd パッケージをアップグレードできます。

Amazon Linux 2023

```
sudo yum update -y
sudo yum install -y containerd
```

RHEL

```
sudo yum install -y yum-utils
sudo yum-config-manager --add-repo https://download.docker.com/linux/rhel/docker-
ce.repo
sudo yum update -y
sudo yum install -y containerd
```

Ubuntu

```
sudo mkdir -p /etc/apt/keyrings
sudo curl -fsSL https://download.docker.com/linux/ubuntu/gpg -o /etc/apt/keyrings/
docker.asc
```

```
echo \
  "deb [arch=$(dpkg --print-architecture) signed-by=/etc/apt/keyrings/docker.asc]
 https://download.docker.com/linux/ubuntu \
  $(. /etc/os-release && echo "${UBUNTU_CODENAME:-$VERSION_CODENAME}") stable" | \
  sudo tee /etc/apt/sources.list.d/docker.list > /dev/null
sudo apt update -y
sudo apt install -y --only-upgrade containerd.io
```

ステップ 2 c: **Containerd** CVE パッチがパッケージマネージャーに公開されていない場合の対応

パッチ適用済みの containerd バージョンがパッケージマネージャー以外の手段でのみ使用できる場合、例えば GitHub リリースのみであれば GitHub の公式サイトから containerd をインストールできます。

1. マシンがハイブリッドノードとしてクラスターに既に参加している場合は、nodeadm uninstall コマンドを実行する必要があります。
2. containerd の公式のバイナリをインストールします。GitHub の公式のインストール手順を使用できます。
3. --containerd-source 引数を none に設定して nodeadm install コマンドを実行します。nodeadm から containerd をインストールする操作がスキップされます。ノードでどのオペレーティングシステムが実行されていても、containerd ソースで none の値を使用できます。

```
nodeadm install K8S_VERSION --credential-provider CREDS_PROVIDER --containerd-source
 none
```

ハイブリッドノードを削除する

このトピックではAmazon EKS クラスターからハイブリッドノードを削除する方法について説明します。kubectl などの Kubernetes 互換ツールを選択して、ハイブリッドノードを削除する必要があります。ハイブリッドノードの課金はノードオブジェクトが Amazon EKS クラスターから削除されると停止します。ハイブリッドノードの料金の詳細については「Amazon EKS 料金表」を参照してください。

> **Important**
>
> ノードを削除すると、ノードで実行されているワークロードが中断されます。ハイブリッドノードを削除する前に、まずノードをドレインしてポッドを別のアクティブノードに移動す

ることが推奨されます。ノードドレインのドレイン処理に関する詳細についてはKubernetes ドキュメントの「ノードの安全な排水」を参照してください。

Amazon EKS クラスターの Kubernetes API エンドポイントとのやり取りに使用するローカルマシンまたはインスタンスから、以下の kubectl ステップを実行してください。特定の kubeconfig ファイルを使用している場合は--kubeconfig フラグを使用します。

ステップ 1: ノードを一覧表示する

```
kubectl get nodes
```

ステップ 2: ノードをドレインする

kubectl drain コマンドの詳細についてはKubernetes ドキュメントの「kubectl drain」を参照してください。

```
kubectl drain --ignore-daemonsets <node-name>
```

ステップ 3: ハイブリッドノードアーティファクトを停止およびアンインストールする

Amazon EKS Hybrid Nodes CLI (nodeadm) を使用して、ホストからハイブリッドノードアーティファクトを停止および削除できます。root/sudo 権限を持つユーザーで nodeadm を実行する必要があります。デフォルトではノードにポッドが残っている場合、nodeadm uninstall は続行されません。AWS システム・マネージャー (SSM) を認証情報プロバイダーとして使用している場合、nodeadm uninstall コマンドはホストの AWS SSM マネージドインスタンスとしての登録を解除します。詳細については「the section called “ハイブリッドノード用 nodeadm”」を参照してください。

```
nodeadm uninstall
```

ステップ 4: クラスターからノードを削除する

ハイブリッドノードのアーティファクトを停止およびアンインストールしたら、クラスターからノードリソースを削除します。

```
kubectl delete node <node-name>
```

ステップ 5: 残りのアーティファクトを確認する

CNI の選択によっては上記のステップを実行した後にハイブリッドノードにアーティファクトが残っている場合があります。詳細については「the section called “CNI を設定する”」を参照してください。

ハイブリッドノードのアプリケーションネットワーキング、アドオン、ウェブフックを設定する

ハイブリッドノード用に EKS クラスターを作成したら、アプリケーションネットワーキングに関する追加の機能 (CNI、BGP、Ingress、Load Balancing、Network Policies)、アドオン、ウェブフック、プロキシを設定します。ハイブリッドノードと互換性のある EKS およびコミュニティアドオンの完全なリストは、「the section called “CNI アドオンの設定”」を参照してください。

EKS クラスターインサイト EKS には、クラスターまたはワークロードの機能を損なう可能性のあるハイブリッドノードの設定ミスに関するインサイトチェックが含まれています。クラスターインサイトの詳細については、「the section called “クラスターのインサイト”」を参照してください。

以下に、ハイブリッドノードでよく使用される機能とアドオンを示します。

- Container Networking Interface (CNI): AWS は、ハイブリッドノード向けの CNI として Cilium をサポートしています。詳細については、「the section called “CNI を設定する”」を参照してください。AWS VPC CNI はハイブリッドノードでは使用できないことに注意してください。
- CoreDNS と **`kube-proxy`**: ハイブリッドノードが EKS クラスターと結合したときは、CoreDNS と `kube-proxy` が自動的にインストールされます。これらのアドオンは、クラスターの作成後、EKS アドオンとして管理できます。
- Ingress と Load Balancing: ハイブリッドノード上で実行されているワークロードのターゲットタイプが `ip` の場合、AWS Load Balancer Controller を Application Load Balancer (ALB) または Network Load Balancer (NLB) と共に使用できます。AWS は、ハイブリッドノード上で実行されているワークロード向けに Cilium に組み込みの Ingress、Gateway、Kubernetes Service のロードバランシング機能をサポートしています。詳細については、「the section called “Ingress を設定する”」および「the section called “LoadBalancer Service を設定する”」を参照してください。
- メトリクス: Amazon Managed Service for Prometheus (AMP) エージェントレススクレイパー、AWS Distro for Open Telemetry (ADOT)、Amazon CloudWatch Observability Agent は、ハイブリッドノードで使用できます。ポッドメトリクス用の AMP エージェントレススクレイパーをハイブリッドノードで使用する場合は、Amazon EKS クラスターに使用する VPC からポッドにアクセスできる必要があります。

- ログ: ハイブリッドノードが有効になっているクラスターに対して、EKS コントロールプレーンのログ記録を有効にすることができます。ADOT EKS アドオンと Amazon CloudWatch Observability Agent EKS アドオンを、ハイブリッドノードとポッドのログ記録に使用できます。
- Pod Identity と IRSA: EKS Pod Identity と IAM Roles for Service Accounts (IRSA) を、ハイブリッドノードで実行中のアプリケーションで使用すると、ハイブリッドノードで実行中の Pod に、他の AWS サービスを使ってきめ細かくアクセスすることが可能になります。
- ウェブフック: ウェブフックを実行している場合は、オンプレミスのポッドネットワークをルーティングできない場合に、クラウドノードでウェブフックを選択的に実行する際の考慮事項と手順について「the section called “ウェブフックの設定”」を参照してください。
- Proxy: データセンターまたはエッジ環境から送信されるトラフィックに、オンプレミス環境にあるプロキシサーバーを使用している場合は、そのプロキシサーバーを使用するようにノードとクラスターを設定することができます。詳細については、「the section called “プロキシを設定する”」を参照してください。

トピック

- ハイブリッドノードの CNI を設定する
- ハイブリッドノードのアドオンを構成する
- ハイブリッドノード用のウェブフックを設定する
- ハイブリッドノードのプロキシを設定する
- ハイブリッドノード向けに Cilium BGP を設定する
- ハイブリッドノード用に Kubernetes Ingress を設定する
- ハイブリッドノード用にタイプ LoadBalancer の Service を設定する
- ハイブリッドノード用に Kubernetes ネットワークポリシーを設定する

ハイブリッドノードの CNI を設定する

Cilium は、Amazon EKS Hybrid Nodes 向けに AWS でサポートされるコンテナネットワークインターフェイス (CNI) です。ハイブリッドノードがワークロードを処理できるようにするにはCNI をインストールする必要があります。CNI が稼働するまではハイブリッドノードは `Not Ready` のステータスで表示されます。CNI は、Helm など任意のツールで管理できます。このページでは、Cilium ライフサイクル管理 (インストール、アップグレード、削除) の手順を説明します。Cilium をイングレス、ロードバランシング、ネットワークポリシー用に設定する方法については、「the section called “Cilium Ingress および Cilium Gateway の概要”」、「the section called “サービスタイプ LoadBalancer”」、「the section called “ネットワークポリシーを設定する”」を参照してください。

AWS では、AWS Cloud 内のノードで Cilium を実行する操作はサポートされていません。Amazon VPC CNI はハイブリッドノードと互換性がなく、VPC CNI は `eks.amazonaws.com/compute-type: hybrid` ラベルに対しアンチアフィニティで設定されています。

このページにこれまで記載されていた Calico ドキュメントは、EKS ハイブリッドサンプルリポジトリに移動しました。

バージョン互換性

Cilium バージョン `v1.17.x` および `v1.18.x` は、Amazon EKS でサポートされているすべての Kubernetes バージョンの EKS ハイブリッドノードでサポートされています。

> **Note**
>
> Cilium v1.18.3 カーネル要件: カーネル要件 (Linux カーネル >= 5.10) により、Cilium v1.18.3 は以下ではサポートされていません。

- Ubuntu 20.04
- Red Hat Enterprise Linux (RHEL) 8

システム要件については、「Cilium システム要件」を参照してください。

Amazon EKS でサポートされている Kubernetes のバージョンについては、「Kubernetes version support」を参照してください。EKS Hybrid Nodes には、クラウドノードを使用する Amazon EKS クラスターと同じ Kubernetes バージョンのサポートがあります。

サポートされる機能

AWS は、オープンソース Cilium プロジェクト に基づく Cilium for EKS Hybrid Nodes のビルドを維持します。AWS から Cilium のサポートを受けるには、AWS が維持する Cilium ビルドとサポート対象の Cilium バージョンを使用している必要があります。

AWS は、EKS ハイブリッドノードで使用する Cilium の以下の機能のデフォルト設定についてテクニカルサポートを提供しています。AWS のサポート範囲外の機能を使用する場合は、Cilium の代替商用サポートを受けるか、社内のエキスパートがトラブルシューティングして解決策を Cilium プロジェクトに提供することをお勧めします。

Cilium 機能	AWS によりサポート
Kubernetes ネットワーク適合性	はい
コアクラスターの接続	はい
IP ファミリー	IPv4
ライフサイクル管理	Helm
ネットワークモード	VXLAN カプセル化
IP アドレス管理 (IPAM)	Cilium IPAM クラススコープ
ネットワークポリシー	Kubernetes ネットワークポリシー
ボーダーゲートウェイプロトコル (BGP)	Cilium BGP コントロールプレーン
Kubernetes Ingress	Cilium Ingress、Cilium Gateway
Service LoadBalancer IP 割り当て	Cilium Load Balancer IPAM
Service LoadBalancer の IP アドレスのアドバタイズ	Cilium BGP コントロールプレーン
kube-proxy replacement	はい

Cilium に関する考慮事項

- Helm リポジトリ - AWS は、Amazon Elastic Container Registry Public (Amazon ECR Public) の Amazon EKS Cilium/Cilium で Cilium Helm チャートをホストします。使用可能なバージョンは次のとおりです。
  - Cilium v1.17.9: `oci://public.ecr.aws/eks/cilium/cilium:1.17.9-0`
  - Cilium v1.18.3: `oci://public.ecr.aws/eks/cilium/cilium:1.18.3-0`

    このトピックで示したコマンドでは、このリポジトリを使用しています。`helm repo` コマンドの中に Amazon ECR Public の Helm リポジトリに対して有効でないものがあるため、ローカル

の Helm リポジトリ名からこのリポジトリを参照できません。代わりに、ほとんどのコマンドで URI を完全な形で使用してください。

- デフォルトでは、Cilium は VXLAN をカプセル化方法とするオーバーレイ/トンネルモードで動作するように設定されています。このモードでは、基盤となる物理ネットワークに関する要件が最も少なくなります。
- デフォルトでは、Cilium はクラスターから出ていくすべてのポッドトラフィックの送信元 IP アドレスをノードの IP アドレスにマスカレードします。マスカレードを無効にする場合は、ポッド CIDR がオンプレミスネットワークでルーティング可能である必要があります。
- ハイブリッドノードでウェブフックを実行している場合は、オンプレミスネットワークでポッド CIDR をルーティング可能である必要があります。オンプレミスネットワークでポッド CIDR をルーティングできない場合は、同じクラスター内のクラウドノードでウェブフックを実行することをお勧めします。詳細については、「the section called “ウェブフックの設定”」と「the section called “ネットワークを準備する”」を参照してください。
- AWS では、Cilium に組み込みの BGP 機能を使用して、オンプレミスネットワークでポッド CIDR をルーティング可能にすることをお勧めします。ハイブリッドノードで Cilium BGP を設定する方法の詳細については、「the section called “BGP を設定する”」を参照してください。
- Cilium のデフォルトの IP Address Manager (IPAM) はクラスタースコープと呼ばれ、Cilium オペレーターがユーザー設定のポッド CIDR に基づいてノードごとに IP アドレスを割り当てます。

ハイブリッドノードに Cilium をインストールする

手順

1. `cilium-values.yaml` という名前の YAML ファイルを作成します。次の例では、Cilium のエージェントとオペレーターの `eks.amazonaws.com/compute-type: hybrid` ラベルにアフィニティを設定することで、ハイブリッドノードでのみ動作するように Cilium を設定しています。
   - EKS クラスターのリモートポッドネットワークの場合と同じポッド CIDR で `clusterPoolIpv4PodCIDRList` を設定してください。例えば、`10.100.0.0/24`。Cilium オペレーターは、設定された `clusterPoolIpv4PodCIDRList` IP スペース内から IP アドレススライスを割り当てます。ポッド CIDR は、オンプレミスノード CIDR、VPC CIDR、Kubernetes サービス CIDR と重複してはいけません。
   - ノードあたりに必要なポッドに基づいて `clusterPoolIpv4MaskSize` を設定します。例えば、ノードあたり 128 個のポッドが必要で、そのサイズが /25 セグメントの場合は 25 とします。

- クラスターに Cilium をデプロイした後で `clusterPoolIpv4PodCIDRList` または `clusterPoolIpv4MaskSize` を変更しないでください。詳細については、「Expanding the cluster pool」を参照してください。
- kube-proxy 置き換えモードで Cilium を実行している場合は、Helm 値に `kubeProxyReplacement: "true"` を設定し、Cilium と同じノードで既存の kube-proxy デプロイが実行されていないことを確認します。
- 以下の例では、Cilium が L7 ネットワークポリシーとイングレスに使用する Envoy Layer 7 (L7) プロキシを無効にしています。詳細については、「the section called “ネットワークポリシーを設定する”」および「the section called “Cilium Ingress および Cilium Gateway の概要”」を参照してください。
- 以下の例では、サービストラフィック分散がお使いのサービス用に設定した場合に正しく機能するように、`loadBalancer.serviceTopology: true` を設定しています。詳細については、「the section called “サービストラフィック分散の設定”」を参照してください。
- Cilium の Helm 値の完全なリストについては、Cilium ドキュメントの「Helm reference」を参照してください。

```
affinity:
  nodeAffinity:
    requiredDuringSchedulingIgnoredDuringExecution:
      nodeSelectorTerms:
      - matchExpressions:
        - key: eks.amazonaws.com/compute-type
          operator: In
          values:
          - hybrid
ipam:
  mode: cluster-pool
  operator:
    clusterPoolIPv4MaskSize: 25
    clusterPoolIPv4PodCIDRList:
    - POD_CIDR
loadBalancer:
  serviceTopology: true
operator:
  affinity:
    nodeAffinity:
      requiredDuringSchedulingIgnoredDuringExecution:
        nodeSelectorTerms:
        - matchExpressions:
```

```
          - key: eks.amazonaws.com/compute-type
            operator: In
            values:
              - hybrid
  unmanagedPodWatcher:
    restart: false
loadBalancer:
  serviceTopology: true
envoy:
  enabled: false
kubeProxyReplacement: "false"
```

2. クラスターに Cilium をインストールします。
   - CILIUM_VERSION を Cilium バージョン (1.17.9-0 または 1.18.3-0 など) に置き換えます。Cilium のマイナーバージョンに最新のパッチバージョンを使用することをお勧めします。
   - ノードが、選択したバージョンのカーネル要件を満たしていることを確認します。Cilium v1.18.3 には Linux カーネル >= 5.10 が必要です。
   - 特定の kubeconfig ファイルを使用している場合はHelm install コマンドで --kubeconfig フラグを使用します。

```
helm install cilium oci://public.ecr.aws/eks/cilium/cilium \
    --version CILIUM_VERSION \
    --namespace kube-system \
    --values cilium-values.yaml
```

3. 以下のコマンドを使用して、Cilium のインストールが成功したことを確認します。cilium-operator デプロイと、各ハイブリッドノードで実行されている cilium-agent が確認できるはずです。また、ハイブリッドノードのステータスは Ready になっているはずです。ポッド CIDR をオンプレミスネットワークにアドバタイズするように Cilium BGP を設定する方法については、「the section called “BGP を設定する”」に進んでください。

```
kubectl get pods -n kube-system
```

```
NAME                              READY   STATUS    RESTARTS   AGE
cilium-jjjn8                      1/1     Running   0          11m
cilium-operator-d4f4d7fcb-sc5xn   1/1     Running   0          11m
```

```
kubectl get nodes
```

```
NAME                    STATUS   ROLES    AGE   VERSION
mi-04a2cf999b7112233    Ready    <none>   19m   v1.31.0-eks-a737599
```

ハイブリッドノードで Cilium をアップグレードする

Cilium デプロイをアップグレードする前に、Cilium アップグレードのドキュメントとアップグレードに関する注意事項をよく確認し、対象となる Cilium バージョンの変更について理解してください。

1. コマンドライン環境に helm CLI がインストールされていることを確認します。インストール手順については「Helm のドキュメント」を参照してください。
2. Cilium アップグレードのプリフライトチェックを実行してください。CILIUM_VERSION を対象となる Cilium バージョンに置き換えます。Cilium のマイナーバージョンに対して、最新のパッチバージョンを実行することをお勧めします。特定の Cilium マイナーリリースの最新のパッチリリースはCilium ドキュメントの「安定リリースのセクション」で確認できます。

```
helm install cilium-preflight oci://public.ecr.aws/eks/cilium/cilium --version
 CILIUM_VERSION \
  --namespace=kube-system \
  --set preflight.enabled=true \
  --set agent=false \
  --set operator.enabled=false
```

3. cilium-preflight.yaml の適用後は READY Pod の数が実行中の Cilium Pod の数と同じであることを確認します。

```
kubectl get ds -n kube-system | sed -n '1p;/cilium/p'
```

```
NAME                      DESIRED   CURRENT   READY   UP-TO-DATE   AVAILABLE   NODE
 SELECTOR   AGE
cilium                    2         2         2       2            2           <none>
          1h20m
cilium-pre-flight-check   2         2         2       2            2           <none>
          7m15s
```

4. READY ポッドの数が同じになったら、Cilium プリフライトデプロイも READY 1/1 とマークされていることを確認してください。READY 0/1 と表示された場合はアップグレードを続行する前に、「CNP の検証」セクションを参照しながらデプロイの問題を解決してください。

```
kubectl get deployment -n kube-system cilium-pre-flight-check -w
```

```
NAME                      READY   UP-TO-DATE   AVAILABLE   AGE
cilium-pre-flight-check   1/1     1            0           12s
```

5. プリフライトを削除する

```
helm uninstall cilium-preflight --namespace kube-system
```

6. helm upgrade コマンドを実行する場合は、事前にデプロイの値を existing-cilium-values.yaml に保存します。あるいは、そのアップグレードコマンドを実行する際に、お使いの設定に対して --set コマンドラインオプションを使用します。アップグレード操作は Cilium ConfigMap を上書きするため、アップグレード時に設定値を渡すことが極めて重要です。

```
helm get values cilium --namespace kube-system -o yaml > existing-cilium-values.yaml
```

7. 通常のクラスターオペレーションではすべての Cilium コンポーネントが同じバージョンを実行している必要があります。以下のステップではある安定版リリースからより新しい安定版リリースへと、すべてのコンポーネントをアップグレードする方法について説明します。あるマイナーリリースから別のマイナーリリースにアップグレードする場合はまず既存の Cilium マイナーバージョンの最新のパッチリリースにアップグレードすることをお勧めします。中断を最小限に抑えるには upgradeCompatibility オプションを、このクラスターに最初にインストールされた Cilium のバージョンに設定します。

```
helm upgrade cilium oci://public.ecr.aws/eks/cilium/cilium --version CILIUM_VERSION \
  --namespace kube-system \
  --set upgradeCompatibility=1.X \
  -f existing-cilium-values.yaml
```

8. (オプション) 問題があってアップグレードをロールバックする必要がある場合は次のコマンドを実行してください。

```
helm history cilium --namespace kube-system
helm rollback cilium [REVISION] --namespace kube-system
```

ハイブリッドノードから Cilium を削除する

1. 以下のコマンドを実行して、クラスターからすべての Cilium コンポーネントをアンインストールします。なお、CNI をアンインストールすると、ノードと Pod の正常性に影響する可能性があるため、本番稼働用クラスターでは実行しないでください。

```
helm uninstall cilium --namespace kube-system
```

   CNI がクラスターから削除されても、Cilium によって設定されたインターフェイスとルートはデフォルトでは削除されません。詳細については「GitHub の問題」を参照してください。
2. ディスク上の設定ファイルとリソースをクリーンアップするには標準的な設定ディレクトリを使用している場合はGitHub の Cilium リポジトリの cni-uninstall.sh スクリプトに示されているようにファイルを削除できます。
3. Cilium カスタムリソース定義 (CRD クラスターから削除するには以下のコマンドを実行してください。

```
kubectl get crds -oname | grep "cilium" | xargs kubectl delete
```

ハイブリッドノードのアドオンを構成する

このページでは、Amazon EKS Hybrid Nodes で AWS アドオンとコミュニティアドオンを実行する際の考慮事項について説明します。Amazon EKS アドオンと、クラスターからアドオンを作成、アップグレード、削除するプロセスの詳細については、「the section called “Amazon EKS アドオン”」を参照してください。このページに特に明記されていない限り、Amazon EKS アドオンを作成、アップグレード、削除するプロセスは、ハイブリッドノードを備える Amazon EKS クラスターでも、AWS クラウドでノードが実行される Amazon EKS クラスターでも同じです。このページに記載のアドオンについてのみ、Amazon EKS Hybrid Nodes との互換性が検証されています。

以下の AWS アドオンは、Amazon EKS Hybrid Nodes と互換性があります。

AWS アドオン	互換性のあるアドオンバージョン
kube-proxy	v1.25.14-eksbuild.2 以降
CoreDNS	v1.9.3-eksbuild.7 以降

AWS アドオン	互換性のあるアドオンバージョン
AWS オープンテレメトリー用ディストリビューター (ADOT)	v0.102.1-eksbuild.2 以降
CloudWatch Observability エージェント	v2.2.1-eksbuild.1 以降
EKS Pod Identity エージェント	• v1.3.3-eksbuild.1 以降 (Bottlerocket 以外) • v1.3.7-eksbuild.2 以降 (Bottlerocket)
ノードモニタリングエージェント	v1.2.0-eksbuild.1 以降
CSI スナップショットコントローラー	v8.1.0-eksbuild.1 以降
AWS プライベート CA の Connector for Kubernetes	v1.6.0-eksbuild.1 以降
Amazon FSx CSI ドライバー	v1.7.0-eksbuild.1 以降
AWS Secrets Store CSI Driver プロバイダー	v2.1.1-eksbuild.1 以降

以下のコミュニティアドオンは、Amazon EKS Hybrid Nodes と互換性があります。コミュニティアドオンの詳細については、「the section called “コミュニティアドオン”」を参照してください。

コミュニティアドオン	互換性のあるアドオンバージョン
Kubernetes メトリクスサーバー	v0.7.2-eksbuild.1 以降
cert-manager	v1.17.2-eksbuild.1 以降
Prometheus Node Exporter	v1.9.1-eksbuild.2 以降
kube-state-metrics	v2.15.0-eksbuild.4 以降
外部 DNS	v0.19.0-eksbuild.1 以降

上記の表の Amazon EKS アドオンに加えて、Amazon Managed Service for Prometheus コレクター、およびアプリケーションイングレス (HTTP) およびロードバランシング (TCP/UDP) 用の AWS ロードバランサーコントローラーはハイブリッドノードと互換性があります。

AWS アドオンとコミュニティアドオンには、Amazon EKS Hybrid Nodes と互換性のないものがあります。こうしたアドオンの最新バージョンには、デフォルトの `eks.amazonaws.com/compute-type: hybrid` ラベルをハイブリッドノードに適用するにあたってアンチアフィニティルールがあります。これにより、クラスターにデプロイされたときにハイブリッドノードで実行されるのが防止されます。ハイブリッドノードと AWS クラウドで実行されているノードの両方を備えたクラスターがある場合、クラスター内のこれらのアドオンを AWS クラウドで実行されているノードにデプロイできます。Amazon VPC CNI はハイブリッドノードと互換性がなく、Cilium と Calico は Amazon EKS Hybrid Nodes のコンテナネットワークインターフェイス (CNI としてサポートされています。詳細については「the section called “CNI を設定する”」を参照してください。

AWSアドオン

以降のセクションでは、ハイブリッドノード上の互換性のある AWS アドオンを実行する場合と、他の Amazon EKS コンピューティングタイプで同じアドオンを実行する場合の違いについて説明します。

kube-proxy と CoreDNS

AWS CLI によるものも含め AWS API と AWS SDK で EKS クラスターを作成すると、EKS はデフォルトで kube-proxy と CoreDNS をセルフマネージドアドオンとしてインストールします。こうしたアドオンは、クラスターの作成後に Amazon EKS アドオンで上書きできます。the section called “kube-proxy” および the section called “CoreDNS” の詳細についてはEKS ドキュメントを参照してください。ハイブリッドノードと AWS クラウド内のノードの両方がある混合モードのクラスターを実行している場合、AWS ではハイブリッドノードに少なくとも 1 つの CoreDNS レプリカがあり、AWS クラウド内のノードに少なくとも 1 つの CoreDNS レプリカがあるようにすることをお勧めします。設定手順については、「the section called “CoreDNS レプリカを設定する”」を参照してください。

CloudWatch Observability エージェント

CloudWatch Observability エージェントオペレーターは、ウェブフックを使用します。ハイブリッドノードでオペレーターを実行する場合は、オンプレミスネットワークでオンプレミスポッド CIDR をルーティングし、リモートポッドネットワークで EKS クラスターを設定する必要があります。詳細については、「Configure webhooks for hybrid nodes」を参照してください。

ノードレベルのメトリクスはクラウドコンテナインサイトを見る がノードレベルのメトリクスに関してインスタンスメタデータサービス (IMDS の可用性に依存するため、ハイブリッドノードでは利用できません。クラスター、ワークロード、ポッド、コンテナレベルのメトリクスはハイブリッドノードで利用できます。

「Amazon CloudWatch オベサビリティ を使用して Amazon CloudWatch エージェントをインストールする」で説明されているステップに従ってアドオンをインストールした後、エージェントがハイブリッドノードで正常に実行できるようにするにはアドオンマニフェストを更新する必要があります。以下に示すように、クラスターの amazoncloudwatchagents リソースを編集して RUN_WITH_IRSA 環境変数を追加します。

```
kubectl edit amazoncloudwatchagents -n amazon-cloudwatch cloudwatch-agent
```

```
apiVersion: v1
items:
- apiVersion: cloudwatch.aws.amazon.com/v1alpha1
  kind: AmazonCloudWatchAgent
  metadata:
    ...
    name: cloudwatch-agent
    namespace: amazon-cloudwatch
    ...
  spec:
    ...
    env:
    - name: RUN_WITH_IRSA # <-- Add this
      value: "True" # <-- Add this
    - name: K8S_NODE_NAME
      valueFrom:
        fieldRef:
          fieldPath: spec.nodeName
          ...
```

Amazon Managed Prometheus ハイブリッドノード用マネージドコレクター

Amazon Managed Service for Prometheus (AMP) マネージドコレクターは Amazon EKS クラスター内のリソースからメトリクスを検出して収集するスクレイパーで構成されています。AMP はスクレイパーを自動的に管理するため、インスタンス、エージェント、スクレイパーをユーザーが管理する必要はありません。

AMP マネージドコレクターはハイブリッドノードに固有の追加設定を行わなくても使用できます。ただし、VPC からリモートポッドネットワーク CIDR へのルートや、オンプレミスファイアウォールで開いているポートなど、ハイブリッドノード上のアプリケーションのメトリクスエンドポイントが VPC から到達可能である必要があります。さらに、クラスターにはプライベートクラスターエンドポイントアクセスが必要です。

「Amazon Managed Service for Prometheus ユーザーガイド」の「AWS マネージドコレクターの使用」のステップに従います。

AWS オープンテレメトリー用ディストリビューター (ADOT)

AWS Distro for OpenTelemetry (ADOT) アドオンを使用して、ハイブリッドノードで実行されているアプリケーションからメトリクス、ログ、トレースデータを収集できます。ADOT は、Collector Custom Resource リクエストの変更と検証にアドミッションウェブフックを使用します。ハイブリッドノードで ADOT オペレーターを実行する場合は、オンプレミスネットワークでオンプレミスポッド CIDR をルーティングし、リモートポッドネットワークで EKS クラスターを設定する必要があります。詳細については、「Configure webhooks for hybrid nodes」を参照してください。

AWS オープンテレメトリー用ディストリビューター ドキュメントの「EKS アドオンを使用した AWSオープンテレメトリー用ディストリビューター の開始方法」のステップに従ってください。

AWS ロードバランサーコントローラー

ハイブリッドノード上のワークロードのターゲットタイプが `ip` の場合、AWS Load Balancer Controller を Application Load Balancer (ALB) や Network Load Balancer (NLB) と共に使用できます。ALB または NLB で使用される IP ターゲットは、AWS からルーティング可能である必要があります。AWS Load Balancer Controller も、ウェブフックを使用します。ハイブリッドノードで AWS Load Balancer Controller オペレーターを実行する場合は、オンプレミスネットワークでオンプレミスポッド CIDR をルーティングし、リモートポッドネットワークで EKS クラスターを設定する必要があります。詳細については、「Configure webhooks for hybrid nodes」を参照してください。

AWS ロードバランサーコントローラー をインストールするには「the section called “AWS Application Load Balancer”」または「the section called “AWS Network Load Balancer”」のステップに従います。

ALB でイングレスを使用する場合は以下の注釈を指定する必要があります。詳細については「the section called “アプリケーション負荷分散”」を参照してください。

```
alb.ingress.kubernetes.io/target-type: ip
```

NLB で負荷分散を使用する場合は以下の注釈を指定する必要があります。詳細については「the section called “ネットワーク負荷分散”」を参照してください。

```
service.beta.kubernetes.io/aws-load-balancer-type: "external"
service.beta.kubernetes.io/aws-load-balancer-nlb-target-type: "ip"
```

EKS Pod Identity エージェント

> **Note**
>
> Bottlerocket を実行しているハイブリッドノードに EKS Pod Identity Agent アドオンを正常にデプロイするには、Bottlerocket のバージョンが v1.39.0 以上であることを確認します。Pod Identity エージェントのハイブリッドノード環境では、以前の Bottlerocket バージョンがサポートされていません。

元の Amazon EKS Pod Identity エージェント DaemonSet は必要な AWS 認証情報を取得するために、ノードの EC2 IMDS を利用します。IMDS はハイブリッドノードでは使用できないため、バージョン 1.3.3-eksbuild.1 以降では必要に応じて Pod Identity Agent アドオンに必要な認証情報をマウントする DaemonSet がデプロイされます。Bottlerocket を実行するハイブリッドノードでは、認証情報をマウントするために別の方法が必要です。バージョン 1.3.7-eksbuild.2 以降では、必要に応じて Pod Identity Agent アドオンに Bottlerocket ハイブリッドノードを対象とする専用の DaemonSet がデプロイされます。以下のセクションでは、オプションの DaemonSet を有効にするプロセスについて説明します。

Ubuntu/RHEL/AL2023

1. Ubuntu/RHEL/Al2023 ハイブリッドノードで Pod Identity エージェントを使用するには、以下に示すように nodeadm 設定のハイブリッドセクションで `enableCredentialsFile: true` を設定します。

```
apiVersion: node.eks.aws/v1alpha1
kind: NodeConfig
spec:
    hybrid:
        enableCredentialsFile: true # <-- Add this
```

これにより、eks-pod-identity-agent ポッドで使用される認証情報ファイルをノード上の /eks-hybrid/.aws/credentials に作成するように nodeadm が設定されます。この認証情報ファイルには定期的に更新される一時的な AWS 認証情報が含まれます。

2. 各ノードの nodeadm 設定を更新したら、nodeConfig.yaml で以下の nodeadm init コマンドを実行して、ハイブリッドノードを Amazon EKS クラスターに結合します。ノードが以前にクラスターに参加していた場合でも、nodeadm init コマンドを再度実行してください。

```
nodeadm init -c file://nodeConfig.yaml
```

3. AWS CLI または AWS マネジメントコンソール のいずれかを使用し、ハイブリッドノードのサポートを有効にして eks-pod-identity-agent をインストールします。

   a. AWS CLI: クラスターの管理に使用しているマシンから以下のコマンドを実行して、ハイブリッドノードのサポートを有効にして eks-pod-identity-agent をインストールします。my-cluster を自分のクラスター名に置き換えます。

```
aws eks create-addon \
    --cluster-name my-cluster \
    --addon-name eks-pod-identity-agent \
    --configuration-values '{"daemonsets":{"hybrid":{"create": true}}}'
```

   b. AWS マネジメントコンソール: AWS コンソールから Pod Identity Agent アドオンをインストールする場合は、オプションの設定に以下を追加して、ハイブリッドノードを対象とする DaemonSet をデプロイします。

```
{"daemonsets":{"hybrid":{"create": true}}}
```

Bottlerocket

1. Bottlerocket ハイブリッドノードで Pod Identity エージェントを使用するには、「the section called “Bottlerocket を実行しているハイブリッドノードの接続”」で説明されているように、Bottlerocket ブートストラップコンテナのユーザーデータに使用されるコマンドに --enable-credentials-file=true フラグを追加します。

   a. SSM 認証情報プロバイダーを使用している場合、コマンドは次のようになります。

```
eks-hybrid-ssm-setup --activation-id=<activation-id> --activation-
code=<activation-code> --region=<region> --enable-credentials-file=true
```

b. IAM Roles Anywhere 認証情報プロバイダーを使用している場合、コマンドは次のようになります。

```
eks-hybrid-iam-ra-setup --certificate=<certificate> --key=<private-key> --enable-credentials-file=true
```

これによりブートストラップスクリプトが設定され、/var/eks-hybrid/.aws/credentials で eks-pod-identity-agent ポッドで使用される認証情報ファイルがノード上に作成されます。この認証情報ファイルには定期的に更新される一時的な AWS 認証情報が含まれます。

2. AWS CLI または AWS マネジメントコンソール のいずれかを使用し、Bottlerocket ハイブリッドノードのサポートを有効にして eks-pod-identity-agent をインストールします。

a. AWS CLI: クラスターの管理に使用しているマシンから、以下のコマンドを実行して eks-pod-identity-agent をインストールし、Bottlerocket ハイブリッドノードのサポートを有効にします。my-cluster を自分のクラスター名に置き換えます。

```
aws eks create-addon \
    --cluster-name my-cluster \
    --addon-name eks-pod-identity-agent \
    --configuration-values '{"daemonsets":{"hybrid-bottlerocket":{"create":
 true}}}'
```

b. AWS マネジメントコンソール: AWS コンソールから Pod Identity Agent アドオンをインストールする場合は、オプションの設定に以下を追加して、Bottlerocket ハイブリッドノードを対象とする DaemonSet をデプロイします。

```
{"daemonsets":{"hybrid-bottlerocket":{"create": true}}}
```

CSI スナップショットコントローラー

バージョン v8.1.0-eksbuild.2 以降、CSI スナップショットコントローラーアドオンはハイブリッドノードにソフトアンチアフィニティルールを適用し、コントローラーの deployment を Amazon EKS コントロールプレーンと同じ AWS リージョンの EC2 で実行することを優先します。deployment を Amazon EKS コントロールプレーンと同じ AWS リージョンに配置することで、レイテンシーが改善されます。

コミュニティアドオン

以降のセクションでは、ハイブリッドノード上の互換性のあるコミュニティアドオンを実行する場合と、他の Amazon EKS コンピューティングタイプで同じアドオンを実行する場合の違いについて説明します。

Kubernetes メトリクスサーバー

コントロールプレーンは、Metrics Server のポッド IP (あるいは hostNetwork が有効になっている場合にはノード IP) に到達しなければなりません。このため、hostNetwork モードで Metrics Server を実行しない限り、Amazon EKS クラスターを作成するときにリモートポッドネットワークを設定し、ポッド IP アドレスをルーティング可能にする必要があります。ポッド IP アドレスをルーティング可能にする場合によく使用される方法の 1 つに、CNI でボーダーゲートウェイプロトコル (BGP) を実装することがあります。

cert-manager

`cert-manager` はウェブフックを使用します。ハイブリッドノードで `cert-manager` を実行する場合は、オンプレミスネットワークでオンプレミスポッド CIDR をルーティングし、リモートポッドネットワークで EKS クラスターを設定する必要があります。詳細については、「Configure webhooks for hybrid nodes」を参照してください。

ハイブリッドノード用のウェブフックを設定する

このページでは、ハイブリッドノードでウェブフックを実行する場合の考慮事項について詳しく説明します。ウェブフックは、AWS Load Balancer Controller や CloudWatch Observability エージェントといった Kubernetes アプリケーションとオープンソースプロジェクトで実行時にミューテーション機能と検証機能を利用する目的で使用されます。

ルーティング可能なポッドネットワーク

オンプレミスネットワークでオンプレミスポッド CIDR をルーティング可能にできる場合、ハイブリッドノードでウェブフックを実行できます。オンプレミスネットワークでオンプレミスポッド CIDR をルーティング可能にするために使用できる手法がいくつかあります。具体的には、ボーダーゲートウェイプロトコル (BGP)、静的ルート、その他のカスタムルーティングソリューションなどです。BGP はカスタムまたは手動でルート設定する必要がある代替ソリューションよりもスケーラブルで管理しやすいため、推奨されるソリューションです。AWS は、ポッド CIDR をアドバタイズするための Cilium と Calico の BGP 機能をサポートしています。詳細については、「the section called “CNI を設定する”」および「the section called “ルーティング可能なリモートポッド CIDR”」を参照してください。

ルーティング不可能なポッドネットワーク

オンプレミスポッド CIDR をオンプレミスネットワークでルーティング可能にすることができず、ウェブフックを実行する必要がある場合、すべてのウェブフックをハイブリッドノードと同じ EKS クラスターのクラウドノードで実行することが推奨されます。

混合モードクラスターの考慮事項

混合モードクラスターは、ハイブリッドノードと AWS クラウドで実行中のノードの両方を備えた EKS クラスターであると定義されています。混合モードクラスターを実行する場合は、以下の推奨事項を検討してください。

- AWS Cloud 内のノードでは VPC CNI を実行し、ハイブリッドノードでは Cilium または Calico を実行します。AWS では、AWS Cloud 内のノードで Cilium と Calico を実行する操作はサポートされていません。
- AWS Cloud 内のノードで動作するようにウェブフックを設定します。AWS およびコミュニティアドオンのウェブフックを設定する方法については、「the section called “アドオン用のウェブフックを設定する”」を参照してください。
- アプリケーションの要件により AWS Cloud 内のノードで実行中のポッドがハイブリッドノードで実行中のポッドと直接通信 (「東西通信」) しなければならず、かつ AWS Cloud 内のノードでは VPC CNI が使用され、ハイブリッドノードでは Cilium または Calico が使用されている場合は、オンプレミスネットワークでオンプレミスポッド CIDR がルーティング可能である必要があります。
- AWS クラウドのノードで少なくとも 1 つの CoreDNS レプリカを実行し、ハイブリッドノードで少なくとも 1 つの CoreDNS レプリカを実行してください。
- サービストラフィック分散を設定し、Service トラフィックを発信元のゾーン内に留めます。サービストラフィック分散の詳細については、「the section called “サービストラフィック分散の設定”」を参照してください。
- ハイブリッドノードで実行されているワークロードトラフィックに AWS Application Load Balancer (ALB) または Network Load Balancer (NLB) を使用している場合、ALB または NLB と使用されている IP ターゲットを AWS からルーティング可能である必要があります。
- Metrics Server アドオンでは、EKS コントロールプレーンから Metrics Server ポッド IP アドレスへの接続が必要になります。ハイブリッドノードで Metrics Server アドオンを実行している場合は、オンプレミスネットワークでオンプレミスポッド CIDR をルーティングできる必要があります。
- Amazon Managed Service for Prometheus (AMP) マネージドコレクターを使用してハイブリッドノードに関するメトリクスを収集するには、オンプレミスネットワークでオンプレミスポッド

CIDR をルーティングできる必要があります。または、EKS コントロールプレーンメトリクスおよび AWS クラウドで実行されているリソースに AMP マネージドコレクターを使用し、AWS Distro for OpenTelemetry (ADOT) アドオンを使用してハイブリッドノードのメトリクスを収集することもできます。

混合モードクラスターの設定

クラスターで実行中のミューテーション用と検証用のウェブフックを表示するには、クラスターの EKS コンソールの [リソース] パネルで [拡張機能] リソースタイプを表示するか、以下のコマンドを使用します。EKS は、クラスターオブザーバビリティダッシュボードにウェブフックメトリクスもレポートします。詳細については、「the section called “オブザーバビリティダッシュボード”」を参照してください。

```
kubectl get mutatingwebhookconfigurations
```

```
kubectl get validatingwebhookconfigurations
```

サービストラフィック分散の設定

混合モードクラスターを実行する際、サービストラフィック分散を使用してサービストラフィックを発信元のゾーン内に留めることが推奨されます。サービストラフィック分散 (EKS では Kubernetes バージョン 1.31 以降で利用可能) は、Topology Aware Routing よりも予測性に優れているため、推奨されるソリューションです。サービストラフィック分散を使用すると、ゾーン内の正常なエンドポイントはそのゾーンのすべてのトラフィックが受信されます。Topology Aware Routing を使用すると、カスタムルーティングを適用するために各サービスはそのゾーンの複数の条件を満たす必要があります。そうでない場合、トラフィックはすべてのエンドポイントに均等にルーティングされます。

Cilium を CNI として使用している場合は、`enable-service-topology` を `true` に設定して CNI を実行し、Service Traffic Distribution を有効にする必要があります。この設定は Helm インストールフラグ `--set loadBalancer.serviceTopology=true` で渡すことができます。あるいは、Cilium CLI コマンド `cilium config set enable-service-topology true` を使用して既存のインストールを更新することもできます。既存のインストールの設定を更新した場合は、その後に各ノードで実行されている Cilium エージェントを再起動する必要があります。

CoreDNS サービスにサービストラフィック分散を設定する方法の例は、次のセクションで示されています。意図しない環境間トラフィックを回避するには、クラスター内のすべてのサービスに同じ設定を有効にすることが推奨されます。

CoreDNS レプリカを設定する

ハイブリッドノードと AWS クラウド内のノードの両方を備える混合モードのクラスターを実行している場合は、ハイブリッドノードと AWS クラウド内のノードのそれぞれに CoreDNS レプリカを 1 つ以上作成しておくことが推奨されます。混合モードクラスターの設定でレイテンシーとネットワークの問題を回避するには、Service Traffic Distribution を使って最も近くにある CoreDNS レプリカを優先するように CoreDNS サービスを設定します。

Service Traffic Distribution (EKS で Kubernetes バージョン 1.31 以降で使用可能) は、Topology Aware Routing よりも予測性に優れているため、推奨される方法です。Service Traffic Distribution では、ゾーン内の正常なエンドポイントは、そのゾーンのすべてのトラフィックを受信します。Topology Aware Routing の場合、カスタムルーティングを適用するには各サービスがそのゾーンの複数の条件を満たしている必要があります。満たしていない場合、トラフィックはすべてのエンドポイントに均等にルーティングされます。以下のステップで、Service Traffic Distribution を設定していきます。

Cilium を CNI として使用している場合は、`enable-service-topology` を `true` に設定して CNI を実行し、Service Traffic Distribution を有効にする必要があります。この設定は Helm インストールフラグ `--set loadBalancer.serviceTopology=true` で渡すことができます。あるいは、Cilium CLI コマンド `cilium config set enable-service-topology true` を使用して既存のインストールを更新することもできます。既存のインストールの設定を更新した場合は、その後に各ノードで実行されている Cilium エージェントを再起動する必要があります。

1. ハイブリッドノードごとにトポロジゾーンラベルを追加します。例えば、`topology.kubernetes.io/zone: onprem` とします。または、nodeadm 設定でラベルを指定することで、`nodeadm init` フェーズのラベルを設定できます。「the section called “kubelet をカスタマイズするためのノード設定 (オプション)”」を参照してください。なお、AWS Cloud で実行されているノードは、自身のアベイラビリティーゾーン (AZ) に対応するトポロジゾーンラベルを自動的に適用します。

```
kubectl label node hybrid-node-name topology.kubernetes.io/zone=zone
```

2. トポロジゾーンキーを使用して `podAntiAffinity` を CoreDNS デプロイに追加します。または、EKS アドオンを使用してインストール時に CoreDNS デプロイを設定することもできます。

```
kubectl edit deployment coredns -n kube-system
```

```
spec:
```

```
  template:
    spec:
      affinity:
       ...
        podAntiAffinity:
          preferredDuringSchedulingIgnoredDuringExecution:
          - podAffinityTerm:
              labelSelector:
                matchExpressions:
                - key: k8s-app
                  operator: In
                  values:
                  - kube-dns
              topologyKey: kubernetes.io/hostname
            weight: 100
          - podAffinityTerm:
              labelSelector:
                matchExpressions:
                - key: k8s-app
                  operator: In
                  values:
                  - kube-dns
              topologyKey: topology.kubernetes.io/zone
            weight: 50
      ...
```

3. trafficDistribution: PreferClose 設定を kube-dns サービス設定に追加し、サービストラフィック分散を有効にします。

```
kubectl patch svc kube-dns -n kube-system --type=merge -p '{
  "spec": {
    "trafficDistribution": "PreferClose"
  }
}'
```

4. kube-dns サービスのエンドポイントスライスを表示することで、Service Traffic Distribution が有効になっていることを確認できます。エンドポイントスライスにはトポロジーゾーンのラベルの hints が表示されているはずです。これで、Service Traffic Distribution が有効になっていることが確認できます。各エンドポイントアドレスの hints が表示されなければ、Service Traffic Distribution は有効になっていません。

```
kubectl get endpointslice -A | grep "kube-dns"
```

```
kubectl get endpointslice kube-dns-<id>  -n kube-system -o yaml
```

```
addressType: IPv4
apiVersion: discovery.k8s.io/v1
endpoints:
- addresses:
  - <your-hybrid-node-pod-ip>
  hints:
    forZones:
    - name: onprem
  nodeName: <your-hybrid-node-name>
  zone: onprem
- addresses:
  - <your-cloud-node-pod-ip>
  hints:
    forZones:
    - name: us-west-2a
  nodeName: <your-cloud-node-name>
  zone: us-west-2a
```

アドオン用のウェブフックを設定する

以下のアドオンは、ウェブフックを使用しており、ハイブリッドノードでの使用がサポートされています。

- AWS Load Balancer Controller
- Amazon CloudWatch オベサビリティ エージェント
- AWS オープンテレメトリー用ディストリビューター (ADOT)
- `cert-manager`

これらのアドオンで使用されるウェブフックを AWS クラウド内のノードで実行するように設定する方法については、以下のセクションを参照してください。

AWS Load Balancer Controller

AWS Load Balancer Controller を混合モードクラスターの設定で使用するには、このコントローラーを AWS クラウドのノードで実行する必要があります。それには、Helm 値の設定に以下を追加するか、EKS アドオン設定を使用して値を指定します。

```
affinity:
  nodeAffinity:
    requiredDuringSchedulingIgnoredDuringExecution:
      nodeSelectorTerms:
      - matchExpressions:
        - key: eks.amazonaws.com/compute-type
          operator: NotIn
          values:
          - hybrid
```

Amazon CloudWatch オベサビリティ エージェント

CloudWatch Observability Agent アドオンには、ウェブフックを使用する Kubernetes Operator があります。混合モードクラスター設定の AWS Cloud 内のノードでオペレーターを実行するには、CloudWatch Observability エージェントオペレーター設定を編集します。インストール時に Helm と EKS アドオンを使用してオペレーターアフィニティを設定することはできません (「containers-roadmap issue #2431」を参照のこと)。

```
kubectl edit -n amazon-cloudwatch deployment amazon-cloudwatch-observability-
controller-manager
```

```
spec:
  ...
  template:
    ...
    spec:
      affinity:
        nodeAffinity:
          requiredDuringSchedulingIgnoredDuringExecution:
            nodeSelectorTerms:
            - matchExpressions:
              - key: eks.amazonaws.com/compute-type
                operator: NotIn
                values:
                - hybrid
```

AWS オープンテレメトリー用ディストリビューター (ADOT)

AWS Distro for OpenTelemetry (ADOT) アドオンには、ウェブフックを使用する Kubernetes Operator があります。混合モードクラスター設定の AWS クラウド内のノードでオペレーターを実行するには、Helm 値設定に以下を追加するか、EKS アドオン設定を使用して値を指定します。

```
affinity:
  nodeAffinity:
    requiredDuringSchedulingIgnoredDuringExecution:
      nodeSelectorTerms:
      - matchExpressions:
        - key: eks.amazonaws.com/compute-type
          operator: NotIn
          values:
          - hybrid
```

ポッド CIDR をオンプレミスネットワークでルーティングできない場合は、ADOT コレクターをハイブリッドノードで実行して、ハイブリッドノードとそこで実行されているワークロードからメトリクスをスクレイピングする必要があります。これを行うには、カスタムリソース定義 (CRD) を編集します。

```
kubectl -n opentelemetry-operator-system edit opentelemetrycollectors.opentelemetry.io
 adot-col-prom-metrics
```

```
spec:
  affinity:
    nodeAffinity:
      requiredDuringSchedulingIgnoredDuringExecution:
        nodeSelectorTerms:
        - matchExpressions:
          - key: eks.amazonaws.com/compute-type
            operator: In
            values:
            - hybrid
```

ADOT コレクター CRD 設定で各 `scrape_configs` に以下の `relabel_configs` を追加することで、ハイブリッドノードとそこで実行されているリソースからのみメトリクスをスクレイピングするように ADOT コレクターを設定できます。

```
relabel_configs:
  - action: keep
    regex: hybrid
    source_labels:
    - __meta_kubernetes_node_label_eks_amazonaws_com_compute_type
```

ADOT アドオンには、ADOT オペレーターウェブフックで使用される TLS 証明書に cert-manager をインストールするための前提条件があります。cert-manager はウェブフックも実行し、次の Helm 値設定を使用して AWS クラウドのノードで実行するように設定することができます。

```
affinity:
  nodeAffinity:
    requiredDuringSchedulingIgnoredDuringExecution:
      nodeSelectorTerms:
      - matchExpressions:
        - key: eks.amazonaws.com/compute-type
          operator: NotIn
          values:
          - hybrid
webhook:
  affinity:
    nodeAffinity:
      requiredDuringSchedulingIgnoredDuringExecution:
        nodeSelectorTerms:
        - matchExpressions:
          - key: eks.amazonaws.com/compute-type
            operator: NotIn
            values:
            - hybrid
cainjector:
  affinity:
    nodeAffinity:
      requiredDuringSchedulingIgnoredDuringExecution:
        nodeSelectorTerms:
        - matchExpressions:
          - key: eks.amazonaws.com/compute-type
            operator: NotIn
            values:
            - hybrid
startupapicheck:
  affinity:
    nodeAffinity:
      requiredDuringSchedulingIgnoredDuringExecution:
        nodeSelectorTerms:
        - matchExpressions:
          - key: eks.amazonaws.com/compute-type
            operator: NotIn
            values:
            - hybrid
```

cert-manager

cert-manager アドオンは、ウェブフックを実行し、以下の Helm 値設定を使用して AWS クラウド内のノードで実行するように設定することができます。

```
affinity:
  nodeAffinity:
    requiredDuringSchedulingIgnoredDuringExecution:
      nodeSelectorTerms:
      - matchExpressions:
        - key: eks.amazonaws.com/compute-type
          operator: NotIn
          values:
          - hybrid
webhook:
  affinity:
    nodeAffinity:
      requiredDuringSchedulingIgnoredDuringExecution:
        nodeSelectorTerms:
        - matchExpressions:
          - key: eks.amazonaws.com/compute-type
            operator: NotIn
            values:
            - hybrid
cainjector:
  affinity:
    nodeAffinity:
      requiredDuringSchedulingIgnoredDuringExecution:
        nodeSelectorTerms:
        - matchExpressions:
          - key: eks.amazonaws.com/compute-type
            operator: NotIn
            values:
            - hybrid
startupapicheck:
  affinity:
    nodeAffinity:
      requiredDuringSchedulingIgnoredDuringExecution:
        nodeSelectorTerms:
        - matchExpressions:
          - key: eks.amazonaws.com/compute-type
            operator: NotIn
            values:
```

```
            - hybrid
```

ハイブリッドノードのプロキシを設定する

オンプレミス環境でデータセンターまたはエッジ環境から出ていくトラフィックにプロキシサーバーを使用している場合は、プロキシサーバーを使用するように個別にノードとクラスターを設定する必要があります。

クラスター

クラスターでは、プロキシサーバーを使用するように kube-proxy を設定する必要があります。Amazon EKS クラスターの作成後に kube-proxy を設定する必要があります。

ノード

ノードでは、プロキシサーバーを使用するようにオペレーティングシステム、containerd、kubelet、Amazon SSM エージェントを設定する必要があります。こうした変更は、オペレーティングシステムイメージのビルドプロセス中に加えることも、各ハイブリッドノードで nodeadm init を実行する前に加えることもできます。

ノードレベルの設定

以下の設定は、オペレーティングシステムイメージに適用するか、各ハイブリッドノードで nodeadm init を実行する前に適用する必要があります。

containerd プロキシ設定

containerd は、Kubernetes のデフォルトのコンテナ管理ランタイムです。インターネットアクセスにプロキシを使用している場合は、Kubernetes と Amazon EKS に必要なコンテナイメージをプルできるように containerd を設定する必要があります。

次の内容で、各ハイブリッドノード上の /etc/systemd/system/containerd.service.d ディレクトリに「http-proxy.conf」という名前のファイルを作成します。proxy-domain および port を環境の値で置き換えます。

```
[Service]
Environment="HTTP_PROXY=http://proxy-domain:port"
Environment="HTTPS_PROXY=http://proxy-domain:port"
Environment="NO_PROXY=localhost"
```

ユーザーデータからの **containerd** 設定

このファイル用に containerd.service.d ディレクトリを作成する必要があります。systemd を再ロードして設定ファイルを選択すると、再起動しなくても済むようになります。AL2023 では、スクリプトを実行した時点で既にサービスが実行されている可能性があります。その場合は再起動も必要です。

```
mkdir -p /etc/systemd/system/containerd.service.d
echo '[Service]' > /etc/systemd/system/containerd.service.d/http-proxy.conf
echo 'Environment="HTTP_PROXY=http://proxy-domain:port"' >> /etc/systemd/system/
containerd.service.d/http-proxy.conf
echo 'Environment="HTTPS_PROXY=http://proxy-domain:port"' >> /etc/systemd/system/
containerd.service.d/http-proxy.conf
echo 'Environment="NO_PROXY=localhost"' >> /etc/systemd/system/containerd.service.d/
http-proxy.conf
systemctl daemon-reload
systemctl restart containerd
```

kubelet プロキシ設定

kubelet は、各 Kubernetes ノードで実行される Kubernetes ノードエージェントであり、そのノードで実行されているノードとポッドの管理を担当します。オンプレミス環境でプロキシを使用している場合は、Amazon EKS クラスターのパブリックエンドポイントまたはプライベートエンドポイントと通信できるように kubelet を設定する必要があります。

次の内容で、各ハイブリッドノードの /etc/systemd/system/kubelet.service.d/ ディレクトリに「http-proxy.conf」という名前のファイルを作成します。proxy-domain および port を環境の値で置き換えます。

```
[Service]
Environment="HTTP_PROXY=http://proxy-domain:port"
Environment="HTTPS_PROXY=http://proxy-domain:port"
Environment="NO_PROXY=localhost"
```

ユーザーデータからの **kubelet** 設定

このファイル用に kubelet.service.d ディレクトリを作成する必要があります。systemd を再ロードして設定ファイルを選択すると、再起動しなくても済むようになります。AL2023 では、スクリプトを実行した時点で既にサービスが実行されている可能性があります。その場合は再起動も必要です。

```
mkdir -p /etc/systemd/system/kubelet.service.d
echo '[Service]' > /etc/systemd/system/kubelet.service.d/http-proxy.conf
echo 'Environment="HTTP_PROXY=http://proxy-domain:port"' >> /etc/systemd/system/
kubelet.service.d/http-proxy.conf
echo 'Environment="HTTPS_PROXY=http://proxy-domain:port"' >> /etc/systemd/system/
kubelet.service.d/http-proxy.conf
echo 'Environment="NO_PROXY=localhost"' >> /etc/systemd/system/kubelet.service.d/http-
proxy.conf
systemctl daemon-reload
systemctl restart kubelet
```

ssm プロキシ設定

ssm は認証情報プロバイダーの 1 つであり、これを使用するとハイブリッドノードを初期化できます。ssm は、AWS での認証を実施し、kubelet によって使用される一時的な認証情報を生成します。オンプレミス環境でプロキシを使用し、ノードで ssm を認証情報プロバイダーとして使用している場合は、Amazon SSM サービスエンドポイントと通信できるように ssm を設定する必要があります。

オペレーティングシステムに応じて、各ハイブリッドノードに http-proxy.conf という名前でファイルを以下のパスに作成します。

- Ubuntu - /etc/systemd/system/snap.amazon-ssm-agent.amazon-ssm-agent.service.d/http-proxy.conf
- Amazon Linux 2023 および Red Hat Enterprise Linux - /etc/systemd/system/amazon-ssm-agent.service.d/http-proxy.conf

ファイルには次の内容を入力します。proxy-domain および port を環境の値で置き換えます。

```
[Service]
Environment="HTTP_PROXY=http://proxy-domain:port"
Environment="HTTPS_PROXY=http://proxy-domain:port"
Environment="NO_PROXY=localhost"
```

ユーザーデータからの **ssm** 設定

このファイル用に ssm システムサービスファイルディレクトリを作成する必要があります。ディレクトリパスは、ノードで使用されているオペレーティングシステムによって異なります。

- Ubuntu - `/etc/systemd/system/snap.amazon-ssm-agent.amazon-ssm-agent.service.d`
- Amazon Linux 2023 および Red Hat Enterprise Linux - `/etc/systemd/system/amazon-ssm-agent.service.d`

ノードで使用されているオペレーティングシステムに応じて、以下の restart コマンドの systemd サービス名を置き換えます。

- Ubuntu - `snap.amazon-ssm-agent.amazon-ssm-agent`
- Amazon Linux 2023 および Red Hat Enterprise Linux - `amazon-ssm-agent`

```
mkdir -p systemd-service-file-directory
echo '[Service]' > [.replaceable]#systemd-service-file-directory/http-proxy.conf
echo 'Environment="HTTP_PROXY=http://[.replaceable]#proxy-domain:port"' >> systemd-
service-file-directory/http-proxy.conf
echo 'Environment="HTTPS_PROXY=http://[.replaceable]#proxy-domain:port"' >>
 [.replaceable]#systemd-service-file-directory/http-proxy.conf
echo 'Environment="NO_PROXY=localhost"' >> [.replaceable]#systemd-service-file-
directory/http-proxy.conf
systemctl daemon-reload
systemctl restart [.replaceable]#systemd-service-name
```

オペレーティングシステムのプロキシ設定

インターネットアクセスにプロキシを使用している場合は、オペレーティングシステムのパッケージマネージャーからハイブリッドノードの依存関係をプルできるようにオペレーティングシステムを設定する必要があります。

Ubuntu

1. 次のコマンドでプロキシを使用するように snap を設定します。

```
sudo snap set system proxy.https=http://proxy-domain:port
sudo snap set system proxy.http=http://proxy-domain:port
```

2. apt のプロキシを有効にするには、/etc/apt/ ディレクトリに apt.conf という名前のファイルを作成します。proxy-domain と port を環境の値に置き換えます。

```
Acquire::http::Proxy "http://proxy-domain:port";
Acquire::https::Proxy "http://proxy-domain:port";
```

Amazon Linux 2023

1. プロキシを使用するように dnf を設定します。環境のプロキシドメイン値とポート値を使用してファイル /etc/dnf/dnf.conf を作成します。

```
proxy=http://proxy-domain:port
```

Red Hat Enterprise Linux

1. プロキシを使用するように yum を設定します。環境のプロキシドメイン値とポート値を使用してファイル /etc/yum.conf を作成します。

```
proxy=http://proxy-domain:port
```

IAM Roles Anywhere のプロキシ設定

IAM Roles Anywhere 認証情報プロバイダーサービスは、enableCredentialsFile フラグと共に IAM Roles Anywhere と使用する際に認証情報を更新する役割があります (the section called "EKS Pod Identity エージェント" を参照)。オンプレミス環境でプロキシを使用している場合は、IAM Roles Anywhere エンドポイントと通信できるようにサービスを設定する必要があります。

次の内容で、/etc/systemd/system/aws_signing_helper_update.service.d/ ディレクトリに「http-proxy.conf」という名前のファイルを作成します。proxy-domain および port を環境の値で置き換えます。

```
[Service]
Environment="HTTP_PROXY=http://proxy-domain:port"
Environment="HTTPS_PROXY=http://proxy-domain:port"
Environment="NO_PROXY=localhost"
```

クラスター全体の設定

このセクションの設定は、Amazon EKS クラスターを作成した後、および各ハイブリッドノードで `nodeadm init` を実行する前に適用する必要があります。

kube-proxy のプロキシ設定

Amazon EKS は、ハイブリッドノードがクラスターに参加すると、各ハイブリッドノードに DaemonSet として自動的に `kube-proxy` をインストールします。`kube-proxy` は、Amazon EKS クラスターのポッドによってバックアップされるサービス間のルーティングを有効にします。各ホストを設定するために、`kube-proxy` には Amazon EKS クラスターエンドポイントの DNS 解決が必要です。

1. 次のコマンドを使用して `kube-proxy` DaemonSet を編集する

```
kubectl -n kube-system edit ds kube-proxy
```

   これにより、設定されたエディタで `kube-proxy` DaemonSet 定義が開きます。

2. `HTTP_PROXY` および `HTTPS_PROXY` の環境変数を追加します。`NODE_NAME` 環境変数は設定に既に存在している必要があります。`proxy-domain` と `port` を環境の値に置き換えます。

```
containers:
  - command:
    - kube-proxy
    - --v=2
    - --config=/var/lib/kube-proxy-config/config - --hostname-override=$(NODE_NAME)
    env:
    - name: HTTP_PROXY
      value: http://proxy-domain:port
    - name: HTTPS_PROXY
      value: http://proxy-domain:port
    - name: NODE_NAME
      valueFrom:
        fieldRef:
          apiVersion: v1
          fieldPath: spec.nodeName
```

ハイブリッドノード向けに Cilium BGP を設定する

このトピックでは、Amazon EKS Hybrid Nodes の Cilium ボーダーゲートウェイプロトコル (BGP) を設定する方法について説明します。Cilium の BGP は Cilium BGP コントロールプレーンと呼ばれる機能で、この機能を使用すると、ポッドアドレスとサービスアドレスをオンプレミスネットワークにアドバタイズできます。これ以外にオンプレミスネットワークでポッド CIDR をルーティング可能にする方法については、「the section called “ルーティング可能なリモートポッド CIDR”」を参照してください。

Cilium BGP を設定する

前提条件

- 「the section called “CNI を設定する”」の手順に従って Cilium がインストールされていること。

手順

1. Cilium で BGP を使用してオンプレミスネットワークでポッドアドレスまたはサービスアドレスをアドバタイズするには、`bgpControlPlane.enabled: true` を使用して Cilium をインストールしている必要があります。既存の Cilium のデプロイで BGP を有効にする場合は、Cilium 演算子を再起動して BGP 設定を適用する必要があります (BGP が以前に有効になっていない場合)。Helm のインストール/アップグレードプロセスの一環として Cilium 演算子を再起動するには、Helm 値の `operator.rollOutPods` を `true` に設定します。

```
helm upgrade cilium oci://public.ecr.aws/eks/cilium/cilium \
  --namespace kube-system \
  --reuse-values \
  --set operator.rollOutPods=true \
  --set bgpControlPlane.enabled=true
```

2. Cilium オペレーターとエージェントが再起動されて実行中であることを確認します。

```
kubectl -n kube-system get pods --selector=app.kubernetes.io/part-of=cilium
```

```
NAME                               READY   STATUS    RESTARTS   AGE
cilium-grwlc                       1/1     Running   0          4m12s
cilium-operator-68f7766967-5nnbl   1/1     Running   0          4m20s
cilium-operator-68f7766967-7spfz   1/1     Running   0          4m20s
cilium-pnxcv                       1/1     Running   0          6m29s
cilium-r7qkj                       1/1     Running   0          4m12s
```

```
cilium-wxhfn                       1/1     Running   0          4m1s
cilium-z7hlb                       1/1     Running   0          6m30s
```

3. CiliumBGPClusterConfig を定義したファイルを cilium-bgp-cluster.yaml という名前で作成します。ネットワーク管理者から次の情報を取得する必要が生じる場合があります。
   - Cilium を実行しているノードの ASN を使用して、localASN を設定します。
   - オンプレミスルーターの ASN を使用して、peerASN を設定します。
   - Cilium を実行している各ノードがピアリングするオンプレミスルーター IP を使用して、peerAddress を設定します。

```
apiVersion: cilium.io/v2alpha1
kind: CiliumBGPClusterConfig
metadata:
  name: cilium-bgp
spec:
  nodeSelector:
    matchExpressions:
    - key: eks.amazonaws.com/compute-type
      operator: In
      values:
      - hybrid
  bgpInstances:
  - name: "rack0"
    localASN: NODES_ASN
    peers:
    - name: "onprem-router"
      peerASN: ONPREM_ROUTER_ASN
      peerAddress: ONPREM_ROUTER_IP
      peerConfigRef:
        name: "cilium-peer"
```

4. Cilium BGP クラスター設定をクラスターに適用します。

```
kubectl apply -f cilium-bgp-cluster.yaml
```

5. CiliumBGPPeerConfig リソースを使用して BGP ピア設定を定義するファイルを cilium-bgp-peer.yaml という名前で作成します。複数のピアが同じ設定を共有し、共通の CiliumBGPPeerConfig リソースを参照することができます。設定オプションの完全なリストについては、Cilium のドキュメントの「BGP Peer configuration」を参照してください。

次の Cilium ピア設定の値は、ピアリング先のオンプレミスルーターの値と一致する必要があります。

- holdTimeSeconds では、BGP ピアがキープアライブメッセージまたは更新メッセージを待機する時間を設定します。この時間を過ぎると、セッションのダウンが宣言されます。デフォルト値は 90 秒です。
- keepAliveTimeSeconds では、BGP ピアが引き続き到達可能で、BGP セッションがアクティブかどうかを設定します。デフォルト値は 30 秒です。
- restartTimeSeconds では、Cilium の BGP コントロールプレーンが再起動後に BGP セッションを再確立するまでにかかると見込まれる時間を設定します。デフォルト値は 120 秒です。

```
apiVersion: cilium.io/v2alpha1
kind: CiliumBGPPeerConfig
metadata:
  name: cilium-peer
spec:
  timers:
    holdTimeSeconds: 90
    keepAliveTimeSeconds: 30
  gracefulRestart:
    enabled: true
    restartTimeSeconds: 120
  families:
    - afi: ipv4
      safi: unicast
      advertisements:
        matchLabels:
          advertise: "bgp"
```

6. Cilium BGP ピア設定をクラスターに適用します。

```
kubectl apply -f cilium-bgp-peer.yaml
```

7. CiliumBGPAdvertisement リソースを使用してポッド CIDR をオンプレミスネットワークにアドバタイズするファイルを cilium-bgp-advertisement-pods.yaml という名前で作成します。
   - CiliumBGPAdvertisement リソースは、関連付けられるアドバタイズタイプと属性を定義するために使用されます。以下の例では、ポッド CIDR のみをアドバタイズするように Cilium を設定しています。サービスアドレスをアドバタイズするように Cilium を設定する方法の詳細に

ついては、「the section called “サービスタイプ LoadBalancer”」と「the section called “Cilium クラスター内ロードバランシング”」の例を参照してください。

- Cilium エージェントを実行している各ハイブリッドノードは、アップストリーム BGP 対応ルーターとピアリングします。以下の例のように、Cilium の `advertisementType` が `PodCIDR` に設定されていると、各ノードは所有するポッド CIDR 範囲をアドバタイズします。詳細については、Cilium のドキュメントの「BGP Advertisements configuration」を参照してください。

```
apiVersion: cilium.io/v2alpha1
kind: CiliumBGPAdvertisement
metadata:
  name: bgp-advertisement-pods
  labels:
    advertise: bgp
spec:
  advertisements:
    - advertisementType: "PodCIDR"
```

8. Cilium BGP アドバタイズ設定をクラスターに適用します。

```
kubectl apply -f cilium-bgp-advertisement-pods.yaml
```

9. `cilium bgp peers` コマンドを使用して、BGP ピアリングが Cilium CLI で機能していることを確認できます。出力には環境に応じた正しい値が表示され、セッションの状態が `established` と表示されているはずです。トラブルシューティングの詳細については「Cilium 資料」の「トラブルシューティングおよび操作ガイド」を参照してください。

   以下の例では、Cilium エージェントを実行しているハイブリッドノードが 5 つあり、各ノードが所有するポッド CIDR 範囲をアドバタイズしています。

```
cilium bgp peers
```

```
Node                   Local AS    Peer AS              Peer Address        Session
 State   Uptime     Family         Received   Advertised
mi-026d6a261e355fba7   NODES_ASN
                  ONPREM_ROUTER_ASN
                  ONPREM_ROUTER_IP    established     1h18m58s   ipv4/unicast   1
      2
mi-082f73826a163626e   NODES_ASN
                  ONPREM_ROUTER_ASN
```

```
                  ONPREM_ROUTER_IP    established     1h19m12s   ipv4/unicast   1
      2
mi-09183e8a3d755abf6   NODES_ASN
                  ONPREM_ROUTER_ASN
                  ONPREM_ROUTER_IP    established     1h18m47s   ipv4/unicast   1
      2
mi-0d78d815980ed202d   NODES_ASN
                  ONPREM_ROUTER_ASN
                  ONPREM_ROUTER_IP    established     1h19m12s   ipv4/unicast   1
      2
mi-0daa253999fe92daa   NODES_ASN
                  ONPREM_ROUTER_ASN
                  ONPREM_ROUTER_IP    established     1h18m58s   ipv4/unicast   1
      2
```

```
cilium bgp routes
```

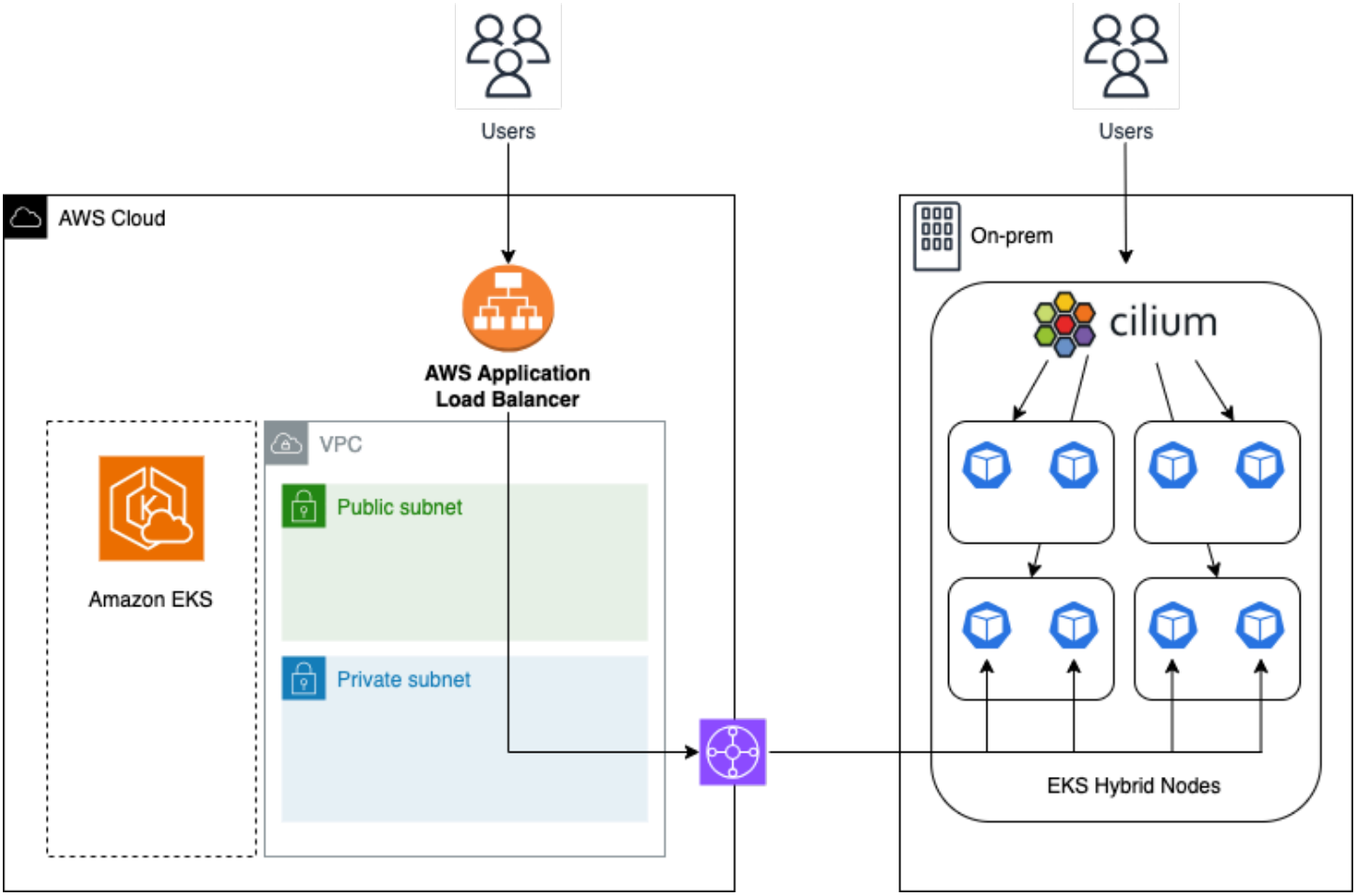
```
Node                   VRouter       Prefix           NextHop   Age         Attrs
mi-026d6a261e355fba7   NODES_ASN     10.86.2.0/26     0.0.0.0   1h16m46s   [{Origin:
 i} {Nexthop: 0.0.0.0}]
mi-082f73826a163626e   NODES_ASN     10.86.2.192/26   0.0.0.0   1h16m46s   [{Origin:
 i} {Nexthop: 0.0.0.0}]
mi-09183e8a3d755abf6   NODES_ASN     10.86.2.64/26    0.0.0.0   1h16m46s   [{Origin:
 i} {Nexthop: 0.0.0.0}]
mi-0d78d815980ed202d   NODES_ASN     10.86.2.128/26   0.0.0.0   1h16m46s   [{Origin:
 i} {Nexthop: 0.0.0.0}]
mi-0daa253999fe92daa   NODES_ASN     10.86.3.0/26     0.0.0.0   1h16m46s   [{Origin:
 i} {Nexthop: 0.0.0.0}]
```

ハイブリッドノード用に Kubernetes Ingress を設定する

このトピックでは、Amazon EKS Hybrid Nodes で実行されているワークロード用に Kubernetes Ingress を設定する方法について説明します。Kubernetes Ingress は、クラスターの外部からクラスター内のサービスに至るまでの HTTP ルートと HTTPS ルートを公開します。Ingress リソースを利用するには、ネットワークトラフィックを処理するネットワークインフラストラクチャとネットワークコンポーネントをセットアップするために Kubernetes Ingress コントローラーが必要です。

AWS は、EKS Hybrid Nodes で実行されているワークロード向けに AWS Application Load Balancer (ALB) と Cilium for Kubernetes Ingress をサポートしています。Ingress に ALB を使用するか Cilium を使用するかは、アプリケーショントラフィックのソースに基づきます。アプリケーショントラ

フィックの発信元が AWS リージョンである場合、AWS では AWS ALB と AWS Load Balancer Controller の使用を推奨しています。アプリケーショントラフィックの発信元がローカルのオンプレミス環境または エッジ環境である場合、AWS では Cilium に組み込みの Ingress 機能の使用を推奨しています。この機能は、お使いの環境にロードバランサーインフラストラクチャを搭載しているかどうかにかかわらず使用できます。



AWS Application Load Balancer

ハイブリッドノード上で実行されているワークロードのターゲットタイプが `ip` の場合、AWS Load Balancer と Application Load Balancer (ALB) を使用できます。ターゲットタイプ `ip` を使用している場合、ALB はトラフィックをポッドに直接転送して、Service レイヤーネットワークパスをバイパスします。ALB がハイブリッドノード上のポッド IP ターゲットに到達するためには、オンプレミスネットワークでオンプレミスポッド CIDR をルーティング可能である必要があります。また、AWS Load Balancer Controller はウェブフックを使用し、EKS コントロールプレーンからの直接通信が必要です。詳細については、「the section called “ウェブフックの設定”」を参照してください。

考慮事項

- AWS Application Load Balancer と AWS Load Balancer Controller の詳細については、「the section called “アプリケーション負荷分散”」と「the section called “Helm を使用してインストールする”」を参照してください。
- AWS Application Load Balancer と AWS Network Load Balancer のどちらかを選択する方法については、「Best Practices for Load Balancing」を参照してください。
- AWS Application Load Balancer で Ingress リソース用に設定できる注釈のリストについては、「AWS Load Balancer Controller Ingress annotations」を参照してください。

前提条件

- 「the section called “CNI を設定する”」の手順に従って Cilium がインストールされていること。
- 「the section called “BGP を設定する”」の手順に従って Cilium BGP コントロールプレーンを有効にしていること。BGP を使用しない場合は、別の方法を使用して、オンプレミスネットワークでオンプレミスポッド CIDR をルーティング可能にする必要があります。オンプレミスポッド CIDR をルーティング可能にしないと、ALB はポッド IP ターゲットの登録も接続もできません。
- コマンドライン環境に Helm がインストールされていること。詳細については、「Setup Helm instructions」を参照してください。
- コマンドライン環境に eksctl がインストールされていること。詳細については、「eksctl install instructions」を参照してください。

手順

1. ユーザーに代わって AWS API を呼び出すことを許可する、AWS Load Balancer Controller 用の IAM ポリシーをダウンロードします。

```
curl -O https://raw.githubusercontent.com/kubernetes-sigs/aws-load-balancer-
controller/refs/heads/main/docs/install/iam_policy.json
```

2. 前のステップでダウンロードしたポリシー を使用して、IAM ポリシーを作成します。

```
aws iam create-policy \
    --policy-name AWSLoadBalancerControllerIAMPolicy \
    --policy-document file://iam_policy.json
```

3. クラスター名 (CLUSTER_NAME)、AWS リージョン (AWS_REGION)、AWS アカウント ID (AWS_ACCOUNT_ID) の各値を自分の設定値に置き換えて、次のコマンドを実行します。

```
eksctl create iamserviceaccount \
    --cluster=CLUSTER_NAME \
    --namespace=kube-system \
    --name=aws-load-balancer-controller \
    --attach-policy-arn=arn:aws:iam::AWS_ACCOUNT_ID:policy/
AWSLoadBalancerControllerIAMPolicy \
    --override-existing-serviceaccounts \
    --region AWS_REGION \
    --approve
```

4. eks-charts Helm チャートリポジトリを追加し、ローカル Helm リポジトリを更新して、チャートを最新の状態にします。

```
helm repo add eks https://aws.github.io/eks-charts
```

```
helm repo update eks
```

5. AWS Load Balancer コントローラをインストールします。クラスター名 (CLUSTER_NAME)、AWS リージョン (AWS_REGION)、VPC ID (VPC_ID)、AWS Load Balancer Controller Helm チャートバージョン (AWS_LBC_HELM_VERSION) の各値を自分の設定値に置き換えて、次のコマンドを実行します。ハイブリッドノードと AWS クラウド内のノードの両方を使用して混合モードクラスターを実行している場合は、「the section called “AWS Load Balancer Controller”」の手順に従ってクラウドノード上で AWS Load Balancer Controller を実行できます。

   - Helm チャートの最新バージョンを確認するには、`helm search repo eks/aws-load-balancer-controller --versions` を実行します。

```
helm install aws-load-balancer-controller eks/aws-load-balancer-controller \
  -n kube-system \
  --version AWS_LBC_HELM_VERSION \
  --set clusterName=CLUSTER_NAME \
  --set region=AWS_REGION \
  --set vpcId=VPC_ID \
  --set serviceAccount.create=false \
  --set serviceAccount.name=aws-load-balancer-controller
```

6. AWS Load Balancer Controller が正常にインストールされたことを確認します。

```
kubectl get -n kube-system deployment aws-load-balancer-controller
```

```
NAME                           READY   UP-TO-DATE   AVAILABLE   AGE
aws-load-balancer-controller   2/2     2            2           84s
```

7. サンプルアプリケーションを作成します。以下の例では、Istio Bookinfo サンプルマイクロサービスアプリケーションを使用しています。

```
kubectl apply -f https://raw.githubusercontent.com/istio/istio/refs/heads/master/
samples/bookinfo/platform/kube/bookinfo.yaml
```

8. 次の内容で、my-ingress-alb.yaml という名前のファイルを作成します。

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: my-ingress
  namespace: default
  annotations:
    alb.ingress.kubernetes.io/load-balancer-name: "my-ingress-alb"
    alb.ingress.kubernetes.io/target-type: "ip"
    alb.ingress.kubernetes.io/scheme: "internet-facing"
    alb.ingress.kubernetes.io/healthcheck-path: "/details/1"
spec:
  ingressClassName: alb
  rules:
  - http:
      paths:
      - backend:
          service:
            name: details
            port:
              number: 9080
        path: /details
        pathType: Prefix
```

9. Ingress 設定をクラスターに適用します。

```
kubectl apply -f my-ingress-alb.yaml
```

10Ingress リソース用に ALB をプロビジョニングするには、数分かかる場合があります。ALB がプロビジョニングされると、ALB デプロイの DNS 名に対応するアドレスが Ingress リソースに割り当てられます。アドレスの形式は <alb-name>-<random-string>.<region>.elb.amazonaws.com です。

```
kubectl get ingress my-ingress
```

```
NAME         CLASS   HOSTS   ADDRESS
    PORTS   AGE
my-ingress   alb     *       my-ingress-alb-<random-
string>.<region>.elb.amazonaws.com   80      23m
```

11ALB のアドレスを使用して Service にアクセスします。

```
curl -s http//my-ingress-alb-<random-string>.<region>.elb.amazonaws.com:80/details/1
 | jq
```

```
{
  "id": 1,
  "author": "William Shakespeare",
  "year": 1595,
  "type": "paperback",
  "pages": 200,
  "publisher": "PublisherA",
  "language": "English",
  "ISBN-10": "1234567890",
  "ISBN-13": "123-1234567890"
  "details": "This is the details page"
}
```

Cilium Ingress および Cilium Gateway の概要

Cilium Ingress は、Cilium のアーキテクチャに組み込みの機能であり、Kubernetes Ingress API または Gateway API を使用して管理できます。既存の Ingress リソースがない場合、AWS では Gateway API から始めることを推奨しています。Kubernetes ネットワークリソースを柔軟かつ表現力豊かに定義および管理できるためです。Kubernetes Gateway API の目的は、Kubernetes クラスターに Ingress、Load Balancing、Service Mesh のネットワークリソースを定義および管理する方法を標準化することにあります。

Cilium の Ingress 機能または Gateway 機能を有効にすると、Cilium オペレーターがクラスター内の Ingress/Gateway オブジェクトを照合し、各ノードの Envoy プロキシがレイヤー 7 (L7) ネットワークトラフィックを処理します。Cilium は、ロードバランサーなどの Ingress/Gateway インフラストラクチャを直接にはプロビジョニングしません。ロードバランサーと共に Cilium Ingress/Gateway を使用する場合は、ロードバランサーのツール (通常は Ingress コントローラーや Gateway コントローラー) を使用して、ロードバランサーのインフラストラクチャをデプロイおよび管理する必要があります。

Ingress/Gateway トラフィックの場合、Cilium がコアネットワークトラフィックと L3/L4 ポリシー適用を処理し、統合 Envoy プロキシが L7 ネットワークトラフィックを処理します。Cilium Ingress/Gateway の場合、Envoy が L7 ルーティングルール、ポリシー、リクエスト操作の適用、高度なトラフィック管理 (トラフィックの分割やミラーリングなど)、TLS の終了と発信を行います。Cilium の Envoy プロキシは、デフォルトでは独立した DaemonSet (`cilium-envoy`) としてデプロイされます。これにより、Envoy と Cilium エージェントを個別に更新、スケール、管理できます。

Cilium Ingress と Cilium Gateway の仕組みの詳細については、Cilium のドキュメントの「Cilium Ingress」ページと「Cilium Gateway」ページを参照してください。

Cilium Ingress と Cilium Gateway の比較

以下の表に、Cilium バージョン 1.17.x の Cilium Ingress 機能と Cilium Gateway 機能をまとめます。

機能	Ingress	ゲートウェイ
サービスタイプ LoadBalancer	はい	はい
サービスタイプ NodePort	はい	いいえ[1]
ホストネットワーク	はい	はい
共有ロードバランサー	はい	はい
専用ロードバランサー	はい	なし[2]
ネットワークポリシー	はい	はい
プロトコル	レイヤー 7 (HTTP(S)、gRPC)	レイヤー 7 (HTTP(S)、gRPC)[3]
TLS パススルー	はい	はい

機能	Ingress	ゲートウェイ
トラフィック管理	パスおよびホストのルーティング	パスおよびホストのルーティング、URL のリダイレクトと書き換え、トラフィックの分割、ヘッダーの変更

[1] Cilium Gateway による NodePort サービスのサポートは、Cilium バージョン 1.18.x (#27273) で予定されています

[2] Cilium Gateway による専用ロードバランサーのサポート (#25567)

[3] Cilium Gateway による TCP/UDP のサポート (#21929)

Cilium Gateway をインストールする

考慮事項

- Cilium では、以下の例に示すように、`nodePort.enabled` を `true` に設定する必要があります。Cilium の kube-proxy 置換機能を使用している場合は、`nodePort.enabled` を `true` に設定する必要はありません。
- Cilium では、以下の例に示すように、`envoy.enabled` を `true` に設定する必要があります。
- Cilium Gateway は、ロードバランサーモード (デフォルト) またはホストネットワークモードでデプロイできます。
- Cilium Gateway をロードバランサーモードで使用する場合は、Gateway リソースに `service.beta.kubernetes.io/aws-load-balancer-type: "external"` 注釈を設定する必要があります。これにより、Cilium が Gateway リソース用にタイプ LoadBalancer の Service を作成する際に、レガシー AWS クラウドプロバイダーがその Service 用に Classic Load Balancer を作成しなくなります。
- Cilium Gateway をホストネットワークモードで使用する場合、タイプ LoadBalancer の Service は無効になります。ホストネットワークモードは、ロードバランサーインフラストラクチャがない環境に有用です。詳細については、「the section called “ホストネットワーク”」を参照してください。

前提条件

1. コマンドライン環境に Helm がインストールされていること。「Setup Helm instructions」を参照してください。
2. 「the section called “CNI を設定する”」の手順に従って Cilium がインストールされていること。

手順

1. Kubernetes Gateway API カスタムリソース定義 (CRD) をインストールします。

```
kubectl apply -f https://raw.githubusercontent.com/kubernetes-sigs/gateway-api/
v1.2.1/config/crd/standard/gateway.networking.k8s.io_gatewayclasses.yaml
kubectl apply -f https://raw.githubusercontent.com/kubernetes-sigs/gateway-api/
v1.2.1/config/crd/standard/gateway.networking.k8s.io_gateways.yaml
kubectl apply -f https://raw.githubusercontent.com/kubernetes-sigs/gateway-api/
v1.2.1/config/crd/standard/gateway.networking.k8s.io_httproutes.yaml
kubectl apply -f https://raw.githubusercontent.com/kubernetes-sigs/gateway-api/
v1.2.1/config/crd/standard/gateway.networking.k8s.io_referencegrants.yaml
kubectl apply -f https://raw.githubusercontent.com/kubernetes-sigs/gateway-api/
v1.2.1/config/crd/standard/gateway.networking.k8s.io_grpcroutes.yaml
```

2. `cilium-gateway-values.yaml` というファイルを次の内容で作成します。以下の例では、デフォルトのロードバランサーモードを使用するように Cilium Gateway を設定しています。また、Envoy プロキシをハイブリッドノードでのみ実行するように設定している場合には、そのプロキシに別の `cilium-envoy` DaemonSet を使用するように設定しています。

```
gatewayAPI:
  enabled: true
  # uncomment to use host network mode
  # hostNetwork:
  #   enabled: true
nodePort:
  enabled: true
envoy:
  enabled: true
  affinity:
    nodeAffinity:
      requiredDuringSchedulingIgnoredDuringExecution:
        nodeSelectorTerms:
        - matchExpressions:
          - key: eks.amazonaws.com/compute-type
```

```
            operator: In
            values:
            - hybrid
```

3. Helm 値ファイルをクラスターに適用します。

```
helm upgrade cilium oci://public.ecr.aws/eks/cilium/cilium \
  --namespace kube-system \
  --reuse-values \
  --set operator.rollOutPods=true \
  --values cilium-gateway-values.yaml
```

4. Cilium オペレーター、エージェント、Envoy ポッドが動作していることを確認します。

```
kubectl -n kube-system get pods --selector=app.kubernetes.io/part-of=cilium
```

```
NAME                               READY   STATUS    RESTARTS   AGE
cilium-envoy-5pgnd                 1/1     Running   0          6m31s
cilium-envoy-6fhg4                 1/1     Running   0          6m30s
cilium-envoy-jskrk                 1/1     Running   0          6m30s
cilium-envoy-k2xtb                 1/1     Running   0          6m31s
cilium-envoy-w5s9j                 1/1     Running   0          6m31s
cilium-grwlc                       1/1     Running   0          4m12s
cilium-operator-68f7766967-5nnbl   1/1     Running   0          4m20s
cilium-operator-68f7766967-7spfz   1/1     Running   0          4m20s
cilium-pnxcv                       1/1     Running   0          6m29s
cilium-r7qkj                       1/1     Running   0          4m12s
cilium-wxhfn                       1/1     Running   0          4m1s
cilium-z7hlb                       1/1     Running   0          6m30s
```

Cilium Gateway を設定する

Cilium Gateway を Gateway オブジェクトで有効にするには、`gatewayClassName` を `cilium` に設定します。Cilium が Gateway リソース用に作成する Service を設定するには、Gateway オブジェクトの各種フィールドを使用します。ロードバランサーインフラストラクチャを設定するために Gateway コントローラーでよく使用される注釈を設定するには、Gateway オブジェクトの `infrastructure` フィールドを使用します。Cilium の LoadBalancer IPAM を使用する場合 (「the section called “サービスタイプ LoadBalancer”」の例を参照) に、タイプ LoadBalancer の Service に使用する IP アドレスを設定するには、Gateway オブジェクトの `addresses` フィールドを使用しま

す。Gateway 設定の詳細については、「Kubernetes Gateway API specification」を参照してください。

```
apiVersion: gateway.networking.k8s.io/v1
kind: Gateway
metadata:
  name: my-gateway
spec:
  gatewayClassName: cilium
  infrastructure:
    annotations:
      service.beta.kubernetes.io/...
      service.kubernetes.io/...
  addresses:
  - type: IPAddress
    value: <LoadBalancer IP address>
  listeners:
  ...
```

Cilium と Kubernetes Gateway の仕様では、GatewayClass、Gateway、HTTPRoute、GRPCRoute、ReferenceGrant の各リソースがサポートされています。

- 使用可能なフィールドのリストについては、「HTTPRoute」と「GRPCRoute」の仕様を参照してください。
- これらのリソースの使用と設定方法については、以下の「the section called “Cilium Gateway をデプロイする”」セクションの例とCilium のドキュメントの例を参照してください。

Cilium Gateway をデプロイする

1. サンプルアプリケーションを作成します。以下の例では、Istio Bookinfo サンプルマイクロサービスアプリケーションを使用しています。

   ```
   kubectl apply -f https://raw.githubusercontent.com/istio/istio/refs/heads/master/
   samples/bookinfo/platform/kube/bookinfo.yaml
   ```

2. アプリケーションが正常に動作していることを確認します。

   ```
   kubectl get pods
   ```

```
NAME                              READY   STATUS    RESTARTS   AGE
details-v1-766844796b-9965p       1/1     Running   0          81s
productpage-v1-54bb874995-jmc8j   1/1     Running   0          80s
ratings-v1-5dc79b6bcd-smzxz       1/1     Running   0          80s
reviews-v1-598b896c9d-vj7gb       1/1     Running   0          80s
reviews-v2-556d6457d-xbt8v        1/1     Running   0          80s
reviews-v3-564544b4d6-cpmvq       1/1     Running   0          80s
```

3. 次の内容で、my-gateway.yaml という名前のファイルを作成します。以下の例では、service.beta.kubernetes.io/aws-load-balancer-type: "external" 注釈を使用して、Cilium が Gateway リソース用にタイプ LoadBalancer の Service を作成する際に、レガシー AWS クラウドプロバイダーがその Service 用に Classic Load Balancer を作成しないようにしています。

```
---
apiVersion: gateway.networking.k8s.io/v1
kind: Gateway
metadata:
  name: my-gateway
spec:
  gatewayClassName: cilium
  infrastructure:
    annotations:
      service.beta.kubernetes.io/aws-load-balancer-type: "external"
  listeners:
  - protocol: HTTP
    port: 80
    name: web-gw
    allowedRoutes:
      namespaces:
        from: Same
---
apiVersion: gateway.networking.k8s.io/v1
kind: HTTPRoute
metadata:
  name: http-app-1
spec:
  parentRefs:
  - name: my-gateway
    namespace: default
  rules:
  - matches:
```

```
    - path:
        type: PathPrefix
        value: /details
    backendRefs:
    - name: details
      port: 9080
```

4. クラスターに Gateway リソースを適用します。

```
kubectl apply -f my-gateway.yaml
```

5. Gateway リソースとその対応する Service が作成されたことを確認します。この段階では、Gateway リソースの ADDRESS フィールドに IP アドレスもホスト名も入力されておらず、Gateway リソースのタイプ LoadBalancer の Service にも同じく IP アドレスもホスト名も割り当てられていないはずです。

```
kubectl get gateway my-gateway
```

```
NAME         CLASS    ADDRESS   PROGRAMMED   AGE
my-gateway   cilium             True         10s
```

```
kubectl get svc cilium-gateway-my-gateway
```

```
NAME                         TYPE           CLUSTER-IP       EXTERNAL-IP   PORT(S)
    AGE
cilium-gateway-my-gateway   LoadBalancer   172.16.227.247   <pending>     80:30912/
TCP   24s
```

6. the section called “サービスタイプ LoadBalancer” に進み、Cilium Load Balancer IPAM によって割り当てられた IP アドレスを使用するように Gateway リソースを設定します。さらに、the section called “サービスタイプ NodePort” または the section called “ホストネットワーク” に進み、NodePort またはホストネットワークアドレスを使用するように Gateway リソースを設定します。

Cilium Ingress をインストールする

考慮事項

- Cilium では、以下の例に示すように、`nodePort.enabled` を `true` に設定する必要があります。Cilium の kube-proxy 置換機能を使用している場合は、`nodePort.enabled` を `true` に設定する必要はありません。
- Cilium では、以下の例に示すように、`envoy.enabled` を `true` に設定する必要があります。
- `ingressController.loadbalancerMode` を `dedicated` に設定した場合、Cilium は Ingress リソースごとに専用の Service を作成します。`ingressController.loadbalancerMode` を `shared` に設定した場合、Cilium はクラスター内のすべての Ingress リソースに対してタイプ LoadBalancer の共有 Service を作成します。`shared` ロードバランサーモードを使用する場合、`labels`、`annotations`、`type`、`loadBalancerIP` などの共有 Service の設定を Helm 値の `ingressController.service` セクションで行います。詳細については、「Cilium Helm values reference」を参照してください。
- `ingressController.default` を `true` に設定した場合、Cilium はクラスターのデフォルトの Ingress コントローラーとして設定され、`ingressClassName` が Ingress リソースに指定されていない場合でも Ingress エントリを作成します。
- Cilium Ingress は、ロードバランサー (デフォルト)、ノードポート、ホストネットワークモードのいずれかでデプロイできます。Cilium をホストネットワークモードでインストールすると、タイプ LoadBalancer の Service およびタイプ NodePort モードの Service が無効になります。詳細については「the section called “ホストネットワーク”」を参照してください。
- Helm 値では常に `ingressController.service.annotations` を `service.beta.kubernetes.io/aws-load-balancer-type: "external"` に設定します。これにより、レガシー AWS クラウドプロバイダーは、Cilium Helm チャートによって作成されたデフォルトの `cilium-ingress` Service 用に Classic Load Balancer を作成しなくなります。

前提条件

1. コマンドライン環境に Helm がインストールされていること。「Setup Helm instructions」を参照してください。
2. 「the section called “CNI を設定する”」の手順に従って Cilium がインストールされていること。

手順

1. cilium-ingress-values.yaml というファイルを次の内容で作成します。以下に、Cilium Ingress の設定例を示します。デフォルトのロードバランサー dedicated モードを使用すること、および Envoy プロキシに個別の cilium-envoy DaemonSet を使用してプロキシがハイブリッドノードでのみ動作することを設定しています。

```
ingressController:
  enabled: true
  loadbalancerMode: dedicated
  service:
    annotations:
      service.beta.kubernetes.io/aws-load-balancer-type: "external"
nodePort:
  enabled: true
envoy:
  enabled: true
  affinity:
    nodeAffinity:
      requiredDuringSchedulingIgnoredDuringExecution:
        nodeSelectorTerms:
        - matchExpressions:
          - key: eks.amazonaws.com/compute-type
            operator: In
            values:
            - hybrid
```

2. Helm 値ファイルをクラスターに適用します。

```
helm upgrade cilium oci://public.ecr.aws/eks/cilium/cilium \
  --namespace kube-system \
  --reuse-values \
  --set operator.rollOutPods=true \
  --values cilium-ingress-values.yaml
```

3. Cilium オペレーター、エージェント、Envoy ポッドが動作していることを確認します。

```
kubectl -n kube-system get pods --selector=app.kubernetes.io/part-of=cilium
```

```
NAME                               READY   STATUS    RESTARTS   AGE
cilium-envoy-5pgnd                 1/1     Running   0          6m31s
cilium-envoy-6fhg4                 1/1     Running   0          6m30s
```

```
cilium-envoy-jskrk                  1/1     Running   0          6m30s
cilium-envoy-k2xtb                  1/1     Running   0          6m31s
cilium-envoy-w5s9j                  1/1     Running   0          6m31s
cilium-grwlc                        1/1     Running   0          4m12s
cilium-operator-68f7766967-5nnbl    1/1     Running   0          4m20s
cilium-operator-68f7766967-7spfz    1/1     Running   0          4m20s
cilium-pnxcv                        1/1     Running   0          6m29s
cilium-r7qkj                        1/1     Running   0          4m12s
cilium-wxhfn                        1/1     Running   0          4m1s
cilium-z7hlb                        1/1     Running   0          6m30s
```

Cilium Ingress を設定する

Cilium Ingress を Ingress オブジェクトで有効にするには、`ingressClassName` を `cilium` に設定します。Cilium が Ingress リソース用に作成した Service を設定するには、`dedicated` ロードバランサーモードを使用している場合には Ingress オブジェクトを使用し、`shared` ロードバランサーモードを使用している場合には Cilium/Helm 設定を使用します。こうした注釈は、Ingress コントローラーがロードバランサーインフラストラクチャを設定するときや、サービスタイプ、ロードバランサーモード、ポート、TLS パススルーといった Service のその他の属性を設定するときによく使用されます。以下に、主な注釈を示します。サポートされている注釈の完全なリストについては、Cilium のドキュメントの「Cilium Ingress annotations」を参照してください。

注釈	説明
`ingress.cilium.io/loadbalancer-mode`	`dedicated` ：各 Ingress リソースのタイプ LoadBalancer の専用 Service (デフォルト)。 `shared`: すべての Ingress リソースのタイプ LoadBalancer の単一の Service。
`ingress.cilium.io/service-type`	`LoadBalancer` ：Service は、タイプ LoadBalancer になります (デフォルト)。 `NodePort`: Service は、タイプ NodePort になります。
`service.beta.kubernetes.io/aws-load-balancer-type`	`"external"` ：レガシー AWS クラウドプロバイダーは、タイプ LoadBalancer の Service に対して Classic Load Balancer をプロビジョニングしなくなります。

注釈	説明
`lbipam.cilium.io/ips`	Cilium LoadBalancer IPAM から割り当てる IP アドレスのリスト

Cilium と Kubernetes Ingress の仕様は、Ingress パスのマッチングルールとして完全一致、プレフィックス一致、実装固有一致をサポートしています。Cilium は、実装固有マッチングルールとして正規表現をサポートしています。詳細については、Cilium のドキュメントの「Ingress path types and precedence」と「Path types examples」、さらにこのページの「the section called “Cilium Ingress をデプロイする”」セクションの例を参照してください。

以下に、Cilium Ingress オブジェクトの例を示します。

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: my-ingress
  annotations:
    service.beta.kubernetes.io/...
    service.kubernetes.io/...
spec:
  ingressClassName: cilium
  rules:
  ...
```

Cilium Ingress をデプロイする

1. サンプルアプリケーションを作成します。以下の例では、Istio Bookinfo サンプルマイクロサービスアプリケーションを使用しています。

```
kubectl apply -f https://raw.githubusercontent.com/istio/istio/refs/heads/master/
samples/bookinfo/platform/kube/bookinfo.yaml
```

2. アプリケーションが正常に動作していることを確認します。

```
kubectl get pods
```

```
NAME                              READY   STATUS    RESTARTS   AGE
details-v1-766844796b-9965p       1/1     Running   0          81s
```

```
productpage-v1-54bb874995-jmc8j   1/1     Running   0          80s
ratings-v1-5dc79b6bcd-smzxz       1/1     Running   0          80s
reviews-v1-598b896c9d-vj7gb       1/1     Running   0          80s
reviews-v2-556d6457d-xbt8v        1/1     Running   0          80s
reviews-v3-564544b4d6-cpmvq       1/1     Running   0          80s
```

3. 次の内容で、my-ingress.yaml という名前のファイルを作成します。以下の例では、service.beta.kubernetes.io/aws-load-balancer-type: "external" 注釈を使用して、Cilium が Ingress リソース用にタイプ LoadBalancer の Service を作成する際に、レガシー AWS クラウドプロバイダーがその Service 用に Classic Load Balancer を作成しないようにしています。

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: my-ingress
  namespace: default
  annotations:
    service.beta.kubernetes.io/aws-load-balancer-type: "external"
spec:
  ingressClassName: cilium
  rules:
  - http:
      paths:
      - backend:
          service:
            name: details
            port:
              number: 9080
        path: /details
        pathType: Prefix
```

4. Ingress リソースをクラスターに適用します。

```
kubectl apply -f my-ingress.yaml
```

5. Ingress リソースおよびその対応する Service が作成されたことを確認します。この段階では、Ingress リソースの ADDRESS フィールドに IP アドレスもホスト名も入力されておらず、Ingress リソースのタイプ LoadBalancer の共有または専用 Service にも同じく IP アドレスもホスト名も割り当てられていないはずです。

```
kubectl get ingress my-ingress
```

```
NAME         CLASS    HOSTS   ADDRESS   PORTS   AGE
my-ingress   cilium   *                 80      8s
```

ロードバランサーモード shared の場合

```
kubectl -n kube-system get svc cilium-ingress
```

```
NAME             TYPE           CLUSTER-IP      EXTERNAL-IP   PORT(S)
      AGE
cilium-ingress   LoadBalancer   172.16.217.48   <pending>     80:32359/TCP,443:31090/
TCP   10m
```

ロードバランサーモード dedicated の場合

```
kubectl -n default get svc cilium-ingress-my-ingress
```

```
NAME                         TYPE           CLUSTER-IP      EXTERNAL-IP   PORT(S)
                  AGE
cilium-ingress-my-ingress    LoadBalancer   172.16.193.15   <pending>     80:32088/
TCP,443:30332/TCP   25s
```

6. the section called “サービスタイプ LoadBalancer” に進み、Cilium Load Balancer IPAM によって割り当てられた IP アドレスを使用するように Ingress リソースを設定します。さらに、the section called “サービスタイプ NodePort” または the section called “ホストネットワーク” に進み、NodePort またはホストネットワークアドレスを使用するように Ingress リソースを設定します。

サービスタイプ LoadBalancer

既存のロードバランサーインフラストラクチャ

デフォルトでは、Cilium Ingress の場合も Cilium Gateway の場合も、Cilium は Ingress/Gateway リソース用にタイプ LoadBalancer の Kubernetes Service を作成します。Cilium が作成した Service の属性を設定するには、Ingress リソースと Gateway リソースを使用します。Ingress リソースまたは Gateway リソースを作成すると、Ingress または Gateway 用に外部公開された IP アドレスまたはホ

スト名がロードバランサーインフラストラクチャから割り当てられます。このインフラストラクチャは通常、Ingress コントローラーまたは Gateway コントローラーによってプロビジョニングされます。

Ingress コントローラーと Gateway コントローラーの多くが、ロードバランサーインフラストラクチャを検出および設定する際に注釈を使用します。Ingress コントローラーと Gateway コントローラーのこうした注釈は、上記の例に示すように、Ingress リソースまたは Gateway リソースに設定します。サポートされている注釈については、Ingress コントローラーまたは Gateway コントローラーのドキュメントを参照してください。また、よく使用されるコントローラーのリストについては、Kubernetes Ingress のドキュメントと Kubernetes Gateway のドキュメントを参照してください。

> **Important**
>
> EKS Hybrid Nodes では、Cilium Ingress と Gateway を AWS Load Balancer Controller および AWS Network Load Balancer (NLB) と共に使用することはできません。これらを一緒に使用しようとすると、ターゲットが未登録になります。NLB の `target-type` が `ip` に設定されている場合 (EKS Hybrid Nodes で実行されるワークロードで NLB を使用するための要件)、NLB はタイプ LoadBalancer の Service を支援する Pod IP に直接接続しようとするためです。

ロードバランサーインフラストラクチャなし

環境にロードバランサーインフラストラクチャおよびその対応する Ingress/Gateway コントローラーがない場合、Ingress/Gateway リソースおよびその対応するタイプ LoadBalancer の Service を設定するには、Cilium の Load Balancer IP Address Management (LB IPAM) によって割り当てられた IP アドレスを使用します。Cilium LB IPAM は、オンプレミス環境の既知の IP アドレス範囲と共に設定できます。また、Cilium に組み込みのボーダーゲートウェイプロトコル (BGP) サポートまたは L2 のお知らせを使用して、LoadBalancer IP アドレスをオンプレミスネットワークにアドバタイズできます。

以下の例では、Ingress/Gateway リソースに使用する IP アドレスと共に Cilium の LB IPAM を設定する方法と、オンプレミスネットワークで LoadBalancer IP アドレスをアドバタイズするように Cilium BGP コントロールプレーンを設定する方法を示します。Cilium の LB IPAM 機能は、デフォルトで有効になっていますが、`CiliumLoadBalancerIPPool` リソースが作成されるまではアクティブ化されません。

前提条件

- 「the section called “Cilium Ingress をインストールする”」または「the section called “Cilium Gateway をインストールする”」の手順に従って Cilium Ingress または Gateway がインストールされていること。
- 「the section called “Cilium Ingress をデプロイする”」または「the section called “Cilium Gateway をデプロイする”」の手順に従って、Cilium Ingress リソースまたは Gateway リソースがサンプルアプリケーションと共にデプロイされていること。
- 「the section called “BGP を設定する”」の手順に従って Cilium BGP コントロールプレーンを有効にしていること。BGP を使用しない場合にはこの前提条件をスキップできますが、Cilium LB IPAM によって割り当てられた LoadBalancer IP アドレスがオンプレミスネットワークでルーティング可能になるまで、Ingress リソースと Gateway リソースにアクセスできなくなります。

手順

1. 必要に応じて、Ingress リソースまたは Gateway リソースにパッチを適用してタイプ LoadBalancer の Service に特定の IP アドレスを使用するようにリクエストしてください。特定の IP アドレスをリクエストしないと、Cilium は後続のステップで `CiliumLoadBalancerIPPool` リソースに設定する IP アドレス範囲から IP アドレスを割り当てます。以下のコマンドでは、`LB_IP_ADDRESS` をタイプ LoadBalancer の Service 用にリクエストする IP アドレスに置き換えます。

   ゲートウェイ

   ```
   kubectl patch gateway -n default my-gateway --type=merge -p '{
     "spec": {
       "addresses": [{"type": "IPAddress", "value": "LB_IP_ADDRESS"}]
     }
   }'
   ```

   Ingress

   ```
   kubectl patch ingress my-ingress --type=merge -p '{
     "metadata": {"annotations": {"lbipam.cilium.io/ips": "LB_IP_ADDRESS"}}
   }'
   ```

2. `CiliumLoadBalancerIPPool` リソースを使用して Ingress/Gateway リソースの Load Balancer IP アドレス範囲を設定するファイルを `cilium-lbip-pool-ingress.yaml` という名前で作成します。
   - Cilium Ingress を使用している場合、Cilium は Ingress リソース用に作成する Service に `cilium.io/ingress: "true"` ラベルを自動的に適用します。`CiliumLoadBalancerIPPool` リソース定義の `serviceSelector` フィールドでこのラベルを使用すると、LB IPAM の対象となる Service を選択できます。
   - Cilium Gateway を使用している場合は、`CiliumLoadBalancerIPPool` リソース定義の `serviceSelector` フィールドで `gateway.networking.k8s.io/gateway-name` ラベルを使用して、LB IPAM の対象となる Gateway リソースを選択できます。
   - LB_IP_CIDR を Load Balancer の IP アドレスに使用する IP アドレス範囲に置き換えます。単一の IP アドレスを選択するには、/32 CIDR を使用します。詳細については、Cilium のドキュメントの「LoadBalancer IP Address Management」を参照してください。

```
apiVersion: cilium.io/v2alpha1
kind: CiliumLoadBalancerIPPool
metadata:
  name: bookinfo-pool
spec:
  blocks:
  - cidr: "LB_IP_CIDR"
  serviceSelector:
    # if using Cilium Gateway
    matchExpressions:
      - { key: gateway.networking.k8s.io/gateway-name, operator: In, values: [ my-
gateway ] }
    # if using Cilium Ingress
    matchLabels:
      cilium.io/ingress: "true"
```

3. `CiliumLoadBalancerIPPool` リソースをクラスターに適用します。

```
kubectl apply -f cilium-lbip-pool-ingress.yaml
```

4. Ingress/Gateway リソースの IP アドレスが Cilium LB IPAM から割り当てられたことを確認します。

   ゲートウェイ

```
kubectl get gateway my-gateway
```

```
NAME         CLASS    ADDRESS        PROGRAMMED   AGE
my-gateway   cilium   LB_IP_ADDRESS    True         6m41s
```

Ingress

```
kubectl get ingress my-ingress
```

```
NAME         CLASS    HOSTS   ADDRESS        PORTS   AGE
my-ingress   cilium   *       LB_IP_ADDRESS   80      10m
```

5. `CiliumBGPAdvertisement` リソースを使用して Ingress/Gateway リソースの LoadBalancer IP アドレスをアドバタイズするファイルを `cilium-bgp-advertisement-ingress.yaml` という名前で作成します。Cilium BGP を使用していない場合は、このステップをスキップできます。Ingress/Gateway リソースに使用する LoadBalancer IP アドレスは、次のステップでサービスをクエリできるように、オンプレミスネットワークでルーティング可能である必要があります。

```
apiVersion: cilium.io/v2alpha1
kind: CiliumBGPAdvertisement
metadata:
  name: bgp-advertisement-lb-ip
  labels:
    advertise: bgp
spec:
  advertisements:
    - advertisementType: "Service"
      service:
        addresses:
          - LoadBalancerIP
      selector:
        # if using Cilium Gateway
        matchExpressions:
          - { key: gateway.networking.k8s.io/gateway-name, operator: In, values:
 [ my-gateway ] }
        # if using Cilium Ingress
        matchLabels:
          cilium.io/ingress: "true"
```

6. `CiliumBGPAdvertisement` リソースをクラスターに適用します。

```
kubectl apply -f cilium-bgp-advertisement-ingress.yaml
```

7. Cilium LB IPAM から割り当てられた IP アドレスを使用して、サービスにアクセスします。

```
curl -s http://LB_IP_ADDRESS:80/details/1 | jq
```

```
{
  "id": 1,
  "author": "William Shakespeare",
  "year": 1595,
  "type": "paperback",
  "pages": 200,
  "publisher": "PublisherA",
  "language": "English",
  "ISBN-10": "1234567890",
  "ISBN-13": "123-1234567890"
}
```

サービスタイプ NodePort

環境にロードバランサーインフラストラクチャおよびその対応する Ingress コントローラーがない場合、ロードバランサーインフラストラクチャを自己管理している場合、または DNS ベースのロードバランシングを使用している場合は、Ingress リソース用にタイプ NodePort の Service を作成するように Cilium Ingress を設定できます。Cilium Ingress と共に NodePort を使用する場合、タイプ NodePort の Service は各ノード上でポート範囲 30000 ~ 32767 のポートに公開されます。このモードでは、トラフィックは NodePort 上のクラスター内にあるノードに到達すると、サービスを支援するポッドに転送されます。このポッドは、同じノードにあることもあれば、別のノードにあることもあります。

> **Note**
>
> Cilium Gateway による NodePort サービスのサポートは、Cilium バージョン 1.18.x (#27273) で予定されています

前提条件

- 「the section called “Cilium Ingress をインストールする”」の手順に従って Cilium Ingress がインストールされていること。
- 「the section called “Cilium Ingress をデプロイする”」の手順に従って、Cilium Ingress リソースがサンプルアプリケーションと共にデプロイされていること。

手順

1. 既存の Ingress リソース `my-ingress` にパッチを適用して、そのリソースをタイプ LoadBalancer の Service から NodePort に変更します。

```
kubectl patch ingress my-ingress --type=merge -p '{
    "metadata": {"annotations": {"ingress.cilium.io/service-type": "NodePort"}}
}'
```

Ingress リソースをまだ作成していない場合は、次の Ingress 定義をクラスターに適用することで作成できます。以下の Ingress 定義では、「the section called “Cilium Ingress をデプロイする”」で説明されている Istio Bookinfo サンプルアプリケーションを使用していることに注意してください。

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: my-ingress
  namespace: default
  annotations:
    service.beta.kubernetes.io/aws-load-balancer-type: "external"
    "ingress.cilium.io/service-type": "NodePort"
spec:
  ingressClassName: cilium
  rules:
  - http:
      paths:
      - backend:
          service:
            name: details
            port:
              number: 9080
        path: /details
```

```
        pathType: Prefix
```

2. Ingress リソース用の Service が、タイプ NodePort の Service を使用するように更新されたことを確認します。出力にある HTTP プロトコルのポートを書き留めます。以下の例では 32353 がこの HTTP ポートであり、後続のステップではこのポートを使用して Service をクエリします。タイプ NodePort の Service と共に Cilium Ingress を使用する利点は、パスとホストベースのルーティングを適用できるだけでなく、Ingress トラフィック用のネットワークポリシーも適用できることです。これは、Ingress のないタイプ NodePort の標準の Service ではできないことです。

```
kubectl -n default get svc cilium-ingress-my-ingress
```

```
NAME                         TYPE       CLUSTER-IP      EXTERNAL-IP   PORT(S)
             AGE
cilium-ingress-my-ingress    NodePort   172.16.47.153   <none>        80:32353/
TCP,443:30253/TCP   27m
```

3. クラスター内のノードの IP アドレスを取得します。

```
kubectl get nodes -o wide
```

```
NAME                    STATUS   ROLES    AGE   VERSION               INTERNAL-IP
 EXTERNAL-IP   OS-IMAGE             KERNEL-VERSION       CONTAINER-RUNTIME
mi-026d6a261e355fba7   Ready    <none>   23h   v1.32.3-eks-473151a   10.80.146.150
 <none>        Ubuntu 22.04.5 LTS   5.15.0-142-generic   containerd://1.7.27
mi-082f73826a163626e   Ready    <none>   23h   v1.32.3-eks-473151a   10.80.146.32
 <none>        Ubuntu 22.04.4 LTS   5.15.0-142-generic   containerd://1.7.27
mi-09183e8a3d755abf6   Ready    <none>   23h   v1.32.3-eks-473151a   10.80.146.33
 <none>        Ubuntu 22.04.4 LTS   5.15.0-142-generic   containerd://1.7.27
mi-0d78d815980ed202d   Ready    <none>   23h   v1.32.3-eks-473151a   10.80.146.97
 <none>        Ubuntu 22.04.4 LTS   5.15.0-142-generic   containerd://1.7.27
mi-0daa253999fe92daa   Ready    <none>   23h   v1.32.3-eks-473151a   10.80.146.100
 <none>        Ubuntu 22.04.4 LTS   5.15.0-142-generic   containerd://1.7.27
```

4. ノードの IP アドレスと上記でキャプチャした NodePort を使用して、タイプ NodePort の Service にアクセスします。以下の例では、使用されるノード IP アドレスは 10.80.146.32 で、NodePort は 32353 です。これらをお使いの環境の値で置き換えます。

```
curl -s http://10.80.146.32:32353/details/1 | jq
```

```
{
```

```
  "id": 1,
  "author": "William Shakespeare",
  "year": 1595,
  "type": "paperback",
  "pages": 200,
  "publisher": "PublisherA",
  "language": "English",
  "ISBN-10": "1234567890",
  "ISBN-13": "123-1234567890"
}
```

ホストネットワーク

タイプ NodePort の Service と同様に、ロードバランサーインフラストラクチャと Ingress または Gateway コントローラーがない場合や、外部ロードバランサーでロードバランシングを自己管理している場合は、ホストネットワークに Ingress リソースと Gateway リソースを直接公開するように Cilium Ingress および Cilium Gateway を設定できます。ホストネットワークモードが Ingress リソースまたは Gateway リソースに対して有効になっている場合、タイプ LoadBalancer の Service とタイプ NodePort モードの Service は自動的に無効になり、ホストネットワークモードは Ingress リソースまたは Gateway リソースごとにこれらの代替モードと相互に排他的になります。タイプ NodePort モードの Service と比較して、ホストネットワークモードは使用できるポート範囲の柔軟性が高く (NodePort の 30000～32767 という範囲に制限されません)、ホストネットワーク上で Envoy プロキシが動作するノードのサブセットを設定できます。

前提条件

- 「the section called “Cilium Ingress をインストールする”」または「the section called “Cilium Gateway をインストールする”」の手順に従って Cilium Ingress または Gateway がインストールされていること。

手順

ゲートウェイ

1. `cilium-gateway-host-network.yaml` という名前のファイルを作成し、次の内容を記述します。

```
gatewayAPI:
  enabled: true
  hostNetwork:
```

```
    enabled: true
    # uncomment to restrict nodes where Envoy proxies run on the host network
    # nodes:
    #   matchLabels:
    #     role: gateway
```

2. クラスターにホストネットワーク Cilium Gateway 設定を適用します。

```
helm upgrade cilium oci://public.ecr.aws/eks/cilium/cilium \
  --namespace kube-system \
  --reuse-values \
  --set operator.rollOutPods=true \
  -f cilium-gateway-host-network.yaml
```

Gateway リソースをまだ作成していない場合は、次の Gateway 定義をクラスターに適用することで作成できます。以下の Gateway 定義では、「the section called “Cilium Gateway をデプロイする”」で説明されている Istio Bookinfo サンプルアプリケーションを使用しています。以下の例では、Gateway リソースは HTTP リスナーに 8111 ポートを使用するように設定されています。ホストネットワークで動作する Envoy プロキシの共有リスナーポートです。Gateway リソースに特権ポート (1023 未満) を使用する場合は、その手順を Cilium のドキュメントで確認してください。

```
---
apiVersion: gateway.networking.k8s.io/v1
kind: Gateway
metadata:
  name: my-gateway
spec:
  gatewayClassName: cilium
  listeners:
  - protocol: HTTP
    port: 8111
    name: web-gw
    allowedRoutes:
      namespaces:
        from: Same
---
apiVersion: gateway.networking.k8s.io/v1
kind: HTTPRoute
metadata:
  name: http-app-1
spec:
```

```
  parentRefs:
  - name: my-gateway
    namespace: default
  rules:
  - matches:
    - path:
        type: PathPrefix
        value: /details
    backendRefs:
    - name: details
      port: 9080
```

適用した Cilium Envoy 設定を確認するには、次のコマンドを使用します。

```
kubectl get cec cilium-gateway-my-gateway -o yaml
```

cilium-gateway-my-gateway Service の Envoy リスナーポートを取得するには、次のコマンドを使用します。この例では、8111 が共有リスナーポートです。

```
kubectl get cec cilium-gateway-my-gateway -o jsonpath={.spec.services[0].ports[0]}
```

3. クラスター内のノードの IP アドレスを取得します。

```
kubectl get nodes -o wide
```

```
NAME                   STATUS   ROLES    AGE   VERSION               INTERNAL-IP
 EXTERNAL-IP   OS-IMAGE             KERNEL-VERSION       CONTAINER-RUNTIME
mi-026d6a261e355fba7   Ready    <none>   23h   v1.32.3-eks-473151a   10.80.146.150
 <none>        Ubuntu 22.04.5 LTS   5.15.0-142-generic   containerd://1.7.27
mi-082f73826a163626e   Ready    <none>   23h   v1.32.3-eks-473151a   10.80.146.32
 <none>        Ubuntu 22.04.4 LTS   5.15.0-142-generic   containerd://1.7.27
mi-09183e8a3d755abf6   Ready    <none>   23h   v1.32.3-eks-473151a   10.80.146.33
 <none>        Ubuntu 22.04.4 LTS   5.15.0-142-generic   containerd://1.7.27
mi-0d78d815980ed202d   Ready    <none>   23h   v1.32.3-eks-473151a   10.80.146.97
 <none>        Ubuntu 22.04.4 LTS   5.15.0-142-generic   containerd://1.7.27
mi-0daa253999fe92daa   Ready    <none>   23h   v1.32.3-eks-473151a   10.80.146.100
 <none>        Ubuntu 22.04.4 LTS   5.15.0-142-generic   containerd://1.7.27
```

4. ノードの IP アドレスと cilium-gateway-my-gateway リソースのリスナーポートを使用して、Service にアクセスします。以下の例では、使用されるノード IP アドレスは 10.80.146.32 であり、リスナーポートは 8111 です。これらをお使いの環境の値で置き換えます。

```
curl -s http://10.80.146.32:8111/details/1 | jq
```

```
{
  "id": 1,
  "author": "William Shakespeare",
  "year": 1595,
  "type": "paperback",
  "pages": 200,
  "publisher": "PublisherA",
  "language": "English",
  "ISBN-10": "1234567890",
  "ISBN-13": "123-1234567890"
}
```

Ingress

アップストリームの Cilium の問題 (#34028) により、ホストネットワークモードの Cilium Ingress では `loadbalancerMode: shared` を使用する必要があります。これにより、クラスター内のすべての Ingress リソースに対してタイプ ClusterIP の Service が 1 つだけ作成されます。Ingress リソースに特権ポート (1023 未満) を使用する場合は、その手順を Cilium のドキュメントで確認してください。

1. `cilium-ingress-host-network.yaml` という名前のファイルを作成し、次の内容を記述します。

```
ingressController:
  enabled: true
  loadbalancerMode: shared
  # This is a workaround for the upstream Cilium issue
  service:
    externalTrafficPolicy: null
    type: ClusterIP
  hostNetwork:
    enabled: true
    # ensure the port does not conflict with other services on the node
    sharedListenerPort: 8111
    # uncomment to restrict nodes where Envoy proxies run on the host network
    # nodes:
    #   matchLabels:
```

```
    #     role: ingress
```

2. クラスターにホストネットワーク Cilium Ingress 設定を適用します。

```
helm upgrade cilium oci://public.ecr.aws/eks/cilium/cilium \
  --namespace kube-system \
  --reuse-values \
  --set operator.rollOutPods=true \
  -f cilium-ingress-host-network.yaml
```

Ingress リソースをまだ作成していない場合は、次の Ingress 定義をクラスターに適用することで作成できます。以下の Ingress 定義では、「the section called “Cilium Ingress をデプロイする”」で説明されている Istio Bookinfo サンプルアプリケーションを使用しています。

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: my-ingress
  namespace: default
spec:
  ingressClassName: cilium
  rules:
  - http:
      paths:
      - backend:
          service:
            name: details
            port:
              number: 9080
        path: /details
        pathType: Prefix
```

適用した Cilium Envoy 設定を確認するには、次のコマンドを使用します。

```
kubectl get cec -n kube-system cilium-ingress -o yaml
```

`cilium-ingress` Service の Envoy リスナーポートを取得するには、次のコマンドを使用します。この例では、8111 が共有リスナーポートです。

```
kubectl get cec -n kube-system cilium-ingress -o
 jsonpath={.spec.services[0].ports[0]}
```

3. クラスター内のノードの IP アドレスを取得します。

```
kubectl get nodes -o wide
```

```
NAME                   STATUS   ROLES    AGE   VERSION               INTERNAL-IP
 EXTERNAL-IP   OS-IMAGE             KERNEL-VERSION       CONTAINER-RUNTIME
mi-026d6a261e355fba7   Ready    <none>   23h   v1.32.3-eks-473151a   10.80.146.150
 <none>        Ubuntu 22.04.5 LTS   5.15.0-142-generic   containerd://1.7.27
mi-082f73826a163626e   Ready    <none>   23h   v1.32.3-eks-473151a   10.80.146.32
 <none>        Ubuntu 22.04.4 LTS   5.15.0-142-generic   containerd://1.7.27
mi-09183e8a3d755abf6   Ready    <none>   23h   v1.32.3-eks-473151a   10.80.146.33
 <none>        Ubuntu 22.04.4 LTS   5.15.0-142-generic   containerd://1.7.27
mi-0d78d815980ed202d   Ready    <none>   23h   v1.32.3-eks-473151a   10.80.146.97
 <none>        Ubuntu 22.04.4 LTS   5.15.0-142-generic   containerd://1.7.27
mi-0daa253999fe92daa   Ready    <none>   23h   v1.32.3-eks-473151a   10.80.146.100
 <none>        Ubuntu 22.04.4 LTS   5.15.0-142-generic   containerd://1.7.27
```

4. ノードの IP アドレスと `cilium-ingress` リソースの `sharedListenerPort` を使用して、Service にアクセスします。以下の例では、使用されるノード IP アドレスは `10.80.146.32` であり、リスナーポートは 8111 です。これらをお使いの環境の値で置き換えます。

```
curl -s http://10.80.146.32:8111/details/1 | jq
```

```
{
  "id": 1,
  "author": "William Shakespeare",
  "year": 1595,
  "type": "paperback",
  "pages": 200,
  "publisher": "PublisherA",
  "language": "English",
  "ISBN-10": "1234567890",
  "ISBN-13": "123-1234567890"
}
```

ハイブリッドノード用にタイプ LoadBalancer の Service を設定する

このトピックでは、Amazon EKS Hybrid Nodes で動作するアプリケーションに対してレイヤー 4 (L4) ロードバランシングを設定する方法について説明します。タイプ LoadBalancer の Kubernetes Service は、クラスターの外部に Kubernetes アプリケーションを公開するために使用されます。タイプ LoadBalancer の Service は、ワークロードのトラフィックを処理するために、クラウド環境やオンプレミス環境の物理的なロードバランサーインフラストラクチャと共によく使用されます。このロードバランサーインフラストラクチャは通常、環境固有のコントローラーでプロビジョニングされます。

AWS は、EKS Hybrid Nodes で実行されているタイプ LoadBalancer の Service 用に AWS Network Load Balancer (NLB) と Cilium をサポートしています。NLB を使用するか、Cilium を使用するかは、アプリケーショントラフィックのソースに基づきます。AWS では、アプリケーショントラフィックの発信元が AWS リージョンである場合には AWS NLB と AWS Load Balancer Controller を使用することを推奨しています。AWS では、アプリケーショントラフィックの発信元がローカルのオンプレミス環境またはエッジ環境である場合には Cilium に組み込みのロードバランシング機能を使用することを推奨しています。この機能は、お使いの環境にロードバランサーインフラストラクチャを搭載しているかどうかにかかわらず使用できます。

レイヤー 7 (L7) アプリケーショントラフィックのロードバランシングについては、「the section called “Ingress を設定する”」を参照してください。EKS によるロードバランシングの全般的な情報については、「Best Practices for Load Balancing」を参照してください。

AWS Network Load Balancer

ハイブリッドノード上で実行されているワークロードのターゲットタイプが `ip` の場合、AWS Load Balancer および NLB を使用できます。ターゲットタイプ `ip` を使用している場合、NLB はトラフィックをポッドに直接転送して、Service レイヤーネットワークパスをバイパスします。NLB がハイブリッドノードのポッド IP ターゲットに到達するためには、オンプレミスネットワークでオンプレミスポッド CIDR をルーティング可能である必要があります。また、AWS Load Balancer Controller はウェブフックを使用し、EKS コントロールプレーンからの直接通信が必要です。詳細については、「the section called “ウェブフックの設定”」を参照してください。

- サブネット設定要件については「the section called “ネットワーク負荷分散”」を参照し、AWS Network Load Balancer と AWS Load Balancer Controller の詳細については「the section called “Helm を使用してインストールする”」と「Best Practices for Load Balancing」を参照してください。
- AWS Network Load Balancer でタイプ `LoadBalancer` の Service に適用できる設定については、「AWS Load Balancer Controller NLB configurations」を参照してください。

前提条件

- 「the section called “CNI を設定する”」の手順に従って Cilium がインストールされていること。
- 「the section called “BGP を設定する”」の手順に従って Cilium BGP コントロールプレーンを有効にしていること。BGP を使用しない場合は、別の方法を使用して、オンプレミスネットワークでオンプレミスポッド CIDR をルーティング可能にする必要があります。詳細については、「the section called “ルーティング可能なリモートポッド CIDR”」を参照してください。
- コマンドライン環境に Helm がインストールされていること。「Setup Helm instructions」を参照してください。
- コマンドライン環境に eksctl がインストールされていること。「Setup eksctl instructions」を参照してください。

手順

1. ユーザーに代わって AWS API を呼び出すことを許可する、AWS Load Balancer Controller 用の IAM ポリシーをダウンロードします。

```
curl -O https://raw.githubusercontent.com/kubernetes-sigs/aws-load-balancer-
controller/refs/heads/main/docs/install/iam_policy.json
```

2. 前のステップでダウンロードしたポリシー を使用して、IAM ポリシーを作成します。

```
aws iam create-policy \
    --policy-name AWSLoadBalancerControllerIAMPolicy \
    --policy-document file://iam_policy.json
```

3. クラスター名 (CLUSTER_NAME)、AWS リージョン (AWS_REGION)、AWS アカウント ID (AWS_ACCOUNT_ID) の各値を自分の設定値に置き換えて、次のコマンドを実行します。

```
eksctl create iamserviceaccount \
    --cluster=CLUSTER_NAME \
    --namespace=kube-system \
    --name=aws-load-balancer-controller \
    --attach-policy-arn=arn:aws:iam::AWS_ACCOUNT_ID:policy/
AWSLoadBalancerControllerIAMPolicy \
    --override-existing-serviceaccounts \
    --region AWS_REGION \
    --approve
```

4. eks-charts Helm チャートリポジトリを追加します。AWS は、このリポジトリを GitHub で管理しています。

```
helm repo add eks https://aws.github.io/eks-charts
```

5. ローカル Helm リポジトリを更新して、最新のチャートがあることを確認します。

```
helm repo update eks
```

6. AWS Load Balancer コントローラをインストールします。クラスター名 (CLUSTER_NAME)、AWS リージョン (AWS_REGION)、VPC ID (VPC_ID)、AWS Load Balancer Controller Helm チャートバージョン (AWS_LBC_HELM_VERSION) の各値を自分の設定値に置き換えます。Helm チャートの最新バージョンを確認するには、`helm search repo eks/aws-load-balancer-controller --versions` を実行します。ハイブリッドノードと AWS クラウド内のノードの両方を使用して混合モードクラスターを実行している場合は、「the section called “AWS Load Balancer Controller”」の手順に従ってクラウドノード上で AWS Load Balancer Controller を実行できます。

```
helm install aws-load-balancer-controller eks/aws-load-balancer-controller \
  -n kube-system \
  --version AWS_LBC_HELM_VERSION \
  --set clusterName=CLUSTER_NAME \
  --set region=AWS_REGION \
  --set vpcId=VPC_ID \
  --set serviceAccount.create=false \
  --set serviceAccount.name=aws-load-balancer-controller
```

7. AWS Load Balancer Controller が正常にインストールされたことを確認します。

```
kubectl get -n kube-system deployment aws-load-balancer-controller
```

```
NAME                           READY   UP-TO-DATE   AVAILABLE   AGE
aws-load-balancer-controller   2/2     2            2           84s
```

8. `tcp-sample-app.yaml` という名前のファイルにサンプルアプリケーションを定義します。以下の例では、TCP ポートでシンプルな NGINX デプロイを使用しています。

```
apiVersion: apps/v1
kind: Deployment
metadata:
```

```
  name: tcp-sample-app
  namespace: default
spec:
  replicas: 3
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
        - name: nginx
          image: public.ecr.aws/nginx/nginx:1.23
          ports:
            - name: tcp
              containerPort: 80
```

9. デプロイをクラスターに適用します。

```
kubectl apply -f tcp-sample-app.yaml
```

10. `tcp-sample-service.yaml` という名前のファイルにデプロイ用のタイプ LoadBalancer の Service を定義します。

```
apiVersion: v1
kind: Service
metadata:
  name: tcp-sample-service
  namespace: default
  annotations:
    service.beta.kubernetes.io/aws-load-balancer-type: external
    service.beta.kubernetes.io/aws-load-balancer-nlb-target-type: ip
    service.beta.kubernetes.io/aws-load-balancer-scheme: internet-facing
spec:
  ports:
    - port: 80
      targetPort: 80
      protocol: TCP
  type: LoadBalancer
  selector:
    app: nginx
```

11.サービス設定をクラスターに適用します。

```
kubectl apply -f tcp-sample-service.yaml
```

12.Service 用に NLB をプロビジョニングするには、数分かかる場合があります。NLB がプロビジョニングされると、NLB デプロイの DNS 名に対応するアドレスが Service に割り当てられます。

```
kubectl get svc tcp-sample-service
```

```
NAME                 TYPE           CLUSTER-IP       EXTERNAL-IP
                                                 PORT(S)        AGE
tcp-sample-service   LoadBalancer   172.16.115.212   k8s-default-tcpsampl-xxxxxxxxxx-
xxxxxxxxxxxxxxxx.elb.<region>.amazonaws.com   80:30396/TCP   8s
```

13.NLB のアドレスを使用して Service にアクセスします。

```
curl k8s-default-tcpsampl-xxxxxxxxxx-xxxxxxxxxxxxxxxx.elb.<region>.amazonaws.com
```

以下に、出力例を示します。

```
<!DOCTYPE html>
<html>
<head>
<title>Welcome to nginx!</title>
[...]
```

14.作成した リソースをクリーンアップします。

```
kubectl delete -f tcp-sample-service.yaml
kubectl delete -f tcp-sample-app.yaml
```

Cilium クラスター内ロードバランシング

Cilium は、EKS Hybrid Nodes で実行されているワークロードのクラスター内ロードバランサーとして使用できます。お使いの環境にロードバランサーインフラストラクチャがない場合に便利です。Cilium のロードバランシング機能は、kube-proxy 置換、Load Balancer IP Address Management (IPAM)、BGP コントロールプレーンなどの Cilium 機能を組み合わせて構築されています。これらの機能が担当する作業は以下のとおりです。

- Cilium kube-proxy 置換: バックエンドポッドへの Service トラフィックのルーティングを処理します。
- Cilium Load Balancer IPAM: タイプ `LoadBalancer` の Service に割り当てることができる IP アドレスを管理します。
- Cilium BGP コントロールプレーン: Load Balancer IPAM が割り当てた IP アドレスをオンプレミスネットワークにアドバタイズします。

Cilium の kube-proxy 置換を使用していない場合でも、Cilium Load Balancer IPAM と BGP コントロールプレーンを使用して、タイプ LoadBalancer の Service 用に IP アドレスを配分して割り当てることができます。Cilium の kube-proxy 置換を使用しない場合、バックエンドポッドへの Service のロードバランシングは、デフォルトでは EKS の kube-proxy ルールと iptables ルールによって処理されます。

前提条件

- kube-proxy 置換が有効かどうかにかかわらず、「the section called “CNI を設定する”」の手順に従って Cilium がインストールされていること。Cilium の kube-proxy 置換を使用するには、v4.19.57、v5.1.16、v5.2.0 以降の最新の Linux カーネルでオペレーティングシステムを実行している必要があります。ハイブリッドノード上での用途に対応した最新バージョンのオペレーティングシステムであれば、Red Hat Enterprise Linux (RHEL) 8.x を除き、すべてこの基準を満たしています。
- 「the section called “BGP を設定する”」の手順に従って Cilium BGP コントロールプレーンを有効にしていること。BGP を使用しない場合は、別の方法を使用して、オンプレミスネットワークでオンプレミスポッド CIDR をルーティング可能にする必要があります。詳細については、「the section called “ルーティング可能なリモートポッド CIDR”」を参照してください。
- コマンドライン環境に Helm がインストールされていること。「Setup Helm instructions」を参照してください。

手順

1. `CiliumLoadBalancerIPPool` リソースを使用してタイプ Load Balancer の Service 用に LoadBalancer IP アドレス範囲を設定するファイルを `cilium-lbip-pool-loadbalancer.yaml` という名前で作成します。
   - `LB_IP_CIDR` を Load Balancer の IP アドレスに使用する IP アドレス範囲に置き換えます。単一の IP アドレスを選択するには、`/32` CIDR を使用します。詳細については、Cilium のドキュメントの「LoadBalancer IP Address Management」を参照してください。

- serviceSelector フィールドは、後続のステップで作成する Service の名前と一致するように設定されています。この設定の場合、このプールの IP は tcp-sample-service という名前の Service にのみ割り当てられます。

```
apiVersion: cilium.io/v2alpha1
kind: CiliumLoadBalancerIPPool
metadata:
  name: tcp-service-pool
spec:
  blocks:
  - cidr: "LB_IP_CIDR"
  serviceSelector:
    matchLabels:
      io.kubernetes.service.name: tcp-sample-service
```

2. CiliumLoadBalancerIPPool リソースをクラスターに適用します。

```
kubectl apply -f cilium-lbip-pool-loadbalancer.yaml
```

3. プール内に使用可能な IP アドレスが少なくとも 1 つあることを確認します。

```
kubectl get ciliumloadbalancerippools.cilium.io
```

```
NAME               DISABLED   CONFLICTING   IPS AVAILABLE   AGE
tcp-service-pool   false      False         1               24m
```

4. CiliumBGPAdvertisement リソースを使用して、次のステップで作成する Service 用にロードバランサー IP アドレスをアドバタイズするファイルを cilium-bgp-advertisement-loadbalancer.yaml という名前で作成します。Cilium BGP を使用していない場合は、このステップをスキップできます。Service に使用するロードバランサー IP アドレスは、最終ステップでサービスをクエリできるように、オンプレミスネットワークでルーティング可能である必要があります。

   - LoadBalancerIP をタイプ LoadBalancer の Service 用にのみアドバタイズするように、advertisementType フィールドは Service に設定され、service.addresses は LoadBalancerIP に設定されます。
   - selector フィールドは、後続のステップで作成する Service の名前と一致するように設定されています。この設定の場合、tcp-sample-service という名前の Service 用に LoadBalancerIP のみがアドバタイズされます。

```
apiVersion: cilium.io/v2alpha1
kind: CiliumBGPAdvertisement
metadata:
  name: bgp-advertisement-tcp-service
  labels:
    advertise: bgp
spec:
  advertisements:
    - advertisementType: "Service"
      service:
        addresses:
          - LoadBalancerIP
      selector:
        matchLabels:
          io.kubernetes.service.name: tcp-sample-service
```

5. CiliumBGPAdvertisement リソースをクラスターに適用します。Cilium BGP を使用していない場合は、このステップをスキップできます。

```
kubectl apply -f cilium-bgp-advertisement-loadbalancer.yaml
```

6. tcp-sample-app.yaml という名前のファイルにサンプルアプリケーションを定義します。以下の例では、TCP ポートでシンプルな NGINX デプロイを使用しています。

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: tcp-sample-app
  namespace: default
spec:
  replicas: 3
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
        - name: nginx
          image: public.ecr.aws/nginx/nginx:1.23
```

```
          ports:
            - name: tcp
              containerPort: 80
```

7. デプロイをクラスターに適用します。

```
kubectl apply -f tcp-sample-app.yaml
```

8. `tcp-sample-service.yaml` という名前のファイルにデプロイ用のタイプ LoadBalancer の Service を定義します。
   - Service オブジェクトの `lbipam.cilium.io/ips` 注釈を使用して、ロードバランサー IP プールから特定の IP アドレスをリクエストできます。Service 用に特定の IP アドレスをリクエストしない場合は、この注釈を削除してもかまいません。
   - レガシー AWS クラウドプロバイダーが Service の Classic Load Balancer を作成できないようにするには、`loadBalancerClass` 仕様フィールドが必要です。以下の例では、`io.cilium/bgp-control-plane` が Cilium の BGP コントロールプレーンをロードバランサークラスとして使用するように、このフィールドを設定しています。また、`io.cilium/l2-announcer` が Cilium の L2 Announcements 機能を使用するように、このフィールドを設定することもできます (現時点ではベータ版であり、AWS による正式なサポートはありません)。

```
apiVersion: v1
kind: Service
metadata:
  name: tcp-sample-service
  namespace: default
  annotations:
    lbipam.cilium.io/ips: "LB_IP_ADDRESS"
spec:
  loadBalancerClass: io.cilium/bgp-control-plane
  ports:
    - port: 80
      targetPort: 80
      protocol: TCP
  type: LoadBalancer
  selector:
    app: nginx
```

9. サービスをクラスターに適用します。Service は、アプリケーションへのアクセスに使用できる外部 IP アドレスと共に作成されます。

```
kubectl apply -f tcp-sample-service.yaml
```

10. Service が正常に作成されて、前のステップで作成した CiliumLoadBalancerIPPool から IP が割り当てられていることを確認します。

```
kubectl get svc tcp-sample-service
```

```
NAME                TYPE           CLUSTER-IP      EXTERNAL-IP     PORT(S)
 AGE
tcp-sample-service   LoadBalancer   172.16.117.76   LB_IP_ADDRESS   80:31129/TCP
 14m
```

11. kube-proxy 置換モードで Cilium を使用している場合は、次のコマンドを実行して、Cilium が Service のロードバランシングを処理していることを確認できます。以下の出力では、10.86.2.x アドレスは Service のバックエンドポッドのポッド IP アドレスです。

```
kubectl -n kube-system exec ds/cilium -- cilium-dbg service list
```

```
ID   Frontend              Service Type   Backend
...
41   LB_IP_ADDRESS:80/TCP   LoadBalancer   1 => 10.86.2.76:80/TCP (active)
                                           2 => 10.86.2.130:80/TCP (active)
                                           3 => 10.86.2.141:80/TCP (active)
```

12. Cilium が BGP を介してオンプレミスネットワークに IP アドレスをアドバタイズしていることを確認します。以下の例では、ハイブリッドノードが 5 つあり、それぞれが tcp-sample-service Service の LB_IP_ADDRESS をオンプレミスネットワークにアドバタイズしています。

```
Node                    VRouter      Prefix             NextHop   Age     Attrs
mi-026d6a261e355fba7    NODES_ASN
                   LB_IP_ADDRESS/32   0.0.0.0   12m3s   [{Origin: i} {Nexthop:
 0.0.0.0}]
mi-082f73826a163626e    NODES_ASN
                   LB_IP_ADDRESS/32   0.0.0.0   12m3s   [{Origin: i} {Nexthop:
 0.0.0.0}]
mi-09183e8a3d755abf6    NODES_ASN
                   LB_IP_ADDRESS/32   0.0.0.0   12m3s   [{Origin: i} {Nexthop:
 0.0.0.0}]
mi-0d78d815980ed202d    NODES_ASN
```

```
                  LB_IP_ADDRESS/32   0.0.0.0   12m3s   [{Origin: i} {Nexthop:
 0.0.0.0}]
mi-0daa253999fe92daa   NODES_ASN
                  LB_IP_ADDRESS/32   0.0.0.0   12m3s   [{Origin: i} {Nexthop:
 0.0.0.0}]
```

13.割り当てられたロードバランサー IP アドレスを使用して Service にアクセスします。

```
curl LB_IP_ADDRESS
```

以下に、出力例を示します。

```
<!DOCTYPE html>
<html>
<head>
<title>Welcome to nginx!</title>
[...]
```

14.作成した リソースをクリーンアップします。

```
kubectl delete -f tcp-sample-service.yaml
kubectl delete -f tcp-sample-app.yaml
kubectl delete -f cilium-lb-ip-pool.yaml
kubectl delete -f cilium-bgp-advertisement.yaml
```

ハイブリッドノード用に Kubernetes ネットワークポリシーを設定する

AWS は、EKS Hybrid Nodes と共に Cilium を CNI として使用する際に、ポッドの受信トラフィックと送信トラフィック用に Kubernetes ネットワークポリシー (レイヤー 3/レイヤー 4) をサポートします。AWS クラウド内のノードで EKS クラスターを実行している場合、AWS は Kubernetes ネットワークポリシー用の Amazon VPC CNI をサポートします。

このトピックでは、EKS Hybrid Nodes で Cilium および Kubernetes ネットワークポリシーを設定する方法について説明します。Kubernetes ネットワークポリシーの詳細については、Kubernetes のドキュメントの「Kubernetes Network Policies」を参照してください。

ネットワークポリシーの設定

考慮事項

- AWS は、ポッドの送受信に関するアップストリームの Kubernetes ネットワークポリシーと仕様をサポートしています。AWS は、現時点では `CiliumNetworkPolicy` や `CiliumClusterwideNetworkPolicy` をサポートしていません。
- `policyEnforcementMode` Helm 値を使用すると、デフォルトの Cilium ポリシーの適用動作を制御できます。デフォルトの動作では、すべての送受信トラフィックが許可されます。ネットワークポリシーによってエンドポイントが選択されると、default-deny 状態に遷移して、明示的に許可したトラフィックのみが許可されます。デフォルトポリシーモードとポリシー適用モードの詳細については、Cilium のドキュメントを参照してください。
- 既存の Cilium インストールで `policyEnforcementMode` を変更する場合は、Cilium エージェント DaemonSet を再起動して、変更後のポリシー適用モードを適用する必要があります。
- `namespaceSelector` および `podSelector` を使用すると、ラベルが一致する名前空間およびポッドに対してトラフィックを許可または拒否できます。`namespaceSelector` および `podSelector` を `matchLabels` または `matchExpressions` と共に使用すると、それぞれのラベルに基づいて名前空間とポッドを選択できます。
- `ingress.ports` および `egress.ports` を使用すると、ポートおよびプロトコルに対してトラフィックを許可または拒否できます。
- `ipBlock` フィールドでは、ポッド IP アドレス (#9209) に対してトラフィックを選択的に許可または拒否することはできません。ノード IP に `ipBlock` セレクターを使用する機能は、Cilium のベータ機能であり、AWS ではサポートされていません。
- Kubernetes ネットワークポリシーに使用可能なフィールドの詳細については、Kubernetes のドキュメントの「NetworkPolicy resource」を参照してください。

前提条件

- 「the section called “CNI を設定する”」の手順に従って Cilium がインストールされていること。
- コマンドライン環境に Helm がインストールされていること。「Setup Helm instructions」を参照してください。

手順

次の手順では、コンポーネントがアプリケーションの動作に必要な他のコンポーネントとのみ通信できるように、サンプルマイクロサービスアプリケーション用のネットワークポリシーを設定します。この手順では、Istio Bookinfo サンプルマイクロサービスアプリケーションを使用します。

Bookinfo アプリケーションは、4 つの個別のマイクロサービスで構成されており、以下のような関係があります。

- productpage。productpage マイクロサービスは、details マイクロサービスと reviews マイクロサービスを呼び出して、ページに値を入力します。
- details。details マイクロサービスには、書籍情報が含まれています。
- reviews。reviews マイクロサービスには、書籍レビューが含まれています。また、ratings マイクロサービスも呼び出します。
- ratings。ratings マイクロサービスには、書籍レビューに伴う書籍ランキング情報が含まれています。

1. サンプルアプリケーションを作成します。

```
kubectl apply -f https://raw.githubusercontent.com/istio/istio/refs/heads/master/
samples/bookinfo/platform/kube/bookinfo.yaml
```

2. サンプルアプリケーションが正常に動作していることを確認し、productpage マイクロサービスのポッド IP アドレスを書き留めます。後続のステップでは、このポッド IP アドレスを使用して、各マイクロサービスをクエリします。

```
kubectl get pods -o wide
```

```
NAME                              READY   STATUS    RESTARTS   AGE   IP
 NODE
details-v1-766844796b-9wff2       1/1     Running   0          7s    10.86.3.7
 mi-0daa253999fe92daa
productpage-v1-54bb874995-lwfgg   1/1     Running   0          7s    10.86.2.193
 mi-082f73826a163626e
ratings-v1-5dc79b6bcd-59njm       1/1     Running   0          7s    10.86.2.232
 mi-082f73826a163626e
reviews-v1-598b896c9d-p2289       1/1     Running   0          7s    10.86.2.47
 mi-026d6a261e355fba7
reviews-v2-556d6457d-djktc        1/1     Running   0          7s    10.86.3.58
 mi-0daa253999fe92daa
```

```
reviews-v3-564544b4d6-g8hh4       1/1     Running   0          7s    10.86.2.69
 mi-09183e8a3d755abf6
```

3. ネットワークポリシーをテストするために全体を通して使用されるポッドを作成します。このポッドは、access: true というラベルが付けられて default 名前空間に作成されることに注意してください。

```
kubectl run curl-pod --image=curlimages/curl -i --tty --labels=access=true --
namespace=default --overrides='{"spec": { "nodeSelector": {"eks.amazonaws.com/
compute-type": "hybrid"}}}' -- /bin/sh
```

4. productpage マイクロサービスへのアクセスをテストします。以下の例では、productpage ポッド (10.86.2.193) のポッド IP アドレスを使用して、マイクロサービスをクエリしています。この IP アドレスをお使いの環境の productpage ポッドのポッド IP アドレスに置き換えます。

```
curl -s http://10.86.2.193:9080/productpage | grep -o "<title>.*</title>"
```

```
<title>Simple Bookstore App</title>
```

5. exit と入力してテスト用 curl ポッドを終了し、次のコマンドを実行してポッドに再アタッチできます。

```
kubectl attach curl-pod -c curl-pod -i -t
```

6. 以降の手順でネットワークポリシーの効果を実証するには、まず BookInfo マイクロサービスに対するトラフィックをすべて拒否するネットワークポリシーを作成します。deny ネットワークポリシーを定義するファイルを network-policy-deny-bookinfo.yaml という名前で作成します。

```
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: deny-bookinfo
  namespace: default
spec:
  podSelector:
    matchExpressions:
    - key: app
      operator: In
      values: ["productpage", "details", "reviews", "ratings"]
  policyTypes:
```

```
  - Ingress
  - Egress
```

7. クラスターに deny ネットワークポリシーを適用します。

```
kubectl apply -f network-policy-default-deny-bookinfo.yaml
```

8. BookInfo アプリケーションへのアクセスをテストします。以下の例では、productpage ポッド (10.86.2.193) のポッド IP アドレスを使用して、マイクロサービスをクエリしています。この IP アドレスをお使いの環境の productpage ポッドのポッド IP アドレスに置き換えます。

```
curl http://10.86.2.193:9080/productpage --max-time 10
```

```
curl: (28) Connection timed out after 10001 milliseconds
```

9. productpage ネットワークポリシーを定義するファイルを network-policy-productpage.yaml という名前で作成します。このポリシーには、以下のルールがあります。
   - access: true というラベルが付いたポッド (前のステップで作成した curl ポッド) からの受信トラフィックを許可する
   - ポート 9080 で、details、reviews、ratings の各マイクロサービスに関する送信 TCP トラフィックを許可する
   - ポート 53 で、kube-system 名前空間内で動作する CoreDNS に関する送信 TCP/UDP トラフィックを許可する

```
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: productpage-policy
  namespace: default
spec:
  podSelector:
    matchLabels:
      app: productpage
  policyTypes:
  - Ingress
  - Egress
  ingress:
  - from:
    - podSelector:
```

```
        matchLabels:
          access: "true"
  egress:
  - to:
    - podSelector:
        matchExpressions:
        - key: app
          operator: In
          values: ["details", "reviews", "ratings"]
    ports:
    - port: 9080
      protocol: TCP
  - to:
    - namespaceSelector:
        matchLabels:
          kubernetes.io/metadata.name: kube-system
      podSelector:
        matchLabels:
          k8s-app: kube-dns
    ports:
    - port: 53
      protocol: UDP
    - port: 53
      protocol: TCP
```

10.クラスターに productpage ネットワークポリシーを適用します。

```
kubectl apply -f network-policy-productpage.yaml
```

11.curl ポッドに接続して、Bookinfo アプリケーションへのアクセスをテストします。productpage マイクロサービスへのアクセスが許可されるようになりましたが、他のマイクロサービスは依然として deny ネットワークポリシーの対象であるため、引き続き拒否されます。以下の例では、productpage ポッド (10.86.2.193) のポッド IP アドレスを使用して、マイクロサービスをクエリします。この IP アドレスをお使いの環境の productpage ポッドのポッド IP アドレスに置き換えます。

```
kubectl attach curl-pod -c curl-pod -i -t
```

```
curl -s http://10.86.2.193:9080/productpage | grep -o "<title>.*</title>"
<title>Simple Bookstore App</title>
```

```
curl -s http://10.86.2.193:9080/api/v1/products/1
{"error": "Sorry, product details are currently unavailable for this book."}
```

```
curl -s http://10.86.2.193:9080/api/v1/products/1/reviews
{"error": "Sorry, product reviews are currently unavailable for this book."}
```

```
curl -s http://10.86.2.193:9080/api/v1/products/1/ratings
{"error": "Sorry, product ratings are currently unavailable for this book."}
```

12details ネットワークポリシーを定義するファイルを network-policy-details.yaml という名前で作成します。このポリシーは、productpage マイクロサービスからの受信トラフィックのみを許可します。

```
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: details-policy
  namespace: default
spec:
  podSelector:
    matchLabels:
      app: details
  policyTypes:
  - Ingress
  ingress:
  - from:
    - podSelector:
        matchLabels:
          app: productpage
```

13reviews ネットワークポリシーを定義するファイルを network-policy-reviews.yaml という名前で作成します。このポリシーは、productpage マイクロサービスからの受信トラフィックと、ratings マイクロサービスと CoreDNS への送信トラフィックのみを許可します。

```
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: reviews-policy
  namespace: default
spec:
```

```
  podSelector:
    matchLabels:
      app: reviews
  policyTypes:
  - Ingress
  - Egress
  ingress:
  - from:
    - podSelector:
        matchLabels:
          app: productpage
  egress:
  - to:
    - podSelector:
        matchLabels:
          app: ratings
  - to:
    - namespaceSelector:
        matchLabels:
          kubernetes.io/metadata.name: kube-system
      podSelector:
        matchLabels:
          k8s-app: kube-dns
    ports:
    - port: 53
      protocol: UDP
    - port: 53
      protocol: TCP
```

14ratings ネットワークポリシーを定義するファイルを `network-policy-ratings.yaml` という名前で作成します。このポリシーは、productpage マイクロサービスと reviews マイクロサービスからの受信トラフィックのみを許可します。

```
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: ratings-policy
  namespace: default
spec:
  podSelector:
    matchLabels:
      app: ratings
  policyTypes:
```

```
  - Ingress
  ingress:
  - from:
    - podSelector:
        matchExpressions:
        - key: app
          operator: In
          values: ["productpage", "reviews"]
```

15.クラスターに details、reviews、ratings の各ネットワークポリシーを適用します。

```
kubectl apply -f network-policy-details.yaml
kubectl apply -f network-policy-reviews.yaml
kubectl apply -f network-policy-ratings.yaml
```

16.curl ポッドに接続して、Bookinfo アプリケーションへのアクセスをテストします。以下の例では、productpage ポッド (10.86.2.193) のポッド IP アドレスを使用して、マイクロサービスをクエリします。この IP アドレスをお使いの環境の productpage ポッドのポッド IP アドレスに置き換えます。

```
kubectl attach curl-pod -c curl-pod -i -t
```

details マイクロサービスをテストします。

```
curl -s http://10.86.2.193:9080/api/v1/products/1
```

```
{"id": 1, "author": "William Shakespeare", "year": 1595, "type": "paperback",
 "pages": 200, "publisher": "PublisherA", "language": "English", "ISBN-10":
 "1234567890", "ISBN-13": "123-1234567890"}
```

reviews マイクロサービスをテストします。

```
curl -s http://10.86.2.193:9080/api/v1/products/1/reviews
```

```
{"id": "1", "podname": "reviews-v1-598b896c9d-p2289", "clustername": "null",
 "reviews": [{"reviewer": "Reviewer1", "text": "An extremely entertaining play
 by Shakespeare. The slapstick humour is refreshing!"}, {"reviewer": "Reviewer2",
 "text": "Absolutely fun and entertaining. The play lacks thematic depth when
 compared to other plays by Shakespeare."}]}
```

ratings マイクロサービスをテストします。

```
curl -s http://10.86.2.193:9080/api/v1/products/1/ratings
```

```
{"id": 1, "ratings": {"Reviewer1": 5, "Reviewer2": 4}}
```

17.この手順で作成したリソースをクリーンアップします。

```
kubectl delete -f network-policy-deny-bookinfo.yaml
kubectl delete -f network-policy-productpage.yaml
kubectl delete -f network-policy-details.yaml
kubectl delete -f network-policy-reviews.yaml
kubectl delete -f network-policy-ratings.yaml
kubectl delete -f https://raw.githubusercontent.com/istio/istio/refs/heads/master/
samples/bookinfo/platform/kube/bookinfo.yaml
kubectl delete pod curl-pod
```

ハイブリッドノードの概念

Amazon EKS Hybrid Nodes を使用すると、オンプレミス環境またはエッジ環境で実行されている物理マシンまたは仮想マシンを、AWS クラウドで実行されている Amazon EKS クラスターに接続できます。このアプローチには多くの利点がありますが、単一のネットワーク環境で Kubernetes クラスターを運用することに慣れているユーザーにとっては、新しいネットワークの概念やアーキテクチャが導入されることにもなります。

以下のセクションでは、EKS Hybrid Nodes での Kubernetes とネットワーキングの概念、およびハイブリッドアーキテクチャにおけるトラフィックフローについて詳しく説明します。これらのセクションを理解するには、ポッド、ノード、サービス、Kubernetes コントロールプレーン、kubelet、kube-proxy の概念など、Kubernetes ネットワークに関する基本的な知識に精通している必要があります。

これらのページは、「the section called “ネットワーキングの概念”」、「the section called “Kubernetes の概念”」、「the section called “トラフィックフロー”」の順に読むことをお勧めします。

トピック

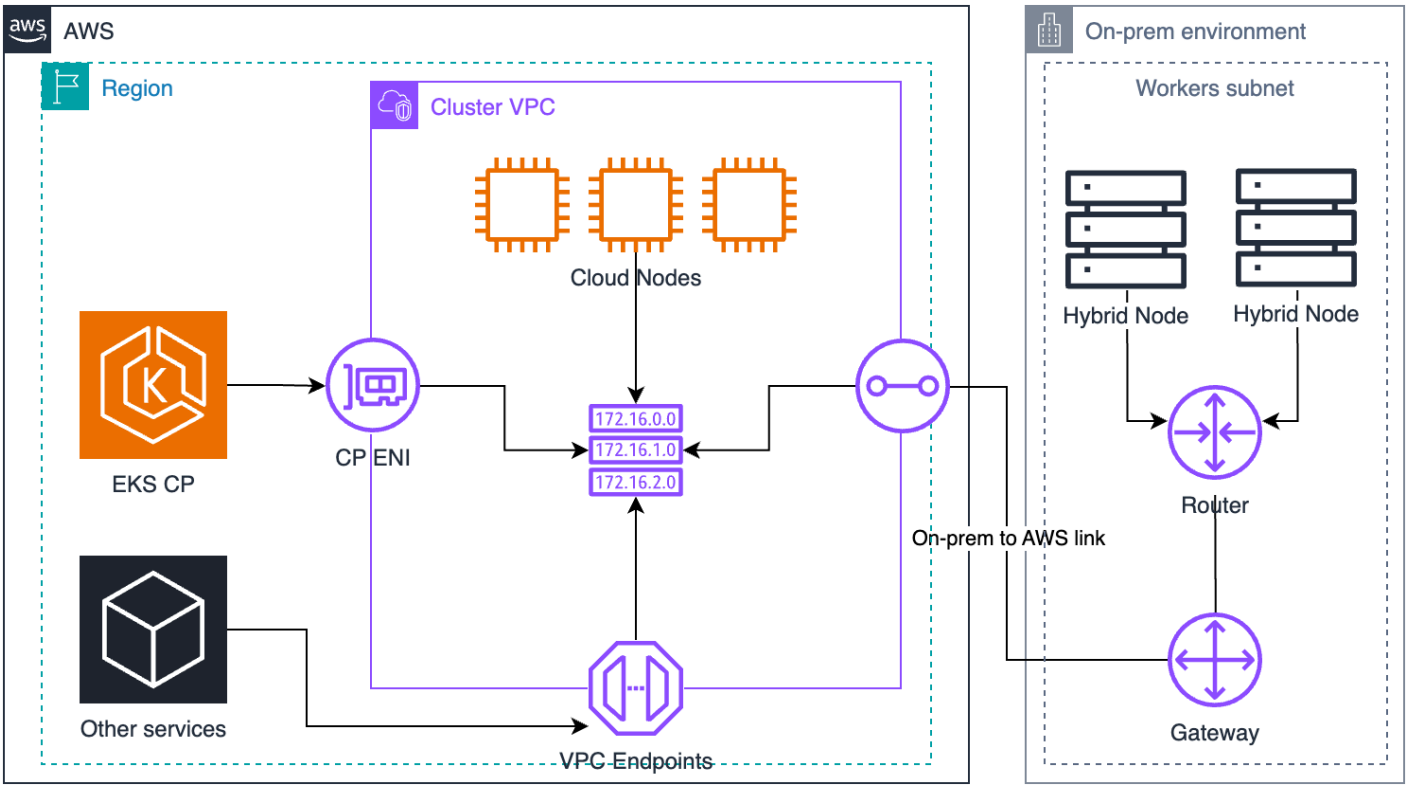
- ハイブリッドノードにおけるネットワーキングの概念
- ハイブリッドノードにおける Kubernetes の概念

- ハイブリッドノード向けネットワークトラフィックフロー

ハイブリッドノードにおけるネットワーキングの概念

このセクションでは、EKS Hybrid Nodes のネットワークトポロジー設計の際に考慮すべき、コアネットワーキングの概念と制約について詳しく説明します。

EKS Hybrid Nodes におけるネットワーキングの概念



ネットワークハブとしての VPC

クラウドの境界を越えるすべてのトラフィックは VPC を経由しますが、これには、AWS で稼働する EKS コントロールプレーンまたはポッドや、ハイブリッドノードまたはそうしたノードで稼働しているポッド間のトラフィックも含まれます。クラスターの VPC はハイブリッドノードと残りのクラスター部分のネットワークハブとして機能している、と考えるとよいでしょう。このアーキテクチャによってトラフィックとそのルーティングを完全に制御できますが、これを導入するには、VPC のルート、セキュリティグループ、ファイアウォールを適切に設定する責任を負うことにもなります。

EKS コントロールプレーンから VPC に向かうトラフィック

EKS コントロールプレーンにより、Elastic Network Interface (ENI) が VPC にアタッチされ、これらの ENI によって、EKS API サーバーとの間でトラフィックが処理されます。EKS コントロールプレーンの ENI の配置は、クラスターを設定する際に制御します。なぜなら、クラスターの作成中に渡すサブネットに ENI がアタッチされるからです。

EKS では、セキュリティグループを、サブネットにアタッチする ENI に関連付けます。こうしたセキュリティグループにより、EKS コントロールプレーンとの間のトラフィックを、ENI を介して許可します。この点が EKS Hybrid Nodes では重要となります。ハイブリッドノードとそこで稼働しているポッドから EKS コントロールプレーン ENI へのトラフィックを許可する必要があるからです。

リモートノードネットワーク

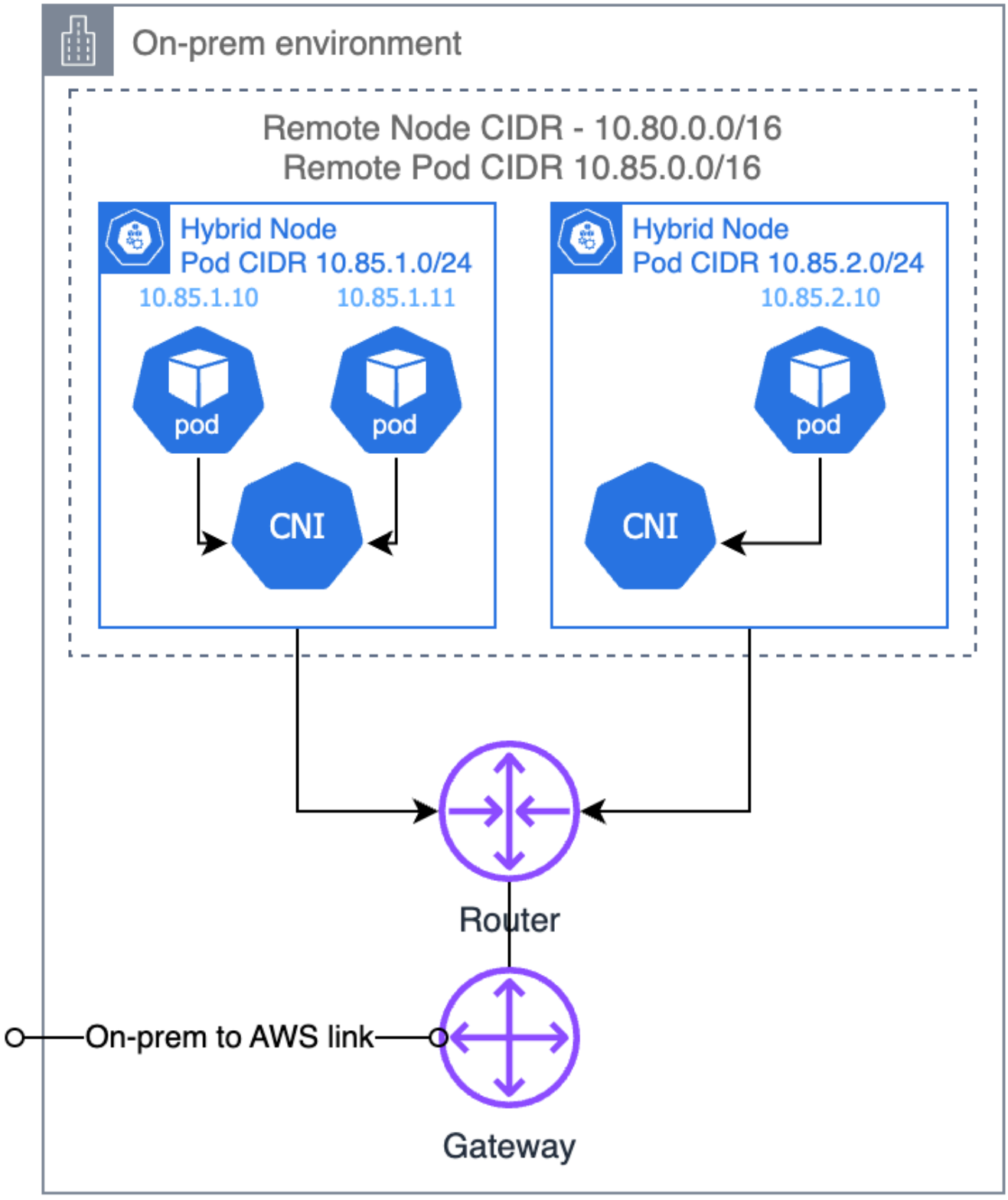
リモートノードネットワーク、具体的にはリモートノード CIDR は、ハイブリッドノードとして使用するマシンに割り当てる IP アドレスの範囲を意味します。ハイブリッドノードをプロビジョニングすると、それらはオンプレミスのデータセンターやエッジロケーションに配置されます。これらのネットワークドメインは、EKS コントロールプレーンおよび VPC のドメインとは異なります。また、各ハイブリッドノードには、IP アドレスが 1 つまたは複数あり、VPC 内のサブネットとは異なるリモートノード CIDR に基づいて設定されています。

こうしたリモートノード CIDR を使用して EKS クラスターを構成すると、EKS でそれが認識され、ハイブリッドノードの IP アドレスを宛先としたすべてのトラフィック (kubelet API へのリクエストなど) をクラスターの VPC 経由でルーティングできるようになります。`kubelet` API への接続は、`kubectl attach`、`kubectl cp`、`kubectl exec`、`kubectl logs`、`kubectl port-forward` の各コマンドで使用されます。

リモートポッドネットワーク

ここでのリモートポッドネットワークとは、ハイブリッドノードで稼働するポッドに割り当てる IP アドレスの範囲を意味します。一般的には、これらの範囲で CNI を設定すると、CNI の IP Address Manager (IPAM) 機能によって、その範囲の一部が各ハイブリッドノードに割り当てられます。また、新規作成したポッドには、ポッドをスケジュールしたノードに割り当てた IP アドレスからアドレスが割り当てられます。

こうしたリモートポッド CIDR を使用して EKS クラスターを設定すると、EKS コントロールプレーンでそれが認識され、ハイブリッドノードで稼働しているポッドを宛先とするすべてのトラフィック (ウェブフックとの通信など) をクラスターの VPC 経由でルーティングできます。



オンプレミスから VPC に向かうトラフィック

ハイブリッドノードに使用するオンプレミスネットワークのトラフィックは、EKS クラスターに使用する VPC にルーティングする必要があります。オンプレミスネットワークを VPC に接続する場合、複数用意されている Network-to-Amazon VPC 接続オプションを利用することも、独自の VPN ソリューションを使用することもできます。

ここで重要なのは、AWS クラウド側の VPC とオンプレミスネットワーク間のルーティングを正しく設定することで、両方のネットワークの接続点を介して、適切なトラフィックが相互にルーティングされるようにすることです。

VPC では、リモートノードおよびリモートポッドネットワークに向かうすべてのトラフィックを、オンプレミスネットワークとの接続点 (ゲートウェイ) 経由でルーティングする必要があります。サブネットの一部に異なるルートテーブルがある場合は、ハイブリッドノードとそこで稼働しているポッドのルートを各ルートテーブルに設定する必要があります。これは、EKS コントロールプレーンの ENI がアタッチされるサブネットや、ハイブリッドノードと通信する必要のある EC2 のノードまたはポッドが含まれるサブネットにも当てはまります。

オンプレミスネットワークでは、ネットワークの設定によって、EKS クラスターの VPC や、ハイブリッドノードに必要なその他の AWS サービスとの間で生じるトラフィックを許可する必要があります。EKS クラスターのトラフィックは、ゲートウェイを双方向に通過することになります。

ネットワーキング上の制約

完全にルーティングされたネットワーク

ここでの主な制約は、EKS コントロールプレーンとすべてのノード (クラウドまたはハイブリッドのノード) で生じるトラフィックが完全にルーティングされるネットワークを形成する必要があることです。つまり、すべてのノードが IP アドレスを使用してレイヤー 3 で互いに到達可能でなければなりません。

EKS コントロールプレーンとクラウドノードはフラットなネットワーク (VPC) 内にあるため、相互に到達可能な状態にありますが、ハイブリッドノードは異なるネットワークドメインに存在しています。そのため、VPC とオンプレミスネットワークでルーティング設定を追加して、ハイブリッドノードと他のクラスター部分との間で生じるトラフィックをルーティングする必要があります。ハイブリッドノードが互いに到達可能で、VPC からも到達可能な場合、ハイブリッドノードは、単一のフラットネットワークに配置することも、セグメント化した複数のネットワークに配置することもできます。

ルーティング可能なリモートポッド CIDR

EKS コントロールプレーンがハイブリッドノードで稼働中のポッド (ウェブフックやメトリクスサーバーなど) と通信したり、クラウドノードで稼働中のポッドがハイブリッドノードで稼働中のポッド

と通信 (ワークロードの East-West 通信) したりするには、リモートポッドの CIDR へのトラフィックが VPC からルーティング可能でなければなりません。つまり、VPC では、ポッドの CIDR に向かうトラフィックをゲートウェイ経由でオンプレミスネットワークにルーティングでき、オンプレミスネットワークでは、ポッドのトラフィックを適切なノードにルーティングできる必要があります。

ここで重要なのは、VPC とオンプレミスでポッドのルーティング要件に違いがあることです。VPC には、リモートポッドへのトラフィックがゲートウェイを通過する必要があることを認識させるだけで済み、リモートポッドの CIDR が 1 つしかない場合は、必要なルートも 1 つのみです。

この要件は、オンプレミスネットワーク内の、ハイブリッドノードと同じサブネットでローカルルーターに至るまでに生じるすべてのホップに当てはまります。このルーターは、各ノードに割り当てられたポッド CIDR スライスを認識する必要がある唯一のルーターです。これにより、特定のポッドに向かうトラフィックを、そのポッドがスケジュールされているノードに配信できます。

オンプレミスのポッド CIDR へのこうしたルートを、ローカルのオンプレミスルーターから VPC ルートテーブルに伝達することもできますが、これは必須ではありません。オンプレミスのポッド CIDR を頻繁に変更しているため、VPC ルートテーブルを更新してそれらの変更を反映する必要がある場合は、オンプレミスのポッド CIDR を VPC ルートテーブルに伝達することが推奨されます。しかし、こうしたケースはまれです。

オンプレミスのポッド CIDR へのルーティングは、必ずしも設定する必要はありません。ハイブリッドノードでウェブフックを実行する必要がない場合、またはクラウドノード上のポッドがハイブリッドノード上のポッドと通信する必要がない場合は、オンプレミスネットワークのポッド CIDR へのルーティングを設定しなくても済みます。

では、ハイブリッドノードを使用するときに、オンプレミスのポッド CIDR 宛てトラフィックをルーティング可能にする理由について説明します。

VPC CNI を備えた EKS をクラウドノードに使用している場合、VPC CNI によって VPC からポッドに直接 IP が割り当てられます。つまり、クラウドポッドも EKS コントロールプレーンもポッド IP に直接到達できるため、特別なルーティングは必要ありません。

オンプレミスでポッドが稼働している場合 (クラウドで他の CNI を使用)、ポッドは、通常、分離されたオーバーレイネットワークで実行され、CNI によってポッド間のトラフィック配信が処理されます。これは通常、カプセル化によって行われます。つまり、ポッド間のトラフィックをノード間のトラフィックに変換し、両端でカプセル化とカプセル化解除を行います。そのため、ノードとルーターに設定をさらに追加する必要はありません。

ハイブリッドノードを使用したネットワークは、両方のトポロジーが組み合わさっているという点で独特なものです。つまり、EKS コントロールプレーンとクラウドノード (VPC CNI を使用) は、ノー

ドとポッドが含まれるフラットなネットワークを想定して動作し、一方、ハイブリッドノードで稼働するポッドは、オーバーレイネットワーク内に存在し、カプセル化には VXLAN (Cilium のデフォルト) が使用されます。ハイブリッドノードで稼働するポッドは、オンプレミスネットワークと VPC 間のルーティングが可能であることを前提に、EKS コントロールプレーンとクラウドノードで稼働中のポッドに到達できます。ただし、オンプレミスネットワークのポッド CIDR 宛てトラフィックをルーティングしておらず、オーバーレイネットワークに到達したトラフィックを適切なノードに転送できない場合、オンプレミスのポッド IP に戻るトラフィックは、最終的にドロップされます。

ハイブリッドノードにおける Kubernetes の概念

このページでは、EKS Hybrid Nodes システムアーキテクチャを支える Kubernetes の主要な概念について詳しく説明します。

VPC の EKS コントロールプレーン

EKS コントロールプレーン ENI の IP は、`default` デフォルトの名前空間にある `kubernetes` `Endpoints` オブジェクトに保存されます。EKS では、ENI の新規作成や削除が行われると、このオブジェクトが更新され、IP リストが常に最新状態に保たれます。

これらのエンドポイントは、`kubernetes` サービスを通じて使用できます。このサービスも `default` の名前空間に存在します。この `ClusterIP` タイプのサービスには、クラスターのサービス CIDR に含まれる最初の IP が割り当てられます。例えば、サービス CIDR が `172.16.0.0/16` の場合、サービス IP は `172.16.0.1` になります。

通常、ポッドは (稼働がクラウドノード上かハイブリッドノード上かに関係なく) そのように EKS Kubernetes API サーバーにアクセスします。サービス IP を宛先 IP として使用し、これをいずれかの EKS コントロールプレーン ENI に設定されている実際の IP に変換します。こうした動作の主な例外として `kube-proxy` があります。これによって変換を設定するからです。

EKS API サーバーエンドポイント

EKS API サーバーには `kubernetes` サービス IP 以外の方法でもアクセスできます。EKS では、クラスター作成時に Route53 DNS 名も作成されます。これが、EKS の `endpoint` API アクションを呼び出す際に、EKS クラスターの `DescribeCluster` フィールド値になります。

```
{
    "cluster": {
        "endpoint": "https://xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx.gr7.us-
west-2.eks.amazonaws.com",
        "name": "my-cluster",
```

```
        "status": "ACTIVE"
    }
}
```

パブリックエンドポイントアクセスクラスター、もしくはパブリックおよびプライベートエンドポイントアクセスクラスターでは、ハイブリッドノードで、この DNS 名がインターネット経由でルーティング可能なパブリック IP に解決されます (デフォルトの場合)。プライベートエンドポイントアクセスクラスターの場合、DNS 名は、EKS コントロールプレーン ENI のプライベート IP に解決されます。

kubelet と kube-proxy は、そのように Kubernetes API サーバーにアクセスします。すべての Kubernetes クラスタートラフィックを VPC 経由にする場合、クラスターをプライベートアクセスモードで設定するか、オンプレミスの DNS サーバーを変更して EKS クラスターエンドポイントを EKS コントロールプレーン ENI のプライベート IP に解決する必要があります。

kubelet エンドポイント

kubelet では複数の REST エンドポイントが公開されており、これによってシステムの他の要素は、kubelet と通信し、各ノードの情報を収集しています。ほとんどのクラスターにおいて、kubelet サーバーに向かうトラフィックの大部分は、コントロールプレーンで生じたものですが、特定のモニタリングエージェントも kubelet と通信する場合があります。

このインターフェイスを通じて、kubelet では、ログの取得 (kubectl logs)、コンテナ内でのコマンド実行 (kubectl exec)、トラフィックのポート転送 (kubectl port-forward) といった各種リクエストが処理されます。こうしたリクエストでは、基盤のコンテナランタイム環境との通信が kubelet を介して行われるため、クラスター管理者や開発者にとってシームレスなユーザー体験が実現します。

この API で最も一般的なコンシューマーとなるのは Kubernetes API サーバーです。上記の kubectl コマンドのいずれかを使用すると、kubectl が API サーバーに API リクエストを送信し、その後 API サーバーが、ポッドが稼働しているノードの kubelet API を呼び出します。そのため、多くの場合、ノード IP が EKS コントロールプレーンから到達可能である必要があります。また、ノードルートを誤って設定すると、ポッドが稼働中であっても、ログや exec コマンドを利用できません。

ノード IP

EKS コントロールプレーンは、ノードと通信する際に、Node オブジェクトのステータス (status.addresses) で報告されるアドレスの 1 つを使用します。

EKS クラウドノードでは、一般的に、ノード登録時に kubelet が EC2 インスタンスのプライベート IP を `InternalIP` として報告します。その後、Cloud Controller Manager (CCM) がこの IP を検証し、それが EC2 インスタンスに属していることを確認します。さらに、CCM は通常、インスタンスのパブリック IP (`ExternalIP`) と DNS 名 (`InternalDNS` と `ExternalDNS`) をノードステータスに追加します。

ただし、ハイブリッドノードには CCM はありません。EKS Hybrid Nodes CLI (`nodeadm`) を使用してハイブリッドノードを登録すると、CCM を介さずにノードのステータスでマシンの IP が直接報告されるように kubelet が設定されます。

```
apiVersion: v1
kind: Node
metadata:
  name: my-node-1
spec:
  providerID: eks-hybrid:///us-west-2/my-cluster/my-node-1
status:
  addresses:
  - address: 10.1.1.236
    type: InternalIP
  - address: my-node-1
    type: Hostname
```

マシンに複数の IP がある場合、kubelet は独自のロジックに従ってそれらの 1 つを選択します。選択される IP アドレスは `--node-ip` フラグで制御でき、このフラグは、`nodeadm` config の `spec.kubelet.flags` に渡すことが可能です。`Node` オブジェクトで報告された IP アドレスにのみ VPC からのルートが必要であり、マシンには、クラウドからアクセスできない他の IP アドレスを割り当てることができます。

kube-proxy

`kube-proxy` は、各ノードのネットワークレイヤーでサービス抽象化を実装する役割を果たし、Kubernetes サービスに向かうトラフィックのネットワークプロキシおよびロードバランサーとして機能します。また、`kube-proxy` は、Kubernetes API サーバーを監視し、サービスとエンドポイントに関連した変更を継続的に確認することで、基盤となるホストのネットワークルールを動的に更新し、適切なトラフィック転送を維持します。

`iptables` モードの `kube-proxy` は、複数の `netfilter` チェーンをプログラミングして、サービストラフィックを処理します。こうしたルールによって、以下の階層が形成されます。

1. KUBE-SERVICES チェーン: すべてのサービストラフィックのエントリポイント。各サービスの ClusterIP とポートに一致するルールが定義されています。
2. KUBE-SVC-XXX チェーン: サービス固有のチェーンに、各サービスの負荷分散ルールが定義されています。
3. KUBE-SEP-XXX チェーン: エンドポイント固有のチェーンに、実際の DNAT ルールが定義されています。

次に示す default 名前空間内のサービス test-server の状況を確認してみましょう。* サービス ClusterIP: 172.16.31.14 * サービスポート: 80 * バッキングポッド: 10.2.0.110, 10.2.1.39, 10.2.2.254

iptables ルールを検査するとき (iptables-save 0 grep -A10 KUBE-SERVICES を使用):

1. KUBE-SERVICES チェーンには、以下のように、サービスに一致するルールがあります。

```
-A KUBE-SERVICES -d 172.16.31.14/32 -p tcp -m comment --comment "default/test-server
 cluster IP" -m tcp --dport 80 -j KUBE-SVC-XYZABC123456
```

   - このルールは、172.16.31.14:80 宛てのパケットに一致させる
   - コメントは、このルールの目的を示している: default/test-server cluster IP
   - 一致するパケットは KUBE-SVC-XYZABC123456 チェーンにジャンプさせる
2. KUBE-SVC-XYZABC123456 チェーンには、確率ベースの負荷分散ルールが定義されています。

```
-A KUBE-SVC-XYZABC123456 -m statistic --mode random --probability 0.33333333349 -j
 KUBE-SEP-POD1XYZABC
-A KUBE-SVC-XYZABC123456 -m statistic --mode random --probability 0.50000000000 -j
 KUBE-SEP-POD2XYZABC
-A KUBE-SVC-XYZABC123456 -j KUBE-SEP-POD3XYZABC
```

   - 最初のルール: 33.3% の確率で KUBE-SEP-POD1XYZABC にジャンプさせる
   - 2 番目のルール: 50% の確率で残りのトラフィック (全体の 33.3%) を KUBE-SEP-POD2XYZABC にジャンプさせる
   - 最後のルール: 残りのすべてのトラフィック (合計の 33.3%) を KUBE-SEP-POD3XYZABC にジャンプさせる
3. 個々の KUBE-SEP-XXX チェーンで、DNAT (宛先 NAT) を実行します。

```
-A KUBE-SEP-POD1XYZABC -p tcp -m tcp -j DNAT --to-destination 10.2.0.110:80
```

```
-A KUBE-SEP-POD2XYZABC -p tcp -m tcp -j DNAT --to-destination 10.2.1.39:80
-A KUBE-SEP-POD3XYZABC -p tcp -m tcp -j DNAT --to-destination 10.2.2.254:80
```

- これらの DNAT ルールでは、宛先 IP とポートを書き換えて、トラフィックを特定のポッドに転送します。
- 各ルールによって、トラフィックの約 33.3% を処理し、10.2.0.110、10.2.1.39、10.2.2.254 の間で均等な負荷分散を行います。

このマルチレベルチェーン構造により、kube-proxy では、データパスにプロキシプロセスを必要とせずに、カーネルレベルのパケット操作によってサービスの負荷分散およびリダイレクトを効率的に実装できます。

Kubernetes 運用への影響

ノードで kube-proxy の機能が失われると、ノードはサービストラフィックを適切にルーティングできなくなり、クラスターサービスに依存するポッドのタイムアウトや接続の失敗が発生します。この問題が特に深刻になりうるのは、ノードが最初に登録されたときです。CNI では、ポッドネットワークを設定する前に、Kubernetes API サーバーと通信して、ノードのポッド CIDR などの情報を取得する必要があります。それを行うには、kubernetes サービス IP を使用します。ただし、kube-proxy が起動できなかったり、適切な iptables ルールを設定できなかったりした場合、kubernetes サービス IP へのリクエストは EKS コントロールプレーン ENI の実際の IP に変換されません。その結果、CNI がクラッシュループに陥り、どのポッドも正しく稼働しなくなります。

前述したように、ポッドは Kubernetes API サーバーとの通信に kubernetes のサービス IP を使用しますが、そのように動作させるには、最初に kube-proxy に iptables ルールを設定する必要があります。

kube-proxy は、どのように API サーバーと通信するのでしょうか?

kube-proxy は、Kubernetes API サーバーの実際の IP アドレス、またはそれらに解決される DNS 名を使用するように設定する必要があります。EKS の場合、デフォルトの kube-proxy が、クラスター作成時に EKS によって作成される Route53 DNS 名を指すように設定されます。この値は、kube-system 名前空間の kube-proxy ConfigMap で確認できます。この ConfigMap の内容は、kube-proxy ポッドに注入される kubeconfig であるため、clusters0.cluster.server フィールドを確認してください。この値は、EKS DescribeCluster API を呼び出した結果にある、EKS クラスターの endpoint フィールドと一致しています。

```
apiVersion: v1
```

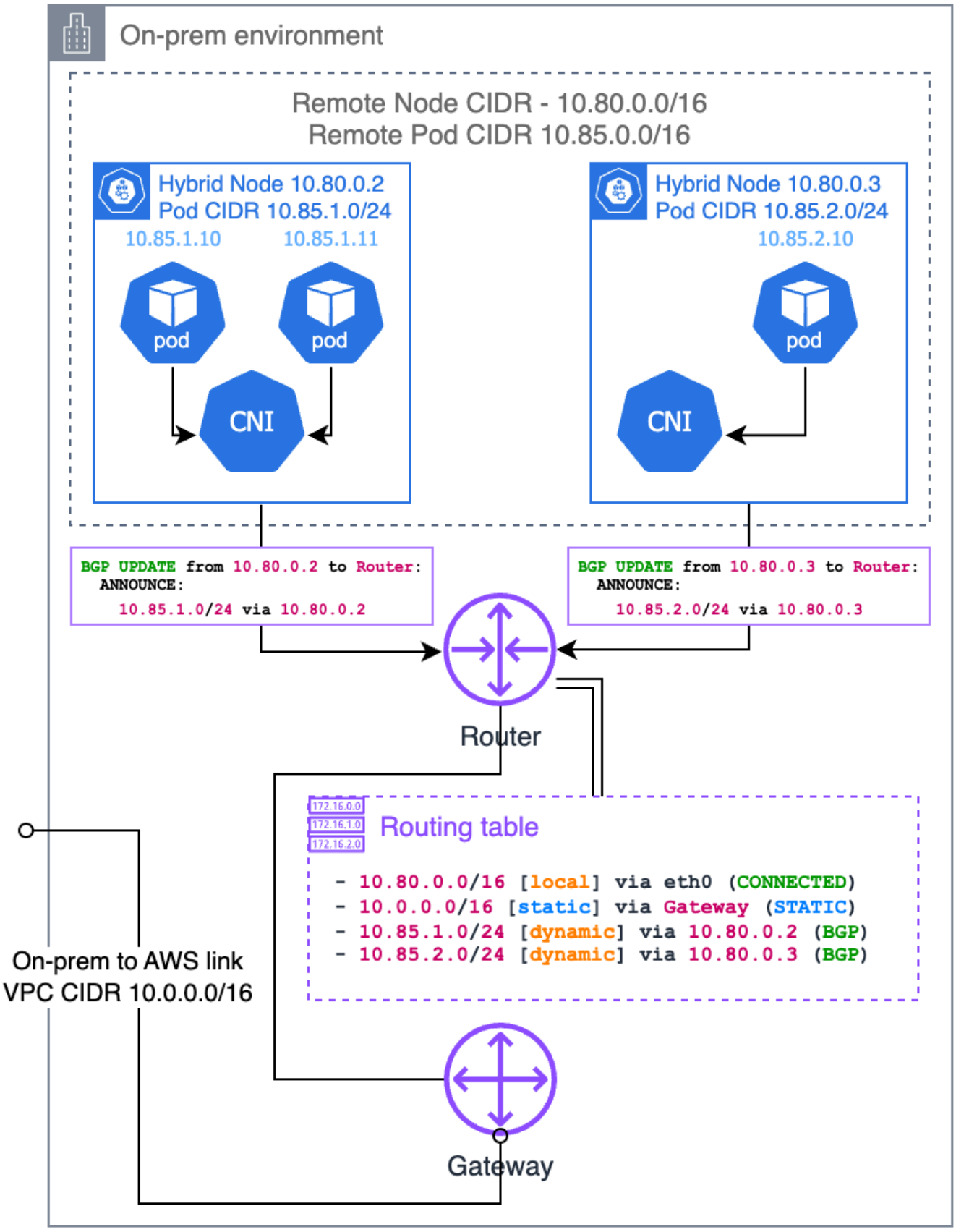
```
data:
  kubeconfig: |-
    kind: Config
    apiVersion: v1
    clusters:
    - cluster:
        certificate-authority: /var/run/secrets/kubernetes.io/serviceaccount/ca.crt
        server: https://xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx.gr7.us-
west-2.eks.amazonaws.com
      name: default
    contexts:
    - context:
        cluster: default
        namespace: default
        user: default
      name: default
    current-context: default
    users:
    - name: default
      user:
        tokenFile: /var/run/secrets/kubernetes.io/serviceaccount/token
kind: ConfigMap
metadata:
  name: kube-proxy
  namespace: kube-system
```

ルーティング可能なリモートポッド CIDR

この the section called “ネットワーキングの概念” のページでは、ハイブリッドノードでウェブフックを実行したり、クラウドノードで稼働しているポッドがハイブリッドノードで稼働しているポッドと通信したりするための要件について詳しく説明します。ここでは重要な要件があります。特定のポッド IP を担当するノードを、オンプレミスルーターに認識させる必要があることです。これを実現する方法として、ボーダーゲートウェイプロトコル (BGP)、静的ルート、アドレス解決プロトコル (ARP) プロキシなどが挙げられます。以下のセクションでは、これらの方法について説明します。

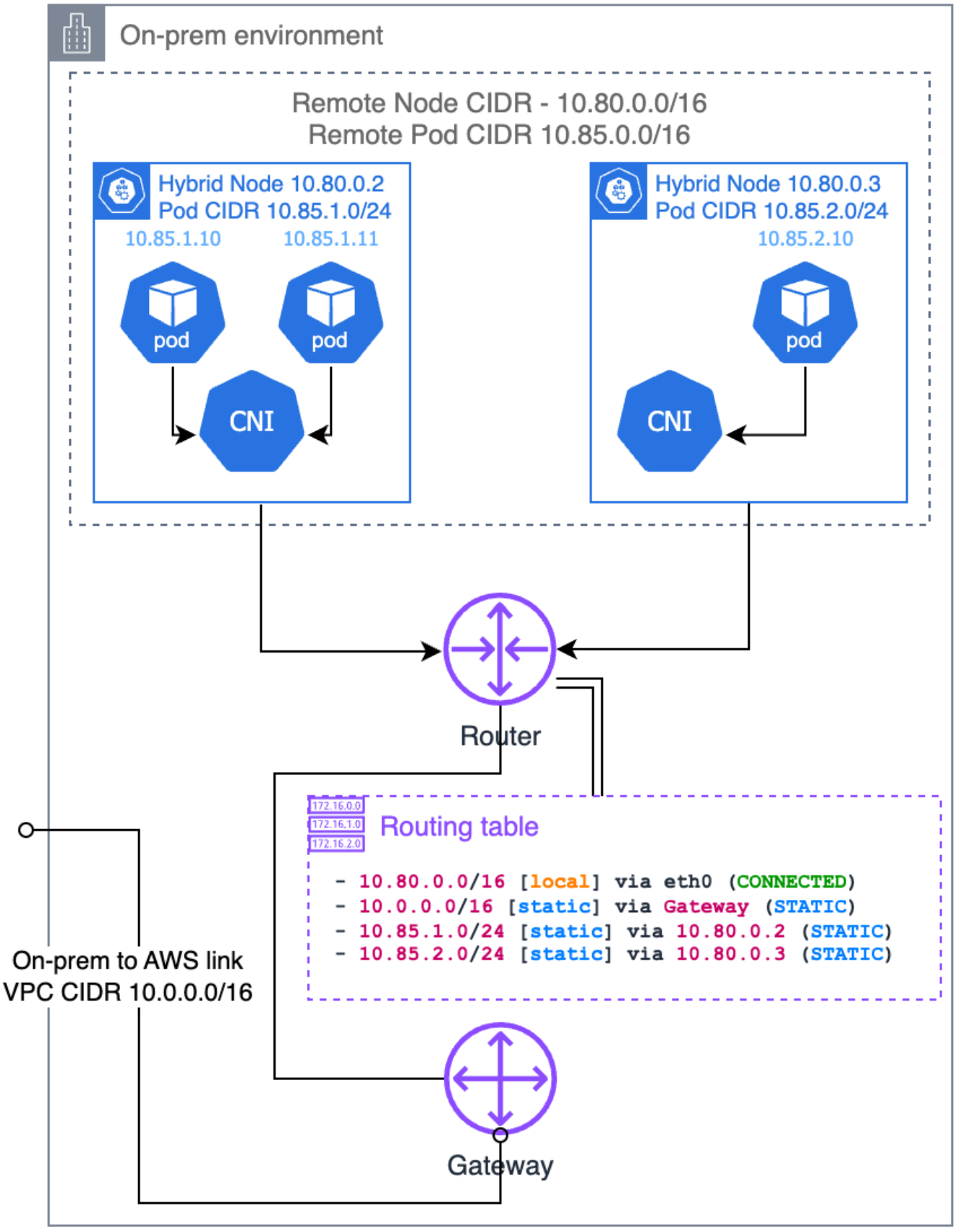
ボーダーゲートウェイプロトコル (BGP)

CNI でサポートされている場合 (Cilium や Calico など)、CNI の BGP モードを使用して、ノードごとのポッド CIDR へのルートをノードからローカルルーターに伝達できます。CNI の BGP モードを使用する場合、CNI は仮想ルーターとして機能するため、ローカルルーターは、ポッド CIDR が別のサブネットに属し、ノードはそのサブネットへのゲートウェイであると認識します。


On-prem environment
Remote Node CIDR - 10.80.0.0/16
Remote Pod CIDR 10.85.0.0/16
Hybrid Node 10.80.0.2
Pod CIDR 10.85.1.0/24
10.85.1.10
10.85.1.11
pod
pod
CNI
Hybrid Node 10.80.0.3
Pod CIDR 10.85.2.0/24
10.85.2.10
pod
CNI
BGP UPDATE from 10.80.0.2 to Router:
ANNOUNCE:
10.85.1.0/24 via 10.80.0.2
BGP UPDATE from 10.80.0.3 to Router:
ANNOUNCE:
10.85.2.0/24 via 10.80.0.3
Router
172.16.0.0
172.16.1.0
172.16.2.0
Routing table
- 10.80.0.0/16 [local] via eth0 (CONNECTED)
- 10.0.0.0/16 [static] via Gateway (STATIC)
- 10.85.1.0/24 [dynamic] via 10.80.0.2 (BGP)
- 10.85.2.0/24 [dynamic] via 10.80.0.3 (BGP)
On-prem to AWS link
VPC CIDR 10.0.0.0/16
Gateway

静的ルート

または、ローカルルーターで静的ルートを設定することもできます。この方法を取ると、オンプレミスポッド CIDR を VPC に非常に簡単にルーティングできますが、最も エラーが発生しやすく、保守も困難になります。ルートが既存のノードと割り当てられたポッド CIDR で、常に最新の状態であることを確認しなければなりません。ノードの数が少なく、インフラストラクチャが静的である場合、これは実行可能なオプションであり、ルーターでの BGP 対応も不要になります。この方法を取る場合は、アドレスを IPAM によって決定せず、各ノードに割り当てるポッド CIDR スライスを使用して CNI を設定することをお勧めします。


On-prem environment
Remote Node CIDR - 10.80.0.0/16
Remote Pod CIDR 10.85.0.0/16
Hybrid Node 10.80.0.2
Pod CIDR 10.85.1.0/24
10.85.1.10
10.85.1.11
pod
pod
CNI
Hybrid Node 10.80.0.3
Pod CIDR 10.85.2.0/24
10.85.2.10
pod
CNI
Router
172.16.0.0
172.16.1.0
172.16.2.0
Routing table
- 10.80.0.0/16 [local] via eth0 (CONNECTED)
- 10.0.0.0/16 [static] via Gateway (STATIC)
- 10.85.1.0/24 [static] via 10.80.0.2 (STATIC)
- 10.85.2.0/24 [static] via 10.80.0.3 (STATIC)
On-prem to AWS link
VPC CIDR 10.0.0.0/16
Gateway

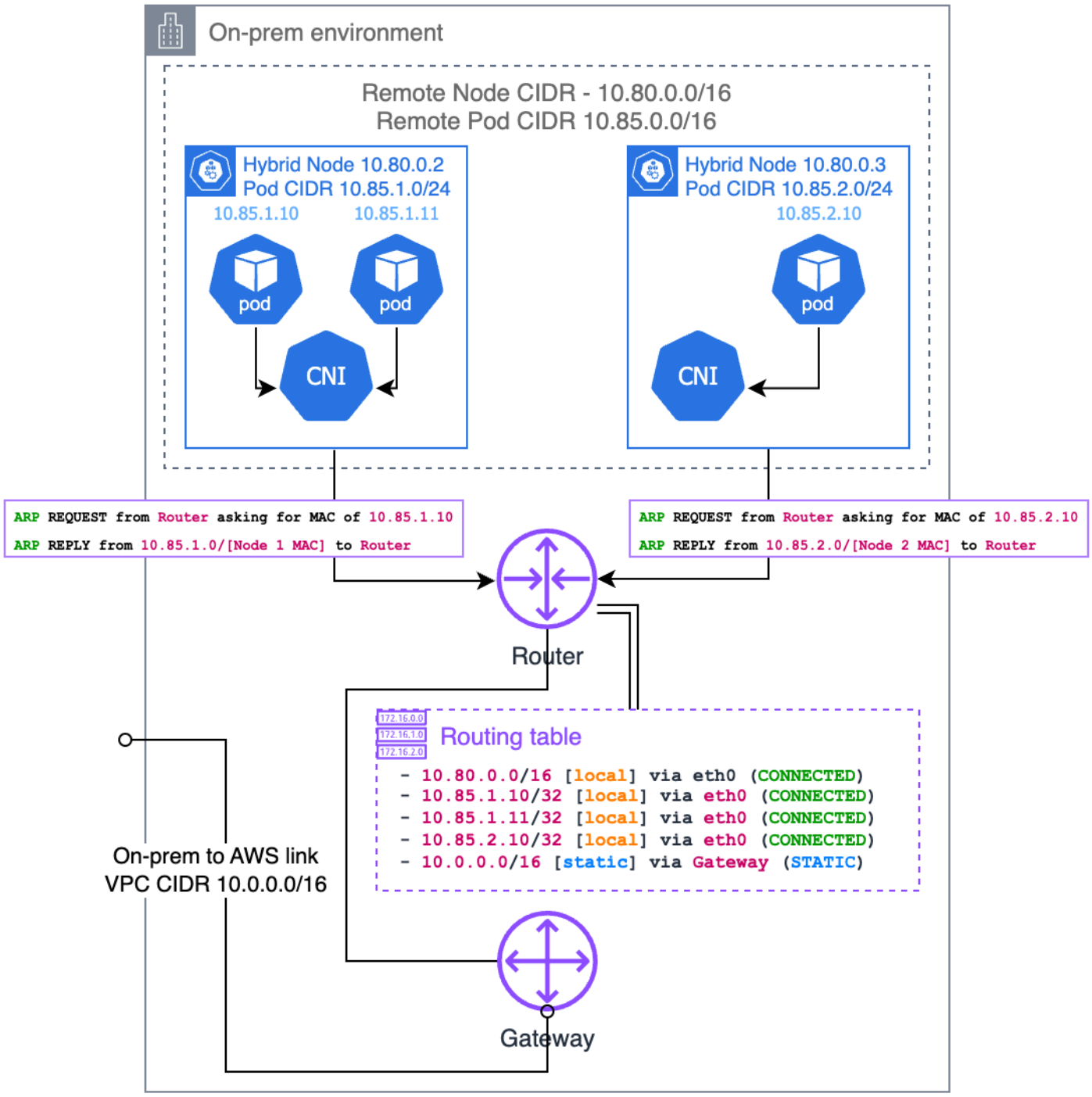
アドレス解決プロトコル (ARP) プロキシ

ARP プロキシは、オンプレミスのポッド IP をルーティング可能にするもう 1 つのアプローチであり、特に、ハイブリッドノードがローカルルーターと同じレイヤー 2 ネットワークにある場合に便利です。ARP プロキシを有効にすると、ノードは、その IP が別のサブネットに属している場合でも、ホストするポッド IP の ARP リクエストに応答します。

ローカルネットワーク上のデバイスがポッド IP に到達を試みる場合、最初に「この IP を持っているなら応答せよ」という ARP リクエストを送信します。該当するポッドをホストしているハイブリッドノードは、自分の MAC アドレスで応答し、「その IP のトラフィックを処理できます」と伝えます。これによって、ルーター設定を必要とせずに、ローカルネットワーク上のデバイスとポッドの間に直接のパスが作成されます。

これを機能させるには、CNI がプロキシ ARP 機能に対応している必要があります。Cilium は、設定によってプロキシ ARP を有効化できるように設計されています。ここで特に考慮すべき点は、ポッド CIDR が環境内の他のネットワークと重複しないようにすることです。重複すると、ルーティングの競合が生じる可能性があるからです。

このアプローチには、次のような利点があります。* BGP でルーターを設定する必要はなく、静的ルートを維持する必要もない。* ルーター設定を制御できない環境でも効果的に機能する。



ポッド間でのカプセル化

オンプレミス環境の CNI では、通常、カプセル化プロトコルによって、物理ネットワーク上で動作するオーバーレイネットワークが作成されます。この場合、再設定が不要になります。このセクションでは、このカプセル化の仕組みについて説明します。使用している CNI によっては、一部の詳細情報が異なる場合があります。

カプセル化では、元のポッドネットワークパケットを、基盤の物理ネットワークを介してルーティング可能な別のネットワークパケット内にラップします。これによって、ポッドの CIDR のルーティング方法が物理ネットワークで認識されていなくても、同じ CNI を実行しているノード間でポッドが通信できます。

Kubernetes で使用される最も一般的なカプセル化プロトコルは Virtual Extensible LAN (VXLAN) ですが、CNI に応じて、他のプロトコル (Geneve など) も利用できます。

VXLAN カプセル化

VXLAN では、UDP パケット内にレイヤー 2 イーサネットフレームをカプセル化します。ポッドが別のノードにある別のポッドにトラフィックを送信すると、CNI では、以下が実行されます。

1. ポッド A からのパケットをインターセプトする
2. 元のパケットを VXLAN ヘッダーにラップする
3. ラップしたパケットを、ノードの通常のネットワークスタックを介して宛先ノードに送信する
4. 宛先ノードの CNI が、パケットをラップ解除し、ポッド B に配信する

VXLAN カプセル化中のパケット構造の変化を次に示します。

元のポッド間パケット

```
+-----------------+---------------+-------------+-----------------+
| Ethernet Header | IP Header     | TCP/UDP     | Payload         |
| Src: Pod A MAC  | Src: Pod A IP | Src Port    |                 |
| Dst: Pod B MAC  | Dst: Pod B IP | Dst Port    |                 |
+-----------------+---------------+-------------+-----------------+
```

VXLAN カプセル化後

```
+-----------------+-------------+--------------+------------
+---------------------------+
| Outer Ethernet  | Outer IP    | Outer UDP    | VXLAN      | Original Pod-to-Pod
 |
| Src: Node A MAC | Src: Node A | Src: Random  | VNI: xx    | Packet (unchanged
 |
| Dst: Node B MAC | Dst: Node B | Dst: 4789    |            | from above)
 |
+-----------------+-------------+--------------+------------
+---------------------------+
```

この VXLAN ネットワーク識別子 (VNI) によって、異なるオーバーレイネットワークが区別されます。

ポッド通信のシナリオ

同じハイブリッドノード上のポッド

同じハイブリッドノード上のポッドが通信する場合、通常はカプセル化は必要ありません。CNI では、ローカルルートが設定されます。これにより、ノードの内部仮想インターフェイスを介してポッド間のトラフィックを転送します。

```
Pod A -> veth0 -> node's bridge/routing table -> veth1 -> Pod B
```

パケットはノードから出ることはなく、カプセル化を必要としません。

異なるハイブリッドノード上のポッド

異なるハイブリッドノード上のポッド間で通信するには、カプセル化が必要です。

```
Pod A -> CNI -> [VXLAN encapsulation] -> Node A network -> router or gateway -> Node B
 network -> [VXLAN decapsulation] -> CNI -> Pod B
```

これにより、物理ネットワークでポッド IP のルーティングが認識されていなくても、ポッドトラフィックは、物理ネットワークインフラストラクチャを通過できます。

ハイブリッドノード向けネットワークトラフィックフロー

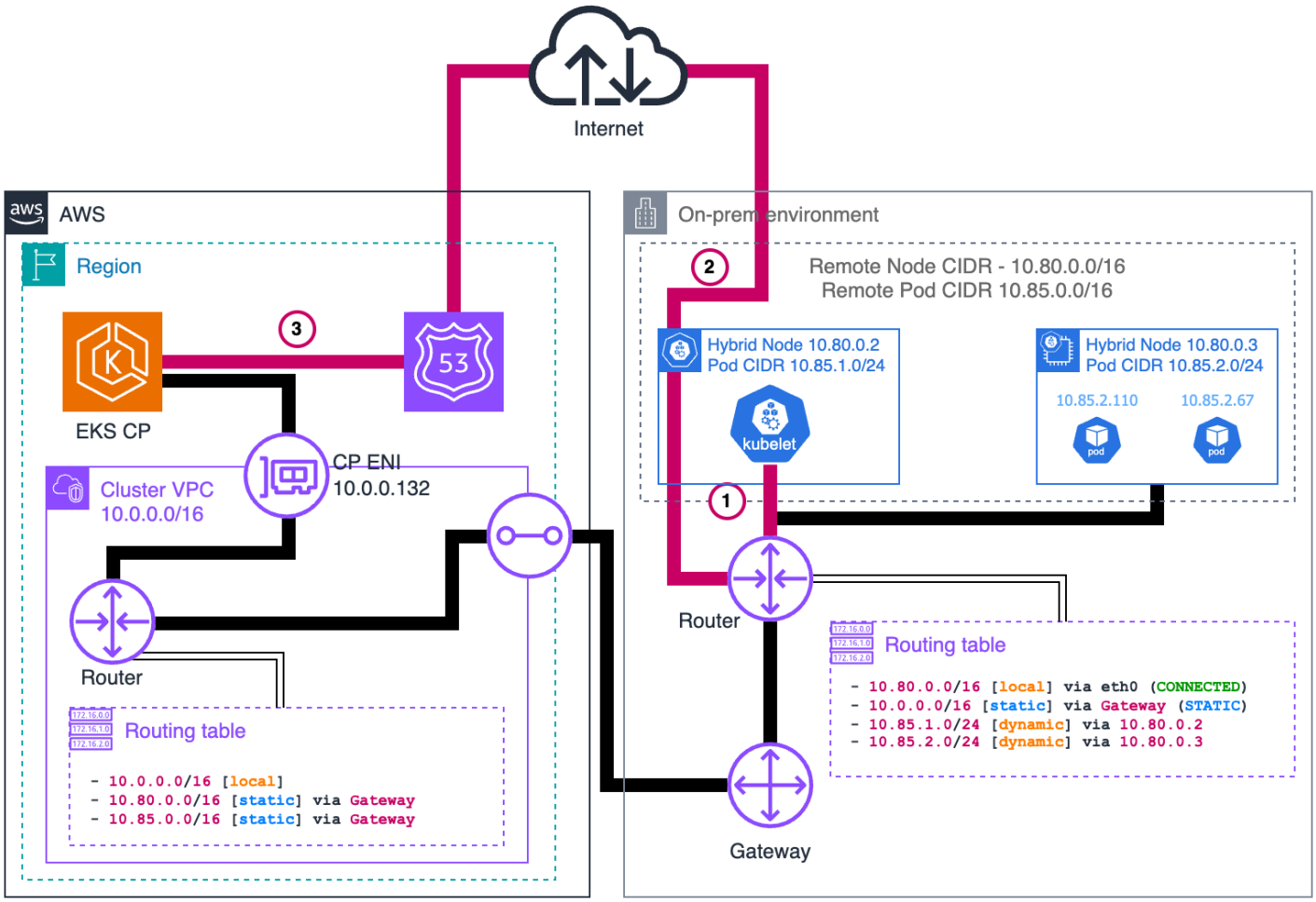
このページでは、EKS ハイブリッドノード向けネットワークトラフィックフローについて、いくつかのトラフィックタイプ別にエンドツーエンドでネットワークパスの図を示しながら詳しく説明します。

以下のトラフィックフローを取り上げます。

- the section called “ハイブリッドノード kubelet から EKS コントロールプレーンまでのフロー”
- the section called “EKS コントロールプレーンからハイブリッドノード (kubelet サーバー) までのフロー”
- the section called “ハイブリッドノードで実行されているポッドから EKS コントロールプレーンまでのフロー”

- the section called “EKS コントロールプレーンからハイブリッドノードで実行されているポッドまでのフロー (ウェブフック)”
- the section called “ハイブリッドノードで実行されているポッド間”
- the section called “クラウドノード上のポッドからハイブリッドノード上のポッドまでのフロー (東西トラフィック)”

ハイブリッドノード **kubelet** から EKS コントロールプレーンまでのフロー



[リクエスト]

1**kubelet**. がリクエストを開始する

ハイブリッドノード上の kubelet が EKS コントロールプレーンと通信する必要がある場合 (例えば、ノードステータスを報告するため、あるいはポッドの仕様を取得するため) には、ノード登録時に提供された kubeconfig ファイルが使用されます。この kubeconfig には、ダイレクト IP アドレスではなく、API サーバーエンドポイント URL (Route53 DNS 名) が記載されています。

kubelet がエンドポイントに対して DNS ルックアップを実行します (例: https://xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx.gr7.us-west-2.eks.amazonaws.com)。パブリックアクセスクラスターでは、これは AWS で実行中の EKS サービスに属するパブリック IP アドレス (例: 54.239.118.52) に解決されます。次に kubelet は、このエンドポイントへの安全な HTTPS リクエストを作成します。初期パケットは以下のようになります。

```
+-------------------+--------------------+-----------------+
| IP Header         | TCP Header         | Payload         |
| Src: 10.80.0.2    | Src: 52390 (random) |                 |
| Dst: 54.239.118.52 | Dst: 443           |                 |
+-------------------+--------------------+-----------------+
```

2. ローカルルーターのルーティング

宛先 IP はパブリック IP アドレスであり、ローカルネットワークの一部ではないため、kubelet はこのパケットをそのデフォルトゲートウェイ (ローカルオンプレミスルーター) に送信します。ルーターは、宛先 IP を確認して、パブリック IP アドレスであると判断します。

パブリックトラフィックの場合、ルーターは通常、パケットをインターネットゲートウェイ、つまりインターネットへのアウトバウンドトラフィックを処理するボーダールーターに転送します。図では、これを省略しています。実際の転送は、オンプレミスネットワークがどのように設定されているかによって異なります。パケットは、オンプレミスのネットワークインフラストラクチャを通っていき、最終的にインターネットサービスプロバイダーのネットワークに到達します。

3. EKS コントロールプレーンまでの配信

パケットは、パブリックインターネットとトランジットネットワークを横断していき、AWS のネットワークに到達します。AWS のネットワークは、パケットを適切なリージョンの EKS サービスエンドポイントにルーティングします。EKS サービスに到達したパケットは、クラスターの実際の EKS コントロールプレーンに転送されます。

これはパブリックインターネットを通るルーティングであり、他のトラフィックフローで見られるようなプライベート VPC をルーティングされていくパスとは異なります。その主な違いは、パブリックアクセスモードを使用した場合、ポッドからではなくオンプレミスの kubelet から EKS コントロールプレーンへのトラフィックが VPC を経由せず、代わりにグローバルインターネットインフラストラクチャを使用することです。

[応答]

EKS コントロールプレーンは、kubelet リクエストを処理してレスポンスを返します。

3. EKS コントロールプレーンがレスポンスを送信する

EKS コントロールプレーンはレスポンスパケットを作成します。このパケットには、ソースとしてパブリック IP、送信先としてハイブリッドノードの IP が含まれています。

```
+--------------------+---------------------+-----------------+
| IP Header          | TCP Header          | Payload         |
| Src: 54.239.118.52 | Src: 443            |                 |
| Dst: 10.80.0.2     | Dst: 52390          |                 |
+--------------------+---------------------+-----------------+
```

2. インターネットでのルーティング

レスポンスパケットは、インターネットサービスプロバイダーが決定したルーティングパスをたどってインターネット経由で戻り、オンプレミスネットワークのエッジルーターに到達します。

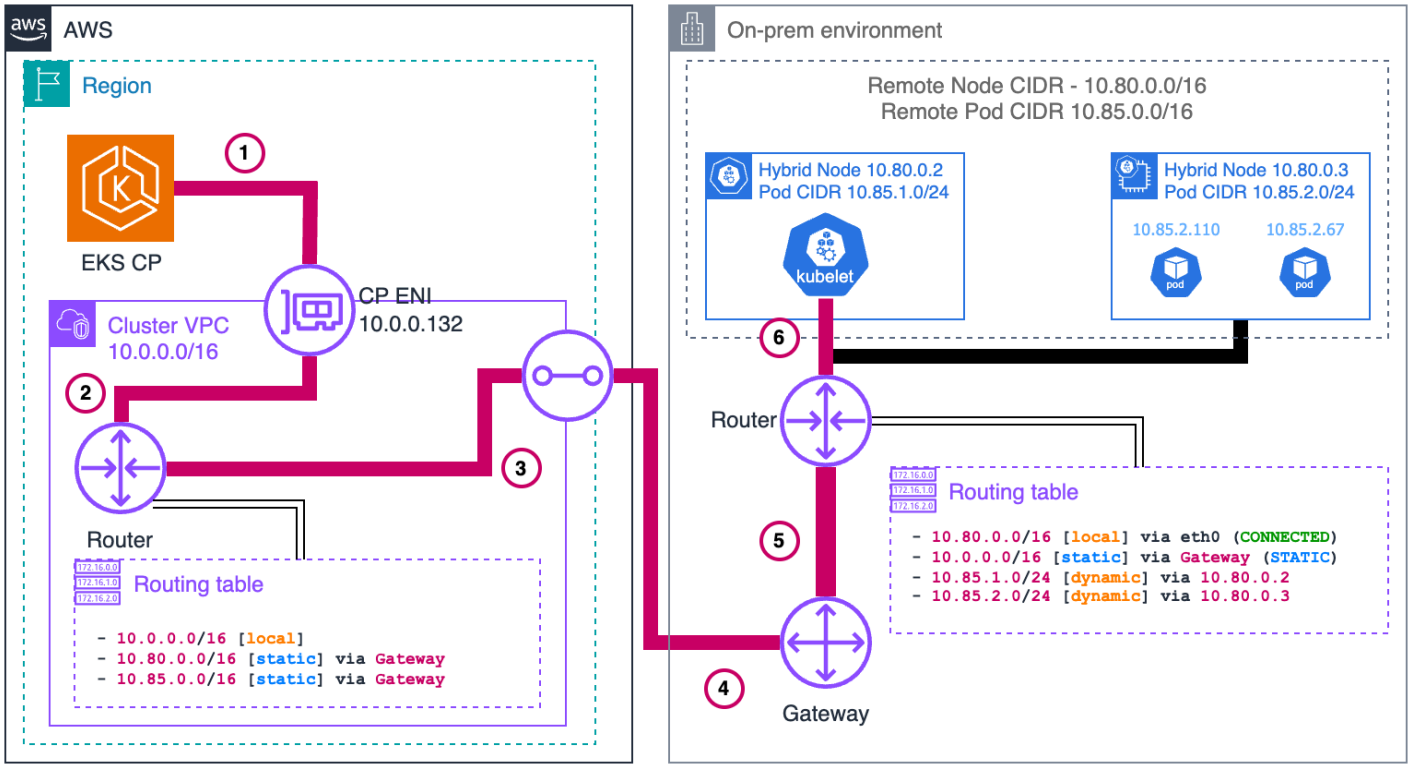
1. ローカルでの配信

オンプレミスのルーターは、パケットを受信し、宛先 IP (`10.80.0.2`) がローカルネットワークに属していると認識します。次に、受信したパケットをローカルネットワークインフラストラクチャ経由で転送します。パケットがターゲットのハイブリッドノードに到達すると、`kubelet` がレスポンスを受信して処理します。

ハイブリッドノード **`kube-proxy`** から EKS コントロールプレーンまでのフロー

クラスターのパブリックエンドポイントアクセスを有効にすると、戻りのトラフィックはパブリックインターネットを使用します。このトラフィックは、ハイブリッドノードの `kube-proxy` から EKS コントロールプレーンへと進み、`kubelet` から EKS コントロールプレーンへのトラフィックと同じパスを通ります。

EKS コントロールプレーンからハイブリッドノード (**kubelet** サーバー) までのフロー



[リクエスト]

1. EKS Kubernetes API サーバーがリクエストを開始する

EKS Kubernetes API サーバーは、ノードオブジェクトのステータスからノードの IP アドレス (`10.80.0.2`) を取得します。次に、送信先 IP が設定済みのリモートノード CIDR (`10.80.0.0/16`) に属しているため、このリクエストを VPC 内の ENI 経由でルーティングします。初期パケットは以下のようになります。

```
+-----------------+---------------------+-----------------+
| IP Header       | TCP Header          | Payload         |
| Src: 10.0.0.132 | Src: 67493 (random) |                 |
| Dst: 10.80.0.2  | Dst: 10250          |                 |
+-----------------+---------------------+-----------------+
```

2. VPC ネットワークでの処理

パケットは、ENI を出て VPC ネットワーキングレイヤーに入り、さらにルーティングするためにサブネットのゲートウェイ宛てに送信されます。

3. VPC ルートテーブルのルックアップ

EKS コントロールプレーン ENI が含まれているサブネット用の VPC ルートテーブルには、リモートノード CIDR に固有のルート (図の 2 番目のルート) が登録されています。このルーティングルールに基づいて、パケットが VPC-to-onprem ゲートウェイに向けて送信されます。

4. 境界を越えて移動する

ゲートウェイは、確立済みの接続 (Direct Connect や VPN など) 経由でクラウド境界を越えてオンプレミスネットワークにパケットを転送します。

5. オンプレミスネットワークでの受信

パケットは、ハイブリッドノードが配置されているサブネット宛てのトラフィックを処理するローカルオンプレミスルーターに到達します。

6. 最後の配信

ローカルルーターは、宛先 IP (`10.80.0.2`) アドレスが自身に直接接続されているネットワークに属しているかどうかを識別して、ターゲットのハイブリッドノードに直接パケットを転送し、そこで `kubelet` がリクエストを受信して処理します。

[応答]

ハイブリッドノードの `kubelet` は、リクエストを処理してレスポンスを返します。レスポンスは、同じパスを逆方向にたどります。

```
+-----------------+---------------------+-----------------+
| IP Header       | TCP Header          | Payload         |
| Src: 10.80.0.2  | Src: 10250          |                 |
| Dst: 10.0.0.132 | Dst: 67493          |                 |
+-----------------+---------------------+-----------------+
```

6. **`kubelet`** レスポンスを送信する

ハイブリッドノード (`10.80.0.2`) 上の `kubelet` は、送信元の IP を宛先としてレスポンスパケットを作成します。その宛先はローカルネットワークに属していないため、パケットはホストのデフォルトゲートウェイ、つまりローカルルーターに送信されます。

5. ローカルルーターのルーティング

ルーターは、宛先 IP (`10.0.0.132`) が `10.0.0.0/16` に属していると判断します。ここには、AWS に接続しているゲートウェイを指すルートがあります。

4. 境界を越えて戻る

パケットは、同じオンプレミスを通って VPC 接続 (Direct Connect や VPN など) まで戻り、クラウド境界を逆方向に横断します。

3. VPC でのルーティング

パケットが VPC に到達すると、ルートテーブルは宛先 IP が VPC CIDR に属しているかどうかを識別します。パケットは VPC 内にルーティングされます。

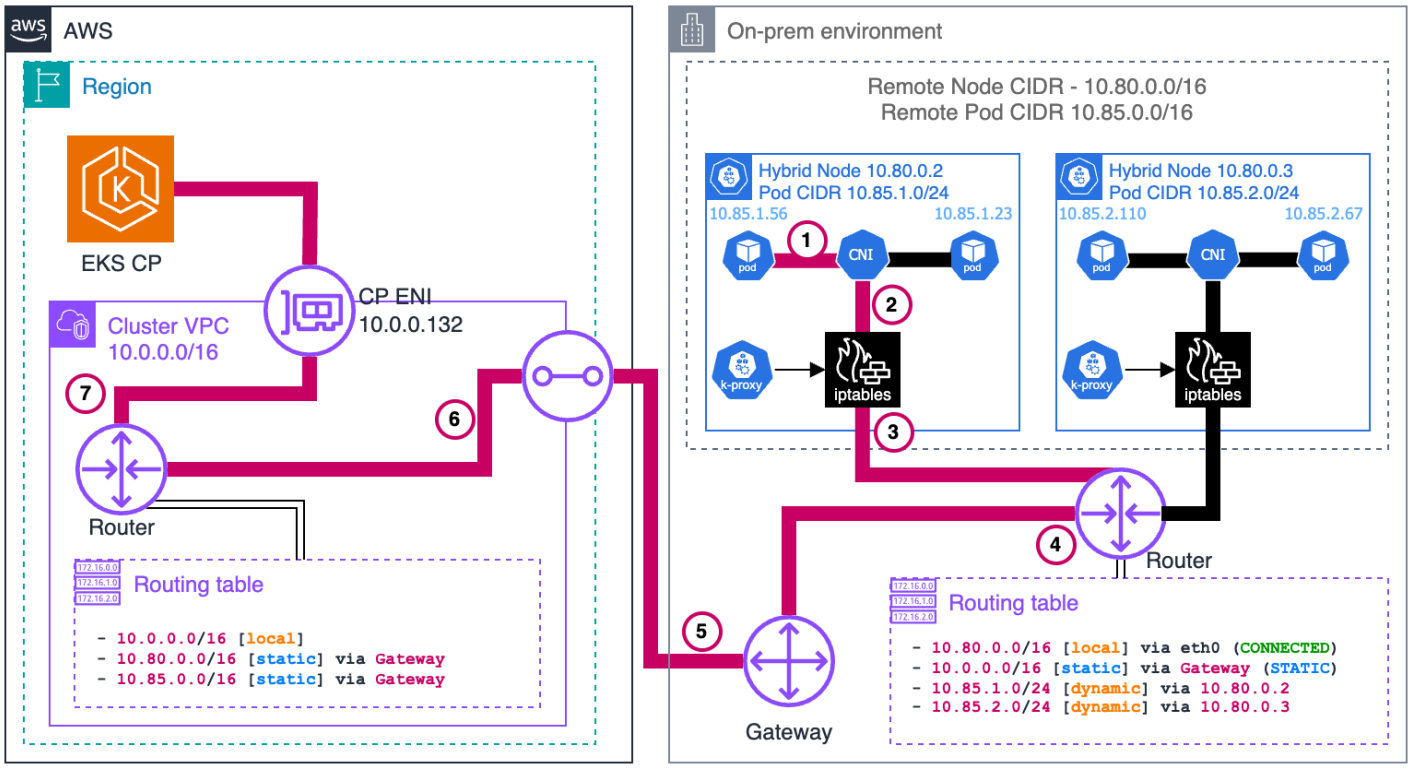
2. VPC ネットワークでの配信

VPC ネットワーキングレイヤーは、EKS コントロールプレーン ENI (`10.0.0.132`) を備えたサブネットにパケットを転送します。

1. ENI での受信

パケットは、Kubernetes API サーバーにアタッチされている EKS コントロールプレーン ENI に到達します。これでラウンドトリップは完了です。

ハイブリッドノードで実行されているポッドから EKS コントロールプレーンまでのフロー



CNI NAT なし

[リクエスト]

ポッドは一般に、kubernetes サービスを介して Kubernetes API サーバーと対話します。サービス IP がクラスターのサービス CIDR の最初の IP となります。この規約により、CoreDNS が使用可能になる前に稼働する必要があるポッドが API サーバー (例えば、CNI) に到達できるようになります。リクエストは、サービス IP を宛先としてポッドを出ていきます。例えば、サービス CIDR が 172.16.0.0/16 である場合、サービス IP は 172.16.0.1 になります。

1. ポッドがリクエストを開始する

ポッドは、ランダムな送信元ポートから API サーバーポート (443) 上の kubernetes サービス IP (172.16.0.1) にリクエストを送信します。パケットは以下のようになります。

```
+-----------------+---------------------+-----------------+
| IP Header       | TCP Header          | Payload         |
| Src: 10.85.1.56 | Src: 67493 (random) |                 |
| Dst: 172.16.0.1 | Dst: 443            |                 |
+-----------------+---------------------+-----------------+
```

2. CNI での処理

CNI は、宛先 IP が自身の管理するポッド CIDR に属していないことを検出します。送信 NAT が無効になっているため、CNI はパケットを変更せずにホストネットワークスタックに渡します。

3. ノードネットワークでの処理

パケットはノードのネットワークスタックに入り、そこで netfilter フックが kube-proxy によって設定された iptables ルールをトリガーします。以下の順序でルールがいくつか適用されます。

1. パケットはまず KUBE-SERVICES チェーンをヒットします。ここには、各サービスの ClusterIP とポートに対応するルールが含まれています。
2. その対応するルールは、kubernetes サービスの KUBE-SVC-XXX チェーンにジャンプします (172.16.0.1:443 宛てのパケット)。ここには、ロードバランシングルールが含まれています。
3. ロードバランシングルールは、コントロールプレーン ENI IP (10.0.0.132 または 10.0.1.23) 用に KUBE-SEP-XXX チェーンのいずれかをランダムに選択します。
4. 選択した KUBE-SEP-XXX チェーンには実際のルールがあり、それにより宛先 IP はサービス IP から選択されている IP に変更されます。これは、Destination Network Address Translation (宛先ネットワークアドレス変換、DNAT) と呼ばれます。

選択されている EKS コントロールプレーン ENI の IP を 10.0.0.132 とした場合、これらのルールが適用されると、パケットは以下のようになります。

```
+-----------------+---------------------+-----------------+
| IP Header       | TCP Header          | Payload         |
| Src: 10.85.1.56 | Src: 67493 (random) |                 |
| Dst: 10.0.0.132 | Dst: 443            |                 |
+-----------------+---------------------+-----------------+
```

宛先 IP がローカルネットワーク内にないため、ノードはパケットをデフォルトゲートウェイに転送します。

4. ローカルルーターのルーティング

ローカルルーターは、宛先 IP (10.0.0.132) が VPC CIDR (10.0.0.0/16) に属していると判断して、AWS に接続しているゲートウェイに転送します。

5. 境界を越えて移動する

パケットは、確立済みの接続 (Direct Connect や VPN など) 経由でクラウド境界を越えて VPC に移動します。

6. VPC ネットワークでの配信

VPC ネットワーキングレイヤーは、EKS コントロールプレーン ENI (10.0.0.132) が存在する適切なサブネットにパケットをルーティングします。

7. ENI での受信

パケットは、Kubernetes API サーバーにアタッチされている EKS コントロールプレーン ENI に到達します。

[応答]

EKS コントロールプレーンは、リクエストを処理してポッドにレスポンスを返します。

7. API サーバーがレスポンスを送信する

EKS Kubernetes API サーバーは、送信元の IP を宛先としてレスポンスパケットを作成します。パケットは以下のようになります。

```
+-----------------+---------------------+-----------------+
| IP Header       | TCP Header          | Payload         |
```

```
| Src: 10.0.0.132 | Src: 443            |                 |
| Dst: 10.85.1.56 | Dst: 67493          |                 |
+-----------------+---------------------+-----------------+
```

宛先 IP が設定済みのリモートポッド CIDR (10.85.0.0/16) に属しているため、パケットはサブネットのルーターを次のホップとして VPC 内の ENI 経由で送信されます。

6. VPC でのルーティング

VPC ルートテーブルにはリモートポッド CIDR (10.85.0.0/16) のエントリがあるため、このトラフィックは VPC-to-onprem ゲートウェイ宛てに送信されます。

5. 境界を越えて移動する

ゲートウェイは、確立済みの接続 (Direct Connect や VPN など) 経由でクラウド境界を越えてオンプレミスネットワークにパケットを転送します。

4. オンプレミスネットワークでの受信

パケットは、ローカルオンプレミスルーターに到達します。

3. ノードへの配信

ルーターのテーブルには 10.80.0.2 を次のホップとした 10.85.1.0/24 のエントリがあるため、パケットはノードに配信されます。

2. ノードネットワークでの処理

パケットがノードのネットワークスタックによって処理されると、conntrack (netfilter の一部) はポッドが最初に確立した接続とパケットを照合します。DNAT が最初に適用されたため、conntrack はソース IP を EKS コントロールプレーン ENI の IP から kubernetes サービス IP に書き換えることで DNAT を元に戻します。

```
+-----------------+---------------------+-----------------+
| IP Header       | TCP Header          | Payload         |
| Src: 172.16.0.1 | Src: 443            |                 |
| Dst: 10.85.1.56 | Dst: 67493          |                 |
+-----------------+---------------------+-----------------+
```

1. CNI での処理

CNI は、宛先 IP が自身のネットワーク内のポッドに属しているかどうかを識別して、適切なポッドネットワーク名前空間にパケットを配信します。

このフローは、なぜリモートポッド CIDR を VPC から各ポッドをホストする特定のノードに至るまで正しくルーティングできなければならないかを示しています。戻りパス全体は、クラウドとオンプレミスの両方のネットワーク全体にわたってポッド IP が適切にルーティングされるかどうかによって異なります。

CNI NAT あり

このフローは、CNI NAT がないものと非常によく似ていますが、重要な違いが 1 つあります。CNI は、パケットに送信元 NAT (SNAT) を適用してから、ノードのネットワークスタックにパケットを送信します。これにより、パケットの送信元 IP がノードの IP に変更されるため、追加でルーティングを設定することなく、ノードにパケットをルーティングできます。

[リクエスト]

1. ポッドがリクエストを開始する

ポッドは、ランダムな送信元ポートから EKS Kubernetes API サーバーポート (443) 上の `kubernetes` サービス IP (`172.16.0.1`) にリクエストを送信します。パケットは以下のようになります。

```
+-----------------+---------------------+-----------------+
| IP Header       | TCP Header          | Payload         |
| Src: 10.85.1.56 | Src: 67493 (random) |                 |
| Dst: 172.16.0.1 | Dst: 443            |                 |
+-----------------+---------------------+-----------------+
```

2. CNI での処理

CNI は、宛先 IP が自身の管理するポッド CIDR に属していないことを検出します。送信 NAT が有効になっているため、CNI は SNAT をパケットに適用して送信元 IP をノードの IP に変更してから、パケットをノードのネットワークスタックに渡します。

```
+-----------------+---------------------+-----------------+
| IP Header       | TCP Header          | Payload         |
| Src: 10.80.0.2  | Src: 67493 (random) |                 |
| Dst: 172.16.0.1 | Dst: 443            |                 |
+-----------------+---------------------+-----------------+
```

注: この例ではわかりやすくするために CNI と `iptables` をそれぞれ別のブロックに分けていますが、実際には `iptables` を使用して NAT を適用する CNI もあります。

3. ノードネットワークでの処理

ここでは、kube-proxy によって設定された iptables ルールが前の例と同じように動作して、パケットを EKS コントロールプレーン ENI のいずれかに負荷分散します。これでパケットは以下のようになります。

```
+-----------------+---------------------+-----------------+
| IP Header       | TCP Header          | Payload         |
| Src: 10.80.0.2  | Src: 67493 (random) |                 |
| Dst: 10.0.0.132 | Dst: 443            |                 |
+-----------------+---------------------+-----------------+
```

宛先 IP がローカルネットワーク内にないため、ノードはパケットをデフォルトゲートウェイに転送します。

4. ローカルルーターのルーティング

ローカルルーターは、宛先 IP (10.0.0.132) が VPC CIDR (10.0.0.0/16) に属していると判断して、AWS に接続しているゲートウェイに転送します。

5. 境界を越えて移動する

パケットは、確立済みの接続 (Direct Connect や VPN など) 経由でクラウド境界を越えて VPC に移動します。

6. VPC ネットワークでの配信

VPC ネットワーキングレイヤーは、EKS コントロールプレーン ENI (10.0.0.132) が存在する適切なサブネットにパケットをルーティングします。

7. ENI での受信

パケットは、Kubernetes API サーバーにアタッチされている EKS コントロールプレーン ENI に到達します。

[応答]

EKS コントロールプレーンは、リクエストを処理してポッドにレスポンスを返します。

7. API サーバーがレスポンスを送信する

EKS Kubernetes API サーバーは、送信元の IP を宛先としてレスポンスパケットを作成します。パケットは以下のようになります。

```
+-----------------+---------------------+-----------------+
| IP Header       | TCP Header          | Payload         |
| Src: 10.0.0.132 | Src: 443            |                 |
| Dst: 10.80.0.2  | Dst: 67493          |                 |
+-----------------+---------------------+-----------------+
```

宛先 IP が設定済みのリモートノード CIDR (10.80.0.0/16) に属しているため、パケットはサブネットのルーターを次のホップとして VPC 内の ENI 経由で送信されます。

6. VPC でのルーティング

VPC ルートテーブルにはリモートノード CIDR (10.80.0.0/16) のエントリがあるため、このトラフィックは VPC-to-onprem ゲートウェイ宛てに送信されます。

5. 境界を越えて移動する

ゲートウェイは、確立済みの接続 (Direct Connect や VPN など) 経由でクラウド境界を越えてオンプレミスネットワークにパケットを転送します。

4. オンプレミスネットワークでの受信

パケットは、ローカルオンプレミスルーターに到達します。

3. ノードへの配信

ローカルルーターは、宛先 IP (10.80.0.2) アドレスが自身に直接接続されているネットワークに属しているかどうかを識別して、ターゲットのハイブリッドノードに直接パケットを転送します。

2. ノードネットワークでの処理

パケットがノードのネットワークスタックによって処理されると、conntrack (netfilter の一部) はパケットをポッドが最初に確立した接続と照合します。元々 DNAT が適用されていたため、送信元 IP を EKS コントロールプレーン ENI の IP から kubernetes サービス IP に書き換えることで元に戻します。

```
+-----------------+---------------------+-----------------+
| IP Header       | TCP Header          | Payload         |
| Src: 172.16.0.1 | Src: 443            |                 |
| Dst: 10.80.0.2  | Dst: 67493          |                 |
+-----------------+---------------------+-----------------+
```

1. CNI での処理

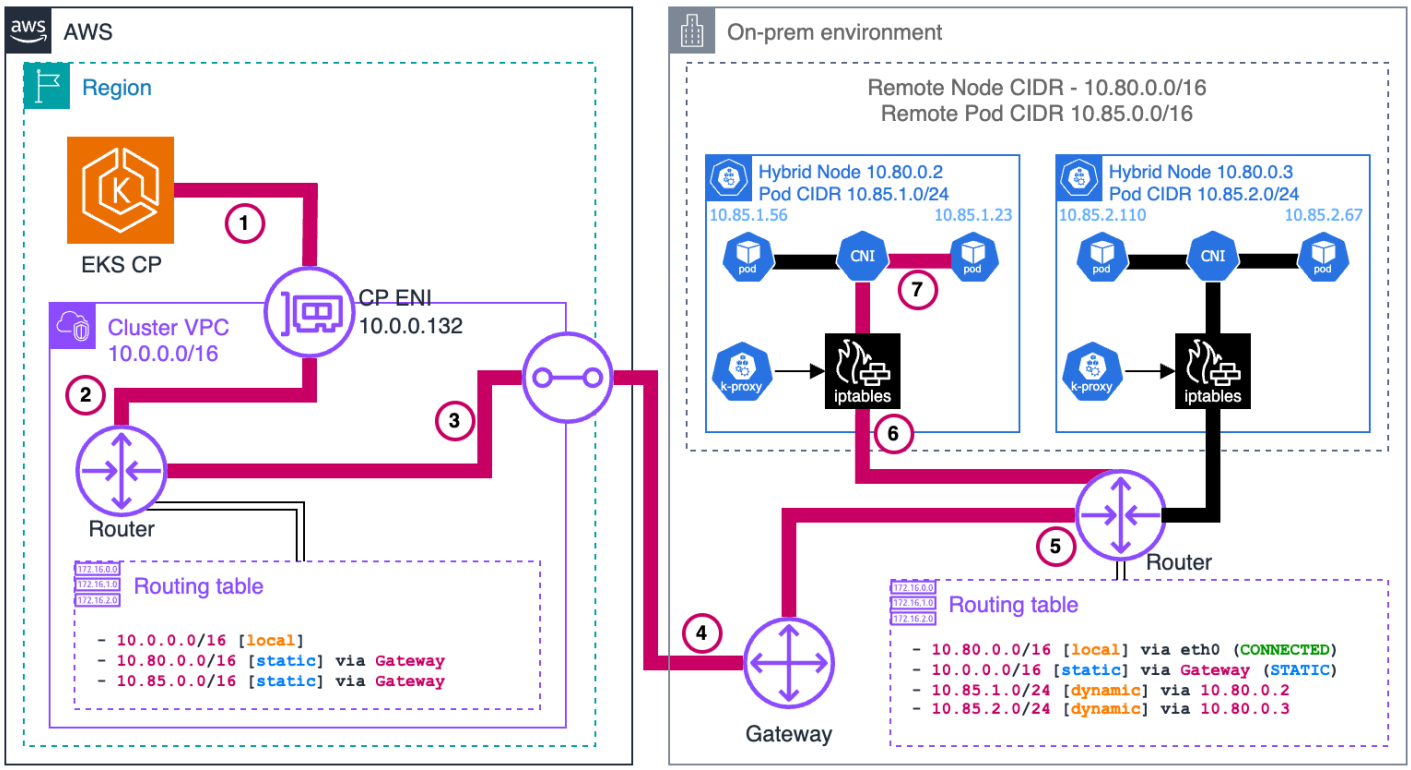
CNI は、このパケットが以前に SNAT が適用された接続に属しているかどうかを識別します。これにより、SNAT が逆方向に適用されるため、宛先 IP がポッドの IP に戻ります。

```
+-----------------+---------------------+-----------------+
| IP Header       | TCP Header          | Payload         |
| Src: 172.16.0.1 | Src: 443            |                 |
| Dst: 10.85.1.56 | Dst: 67493          |                 |
+-----------------+---------------------+-----------------+
```

CNI は、宛先 IP が自身のネットワーク内のポッドに属していることを検出して、適切なポッドネットワーク名前空間にパケットを配信します。

このフローは、CNI NAT の適用により、追加でポッド CIDR のルーティングを行うことなくパケットをノードにルーティングできるようにすることで、設定をどのように簡素化できるかを示しています。

EKS コントロールプレーンからハイブリッドノードで実行されているポッドまでのフロー (ウェブフック)



このトラフィックパターンはウェブフックで最もよく見られるものであり、ウェブフックではハイブリッドノード上のポッドで実行されているウェブフックサーバーへの接続を EKS コントロールプ

レーンが直接開始する必要があります。例えば、アドミッションウェブフックを検証して変更するといった場合、このウェブフックはリソースの検証と変更のプロセス中に API サーバーによって呼び出されます。

[リクエスト]

1. EKS Kubernetes API サーバーがリクエストを開始する

クラスターに設定されたウェブフックが関連する API オペレーションによってトリガーされた場合、EKS Kubernetes API サーバーはそのウェブフックサーバーポッドに直接接続する必要があります。API サーバーはまず、そのウェブフックに関連付けられているサービスリソースまたはエンドポイントリソースでポッドの IP アドレスを検索します。

IP が 10.85.1.23 のハイブリッドノード上でウェブフックポッドが実行されている場合、EKS Kubernetes API サーバーはウェブフックエンドポイントへの HTTPS リクエストを作成します。10.85.1.23 という宛先 IP は設定済みのリモートポッド CIDR (10.85.0.0/16) に属しているため、初期パケットは VPC の EKS コントロールプレーン ENI を介して送信されます。パケットは以下のようになります。

```
+-----------------+---------------------+-----------------+
| IP Header       | TCP Header          | Payload         |
| Src: 10.0.0.132 | Src: 41892 (random) |                 |
| Dst: 10.85.1.23 | Dst: 8443           |                 |
+-----------------+---------------------+-----------------+
```

2. VPC ネットワークでの処理

パケットは、EKS コントロールプレーン ENI を出て、サブネットのルーターを次のホップとする VPC ネットワーキングレイヤーに入ります。

3. VPC ルートテーブルのルックアップ

EKS コントロールプレーン ENI が含まれているサブネット用の VPC ルートテーブルには、リモートポッド CIDR (10.85.0.0/16) に固有のルートが登録されています。このルーティングルールにより、パケットは VPC-to-onprem ゲートウェイ (例えば、Direct Connect や VPN 接続の場合は仮想プライベートゲートウェイ) 宛てに送信されます。

```
Destination     Target
10.0.0.0/16     local
10.85.0.0/16    vgw-id (VPC-to-onprem gateway)
```

4. 境界を越えて移動する

ゲートウェイは、確立済みの接続 (Direct Connect や VPN など) 経由でクラウド境界を越えてオンプレミスネットワークにパケットを転送します。パケットがこの接続を通過するとき、パケットの送信元 IP アドレスと宛先 IP アドレスは元のまま維持されます。

5. オンプレミスネットワークでの受信

パケットは、ローカルオンプレミスルーターに到達します。ルーターは、自身のルーティングテーブルに問い合わせて、10.85.1.23 アドレスに到達する方法を決定します。そのためにオンプレミスネットワークには、適切なハイブリッドノード宛てにトラフィックを送信するポッド CIDR へのルートが必要です。

この例の場合、ルーターのルートテーブルには IP が `10.80.0.2` のハイブリッドノード経由で `10.85.1.0/24` サブネットに到達できることを示すエントリが登録されています。

```
Destination     Next Hop
10.85.1.0/24    10.80.0.2
```

6. ノードへの配信

ルーターは、ルーティングテーブルのエントリに基づいて、パケットをハイブリッドノード (`10.80.0.2`) に転送します。ノードに到達したとき、パケットは EKS Kubernetes API サーバーから送信されたときと同じままで、宛先 IP は依然としてポッドの IP です。

7. CNI での処理

ノードのネットワークスタックは、パケットを受信し、宛先 IP がノード自体の IP でないことを確認して、さらに処理するために CNI に渡します。CNI は、宛先 IP がこのノードでローカルに実行されているポッドに属しているかどうかを識別し、適切な仮想インターフェイス経由で適切なポッドにパケットを転送します。

```
Original packet -> node routing -> CNI -> Pod's network namespace
```

ポッド内のウェブフックサーバーは、リクエストを受信して処理します。

[応答]

ウェブフックポッドは、リクエストを処理してレスポンスを返します。レスポンスは、同じパスを逆方向にたどります。

7. ポッドがレスポンスを送信する

ウェブフックポッドは、自身の IP を送信元とし、元のリクエスタ (EKS コントロールプレーン ENI) を宛先として、レスポンスパケットを作成します。

```
+-----------------+---------------------+-----------------+
| IP Header       | TCP Header          | Payload         |
| Src: 10.85.1.23 | Src: 8443           |                 |
| Dst: 10.0.0.132 | Dst: 41892          |                 |
+-----------------+---------------------+-----------------+
```

CNI は、このパケットが外部ネットワーク (ローカルポッドではない) に送信され、パケットが、元のソース IP を保持したままノードのネットワークスタックに渡されたことを確認します。

6. ノードネットワークでの処理

ノードは、宛先 IP (`10.0.0.132`) がローカルネットワーク内にないと判断して、自身のデフォルトゲートウェイ (ローカルルーター) にパケットを転送します。

5. ローカルルーターのルーティング

ローカルルーターは、自身のルーティングテーブルに問い合わせて、宛先 IP (`10.0.0.132`) が VPC CIDR (`10.0.0.0/16`) に属していると判断します。次に、AWS に接続しているゲートウェイにパケットを転送します。

4. 境界を越えて移動する

パケットは、同じオンプレミスを通って VPC 接続まで戻り、クラウド境界を逆方向に横断します。

3. VPC でのルーティング

パケットが VPC に到達すると、ルートテーブルは宛先 IP が VPC 内のサブネットに属しているかどうかを識別します。それに応じて、パケットは VPC 内にルーティングされます。

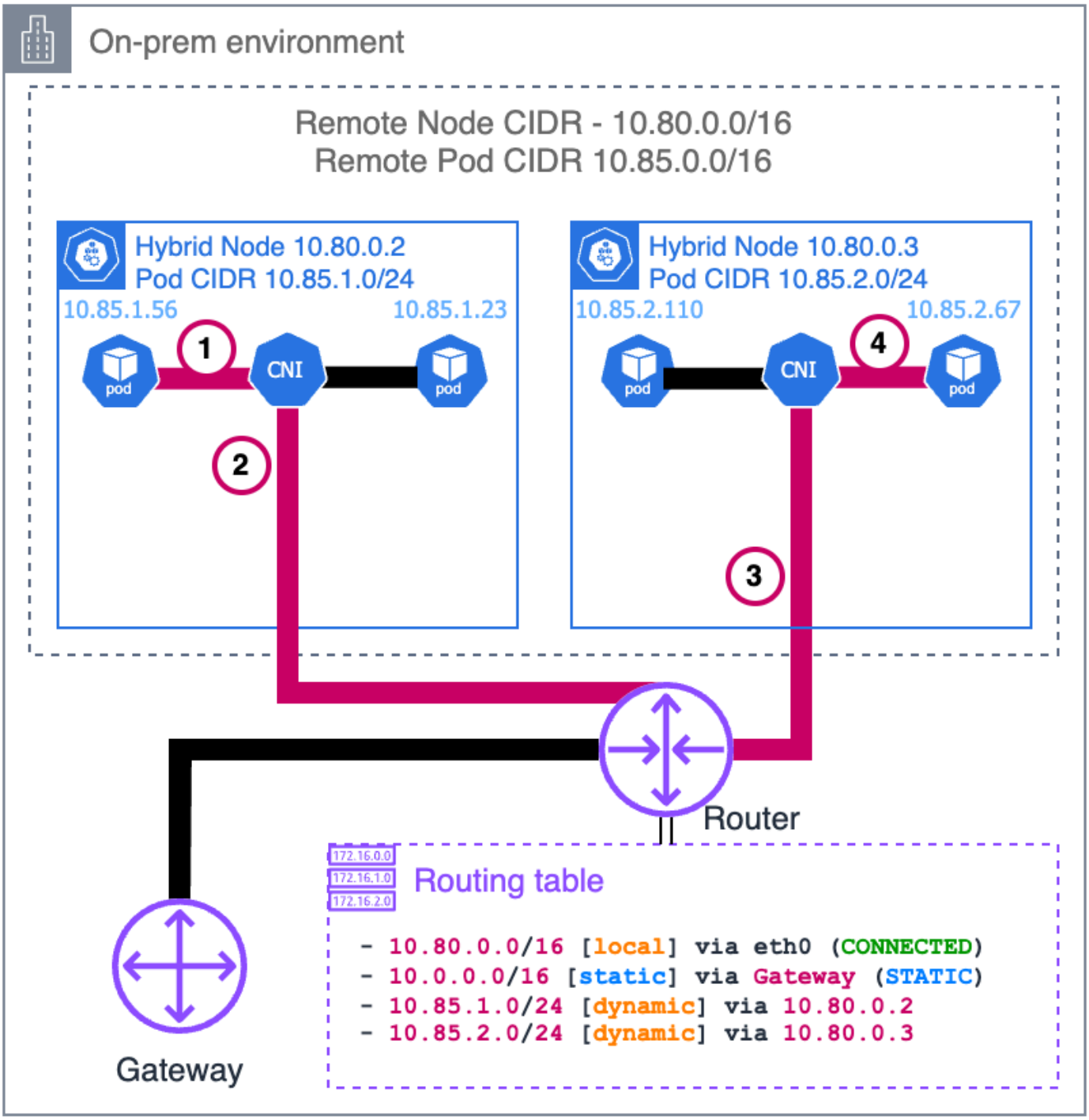
2. と 1. EKS コントロールプレーン ENI での受信

パケットは、Kubernetes API サーバーにアタッチされている ENI に到達します。これでラウンドトリップは完了です。API サーバーは、ウェブフックレスポンスを受信し、このレスポンスに基づいて元の API リクエストの処理を続行します。

このトラフィックフローは、なぜリモートポッド CIDR を正しく設定してルーティングする必要があるかを示しています。

- VPC には、オンプレミスゲートウェイを指すリモートポッド CIDR のルートが登録されている必要があります。
- オンプレミスネットワークには、そうしたポッドをホストしている特定のノード宛てにトラフィックを送信するポッド CIDR のルートが登録されている必要があります。
- このルーティング設定がないと、ハイブリッドノード上のポッドで実行されているウェブフックやその他の同じようなサービスに EKS コントロールプレーンから到達できなくなります。

ハイブリッドノードで実行されているポッド間



このセクションでは、異なるハイブリッドノード上で実行されているポッド同士がどのように通信しているかについて説明します。この例では、CNI がカプセル化に VXLAN を使用しているものとしま

す。これは、Cilium や Calico などの CNI でよく見られることです。プロセス全体は、Geneve や IP-in-IP など、他のカプセル化プロトコルに似ています。

[リクエスト]

1. ポッド A が通信を開始する

ノード 1 上のポッド A (10.85.1.56) が、ノード 2 上のポッド B (10.85.2.67) にトラフィックを送信しようとしています。初期パケットは以下のようになります。

```
+------------------+-----------------+-------------+-----------------+
| Ethernet Header  | IP Header       | TCP/UDP     | Payload         |
| Src: Pod A MAC   | Src: 10.85.1.56 | Src: 43721  |                 |
| Dst: Gateway MAC | Dst: 10.85.2.67 | Dst: 8080   |                 |
+------------------+-----------------+-------------+-----------------+
```

2. CNI がパケットをインターセプトして処理する

ポッド A のパケットが自身のネットワーク名前空間を出ると、CNI がそのパケットをインターセプトします。CNI は自身のルーティングテーブルに問い合わせて、宛先 IP (10.85.2.67) はポッド CIDR に属している、この IP はローカルノードではなくノード 2 (10.80.0.3) に属している、パケットは VXLAN でカプセル化する必要がある、と判断します。

カプセル化するという判断は重要です。基盤となる物理ネットワークが、ノード IP 間のトラフィックをルーティングする方法は認識しているものの、ポッド CIDR を直接ルーティングする方法は認識していないためです。

CNI は、元のパケット全体を VXLAN フレーム内にカプセル化します。これにより、「パケット内のパケット」が次のような新しいヘッダー付きで効果的に作成されます。

```
+-----------------+----------------+-------------+-----------
+--------------------------+
| Outer Ethernet  | Outer IP       | Outer UDP   | VXLAN      | Original Pod-to-Pod
    |
| Src: Node1 MAC  | Src: 10.80.0.2 | Src: Random | VNI: 42    | Packet (unchanged
    |
| Dst: Router MAC | Dst: 10.80.0.3 | Dst: 8472   |            | from above)
    |
+-----------------+----------------+-------------+-----------
+--------------------------+
```

このカプセル化の重要なポイントは、外側のパケットにはノード 1 (10.80.0.2) からノード 2 (10.80.0.3) までのアドレスが指定されること、UDP ポート 8472 は Cilium がデフォルトで使用する VXLAN ポートであること、VXLAN Network Identifier (VNI) はこのパケットが属するオーバーレイネットワークを識別すること、元のパケット (ポッド A の IP を送信元とし、ポッド B の IP を宛先とする) 全体が内側でもそのまま保持されることです。

これで、カプセル化されたパケットは、ノード 1 の通常のネットワークスタックに入って、他のパケットと同じ方法で処理されます。

1. ノードネットワークでの処理: ノード 1 のネットワークスタックは、宛先 (10.80.0.3) に基づいてパケットをルーティングします。
2. ローカルネットワークでの配信:
   - 両方のノードが同じレイヤー 2 ネットワークにある場合、パケットはノード 2 に直接送信されます。
   - それぞれ別のサブネットにある場合、パケットはまずローカルルーターに転送されます。
3. ルーターでの処理: ルーターは、自身のルーティングテーブルに基づいてパケットを転送して、ノード 2 に配信します。

3. 受信側のノードでの処理

カプセル化されたパケットがノード 2 (10.80.0.3) に到達すると、次のようになります。

1. ノードのネットワークスタックは、パケットを受信し、VXLAN パケット (UDP ポート 4789) であると識別します。
2. パケットは、さらに処理するために CNI の VXLAN インターフェイスに渡されます。

4. VXLAN カプセル化解除

ノード 2 上の CNI は、次のように VXLAN パケットを処理します。

1. 外側のヘッダー (イーサネット、IP、UDP、VXLAN) を取り除きます。
2. 内側の元のパケットを抽出します。
3. これで、パケットは元の形式に戻ります。

```
+------------------+-----------------+-------------+-----------------+
| Ethernet Header  | IP Header       | TCP/UDP     | Payload         |
```

```
| Src: Pod A MAC   | Src: 10.85.1.56 | Src: 43721  |                 |
| Dst: Gateway MAC | Dst: 10.85.2.67 | Dst: 8080   |                 |
+------------------+-----------------+-------------+-----------------+
```

ノード 2 上の CNI は、宛先 IP (10.85.2.67) を確認して、次の処理を行います。

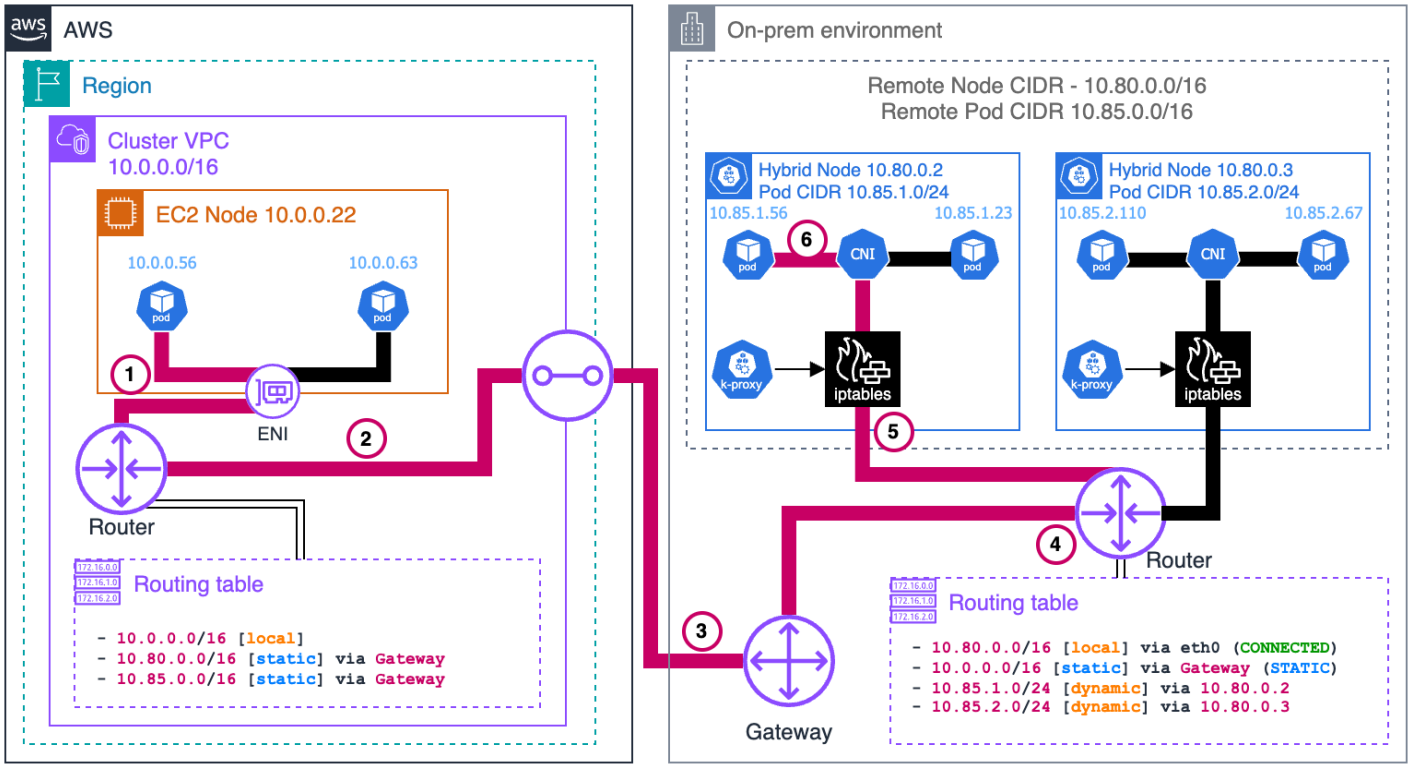
1. この IP がローカルポッドに属しているかどうかを識別します。
2. 適切な仮想インターフェイス経由でパケットをルーティングします。
3. ポッド B のネットワーク名前空間にパケットを配信します。

[応答]

ポッド B がポッド A に応答するときは、プロセス全体が逆方向になります。

4. ポッド B がポッド A (10.85.1.56) にパケットを送信します。
5. ノード 2 の CNI は、そのパケットを VXLAN でカプセル化し、宛先をノード 1 (10.80.0.2) に設定します。
6. カプセル化されたパケットがノード 1 に配信されます。
7. ノード 1 の CNI は、パケットをカプセル化解除し、元のレスポンスをポッド A に配信します。

クラウドノード上のポッドからハイブリッドノード上のポッドまでのフロー (東西トラフィック)



[リクエスト]

1. ポッド A が通信を開始する

EC2 ノード上のポッド A (10.0.0.56) がハイブリッドノード上のポッド B (10.85.1.56) にトラフィックを送信しようとしています。初期パケットは以下のようになります。

```
+-----------------+---------------------+-----------------+
| IP Header       | TCP Header          | Payload         |
| Src: 10.0.0.56  | Src: 52390 (random) |                 |
| Dst: 10.85.1.56 | Dst: 8080           |                 |
+-----------------+---------------------+-----------------+
```

VPC CNI では、ポッド A に VPC CIDR からの IP があるため、ポッド A は EC2 インスタンス上の ENI に直接アタッチされます。ポッドのネットワーク名前空間は VPC ネットワークに接続されているため、パケットは VPC ルーティングインフラストラクチャに直接入ります。

2. VPC でのルーティング

VPC ルートテーブルにはリモートポッド CIDR (10.85.0.0/16) に固有のルートが登録されているため、このトラフィックは VPC-to-onprem ゲートウェイ宛てに送信されます。

```
Destination     Target
10.0.0.0/16     local
10.85.0.0/16    vgw-id (VPC-to-onprem gateway)
```

このルーティングルールに基づいて、パケットはオンプレミスネットワークに接続しているゲートウェイに向けて送信されます。

3. 境界を越えて移動する

ゲートウェイは、確立済みの接続 (Direct Connect や VPN など) 経由でクラウド境界を越えてオンプレミスネットワークにパケットを転送します。このトランジットを全体を通して、パケットの送信元 IP アドレスと宛先 IP アドレスは元のまま維持されます。

4. オンプレミスネットワークでの受信

パケットは、ローカルオンプレミスルーターに到達します。ルーターは、自身のルーティングテーブルに問い合わせて、10.85.1.56 アドレスに到達するための次のホップを決定します。オンプレミスルーターには、適切なハイブリッドノード宛てにトラフィックを送信するポッド CIDR へのルートが必要です。

ルーターのルートテーブルには IP 10.80.0.2 のハイブリッドノード経由で 10.85.1.0/24 サブネットに到達できることを示すエントリが登録されています。

```
Destination     Next Hop
10.85.1.0/24    10.80.0.2
```

5. ノードネットワークでの処理

ルーターは、パケットをハイブリッドノード (10.80.0.2) に転送します。パケットがノードに到達したとき、パケットのポッド A の IP は送信元のまま、ポッド B の IP は宛先のままです。

6. CNI での処理

ノードのネットワークスタックは、パケットを受信し、宛先 IP が自身のものでないことを確認して、さらに処理するために CNI に渡します。CNI は、宛先 IP がこのノードでローカルに実行されているポッドに属しているかどうかを識別し、適切な仮想インターフェイス経由で適切なポッドにパケットを転送します。

```
Original packet -> node routing -> CNI -> Pod B's network namespace
```

ポッド B は、パケットを受信し、必要に応じて処理します。

[応答]

6. ポッド B がレスポンスを送信する

ポッド B は、自身の IP を送信元、ポッド A の IP を宛先として、レスポンスパケットを作成します。

```
+-----------------+---------------------+-----------------+
| IP Header       | TCP Header          | Payload         |
| Src: 10.85.1.56 | Src: 8080           |                 |
| Dst: 10.0.0.56  | Dst: 52390          |                 |
+-----------------+---------------------+-----------------+
```

CNI は、このパケットが外部ネットワーク宛てかどうかを識別して、ノードのネットワークスタックに渡します。

5. ノードネットワークでの処理

ノードは、宛先 IP (`10.0.0.56`) がローカルネットワークに属していないと判断し、自身のデフォルトゲートウェイ (ローカルルーター) にパケットを転送します。

4. ローカルルーターのルーティング

ローカルルーターは、自身のルーティングテーブルに問い合わせて、宛先 IP (`10.0.0.56`) が VPC CIDR (`10.0.0.0/16`) に属していると判断します。次に、AWS に接続しているゲートウェイにパケットを転送します。

3. 境界を越えて移動する

パケットは、同じオンプレミスを通って VPC 接続まで戻り、クラウド境界を逆方向に横断します。

2. VPC でのルーティング

パケットが VPC に到達すると、ルーティングシステムは宛先 IP が VPC 内のサブネットに属しているかどうかを識別します。パケットは、ポッド A をホストしている EC2 インスタンスに向けて、VPC ネットワーク経由でルーティングされます。

1. ポッド A がレスポンスを受信する

パケットは、EC2 インスタンスに到達し、そこにアタッチされている ENI 経由でポッド A に直接配信されます。VPC CNI は VPC 内のポッドにオーバーレイネットワーキングを使用しないため、別途カプセル化解除は必要なく、パケットは元のヘッダーのまま届きます。

この東西トラフィックフローは、なぜリモートポッド CIDR を正しく設定して、両方向からルーティング可能にする必要があるかを示しています。

- VPC には、オンプレミスゲートウェイを指すリモートポッド CIDR のルートが登録されている必要があります。
- オンプレミスネットワークには、そうしたポッドをホストしている特定のノード宛てにトラフィックを送信するルートがポッド CIDR で登録されている必要があります。

ハイブリッドノード nodeadm 参照

Amazon EKS Hybrid Nodes CLI (nodeadm) は、ハイブリッドノードコンポーネントのインストール、設定、登録、アンインストールを簡素化します。オペレーティングシステムイメージに nodeadm を含めると、ハイブリッドノードのブートストラップを自動化できます。詳細については、「the section called “オペレーティングシステムを準備する”」を参照してください。

ハイブリッドノードの nodeadm バージョンは、Amazon EKS クラスター内のノードとして Amazon EC2 インスタンスをブートストラップするために使用される nodeadm バージョンとは異なります。適切な nodeadm バージョンのドキュメントとリファレンスに従ってください。このドキュメントページは、ハイブリッドノードの nodeadm バージョンを対象としています。

ハイブリッドノード nodeadm のソースコードは、https://github.com/aws/eks-hybrid GitHub リポジトリで公開されています。

> **Important**
>
> root/sudo 権限を持つユーザーで nodeadm を実行する必要があります。

> **SSM 認証情報プロバイダーに必要な nodeadm バージョン**
>
> ハイブリッドノードの認証情報プロバイダーとして AWS Systems Manager (SSM) を使用する場合は、新しいインストールとアップグレードに nodeadm バージョン `1.0.19` 以降を使用する必要があります。以前のバージョンの nodeadm には古い SSM 署名キーが含まれてお

り、nodeadm install と nodeadm upgrade の実行中に失敗し、次の署名検証エラーが発生します:

```
"msg":"Command failed","error":"failed to install ssm installer: validating ssm-setup-cli signature: Signature Verification Error: No matching signature"
```

このエラーを解決するには、nodeadm install または nodeadm upgrade を実行する前に nodeadm の最新バージョンをダウンロードします。

nodeadm のダウンロード

nodeadm のハイブリッドノード版は Amazon CloudFront をフロントに置いた Amazon S3 でホストされます。各オンプレミスホストに nodeadm をインストールするにはオンプレミスホストから以下のコマンドを実行してください。

x86_64 ホストの場合

```
curl -OL 'https://hybrid-assets.eks.amazonaws.com/releases/latest/bin/linux/amd64/nodeadm'
```

ARM ホストの場合

```
curl -OL 'https://hybrid-assets.eks.amazonaws.com/releases/latest/bin/linux/arm64/nodeadm'
```

各ホストでダウンロードしたバイナリに実行可能ファイルのアクセス許可を追加します。

```
chmod +x nodeadm
```

nodeadm install

nodeadm install コマンドはハイブリッドノードを実行して Amazon EKS クラスターに結合するために必要なアーティファクトと依存関係をインストールするために使用されます。nodeadm install コマンドは各ハイブリッドノードで個別に実行することも、イメージビルドパイプライン中に実行して、オペレーティングシステムイメージにハイブリッドノードの依存関係をプリインストールすることもできます。

使用方法

```
nodeadm install [KUBERNETES_VERSION] [flags]
```

位置引数

(必須 KUBERNETES_VERSION インストールする EKS Kubernetes のメジャーバージョンとマイナーバージョン (例: 1.32)

Flags

名前	必要	説明
-p, --credential-provider	正	インストールする認証情報プロバイダー。サポートされている値はiam-ra および ssm です。詳細については「the section called “認証情報を準備する”」を参照してください。
-s, --containerd-source	誤	containerd のソース。nodeadm はOS ディストリビューション、Docker パッケージからの containerd のインストール、および containerd のインストールのスキップをサポートしています。 値 distro - これはデフォルト値です。nodeadm は、EKS Kubernetes バージョンと互換性のある、ノード OS によって配布される最新の containerd パッケージをインストールします。distro は Red Hat Enterprise Linux (RHEL) オペレーティングシス

名前	必要	説明
		テムでサポートされる値ではありません。 docker — nodeadm は、EKS Kubernetes バージョンと互換性のある、Docker によって構築および配布された最新の containerd パッケージをインストールします。docker は Amazon Linux 2023 でサポートされる値ではありません。 none — nodeadm は containerd パッケージをインストールしません。nodeadm init を実行する前に、containerd を手動でインストールする必要があります。
-r, --region	誤	SSM エージェントといったアーティファクトをダウンロードするための AWS リージョンを指定します。デフォルトは us-west-2 です。
-t, --timeout	誤	install コマンドの最大実行時間。入力は時間形式に従います。例: 1h23m。install コマンドのデフォルトのダウンロードタイムアウトは 20 分に設定されています。

名前	必要	説明
-h, --help	誤	使用可能なフラグ、サブコマンド、位置値パラメータを含むヘルプメッセージを表示します。

例

認証情報プロバイダーとして AWS システムマネージャー (SSM 使用して Kubernetes バージョン 1.32 をインストールする

```
nodeadm install 1.32 --credential-provider ssm
```

AWS システムマネージャー (SSM 認証情報プロバイダーとして、Docker を containerd のソースとして、ダウンロードタイムアウトを 20 分にして Kubernetes バージョン 1.32 をインストールします。

```
nodeadm install 1.32 --credential-provider ssm --containerd-source docker --timeout 20m
```

認証情報プロバイダーとして AWS IAM Roles Anywhere を使用して Kubernetes バージョン 1.32 をインストールする

```
nodeadm install 1.32 --credential-provider iam-ra
```

nodeadm config check

nodeadm config check コマンドは提供されたノード設定にエラーがないか確認します。このコマンドはハイブリッドノード設定ファイルの正確性を検証するために使用できます。

使用方法

```
nodeadm config check [flags]
```

フラグ

名前	必要	説明
-c, --config-source	正	nodeadm 設定のソース。ハイブリッドノードの場合、入力は file スキームを使用する URI に従う必要があります。
-h, --help	誤	使用可能なフラグ、サブコマンド、位置値パラメータを含むヘルプメッセージを表示します。

例

```
nodeadm config check -c file://nodeConfig.yaml
```

nodeadm init

nodeadm init コマンドはハイブリッドノードを起動し、設定された Amazon EKS クラスターに接続します。nodeConfig.yaml ファイルの設定方法の詳細については、「the section called “SSM ハイブリッドアクティベーション用のノード設定”」または「the section called “IAM Roles Anywhere 用のノード設定”」を参照してください。

使用方法

```
nodeadm init [flags]
```

フラグ

名前	必要	説明
-c, --config-source	正	nodeadm 設定のソース。ハイブリッドノードの場合、入力は file スキームを使用する URI に従う必要があります。
-s,	誤	スキップする init のフェーズ。問題の修正に役立つ場

名前	必要	説明
--skip		合を除き、どのフェーズもスキップすることはお勧めしません。 値 install-validation は、先行する install コマンドが正常に実行されたかどうかの確認をスキップします。 ノードでファイアウォールが有効になっている場合、cni-validation は Cilium または Calico CNI の VXLAN ポートが開かれているかどうかの確認をスキップします。 node-ip-validation は、ノード IP がリモートノードネットワークの CIDR 内に含まれているかどうかのチェックをスキップします。
-h, --help	誤	使用可能なフラグ、サブコマンド、位置値パラメータを含むヘルプメッセージを表示します。

例

```
nodeadm init -c file://nodeConfig.yaml
```

nodeadm upgrade

nodeadm upgrade コマンドはインストールされているすべてのアーティファクトを最新バージョンにアップグレードし、ノードをブートストラップしてアップグレードされたアーティファクトを設定し、AWS の EKS クラスターに参加させます。アップグレードはノードで実行されているワークロードに対する破壊的なコマンドです。アップグレードを実行する前にはワークロードを別のノードに移動してください。

使用方法

```
nodeadm upgrade [KUBERNETES_VERSION] [flags]
```

位置引数

(必須 KUBERNETES_VERSION インストールする EKS Kubernetes のメジャーバージョンとマイナーバージョン (例: 1.32)

Flags

名前	必要	説明
-c, --config-source	正	nodeadm 設定のソース。ハイブリッドノードの場合、入力は file スキームを使用する URI に従う必要があります。
-t, --timeout	誤	アーティファクトのダウンロードのタイムアウト。入力は時間形式に従います。たとえば、1h23m です。upgrade コマンドのデフォルトのダウンロードタイムアウトは 10 分に設定されています。
-s, --skip	誤	スキップするアップグレードフェーズ。問題の修正に役立つ場合を除き、どのフェーズもスキップすることはお勧めしません。

名前	必要	説明
		値 pod-validation はデーモンセットと静的ポッドを除くすべてのポッドがノード上で実行されていないことのチェックをスキップします。 node-validation はノードが隔離されているかどうかの確認をスキップします。 init-validation はアップグレードを実行する前にノードが正常に初期化されたかどうかの確認をスキップします。 containerd-major-version-upgrade は、ノードのアップグレード時に containerd のメジャーバージョンがアップグレードされないようにします。
-h, --help	誤	使用可能なフラグ、サブコマンド、位置値パラメータを含むヘルプメッセージを表示します。

例

```
nodeadm upgrade 1.32 -c file://nodeConfig.yaml
```

```
nodeadm upgrade 1.32 -c file://nodeConfig.yaml --timeout 20m
```

nodeadm uninstall

nodeadm uninstall コマンドは kubelet や containerd など、nodeadm install の実行中に nodeadm がインストールしたアーティファクトを停止および削除してください。uninstall コマンドはクラスターからハイブリッドノードをドレインまたは削除しないことに注意してください。ドレインと削除の操作は個別に実行する必要があります。詳細については「the section called “ハイブリッドノードを削除する”」を参照してください。デフォルトではノードにポッドが残っている場合、nodeadm uninstall は続行されません。同様に、nodeadm uninstall は CNI 依存関係や、クラスターで実行している他の Kubernetes アドオンの依存関係を削除しません。ホストから CNI インストールを完全に削除するには「the section called “CNI を設定する”」の手順を参照してください。オンプレミス認証情報プロバイダーとして AWS SSM ハイブリッドアクティベーションを使用している場合、nodeadm uninstall コマンドはホストを AWS SSM マネージドインスタンスから登録解除します。

使用方法

```
nodeadm uninstall [flags]
```

フラグ

名前	必要	説明
-s, --skip	誤	アンインストール時にスキップするフェーズ。問題の修正に役立つ場合を除き、どのフェーズもスキップすることはお勧めしません。 値 pod-validation はデーモンセットと静的ポッドを除くすべてのポッドがノード上で実行されていないことのチェックをスキップします。 node-validation はノードが隔離されているかどうかの確認をスキップします。

名前	必要	説明
		`init-validation` はアンインストールを実行する前にノードが正常に初期化されたかどうかの確認をスキップします。

`-h,`
 	
`--help`
	誤	使用可能なフラグ、サブコマンド、位置値パラメータを含むヘルプメッセージを表示します。

名前	必要	説明
-f, --force	誤	Kubernetes および CNI コンポーネントからの残存ファイルを含む可能性のある追加のディレクトリを強制的に削除します。 警告 これにより、デフォルトの Kubernetes および CNI ディレクトリのすべてのコンテンツ (/var/lib/cni 、/etc/cni/net.d など) が削除されます。これらの場所に独自のデータを保存する場合は、このフラグを使用しないでください。 nodeadm v1.0.9 以降、./nodeadm uninstall --skip node-validation,pod-validation --force コマンドは /var/lib/kubelet ディレクトリを削除しなくなりました。これは、マウントされたノードファイルシステムを含むことがある Pod ボリュームと volume-subpath ディレクトリが含まれている可能性があるためです。 安全な処理のヒント - マウントされたパスを削除すると、実際にマウントさ

名前	必要	説明
		れたノードファイルシステムが誤って削除される可能性があります。/var/lib/kubelet ディレクトリを手動で削除する前に、すべてのアクティブなマウントを慎重に検査し、ボリュームを安全にマウント解除して、データ損失を回避します。

例

```
nodeadm uninstall
```

```
nodeadm uninstall --skip node-validation,pod-validation
```

nodeadm debug

nodeadm debug コマンドは異常または誤設定のハイブリッドノードのトラブルシューティングに使用できます。以下の要件が満たされていることを検証します。

- ノードが認証情報を取得するために必要な AWS API へのネットワークアクセスがあること
- ノードが設定されたハイブリッドノードの IAM ロールに対し AWS 認証情報を取得できること
- ノードに EKS Kubernetes API エンドポイントへのネットワークアクセスがあり、EKS Kubernetes API エンドポイント証明書が有効であること
- ノードが EKS クラスターに対し認証でき、クラスター内での ID が有効で、EKS クラスター用に設定された VPC を介して EKS クラスターにアクセスできること。

エラーが見つかった場合、コマンドの出力はトラブルシューティングのステップを提示します。特定の検証ステップは子プロセスを示しています。これらが失敗した場合、出力は検証エラーの下の stderr セクションに表示されます。

使用方法

```
nodeadm debug [flags]
```

フラグ

名前	必要	説明
-c, --config-source	正	nodeadm 設定のソース。ハイブリッドノードの場合、入力は file スキームを使用する URI に従う必要があります。
--no-color	誤	色の出力を無効にします。自動化に役立ちます。
-h, --help	誤	使用可能なフラグ、サブコマンド、位置値パラメータを含むヘルプメッセージを表示します。

例

```
nodeadm debug -c file://nodeConfig.yaml
```

Nodeadm ファイルの場所

nodeadm のインストール

`nodeadm install` を実行すると、以下のファイルおよびファイルの場所が設定されます。

アーティファクト	パス
IAM Roles Anywhere	/usr/local/bin/aws_signing_helper
Kubelet binary	/usr/bin/kubelet
Kubectl binary	usr/local/bin/kubectl

アーティファクト	パス
ECR Credentials Provider	/etc/eks/image-credential-provider/ecr-credential-provider
AWS IAM オーセンティケーター	/usr/local/bin/aws-iam-authenticator
SSM Setup CLI	/opt/ssm/ssm-setup-cli
SSM Agent	Ubuntu の場合 - /snap/amazon-ssm-agent/current/amazon-ssm-agent RHEL および AL2023 の場合 - /usr/bin/amazon-ssm-agent
Containerd	Ubuntu および AL2023 の場合 - /usr/bin/containerd RHEL の場合 - /bin/containerd
Iptables	Ubuntu および AL2023 の場合 - /usr/sbin/iptables RHEL の場合 - /sbin/iptables
CNI プラグイン	/opt/cni/bin
インストール済みアーティファクトトラッカー	/opt/nodeadm/tracker

nodeadm init

`nodeadm init` を実行すると、以下のファイルおよびファイルの場所が設定されます。

名前	パス
Kubelet kubeconfig	/var/lib/kubelet/kubeconfig
Kubelet 設定	/etc/kubernetes/kubelet/config.json
Kubelet systemd ユニット	/etc/systemd/system/kubelet.service

名前	パス
イメージ認証情報プロバイダー設定	/etc/eks/image-credential-provider/config.json
Kubelet 環境ファイル	/etc/eks/kubelet/environment
Kubelet 証明書	/etc/kubernetes/pki/ca.crt
Containerd 設定	/etc/containerd/config.toml
Containerd カーネルモジュール設定	/etc/modules-load.d/containerd.conf
AWS 設定ファイル	/etc/aws/hybrid/config
AWS 認証情報ファイル (認証情報ファイルを有効にする場合)	/eks-hybrid/.aws/credentials
AWS 署名ヘルパーシステムユニット	/etc/systemd/system/aws_signing_helper_update.service
Sysctl 設定ファイル	/etc/sysctl.d/99-nodeadm.conf
Ca-certificates	/etc/ssl/certs/ca-certificates.crt
Gpg キーファイル	/etc/apt/keyrings/docker.asc
Docker リポジトリソースファイル	/etc/apt/sources.list.d/docker.list

SSM ハイブリッドアクティベーション用のノード設定

ハイブリッドノードの認証情報に AWS SSM ハイブリッドアクティベーションを使用する場合の `nodeConfig.yaml` の例を以下に示します。

```
apiVersion: node.eks.aws/v1alpha1
kind: NodeConfig
spec:
  cluster:
    name:             # Name of the EKS cluster
    region:           # AWS Region where the EKS cluster resides
  hybrid:
    ssm:
```

```
      activationCode: # SSM hybrid activation code
      activationId:   # SSM hybrid activation id
```

IAM Roles Anywhere 用のノード設定

ハイブリッドノードの認証情報用の AWS IAM Roles Anywhere の `nodeConfig.yaml` の例を以下に示します。

AWS IAM Roles Anywhere をオンプレミス認証情報プロバイダーとして使用する場合、nodeadm 設定で使用する nodeName はハイブリッドノードの IAM ロールを対象としたアクセス許可と一致する必要があります。たとえば、ハイブリッドノードの IAM ロールのアクセス許可ではロールセッション名がホスト証明書の CN と一致する場合しか AWS IAM Roles Anywhere がロールを引き受けることは許可されない場合、nodeadm 設定の nodeName は証明書の CN と同じである必要があります。使用する nodeName は 64 文字以下にする必要があります。詳細については、the section called “認証情報を準備する” を参照してください。

```
apiVersion: node.eks.aws/v1alpha1
kind: NodeConfig
spec:
  cluster:
    name:               # Name of the EKS cluster
    region:             # AWS Region where the EKS cluster resides
  hybrid:
    iamRolesAnywhere:
      nodeName:         # Name of the node
      trustAnchorArn:   # ARN of the IAM Roles Anywhere trust anchor
      profileArn:       # ARN of the IAM Roles Anywhere profile
      roleArn:          # ARN of the Hybrid Nodes IAM role
      certificatePath: # Path to the certificate file to authenticate with the IAM
 Roles Anywhere trust anchor
      privateKeyPath:  # Path to the private key file for the certificate
```

kubelet をカスタマイズするためのノード設定 (オプション)

nodeadm 設定で kubelet 設定と flags を渡すことができます。追加のノードラベル `abc.example.com/test-label` を追加する方法と、kubelet config `shutdownGracePeriod` を 30 秒に設定する方法については、以下の例を参照してください。kubelet 設定オプションの詳細については、Kubernetes ドキュメントの「Kubelet 設定 (v1beta1) リファレンス」を参照してください。kubelet コマンドラインフラグの詳細については、Kubernetes ドキュメントの「kubelet CLI リファレンス」を参照してください。

```
apiVersion: node.eks.aws/v1alpha1
kind: NodeConfig
spec:
  cluster:
    name:             # Name of the EKS cluster
    region:           # AWS Region where the EKS cluster resides
  kubelet:
    config:           # Map of kubelet config and values
       shutdownGracePeriod: 30s
    flags:            # List of kubelet flags
       - --node-labels=abc.example.com/test-label=true
  hybrid:
    ssm:
      activationCode: # SSM hybrid activation code
      activationId:   # SSM hybrid activation id
```

containerd をカスタマイズするためのノード設定 (オプション)

nodeadm 設定でカスタム containerd 設定を渡すことができます。nodeadm の containerd 設定はインライン TOML を受け入れます。containerd コンテンツストアで解凍されたイメージレイヤーの削除を無効にするための containerd の設定方法については以下の例を参照してください。

```
apiVersion: node.eks.aws/v1alpha1
kind: NodeConfig
spec:
  cluster:
    name:             # Name of the EKS cluster
    region:           # AWS Region where the EKS cluster resides
  containerd:
    config: |         # Inline TOML containerd additional configuration
       [plugins."io.containerd.grpc.v1.cri".containerd]
       discard_unpacked_layers = false
  hybrid:
    ssm:
      activationCode: # SSM hybrid activation code
      activationId:   # SSM hybrid activation id
```

> **Note**
>
> containerd のバージョン 1.x と 2.x では、異なる設定形式が使用されます。containerd 1.x は設定バージョン 2 を使用し、containerd 2.x は設定バージョン 3 を使用します。containerd

2.x は設定バージョン 2 と下位互換性がありますが、最適なパフォーマンスを得るには設定バージョン 3 をお勧めします。`containerd --version` を使用して containerd バージョンを確認するか、nodeadm のインストールログを確認します。設定のバージョニングの詳細については、https://containerd.io/releases/ を参照してください。

containerd 設定を使用して SELinux サポートを有効にすることもできます。containerd で SELinux が有効になっている場合はノードでスケジュールされたポッドで適切な securityContext と seLinuxOptions が有効になっていることを確認してください。セキュリティコンテキストの設定の詳細については Kubernetes ドキュメントを参照してください。

Note

Red Hat Enterprise Linux (RHEL 8 および RHEL 9 では SELinux がデフォルトで有効になっており、ホスト上で strict に設定されています。Amazon Linux 2023 はデフォルトで SELinux が有効になっており、許可モードに設定されています。SELinux がホストで許可モードに設定されている場合、containerd で有効にしてもリクエストはブロックされませんが、ホストの SELinux 設定に従ってログに記録されます。

```
apiVersion: node.eks.aws/v1alpha1
kind: NodeConfig
spec:
  cluster:
    name:             # Name of the EKS cluster
    region:           # AWS Region where the EKS cluster resides
  containerd:
    config: |         # Inline TOML containerd additional configuration
       [plugins."io.containerd.grpc.v1.cri"]
       enable_selinux = true
  hybrid:
    ssm:
      activationCode: # SSM hybrid activation code
      activationId:   # SSM hybrid activation id
```

ハイブリッドノードのトラブルシューティング

このトピックでは、Amazon EKS Hybrid Nodes の使用中に表示される一般的なエラーとその修復方法について説明します。その他のトラブルシューティング情報については、「トラブルシューティン

グ」および AWS re:Post の「Amazon EKS のナレッジセンタータグ」を参照してください。問題を解決できない場合は、AWS サポートにお問い合わせください。

> ⚠ SSM 認証情報プロバイダーに必要な nodeadm バージョン
>
> ハイブリッドノードの認証情報プロバイダーとして AWS Systems Manager (SSM) を使用する場合は、新しいインストールとアップグレードに nodeadm バージョン `1.0.19` 以降を使用する必要があります。以前のバージョンの nodeadm には古い SSM 署名キーが含まれており、`nodeadm install` と `nodeadm upgrade` の実行中に失敗し、次の署名検証エラーが発生します:
>
> ```
> "msg":"Command failed","error":"failed to install ssm installer: validating ssm-setup-cli signature: Signature Verification Error: No matching signature"
> ```
>
> このエラーを解決するには、`nodeadm install` または `nodeadm upgrade` を実行する前に nodeadm の最新バージョンをダウンロードします。

`nodeadm debug` を使用したノードのトラブルシューティング ハイブリッドノードから nodeadm debug コマンドを実行して、ネットワークと認証情報の要件が満たされていることを検証できます。`nodeadm debug` コマンドの詳細については、the section called “ハイブリッドノード用 nodeadm” を参照してください。

クラスターインサイトを使用してハイブリッドノードの問題を検出する Amazon EKS クラスターインサイトには、クラスター内の EKS Hybrid Nodes の設定に関する一般的な問題を検出するインサイトチェックが含まれます。すべてのインサイトチェックの結果は AWS マネジメントコンソール、AWS CLI、および AWS SDK から表示できます。クラスターインサイトの詳細については、「the section called “クラスターのインサイト”」を参照してください。

ハイブリッドノードのインストールに関するトラブルシューティング

以下のトラブルシューティングのトピックは、`nodeadm install` コマンドを使用してホストにハイブリッドノードの依存関係をインストールすることに関連しています。

`nodeadm` コマンドが `must run as root` に失敗した

`nodeadm install` コマンドは、ホストに対して root または sudo 権限を持つユーザーに実行させる必要があります。root または sudo 権限を持たないユーザーに `nodeadm install` を実行させると、nodeadm 出力に次のエラーが表示されます。

```
"msg":"Command failed","error":"must run as root"
```

依存関係に接続できない

nodeadm install コマンドは、ハイブリッドノードに必要な依存関係をインストールします。ハイブリッドノードの依存関係には、containerd、kubelet、kubectl、および AWS SSM または AWS IAM Roles Anywhere コンポーネントがあります。これらの依存関係をダウンロードするには、nodeadm install を実行している場所からアクセスする必要があります。アクセスする必要があるロケーションのリストの詳細については、the section called “ネットワークを準備する” を参照してください。アクセス権限がない場合は、次のようなエラーが nodeadm install 出力に表示されます。

```
"msg":"Command failed","error":"failed reading file from url: ...: max retries achieved
 for http request"
```

パッケージマネージャーの更新に失敗しました

nodeadm install コマンドは、ハイブリッドノードの依存関係をインストールする前に、apt update、yum update、dnf update のいずれかを実行します。このステップが失敗すると、次のようなエラーが表示される可能性があります。これを修正するには、apt update、yum update、dnf update のいずれかを実行してから nodeadm install を実行するか、nodeadm install を再実行します。

```
failed to run update using package manager
```

タイムアウトまたはコンテキストの期限を超過しました

nodeadm install を実行するときに、インストールプロセスのさまざまな段階でタイムアウトまたはコンテキストの期限を超過したことを示すエラーの問題が発生した場合、接続が遅くなり、タイムアウトが完了する前にハイブリッドノードの依存関係がインストールされない可能性があります。これらの問題を回避するには、nodeadm の --timeout フラグを使用して、依存関係のダウンロードのタイムアウト期間を延長できます。

```
nodeadm install K8S_VERSION --credential-provider CREDS_PROVIDER --timeout 20m0s
```

ハイブリッドノードの接続のトラブルシューティング

このセクションのトラブルシューティングトピックは、nodeadm init コマンドを使用してハイブリッドノードを EKS クラスターに接続するプロセスに関連しています。

オペレーションエラーまたはサポートされていないスキーム

nodeadm init を実行するときに、operation error または unsupported scheme に関連するエラーが表示された場合は、nodeConfig.yaml が正しくフォーマットされ、nodeadm に渡されているか確認します。nodeConfig.yaml のフォーマットのオプションの詳細については、「the section called “ハイブリッドノード用 nodeadm”」を参照してください。

```
"msg":"Command failed","error":"operation error ec2imds: GetRegion, request canceled,
 context deadline exceeded"
```

ハイブリッドノード IAM ロールに **eks:DescribeCluster** アクションのアクセス許可がありません

nodeadm init を実行すると、nodeadm は EKS DescribeCluster アクションを呼び出して EKS クラスターに関する情報の収集を試みます。ハイブリッドノード IAM ロールに eks:DescribeCluster アクションのアクセス許可がない場合は、nodeadm init init の実行時に nodeadm に渡すノード設定で、Kubernetes API エンドポイント、クラスター CA バンドル、およびサービス IPv4 CIDR を渡す必要があります。ハイブリッドノード IAM ロールに必須のアクセス許可の詳細については、「the section called “認証情報を準備する”」を参照してください。

```
"msg":"Command failed","error":"operation error EKS: DescribeCluster, https response
 error StatusCode: 403 ... AccessDeniedException"
```

ハイブリッドノード IAM ロールに **eks:ListAccessEntries** アクションのアクセス許可がありません

nodeadm init を実行すると、nodeadm は EKS ListAccessEntriesアクションを呼び出して、ハイブリッドノード IAM ロールに関連付けられた HYBRID_LINUX タイプのアクセスエントリが EKS クラスターにあるかどうかを検証しようとします。ハイブリッドノード IAM ロールに eks:ListAccessEntries アクションのアクセス許可がない場合は、nodeadm init コマンドの実行時に --skip cluster-access-validation フラグを渡す必要があります。ハイブリッドノード IAM ロールに必須のアクセス許可の詳細については、「the section called “認証情報を準備する”」を参照してください。

```
"msg":"Command failed","error":"operation error EKS: ListAccessEntries, https response
 error StatusCode: 403 ... AccessDeniedException"
```

リモートノードネットワークの CIDR にないノード IP

nodeadm init を実行するときに、指定されたリモートノードネットワークの CIDR の中にノードの IP アドレスがない場合、エラーが発生することがあります。エラーは次の例のような見た目になります。

```
node IP 10.18.0.1 is not in any of the remote network CIDR blocks [10.0.0.0/16
 192.168.0.0/16]
```

こちらの例では、IP 10.18.0.1 のノードが、リモートネットワークの CIDR 10.0.0.0/16 と 192.168.0.0/16 でクラスターを結合しようとしています。10.18.0.1 がいずれの範囲にもないため、エラーが発生しています。

RemoteNodeNetworks が正しく設定され、すべてのノード IP アドレスが含まれていることを確認します。ネットワーク設定の詳細については、「the section called “ネットワークを準備する”」を参照してください。

- お使いのクラスターが配置されているリージョンで次のコマンドを実行し、RemoteNodeNetwork 設定をチェックします。出力に表示された CIDR ブロックに、ご自身のノードの IP 範囲が含まれ、エラーメッセージに表示された CIDR ブロックと同一であることを確認します。同じでない場合は、クラスター名と nodeConfig.yaml のリージョンが、対象となるクラスターと一致していることを確認します。

```
aws eks describe-cluster --name CLUSTER_NAME --region REGION_NAME --query
 cluster.remoteNetworkConfig.remoteNodeNetworks
```

- 対象のノードで作業していることを確認します。
  - ホスト名と IP アドレスが、クラスターに登録しようとしているものと一致することを確認して、正しいノードで作業していることを確認します。
  - このノードが正しいオンプレミスネットワーク (クラスターのセットアップ中に CIDR 範囲が RemoteNodeNetwork として登録されたネットワーク) にあることを確認します。

それでもやはりノード IP が想定されるものと異なる場合は、以下を確認します。

- IAM Roles Anywhere を使用している場合、kubelet が DNS ルックアップを IAM Roles Anywhere で実行し、nodeName がノード名 (使用できる場合) に関連付けられた IP を使用している。ノードの DNS エントリを維持している場合、これらのエントリがリモートノードネットワークの CIDR 内の IP を指していることを確認する。
- ノードに複数のネットワークインターフェイスがある場合、kubelet は、リモートノードネットワークの CIDR の、範囲外の IP アドレスを持つインターフェイスをデフォルトとして選択することがあります。別のインターフェイスを使用するには、ご自身の nodeConfig.yaml の --node-ip kubelet フラグを使用して、その IP アドレスを指定します。詳細については、the section called “ハイブリッドノード用 nodeadm” を参照してください。ノードで次のコマンドを実行すると、ノードのネットワークインターフェイスとその IP アドレスを確認できます。

```
ip addr show
```

ハイブリッドノードが EKS クラスターに表示されない

nodeadm init を実行して完了しても、ハイブリッドノードがクラスターに表示されない場合、ハイブリッドノードと EKS コントロールプレーン間のネットワーク接続に問題があるか、必要なセキュリティグループのアクセス許可が設定されていないか、Kubernetes ロールベースアクセスコントロール (RBAC) へのハイブリッドノード IAM ロールの必要なマッピングがない可能性があります。次のコマンドを使用して kubelet と kubelet ログのステータスを確認することで、デバッグプロセスを開始できます。クラスターに参加できなかったハイブリッドノードから次のコマンドを実行します。

```
systemctl status kubelet
```

```
journalctl -u kubelet -f
```

クラスターと通信できません

ハイブリッドノードがクラスターコントロールプレーンと通信できなかった場合は、次のようなログが表示されることがあります。

```
"Failed to ensure lease exists, will retry" err="Get ..."
```

```
"Unable to register node with API server" err="Post ..."
```

```
Failed to contact API server when waiting for CSINode publishing ... dial tcp <ip
 address>: i/o timeout
```

これらのメッセージが表示された場合は、以下をチェックして、the section called “ネットワークを準備する” で説明されているハイブリッドノード要件を満たしていることを確認します。

- EKS クラスターに渡される VPC に、オンプレミスノードとオプションでポッド CIDR の Transit Gateway (TGW) または仮想プライベートゲートウェイ (VGW) へのルートがあることを確認します。
- EKS クラスターに追加のセキュリティグループがあり、オンプレミスノード CIDR とオプションでポッド CIDR のインバウンドルールがあることを確認します。
- EKS コントロールプレーンとの間のトラフィックを許可するようにオンプレミスルーターが設定されていることを確認します。

Unauthorized

ハイブリッドノードが EKS コントロールプレーンと通信できたが、登録できなかった場合、次のようなログが表示されることがあります。以下のログメッセージの主な違いは `Unauthorized` エラーであることに注意してください。これは、必要なアクセス許可がないため、ノードがタスクを実行できなかったことを示します。

```
"Failed to ensure lease exists, will retry" err="Unauthorized"
```

```
"Unable to register node with API server" err="Unauthorized"
```

```
Failed to contact API server when waiting for CSINode publishing: Unauthorized
```

これらのメッセージが表示された場合は、以下をチェックして、the section called “認証情報を準備する” および the section called “クラスターアクセスを準備する” のハイブリッドノードの要件の詳細を満たしていることを確認します。

- ハイブリッドノードのアイデンティティが、想定のハイブリッドノードの IAM ロールと一致することを確認します。これを行うには、ハイブリッドノードから `sudo aws sts get-caller-identity` を実行します。
- ハイブリッドノード IAM ロールに必要なアクセス許可があることを確認します。

- クラスターにハイブリッドノード IAM ロールの EKS アクセスエントリがあることを確認するか、aws-auth ConfigMap にハイブリッドノード IAM ロールのエントリがあることを確認します。EKS アクセスエントリを使用している場合は、ハイブリッドノード IAM ロールのアクセスエントリが HYBRID_LINUX アクセスタイプであることを確認します。aws-auth ConfigMap を使用している場合は、ハイブリッドノード IAM ロールのエントリが、「the section called “クラスターアクセスを準備する”」で説明されている要件と形式を満たしていることを確認します。

ハイブリッドノードが EKS クラスターに登録されているのにステータス **Not Ready** が表示される

ハイブリッドノードが EKS クラスターに正常に登録されたが、ハイブリッドノードのステータスが Not Ready である場合、最初に確認すべきことは、コンテナネットワークインターフェイス (CNI) のステータスです。CNI をインストールしていない場合は、ハイブリッドノードのステータスが Not Ready であることが予想されます。CNI が正常にインストールされ実行されると、ノードのステータスは Ready に移行します。CNI をインストールしようとしたが、正常に実行されていない場合は、このページの the section called “ハイブリッドノード CNI のトラブルシューティング” を参照してください。

証明書署名リクエスト (CSR) が保留中のままである

ハイブリッドノードを EKS クラスターに接続した後、ハイブリッドノードの保留中の CSR がある場合、ハイブリッドノードは自動承認の要件を満たしていません。ハイブリッドノードの CSR は、ハイブリッドノードの CSR が eks.amazonaws.com/compute-type: hybrid ラベル付きノードによって作成され、CSR に次のサブジェクト代替名 (SAN) がある場合、自動的に承認および発行されます: 少なくとも 1 つの DNS SAN はノード名と等しく、IP SAN はリモートノードネットワークの CIDR に属するものです。

ハイブリッドプロファイルは既に存在します

nodeadm 設定を変更し、ノードを新しい設定に再登録しようとすると、ハイブリッドプロファイルが既に存在するが、その内容が変更されたことを示すエラーが表示されることがあります。設定変更の合間に nodeadm init を実行する代わりに、nodeadm uninstall を実行し、その後に nodeadm install と nodeadm init を実行します。これにより、設定の変更に伴う適切なクリーンアップが保証されます。

```
"msg":"Command failed","error":"hybrid profile already exists at /etc/aws/hybrid/config
 but its contents do not align with the expected configuration"
```

ハイブリッドノードがプライベート API の解決に失敗しました

nodeadm init 実行後、no such host が存在するために EKS Kubernetes API サーバーへの接続に失敗したことを示すエラーが kubelet ログに表示された場合は、オンプレミスネットワークまたはホストレベルで EKS Kubernetes API エンドポイントの DNS エントリを変更する必要がある場合があります。「 AWS Route53 documentation」の「Forwarding inbound DNS queries to your VPC」を参照してください。

```
Failed to contact API server when waiting for CSINode publishing: Get ... no such host
```

EKS コンソールでハイブリッドノードを表示できない

ハイブリッドノードが登録されているのに、EKS コンソールで表示できない場合は、コンソールの表示に使用している IAM プリンシパルのアクセス許可を確認してください。使用している IAM プリンシパルには、コンソールでリソースを表示するための特定の最小の IAM および Kubernetes アクセス許可が必要です。詳細については、the section called “クラスターリソースにアクセスする” を参照してください。

ハイブリッドノードのトラブルシューティングの実行

EKS クラスターに登録されたハイブリッドノードのステータスが Ready で、その後ステータスが Not Ready に移行した場合、ノードに CPU、メモリ、使用可能なディスク容量の十分なリソースがない、ノードが EKS コントロールプレーンから切断されているなど、さまざまな問題が異常なステータスの原因となった可能性があります。以下のステップを使用してノードのトラブルシューティングを行うことができます。問題を解決できない場合は、AWS サポートにお問い合わせください。

ハイブリッドノードから nodeadm debug を実行して、ネットワークと認証情報の要件が満たされていることを確認します。nodeadm debug コマンドの詳細については、「the section called “ハイブリッドノード用 nodeadm”」を参照してください。

ノードのステータスを取得する

```
kubectl get nodes -o wide
```

ノードの条件とイベントを確認する

```
kubectl describe node NODE_NAME
```

ポッドのステータスを取得する

```
kubectl get pods -A -o wide
```

ポッドの条件とイベントを確認する

```
kubectl describe pod POD_NAME
```

ポッドログを確認する

```
kubectl logs POD_NAME
```

kubelet ログを確認する

```
systemctl status kubelet
```

```
journalctl -u kubelet -f
```

ポッドのライブネスプローブが失敗しているか、ウェブフックが機能していない

ハイブリッドノードで動作しているアプリケーション、アドオン、またはウェブフックが正しく起動しない場合、ポッドへの通信をブロックするネットワークの問題が発生する可能性があります。EKS コントロールプレーンがハイブリッドノードで動作しているウェブフックに接続するには、リモートポッドネットワークで EKS クラスターを設定し、VPC ルーティングテーブルにオンプレミスポッド CIDR のルートを設定し、ターゲットを Transit Gateway (TGW)、仮想プライベートゲートウェイ (VGW)、または VPC をオンプレミスネットワークに接続するために使用するその他のゲートウェイとして設定する必要があります。ハイブリッドノードのネットワーク要件の詳細については、the section called “ネットワークを準備する” を参照してください。さらに、オンプレミスのファイアウォールでこのトラフィックを許可し、ルーターがポッドに適切にルーティングできることを確認する必要があります。ハイブリッドノードでウェブフックを実行するための要件の詳細については、「the section called “ウェブフックの設定”」を参照してください。

このシナリオの一般的なポッドログメッセージを以下に示します。ここでは、ip-address は Kubernetes サービスのクラスター IP です。

```
dial tcp <ip-address>:443: connect: no route to host
```

`kubectl logs` または `kubectl exec` コマンドが機能していない (`kubelet` API コマンド)

他の `kubectl` コマンドは正常に実行されているのに、`kubectl attach`、`kubectl cp`、`kubectl exec`、`kubectl logs`、`kubectl port-forward` コマンドがタイムアウトした場合は、リモートネットワークの設定に問題がある可能性があります。これらのコマンドはクラスターを介してノードの `kubelet` エンドポイントに接続しています。詳細については、the section called “kubelet エンドポイント” を参照してください。

ノード IP とポッド IP が、ご自身のクラスター向けに設定したリモートノードネットワークおよびリモートポッドネットワーク CIDR の範囲に含まれていることを確認します。以下のコマンドを使用して IP の割り当てを調べます。

```
kubectl get nodes -o wide
```

```
kubectl get pods -A -o wide
```

これらの IP を、設定したリモートネットワークの CIDR と比較して、適切にルーティングされていることを確認します。ネットワークの設定要件については、「the section called “ネットワークを準備する”」を参照してください。

ハイブリッドノード CNI のトラブルシューティング

ハイブリッドノードで最初に Cilium または Calico を起動する際に問題が発生した場合は、多くの場合、ハイブリッドノードまたはハイブリッドノードで稼働する CNI ポッドと EKS コントロールプレーン間のネットワークの問題が原因です。環境が「ハイブリッドノードのネットワークを準備する」の要件を満たしていることを確認します。問題を分割すると分かりやすくなります。

EKS クラスター設定

- RemoteNodeNetwork と RemotePodNetwork の設定は正しいですか?

VPC 設定

- VPC ルーティングテーブルに、Transit Gateway または Virtual Private Gateway のターゲットを持つ RemoteNodeNetwork と RemotePodNetwork のルートはありますか?

セキュリティグループの設定

- RemoteNodeNetwork および RemotePodNetwork のインバウンドルールとアウトバウンドルールはありますか?

オンプレミスのネットワーク

EKS コントロールプレーンとハイブリッドノード、およびハイブリッドノードで実行されているポッドとの間のルートとアクセスはありますか?

CNI 設定

オーバーレイネットワークを使用している場合、CNI の IP プール設定は、ウェブフックを使用している場合に EKS クラスター用に設定された RemotePodNetwork と一致していますか?

CNI がインストールされていないハイブリッドノードのステータス **Ready**

ハイブリッドノードのステータスが Ready と表示されているが、クラスターに CNI をインストールしていない場合は、ハイブリッドノードに古い CNI アーティファクトが存在する可能性があります。デフォルトでは、Helm などのツールを使用して Cilium と Calico をアンインストールしても、ディスク上のリソースは物理マシンや仮想マシンからは削除されません。さらに、これらの CNI のカスタムリソース定義 (CRD) は、古いインストールからクラスターにまだ存在する場合があります。詳細については、the section called “CNI を設定する” の「Cilium と Calico の削除」セクションを参照してください。

Cilium のトラブルシューティング

ハイブリッドノードで Cilium の実行に問題がある場合は、Cilium ドキュメントの「トラブルシューティング手順」を参照してください。以下のセクションでは、ハイブリッドノードへの Cilium のデプロイに固有の問題について説明します。

Cilium が起動しない

各ハイブリッドノードで実行されている Cilium エージェントが開始されていない場合は、Cilium エージェントポッドのログにエラーがないか確認します。Cilium エージェントを起動するには、EKS Kubernetes API エンドポイントへの接続が必要です。この接続が正しく設定されていない場合、Cilium エージェントの起動は失敗します。この場合、Cilium エージェントポッドログに次のようなログメッセージが表示されます。

```
msg="Unable to contact k8s api-server"
level=fatal msg="failed to start: Get \"https://<k8s-cluster-ip>:443/api/v1/namespaces/
kube-system\": dial tcp <k8s-cluster-ip>:443: i/o timeout"
```

Cilium エージェントはホストネットワークで実行されます。EKS クラスターは、Cilium 接続用に RemoteNodeNetwork で設定する必要があります。EKS クラスターに RemoteNodeNetwork のインバウンドルールを持つ追加のセキュリティグループがあり、VPC にRemoteNodeNetwork のルー

トがあり、EKS コントロールプレーンへの接続を許可するようにオンプレミスネットワークが正しく設定されていることを確認します。

Cilium オペレータが実行されていて、一部の Cilium エージェントが動作しているが、すべてが動作していない場合は、クラスター内のすべてのノードに割り当てるポッド IP があることを確認します。Cilium 設定で `clusterPoolIPv4PodCIDRList` でクラスタープール IPAM を使用するときに、割り当て可能なポッド CIDR のサイズを設定します。ノードごとの CIDR サイズは、Cilium 設定の `clusterPoolIPv4MaskSize` 設定で設定されます。詳細については、Cilium ドキュメントの「Expanding the cluster pool」を参照してください。

Cilium BGP が機能しない

Cilium BGP コントロールプレーンを使用して、ポッドまたはサービスアドレスをオンプレミスネットワークにアドバタイズする場合は、次の Cilium CLI コマンドを使用して、BGP がリソースへのルートをアドバタイズしているかどうかを確認できます。Cilium CLI をインストールする手順については、Cilium ドキュメントの「Cilium CLI のインストール」を参照してください。

BGP が正しく動作している場合は、ハイブリッドノードの出力に Session State `established` が表示されるはずです。場合によっては、ネットワークチームと協力して、環境のローカル AS、ピア AS、ピアアドレスの正しい値を特定する必要があります。

```
cilium bgp peers
```

```
cilium bgp routes
```

Cilium BGP を使用してタイプ `LoadBalancer` のサービスの IP をアドバタイズする場合は、`CiliumLoadBalancerIPPool` と Service の両方に同じラベルが必要です。ラベルは、`CiliumBGPAdvertisement` のセレクタで使用する必要があります。次に例を示します。Cilium BGP を使用して LoadBalancer タイプのサービスの IP をアドバタイズしている場合、Cilium エージェントの再起動中に BGP ルートが中断される可能性があります。詳細については、「Cilium documentation」の「Failure Scenarios」を参照してください。

サービス

```
kind: Service
apiVersion: v1
metadata:
  name: guestbook
  labels:
```

```
    app: guestbook
spec:
  ports:
  - port: 3000
    targetPort: http-server
  selector:
    app: guestbook
  type: LoadBalancer
```

CiliumLoadBalancerIPPool

```
apiVersion: cilium.io/v2alpha1
kind: CiliumLoadBalancerIPPool
metadata:
  name: guestbook-pool
  labels:
    app: guestbook
spec:
  blocks:
  - cidr: <CIDR to advertise>
  serviceSelector:
    matchExpressions:
      - { key: app, operator: In, values: [ guestbook ] }
```

CiliumBGPAdvertisement

```
apiVersion: cilium.io/v2alpha1
kind: CiliumBGPAdvertisement
metadata:
  name: bgp-advertisements-guestbook
  labels:
    advertise: bgp
spec:
  advertisements:
    - advertisementType: "Service"
      service:
        addresses:
          - ExternalIP
          - LoadBalancerIP
      selector:
        matchExpressions:
          - { key: app, operator: In, values: [ guestbook ] }
```

Calico のトラブルシューティング

ハイブリッドノードで Calico の実行に問題がある場合は、Calico ドキュメントの「トラブルシューティング手順」を参照してください。以下のセクションでは、Calico をハイブリッドノードにデプロイする際に特有の問題について説明します。

次の表は、Calico コンポーネントと、それらがデフォルトでノードネットワークまたはポッドネットワークで実行されているかどうかをまとめたものです。ポッドの送信トラフィックに NAT を使用するように Calico を設定した場合、オンプレミスネットワークはオンプレミスノード CIDR にトラフィックをルーティングするように設定する必要があり、VPC ルーティングテーブルはトランジットゲートウェイ (TGW) または仮想プライベートゲートウェイ (VGW) をターゲットとするオンプレミスノード CIDR のルートで設定する必要があります。ポッドの送信トラフィックに NAT を使用するように Calico を設定していない場合は、オンプレミスネットワークがオンプレミスポッド CIDR にトラフィックをルーティングするように設定し、VPC ルーティングテーブルをトランジットゲートウェイ (TGW) または仮想プライベートゲートウェイ (VGW) をターゲットとするオンプレミスポッド CIDR のルートで設定する必要があります。

コンポーネント	Network
Calico API サーバー	ノード
Kubernetes 用の Calico コントローラー	ポッド
Calico ノードエージェント	ノード
Calico typha	ノード
Calico CSI ノードドライバー	ポッド
Calico 演算子	ノード

Calico リソースは、遮断されたノードでスケジュールまたは実行されています

DaemonSet として実行されない Calico リソースには、デフォルトで柔軟な許容範囲があり、ポッドのスケジュールや実行の準備が整っていない遮断されたノードでスケジュールできます。以下を含めるようにオペレータのインストールを変更することで、DaemonSet Calico 以外のリソースの許容範囲を厳しくすることができます。

```
installation:
```

```
...
controlPlaneTolerations:
- effect: NoExecute
  key: node.kubernetes.io/unreachable
  operator: Exists
  tolerationSeconds: 300
- effect: NoExecute
  key: node.kubernetes.io/not-ready
  operator: Exists
  tolerationSeconds: 300
calicoKubeControllersDeployment:
  spec:
    template:
      spec:
        tolerations:
        - effect: NoExecute
          key: node.kubernetes.io/unreachable
          operator: Exists
          tolerationSeconds: 300
        - effect: NoExecute
          key: node.kubernetes.io/not-ready
          operator: Exists
          tolerationSeconds: 300
typhaDeployment:
  spec:
    template:
      spec:
        tolerations:
        - effect: NoExecute
          key: node.kubernetes.io/unreachable
          operator: Exists
          tolerationSeconds: 300
        - effect: NoExecute
          key: node.kubernetes.io/not-ready
          operator: Exists
          tolerationSeconds: 300
```

認証情報のトラブルシューティング

AWS SSM ハイブリッドアクティベーションと AWS IAM Roles Anywhere の両方で、ハイブリッドノードから次のコマンドを実行して、ハイブリッドノード IAM ロールの認証情報がハイブリッドノードで正しく設定されていることを検証できます。ノード名とハイブリッドノード IAM ロール名が想定どおりであることを確認します。

```
sudo aws sts get-caller-identity
```

```
{
    "UserId": "ABCDEFGHIJKLM12345678910:<node-name>",
    "Account": "<aws-account-id>",
    "Arn": "arn:aws:sts::<aws-account-id>:assumed-role/<hybrid-nodes-iam-role/<node-
name>"
}
```

AWSSystems Manager (SSM) のトラブルシューティング

ハイブリッドノードの認証情報に AWS SSM ハイブリッドアクティベーションを使用している場合は、nodeadm によってハイブリッドノードにインストールされている次の SSM ディレクトリとアーティファクトに注意してください。SSM エージェントの詳細については、「AWS Systems Manager ユーザーガイド」の「SSM Agent の使用」を参照してください。

説明	場所
SSM エージェント	Ubuntu - `/snap/amazon-ssm-agent/current/amazon-ssm-agent` RHEL & AL2023 - `/usr/bin/amazon-ssm-agent`
SSM エージェントログ	`/var/log/amazon/ssm`
AWS 認証情報	`/root/.aws/credentials`
SSM Setup CLI	`/opt/ssm/ssm-setup-cli`

SSM エージェントを再起動する

SSM エージェントを再起動することで解決できる問題もあります。以下のコマンドを使用して再起動できます。

AL2023 およびその他のオペレーティングシステム

```
systemctl restart amazon-ssm-agent
```

Ubuntu -

```
systemctl restart snap.amazon-ssm-agent.amazon-ssm-agent
```

SSM エンドポイントへの接続を確認する

ハイブリッドノードから SSM エンドポイントに接続できることを確認します。SSM エンドポイントのリストについては、「AWS Systems Manager endpoints and quotas」を参照してください。以下のコマンドの us-west-2 を、AWS SSM ハイブリッドアクティベーションの AWS リージョンに置き換えます。

```
ping ssm.us-west-2.amazonaws.com
```

登録された SSM インスタンスの接続ステータスを表示する

次の AWS CLI コマンドを使用して、SSM ハイブリッドアクティベーションに登録されているインスタンスの接続ステータスを確認できます。マシン ID をインスタンスのマシン ID に置き換えます。

```
aws ssm get-connection-status --target mi-012345678abcdefgh
```

SSM セットアップ CLI チェックサムの不一致

nodeadm install の実行時に、ssm-setup-cli チェックサムの不一致に問題がある場合は、ホストに古い既存の SSM インストールがないことを確認する必要があります。ホストに古い SSM インストールがある場合は、それらを削除して nodeadm install を再実行し、問題を解決します。

```
Failed to perform agent-installation/on-prem registration: error while verifying
 installed ssm-setup-cli checksum: checksum mismatch with latest ssm-setup-cli.
```

SSM **InvalidActivation**

インスタンスを AWS SSM に登録する際にエラーが表示された場合は、nodeConfig.yaml の region、activationCode、および activationId が正しいことを確認します。EKS クラスターの AWS リージョンは、SSM ハイブリッドアクティベーションのリージョンと一致する必要があります。これらの値が誤って設定されている場合は、次のようなエラーが表示されることがあります。

```
ERROR Registration failed due to error registering the instance with AWS SSM.
 InvalidActivation
```

SSM **ExpiredTokenException**: リクエストに含まれているセキュリティトークンが有効期限切れです

SSM エージェントが認証情報を更新できない場合は、「ExpiredTokenException」が表示されることがあります。このシナリオでは、ハイブリッドノードから SSM エンドポイントに接続できる場合、認証情報の更新を強制するために SSM エージェントを再起動する必要がある場合があります。

```
"msg":"Command failed","error":"operation error SSM: DescribeInstanceInformation, https
 response error StatusCode: 400, RequestID: eee03a9e-f7cc-470a-9647-73d47e4cf0be, api
 error ExpiredTokenException: The security token included in the request is expired"
```

register machine コマンドの実行中の SSM エラー

マシンを SSM に登録する際にエラーが発生した場合は、nodeadm install を再実行して、すべての SSM 依存関係が正しくインストールされていることを確認する必要があります。

```
"error":"running register machine command: , error: fork/exec /opt/aws/ssm-setup-cli:
 no such file or directory"
```

SSM **ActivationExpired**

nodeadm init の実行時に、アクティベーションの有効期限が切れたために SSM へのインスタンスの登録がエラーになった場合は、新しい SSM ハイブリッドアクティベーションを作成し、新しい SSM ハイブリッドアクティベーションの activationCode と activationId で nodeConfig.yaml を更新して、nodeadm init を再実行する必要があります。

```
"msg":"Command failed","error":"SSM activation expired. Please use a valid activation"
```

```
ERROR Registration failed due to error registering the instance with AWS SSM.
 ActivationExpired
```

SSM がキャッシュされた認証情報を更新できませんでした

キャッシュされた認証情報の更新に失敗した場合、/root/.aws/credentials ファイルはホストで削除されている可能性があります。まず、SSM ハイブリッドアクティベーションをチェックして、アクティブであること、およびアクティベーションを使用するようにハイブリッドノードが正しく設定されていることを確認します。/var/log/amazon/ssm で SSM エージェントログを確認し、SSM 側で問題を解決したら nodeadm init コマンドを再実行します。

```
"Command failed","error":"operation error SSM: DescribeInstanceInformation, get
 identity: get credentials: failed to refresh cached credentials"
```

SSM のクリーンアップ

ホストから SSM エージェントを削除するには、次のコマンドを実行します。

```
dnf remove -y amazon-ssm-agent
sudo apt remove --purge amazon-ssm-agent
snap remove amazon-ssm-agent
rm -rf /var/lib/amazon/ssm/Vault/Store/RegistrationKey
```

AWSIAM Roles Anywhere のトラブルシューティング

ハイブリッドノードの認証情報に AWS IAM Roles Anywhere を使用している場合は、nodeadm によってハイブリッドノードにインストールされている次のディレクトリとアーティファクトに注意してください。IAM Roles Anywhere のトラブルシューティングの詳細については、「AWS IAM Roles Anywhere User Guide」の「Troubleshooting AWS IAM Roles Anywhere identity and access」を参照してください。

説明	場所
IAM Roles Anywhere CLI	`/usr/local/bin/aws_signing_helper`
デフォルトの証明書の場所と名前	`/etc/iam/pki/server.pem`
デフォルトのキーの場所と名前	`/etc/iam/pki/server.key`

IAM Roles Anywhere がキャッシュされた認証情報の更新に失敗しました

キャッシュされた認証情報の更新に失敗した場合は、`/etc/aws/hybrid/config` の内容を確認し、nodeadm 設定で IAM Roles Anywhere が正しく設定されていることを確認します。`/etc/iam/pki` が存在することを確認します。各ノードには一意の証明書とキーが必要です。デフォルトでは、認証情報プロバイダーとして IAM Roles Anywhere を使用する場合、nodeadm は証明書の場所と名前に `/etc/iam/pki/server.pem` を使用し、プライベートキーに `/etc/iam/pki/server.key` を使用します。`sudo mkdir -p /etc/iam/pki` を使用してディレクトリに証明書とキーを配置する前に、ディレクトリの作成が必要になる場合があります。証明書の内容は、以下のコマンドで確認できます。

```
openssl x509 -text -noout -in server.pem
```

```
open /etc/iam/pki/server.pem: no such file or directory
could not parse PEM data
Command failed {"error": "... get identity: get credentials: failed to refresh cached
 credentials, process provider error: error in credential_process: exit status 1"}
```

の実行が許可されていない IAM Roles Anywhere**sts:AssumeRole**

kubelet ログで、IAM Roles Anywhere の使用時に sts:AssumeRole オペレーションに対するアクセス拒否の問題が表示された場合は、ハイブリッドノード IAM ロールの信頼ポリシーをチェックして、IAM Roles Anywhere サービスプリンシパルがハイブリッドノード IAM ロールを引き受けることができることを確認します。さらに、ハイブリッドノード IAM ロールの信頼ポリシーでトラストアンカー ARN が正しく設定されていること、およびハイブリッドノード IAM ロールが IAM Roles Anywhere プロファイルに追加されていることを確認します。

```
could not get token: AccessDenied: User: ... is not authorized to perform:
 sts:AssumeRole on resource: ...
```

を設定することが許可されていない IAM Roles Anywhere**roleSessionName**

kubelet ログで、roleSessionName の設定にアクセス拒否の問題が発生した場合は、IAM Roles Anywhere プロファイルで acceptRoleSessionName を true に設定していることを確認します。

```
AccessDeniedException: Not authorized to set roleSessionName
```

オペレーティングシステムのトラブルシューティング

RHEL

使用権限管理者またはサブスクリプションマネージャーの登録失敗

nodeadm install を実行中で、使用権限登録の問題によりハイブリッドノードの依存関係のインストールに失敗した場合、ホストに Red Hat ユーザー名とパスワードを適切に設定していることを確認してください。

```
This system is not registered with an entitlement server
```

Ubuntu

GLIBC が見つかりません

オペレーティングシステムに Ubuntu を使用し、ハイブリッドノードを持つ認証情報プロバイダーに IAM Roles Anywhere を使用していて、GLIBC が見つからない問題がある場合は、その依存関係を手動でインストールすることで問題を解決できます。

```
GLIBC_2.32 not found (required by /usr/local/bin/aws_signing_helper)
```

次のコマンドを実行して依存関係をインストールします。

```
ldd --version
sudo apt update && apt install libc6
sudo apt install glibc-source
```

Bottlerocket

Bottlerocket 管理コンテナを有効にしている場合、SSH を使用してこれにアクセスし、昇格された権限を使って高度なデバッグとトラブルシューティングを行うことができます。以下のセクションでは、Bottlerocket ホストのコンテキストで実行する必要があるコマンドを記載しています。管理コンテナにアクセスすると、`sheltie` を実行すれば Bottlerocket ホストで完全なルートシェルを取得することができます。

```
sheltie
```

以下のセクションにあるコマンドは、各コマンドの前に `sudo chroot /.bottlerocket/rootfs` を付けることで管理者コンテナシェルから実行することもできます。

```
sudo chroot /.bottlerocket/rootfs <command>
```

ログ収集に logdog を使用する

Bottlerocket には、トラブルシューティングを目的に、ログとシステム情報とを効率的に収集する `logdog` ユーティリティがあります。

```
logdog
```

logdog ユーティリティは、Bottlerocket ホストのさまざまな場所からログを収集し、それらを tarball と組み合わせます。デフォルトでは、tarball は /var/log/support/bottlerocket-logs.tar.gz で作成され、/.bottlerocket/support/bottlerocket-logs.tar.gz にあるホストコンテナからアクセスすることができます。

journalctl を使用したシステムログへのアクセス

次のコマンドを使用することで、kubelet、containerd などのさまざまなシステムサービスのステータスをチェックしたりログを表示したりできます。-f フラグは、ログをリアルタイムでフォローします。

kubelet サービスのステータスを確認し kubelet ログを取得するときは、以下を実行します。

```
systemctl status kubelet
journalctl -u kubelet -f
```

containerd サービスのステータスを確認し、オーケストレーションされた containerd インスタンスのログを取得するときは、以下を実行します。

```
systemctl status containerd
journalctl -u containerd -f
```

host-containerd サービスのステータスを確認し、ホスト containerd インスタンスのログを取得するときは、以下を実行します。

```
systemctl status host-containerd
journalctl -u host-containerd -f
```

ブートストラップコンテナとホストコンテナのログを取得するときは、以下を実行します。

```
journalctl _COMM=host-ctr -f
```

Amazon EKS Hybrid Nodes ゲートウェイ

Amazon EKS Hybrid Nodes ゲートウェイは、Amazon EKS クラスター VPC と EKS Hybrid Nodes で実行されている Kubernetes ポッド間のネットワークを自動化します。ゲートウェイにより、オンプレミスのポッドネットワークを VPC からルーティング可能にしたり、ネットワークインフラストラクチャの変更を調整したりする必要がなくなります。VPC の EC2 ベースのゲートウェイノードとオンプレミス環境の Cilium マネージドハイブリッドノードの間に VXLAN トンネルを作成し、トラ

フィックが正しいゲートウェイインスタンスに到達するように VPC ルートテーブルエントリを自動的に維持します。

ユースケース

Hybrid Nodes ゲートウェイは、VPC とオンプレミス環境間の以下のトラフィックフローを有効にします。

- コントロールプレーンからウェブフックへの通信 — Kubernetes API サーバーは、ハイブリッドノードで実行されているウェブフックエンドポイントに到達できます。ゲートウェイがない場合、ハイブリッドノードのウェブフックは、オンプレミス環境でポッド CIDR をルーティング可能にしない限り、コントロールプレーンからアクセスできません。
- クラウドとオンプレミス間のポッド間トラフィック — VPC の EC2 ノードで実行されているポッドは、ハイブリッドノードで実行されているポッドと直接通信できます。その逆も同様です。
- ハイブリッドポッドへの AWS サービス接続 — Application Load Balancer、Network Load Balancer、Amazon Managed Service for Prometheus などの AWS サービスは、ハイブリッドノードで実行されているポッドに到達できます。

アーキテクチャ

2 つのゲートウェイポッドは、ラベル付き EC2 ノードでデプロイとして実行されます。Kubernetes リースベースのリーダー選出により、アクティブなポッドが決まります。どちらのポッドも起動時に VXLAN インターフェイスを作成し、`CiliumNode` オブジェクトを監視するノード調整を実行するため、スタンバイは常にフェイルオーバー時に 3 ~ 5 秒以内にトラフィックを転送する準備ができています。リーダー固有のアクション (VPC ルートテーブルの更新と `CiliumVTEPConfig` 管理) のみが、リーダーが変わったときに転送されます。

仕組み

Hybrid Nodes ゲートウェイは、以下の 4 つのメカニズムを使用して接続を有効にします。

VXLAN トンネリング — ゲートウェイは、UDP ポート 8472 (Cilium のデフォルト) に VNI 2 を持つ VXLAN インターフェイス (`hybrid_vxlan0`) を作成します。VXLAN インターフェイスで FDB エントリ、ARP エントリ、ルートをプログラミングすることで、各ハイブリッドノードへのトンネルを確立します。ノードコントローラーは `CiliumNode` オブジェクトを監視し、ハイブリッドノードがクラスターに参加またはクラスターを離れるとトンネルを自動的に追加または削除します。

VPC ルートテーブル管理 — ゲートウェイがリーダーになると、指定された VPC ルートテーブルでルートを作成または置き換えます。各ルートはハイブリッドポッド CIDR をリーダーのプライマリ

ENI を指すため、ハイブリッドポッド宛ての VPC トラフィックはアクティブなゲートウェイインスタンスに転送されます。

Cilium VTEP 統合 — ゲートウェイは、ハイブリッドノード上の Cilium エージェントに VPC バウンドトラフィックの送信先を指定する `CiliumVTEPConfig` カスタムリソースを作成します。設定には、トンネルエンドポイントとしてリーダーのノード IP と VXLAN インターフェイス MAC アドレスが含まれます。ハイブリッドポッドが VPC アドレスにトラフィックを送信すると、Cilium はそのトラフィックを VXLAN パケットにカプセル化し、ゲートウェイに送信します。

リーダー選出 — ゲートウェイはアクティブスタンバイモデルで Kubernetes リースベースのリーダー選出を使用します。2 つのゲートウェイポッドは、ポッドアンチアフィニティによって強制される個別のノードで実行されます。どちらのポッドも起動時に VXLAN インターフェイスを作成し、すべてのハイブリッドノードの VTEP エントリを維持するノード調整を実行します。リーダーが VPC ルートテーブルの更新と Cilium VTEP 設定を実行します。リーダーが失敗した場合、スタンバイはリースの有効期限を検出し、リースを取得し、リーダーセットアップシーケンスを実行します。予想されるフェイルオーバー時間は、約 3～5 秒です。

デプロイモデル

Hybrid Nodes ゲートウェイは VPC の EC2 インスタンスで実行され、Helm チャートを使用してデプロイされます。ゲートウェイは、以下のデプロイターゲットをサポートしています。

- EKS Auto Mode — 正しいラベル、テイント、送信元/送信先チェック設定を使用してゲートウェイノードを自動的にプロビジョニングする `NodePool` と `NodeClass` を作成します。これは推奨設定です。
- EKS マネージド型ノードグループ — ゲートウェイラベル、テイント、送信元/送信先チェックを無効にした専用のマネージド型ノードグループを作成し、Helm 値で `autoMode.enabled=false` を設定します。

すべてのデプロイターゲットでは、高可用性のために少なくとも 2 つのノードが推奨されます。詳細については、「the section called “はじめに”」を参照してください。

料金

Amazon EKS Hybrid Nodes ゲートウェイには追加料金はかかりませんが、該当する場合は EC2 インスタンスや EKS Auto Mode 管理料金など、ゲートウェイを実行するためのインフラストラクチャコストが課金されます。詳細については「Amazon EKS の料金」を参照してください。

リージョンの可用性

Amazon EKS Hybrid Nodes ゲートウェイは、中国リージョンを除く、EKS Hybrid Nodes が利用可能なすべての AWS リージョンで使用できます。現在サポートされているリージョンのリストについては、「the section called “ハイブリッドノード”」を参照してください。

オープンソース

Amazon EKS Hybrid Nodes ゲートウェイコードベースはオープンソースです。ソースコードを表示したり、問題を報告したり、GitHub リポジトリに貢献したりできます。

制約事項と考慮事項

Hybrid Nodes ゲートウェイをデプロイする前に、以下の点を考慮してください。

- トラフィックの暗号化なし — ゲートウェイによって作成された VXLAN トンネルはトラフィックを暗号化しません。VPC とオンプレミス環境間の転送中に暗号化が必要な場合は、MACsec を使用した AWS Direct Connect や VPN 接続などの暗号化されたトランスポートを使用します。詳細については、「Amazon EKS とは」を参照してください。
- 単一クラスター — 各ゲートウェイデプロイは、単一の EKS クラスターを提供します。ハイブリッドノードを持つクラスターが複数ある場合は、クラスターごとに個別のゲートウェイをデプロイします。
- Cilium VTEP が必要 — ゲートウェイでは、ハイブリッドノードで VTEP サポートが有効になっている Cilium CNI の EKS バージョンが必要です。他の CNI プラグインはサポートされていません。

次のステップ

- the section called “はじめに”
- the section called “設定”
- the section called “オペレーション”
- the section called “トラブルシューティング”

EKS Hybrid Nodes ゲートウェイの使用を開始する

このページでは、Amazon EKS Hybrid Nodes ゲートウェイの前提条件、環境の準備、インストール、検証、削除について説明します。ゲートウェイとそのアーキテクチャの概要については、「the section called “Hybrid Nodes ゲートウェイ”」を参照してください。

前提条件

Hybrid Nodes ゲートウェイをインストールする前に、環境が以下の要件を満たしていることを確認してください。

- Cilium CNI と VTEP をサポートする EKS クラスター — EKS クラスターはハイブリッドノードの CNI として Cilium の EKS バージョンを使用し、Cilium VTEP を有効にする必要があります。詳細については、「the section called “ゲートウェイの CNI を設定する”」を参照してください。
- クラウドノード上の AWS VPC CNI — クラスター内のゲートウェイノードと他のクラウドノードは AWS VPC CNI を使用する必要があります。ゲートウェイは VPC ネイティブルーティングを使用して、VPC と VXLAN トンネル間でトラフィックを転送します。
- ハイブリッド接続 — VPC とオンプレミス環境間のプライベート接続が必要です。AWS Direct Connect、AWS Site-to-Site VPN、または独自の VPN ソリューションを使用できます。詳細については、「the section called “ネットワークを準備する”」を参照してください。
- VXLAN トラフィック許可 — ゲートウェイ EC2 インスタンスにアタッチされたセキュリティグループは、ポート 8472 でのインバウンド UDP トラフィックとアウトバウンド UDP トラフィックを許可する必要があります。ハイブリッドノード側では、オンプレミスファイアウォールルールでゲートウェイノード IP アドレスとの間の UDP ポート 8472 トラフィックも許可する必要があります。
- ルートテーブル管理の IAM アクセス許可 — ゲートウェイでは、VPC ルートテーブルを管理するために以下の EC2 アクションが必要です。
  - `ec2:DescribeRouteTables`
  - `ec2:CreateRoute`
  - `ec2:ReplaceRoute`
  - `ec2:DescribeInstances`

  これらのアクセス許可は、以下のいずれかの方法で付与できます。

  EKS Pod Identity (recommended)

  EKS Pod Identity を使用して、ゲートウェイポッドのサービスアカウントにのみアクセス許可を付与します。

a. EKS Pod Identity Agent アドオンがまだインストールされていない場合は、クラスターにインストールします。

```
aws eks create-addon \
  --cluster-name CLUSTER_NAME \
  --addon-name eks-pod-identity-agent
```

b. 必要な EC2 アクセス許可と EKS Pod Identity の信頼ポリシーを持つ IAM ロールを作成します。

```
cat > gateway-trust-policy.json << 'EOF'
{
  "Version": "2012-10-17"        ,
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": { "Service": "pods.eks.amazonaws.com" },
      "Action": ["sts:AssumeRole", "sts:TagSession"]
    }
  ]
}
EOF

aws iam create-role \
  --role-name EKSHybridNodesGatewayRole \
  --assume-role-policy-document file://gateway-trust-policy.json

aws iam put-role-policy \
  --role-name EKSHybridNodesGatewayRole \
  --policy-name HybridNodesGatewayRouteTable \
  --policy-document '{
    "Version": "2012-10-17"        ,
    "Statement": [{
      "Effect": "Allow",
      "Action": [
        "ec2:DescribeRouteTables",
        "ec2:CreateRoute",
        "ec2:ReplaceRoute",
        "ec2:DescribeInstances"
      ],
      "Resource": "*"
    }]
```

```
  }'
```

c. IAM ロールをゲートウェイサービスアカウントにリンクする Pod Identity 関連付けを作成します。

```
aws eks create-pod-identity-association \
  --cluster-name CLUSTER_NAME \
  --namespace eks-hybrid-nodes-gateway \
  --service-account eks-hybrid-nodes-gateway \
  --role-arn arn:aws:iam::ACCOUNT_ID:role/EKSHybridNodesGatewayRole
```

Node IAM role

ゲートウェイノードのインスタンスプロファイルに関連付けられた IAM ロールに必要なアクセス許可を持つインラインポリシーまたはマネージドポリシーをアタッチします。

```
aws iam put-role-policy \
  --role-name NODE_ROLE_NAME \
  --policy-name HybridNodesGatewayRouteTable \
  --policy-document '{
    "Version": "2012-10-17"        ,
    "Statement": [{
      "Effect": "Allow",
      "Action": [
        "ec2:DescribeRouteTables",
        "ec2:CreateRoute",
        "ec2:ReplaceRoute",
        "ec2:DescribeInstances"
      ],
      "Resource": "*"
    }]
  }'
```

ゲートウェイノード上のすべてのポッドには、これらのアクセス許可があります。

- EKS Auto Mode (ゲートウェイノードに Auto Mode を使用している場合) — EKS Auto Mode を使用してゲートウェイノードをプロビジョニングする場合は、EKS クラスターで Auto Mode を有効にする必要があります。詳細については、「既存の EKS クラスターで EKS 自動モードを有効にする」を参照してください。

ゲートウェイノードを準備する

ゲートウェイには、高可用性のために少なくとも 2 つの EC2 ノードが必要です。ゲートウェイでは、以下の 2 つのノード設定オプションがサポートされています。

- the section called “EKS Auto Mode (推奨)” — ノードは NodePool と NodeClass を使用して自動的にプロビジョニングされます。送信元/送信先チェック、ラベル、テイントはすべて宣言的に設定されます。
- the section called “マネージドノードグループ” — マネージド型ノードグループまたはセルフマネージド型ノードを使用してノードをプロビジョニングします。ラベルはマネージド型ノードグループ API を使用して設定でき、送信元/送信先チェックはユーザーデータを含むカスタム起動テンプレートを使用して無効にできます。

EKS Auto Mode (推奨)

EKS Auto Mode を使用する場合は、Helm チャートをインストールする前に NodePool と NodeClass を作成する必要があります。NodePool は、正しいラベル、テイント、送信元/送信先チェック設定を使用して EC2 インスタンスをプロビジョニングします。ノードを手動でプロビジョニングまたは設定する必要はありません。

以下のリソースをクラスターに適用し、プレースホルダー値を置き換えます。

- *YOUR_NODE_ROLE* — EKS Auto Mode がノードをプロビジョニングするために使用するノード IAM ロールの名前。これは、クラスターで Auto Mode を有効にしたときに設定した (または EKS が作成した) ロールです。詳細については、「Amazon EKS 自動モードノードの IAM ロール」を参照してください。
- *YOUR_CLUSTER_NAME* — EKS クラスター名。
- *SUBNET_ID_1*、*SUBNET_ID_2* — ゲートウェイノードがプロビジョニングされる異なるアベイラビリティーゾーンのサブネット ID。

```
apiVersion: eks.amazonaws.com/v1
kind: NodeClass
metadata:
  name: hybrid-gateway
spec:
  advancedNetworking:
    sourceDestCheck: DisabledPrimaryENI
  role: YOUR_NODE_ROLE
```

```
  securityGroupSelectorTerms:
    - tags:
        aws:eks:cluster-name: YOUR_CLUSTER_NAME
  subnetSelectorTerms:
    - id: SUBNET_ID_1
    - id: SUBNET_ID_2
---
apiVersion: karpenter.sh/v1
kind: NodePool
metadata:
  name: hybrid-gateway
spec:
  template:
    metadata:
      labels:
        hybrid-gateway-node: "true"
    spec:
      expireAfter: 336h
      nodeClassRef:
        group: eks.amazonaws.com
        kind: NodeClass
        name: hybrid-gateway
      requirements:
        - key: karpenter.sh/capacity-type
          operator: In
          values:
            - on-demand
        - key: eks.amazonaws.com/instance-category
          operator: In
          values:
            - c
            - m
            - r
        - key: eks.amazonaws.com/instance-generation
          operator: Gt
          values:
            - "4"
        - key: kubernetes.io/arch
          operator: In
          values:
            - amd64
        - key: kubernetes.io/os
          operator: In
          values:
```

```
            - linux
      taints:
        - key: hybrid-gateway-node
          effect: NoSchedule
      terminationGracePeriod: 24h0m0s
  disruption:
    budgets:
      - nodes: 10%
    consolidateAfter: 30s
    consolidationPolicy: WhenEmptyOrUnderutilized
```

この設定のキーフィールド:

- advancedNetworking.sourceDestCheck: DisabledPrimaryENI — ノードのプライマリ ENI の EC2 送信元/送信先チェックを無効にして、ゲートウェイがそれ自体にアドレス指定されていないトラフィックを転送できるようにします。
- taints — hybrid-gateway-node: NoSchedule テイントは、これらのノードで許容範囲スケジュールが一致するゲートウェイポッドのみを確保します。
- labels — hybrid-gateway-node: "true" ラベルは、Helm チャートのノードセレクタによって使用され、ゲートウェイポッドをこれらのノードにターゲットにします。
- nodeClassRef — NodePool を、送信元/送信先チェック設定と NodeClass にリンクします。

マネージドノードグループ

マネージド型ノードグループを使用する場合は、必要なラベル、テイント、および起動時に送信元/送信先チェックを無効にするカスタム起動テンプレートを使用して、ゲートウェイ専用のノードグループを作成します。

ステップ 1: 起動テンプレートを作成する

インスタンスの起動時にプライマリ ENI の送信元/送信先チェックを無効にするユーザーデータを含む起動テンプレートを作成します。

```
# Create the launch template with user data to disable source/dest check
USERDATA=$(cat <<'SCRIPT' | base64 -w 0
MIME-Version: 1.0
Content-Type: multipart/mixed; boundary="==BOUNDARY=="

--==BOUNDARY==
Content-Type: text/x-shellscript; charset="us-ascii"
```

```
#!/bin/bash
TOKEN=$(curl -s -X PUT "http://169.254.169.254/latest/api/token" \
  -H "X-aws-ec2-metadata-token-ttl-seconds: 60")
MAC=$(curl -s -H "X-aws-ec2-metadata-token: $TOKEN" \
  http://169.254.169.254/latest/meta-data/mac)
ENI_ID=$(curl -s -H "X-aws-ec2-metadata-token: $TOKEN" \
  "http://169.254.169.254/latest/meta-data/network/interfaces/macs/${MAC}/interface-
id")
REGION=$(curl -s -H "X-aws-ec2-metadata-token: $TOKEN" \
  http://169.254.169.254/latest/meta-data/placement/region)

aws ec2 modify-network-interface-attribute \
  --network-interface-id "$ENI_ID" \
  --no-source-dest-check \
  --region "$REGION"

--==BOUNDARY==--
SCRIPT
)

aws ec2 create-launch-template \
  --launch-template-name YOUR_CLUSTER_NAME-gateway-lt \
  --launch-template-data "{\"UserData\":\"${USERDATA}\",\"MetadataOptions\":
{\"HttpTokens\":\"required\",\"HttpPutResponseHopLimit\":2}}"
```

> **Note**
>
> ノードの IAM ロールには、ユーザーデータスクリプトを成功させるための `ec2:ModifyNetworkInterfaceAttribute` アクセス許可が必要です。MIME マルチパート形式により、ユーザーデータは EKS ブートストラップスクリプトと一緒に実行されます。

ステップ 2: マネージド型ノードグループを作成する

ステップ 1 のゲートウェイラベル、テイント、起動テンプレートを使用して、専用のマネージド型ノードグループを作成します。

```
aws eks create-nodegroup \
  --cluster-name YOUR_CLUSTER_NAME \
  --nodegroup-name YOUR_CLUSTER_NAME-gateway-nodes \
```

```
  --subnets SUBNET_ID_1 SUBNET_ID_2 \
  --node-role YOUR_NODE_ROLE_ARN \
  --instance-types INSTANCE_TYPE \
  --ami-type AL2023_x86_64_STANDARD \
  --scaling-config desiredSize=2,maxSize=2,minSize=2 \
  --labels hybrid-gateway-node=true \
  --taints "key=hybrid-gateway-node,effect=NO_SCHEDULE" \
  --launch-template "name=YOUR_CLUSTER_NAME-gateway-lt,version=1"
```

これにより、以下の設定を持つ 2 ノードのマネージド型ノードグループが作成されます。

- `--labels` は、Helm チャートのノードセレクタがこれらのノードをターゲットにするように `hybrid-gateway-node=true` を設定します。
- `--taints` は NoSchedule テイントを追加して、これらのノードで許容範囲スケジュールが一致するゲートウェイポッドのみを追加します。
- `--launch-template` は、起動時に送信元/送信先チェックを無効にする起動テンプレートをアタッチします。

高可用性を実現するために、異なるアベイラビリティーゾーンのサブネットを使用します。

Helm を使用してインストールする

EKS 自動モード

以下のコマンドを実行して、EKS Auto Mode でゲートウェイをインストールします。

```
helm install eks-hybrid-nodes-gateway \
  oci://public.ecr.aws/eks/eks-hybrid-nodes-gateway \
  --version 1.0.0 \
  --namespace eks-hybrid-nodes-gateway \
  --create-namespace \
  --set vpcCIDR=VPC_CIDR \
  --set podCIDRs=POD_CIDRS \
  --set routeTableIDs=ROUTE_TABLE_IDS
```

マネージド型ノードグループまたはセルフマネージド型ノード

マネージド型ノードグループまたはセルフマネージド型ノードの場合は、`autoMode.enabled=false` を設定します。

```
helm install eks-hybrid-nodes-gateway \
  oci://public.ecr.aws/eks/eks-hybrid-nodes-gateway \
  --version 1.0.0 \
  --namespace eks-hybrid-nodes-gateway \
  --create-namespace \
  --set autoMode.enabled=false \
  --set vpcCIDR=VPC_CIDR \
  --set podCIDRs=POD_CIDRS \
  --set routeTableIDs=ROUTE_TABLE_IDS
```

必要な Helm 値

すべてのインストールには、以下の値が必要です。

値	説明
vpcCIDR	EKS クラスター VPC の CIDR ブロック (例: 10.0.0.0/16)。ハイブリッドノードがゲートウェイを介して VPC バウンドトラフィックをルーティングするように Cilium VTEP 設定に使用されます。
podCIDRs	ハイブリッドノードで Cilium が使用するポッド CIDR のカンマ区切りリスト (例: 10.100.0.0/16,10.101.0.0/16)。ゲートウェイは、これらの CIDR の VPC ルートテーブルエントリと VXLAN ルートを作成します。
routeTableIDs	プログラムする VPC ルートテーブル ID のカンマ区切りリスト (例: rtb-0abc1234def567890,rtb-0fed9876cba543210)。ゲートウェイは、ハイブリッドポッド CIDR がアクティブなゲートウェイインスタンスを指すように、ルートをこれらのテーブルに作成します。

設定可能な値の完全なリストについては、「the section called “設定”」を参照してください。

インストールの検証

ゲートウェイをインストールしたら、ゲートウェイが実行中で正常であることを確認します。

ポッドのステータスを確認する

2 つのゲートウェイポッドが実行されていることを確認します。

```
kubectl get pods -n eks-hybrid-nodes-gateway
```

次のような出力が表示されます:

```
NAME                                          READY   STATUS    RESTARTS   AGE
eks-hybrid-nodes-gateway-5d4f6a7b8c-abc12   1/1     Running   0          2m
eks-hybrid-nodes-gateway-5d4f6a7b8c-def34   1/1     Running   0          2m
```

リーダー選出を確認する

1 つのポッドがリーダー選出リースを取得したことを確認します。

```
kubectl get lease -n eks-hybrid-nodes-gateway
```

出力には、HOLDER 列に現在のリーダーが表示されます。

```
NAME                     HOLDER                                        AGE
hybrid-gateway-leader    eks-hybrid-nodes-gateway-5d4f6a7b8c-abc12   2m
```

ヘルスエンドポイントを確認する

ポート転送を使用して、ヘルスエンドポイントがリーダーポッドで応答していることを確認します。

```
kubectl port-forward -n eks-hybrid-nodes-gateway LEADER_POD_NAME 8088:8088 &
curl -s http://localhost:8088/healthz
```

正常なゲートウェイは HTTP 200 レスポンスを返します。

VPC ルートテーブルエントリを検証する

Amazon VPC コンソールまたは AWS CLI を使用して、VPC ルートテーブルに、リーダーゲートウェイインスタンスの ENI を指すハイブリッドポッド CIDR のエントリが含まれていることを確認します。

```
aws ec2 describe-route-tables \
  --route-table-ids ROUTE_TABLE_ID \
  --query "RouteTables[].Routes[?DestinationCidrBlock=='POD_CIDR']"
```

各ハイブリッドポッド CIDR には、リーダーインスタンスのプライマリ ENI に NetworkInterfaceId が設定されたルートが必要です。

アンインストール

Hybrid Nodes ゲートウェイを削除するには、以下を実行します。

```
helm uninstall eks-hybrid-nodes-gateway --namespace eks-hybrid-nodes-gateway
```

> **Note**
>
> Helm チャートをアンインストールしても、ゲートウェイによって作成された VPC ルートテーブルエントリは自動的に削除されません。アンインストール後、ハイブリッドポッド CIDR のルートを VPC ルートテーブルから手動で削除し、ゲートウェイが実行されなくなったインスタンスにトラフィックがルーティングされないようにします。AWS CLI を使用してルートを削除できます。
>
> ```
> aws ec2 delete-route \
>   --route-table-id ROUTE_TABLE_ID \
>   --destination-cidr-block POD_CIDR
> ```

次のステップ

- the section called “設定” — Helm 値、CLI フラグ、リーダー選出パラメータをカスタマイズします。
- the section called “オペレーション” — ゲートウェイをモニタリングし、フェイルオーバーの動作を理解し、スケーリングを計画します。
- the section called “トラブルシューティング” — 一般的な問題を診断して解決します。

ハイブリッドノードゲートウェイの CNI を設定する

Amazon EKS Hybrid Nodes ゲートウェイでは、VPC とハイブリッドノード間の VXLAN ベースの接続を有効にするために、特定の Cilium 設定が必要です。Cilium VTEP (VXLAN トンネルエンドポイント) サポートを有効にし、L7 プロキシを無効にする必要があります。

前提条件

- Cilium はハイブリッドノードに既にインストールされています。インストール手順については、「the section called “CNI を設定する”」を参照してください。

> **Note**
>
> Cilium VTEP は、以下の EKS Cilium の最小バージョンでのみサポートされています。
>
> - Cilium 1.17.13-1 以降
> - Cilium 1.18.8-1 以降
> - Cilium 1.19.2-1 以降
>
> 古いバージョンを実行している場合は、VTEP を有効にする前に Cilium をアップグレードします。更新する方法については、the section called “CNI を設定する” を参照してください。

VTEP を有効にして L7 プロキシを無効にする

Cilium バージョンを設定し、`helm upgrade` コマンドを実行して VTEP を有効にし、既存の Cilium インストールで L7 プロキシを無効にします。利用可能な最新のパッチバージョンは、Amazon ECR Public: eks/cilium/cilium で確認できます。

```
helm upgrade cilium oci://public.ecr.aws/eks/cilium/cilium \
  --version CILIUM_VERSION \
  --namespace kube-system \
  --reuse-values \
  --set vtep.enabled=true \
  --set l7Proxy=false
```

アップグレードが完了したら、Cilium DaemonSet を再起動して新しい設定を適用します。

```
kubectl rollout restart daemonset/cilium -n kube-system
kubectl rollout status daemonset/cilium -n kube-system
```

必須の設定

設定	値	説明
vtep.enabled	true	Cilium の VXLAN トンネルエンドポイントのサポートを有効にします。これにより、ハイブリッドノードの Cilium エージェントは VPC バウンドトラフィックをカプセル化し、ゲートウェイに送信できます。

設定	値	説明
l7Proxy	false	Cilium の L7 プロキシを無効にします。これは、VTEP が正しく機能するために必要です。L7 プロキシが有効になっている場合、VTEP トラフィックが傍受およびドロップされる可能性があります。

設定を確認する

Cilium をアップグレードしたら、VTEP が有効で、L7 プロキシが無効になっていることを確認します。

```
kubectl get configmap cilium-config -n kube-system -o yaml | grep -E "enable-vtep|
enable-l7-proxy"
```

次のような出力が表示されます:

```
enable-l7-proxy: "false"
enable-vtep: "true"
```

ハイブリッドポッドトラフィックの VPC CNI を設定する

デフォルトでは、AWS VPC CNI はノードから出るすべてのポッドトラフィックにソース NAT (SNAT) を適用します。ハイブリッドポッドエンドポイントを持つ Kubernetes ClusterIP サービスがクラウドノードから正しく動作するようにするには、ハイブリッドポッド CIDR を SNAT から除外します。これにより、kube-proxy DNAT'd トラフィックは VPC ルーティングを正しく使用できます。

aws-node DaemonSet の AWS_VPC_K8S_CNI_EXCLUDE_SNAT_CIDRS 環境変数をハイブリッドポッド CIDR (複数可) に設定します。

```
kubectl set env daemonset aws-node -n kube-system \
  AWS_VPC_K8S_CNI_EXCLUDE_SNAT_CIDRS=POD_CIDRS
```

値を実際のハイブリッドポッド CIDR に置き換えます (複数ある場合はカンマで区切ります)。

この設定では、ハイブリッドポッド CIDR を VPC CNI の ip ルールに追加し、ハイブリッドポッド宛てのトラフィックがゲートウェイ ENI への VPC ルートを含むメインルーティングテーブルを介してルーティングされるようにします。

> **Note**
>
> このステップは、クラウドポッドからハイブリッドポッドエンドポイントへの ClusterIP サービストラフィックに必要です。IP アドレスによるポッド間の直接接続は、この設定なしで機能します。

次のステップ

Cilium と VPC CNI を設定したら、Hybrid Nodes ゲートウェイのインストールに進みます。「the section called “はじめに”」を参照してください。

Amazon EKS Hybrid Nodes ゲートウェイ設定リファレンス

このページでは、Amazon EKS Hybrid Nodes ゲートウェイの設定可能なすべてのパラメータについて説明します。インストール手順については、「the section called “はじめに”」を参照してください。ゲートウェイの概要については、「the section called “Hybrid Nodes ゲートウェイ”」を参照してください。

Helm チャート値

以下の表に、`eks-hybrid-nodes-gateway` Helm チャート内のすべての値を一覧表示します。これらの値は、`helm install` または `helm upgrade` 中に `--set` フラグまたはカスタム `values.yaml` ファイルを使用して設定できます。

値	タイプ	デフォルト	必須	説明
`image.repository`	string	`public.ecr.aws/eks/eks-hybrid-nodes-gateway`	いいえ	ゲートウェイイメージのコンテナイメージリポジトリ。デフォルトは、パブリック Amazon ECR レジストリです。

値	タイプ	デフォルト	必須	説明
image.tag	string	Chart appVersion	いいえ	イメージタグ。デフォルトは、Chart.yaml で定義されている appVersion です。
image.pullPolicy	string	IfNotPresent	いいえ	イメージプルポリシー。有効な値は、Always、IfNotPresent 、Never です。
replicas	integer	2	いいえ	ゲートウェイポッドレプリカの数。本番環境で高可用性を実現するには、2 つのレプリカが推奨設定です。
nodeLabel	string	hybrid-gateway-node	いいえ	ゲートウェイノードの選択に使用されるノードラベルキー。ノードには、このラベルを "true" に設定する必要があります。
vpcCIDR	string	""	はい	Cilium VTEP 設定に使用される VPC CIDR ブロック。例えば、10.0.0.0/16 などです。
podCIDRs	string	""	はい	ハイブリッドノードで Cilium が使用するポッド CIDR のカンマ区切りリスト。例えば、10.85.0.0/16,10.86.0.0/16 などです。
routeTableIDs	string	""	はい	ハイブリッドポッドルートでプログラムする VPC ルートテーブル ID のカンマ区切りリスト。例えば、rtb-0123456789abcdef0,rtb-0123456789abcdef1 などです。

値	タイプ	デフォルト	必須	説明
autoMode.enabled	boolean	true	いいえ	EKS Auto Mode 統合を有効にします。true の場合、チャートは eks.amazonaws.com/compute-type: auto ノードセレクタを追加し、ダウンタイムのないローリング更新戦略を使用します。マネージド型ノードグループまたはセルフマネージド型ノードの場合は false に設定します。

インストールコマンドの例

EKS Auto Mode (デフォルト):

```
helm install eks-hybrid-nodes-gateway oci://public.ecr.aws/eks/eks-hybrid-nodes-gateway
 \
  --version 1.0.0 \
  --namespace eks-hybrid-nodes-gateway \
  --create-namespace \
  --set vpcCIDR=VPC_CIDR \
  --set podCIDRs=POD_CIDRS \
  --set routeTableIDs=ROUTE_TABLE_IDS
```

マネージド型ノードグループまたはセルフマネージド型ノード:

```
helm install eks-hybrid-nodes-gateway oci://public.ecr.aws/eks/eks-hybrid-nodes-gateway
 \
  --version 1.0.0 \
  --namespace eks-hybrid-nodes-gateway \
  --create-namespace \
  --set autoMode.enabled=false \
  --set vpcCIDR=VPC_CIDR \
  --set podCIDRs=POD_CIDRS \
  --set routeTableIDs=ROUTE_TABLE_IDS
```

Auto Mode とマネージド型ノードグループ設定の比較

`autoMode.enabled` Helm 値は、デプロイテンプレートの 2 つの動作を制御します。

行動	**`autoMode.enabled=true`** (デフォルト)	**`autoMode.enabled=false`**
ノードセレクタ	ゲートウェイノードラベルに加えて `eks.amazonaws.com/compute-type: auto` を追加します。	ゲートウェイノードラベル (`hybrid-gateway-node: "true"`) のみを使用します。
ローリング更新戦略	`maxSurge: 1`、`maxUnavailable: 0` — 古いポッドを終了する前に新しいポッドを起動することで、アップグレード中のダウンタイムをゼロにします。	`maxSurge: 0`、`maxUnavailable: 1` — ノード容量は事前プロビジョニングされているため、新しいポッドを開始する前に古いポッドを終了します。

EKS Auto Mode を使用する場合は、Helm チャートをインストールする前に `NodePool` と `NodeClass` を作成する必要があります。NodePool は、正しいラベル、テイント、送信元/送信先チェック設定を使用して EC2 インスタンスをプロビジョニングします。必要な YAML については、「the section called “はじめに”」を参照してください。マネージド型ノードグループまたはセルフマネージド型ノードを使用する場合は、チャートをインストールする前に、ノードを自分でプロビジョニングしてラベルを付ける必要があります。

すべてのデプロイターゲットについて、デプロイは `hostNetwork: true` を使用し、`NET_ADMIN` 機能を必要とします。ポッドアンチアフィニティにより、2 つのゲートウェイポッドが別々のノードで実行されます。

リーダー選出調整

ゲートウェイは、Kubernetes リースベースのリーダー選出を使用してアクティブスタンバイモデルを維持します。デフォルトのパラメータは、高速フェイルオーバー用に調整されます。

パラメータ	デフォルト	説明
`--leader-election-`	`3s`	非リーダーが、最後に確認されたリースの更新後、リースの取得を試みるまで待機する時間。値が低いほどリー

パラメータ	デフォルト	説明
lease-duration		ダーの障害をより迅速に検出できますが、一時的なネットワーク問題中に誤ったフェイルオーバーが発生するリスクが高まります。
--leader-election-renew-deadline	2s	アクティブなリーダーがリーダーシップを放棄する前にリースの更新に費やす最大時間。リース期間より短くする必要があります。
--leader-election-retry-period	1s	候補がリースの取得または更新を再試行する頻度。

デフォルト値では、アクティブゲートウェイが失敗すると予想されるフェイルオーバー時間は約 3 ~ 5 秒です。これには、スタンバイがリースの有効期限を検出し、リースを取得し、リーダーセットアップシーケンス (VPC ルートテーブルの更新と Cilium VTEP 設定) を実行するまでの時間が含まれます。

リーダー選出パラメータを調整するタイミング

- ゲートウェイポッドと Kubernetes API サーバー間の一時的なネットワークレイテンシーが原因でリーダーが頻繁に移行する場合は、リース期間を長くします。
- フェイルオーバー検出を高速化する必要があり、ゲートウェイノードと API サーバー間のネットワークの信頼性が高く、レイテンシーが低い場合は、リース期間を短縮します。
- 大規模なクラスターでリーダー選出による API サーバーの負荷を減らすには、再試行期間を長くします。

> **Note**
>
> `--leader-election-renew-deadline` は常に `--leader-election-lease-duration` より小さくなければなりません。更新期限がリース期間を超える場合、リーダーは更新前にリースを失う可能性があります。

関連トピック

- the section called “Hybrid Nodes ゲートウェイ”
- the section called “はじめに”
- the section called “オペレーション”
- the section called “トラブルシューティング”

Amazon EKS Hybrid Nodes ゲートウェイオペレーション

このページでは、高可用性、フェイルオーバー動作、モニタリング、スケーリング、VXLAN トンネルライフサイクルなど、Amazon EKS Hybrid Nodes ゲートウェイの 2 日目のオペレーションについて説明します。インストール手順については、「the section called “はじめに”」を参照してください。

高可用性とフェイルオーバー

Hybrid Nodes ゲートウェイは、Kubernetes リースベースのリーダー選出でアクティブスタンバイモデルを使用します。2 つのゲートウェイポッドは別々の EC2 ノードで実行され、ポッドのアンチアフィニティによって強制されます。どちらのポッドも起動時に VXLAN インターフェイスを作成し、すべてのハイブリッドノードの VTEP エントリを維持するノード調整を実行します。リーダーポッドのみが VPC ルートテーブルと `CiliumVTEPConfig` CRD を管理します。スタンバイポッドには、既にトンネルエントリの完全なセットがあるため、フェイルオーバー時に常に 3~5 秒以内にトラフィックを転送する準備ができています。

フェイルオーバーシーケンス

アクティブなゲートウェイインスタンスが失敗すると、以下のシーケンスが発生します。

1. スタンバイポッドは、リーダーリースの有効期限が切れたことを検出します。
2. スタンバイポッドはリースを取得し、新しいリーダーになります。
3. 新しいリーダーはリーダーセットアップシーケンスを実行します。
   - VPC ルートテーブルエントリを更新して、ハイブリッドポッド CIDR が新しいリーダーのプライマリ ENI を指すようにします。
   - 新しいリーダーのノード IP と VXLAN MAC アドレスを使用して `CiliumVTEPConfig` カスタムリソースをアップサートします。
4. 新しいリーダーを通過するトラフィックが再開されます。

両方のポッドは常に VXLAN インターフェイスと VTEP エントリを保持するため、新しいリーダーはフェイルオーバー中に VXLAN インターフェイスを再作成したり、トンネルエントリを再プログラムしたりする必要はありません。VPC ルートテーブルと `CiliumVTEPConfig` 更新のみが必要です。

予想されるフェイルオーバー時間は、約 3～5 秒です。フェイルオーバー中、VPC とハイブリッドポッド間のトラフィックは中断されます。

アベイラビリティーゾーンの推奨事項

2 つのアベイラビリティーゾーンにゲートウェイノードを分散して、AZ 障害がリーダーとスタンバイの両方を取り出さないようにします。EKS Auto Mode を使用する場合は、複数の AZ でサブネットセレクタを使用して `NodeClass` を設定します。マネージド型ノードグループまたはセルフマネージド型ノードの場合は、ラベル付け時に異なる AZ のノードを選択します。

> **Note**
>
> ゲートウェイと VPC 内の他のリソース間のクロス AZ トラフィックには、標準の AWS クロス AZ データ転送料金が発生します。

リーダー選出パラメータ

デフォルトのリーダー選出パラメータは、高速フェイルオーバー用に調整されます。

パラメータ	デフォルト	説明
`--leader-election-lease-duration`	`3s`	リーダーが更新を停止した後、非リーダーがリースの取得を試みるまで待機する時間。
`--leader-election-renew-deadline`	`2s`	リーダーがリースの更新を試行してから断念するまでの時間。
`--leader-election-`	`1s`	候補がリースの取得を再試行する頻度。

パラメータ	デフォルト	説明
retry-period		

これらの値を下げると、フェイルオーバー時間が短縮されますが、ネットワークパーティションで誤ったフェイルオーバーが発生するリスクが高まります。ほとんどのデプロイでは、デフォルトが適切です。詳細については、「the section called “設定”」を参照してください。

VPC ルートテーブル管理

ゲートウェイは VPC ルートテーブルエントリを管理し、ハイブリッドポッド CIDR 宛てのトラフィックがアクティブなゲートウェイインスタンスに到達するようにします。

ルートの管理方法

ゲートウェイポッドがリーダーになると、設定された各 VPC ルートテーブルでルートを作成または置き換えます。各ルートは、送信先 CIDR をハイブリッドポッド CIDR に設定し、ターゲットをリーダーのプライマリ ENI に設定します。ルートが既に存在し、正しい ENI を指している場合、ゲートウェイは更新をスキップします。

フェイルオーバー中、新しいリーダーは既存のルートを置き換えて、自身の ENI を指すようにします。これは、VPC トラフィックを新しいアクティブゲートウェイにリダイレクトするメカニズムです。

ルートテーブルエントリの例

ゲートウェイがルートを設定すると、VPC ルートテーブルには以下のようなエントリが含まれます。

ルーティング先	ターゲット	ステータス
10.0.0.0/16	ローカル	アクティブ
HYBRID_POD_CIDR	*eni-LEADER_ENI_ID*	アクティブ

IAM アクセス許可

ゲートウェイでは、ルートテーブルを管理するために以下の IAM アクションが必要です。

- ec2:DescribeRouteTables
- ec2:CreateRoute
- ec2:ReplaceRoute
- ec2:DescribeInstances

これらのアクセス許可を、ゲートウェイノードのインスタンスプロファイル、Pod Identity、または IRSA 設定に関連付けられた IAM ロールにアタッチします。

モニタリング

ヘルスエンドポイントと準備状況エンドポイント

ゲートウェイは、ポート 8088 でヘルスエンドポイントと準備状況エンドポイントを公開します。

エンドポイント	パス	説明
ヘルスチェック	/healthz	ゲートウェイプロセスが正常であれば、HTTP 200 を返します。Kubernetes ライブネスプローブによって使用されます。
準備状況チェック	/readyz	ゲートウェイがトラフィックを処理する準備ができたら、HTTP 200 を返します。Kubernetes 準備状況プローブによって使用されます。

診断のためにこれらのエンドポイントを手動でクエリするには、一時的なデバッグコンテナを実行するか、ポート転送を実行します。

```
kubectl port-forward -n eks-hybrid-nodes-gateway POD_NAME 8088:8088 &
curl -s http://localhost:8088/healthz
curl -s http://localhost:8088/readyz
```

メトリクスエンドポイント

ゲートウェイは、/metrics パスのポート 10080 で Prometheus 互換メトリクスを公開します。標準のコントローラーランタイムメトリクスに加えて、以下のカスタムメトリクスを使用できます。

ゲートウェイ情報:

メトリクス	型	説明
hybrid_gateway_info	ゲージ	ゲートウェイインスタンスに関する静的情報。常に 1。ラベル: node_ip、node_name 、vxlan_interface 、vpc_cidr、pod_cidr。

ハイブリッドノード:

メトリクス	型	説明
hybrid_gateway_hybrid_nodes_configured	ゲージ	VTEP エントリが設定されたハイブリッドノードの現在の数。

VTEP オペレーション:

メトリクス	型	説明
hybrid_gateway_vtep_add_total	Counter	成功した VTEP 追加オペレーションの合計。
hybrid_gateway_vtep_add_errors_total	Counter	失敗した VTEP 追加オペレーションの合計。
hybrid_gateway_vtep_remove_total	Counter	正常な VTEP 削除オペレーションの合計。
hybrid_gateway_vtep_remove_errors_total	Counter	失敗した VTEP 削除オペレーションの合計。

リーダー選出とルートテーブル:

メトリクス	型	説明
hybrid_gateway_leader_is_active	ゲージ	このポッドがアクティブなリーダーの場合は 1、スタンバイの場合は 0。
hybrid_gateway_leader_setup_duration_seconds	ヒストグラム	リーダーセットアップオペレーション (ルートテーブル + CiliumVTEPConfig) の秒単位の所要時間。
hybrid_gateway_aws_route_table_update_total	Counter	AWS ルートテーブルの更新オペレーションが成功した合計。
hybrid_gateway_aws_route_table_update_errors_total	Counter	失敗した AWS ルートテーブルの更新オペレーションの合計。
hybrid_gateway_aws_route_table_update_duration_seconds	ヒストグラム	AWS ルートテーブルの更新オペレーションの秒単位の期間。

ネットワーク統計 (スクレイプごとにオンデマンドで収集):

メトリクス	型	説明
hybrid_gateway_vxlan_rx_bytes_total	ゲージ	VXLAN インターフェイスで受信した合計バイト数。
hybrid_gateway_vxlan_tx_bytes_total	ゲージ	VXLAN インターフェイスで送信された合計バイト数。
hybrid_gateway_vxlan_rx_packets_total	ゲージ	VXLAN インターフェイスで受信したパケットの合計。
hybrid_gateway_vxlan_tx_packets_total	ゲージ	VXLAN インターフェイスで送信されたパケットの合計。

メトリクス	型	説明
hybrid_gateway_vxlan_rx_dropped_total	ゲージ	VXLAN インターフェイスによって受信時にドロップされたパケットの合計。
hybrid_gateway_vxlan_tx_dropped_total	ゲージ	VXLAN インターフェイスによって送信時にドロップされたパケットの合計。
hybrid_gateway_vxlan_rx_errors_total	ゲージ	VXLAN インターフェイスの合計受信エラー。
hybrid_gateway_vxlan_tx_errors_total	ゲージ	VXLAN インターフェイスの合計送信エラー。
hybrid_gateway_vxlan_interface_up	ゲージ	VXLAN インターフェイスが UP の場合は 1、それ以外の場合は 0。
hybrid_gateway_vxlan_fdb_entries	ゲージ	VXLAN インターフェイスの現在の FDB エントリの数。
hybrid_gateway_vxlan_route_count	ゲージ	VXLAN インターフェイス経由の現在のルート数。
hybrid_gateway_primary_nic_rx_bytes_total	ゲージ	プライマリネットワークインターフェイスで受信した合計バイト数。
hybrid_gateway_primary_nic_tx_bytes_total	ゲージ	プライマリネットワークインターフェイスで送信された合計バイト数。
hybrid_gateway_primary_nic_rx_packets_total	ゲージ	プライマリネットワークインターフェイスで受信したパケットの合計。
hybrid_gateway_primary_nic_tx_packets_total	ゲージ	プライマリネットワークインターフェイスで送信されたパケットの合計。

メトリクス	型	説明
hybrid_gateway_primary_nic_rx_dropped_total	ゲージ	プライマリ NIC によって受信時にドロップされたパケットの合計。
hybrid_gateway_primary_nic_tx_dropped_total	ゲージ	プライマリ NIC によって送信時にドロップされたパケットの合計数。
hybrid_gateway_primary_nic_rx_errors_total	ゲージ	プライマリ NIC の合計受信エラー数。
hybrid_gateway_primary_nic_tx_errors_total	ゲージ	プライマリ NIC の合計送信エラー数。
hybrid_gateway_primary_nic_info	ゲージ	プライマリ NIC 名。常に 1。ラベル: interface_name 。

CloudWatch Observability のアドオン

Amazon CloudWatch Observability アドオンを使用して、ゲートウェイメトリクスとログを収集できます。ポート 10080 でゲートウェイ名前空間 (eks-hybrid-nodes-gateway) をスクレイプするようにアドオンを設定します。正しい設定形式については、上記のアドオンドキュメントを参照してください。

スケーリングに関する考慮事項

Hybrid Nodes ゲートウェイはリーダー選出でアクティブスタンバイモデルを使用するため、一度にトラフィックを処理するポッドは 1 つだけです。ゲートウェイを水平スケーリングする (レプリカの数を増やす) と、フェイルオーバー中に引き継ぐ準備ができている追加のスタンバイポッドを提供することで可用性を向上させることができますが、トラフィックがレプリカ全体に分散されないため、パフォーマンスやスループットは向上しません。パフォーマンスをスケールするには、トラフィック量に十分なネットワーク帯域幅を持つ EC2 インスタンスタイプを選択して、垂直方向にスケールします。

インスタンスタイプのガイダンス

ゲートウェイのスループットは、EC2 インスタンスのネットワークパフォーマンスによって制限されます。インスタンスタイプを選択するときは、以下の点を考慮してください。

- ネットワーク帯域幅 — ゲートウェイは VPC とハイブリッドポッド間のすべてのトラフィックを転送します。ネットワーク帯域幅がピークトラフィック要件を満たすインスタンスタイプを選択します。
- パケット/秒 (PPS) — VXLAN カプセル化はパケットあたりのオーバーヘッドを追加します。多数の小さなパケットを含むワークロード (リクエストレートが高いマイクロサービスなど) は、PPS 制限が高いインスタンスタイプからメリットを得られます。
- ハイブリッドノードの数 — 各ハイブリッドノードは、ゲートウェイがトラフィックを転送する VXLAN トンネルエンドポイントを追加します。ハイブリッドノードの数が増えると、ゲートウェイ経由の集約トラフィックが増加します。クラスターのピーク時のネットワーク間トラフィックを処理するのに十分なネットワーク帯域幅を持つインスタンスタイプを選択します。

推奨インスタンスタイプ

本番稼働 (10 ~ 100 ハイブリッドノード、中程度のトラフィック)

安定したネットワーク間トラフィックを持つ標準の本番稼働ワークロードに適しています。

インスタンスタイプ	vCPU	メモリ	Network	注意事項
`c6i.xlarge`	4	8 GiB	最大 12.5 Gbps	コストとパフォーマンスの適切なバランス
`c6in.xlarge`	4	8 GiB	最大 30 Gbps	ネットワーク最適化。本番環境に推奨
`c7i.xlarge`	4	8 GiB	最大 12.5 Gbps	最新世代のコンピューティング最適化
`m6i.xlarge`	4	16 GiB	最大 12.5 Gbps	ゲートウェイノードで他のワークロードを共同配置する場合に適しています

高スループットの本番稼働 (100 以上のハイブリッドノード、大量のトラフィック)

データ集約型のワークロードや多数の同時接続など、ネットワーク間の帯域幅要件が大きい環境向け。

インスタンスタイプ	vCPU	メモリ	Network	注意事項
c6in.2xlarge	8	16 GiB	最大 40 Gbps	高スループットの本番稼働に推奨
c5n.2xlarge	8	21 GiB	最大 25 Gbps	旧世代のネットワーク最適化、コスト効率
c6in.4xlarge	16	32 GiB	最大 50 Gbps	非常に負荷の高いワークロードの最大スループット
c5n.4xlarge	16	42 GiB	最大 25 Gbps	極端なパケットレートの vCPU 数が多い

ゲートウェイメトリクスを使用してネットワーク使用率をモニタリングし (「the section called “メトリクスエンドポイント”」を参照)、必要に応じてインスタンスタイプを調整します。

VXLAN トンネルのライフサイクル

ゲートウェイは、ハイブリッドノードがクラスターに参加または終了するときに、ハイブリッドノードへの VXLAN トンネルを自動的に維持します。

トンネルの管理方法

ノードコントローラーは、クラスター内の CiliumNode オブジェクトを監視します。リーダーとスタンバイの両方が最新のトンネル状態になるように、コントローラーは (リーダーだけでなく) すべてのゲートウェイポッドで実行されます。CiliumNode イベントが発生すると、コントローラーは eks.amazonaws.com/compute-type: hybrid ラベルを探してノードがハイブリッドノードであるかどうかをチェックします。

ハイブリッドノードがクラスターに参加する場合:

1. コントローラーは新しい CiliumNode オブジェクトを検出します。
2. ノードの内部 IP アドレスとポッド CIDR を CiliumNode 仕様から抽出します。

3. VXLAN インターフェイスで以下をプログラムします。
   - VXLAN インターフェイスを介したノードの IP を介したノードのポッド CIDR のルート。
   - ノードの IP を決定論的な MAC アドレスにマッピングする静的 ARP エントリ。
   - カプセル化されたパケットをノードの IP に送信するように VXLAN モジュールに指示する FDB エントリ。

ハイブリッドノードがクラスターを離れる場合:

1. コントローラーは `CiliumNode` 削除を検出します。
2. VXLAN インターフェイスからそのノードのルート、ARP エントリ、および FDB エントリを削除します。

このライフサイクルは完全に自動です。ハイブリッドノードを追加または削除するときに、トンネルを手動で設定する必要はありません。

次のステップ

- the section called “設定” — Helm 値、CLI フラグ、リーダー選出パラメータをカスタマイズします。
- the section called “トラブルシューティング” — 一般的な問題を診断して解決します。
- the section called “Hybrid Nodes ゲートウェイ” — 概要ページに戻ります。

Amazon EKS Hybrid Nodes ゲートウェイのトラブルシューティング

このページでは、Amazon EKS Hybrid Nodes ゲートウェイの一般的な問題の診断と解決に関するガイダンスを提供します。それぞれのセクションで、症状、考えられる原因、診断手順、解決策を説明しています。運用の詳細については、「the section called “オペレーション”」を参照してください。

ハイブリッドノードのポッドに VPC からアクセスできない

ハイブリッドノードで実行されているポッドに、EC2 インスタンス、ロードバランサー、Kubernetes コントロールプレーンなどの VPC のリソースからアクセスできません。

考えられる原因:

- VPC ルートテーブルエントリがないか、誤った ENI を指している。
- ゲートウェイリーダーポッドが実行されていないか、セットアップが完了していない。

- ハイブリッドノードで Cilium VTEP が有効になっていないか設定されていない。
- ゲートウェイ EC2 インスタンスで送信元/送信先チェックが有効になっている。

診断手順:

1. VPC ルートテーブルエントリを確認します。ハイブリッドポッド CIDR のルートが存在し、アクティブなゲートウェイインスタンスのプライマリ ENI を指していることを確認します。

```
aws ec2 describe-route-tables \
  --route-table-ids ROUTE_TABLE_ID \
  --query "RouteTables[].Routes[?DestinationCidrBlock=='POD_CIDR']"
```

   ルートがない場合は、ゲートウェイログでルートテーブルエラーがないか確認します。ルートが間違った ENI を指している場合は、フェイルオーバーが正常に完了していない可能性があります。

2. ゲートウェイポッドのステータスとリーダー選出を確認します。2 つのゲートウェイポッドが実行されていて、一方がリーダーリースを保持していることを確認します。

```
kubectl get pods -n eks-hybrid-nodes-gateway
kubectl get lease -n eks-hybrid-nodes-gateway
```

   リースを保持しているポッドがない場合は、「the section called “リーダー選出の問題”」を参照してください。

3. ハイブリッドノードの Cilium VTEP 設定を確認します。`CiliumVTEPConfig` リソースが存在し、リーダーのノード IP が含まれていることを確認します。

```
kubectl get ciliumvtepconfig hybrid-gateway -o yaml
```

   `spec.endpoints[0].tunnelEndpoint` は、リーダーゲートウェイノードの IP アドレスと一致している必要があります。このリソースがないか、古い IP になっている場合、ゲートウェイでリーダーのセットアップが完了していない可能性があります。

4. 送信元/送信先チェックを確認します。ゲートウェイ EC2 インスタンスで送信元/送信先チェックが無効になっていることを確認します。

```
aws ec2 describe-instance-attribute \
  --instance-id GATEWAY_INSTANCE_ID \
```

```
  --attribute sourceDestCheck
```

sourceDestCheck が true になっている場合は無効にします。「the section called “はじめに”」を参照してください。

ハイブリッドノードのウェブフック呼び出しが失敗する

ハイブリッドノードで実行されているウェブフックエンドポイントに Kubernetes API サーバーから到達できません。ウェブフックアドミッションリクエストがタイムアウトするか、接続エラーが返されます。

考えられる原因:

- コントロールプレーンからハイブリッドポッドへのトラフィックがゲートウェイでルーティングされていない。
- CiliumVTEPConfig リソースがないか、古いエンドポイント IP になっている。

診断手順:

1. コントロールプレーンからゲートウェイノード IP に到達できることを確認します。トラフィックはコントロールプレーンから VPC ルートテーブルに送信され、そこからゲートウェイの ENI に転送されます。「the section called “ハイブリッドノードのポッドに VPC からアクセスできない”」の手順を使用して、VPC ルートテーブルエントリが正しいことを確認します。
2. CiliumVTEPConfig リソースを確認します。このリソースが存在し、tunnelEndpoint が現在のリーダーのノード IP と一致していることを確認します。

```
kubectl get ciliumvtepconfig hybrid-gateway -o yaml
```

トンネルエンドポイントが古くなっている場合 (以前のリーダーを指している場合)、ゲートウェイでリーダーセットアップシーケンスが完了していない可能性があります。ゲートウェイログで CiliumVTEPConfig アップサート時のエラーがないか確認してください。

VPC ルートテーブルの更新が失敗する

ゲートウェイログに VPC ルートテーブルオペレーションに関連するエラーが表示され、ハイブリッドポッド CIDR のルートが作成または更新されません。

考えられる原因:

- ゲートウェイの IAM ロールに必要な EC2 アクセス許可がない。
- 設定のルートテーブル ID が正しくないか、ルートテーブルが存在しない。
- ゲートウェイから EC2 API エンドポイントに到達できない。

診断手順:

1. IAM アクセス許可を確認します。ゲートウェイには以下の IAM アクションが必要です。
   - `ec2:DescribeRouteTables`
   - `ec2:CreateRoute`
   - `ec2:ReplaceRoute`
   - `ec2:DescribeInstances`

   ゲートウェイノードのインスタンスプロファイルまたは Pod Identity 設定にアタッチされている IAM ロールを確認します。
2. 設定でルートテーブル ID を確認します。ROUTE_TABLE_IDS 環境変数にゲートウェイデプロイの有効なルートテーブル ID が含まれていることを確認します。

   ```
   kubectl get deployment eks-hybrid-nodes-gateway -n eks-hybrid-nodes-gateway -o
    jsonpath='{.spec.template.spec.containers[0].env}' | jq .
   ```

   VPC にルートテーブル ID が存在することを確認します。

   ```
   aws ec2 describe-route-tables --route-table-ids ROUTE_TABLE_ID
   ```

3. ゲートウェイログでルートテーブルエラーがないか確認します。ルートテーブルオペレーションに関連するエラーメッセージを探します。

   ```
   kubectl logs -n eks-hybrid-nodes-gateway LEADER_POD | grep -i "route table"
   ```

   一般的なエラーメッセージとしては以下があります。
   - `Failed to verify route table access` — ゲートウェイでルートテーブルを記述できません。IAM アクセス許可とルートテーブル ID を確認します。
   - `Failed to update route tables` — ゲートウェイでルートの作成または置き換えができません。IAM 許可を確認します。

- `failed to access route table` — ルートテーブル ID が正しくないか、IAM ロールに `ec2:DescribeRouteTables` がない可能性があります。

ゲートウェイポッドの起動に失敗するか異常がある

ゲートウェイポッドの状態が `CrashLoopBackOff`、`Error`、または `Pending` であるか、ヘルスエンドポイントからエラーが返されます。

考えられる原因:

- 必要な環境変数 (`VPC_CIDR`、`POD_CIDRS`、`ROUTE_TABLE_IDS`) が設定されていない。
- ゲートウェイノードで IP 転送が有効になっていない。
- ノードラベルまたはアンチアフィニティの制約でスケジューリングが禁止されている。

診断手順:

1. ポッドログを確認します。問題があるポッドのログを確認してエラーを特定します。

```
kubectl logs -n eks-hybrid-nodes-gateway LEADER_POD
```

2. 必要な環境変数を確認します。ゲートウェイには、`NODE_IP`、`VPC_CIDR`、および `POD_CIDRS` が必要です。いずれかがない場合、ゲートウェイはすぐに終了します。ポッド仕様を確認します。

```
kubectl get pod -n eks-hybrid-nodes-gateway LEADER_POD -o
 jsonpath='{.spec.containers[0].env}' | jq .
```

   - `NODE_IP` は、ポッド仕様の `status.hostIP` から自動的に設定されます。空の場合、ポッドがノードでまだスケジュールされていない可能性があります。
   - `VPC_CIDR` と `POD_CIDRS` は Helm 値から取得されます。正しく設定されていることを確認してください。
3. IP 転送を確認します。ゲートウェイの起動時に IP 転送が有効になっているかが確認され、有効になっていないとゲートウェイは終了します。ポッドログで `IP forwarding is not enabled` というエラーメッセージを探します。ノードで IP 転送を有効にします。

```
# Check current setting
cat /proc/sys/net/ipv4/ip_forward
```

```
# Enable if not set
sudo sysctl -w net.ipv4.ip_forward=1
```

永続的な設定にするには、kubelet から IP 転送を設定するか、`net.ipv4.ip_forward=1` を `/etc/sysctl.d/` に追加します。

4. ノードラベルとスケジューリングの制約を確認します。ゲートウェイポッドのノードには `hybrid-gateway-node=true` というラベルが必要です。ポッドアンチアフィニティにより、各ポッドが個別のノードで実行されます。ポッドが Pending の場合は、スケジューリングの問題がないか確認します。

```
kubectl describe pod -n eks-hybrid-nodes-gateway LEADER_POD
```

ノードの不足、ラベルの欠落、アンチアフィニティの競合を示すイベントを探します。

リーダー選出の問題

ゲートウェイポッドが実行されていますが、いずれのポッドもリーダーリースを取得しないか、リーダーシップの移行が頻繁に発生します。

考えられる原因:

- Lease オブジェクトに対する RBAC アクセス許可がない。
- ゲートウェイポッドと Kubernetes API サーバーの間のネットワーク接続が不安定である。
- リーダー選出パラメータの設定が間違っている。

診断手順:

1. Lease オブジェクトを確認します。Lease が存在することを確認し、現在のホルダーを調べます。

```
kubectl get lease -n eks-hybrid-nodes-gateway hybrid-gateway-leader -o yaml
```

現在のリーダーは `spec.holderIdentity` フィールドに表示されます。リースが最後にいつ更新されたかが `spec.renewTime` に表示されます。`renewTime` が古くなっている場合、リーダーの API サーバーへの接続が失われている可能性があります。

2. RBAC アクセス許可を確認します。ゲートウェイサービスアカウントには、ゲートウェイ名前空間で Lease オブジェクトを取得、作成、更新するためのアクセス許可が必要です。Role と RoleBinding を確認します。

```
kubectl get role -n eks-hybrid-nodes-gateway
kubectl get rolebinding -n eks-hybrid-nodes-gateway
```

   Role の coordination.k8s.io API グループに、leases リソースに対する get、create、および update の動詞が含まれている必要があります。

3. ポッドログでリースエラーがないか確認します。ポッドログでリーダー選出エラーを探します。

```
kubectl logs -n eks-hybrid-nodes-gateway LEADER_POD | grep -i "leader\|lease"
```

   一般的な問題は次のとおりです。

   - Failed to acquire lease — ポッドでリースオブジェクトを作成または更新できません。RBAC アクセス許可を確認します。
   - Leadership ended に続いて Leader setup complete というメッセージが頻繁に発生 — リーダーがリースを失って再取得しています。これは、ポッドと API サーバーの間のネットワークが不安定であることを示している可能性があります。--leader-election-lease-duration を増やすことを検討してください。

4. リーダー選出パラメータを確認します。設定されている値を確認します。

```
kubectl get deployment eks-hybrid-nodes-gateway -n eks-hybrid-nodes-gateway -o
 jsonpath='{.spec.template.spec.containers[0].args}'
```

   --leader-election-renew-deadline が --leader-election-lease-duration より小さいことを確認します。更新期限がリース期間を超えている場合、リーダーは更新前にリースを失います。詳細については、「the section called “リーダー選出調整”」を参照してください。

一般的なエラーメッセージ

次の表に、ゲートウェイポッドログに表示される可能性があるエラーメッセージとその解決策を示します。

エラーメッセージ	原因	解決策
IP forwarding is not enabled	ゲートウェイノードでカーネルパラメータ net.ipv4.ip_forward が 1 に設定されていません。	kubelet 設定を使用するか sysctl -w net.ipv4.ip_forward=1 を実行して IP 転送を有効にします。
Failed to setup VXLAN	ゲートウェイで VXLAN ネットワークインターフェイスを作成できません。通常、これはポッドに NET_ADMIN 機能がない場合に発生します。	デプロイ仕様で securityContext.capabilities.add に NET_ADMIN が含まれていることを確認します。Helm チャートが正しくデプロイされていることを確認します。
Failed to verify route table access	起動時に、ゲートウェイで 1 つ以上の VPC ルートテーブルを記述できません。	IAM ロールに ec2:DescribeRouteTables アクセス許可があり、設定のルートテーブル ID が正しいことを確認します。
Failed to update route tables	ゲートウェイで VPC ルートテーブルのルートの作成または置き換えができません。	IAM ロールに ec2:CreateRoute および ec2:ReplaceRoute のアクセス許可があることを確認します。
Failed to create route table manager	ゲートウェイで AWS EC2 クライアントを初期化できないか、インスタンスのプライマリ ENI を取得できません。	IAM ロールに ec2:DescribeInstances アクセス許可があり、インスタンスメタデータサービス (IMDS) にアクセスできることを確認します。
NODE_IP is required	NODE_IP 環境変数または --node-ip フラグが設定されていません。	ポッド仕様で fieldRef を使用して status.hostIP の NODE_IP が設定されていることを確認します。Helm チャートが正しくデプロイされていることを確認します。

エラーメッセージ	原因	解決策
`Invalid NODE_IP`	NODE_IP に指定された値が有効な IP アドレスではありません。	ポッド仕様で NODE_IP 環境変数の値を確認します。
`pod-cidrs and vpc-cidr are required`	POD_CIDRS または VPC_CIDR 環境変数が空です。	インストール時に podCIDRs と vpcCIDR の Helm 値を設定します。
`No valid route table IDs provided`	ROUTE_TABLE_IDS の値が設定されていますが、有効なルートテーブル ID が解析で見つかりません。	routeTableIDs の Helm 値にフォーマットエラーがないか確認します。ルートテーブル ID はカンマで区切る必要があります (例: `rtb-abc123,rtb-def456`)。
`Failed to auto-detect AWS region`	ゲートウェイで EC2 インスタンスメタデータから AWS リージョンを取得できません。	インスタンスメタデータサービス (IMDS) にアクセスできることを確認します。あるいは、`--aws-region` フラグまたは AWS_REGION 環境変数を明示的に設定します。
`Failed to auto-detect AWS instance ID`	ゲートウェイで EC2 インスタンスメタデータからインスタンス ID を取得できません。	インスタンスメタデータサービス (IMDS) にアクセスできることを確認します。あるいは、`--aws-instance-id` フラグまたは AWS_INSTANCE_ID 環境変数を明示的に設定します。
`CiliumNode has no internal IP`	ハイブリッドノードの CiliumNode オブジェクトの仕様に内部 IP アドレスがありません。	ハイブリッドノードが正しく登録され、Cilium エージェントが実行されていることを確認します。ノードの CiliumNode リソースを確認します。

エラーメッセージ	原因	解決策
`CiliumNode <name> has no pod CIDRs allocated`	ハイブリッドノードの `CiliumNode` オブジェクトに、Cilium IPAM によって割り当てられたポッド CIDR がありません。	ハイブリッドノードで Cilium IPAM が正しく設定されていることを確認します。`CiliumNode` リソースでノードの IPAM ステータスを確認します。
`Failed to upsert CiliumVTEPConfig`	ゲートウェイで `CiliumVTEPConfig` カスタムリソースを作成または更新できません。	クラスターに CRD がインストールされており、ゲートウェイサービスアカウントに `CiliumVTEPConfig` リソースを管理するためのアクセス許可があることを確認します。
`Unable to create manager`	コントローラーランタイムマネージャーの初期化に失敗しました。	ポッドログで追加のコンテキストを確認します。一般的な原因には、無効な kubeconfig や Kubernetes API サーバーに到達できないことなどがあります。
`Failed to add gateway setup`	リーダー選出ランナブルをコントローラーマネージャーに登録できませんでした。	通常、これは内部エラーです。ポッドログ全体で追加のコンテキストを確認し、GitHub リポジトリで問題を報告します。
`Unable to create Node controller`	CiliumNode リコンシラーをコントローラーマネージャーに登録できませんでした。	ポッドログで追加のコンテキストを確認します。クラスターに CiliumNode CRD がインストールされていることを確認します。
`Problem running manager`	コントローラーマネージャーが予期せず終了しました。	ポッドログで原因となっているエラーを確認します。一般的な原因には、Kubernetes API サーバーへの接続の喪失、メトリクスまたはヘルスプローブのバインドアドレスにおけるポートの競合などがあります。

エラーメッセージ	原因	解決策
`failed to access route table <id>`	起動時の検証チェックにおいて、ゲートウェイで特定の VPC ルートテーブルを記述できません。	IAM ロールに `ec2:DescribeRouteTables` アクセス許可があり、ルートテーブル ID が正しいことを確認します。ルートテーブルは、ゲートウェイインスタンスと同じリージョンに存在する必要があります。

関連トピック

- the section called “Hybrid Nodes ゲートウェイ” — ゲートウェイアーキテクチャの概要とユースケース。
- the section called “はじめに” — 前提条件とインストール手順。
- the section called “設定” — Helm 値、CLI フラグ、環境変数の完全なリファレンス。
- the section called “オペレーション” — モニタリング、フェイルオーバー動作、スケーリングのガイダンス。

クラスターのアプリケーションデータストレージを使用する

アプリケーションのストレージニーズに応じて、Amazon EKS でさまざまな AWS ストレージサービスを使用できます。AWS でサポートされている幅広い Container Storage Interface (CSI) ドライバーを使用すると、Amazon EKS で実行されているアプリケーションのストレージニーズに応じて、Amazon EBS、Amazon S3、Amazon S3 Files、Amazon EFS、Amazon FSx、および Amazon File Cache を簡単に使用できます。Amazon EKS クラスターのバックアップを管理するには、「Amazon EKS の AWS バックアップサポート」を参照してください。

この章では、Amazon EKS クラスターのストレージオプションについて説明します。

トピック

- Amazon EBS で Kubernetes ボリュームストレージを使用する
- Kubernetes ボリュームストレージ用の Amazon EC2 インスタンスストア
- Amazon EFS で Elastic File System ストレージを使用する
- Amazon EFS CSI ドライバーで、Amazon S3 Filesシステムストレージを使用する
- Amazon FSx for Lustre で高性能なアプリケーションストレージを使用する
- FSx for NetApp ONTAP で高性能なアプリケーションストレージを使用する
- Amazon FSx for OpenZFS でデータストレージを使用する
- Amazon File Cache を使用してレイテンシーを最小化する
- Mountpoint for Amazon S3 CSI ドライバーを使用して Amazon S3 オブジェクトにアクセスする
- CSI ボリュームのためにスナップショット機能を有効にする

Amazon EBS で Kubernetes ボリュームストレージを使用する

> **Note**
>
> 新規: Amazon EKS Auto Mode は、ブロックストレージのルーチンタスクを自動化します。the section called “ステートフルワークロードをデプロイする” では、その方法を説明しています。

Amazon Elastic Block Store (Amazon EBS) コンテナストレージインターフェイス (CSI) ドライバーは、Amazon EBS ボリュームのライフサイクルを、作成した Kubernetes ボリュームのストレー

ジとして管理します。Amazon EBS CSI ドライバーは、ジェネリック「エフェメラルボリューム」と「永続ボリューム」という種類の Kubernetes ボリューム用の Amazon EBS ボリュームを作成します。

考慮事項

- EKS Auto Mode クラスターに Amazon EBS CSI コントローラーをインストールする必要はありません。
- Amazon EBS ボリュームを Fargate Pod にマウントすることはできません。
- Amazon EBS CSI コントローラーは Fargate ノードで実行できますが、Amazon EBS CSI ノード `DaemonSet` は Amazon EC2 インスタンスでのみ実行できます。
- Amazon EBS ボリュームと Amazon EBS CSI ドライバーは Amazon EKS Hybrid Nodes と互換性がありません。
- 最新のアドオンバージョンと 1 つの以前のバージョンがサポートされます。最新バージョンで見つかったバグや脆弱性の修正は、新しいマイナーバージョンの以前のリリースにバックポートされます。
- EKS Auto Mode では、プロビジョナーとして `ebs.csi.eks.amazonaws.com` を使用するストレージクラスが必要です。標準の Amazon EBS CSI ドライバー (`ebs.csi.aws.com`) は、独自のボリュームを個別に管理します。EKS Auto Mode で既存のボリュームを使用するには、ボリュームスナップショットを使用して、Auto Mode プロビジョナーを使用するストレージクラスに移行します。

> **Important**
>
> Amazon EBS CSI ドライバーのスナップショット機能を使用するには、まず CSI スナップショットコントローラーをインストールする必要があります。詳細については、「the section called “CSI スナップショットコントローラー”」を参照してください。

前提条件

- 既存のクラスター。必要なプラットフォームバージョンを確認するには、次のコマンドを実行します。

```
aws eks describe-addon-versions --addon-name aws-ebs-csi-driver
```

- EBS CSI ドライバーには AWS IAM アクセス許可が必要です。
  - AWS では、EKS Pod Identity の使用を推奨しています。詳細については、「the section called “EKS Pod Identity のセットアップの概要”」を参照してください。
  - サービスアカウントの IAM ロールの詳細については、the section called “IAM OIDC プロバイダー” を参照してください。

ステップ 1: IAM ロールを作成する

Amazon EBS CSI プラグインでは、ユーザーに代わって AWS API の呼び出しを行うための IAM アクセス許可が必要です。これらのステップを実行しない場合、アドオンをインストールして `kubectl describe pvc` を実行しようとすると、`could not create volume in EC2: UnauthorizedOperation` エラーとともに `failed to provision volume with StorageClass` が表示されます。詳細については、GitHub の「Set up driver permissions」を参照してください。

> **Note**
>
> Pod は、IMDS へのアクセスをブロックする場合を除き、IAM ロールに割り当てられたアクセス許可にアクセスできます。詳細については、the section called “ベストプラクティス” を参照してください。

以下の手順は、IAM ロールを作成し、それに AWS マネージドポリシーをアタッチする方法を示しています。この手順を実装するには、次のいずれかのツールを使用できます。

- the section called “`eksctl`”
- the section called “AWS マネジメントコンソール”
- the section called “AWS CLI”

> **Note**
>
> スコープダウンされたアクセス許可を持つセルフマネージドポリシーを作成できます。`AmazonEBSCSIDriverPolicyV2` を確認し、アクセス許可を減らしたカスタム IAM ポリシーを作成します。`AmazonEBSCSIDriverPolicy` から移行する場合は、「EBS CSI ドライバーポリシーの移行」を参照してください。

> **Note**
>
> この手順には、ドライバーを Amazon EKS アドオンとして使用するための特定のステップが書かれています。ドライバーをセルフマネージドのアドオンとして使用するには、さまざまなステップが必要です。詳細については、GitHub の「Set up driver permissions」を参照してください。

`eksctl`

1. IAM ロールを作成して、ポリシーをアタッチします。AWS が AWS マネージドポリシーを維持しますが、ユーザーが独自のカスタムポリシーを作成することもできます。次のコマンドを使用して、IAM ロールを作成し、それに AWS マネージドポリシーをアタッチできます。*`my-cluster`* の部分は自分のクラスター名に置き換えます。このコマンドは IAM ロールを作成して IAM ポリシーをアタッチする AWS CloudFormation スタックをデプロイします。

   ```
   eksctl create iamserviceaccount \
           --name ebs-csi-controller-sa \
           --namespace kube-system \
           --cluster my-cluster \
           --role-name AmazonEKS_EBS_CSI_DriverRole \
           --role-only \
           --attach-policy-arn arn:aws:iam::aws:policy/service-role/
   AmazonEBSCSIDriverPolicyV2 \
           --approve
   ```

2. カスタム KMS キーを使用しない場合は、このステップをスキップできます。このキーを Amazon EBS ボリュームでの暗号化に使用する場合、必要に応じて IAM ロールをカスタマイズします。例えば、以下を実行してください。

   a. 次のコードをコピーし、新しい `kms-key-for-encryption-on-ebs.json` ファイルに貼り付けます。*`custom-key-arn`* をカスタム KMS key ARN に置き換えます。

      ```
      {
            "Version":"2012-10-17",
            "Statement": [
              {
                "Effect": "Allow",
                "Action": [
                  "kms:CreateGrant",
                  "kms:ListGrants",
      ```

```
            "kms:RevokeGrant"
          ],
          "Resource": ["arn:aws:kms:us-
east-1:123456789012:key/1234abcd-12ab-34cd-56ef-1234567890ab"],
          "Condition": {
            "Bool": {
              "kms:GrantIsForAWSResource": "true"
            }
          }
        },
        {
          "Effect": "Allow",
          "Action": [
            "kms:Encrypt",
            "kms:Decrypt",
            "kms:ReEncrypt*",
            "kms:GenerateDataKey*",
            "kms:DescribeKey"
          ],
          "Resource": ["arn:aws:kms:us-
east-1:123456789012:key/1234abcd-12ab-34cd-56ef-1234567890ab"]
        }
      ]
    }
```

b. ポリシーを作成します。*KMS_Key_For_Encryption_On_EBS_Policy* を別の名前に変更できます。ただし、変更する場合は、必ず後の手順で変更してください。

```
aws iam create-policy \
      --policy-name KMS_Key_For_Encryption_On_EBS_Policy \
      --policy-document file://kms-key-for-encryption-on-ebs.json
```

c. 次のコマンドを使用して、IAM ポリシーをロールに添付します。*111122223333* は、ご自分のアカウント ID に置き換えます。

```
aws iam attach-role-policy \
      --policy-arn arn:aws:iam::111122223333:policy/
KMS_Key_For_Encryption_On_EBS_Policy \
      --role-name AmazonEKS_EBS_CSI_DriverRole
```

AWS マネジメントコンソール

1. IAM コンソール (https://console.aws.amazon.com/iam/) を開きます。
2. 左のナビゲーションペインで、[ロール] を選択してください。
3. [ロール] ページで、[ロールの作成] を選択してください。
4. [信頼されたエンティティを選択] ページで、以下の操作を実行します。
   a. [信頼されたエンティティの種類] セクションで、[ウェブ アイデンティティ] を選択します。
   b. [ID プロバイダー] で、(Amazon EKS の [概要] に示されているように) クラスターに [OpenID Connect プロバイダーの URL] を選択します。
   c. [対象者] で [sts.amazonaws.com] を選択します。
   d. [次へ] を選択します。
5. [アクセス許可を追加] ページで、以下を実行します。
   a. [フィルタポリシー] ボックスに AmazonEBSCSIDriverPolicyV2 と入力します。
   b. 検索で返された AmazonEBSCSIDriverPolicyV2 の左にあるチェックボックスを選択します。
   c. [次へ] を選択します。
6. [名前を付けて、レビューし、作成する] ページで、以下の操作を実行します。
   a. [ロール名] に、*AmazonEKS_EBS_CSI_DriverRole* などのロールの一意の名前を入力します。
   b. [タグの追加 (オプション)] で、タグをキーバリューのペアとして添付して、メタデータをロールに追加します。IAM でのタグの使用に関する詳細については、IAM ユーザーガイドの「IAM リソースにタグを付ける」を参照してください。
   c. [ロールの作成] を選択してください。
7. ロールが作成されたら、コンソールでロールを選択して編集用に開きます。
8. [信頼関係] タブを選択し、続いて [信頼ポリシーの編集] を選択します。
9. 次の行と似ている行を探してます。

```
"oidc.eks.region-code.amazonaws.com/id/EXAMPLED539D4633E53DE1B71EXAMPLE:aud":
 "sts.amazonaws.com"
```

前の行の末尾にカンマを追加し、前の行の後に次の行を追加します。*region-code* を、クラスターのある AWS リージョンに置き換えます。*EXAMPLED539D4633E53DE1B71EXAMPLE* をクラスターの OIDC プロバイダー ID に置き換えます。

```
"oidc.eks.region-code.amazonaws.com/id/EXAMPLED539D4633E53DE1B71EXAMPLE:sub":
 "system:serviceaccount:kube-system:ebs-csi-controller-sa"
```

10[ポリシーの更新] を選択して終了します。

11Amazon EBS ボリュームでの暗号化にカスタム「KMS キー」を使用する場合、必要に応じて IAM ロールをカスタマイズします。例えば、以下を実行してください。

a. 左のナビゲーションペインの [ポリシー] を選択します。

b. [ポリシー] ページで、[ポリシーの作成] を選択します。

c. [ポリシーの作成] ページで、[JSON] タブを選択します。

d. 次のコードをコピーしてエディタに貼り付け、*[custom-key-arn]* をカスタム KMS key ARN に置き換えます。

```
{
      "Version":"2012-10-17",
      "Statement": [
        {
          "Effect": "Allow",
          "Action": [
            "kms:CreateGrant",
            "kms:ListGrants",
            "kms:RevokeGrant"
          ],
          "Resource": ["arn:aws:kms:us-
east-1:123456789012:key/1234abcd-12ab-34cd-56ef-1234567890ab"],
          "Condition": {
            "Bool": {
              "kms:GrantIsForAWSResource": "true"
            }
          }
        },
        {
          "Effect": "Allow",
          "Action": [
            "kms:Encrypt",
            "kms:Decrypt",
            "kms:ReEncrypt*",
            "kms:GenerateDataKey*",
            "kms:DescribeKey"
          ],
```

```
            "Resource": ["arn:aws:kms:us-
east-1:123456789012:key/1234abcd-12ab-34cd-56ef-1234567890ab"]
          }
        ]
    }
```

e. [次へ: タグ] を選択します。

f. [タグの追加 (オプション)] ページで、[次へ: 確認] を選択します。

g. 名前 には、ポリシーの一意の名前を入力します (例: *KMS_Key_For_Encryption_On_EBS_Policy*)。

h. [ポリシーの作成] を選択します。

i. 左のナビゲーションペインで、[ロール] を選択してください。

j. コンソールで ***[AmazonEKS_EBS_CSI_DriverRole]*** を選択し、編集用に開きます。

k. [アクセス許可を追加] ドロップダウンリストから [ポリシーをアタッチ] を選択します。

l. [フィルターポリシー] ボックスに、*KMS_Key_For_Encryption_On_EBS_Policy* と入力します。

m. 検索で返された *KMS_Key_For_Encryption_On_EBS_Policy* の左側にあるチェックボックスをオンにします。

n. [ポリシーのアタッチ] を選択します。

AWS CLI

1. クラスターの OIDC プロバイダーの URL を表示します。*my-cluster* の部分は自分のクラスター名に置き換えます。コマンドの出力が None の場合は、「前提条件」を確認してください。

```
aws eks describe-cluster --name my-cluster --query "cluster.identity.oidc.issuer" --
output text
```

出力例は次のとおりです。

```
https://oidc.eks.region-code.amazonaws.com/id/EXAMPLED539D4633E53DE1B71EXAMPLE
```

2. IAM ロールを作成して AssumeRoleWithWebIdentity アクションを付与します。

a. 次の内容を aws-ebs-csi-driver-trust-policy.json という名前のファイルにコピーします。*111122223333* は、ご自分のアカウント ID に置き換えま

す。*EXAMPLED539D4633E53DE1B71EXAMPLE* および *region-code* を、前のステップで返された値にそれぞれ置き換えます。

```
{
      "Version":"2012-10-17",
      "Statement": [
        {
          "Effect": "Allow",
          "Principal": {
            "Federated": "arn:aws:iam::111122223333:oidc-provider/oidc.eks.us-
east-1.amazonaws.com/id/EXAMPLED539D4633E53DE1B71EXAMPLE"
          },
          "Action": "sts:AssumeRoleWithWebIdentity",
          "Condition": {
            "StringEquals": {
              "oidc.eks.us-east-1.amazonaws.com/id/
EXAMPLED539D4633E53DE1B71EXAMPLE:aud": "sts.amazonaws.com",
              "oidc.eks.us-east-1.amazonaws.com/id/
EXAMPLED539D4633E53DE1B71EXAMPLE:sub": "system:serviceaccount:kube-system:ebs-csi-
controller-sa"
            }
          }
        }
      ]
    }
```

b. ロールを作成します。*AmazonEKS_EBS_CSI_DriverRole* を別の名前に変更できます。変更する場合は、必ず後の手順で変更してください。

```
aws iam create-role \
      --role-name AmazonEKS_EBS_CSI_DriverRole \
      --assume-role-policy-document file://"aws-ebs-csi-driver-trust-policy.json"
```

3. ポリシーをアタッチします。AWS が AWS マネージドポリシーを維持しますが、ユーザーが独自のカスタムポリシーを作成することもできます。次のコマンドを使用して、AWS マネージドポリシーをロールにアタッチします。

```
aws iam attach-role-policy \
      --policy-arn arn:aws:iam::aws:policy/service-role/AmazonEBSCSIDriverPolicyV2 \
      --role-name AmazonEKS_EBS_CSI_DriverRole
```

4. Amazon EBS ボリュームでの暗号化にカスタム「KMS キー」を使用する場合、必要に応じて IAM ロールをカスタマイズします。例えば、以下を実行してください。

   a. 次のコードをコピーし、新しい kms-key-for-encryption-on-ebs.json ファイルに貼り付けます。*custom-key-arn* をカスタム KMS key ARN に置き換えます。

```
{
      "Version":"2012-10-17",
      "Statement": [
        {
          "Effect": "Allow",
          "Action": [
            "kms:CreateGrant",
            "kms:ListGrants",
            "kms:RevokeGrant"
          ],
          "Resource": ["arn:aws:kms:us-
east-1:123456789012:key/1234abcd-12ab-34cd-56ef-1234567890ab"],
          "Condition": {
            "Bool": {
              "kms:GrantIsForAWSResource": "true"
            }
          }
        },
        {
          "Effect": "Allow",
          "Action": [
            "kms:Encrypt",
            "kms:Decrypt",
            "kms:ReEncrypt*",
            "kms:GenerateDataKey*",
            "kms:DescribeKey"
          ],
          "Resource": ["arn:aws:kms:us-
east-1:123456789012:key/1234abcd-12ab-34cd-56ef-1234567890ab"]
        }
      ]
    }
```

   b. ポリシーを作成します。*KMS_Key_For_Encryption_On_EBS_Policy* を別の名前に変更できます。ただし、変更する場合は、必ず後の手順で変更してください。

```
aws iam create-policy \
```

```
      --policy-name KMS_Key_For_Encryption_On_EBS_Policy \
      --policy-document file://kms-key-for-encryption-on-ebs.json
```

c. 次のコマンドを使用して、IAM ポリシーをロールに添付します。*111122223333* は、ご自分のアカウント ID に置き換えます。

```
aws iam attach-role-policy \
      --policy-arn arn:aws:iam::111122223333:policy/
KMS_Key_For_Encryption_On_EBS_Policy \
      --role-name AmazonEKS_EBS_CSI_DriverRole
```

Amazon EBS CSI ドライバーの IAM ロールを作成したので、次のセクションに進むことができます。この IAM ロールを使用してアドオンをデプロイすると、`ebs-csi-controller-sa` という名前のサービスアカウントが作成され、それを使用されるように設定されます。サービスアカウントは Kubernetes `clusterrole` にバインドされます。これには、必要な Kubernetes アクセス許可が割り当てられています。

ステップ 2: Amazon EBS CSI ドライバーを入手する

セキュリティを強化し、作業量を削減するために、Amazon EKS アドオンを通じて Amazon EBS CSI ドライバーをインストールすることをお勧めします。Amazon EKS アドオンをクラスターに追加するには、the section called “アドオンを作成する” を参照してください。アドオンの詳細については、「the section called “Amazon EKS アドオン”」を参照してください。

> **Important**
>
> Amazon EBS ドライバーを Amazon EKS アドオンとして追加する前に、クラスターにセルフマネージドバージョンのドライバーがインストールされていないことを確認してください。インストールされている場合は、GitHub の「Uninstalling a self-managed Amazon EBS CSI driver」を参照してください。

> **Note**
>
> デフォルトでは、EBS CSI が使用する RBAC ロールには、テイント削除機能をサポートするためにノードを変更するアクセス許可があります。Kubernetes RBAC の制限により、このアクセス許可によってクラスター内の他の任意のノードも変更できます。Helm チャートには、ebs-csi-node サービスアカウントのノード RBAC 変更アクセス許可を無効にするパラ

メータ (node.serviceAccount.disableMutation) があります。有効にすると、テイント削除などのドライバー機能は動作しません。

または、Amazon EBS CSI ドライバーのセルフマネージドインストールが必要な場合は、GitHub の「Installation」を参照してください。

ステップ 3: サンプルアプリケーションをデプロイする

さまざまなサンプルアプリケーションをデプロイし、必要に応じて変更できます。詳細については、GitHub の「Kubernetes Examples」を参照してください。

Kubernetes ボリュームストレージ用の Amazon EC2 インスタンスストア

Amazon EC2 Instance Store CSI ドライバーは、Kubernetes での EC2 インスタンスストアボリュームの使用を可能にするコンテナストレージインターフェイス (CSI) プラグインです。インスタンスストアボリュームは、ホストコンピュータに物理的にアタッチされた一時的なブロックレベルストレージを提供します。このドライバーは、それらの NVMe ストレージボリュームのライフサイクルを管理し、Kubernetes 永続ボリュームとして利用できるようにします。

Amazon EC2 Instance Store CSI ドライバーは、ローカルの Linux オペレーションを通じてバックエンドの NVMe デバイスとやり取りします。AWS API コールは伴わないため、IAM ロールは必要ありません。

考慮事項

> **Important**
>
> Amazon EC2 Instance Store CSI ドライバーをインストールすると、インストール中にエフェメラルディスク上のデータはすべて消去されます。別の CSI ドライバーやボリューム管理システム (LVM、raw ファイルシステム、LUKS など) でエフェメラルディスクを管理している場合は、データが失われないように、Amazon EC2 Instance Store CSI ドライバーをインストールする前にデータをバックアップしてください。

- Amazon EC2 Instance Store CSI ドライバーは EKS オートモードでは機能しません。

- Amazon EC2 インスタンスストアボリュームを Fargate ポッドにマウントすることはできません。
- Amazon EC2 Instance Store CSI ドライバーは、次のインスタンスタイプでは使用できません。C1、C3、C4、C5d、C5ad、C6gd、D2、D3、D3en、DL1、E3、Edge1gd、F1、G2、G3、G4ad、G
- 最新のアドオンバージョンと 1 つ前のバージョンがサポートされます。最新バージョンで見つかったバグや脆弱性の修正は、新しいマイナーバージョンの以前のリリースにバックポートされます。
- インスタンスストレージは一時的なものです。ノードが終了した時点でデータは失われます。重要なデータについては、アプリケーションレベルのレプリケーションまたはバックアップを実施してください。
- ボリューム拡張はサポートされません。
- ワークロード間で帯域幅を公平に共有できるように、I/O スロットリングがすべてのボリュームでデフォルトで有効になっています。詳細については、「the section called “I/O スロットリング”」を参照してください。
- インスタンスを新たに起動すると、同じタイプの古いインスタンスと比較すると、NVMe 識別コマンドレスポンスが更新されています。ツーリングが特定の NVMe 識別値に依存している場合は、ロジックの互換性を確認してください。詳細については、「the section called “新たに起動したインスタンスでの NVMe 名前空間の変更”」を参照してください。

前提条件

作業を開始する前に、次の項目があることを確認します。

- 既存の Amazon EKS クラスター。アドオンの使用可能なすべてのバージョンを表示するには、次の AWS CLI コマンドを実行します。

```
aws eks describe-addon-versions --addon-name aws-ec2-local-instance-store-csi-driver
```

Amazon EC2 Instance Store CSI ドライバーをインストールする

Amazon EC2 Instance Store CSI ドライバーは Amazon EKS のアドオンとして提供されています。ドライバーをインストールすると、クラスターでポッドストレージにインスタンスストアボリュームを使用できるようになります。

ドライバーをインストールするには、Amazon EC2 Instance Store CSI ドライバーアドオンをクラスターに追加します。手順については、「Amazon EKS アドオンを作成する」を参照してください。EKS アドオンの詳細については、「Amazon EKS アドオン」を参照してください。

サンプルアプリケーションをデプロイする

ドライバーをインストールしたら、サンプルアプリケーションをデプロイして、インスタンスストアボリュームがクラスターで正しく機能することを確認できます。さまざまなサンプルアプリケーションをデプロイし、必要に応じて変更できます。

I/O スロットリング

Amazon EC2 インスタンスストア CSI ドライバーには、同じ NVMe コントローラー上のワークロード間で帯域幅を公平に共有できるように I/O スロットリングが組み込まれています。スロットリングは、デフォルトですべてのボリュームで有効になっています。単一のワークロードが、同じノード上の他のワークロードを犠牲にして I/O 帯域幅を不均衡に消費するのを防ぎます。

スロットリングの設定

スロットリングは、2 つのレベルで設定できます。

- StorageClass パラメータ – StorageClass 定義で `throttling` パラメータを `"true"` (デフォルト) または `"false"` に設定します。
- PVC 注釈 – 個々の PersistentVolumeClaims で `lis.csi.aws.com/throttling` 注釈を `"true"`または `"false"` に設定します。これにより、StorageClass 設定が存在していれば上書きされます。

次の表に、StorageClass パラメータと PVC 注釈がどのように相互作用するかを示します。

PVC 注釈	StorageClass パラメータ	結果
未設定	未設定	スロットリングが有効 (デフォルト)
未設定	`"false"`	スロットリングが無効
`"false"`	`"true"` または未設定	スロットリングが無効
`"true"`	`"false"`	スロットリングの有効化

スロットリングされない StorageClass の作成

StorageClass 内のすべてのボリュームでスロットリングを無効にするには:

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: ec2-instance-store-unthrottled
provisioner: lis.csi.aws.com
parameters:
  throttling: "false"
volumeBindingMode: WaitForFirstConsumer
```

単一の PVC のスロットリングの無効化

スロットリングされている StorageClass を使用すると共に、個々の PVC のスロットリングを無効にするには:

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: high-perf-pvc
  annotations:
    lis.csi.aws.com/throttling: "false"
spec:
  accessModes:
    - ReadWriteOnce
  storageClassName: ec2-instance-store-sc
  resources:
    requests:
      storage: 100Gi
```

スロットリングについての考慮事項

- NVMe コントローラーを共有するすべてのワークロードに予測可能なパフォーマンスを提供できるように、スロットリングがデフォルトで有効になっています。
- ボリュームのスロットリングを無効にすると、消費できる I/O 帯域幅が増えますが、同じコントローラーにある他のスロットリングされたボリュームのパフォーマンスに影響が及ぶ可能性があります。
- スロットリングされていないワークロードを専用のノードに分離することを検討してください。

- RAID-0 ボリュームが複数のコントローラーにまたがっている場合、スロットリングはすべてのメンバーパーティションに適用されます。
- スロットリングの状態は、ノードの再起動後も維持されます。

新たに起動したインスタンスでの NVMe 名前空間の変更

ローカルインスタンスストレージを備えた EC2 インスタンスを新たに起動すると、NVMe の識別コントローラーと名前空間識別のコマンドレスポンスが更新されます。この更新により、EC2 ローカルインスタンスストレージで NVMe 名前空間の管理がサポートされます。既に実行されているインスタンスは影響を受けず、値はインスタンスのライフサイクルを通じて安定しています。

フィールド値の更新

新たに起動したインスタンスでは、同じタイプの古いインスタンスと比較すると、次のフィールドが変更されています。

識別コントローラー (**nvme id-ctrl**)

- OACS のビット 3: 0 から 1 に変更
- OAES のビット 8: 0 から 1 に変更
- NN: 1 から 256 以下の値に変更
- TNVMCAP: 0 からローカルストレージの実際のサイズに変更

識別名前空間 (**nvme id-ns**)

- NVMCAP: 0 からローカルストレージの実際のサイズに変更
- NGUID: すべてゼロから有効な NGUID に変更

更新された値は、すべて NVMe 仕様に準拠しています。

変更の範囲

いずれの変更も、EC2 ローカルインスタンスストレージにのみ適用されます。Amazon EBS ボリュームは影響を受けません。EBS ボリュームとローカルインスタンスストレージとを区別するには、次のコマンドを実行します。

```
sudo nvme list
```

出力では、次のようになります。

- `Amazon Elastic Block Store` = EBS ボリューム
- `Amazon EC2 NVMe Instance Storage` = ローカルインスタンスストレージ

NVMe 識別値の取得

`nvme-cli` を使用してこれらの値をクエリするには:

```
# Identify Controller
sudo nvme id-ctrl /dev/nvme1

# Identify Namespace
sudo nvme id-ns /dev/nvme1n1
```

必要なアクション

アクションは必要ありません。これらの変更は、下位互換性があり、NVMe 仕様に準拠しています。ただし、アプリケーションやツーリングが NVMe 識別コマンドの特定の値を検証する場合や特定の値に依存している場合は、ロジックを確認して、新たに起動したインスタンスで更新された値との互換性を確認してください。

Amazon EFS で Elastic File System ストレージを使用する

Amazon Elastic File System (Amazon EFS) は、サーバーレスで伸縮自在なファイルストレージを提供するため、ストレージ容量およびパフォーマンスのプロビジョニングや管理を行うことなくファイルデータを共有できます。Amazon EFS Container Storage Interface (CSI) ドライバー を使用すると、AWS 上で実行されている Kubernetes クラスターが Amazon EFS ファイルシステムを永続ボリュームとしてマウントできます。このトピックでは、Amazon EFS CSI ドライバーを Amazon EKS クラスターにデプロイする方法を示します。

考慮事項

- Amazon EFS CSI ドライバーは、Windows ベースのコンテナイメージと互換性がありません。
- Fargate ノードでは永続ボリュームのために動的プロビジョニングを使用できませんが、静的プロビジョニングは使用できます。

- 動的プロビジョニング には、1.2 以降のドライバーが必要です。サポートされている Amazon EKS クラスターバージョンでバージョン 1.1 のドライバーを使用して、永続ボリュームに静的プロビジョニングを使用できます (「Amazon EKS supported versions」を参照)。
- このドライバーのバージョン 1.3.2 以降では、Amazon EC2 Graviton ベースのインスタンスを含む Arm64 アーキテクチャがサポートされています。
- バージョン 1.4.2 以降のドライバーでは、ファイルシステムのマウントに FIPS を使用できます。
- Amazon EFS のリソースクォータに注意してください。詳細については、Amazon SNS クォータを参照してください。
- バージョン 2.0.0 以降、このドライバーは TLS 接続に `stunnel` から `efs-proxy` を使用するように切り替えました。`efs-proxy` を使用すると、実行中のノードのコア数に 1 を足した数のスレッドが開きます。
- Amazon EFS CSI ドライバーは Amazon EKS Hybrid Nodes と互換性がありません。

前提条件

- Amazon EFS CSI ドライバーには AWS Identity and Access Management (IAM) のアクセス権限が必要です。
  - AWS では、EKS Pod Identity の使用を推奨しています。詳細については、the section called “EKS Pod Identity のセットアップの概要” を参照してください。
  - サービスアカウントの IAM ロールと、クラスターの IAM OpenID Connect (OIDC) プロバイダーの設定については、the section called “IAM OIDC プロバイダー” で詳しく確認できます。
- ご使用のデバイスまたは AWS CloudShell で、バージョン `2.12.3` 以降、または AWS コマンドラインインターフェイス (AWS CLI) のバージョン `1.27.160` 以降がインストールおよび設定されていること。現在のバージョンを確認するには、`aws --version | cut -d / -f2 | cut -d ' ' -f1` を参照してください。`yum`、`apt-get`、macOS 用の Homebrew などのパッケージマネージャーは、多くの場合 AWS CLI の最新バージョンより数バージョン古くなっています。最新バージョンをインストールするには、「 AWS コマンドラインインターフェイスユーザーガイド」の「インストール」および「aws configure を使用したクイック設定」を参照してください。AWS クラウドシェル にインストールされている AWS CLI バージョンも最新バージョンより数バージョン遅れることがあります。更新するには、「AWS CloudShell ユーザーガイド」の「ホームディレクトリへの AWS CLI のインストール」を参照してください。
- デバイスまたは AWS CloudShell に、`kubectl` コマンドラインツールがインストールされていること。バージョンはご使用のクラスターの Kubernetes バージョンと同じか、1 つ前のマイナーバージョン以前、あるいはそれより新しいバージョンが使用できます。例えば、クラスターの

バージョンが 1.29 である場合、kubectl のバージョン 1.28、1.29、または 1.30 が使用できます。kubectl をインストールまたはアップグレードする方法については、the section called “kubectl と eksctl のセットアップ” を参照してください。

> **Note**
>
> Fargate 上で実行される Pod は、ドライバーの手動インストールステップなしで、Amazon EFS ファイルシステムを自動的にマウントします。

ステップ 1: IAM ロールを作成する

Amazon EFS CSI ドライバーには、ファイルシステムと対話するための IAM アクセス許可が必要です。IAM ロールを作成して、 arn:aws:iam::aws:policy/service-role/AmazonEFSCSIDriverPolicy マネージドポリシーをアタッチします。

> **Note**
>
> Amazon EFS と Amazon S3 ファイルシステムストレージの両方を使用する場合は、AmazonEFSCSIDriverPolicy と AmazonS3FilesCSIDriverPolicy の両方の管理ポリシーを IAM ロールにアタッチする必要があります。Amazon S3 ファイルシステムストレージの詳細については、the section called “Amazon S3 Files” を参照してください。

この手順を実装するには、次のいずれかのツールを使用できます。

- the section called “eksctl”
- the section called “AWS マネジメントコンソール”
- the section called “AWS CLI”

> **Note**
>
> この手順には、ドライバーを Amazon EKS アドオンとして使用するための特定のステップが書かれています。セルフマネージド型インストールについての詳細は、GitHub の「Set up driver permission」を参照してください。

eksctl

Pod Identity を使用する場合

以下のコマンドを実行します。ここでは、eksctl を使用して IAM ロールと、Pod Identity の関連付けを作成しています。my-cluster をクラスター名に置き換えます。AmazonEKS_EFS_CSI_DriverRole を別の名前に置き換えることもできます。

```
export cluster_name=my-cluster
export role_name=AmazonEKS_EFS_CSI_DriverRole
eksctl create podidentityassociation \
    --service-account-name efs-csi-controller-sa \
    --namespace kube-system \
    --cluster $cluster_name \
    --role-name $role_name \
    --permission-policy-arns arn:aws:iam::aws:policy/service-role/
AmazonEFSCSIDriverPolicy
```

サービスアカウントに IAM ロールを使用している場合

次のコマンドを実行して、eksctl を使用して IAM ロールを作成します。my-cluster をクラスター名に置き換えます。AmazonEKS_EFS_CSI_DriverRole を別の名前に置き換えることもできます。

```
export cluster_name=my-cluster
export role_name=AmazonEKS_EFS_CSI_DriverRole
eksctl create iamserviceaccount \
    --name efs-csi-controller-sa \
    --namespace kube-system \
    --cluster $cluster_name \
    --role-name $role_name \
    --role-only \
    --attach-policy-arn arn:aws:iam::aws:policy/service-role/AmazonEFSCSIDriverPolicy \
    --approve
TRUST_POLICY=$(aws iam get-role --output json --role-name $role_name --query
 'Role.AssumeRolePolicyDocument' | \
    sed -e 's/efs-csi-controller-sa/efs-csi-*/' -e 's/StringEquals/StringLike/')
aws iam update-assume-role-policy --role-name $role_name --policy-document
 "$TRUST_POLICY"
```

AWS マネジメントコンソール

次のコマンドを実行して、AWS マネジメントコンソール を使用して IAM ロールを作成します。

1. IAM コンソール (https://console.aws.amazon.com/iam/) を開きます。
2. 左のナビゲーションペインで、[ロール] を選択してください。
3. [ロール] ページで、[ロールの作成] を選択してください。
4. [信頼されたエンティティを選択] ページで、以下の操作を実行します。
   a. EKS Pod Identity を使用する場合
      i. [信頼するエンティティのタイプ] で [AWS サービス] を選択してください。
      ii. [サービスまたはユースケース] のドロップダウンで、EKS を選択します。
      iii. ユースケースセクションで、[EKS - Pod Identity] を選択します。
      iv. [次へ] を選択します。
   b. サービスアカウントに IAM ロールを使用している場合
      i. [信頼されたエンティティの種類] セクションで、[ウェブ アイデンティティ] を選択します。
      ii. [Identity provider] (ID プロバイダー) で、(Amazon EKS の [Overview] (概要) に示されているように) クラスターに [OpenID Connect provider URL] (OpenID Connect プロバイダーの URL) を選択します。
      iii. [対象者] で [`sts.amazonaws.com`] を選択します。
      iv. [次へ] を選択します。
5. [アクセス許可を追加] ページで、以下を実行します。
   a. [フィルタポリシー] ボックスに `AmazonEFSCSIDriverPolicy` と入力します。
   b. 検索で返された `AmazonEFSCSIDriverPolicy` の左にあるチェックボックスを選択します。
   c. [次へ] を選択します。
6. [名前を付けて、レビューし、作成する] ページで、以下の操作を実行します。
   a. [ロール名] に、`AmazonEKS_EFS_CSI_DriverRole` などのロールの一意の名前を入力します。
   b. [タグの追加 (オプション)] で、タグをキーバリューのペアとして添付して、メタデータをロールに追加します。IAM でのタグの使用に関する詳細については『IAM ユーザーガイド』の「IAM リソースにタグを付ける」を参照してください。
   c. [ロールの作成] を選択してください。
7. ロールの作成後

a. EKS Pod Identity を使用する場合

i. Amazon EKS コンソール を開きます。

ii. 左のナビゲーションペインで、[クラスター] を選択して、EKS Pod Identity の関連付けを設定するクラスターの名前を選択します。

iii. アクセス タブを選択します。

iv. [Pod Identity の関連付け] で [作成] を選択します。

v. [IAM ロール] ドロップダウンを選択して、新しく作成したロールを選択します。

vi. [Kubernetes 名前空間] フィールドを選択して、`kube-system` を入力します。

vii.[Kubernetes サービスアカウント] フィールドを選択して、`efs-csi-controller-sa` を入力します。

viii[作成] を選択します。

ix. Pod Identity の関連付けの作成については、「the section called “Pod Identity の関連付けを作成する (AWS コンソール)”」で詳しく確認できます。

b. サービスアカウントに IAM ロールを使用している場合

i. 編集するロールを選択します。

ii. [信頼関係] タブを選択し、続いて [信頼ポリシーの編集] を選択します。

iii. 次の行と似ている行を探してます。

```
"oidc.eks.region-code.amazonaws.com/id/<EXAMPLED539D4633E53DE1B71EXAMPLE>:aud":
 "sts.amazonaws.com"
```

前の行の上に、次の行を追加します。`<region-code>` を、クラスターのある AWS リージョンに置き換えます。`<EXAMPLED539D4633E53DE1B71EXAMPLE>` をクラスターの OIDC プロバイダー ID に置き換えます。

```
"oidc.eks.<region-code>.amazonaws.com/id/
<EXAMPLED539D4633E53DE1B71EXAMPLE>:sub": "system:serviceaccount:kube-system:efs-
csi-*",
```

iv. `Condition` オペレーターを `"StringEquals"` から `"StringLike"` に修正します。

v. [ポリシーの更新] を選択して終了します。

AWS CLI

次のコマンドを実行して、AWS CLI を使用して IAM ロールを作成します。

Pod Identity を使用する場合

1. pods.eks.amazonaws.com サービスに AssumeRole および TagSession アクションを許可する IAM ロールを作成します。
   a. 次の内容を aws-efs-csi-driver-trust-policy-pod-identity.json という名前のファイルにコピーします。

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "AllowEksAuthToAssumeRoleForPodIdentity",
            "Effect": "Allow",
            "Principal": {
                "Service": "pods.eks.amazonaws.com"
            },
            "Action": [
                "sts:AssumeRole",
                "sts:TagSession"
            ]
        }
    ]
}
```

   b. ロールを作成します。my-cluster をクラスター名に置き換えます。AmazonEKS_EFS_CSI_DriverRole を別の名前に置き換えることもできます。

```
export cluster_name=my-cluster
export role_name=AmazonEKS_EFS_CSI_DriverRole
aws iam create-role \
  --role-name $role_name \
  --assume-role-policy-document file://"aws-efs-csi-driver-trust-policy-pod-
identity.json"
```

2. 次のマンドを実行して、必要な AWS マネージドポリシーをロールに添付します。

```
aws iam attach-role-policy \
  --policy-arn arn:aws:iam::aws:policy/service-role/AmazonEFSCSIDriverPolicy \
```

```
  --role-name $role_name
```

3. 次のコマンドを実行して、Pod Identity の関連付けを作成します。<111122223333> をアカウント ID に置き換えます。

```
aws eks create-pod-identity-association --cluster-name $cluster_name --role-arn {arn-
aws}iam::<111122223333>:role/$role_name --namespace kube-system --service-account
 efs-csi-controller-sa
```

4. Pod Identity の関連付けの作成については、「the section called “Pod Identity の関連付けを作成する (AWS コンソール)”」で詳しく確認できます。

サービスアカウントに IAM ロールを使用している場合

1. クラスターの OIDC プロバイダーの URL を表示します。my-cluster をクラスター名に置き換えます。AmazonEKS_EFS_CSI_DriverRole を別の名前に置き換えることもできます。

```
export cluster_name=my-cluster
export role_name=AmazonEKS_EFS_CSI_DriverRole
aws eks describe-cluster --name $cluster_name --query "cluster.identity.oidc.issuer"
 --output text
```

出力例は次のとおりです。

```
https://oidc.eks.<region-code>.amazonaws.com/id/<EXAMPLED539D4633E53DE1B71EXAMPLE>
```

コマンドの出力が None の場合は、「前提条件」を確認してください。

2. IAM ロールを作成して AssumeRoleWithWebIdentity アクションを付与します。

a. 次の内容を aws-efs-csi-driver-trust-policy.json という名前のファイルにコピーします。<111122223333> はご使用のアカウントの ID に置き換えます。<EXAMPLED539D4633E53DE1B71EXAMPLE> と <region-code> を、前のステップで返された値に置き換えます。

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
```

```
        "Federated": "arn:aws:iam::123456789012:oidc-provider/oidc.eks.us-
east-1.amazonaws.com/id/EXAMPLED539D4633E53DE1B71EXAMPLE"
      },
      "Action": "sts:AssumeRoleWithWebIdentity",
      "Condition": {
        "StringLike": {
          "oidc.eks.us-east-1.amazonaws.com/id/
EXAMPLED539D4633E53DE1B71EXAMPLE:sub": "system:serviceaccount:kube-system:efs-csi-
*",
          "oidc.eks.us-east-1.amazonaws.com/id/
EXAMPLED539D4633E53DE1B71EXAMPLE:aud": "sts.amazonaws.com"
        }
      }
    }
  ]
}
```

b. ロールを作成します。

```
aws iam create-role \
  --role-name $role_name \
  --assume-role-policy-document file://"aws-efs-csi-driver-trust-policy.json"
```

3. 次のマンドを実行して、必要な AWS マネージドポリシーをロールに添付します。

```
aws iam attach-role-policy \
  --policy-arn arn:aws:iam::aws:policy/service-role/AmazonEFSCSIDriverPolicy \
  --role-name $role_name
```

ステップ 2: Amazon EFS CSI ドライバーを取得する

Amazon EFS CSI ドライバーは Amazon EKS アドオンを使用してインストールすることをお勧めします。Amazon EKS アドオンをクラスターに追加するには、the section called “アドオンを作成する” を参照してください。アドオンの詳細については、the section called “Amazon EKS アドオン” を参照してください。Amazon EKS アドオンを使用できない場合は、その理由に関する問題を コンテナロードマップ GitHub リポジトリ に送信することをお勧めします。

> **Important**
>
> Amazon EFS ドライバーを Amazon EKS アドオンとして追加する前に、クラスターにセルフマネージドバージョンのドライバーがインストールされていないことを確認してくださ

い。インストールされている場合は、GitHub の「Uninstalling the Amazon EFS CSI Driver」を参照してください。

または、Amazon EFS CSI ドライバーのセルフマネージド型インストールが必要な場合は、GitHub の「Installation」を参照してください。

ステップ 3:Amazon EFS ファイルシステムを作成する

Amazon EFS ファイルシステムを作成するには、「GitHub」の「Amazon EKS 用の Amazon EFS ファイルシステムを作成」を参照してください。

ステップ 4: サンプルアプリケーションをデプロイする

さまざまなサンプルアプリケーションをデプロイし、必要に応じて変更できます。詳細については、「GitHub」の「例」を参照してください。

Amazon EFS CSI ドライバーで、Amazon S3 Filesシステムストレージを使用する

S3 ファイルは、すべての AWS コンピューティングを Amazon S3 のデータに直接接続する共有ファイルシステムです。これにより、データが S3 を離れることなく、完全なファイルシステムのセマンティクスと低レイテンシーのパフォーマンスを備えたファイルとして、すべての S3 データにすばやく直接アクセスできます。つまり、ファイルベースのアプリケーション、エージェント、チームは、既に依存しているツールを使用して、ファイルシステムとして S3 データにアクセスして操作できます。Amazon EFS Container Storage Interface (CSI) ドライバー を使用すると、AWS で実行されている Kubernetes クラスターは、バージョン 3.0.0 以降の永続的ボリュームとして Amazon S3 Filesシステムをマウントできます。このトピックでは、Amazon EKS クラスター上の Amazon S3 Filesシステムを管理するために、Amazon EFS CSI ドライバーを使用する方法について説明します。

考慮事項

- Amazon EFS CSI ドライバーは、Windows ベースのコンテナイメージと互換性がありません。
- EKS Fargate は、S3 ファイルをサポートしていません。
- Amazon EFS CSI ドライバーは Amazon EKS Hybrid Nodes と互換性がありません。

- Amazon EFS CSI ドライバーでの Amazon S3 Filesのサポートは、バージョン 3.0.0 から開始されます。

前提条件

- Amazon EFS CSI ドライバーには AWS Identity and Access Management (IAM) のアクセス権限が必要です。
  - AWS では、EKS Pod Identity の使用を推奨しています。詳細については、the section called “EKS Pod Identity のセットアップの概要” を参照してください。
  - サービスアカウントの IAM ロールと、クラスターの IAM OpenID Connect (OIDC) プロバイダーの設定については、the section called “IAM OIDC プロバイダー” で詳しく確認できます。
- ご使用のデバイスまたは AWS CloudShell で、バージョン `2.12.3` 以降、または AWS コマンドラインインターフェイス (AWS CLI) のバージョン `1.27.160` 以降がインストールおよび設定されていること。現在のバージョンを確認するには、`aws --version | cut -d / -f2 | cut -d ' ' -f1` を参照してください。`yum`、`apt-get`、macOS 用の Homebrew などのパッケージマネージャーは、多くの場合 AWS CLI の最新バージョンより数バージョン古くなっています。最新バージョンをインストールするには、「 AWS コマンドラインインターフェイスユーザーガイド」の「インストール」および「aws configure を使用したクイック設定」を参照してください。AWS クラウドシェル にインストールされている AWS CLI バージョンも最新バージョンより数バージョン遅れることがあります。更新するには、「AWS CloudShell ユーザーガイド」の「ホームディレクトリへの AWS CLI のインストール」を参照してください。
- デバイスまたは AWS CloudShell に、`kubectl` コマンドラインツールがインストールされていること。バージョンはご使用のクラスターの Kubernetes バージョンと同じか、1 つ前のマイナーバージョン以前、あるいはそれより新しいバージョンが使用できます。例えば、クラスターのバージョンが `1.29` である場合、`kubectl` のバージョン `1.28`、`1.29`、または `1.30` が使用できます。`kubectl` をインストールまたはアップグレードする方法については、the section called “kubectl と eksctl のセットアップ” を参照してください。

ステップ 1: IAM ロールを作成する

Amazon EFS CSI ドライバーには、ファイルシステムと対話するための IAM アクセス許可が必要です。EFS CSI ドライバーは、個別の IAM ロールを持つ 2 つのサービスアカウントを使用します。

- `efs-csi-controller-sa` – コントローラーで使用され、`AmazonS3FilesCSIDriverPolicy` と `AmazonS3FilesClientFullAccess` が必要です。

- `efs-csi-node-sa` — ノードデーモンセットで使用されるには、以下が必要です。
  - `AmazonS3FilesClientFullAccess` – S3 ファイルシステム (`s3files:ClientMount`、`s3files:ClientWrite`、`s3files:ClientRootAccess`) にマウントして書き込むアクセス許可を付与します。
  - `AmazonS3ReadOnlyAccess` — S3 バケットから直接読み取りをストリーミングして、スループットを向上できます。
  - `AmazonElasticFileSystemsUtils` — マウント操作の可視性とトラブルシューティングの容易性を向上させるため、efs-utils のログを Amazon CloudWatch に公開できるようにします。

> **Note**
>
> Amazon S3 Filesシステムと Amazon EFS ストレージの両方を使用する場合は、`AmazonS3FilesCSIDriverPolicy` と `AmazonEFSCSIDriverPolicy` の両方のマネージドポリシーをコントローラーロールにアタッチする必要があります。Amazon EBS ストレージの詳細については、the section called “Amazon EFS” を参照してください。

この手順を実装するには、次のいずれかのツールを使用できます。

- the section called “eksctl”
- the section called “AWS マネジメントコンソール”
- the section called “AWS CLI”

> **Note**
>
> この手順には、ドライバーを Amazon EKS アドオンとして使用するための特定のステップが書かれています。セルフマネージド型インストールについての詳細は、GitHub の「Set up driver permission」を参照してください。

`eksctl`

Pod Identity を使用する場合

以下のコマンドを実行します。ここでは、`eksctl` を使用して IAM ロールと、Pod Identity の関連付けを作成しています。`#######` をご自分の値に置き換えます。

```
export cluster_name=my-cluster

# Create the controller role
eksctl create podidentityassociation \
    --service-account-name efs-csi-controller-sa \
    --namespace kube-system \
    --cluster $cluster_name \
    --role-name AmazonEKS_EFS_CSI_ControllerRole \
    --permission-policy-arns arn:aws:iam::aws:policy/service-role/
AmazonS3FilesCSIDriverPolicy,arn:aws:iam::aws:policy/AmazonS3FilesClientFullAccess

# Create the node role
eksctl create podidentityassociation \
    --service-account-name efs-csi-node-sa \
    --namespace kube-system \
    --cluster $cluster_name \
    --role-name AmazonEKS_EFS_CSI_NodeRole \
    --permission-policy-arns arn:aws:iam::aws:policy/
AmazonS3FilesClientFullAccess,arn:aws:iam::aws:policy/
AmazonS3ReadOnlyAccess,arn:aws:iam::aws:policy/AmazonElasticFileSystemsUtils
```

サービスアカウントに IAM ロールを使用している場合

次のコマンドを実行して、eksctl を使用して IAM ロールを作成します。*my-cluster* はクラスター名に、*region-code* を AWS のリージョンコードに置き換えます。

```
export cluster_name=my-cluster
export region_code=region-code

# Create the controller role
export controller_role_name=AmazonEKS_EFS_CSI_ControllerRole
eksctl create iamserviceaccount \
    --name efs-csi-controller-sa \
    --namespace kube-system \
    --cluster $cluster_name \
    --role-name $controller_role_name \
    --attach-policy-arn arn:aws:iam::aws:policy/service-role/
AmazonS3FilesCSIDriverPolicy \
    --attach-policy-arn arn:aws:iam::aws:policy/AmazonS3FilesClientFullAccess \
    --approve \
    --region $region_code

# Create the node role
```

```
export node_role_name=AmazonEKS_EFS_CSI_NodeRole
eksctl create iamserviceaccount \
    --name efs-csi-node-sa \
    --namespace kube-system \
    --cluster $cluster_name \
    --role-name $node_role_name \
    --attach-policy-arn arn:aws:iam::aws:policy/AmazonS3FilesClientFullAccess \
    --attach-policy-arn arn:aws:iam::aws:policy/AmazonS3ReadOnlyAccess \
    --attach-policy-arn arn:aws:iam::aws:policy/AmazonElasticFileSystemsUtils \
    --approve \
    --region $region_code
```

AWS マネジメントコンソール

次のコマンドを実行して、AWS マネジメントコンソール を使用して IAM ロールを作成します。

1. IAM コンソール (https://console.aws.amazon.com/iam/) を開きます。
2. 左のナビゲーションペインで、[ロール] を選択してください。
3. [ロール] ページで、[ロールの作成] を選択してください。
4. [信頼されたエンティティを選択] ページで、以下の操作を実行します。
   a. EKS Pod Identity を使用する場合
      i. [信頼するエンティティのタイプ] で [AWS サービス] を選択してください。
      ii. [サービスまたはユースケース] のドロップダウンで、EKS を選択します。
      iii. ユースケースセクションで、[EKS - Pod Identity] を選択します。
      iv. [次へ] を選択します。
   b. サービスアカウントに IAM ロールを使用している場合
      i. [信頼されたエンティティの種類] セクションで、[ウェブ アイデンティティ] を選択します。
      ii. [Identity provider] (ID プロバイダー) で、(Amazon EKS の [Overview] (概要) に示されているように) クラスターに [OpenID Connect provider URL] (OpenID Connect プロバイダーの URL) を選択します。
      iii. [対象者] で [`sts.amazonaws.com`] を選択します。
      iv. [次へ] を選択します。
5. [アクセス許可を追加] ページで、以下を実行します。
   a. [フィルタポリシー] ボックスに `AmazonS3FilesCSIDriverPolicy` と入力します。検索結果に表示されたポリシーの左側にあるチェックボックスを選択します。次に、`AmazonS3FilesClientFullAccess` を検索し、選択します。

b. [次へ] を選択します。

6. [名前を付けて、レビューし、作成する] ページで、以下の操作を実行します。

a. [ロール名] に、AmazonEKS_EFS_CSI_ControllerRole などのロールの一意の名前を入力します。

b. [タグの追加 (オプション)] で、タグをキーバリューのペアとして添付して、メタデータをロールに追加します。IAM でのタグの使用に関する詳細については『IAM ユーザーガイド』の「IAM リソースにタグを付ける」を参照してください。

c. [ロールの作成] を選択してください。

7. ロールの作成後

a. EKS Pod Identity を使用する場合

i. Amazon EKS コンソール を開きます。

ii. 左のナビゲーションペインで、[クラスター] を選択して、EKS Pod Identity の関連付けを設定するクラスターの名前を選択します。

iii. アクセス タブを選択します。

iv. [Pod Identity の関連付け] で [作成] を選択します。

v. [IAM ロール] ドロップダウンを選択して、新しく作成したロールを選択します。

vi. [Kubernetes 名前空間] フィールドを選択して、kube-system を入力します。

vii.[Kubernetes サービスアカウント] フィールドを選択して、efs-csi-controller-sa を入力します。

viii[作成] を選択します。

ix. Pod Identity の関連付けの作成については、the section called “Pod Identity の関連付けを作成する (AWS コンソール)” で詳しく確認できます。

x. 上記の手順を繰り返して、ノードサービスアカウントの 2 つ目の役割を作成します。[アクセス許可を追加] ページで、代わりに AmazonS3FilesClientFullAccess、AmazonS3ReadOnlyAccess および AmazonElasticFileSystemsUtils をアタッチします。次に、Kubernetes サービスアカウント フィールドの efs-csi-node-sa との Pod Identity 関連付けを作成します。

b. サービスアカウントに IAM ロールを使用している場合

i. 編集するロールを選択します。

ii. [信頼関係] タブを選択し、続いて [信頼ポリシーの編集] を選択します。

iii. 次の行と似ている行を探してます。

```
"oidc.eks.region-code.amazonaws.com/id/<EXAMPLED539D4633E53DE1B71EXAMPLE>:aud":
 "sts.amazonaws.com"
```

前の行の上に、次の行を追加します。<region-code> を、クラスターのある AWS リージョンに置き換えます。<EXAMPLED539D4633E53DE1B71EXAMPLE> をクラスターの OIDC プロバイダー ID に置き換えます。

```
"oidc.eks.<region-code>.amazonaws.com/id/
<EXAMPLED539D4633E53DE1B71EXAMPLE>:sub": "system:serviceaccount:kube-system:efs-
csi-controller-sa",
```

iv. [ポリシーの更新] を選択して終了します。

v. 上記の手順を繰り返して、ノードサービスアカウントの 2 つ目の役割を作成します。[アクセス許可を追加] ページで、代わりに AmazonS3FilesClientFullAccess、AmazonS3ReadOnlyAccess および AmazonElasticFileSystemsUtils をアタッチします。信頼ポリシーで、:sub 条件値に efs-csi-node-sa を使用します。

AWS CLI

次のコマンドを実行して、AWS CLI を使用して IAM ロールを作成します。

Pod Identity を使用する場合

1. pods.eks.amazonaws.com サービスに AssumeRole および TagSession アクションを許可する IAM ロールを作成します。

   a. 次の内容を aws-efs-csi-driver-trust-policy-pod-identity.json という名前のファイルにコピーします。

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "AllowEksAuthToAssumeRoleForPodIdentity",
            "Effect": "Allow",
            "Principal": {
                "Service": "pods.eks.amazonaws.com"
            },
```

```
            "Action": [
                "sts:AssumeRole",
                "sts:TagSession"
            ]
        }
    ]
}
```

b. ロールを作成します。`#######` の部分は自分のクラスター名に置き換えます。

```
export cluster_name=my-cluster
export controller_role_name=AmazonEKS_EFS_CSI_ControllerRole
aws iam create-role \
  --role-name $controller_role_name \
  --assume-role-policy-document file://"aws-efs-csi-driver-trust-policy-pod-
identity.json"
```

2. コントローラーロールに、必要な AWS マネージドポリシーをアタッチします。

```
aws iam attach-role-policy \
  --policy-arn arn:aws:iam::aws:policy/service-role/AmazonS3FilesCSIDriverPolicy \
  --role-name $controller_role_name

aws iam attach-role-policy \
  --policy-arn arn:aws:iam::aws:policy/AmazonS3FilesClientFullAccess \
  --role-name $controller_role_name
```

3. 同じ信頼ポリシーを使用して、ノードの IAM ロールを作成します。

```
export node_role_name=AmazonEKS_EFS_CSI_NodeRole
aws iam create-role \
  --role-name $node_role_name \
  --assume-role-policy-document file://"aws-efs-csi-driver-trust-policy-pod-
identity.json"
```

4. ノードロールに、必要な AWS マネージドポリシーをアタッチします。

```
aws iam attach-role-policy \
  --policy-arn arn:aws:iam::aws:policy/AmazonS3FilesClientFullAccess \
  --role-name $node_role_name

aws iam attach-role-policy \
  --policy-arn arn:aws:iam::aws:policy/AmazonS3ReadOnlyAccess \
```

```
  --role-name $node_role_name

aws iam attach-role-policy \
  --policy-arn arn:aws:iam::aws:policy/AmazonElasticFileSystemsUtils \
  --role-name $node_role_name
```

5. 次のコマンドを実行して、Pod Identity の関連付けを作成します。<111122223333> は、ご自分のアカウント ID に置き換えます。

```
aws eks create-pod-identity-association --cluster-name $cluster_name --role-arn {arn-aws}iam::<111122223333>:role/$controller_role_name --namespace kube-system --service-account efs-csi-controller-sa
aws eks create-pod-identity-association --cluster-name $cluster_name --role-arn
 {arn-aws}iam::<111122223333>:role/$node_role_name --namespace kube-system --service-account efs-csi-node-sa
```

6. Pod Identity の関連付けの作成については、the section called “Pod Identity の関連付けを作成する (AWS コンソール)” で詳しく確認できます。

サービスアカウントに IAM ロールを使用している場合

1. クラスターの OIDC プロバイダーの URL を表示します。####### の部分は自分のクラスター名に置き換えます。

```
export cluster_name=my-cluster
aws eks describe-cluster --name $cluster_name --query "cluster.identity.oidc.issuer"
 --output text
```

   出力例は次のとおりです。

```
https://oidc.eks.<region-code>.amazonaws.com/id/<EXAMPLED539D4633E53DE1B71EXAMPLE>
```

   コマンドの出力が None の場合は、「前提条件」を確認してください。

2. コントローラーサービスアカウントの IAM ロールを作成します。

   a. 次の内容を controller-trust-policy.json という名前のファイルにコピーします。<111122223333> は、ご自分のアカウント ID に置き換えます。<EXAMPLED539D4633E53DE1B71EXAMPLE> と <region-code> を、前のステップで返された値に置き換えます。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "Federated": "arn:aws:iam::<111122223333>:oidc-provider/
oidc.eks.<region-code>.amazonaws.com/id/<EXAMPLED539D4633E53DE1B71EXAMPLE>"
            },
            "Action": "sts:AssumeRoleWithWebIdentity",
            "Condition": {
                "StringEquals": {
                    "oidc.eks.<region-code>.amazonaws.com/id/
<EXAMPLED539D4633E53DE1B71EXAMPLE>:aud": "sts.amazonaws.com",
                    "oidc.eks.<region-code>.amazonaws.com/id/
<EXAMPLED539D4633E53DE1B71EXAMPLE>:sub": "system:serviceaccount:kube-system:efs-
csi-controller-sa"
                }
            }
        }
    ]
}
```

b. ロールを作成します。

```
export controller_role_name=AmazonEKS_EFS_CSI_ControllerRole
aws iam create-role \
  --role-name $controller_role_name \
  --assume-role-policy-document file://"controller-trust-policy.json"
```

3. コントローラーロールに、必要な AWS マネージドポリシーをアタッチします。

```
aws iam attach-role-policy \
  --policy-arn arn:aws:iam::aws:policy/service-role/AmazonS3FilesCSIDriverPolicy \
  --role-name $controller_role_name

aws iam attach-role-policy \
  --policy-arn arn:aws:iam::aws:policy/AmazonS3FilesClientFullAccess \
  --role-name $controller_role_name
```

4. ノードサービスアカウントの IAM ロールを作成します。

a. 次の内容を node-trust-policy.json という名前のファイルにコピーします。<111122223333> は、ご自分のアカウント ID に置き換えます。<EXAMPLED539D4633E53DE1B71EXAMPLE> と <region-code> を、前のステップで返された値に置き換えます。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "Federated": "arn:aws:iam::<111122223333>:oidc-provider/
oidc.eks.<region-code>.amazonaws.com/id/<EXAMPLED539D4633E53DE1B71EXAMPLE>"
            },
            "Action": "sts:AssumeRoleWithWebIdentity",
            "Condition": {
                "StringEquals": {
                    "oidc.eks.<region-code>.amazonaws.com/id/
<EXAMPLED539D4633E53DE1B71EXAMPLE>:sub": "system:serviceaccount:kube-system:efs-
csi-node-sa",
                    "oidc.eks.<region-code>.amazonaws.com/id/
<EXAMPLED539D4633E53DE1B71EXAMPLE>:aud": "sts.amazonaws.com"
                }
            }
        }
    ]
}
```

b. ロールを作成します。

```
export node_role_name=AmazonEKS_EFS_CSI_NodeRole
aws iam create-role \
  --role-name $node_role_name \
  --assume-role-policy-document file://"node-trust-policy.json"
```

5. ノードロールに、必要な AWS マネージドポリシーをアタッチします。

```
aws iam attach-role-policy \
  --policy-arn arn:aws:iam::aws:policy/AmazonS3FilesClientFullAccess \
  --role-name $node_role_name

aws iam attach-role-policy \
```

```
  --policy-arn arn:aws:iam::aws:policy/AmazonS3ReadOnlyAccess \
  --role-name $node_role_name

aws iam attach-role-policy \
  --policy-arn arn:aws:iam::aws:policy/AmazonElasticFileSystemsUtils \
  --role-name $node_role_name
```

> **Note**
>
> この AmazonS3ReadOnlyAccess ポリシーは、すべての S3 バケットへの読み取りアクセスを許可します。特定のバケットへのアクセスを制限するには、バケットをデタッチし、タグベースのインラインポリシーに置き換えます。詳細については、GitHub の「Amazon EFS CSI ドライバー IAM ポリシードキュメント」を参照してください。

ステップ 2: Amazon EFS CSI ドライバーを取得する

Amazon EFS CSI ドライバーは Amazon EKS アドオンを使用してインストールすることをお勧めします。Amazon EKS アドオンをクラスターに追加するには、the section called “アドオンを作成する” を参照してください。アドオンの詳細については、the section called “Amazon EKS アドオン” を参照してください。Amazon EKS アドオンを使用できない場合は、その理由に関する問題を コンテナロードマップ GitHub リポジトリ に送信することをお勧めします。

> **Important**
>
> Amazon EFS ドライバーを Amazon EKS アドオンとして追加する前に、クラスターにセルフマネージドバージョンのドライバーがインストールされていないことを確認してください。インストールされている場合は、GitHub の「Uninstalling the Amazon EFS CSI Driver」を参照してください。

または、Amazon EFS CSI ドライバーのセルフマネージド型インストールが必要な場合は、GitHub の「Installation」を参照してください。

ステップ 3: Amazon S3 ファイルシステムを作成する

Amazon S3 Filesシステムを作成するには、「GitHub」の「Amazon EKS 用の Amazon S3 Filesシステムを作成」を参照してください。

ステップ 4: サンプルアプリケーションをデプロイする

さまざまなサンプルアプリケーションをデプロイし、必要に応じて変更できます。詳細については、「GitHub」の「例」を参照してください。

Amazon FSx for Lustre で高性能なアプリケーションストレージを使用する

Amazon FSx for Lustre コンテナストレージインターフェイス (CSI) ドライバーは、Amazon EKS クラスターが Amazon FSx for Lustre ファイルシステムのライフサイクルを管理できるようにする CSI インターフェイスを提供します。詳細については、「Amazon FSx for Lustre ユーザーガイド」を参照してください。

Amazon FSx for Lustre CSI ドライバーを Amazon EKS クラスターにデプロイし、動作することを確認する方法の詳細は、「the section called “ドライバーをデプロイする”」を参照してください。

FSx for Lustre ドライバーをデプロイする

このトピックでは、FSx for Lustre CSI ドライバーを Amazon EKS クラスターにデプロイし、動作することを確かめる方法について説明します。最新バージョンのドライバーを使用することをお勧めします。利用可能なバージョンについては、GitHub の「CSI Specification Compatibility Matrix」(CSI 仕様互換性マトリックス) を参照してください。

> **Note**
>
> Fargate では、ドライバーはサポートされていません。

使用可能なパラメータの詳細と、ドライバーの機能を示す完全な例については、GitHub の 「FSx for Lustre Container Storage Interface (CSI) driver」プロジェクトを参照してください。

前提条件

- 既存のクラスター。
- Amazon FSx CSI Driver EKS アドオンは、EKS Pod Identity またはサービスアカウントの IAM ロール (IRSA) による認証をサポートしています。EKS Pod Identity を使用するには、FSx CSI Driver アドオンをデプロイする前後に Pod Identity エージェントをインストールします。詳細につ

いては、the section called “エージェントのセットアップ” を参照してください。代わりに IRSA を使用するには、the section called “IAM OIDC プロバイダー” を参照してください。

- ご使用のデバイスまたは AWS CloudShell で、バージョン `2.12.3` 以降、または AWS コマンドラインインターフェイス (AWS CLI) のバージョン `1.27.160` 以降がインストールおよび設定されていること。現在のバージョンを確認するには、`aws --version | cut -d / -f2 | cut -d ' ' -f1` を参照してください。`yum`、`apt-get`、macOS 用の Homebrew などのパッケージマネージャーは、多くの場合 AWS CLI の最新バージョンより数バージョン古くなっています。最新バージョンをインストールするには、「 AWS コマンドラインインターフェイスユーザーガイド」の「インストール」および「aws configure を使用したクイック設定」を参照してください。AWS クラウドシェル にインストールされている AWS CLI バージョンも最新バージョンより数バージョン遅れることがあります。更新するには、「AWS CloudShell ユーザーガイド」の「ホームディレクトリへの AWS CLI のインストール」を参照してください。
- デバイスまたは AWS CloudShell にインストールされている `eksctl` コマンドラインツールのバージョン `0.215.0` 以降。`eksctl` をインストールまたはアップグレードするには、`eksctl` ドキュメントの「インストール」を参照してください。
- デバイスまたは AWS CloudShell に、`kubectl` コマンドラインツールがインストールされていること。バージョンはご使用のクラスターの Kubernetes バージョンと同じか、1 つ前のマイナーバージョン以前、あるいはそれより新しいバージョンが使用できます。例えば、クラスターのバージョンが `1.29` である場合、`kubectl` のバージョン `1.28`、`1.29`、または `1.30` が使用できます。`kubectl` をインストールまたはアップグレードする方法については、the section called “kubectl と eksctl のセットアップ” を参照してください。

ステップ 1: IAM ロールを作成する

Amazon FSx CSI プラグインでは、ユーザーに代わって AWS API の呼び出しを行うための IAM アクセス許可が必要です。

> **Note**
>
> Pod は、IMDS へのアクセスをブロックする場合を除き、IAM ロールに割り当てられたアクセス許可にアクセスできます。詳細については、the section called “ベストプラクティス” を参照してください。

以下の手順は、IAM ロールを作成し、それに AWS マネージドポリシーをアタッチする方法を示しています。

1. IAM ロールを作成し、次のコマンドで AWS マネージドポリシーをアタッチします。my-cluster を、使用するクラスターの名前に置き換えます。このコマンドは IAM ロールを作成して IAM ポリシーをアタッチする AWS CloudFormation スタックをデプロイします。

```
eksctl create iamserviceaccount \
    --name fsx-csi-controller-sa \
    --namespace kube-system \
    --cluster my-cluster \
    --role-name AmazonEKS_FSx_CSI_DriverRole \
    --role-only \
    --attach-policy-arn arn:aws:iam::aws:policy/AmazonFSxFullAccess \
    --approve
```

サービスアカウントが作成されると、数行の出力が表示されます。出力の最後の数行は、次のようになります。

```
[#]  1 task: {
    2 sequential sub-tasks: {
        create IAM role for serviceaccount "kube-system/fsx-csi-controller-sa",
        create serviceaccount "kube-system/fsx-csi-controller-sa",
    } }
[#]  building iamserviceaccount stack "eksctl-my-cluster-addon-iamserviceaccount-
kube-system-fsx-csi-controller-sa"
[#]  deploying stack "eksctl-my-cluster-addon-iamserviceaccount-kube-system-fsx-csi-
controller-sa"
[#]  waiting for CloudFormation stack "eksctl-my-cluster-addon-iamserviceaccount-
kube-system-fsx-csi-controller-sa"
[#]  created serviceaccount "kube-system/fsx-csi-controller-sa"
```

デプロイされた AWS CloudFormation スタックの名前を書き留めます。前述の出力例では、スタックの名前は eksctl-my-cluster-addon-iamserviceaccount-kube-system-fsx-csi-controller-sa です。

Amazon FSx CSI ドライバーの IAM ロールを作成したので、次のセクションに進むことができます。この IAM ロールを使用してアドオンをデプロイすると、fsx-csi-controller-sa という名前のサービスアカウントが作成され、それを使用されるように設定されます。サービスアカウントは Kubernetes clusterrole にバインドされます。これには、必要な Kubernetes アクセス許可が割り当てられています。

ステップ 2: Amazon FSx CSI ドライバーをインストールする

セキュリティを強化し、作業量を削減するために、Amazon EKS アドオンを通じて Amazon FSx CSI ドライバーをインストールすることをお勧めします。Amazon EKS アドオンをクラスターに追加するには、the section called “アドオンを作成する” を参照してください。アドオンの詳細については、the section called “Amazon EKS アドオン” を参照してください。

> **Important**
>
> クラスターに既存の Amazon FSx CSI ドライバーをインストールすると、アドオンのインストールに失敗する可能性があります。EKS 以外の FSx CSI ドライバーが存在するところに Amazon EKS のアドオンバージョンをインストールしようとすると、リソースの競合によりインストールが失敗します。この問題を解消するには、インストール時に `OVERWRITE` フラグを使用します。
>
> ```
> aws eks create-addon --addon-name aws-fsx-csi-driver --cluster-name my-cluster
>  --resolve-conflicts OVERWRITE
> ```

または、Amazon FSx CSI ドライバーのセルフマネージド型インストールが必要な場合は、GitHub の「Installation」を参照してください。

ステップ 3: ストレージクラス、永続的なボリューム要求、サンプルアプリケーションをデプロイする

この手順では、「FSx for Lustre Container Storage Interface (CSI) driver」(FSx for Lustre コンテナストレージインターフェイス (CSI) ドライバー) GitHub リポジトリを使用して、動的にプロビジョニングされた FSx for Lustre ボリュームを使用します。

1. クラスターのセキュリティグループを書き留めます。AWS マネジメントコンソール の [ネットワーク] セクションまたは次の AWS CLI コマンドを使用して確認できます。`my-cluster` を、使用するクラスターの名前に置き換えます。

   ```
   aws eks describe-cluster --name my-cluster --query
    cluster.resourcesVpcConfig.clusterSecurityGroupId
   ```

2. 「Amazon FSx for Lustre ユーザーガイド」の「Amazon VPC セキュリティグループ」に示す基準に従って、Amazon FSx ファイルシステムのセキュリティグループを作成します。[VPC]

で、[ネットワーク] セクションに示されているようにクラスターの VPC を選択します。「Lustre クライアントに関連付けられているセキュリティグループ」には、クラスターセキュリティグループを使用します。アウトバウンドルールをそのままにして、[すべてのトラフィック] を許可することができます。

3. 次のコマンドを使用して、ストレージクラスマニフェストをダウンロードします。

```
curl -O https://raw.githubusercontent.com/kubernetes-sigs/aws-fsx-csi-driver/master/
examples/kubernetes/dynamic_provisioning/specs/storageclass.yaml
```

4. `storageclass.yaml` ファイルの parameters セクションを編集します。各サンプル値は独自の値に置き換えます。

```
parameters:
  subnetId: subnet-0eabfaa81fb22bcaf
  securityGroupIds: sg-068000ccf82dfba88
  deploymentType: PERSISTENT_1
  automaticBackupRetentionDays: "1"
  dailyAutomaticBackupStartTime: "00:00"
  copyTagsToBackups: "true"
  perUnitStorageThroughput: "200"
  dataCompressionType: "NONE"
  weeklyMaintenanceStartTime: "7:09:00"
  fileSystemTypeVersion: "2.12"
```

- **`subnetId`** – Amazon FSx for Lustre ファイルシステムが作成されるサブネット ID。Amazon FSx for Lustre は、すべてのアベイラビリティーゾーンでサポートされているわけではありません。https://console.aws.amazon.com/fsx/ で Amazon FSx for Lustre コンソールを開き、使用するサブネットが、サポートされているアベイラビリティーゾーンにあることを確認します。次のように、サブネットにノードを含めることも、別のサブネットまたは VPC にすることもできます。
  - AWS マネジメントコンソール で [コンピューティング] セクションのノードグループを選択すると、ノードサブネットを確認できます。
  - 指定するサブネットが、ノードがあるサブネットと同じでない場合は、VPC が接続されている必要があり、セキュリティグループで必要なポートが開いていることを確認する必要があります。
- **`securityGroupIds`** – ファイルシステム用に作成したセキュリティグループの ID。
- **`deploymentType`** (オプション) – ファイルシステムのデプロイのタイプ。有効な値は、`SCRATCH_1`、`SCRATCH_2`、`PERSISTENT_1`、および `PERSISTENT_2` です。デプロイのタ

イプの詳細については、「Amazon FSx for Lustre ファイルシステムを作成する」を参照してください。

- 他のパラメータ (オプション) – 他のパラメータについては、GitHub の「Edit StorageClass」(StorageClass の編集) を参照してください。

5. ストレージクラスマニフェストを作成します。

```
kubectl apply -f storageclass.yaml
```

出力例は次のとおりです。

```
storageclass.storage.k8s.io/fsx-sc created
```

6. 永続的なボリューム要求マニフェストをダウンロードします。

```
curl -O https://raw.githubusercontent.com/kubernetes-sigs/aws-fsx-csi-driver/master/
examples/kubernetes/dynamic_provisioning/specs/claim.yaml
```

7. (オプション) claim.yaml ファイルを編集します。ストレージ要件と前のステップで選択した deploymentType に基づいて、1200Gi を次のいずれかの増分値に変更します。

```
storage: 1200Gi
```

- SCRATCH_2 および PERSISTENT – 1.2 TiB、2.4 TiB、または 2.4 TiB を超えると 2.4 TiB の増分。
- SCRATCH_1 – 1.2 TiB、2.4 TiB、3.6 TiB、または 3.6 TiB を超えると 3.6 TiB の増分。

8. 永続的なボリューム要求を作成します。

```
kubectl apply -f claim.yaml
```

出力例は次のとおりです。

```
persistentvolumeclaim/fsx-claim created
```

9. ファイルシステムがプロビジョニングされていることを確認します。

```
kubectl describe pvc
```

出力例は次のとおりです。

```
Name:          fsx-claim
Namespace:     default
StorageClass:  fsx-sc
Status:        Bound
[...]
```

> **Note**
>
> Status は、Pending になる前に 5 ~ 10 分間 Bound と表示されることがあります。Status が Bound になるまで次のステップに進まないでください。Status が 10 分を超えて Pending になっている場合は、Events 内の警告メッセージを問題に対処するための参考として使用します。

10.サンプルアプリケーションをデプロイします。

```
kubectl apply -f https://raw.githubusercontent.com/kubernetes-sigs/aws-fsx-csi-driver/master/examples/kubernetes/dynamic_provisioning/specs/pod.yaml
```

11.サンプルアプリケーションが実行中であることを確認します。

```
kubectl get pods
```

出力例は次のとおりです。

```
NAME      READY   STATUS              RESTARTS   AGE
fsx-app   1/1     Running             0          8s
```

12.ファイルシステムがアプリケーションによって正しくマウントされていることを確認します。

```
kubectl exec -ti fsx-app -- df -h
```

出力例は次のとおりです。

```
Filesystem                 Size  Used Avail Use% Mounted on
overlay                     80G  4.0G   77G   5% /
tmpfs                       64M     0   64M   0% /dev
tmpfs                      3.8G     0  3.8G   0% /sys/fs/cgroup
192.0.2.0@tcp:/abcdef01    1.1T  7.8M  1.1T   1% /data
/dev/nvme0n1p1              80G  4.0G   77G   5% /etc/hosts
```

```
shm                              64M     0   64M   0% /dev/shm
tmpfs                           6.9G   12K  6.9G   1% /run/secrets/kubernetes.io/
serviceaccount
tmpfs                           3.8G     0  3.8G   0% /proc/acpi
tmpfs                           3.8G     0  3.8G   0% /sys/firmware
```

13.データがサンプルアプリケーションによって FSx for Lustre ファイルシステムに書き込まれたことを確認します。

```
kubectl exec -it fsx-app -- ls /data
```

出力例は次のとおりです。

```
out.txt
```

この出力例は、サンプルアプリケーションが `out.txt` ファイルをファイルシステムに正常に書き込んだことを示しています。

> **Note**
>
> クラスターを削除する前に FSx for Lustre ファイルシステムを必ず削除してください。詳細については、FSx for Lustre ユーザーガイドの「リソースをクリーンアップする」を参照してください。

FSx for Lustre のパフォーマンスチューニング

Amazon EKS で FSx for Lustre を使用する場合、ノードの初期化中に Lustre のチューニングを適用するとパフォーマンスを最適化できます。推奨されるアプローチは、起動テンプレートのユーザーデータを使用して、すべてのノードで一貫した設定を確保することです。

チューニングの内容は次のとおりです。

- ネットワークと RPC の最適化
- Lustre モジュールの管理
- LRU (ロックリソースユニット) の調整
- クライアントキャッシュコントロールの設定
- OST および MDC の RPC コントロール

これらのパフォーマンスチューニングを実装する詳細な手順については、以下を参照してください。

- 標準ノード (EFA 以外) のパフォーマンスを最適化するには、「the section called “最適化 (EFA 以外)”」で起動テンプレートのユーザーデータに追加できる完全なスクリプトを参照してください。
- EFA 対応ノードのパフォーマンスを最適化するには、「the section called “最適化 (EFA)”」を参照してください。

ノードでの Amazon FSx for Lustre のパフォーマンスを最適化する (EFA)

このトピックでは、Amazon EKS と Amazon FSx for Lustre を使用して Elastic Fabric Adapter (EFA) のチューニングを設定する方法について説明します。

> **Note**
>
> - FSx for Lustre CSI ドライバーの作成とデプロイの詳細については、「the section called “ドライバーをデプロイする”」を参照してください。
> - EFA を使用しない標準ノードの最適化については、「the section called “最適化 (EFA 以外)”」を参照してください。

ステップ 1. EKS クラスターを作成する

所定の設定ファイルを使用してクラスターを作成します。

```
# Create cluster using efa-cluster.yaml
eksctl create cluster -f efa-cluster.yaml
```

例 `efa-cluster.yaml`:

```
#efa-cluster.yaml

apiVersion: eksctl.io/v1alpha5
kind: ClusterConfig

metadata:
  name: csi-fsx
  region: us-east-1
  version: "1.30"
```

```
iam:
  withOIDC: true

availabilityZones: ["us-east-1a", "us-east-1d"]

managedNodeGroups:
  - name: my-efa-ng
    instanceType: c6gn.16xlarge
    minSize: 1
    desiredCapacity: 1
    maxSize: 1
    availabilityZones: ["us-east-1b"]
    volumeSize: 300
    privateNetworking: true
    amiFamily: Ubuntu2204
    efaEnabled: true
    preBootstrapCommands:
      - |
        #!/bin/bash
        eth_intf="$(/sbin/ip -br -4 a sh | grep $(hostname -i)/ | awk '{print $1}')"
        efa_version=$(modinfo efa | awk '/^version:/ {print $2}' | sed 's/[^0-9.]//g')
        min_efa_version="2.12.1"

        if [[ "$(printf '%s\n' "$min_efa_version" "$efa_version" | sort -V | head -
n1)" != "$min_efa_version" ]]; then
            sudo curl -O https://efa-installer.amazonaws.com/aws-efa-
installer-1.37.0.tar.gz
            tar -xf aws-efa-installer-1.37.0.tar.gz && cd aws-efa-installer
            echo "Installing EFA driver"
            sudo apt-get update && apt-get upgrade -y
            sudo apt install -y pciutils environment-modules libnl-3-dev libnl-
route-3-200 libnl-route-3-dev dkms
            sudo ./efa_installer.sh -y
            modinfo efa
        else
            echo "Using EFA driver version $efa_version"
        fi

        echo "Installing Lustre client"
        sudo wget -O - https://fsx-lustre-client-repo-public-keys.s3.amazonaws.com/fsx-
ubuntu-public-key.asc | gpg --dearmor | sudo tee /usr/share/keyrings/fsx-ubuntu-public-
key.gpg > /dev/null
```

```
        sudo echo "deb [signed-by=/usr/share/keyrings/fsx-ubuntu-public-key.gpg]
 https://fsx-lustre-client-repo.s3.amazonaws.com/ubuntu jammy main" > /etc/apt/
sources.list.d/fsxlustreclientrepo.list
        sudo apt update | tail
        sudo apt install -y lustre-client-modules-$(uname -r) amazon-ec2-utils | tail
        modinfo lustre

        echo "Loading Lustre/EFA modules..."
        sudo /sbin/modprobe lnet
        sudo /sbin/modprobe kefalnd ipif_name="$eth_intf"
        sudo /sbin/modprobe ksocklnd
        sudo lnetctl lnet configure

        echo "Configuring TCP interface..."
        sudo lnetctl net del --net tcp 2> /dev/null
        sudo lnetctl net add --net tcp --if $eth_intf

        # For P5 instance type which supports 32 network cards,
        # by default add 8 EFA interfaces selecting every 4th device (1 per PCI bus)
        echo "Configuring EFA interface(s)..."
        instance_type="$(ec2-metadata --instance-type | awk '{ print $2 }')"
        num_efa_devices="$(ls -1 /sys/class/infiniband | wc -l)"
        echo "Found $num_efa_devices available EFA device(s)"

        if [[ "$instance_type" == "p5.48xlarge" || "$instance_type" ==
 "p5e.48xlarge" ]]; then
           for intf in $(ls -1 /sys/class/infiniband | awk 'NR % 4 == 1'); do
               sudo lnetctl net add --net efa --if $intf --peer-credits 32
          done
        else
        # Other instances: Configure 2 EFA interfaces by default if the instance
 supports multiple network cards,
        # or 1 interface for single network card instances
        # Can be modified to add more interfaces if instance type supports it
            sudo lnetctl net add --net efa --if $(ls -1 /sys/class/infiniband | head -
n1) --peer-credits 32
            if [[ $num_efa_devices -gt 1 ]]; then
               sudo lnetctl net add --net efa --if $(ls -1 /sys/class/infiniband | tail
 -n1) --peer-credits 32
            fi
        fi

        echo "Setting discovery and UDSP rule"
        sudo lnetctl set discovery 1
```

```
        sudo lnetctl udsp add --src efa --priority 0
        sudo /sbin/modprobe lustre

        sudo lnetctl net show
        echo "Added $(sudo lnetctl net show | grep -c '@efa') EFA interface(s)"
```

ステップ 2. ノードグループの作成

EFA 対応ノードグループを作成します。

```
# Create node group using efa-ng.yaml
eksctl create nodegroup -f efa-ng.yaml
```

Important

これらの値をセクション # 5. Mount FSx filesystem で環境に合わせて調整します。

```
FSX_DNS="<your-fsx-filesystem-dns>" # Needs to be adjusted.
MOUNT_NAME="<your-mount-name>" # Needs to be adjusted.
MOUNT_POINT="</your/mount/point>" # Needs to be adjusted.
```

例 `efa-ng.yaml`:

```
apiVersion: eksctl.io/v1alpha5
kind: ClusterConfig

metadata:
  name: final-efa
  region: us-east-1

managedNodeGroups:
  - name: ng-1
    instanceType: c6gn.16xlarge
    minSize: 1
    desiredCapacity: 1
    maxSize: 1
    availabilityZones: ["us-east-1a"]
    volumeSize: 300
    privateNetworking: true
    amiFamily: Ubuntu2204
```

```
    efaEnabled: true
    preBootstrapCommands:
      - |
        #!/bin/bash
        exec 1> >(logger -s -t $(basename $0)) 2>&1


################################################################################################
        #                                      Configuration Section
         #

################################################################################################

        # File System Configuration
        FSX_DNS="<your-fsx-filesystem-dns>" # Needs to be adjusted.
        MOUNT_NAME="<your-mount-name>" # Needs to be adjusted.
        MOUNT_POINT="</your/mount/point>" # Needs to be adjusted.

        # Lustre Tuning Parameters
        LUSTRE_LRU_MAX_AGE=600000
        LUSTRE_MAX_CACHED_MB=64
        LUSTRE_OST_MAX_RPC=32
        LUSTRE_MDC_MAX_RPC=64
        LUSTRE_MDC_MOD_RPC=50

        # File paths
        FUNCTIONS_SCRIPT="/usr/local/bin/lustre_functions.sh"
        TUNINGS_SCRIPT="/usr/local/bin/apply_lustre_tunings.sh"
        SERVICE_FILE="/etc/systemd/system/lustre-tunings.service"

        #EFA
        MIN_EFA_VERSION="2.12.1"

        # Function to check if a command was successful
        check_success() {
            if [ $? -eq 0 ]; then
                echo "SUCCESS: $1"
            else
                echo "FAILED: $1"
                return 1
            fi
        }

        echo "********Starting FSx for Lustre configuration********"
```

```
        # 1. Install Lustre client
        if grep -q '^ID=ubuntu' /etc/os-release; then
            echo "Detected Ubuntu, proceeding with Lustre setup..."
            # Add Lustre repository
            sudo wget -O - https://fsx-lustre-client-repo-public-keys.s3.amazonaws.com/
fsx-ubuntu-public-key.asc | sudo gpg --dearmor | sudo tee /usr/share/keyrings/fsx-
ubuntu-public-key.gpg > /dev/null

            echo "deb [signed-by=/usr/share/keyrings/fsx-ubuntu-public-key.gpg]
 https://fsx-lustre-client-repo.s3.amazonaws.com/ubuntu jammy main" | sudo tee /etc/
apt/sources.list.d/fsxlustreclientrepo.list

            sudo apt-get update
            sudo apt-get install -y lustre-client-modules-$(uname -r)
            sudo apt-get install -y lustre-client
        else
            echo "Not Ubuntu, exiting"
            exit 1
        fi

        check_success "Install Lustre client"

        # Ensure Lustre tools are in the PATH
        export PATH=$PATH:/usr/sbin

        # 2. Apply network and RPC tunings
        echo "********Applying network and RPC tunings********"
        if ! grep -q "options ptlrpc ptlrpcd_per_cpt_max" /etc/modprobe.d/
modprobe.conf; then
            echo "options ptlrpc ptlrpcd_per_cpt_max=64" | sudo tee -a /etc/modprobe.d/
modprobe.conf
            check_success "Set ptlrpcd_per_cpt_max"
        else
            echo "ptlrpcd_per_cpt_max already set in modprobe.conf"
        fi

        if ! grep -q "options ksocklnd credits" /etc/modprobe.d/modprobe.conf; then
            echo "options ksocklnd credits=2560" | sudo tee -a /etc/modprobe.d/
modprobe.conf
            check_success "Set ksocklnd credits"
        else
            echo "ksocklnd credits already set in modprobe.conf"
        fi
```

```
        # 3. Load Lustre modules
        manage_lustre_modules() {
            echo "Checking for existing Lustre modules..."
            if lsmod | grep -q lustre; then
                echo "Existing Lustre modules found."

                # Check for mounted Lustre filesystems
                echo "Checking for mounted Lustre filesystems..."
                if mount | grep -q "type lustre"; then
                    echo "Found mounted Lustre filesystems. Attempting to unmount..."
                    mounted_fs=$(mount | grep "type lustre" | awk '{print $3}')
                    for fs in $mounted_fs; do
                        echo "Unmounting $fs"
                        sudo umount $fs
                        check_success "Unmount filesystem $fs"
                    done
                else
                    echo "No Lustre filesystems mounted."
                fi

                # After unmounting, try to remove modules
                echo "Attempting to remove Lustre modules..."
                sudo lustre_rmmod
                if [ $? -eq 0 ]; then
                    echo "SUCCESS: Removed existing Lustre modules"
                else
                    echo "WARNING: Could not remove Lustre modules. They may still be
in use."
                    echo "Please check for any remaining Lustre processes or mounts."
                    return 1
                fi
            else
                echo "No existing Lustre modules found."
            fi

            echo "Loading Lustre modules..."
            sudo modprobe lustre
            check_success "Load Lustre modules" || exit 1

            echo "Checking loaded Lustre modules:"
            lsmod | grep lustre
        }
```

```
        # Managing Lustre kernel modules
        echo "********Managing Lustre kernel modules********"
        manage_lustre_modules

        # 4. Initializing Lustre networking
        echo "********Initializing Lustre networking********"
        sudo lctl network up
        check_success "Initialize Lustre networking" || exit 1

        # 4.5 EFA Setup and Configuration
        setup_efa() {
            echo "********Starting EFA Setup********"

            # Get interface and version information
            eth_intf="$(/sbin/ip -br -4 a sh | grep $(hostname -i)/ | awk '{print
 $1}')"
            efa_version=$(modinfo efa | awk '/^version:/ {print $2}' | sed 's/[^0-9.]//
g')
            min_efa_version=$MIN_EFA_VERSION

            # Install or verify EFA driver
            if [[ "$(printf '%s\n' "$min_efa_version" "$efa_version" | sort -V | head -
n1)" != "$min_efa_version" ]]; then
                echo "Installing EFA driver..."
                sudo curl -O https://efa-installer.amazonaws.com/aws-efa-
installer-1.37.0.tar.gz
                tar -xf aws-efa-installer-1.37.0.tar.gz && cd aws-efa-installer

                # Install dependencies
                sudo apt-get update && apt-get upgrade -y
                sudo apt install -y pciutils environment-modules libnl-3-dev libnl-
route-3-200 libnl-route-3-dev dkms

                # Install EFA
                sudo ./efa_installer.sh -y
                modinfo efa
            else
                echo "Using existing EFA driver version $efa_version"
            fi
        }

        configure_efa_network() {
            echo "********Configuring EFA Network********"
```

```
            # Load required modules
            echo "Loading network modules..."
            sudo /sbin/modprobe lnet
            sudo /sbin/modprobe kefalnd ipif_name="$eth_intf"
            sudo /sbin/modprobe ksocklnd

            # Initialize LNet
            echo "Initializing LNet..."
            sudo lnetctl lnet configure

            # Configure TCP interface
            echo "Configuring TCP interface..."
            sudo lnetctl net del --net tcp 2> /dev/null
            sudo lnetctl net add --net tcp --if $eth_intf

            # For P5 instance type which supports 32 network cards,
            # by default add 8 EFA interfaces selecting every 4th device (1 per PCI
bus)
            echo "Configuring EFA interface(s)..."
            instance_type="$(ec2-metadata --instance-type | awk '{ print $2 }')"
            num_efa_devices="$(ls -1 /sys/class/infiniband | wc -l)"
            echo "Found $num_efa_devices available EFA device(s)"

            if [[ "$instance_type" == "p5.48xlarge" || "$instance_type" ==
"p5e.48xlarge" ]]; then
                # P5 instance configuration
                for intf in $(ls -1 /sys/class/infiniband | awk 'NR % 4 == 1'); do
                    sudo lnetctl net add --net efa --if $intf --peer-credits 32
                done
            else
                # Standard configuration
                # Other instances: Configure 2 EFA interfaces by default if the
instance supports multiple network cards,
                # or 1 interface for single network card instances
                # Can be modified to add more interfaces if instance type supports it
                sudo lnetctl net add --net efa --if $(ls -1 /sys/class/infiniband |
head -n1) --peer-credits 32
                if [[ $num_efa_devices -gt 1 ]]; then
                    sudo lnetctl net add --net efa --if $(ls -1 /sys/class/infiniband |
tail -n1) --peer-credits 32
                fi
            fi

            # Configure discovery and UDSP
```

```
        echo "Setting up discovery and UDSP rules..."
        sudo lnetctl set discovery 1
        sudo lnetctl udsp add --src efa --priority 0
        sudo /sbin/modprobe lustre

        # Verify configuration
        echo "Verifying EFA network configuration..."
        sudo lnetctl net show
        echo "Added $(sudo lnetctl net show | grep -c '@efa') EFA interface(s)"
    }

    # Main execution
    setup_efa
    configure_efa_network

    # 5. Mount FSx filesystem
    if [ ! -z "$FSX_DNS" ] && [ ! -z "$MOUNT_NAME" ]; then
        echo "********Creating mount point********"
        sudo mkdir -p $MOUNT_POINT
        check_success "Create mount point"

        echo "Mounting FSx filesystem..."
        sudo mount -t lustre ${FSX_DNS}@tcp:/${MOUNT_NAME} ${MOUNT_POINT}
        check_success "Mount FSx filesystem"
    else
        echo "Skipping FSx mount as DNS or mount name is not provided"
    fi

    # 6. Applying Lustre performance tunings
    echo "********Applying Lustre performance tunings********"

    # Get number of CPUs
    NUM_CPUS=$(nproc)

    # Calculate LRU size (100 * number of CPUs)
    LRU_SIZE=$((100 * NUM_CPUS))

    #Apply LRU tunings
    echo "Apply LRU tunings"
    sudo lctl set_param ldlm.namespaces.*.lru_max_age=${LUSTRE_LRU_MAX_AGE}
    check_success "Set lru_max_age"
    sudo lctl set_param ldlm.namespaces.*.lru_size=$LRU_SIZE
    check_success "Set lru_size"
```

```
        # Client Cache Control
        sudo lctl set_param llite.*.max_cached_mb=${LUSTRE_MAX_CACHED_MB}
        check_success "Set max_cached_mb"

        # RPC Controls
        sudo lctl set_param osc.*OST*.max_rpcs_in_flight=${LUSTRE_OST_MAX_RPC}
        check_success "Set OST max_rpcs_in_flight"

        sudo lctl set_param mdc.*.max_rpcs_in_flight=${LUSTRE_MDC_MAX_RPC}
        check_success "Set MDC max_rpcs_in_flight"

        sudo lctl set_param mdc.*.max_mod_rpcs_in_flight=${LUSTRE_MDC_MOD_RPC}
        check_success "Set MDC max_mod_rpcs_in_flight"

        # 7. Verify all tunings
        echo "********Verifying all tunings********"

        # Function to verify parameter value
        verify_param() {
            local param=$1
            local expected=$2
            local actual=$3

            if [ "$actual" == "$expected" ]; then
                echo "SUCCESS: $param is correctly set to $expected"
            else
                echo "WARNING: $param is set to $actual (expected $expected)"
            fi
        }

        echo "Verifying all parameters:"

        # LRU tunings
        actual_lru_max_age=$(lctl get_param -n ldlm.namespaces.*.lru_max_age | head -1)
        verify_param "lru_max_age" "600000" "$actual_lru_max_age"

        actual_lru_size=$(lctl get_param -n ldlm.namespaces.*.lru_size | head -1)
        verify_param "lru_size" "$LRU_SIZE" "$actual_lru_size"

        # Client Cache
        actual_max_cached_mb=$(lctl get_param -n llite.*.max_cached_mb | grep
"max_cached_mb:" | awk '{print $2}')
        verify_param "max_cached_mb" "64" "$actual_max_cached_mb"
```

```
        # RPC Controls
        actual_ost_rpcs=$(lctl get_param -n osc.*OST*.max_rpcs_in_flight | head -1)
        verify_param "OST max_rpcs_in_flight" "32" "$actual_ost_rpcs"

        actual_mdc_rpcs=$(lctl get_param -n mdc.*.max_rpcs_in_flight | head -1)
        verify_param "MDC max_rpcs_in_flight" "64" "$actual_mdc_rpcs"

        actual_mdc_mod_rpcs=$(lctl get_param -n mdc.*.max_mod_rpcs_in_flight | head -1)
        verify_param "MDC max_mod_rpcs_in_flight" "50" "$actual_mdc_mod_rpcs"

        # Network and RPC configurations from modprobe.conf
        actual_ptlrpc=$(grep "ptlrpc ptlrpcd_per_cpt_max" /etc/modprobe.d/modprobe.conf
| awk '{print $3}')
        verify_param "ptlrpcd_per_cpt_max" "ptlrpcd_per_cpt_max=64" "$actual_ptlrpc"

        actual_ksocklnd=$(grep "ksocklnd credits" /etc/modprobe.d/modprobe.conf | awk
'{print $3}')
        verify_param "ksocklnd credits" "credits=2560" "$actual_ksocklnd"

        # 8. Setup persistence
        setup_persistence() {
            # Create functions file
            cat << EOF > $FUNCTIONS_SCRIPT
        #!/bin/bash

        apply_lustre_tunings() {
            local NUM_CPUS=\$(nproc)
            local LRU_SIZE=\$((100 * NUM_CPUS))

            echo "Applying Lustre performance tunings..."
            lctl set_param ldlm.namespaces.*.lru_max_age=$LUSTRE_LRU_MAX_AGE
            lctl set_param ldlm.namespaces.*.lru_size=\$LRU_SIZE
            lctl set_param llite.*.max_cached_mb=$LUSTRE_MAX_CACHED_MB
            lctl set_param osc.*OST*.max_rpcs_in_flight=$LUSTRE_OST_MAX_RPC
            lctl set_param mdc.*.max_rpcs_in_flight=$LUSTRE_MDC_MAX_RPC
            lctl set_param mdc.*.max_mod_rpcs_in_flight=$LUSTRE_MDC_MOD_RPC
        }
        EOF

            # Create tuning script
            cat << EOF > $TUNINGS_SCRIPT
        #!/bin/bash
        exec 1> >(logger -s -t \$(basename \$0)) 2>&1
```

```
        source $FUNCTIONS_SCRIPT

        # Function to check if Lustre is mounted
        is_lustre_mounted() {
            mount | grep -q "type lustre"
        }

        # Function to mount Lustre
        mount_lustre() {
            echo "Mounting Lustre filesystem..."
            mkdir -p $MOUNT_POINT
            mount -t lustre ${FSX_DNS}@tcp:/${MOUNT_NAME} $MOUNT_POINT
            return \$?
        }

        # Main execution
        # Try to mount if not already mounted
        if ! is_lustre_mounted; then
            echo "Lustre filesystem not mounted, attempting to mount..."
            mount_lustre
        fi

        # Wait for successful mount (up to 5 minutes)
        for i in {1..30}; do
            if is_lustre_mounted; then
                echo "Lustre filesystem mounted, applying tunings..."
                apply_lustre_tunings
                exit 0
            fi
            echo "Waiting for Lustre filesystem to be mounted... (attempt $i/30)"
            sleep 10
        done

        echo "Timeout waiting for Lustre filesystem mount"
        exit 1
        EOF

        # Create systemd service

        # Create systemd directory if it doesn't exist
        sudo mkdir -p /etc/systemd/system/

            # Create service file directly for Ubuntu
            cat << EOF > $SERVICE_FILE
```

```
        [Unit]
        Description=Apply Lustre Performance Tunings
        After=network.target remote-fs.target

        [Service]
        Type=oneshot
        ExecStart=/bin/bash -c 'source $FUNCTIONS_SCRIPT && $TUNINGS_SCRIPT'
        RemainAfterExit=yes

        [Install]
        WantedBy=multi-user.target
        EOF


            # Make scripts executable and enable service
            sudo chmod +x $FUNCTIONS_SCRIPT
            sudo chmod +x $TUNINGS_SCRIPT
            systemctl enable lustre-tunings.service
            systemctl start lustre-tunings.service
        }

        echo "********Setting up persistent tuning********"
        setup_persistence

        echo "FSx for Lustre configuration completed."
```

(オプション) ステップ 3. EFA の設定を検証する

ノードに SSH 接続する:

```
# Get instance ID from EKS console or {aws} CLI
ssh -i /path/to/your-key.pem ec2-user@<node-internal-ip>
```

EFA 設定を検証する:

```
sudo lnetctl net show
```

セットアップログを確認する:

```
sudo cat /var/log/cloud-init-output.log
```

`lnetctl net show` の想定される出力例:

```
net:
    - net type: tcp
      ...
    - net type: efa
      local NI(s):
        - nid: xxx.xxx.xxx.xxx@efa
          status: up
```

デプロイ例

a. claim.yaml を作成する

```
#claim.yaml

apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: fsx-claim-efa
spec:
  accessModes:
    - ReadWriteMany
  storageClassName: ""
  resources:
    requests:
      storage: 4800Gi
  volumeName: fsx-pv
```

claim を適用する:

```
kubectl apply -f claim.yaml
```

b. pv.yaml を作成する

<replaceable-placeholders> を更新:

```
#pv.yaml

apiVersion: v1
kind: PersistentVolume
metadata:
  name: fsx-pv
spec:
```

```
  capacity:
    storage: 4800Gi
  volumeMode: Filesystem
  accessModes:
    - ReadWriteMany
  mountOptions:
    - flock
  persistentVolumeReclaimPolicy: Recycle
  csi:
    driver: fsx.csi.aws.com
    volumeHandle: fs-<1234567890abcdef0>
    volumeAttributes:
      dnsname: fs-<1234567890abcdef0>.fsx.us-east-1.amazonaws.com
      mountname: <abcdef01>
```

永続ボリュームを適用する:

```
kubectl apply -f pv.yaml
```

c。pod.yaml を作成する

```
#pod.yaml

apiVersion: v1
kind: Pod
metadata:
  name: fsx-efa-app
spec:
  containers:
  - name: app
    image: amazonlinux:2
    command: ["/bin/sh"]
    args: ["-c", "while true; do dd if=/dev/urandom bs=100M count=20 > data/test_file;
 sleep 10; done"]
    resources:
      requests:
        vpc.amazonaws.com/efa: 1
      limits:
        vpc.amazonaws.com/efa: 1
    volumeMounts:
    - name: persistent-storage
      mountPath: /data
  volumes:
```

```
  - name: persistent-storage
    persistentVolumeClaim:
      claimName: fsx-claim-efa
```

ポッドを適用する:

```
kubectl apply -f pod.yaml
```

追加の検証コマンド

ポッドがマウントされファイルシステムに書き込まれていることを確認します。

```
kubectl exec -ti fsx-efa-app -- df -h | grep data
# Expected output:
# <192.0.2.0>@tcp:/<abcdef01>  4.5T  1.2G  4.5T   1% /data

kubectl exec -ti fsx-efa-app -- ls /data
# Expected output:
# test_file
```

ノードに SSH 接続してトラフィックが EFA を経由していることを確認します。

```
sudo lnetctl net show -v
```

想定される出力に、トラフィックの統計を含む EFA インターフェイスが表示されます。

関連情報

- the section called “ドライバーをデプロイする”
- the section called “最適化 (EFA 以外)”
- Amazon FSx for Lustre のパフォーマンス
- Elastic Fabric Adapter

ノードでの Amazon FSx for Lustre のパフォーマンスを最適化する (EFA 以外)

起動テンプレートのユーザーデータを使用して、ノードの初期化中にチューニングパラメータを適用することで、Amazon FSx for Lustre のパフォーマンスを最適化することができます。

> **Note**
>
> - FSx for Lustre CSI ドライバーの作成とデプロイの詳細については、「the section called “ドライバーをデプロイする”」を参照してください。
> - EFA 対応ノードでパフォーマンスを最適化するには、「the section called “最適化 (EFA)”」を参照してください。

なぜ起動テンプレートのユーザーデータを使用するのか

- ノードの初期化中にチューニングを自動的に適用するため。
- すべてのノードで一貫した設定を確保するため。
- 手動でのノード設定を不要にするため。

スクリプトの概要の例

このトピックで定義されているスクリプト例では、以下の操作を実行します。

`# 1. Install Lustre client`

- Amazon Linux (AL) OS バージョンを自動検出します。
- 適切な Lustre クライアントパッケージをインストールします。

`# 2. Apply network and RPC tunings`

- 並列 RPC 処理用に `ptlrpcd_per_cpt_max=64` を設定します。
- `ksocklnd credits=2560` を設定してネットワークバッファを最適化します。

`# 3. Load Lustre modules`

- 既存の Lustre モジュールがある場合は、安全に削除します。
- 既存のファイルシステムのアンマウントに対処します。
- 新しい Lustre モジュールをロードします。

 # 4. Lustre Network Initialization

- Lustre のネットワーク設定を初期化します。
- 必要なネットワークパラメータを設定します。

 # 5. Mount FSx filesystem

- このセクションでは、お使いの環境向けに値を調整する必要があります。

 # 6. Apply tunings

- LRU (ロックリソースユニット) の調整:
  - `lru_max_age=600000`
  - `lru_size` CPU 数に基づいて計算
- クライアントキャッシュコントロール: `max_cached_mb=64`
- RPC コントロール:
  - OST `max_rpcs_in_flight=32`
  - MDC `max_rpcs_in_flight=64`
  - MDC `max_mod_rpcs_in_flight=50`

 # 7. Verify tunings

- 適用されたすべてのチューニングを検証します。
- 各パラメータの成功または警告を報告します。

 # 8. Setup persistence

- このセクションでも同様にお使いの環境向けに値を調整する必要があります。
- OS のバージョン (AL2023) を自動検出し、どの Systemd サービスを適用するか判断します。
- システムが起動します。
- Systemd が `lustre-tunings` サービスを開始します (`WantedBy=multi-user.target` により)。
- サービスが `apply_lustre_tunings.sh` 実行します。こちらは、
  - ファイルシステムがマウントされているかどうかを確認します。

  - マウントされていない場合、ファイルシステムをマウントします。
  - マウントが完了するまで待機します (最大 5 分)。
  - マウントが完了すると、チューニングパラメータを適用します。
- 設定は、再起動するまでアクティブな状態のままです。
- スクリプトが完成すると、サービスは終了します。
  - Systemd は、サービスを [アクティブ (終了済み)] にマークします。
- 次回の再起動時にプロセスが繰り返されます。

起動テンプレートの作成

1. Amazon EC2 コンソールの https://console.aws.amazon.com/ec2/ を開いてください。
2. [起動テンプレート] を選択します。
3. [起動テンプレートの作成] を選択してください。
4. [高度な詳細] で、[ユーザーデータ] のセクションを見つけます。
5. 以下のスクリプトを貼り付け、必要に応じて更新します。

> **Important**
>
> セクション `# 5. Mount FSx filesystem` と、セクション `# 8. Setup persistence` の `apply_lustre_tunings.sh` の `setup_persistence()` 関数の両方で、お使いの環境のこれらの値を調整します。
>
> ```
> FSX_DNS="<your-fsx-filesystem-dns>" # Needs to be adjusted.
> MOUNT_NAME="<your-mount-name>" # Needs to be adjusted.
> MOUNT_POINT="</your/mount/point>" # Needs to be adjusted.
> ```

```
MIME-Version: 1.0
Content-Type: multipart/mixed; boundary="==MYBOUNDARY=="
--==MYBOUNDARY==
Content-Type: text/x-shellscript; charset="us-ascii"
#!/bin/bash
exec 1> >(logger -s -t $(basename $0)) 2>&1
# Function definitions
check_success() {
```

```
    if [ $? -eq 0 ]; then
        echo "SUCCESS: $1"
    else
        echo "FAILED: $1"
        return 1
    fi
}
apply_tunings() {
    local NUM_CPUS=$(nproc)
    local LRU_SIZE=$((100 * NUM_CPUS))
    local params=(
        "ldlm.namespaces.*.lru_max_age=600000"
        "ldlm.namespaces.*.lru_size=$LRU_SIZE"
        "llite.*.max_cached_mb=64"
        "osc.*OST*.max_rpcs_in_flight=32"
        "mdc.*.max_rpcs_in_flight=64"
        "mdc.*.max_mod_rpcs_in_flight=50"
    )
    for param in "${params[@]}"; do
        lctl set_param $param
        check_success "Set ${param%%=*}"
    done
}
verify_param() {
    local param=$1
    local expected=$2
    local actual=$3

    if [ "$actual" == "$expected" ]; then
        echo "SUCCESS: $param is correctly set to $expected"
    else
        echo "WARNING: $param is set to $actual (expected $expected)"
    fi
}
verify_tunings() {
    local NUM_CPUS=$(nproc)
    local LRU_SIZE=$((100 * NUM_CPUS))
    local params=(
        "ldlm.namespaces.*.lru_max_age:600000"
        "ldlm.namespaces.*.lru_size:$LRU_SIZE"
        "llite.*.max_cached_mb:64"
        "osc.*OST*.max_rpcs_in_flight:32"
        "mdc.*.max_rpcs_in_flight:64"
        "mdc.*.max_mod_rpcs_in_flight:50"
```

```
    )
    echo "Verifying all parameters:"
    for param in "${params[@]}"; do
        name="${param%%:*}"
        expected="${param#*:}"
        actual=$(lctl get_param -n $name | head -1)
        verify_param "${name##*.}" "$expected" "$actual"
    done
}
setup_persistence() {
    # Create functions file
    cat << 'EOF' > /usr/local/bin/lustre_functions.sh
#!/bin/bash
apply_lustre_tunings() {
    local NUM_CPUS=$(nproc)
    local LRU_SIZE=$((100 * NUM_CPUS))

    echo "Applying Lustre performance tunings..."
    lctl set_param ldlm.namespaces.*.lru_max_age=600000
    lctl set_param ldlm.namespaces.*.lru_size=$LRU_SIZE
    lctl set_param llite.*.max_cached_mb=64
    lctl set_param osc.*OST*.max_rpcs_in_flight=32
    lctl set_param mdc.*.max_rpcs_in_flight=64
    lctl set_param mdc.*.max_mod_rpcs_in_flight=50
}
EOF
    # Create tuning script
    cat << 'EOF' > /usr/local/bin/apply_lustre_tunings.sh
#!/bin/bash
exec 1> >(logger -s -t $(basename $0)) 2>&1
# Source the functions
source /usr/local/bin/lustre_functions.sh
# FSx details
FSX_DNS="<your-fsx-filesystem-dns>" # Needs to be adjusted.
MOUNT_NAME="<your-mount-name>" # Needs to be adjusted.
MOUNT_POINT="</your/mount/point>" # Needs to be adjusted.
# Function to check if Lustre is mounted
is_lustre_mounted() {
    mount | grep -q "type lustre"
}
# Function to mount Lustre
mount_lustre() {
    echo "Mounting Lustre filesystem..."
    mkdir -p ${MOUNT_POINT}
```

```
    mount -t lustre ${FSX_DNS}@tcp:/${MOUNT_NAME} ${MOUNT_POINT}
    return $?
}
# Main execution
# Try to mount if not already mounted
if ! is_lustre_mounted; then
    echo "Lustre filesystem not mounted, attempting to mount..."
    mount_lustre
fi
# Wait for successful mount (up to 5 minutes)
for i in {1..30}; do
    if is_lustre_mounted; then
        echo "Lustre filesystem mounted, applying tunings..."
        apply_lustre_tunings
        exit 0
    fi
    echo "Waiting for Lustre filesystem to be mounted... (attempt $i/30)"
    sleep 10
done
echo "Timeout waiting for Lustre filesystem mount"
exit 1
EOF
    # Create systemd service
    cat << 'EOF' > /etc/systemd/system/lustre-tunings.service
[Unit]
Description=Apply Lustre Performance Tunings
After=network.target remote-fs.target
StartLimitIntervalSec=0
[Service]
Type=oneshot
ExecStart=/usr/local/bin/apply_lustre_tunings.sh
RemainAfterExit=yes
Restart=on-failure
RestartSec=30
[Install]
WantedBy=multi-user.target
EOF
    chmod +x /usr/local/bin/lustre_functions.sh
    chmod +x /usr/local/bin/apply_lustre_tunings.sh
    systemctl enable lustre-tunings.service
    systemctl start lustre-tunings.service
}
echo "Starting FSx for Lustre configuration..."
# 1. Install Lustre client
```

```
if grep -q 'VERSION="2"' /etc/os-release; then
    amazon-linux-extras install -y lustre
elif grep -q 'VERSION="2023"' /etc/os-release; then
    dnf install -y lustre-client
fi
check_success "Install Lustre client"
# 2. Apply network and RPC tunings
export PATH=$PATH:/usr/sbin
echo "Applying network and RPC tunings..."
if ! grep -q "options ptlrpc ptlrpcd_per_cpt_max" /etc/modprobe.d/modprobe.conf; then
    echo "options ptlrpc ptlrpcd_per_cpt_max=64" | tee -a /etc/modprobe.d/
modprobe.conf
    echo "options ksocklnd credits=2560" | tee -a /etc/modprobe.d/modprobe.conf
fi
# 3. Load Lustre modules
modprobe lustre
check_success "Load Lustre modules" || exit 1
# 4. Lustre Network Initialization
lctl network up
check_success "Initialize Lustre networking" || exit 1
# 5. Mount FSx filesystem
FSX_DNS="<your-fsx-filesystem-dns>" # Needs to be adjusted.
MOUNT_NAME="<your-mount-name>" # Needs to be adjusted.
MOUNT_POINT="</your/mount/point>" # Needs to be adjusted.
if [ ! -z "$FSX_DNS" ] && [ ! -z "$MOUNT_NAME" ]; then
    mkdir -p $MOUNT_POINT
    mount -t lustre ${FSX_DNS}@tcp:/${MOUNT_NAME} ${MOUNT_POINT}
    check_success "Mount FSx filesystem"
fi
# 6. Apply tunings
apply_tunings
# 7. Verify tunings
verify_tunings
# 8. Setup persistence
setup_persistence
echo "FSx for Lustre configuration completed."
--==MYBOUNDARY==--
```

6. Amazon EKS ノードグループを作成するときは、この起動テンプレートを選択します。詳細については、「the section called “作成”」を参照してください。

関連情報

- the section called “ドライバーをデプロイする”
- the section called “最適化 (EFA)”
- Amazon FSx for Lustre のパフォーマンス

FSx for NetApp ONTAP で高性能なアプリケーションストレージを使用する

NetApp Trident は、コンテナストレージインターフェイス (CSI) 準拠のドライバーを使用して、動的なストレージオーケストレーションを提供します。これにより、Amazon EKS クラスターが Amazon FSx for NetApp ONTAP ファイルシステムによってバックアップされた永続的ボリューム (PV) のライフサイクルを管理できるようになります。Amazon FSx for NetApp ONTAP CSI ドライバーは Amazon EKS Hybrid Nodes と互換性がないことに注意してください。開始するには、NetApp Trident ドキュメントの「Use Trident with Amazon FSx for NetApp ONTAP」を参照してください。

Amazon FSx for NetApp ONTAP は、クラウドでフルマネージド型の ONTAP ファイルシステムを起動して実行できるストレージサービスです。ONTAP は、NetApp のファイルシステムテクノロジーであり、幅広く採用されているデータアクセス機能とデータ管理機能を提供します。FSx for ONTAP によって、オンプレミスの NetApp ファイルシステムの機能、パフォーマンス、および API に、フルマネージド型 AWS サービスの俊敏性、スケーラビリティが加わり、簡素化されます。詳細については、「FSx for ONTAP ユーザーガイド」を参照してください。

> **Important**
>
> Amazon EBS CSI ドライバーと共に Amazon FSx for NetApp ONTAP を使用して EBS ボリュームをプロビジョニングしている場合は、`multipath.conf` ファイルに EBS デバイスを使用しないように指定する必要があります。サポートされている方法については、「Configuration File Blacklist」を参照してください。以下はその例です。
>
> ```
> defaults {
>        user_friendly_names yes
>        find_multipaths no
>      }
>      blacklist {
>        device {
> ```

```
        vendor "NVME"
        product "Amazon Elastic Block Store"
      }
    }
```

Amazon FSx for OpenZFS でデータストレージを使用する

Amazon FSX for OpenZFS は、オンプレミスの ZFS またはその他の Linux ベースのファイルサーバーから AWS にデータを簡単に移動できるようにするフルマネージド型のファイルストレージサービスです。データの移動は、アプリケーションコードやデータの管理方法を変更しなくても実行できます。Amazon FSX for OpenZFS は、オープンソースの OpenZFS ファイルシステム上に構築された、信頼性が高く、スケーラブルで効率的で機能豊富なファイルストレージを提供しています。これらの機能は、フルマネージド型の AWS サービスの俊敏性、スケーラビリティ、およびシンプルさを兼ね備えています。詳細については、「Amazon FSx for OpenZFS ユーザーガイド」を参照してください。

FSx for OpenZFS コンテナストレージインターフェイス (CSI) ドライバーは、Amazon EKS クラスターが FSx for OpenZFS ボリュームのライフサイクルを管理できるようにする CSI インターフェイスを提供します。Amazon FSx for OpenZFS CSI ドライバーは Amazon EKS Hybrid Nodes と互換性がないことに注意してください。FSx for OpenZFS CSI ドライバーを Amazon EKS クラスターにデプロイするには、GitHub の「aws-fsx-openzfs-csi-driver」を参照してください。

Amazon File Cache を使用してレイテンシーを最小化する

Amazon File Cache は、データの保存場所にかかわらず、ファイルデータの処理に使用される AWS のフルマネージド型の高速キャッシュです。Amazon File Cache は、初回アクセス時にデータをキャッシュに自動的にロードし、使用されていないときにデータを解放します。詳細については、Amazon File Cache ユーザーガイドを参照してください。

Amazon File Cache の Container Storage Interface (CSI) ドライバーは、Amazon EKS クラスターが Amazon ファイルキャッシュのライフサイクルを管理できるようにする CSI インターフェイスを提供します。Amazon File Cache の CSI ドライバーは Amazon EKS Hybrid Nodes と互換性がないことに注意してください。Amazon File Cache の CSI ドライバーを Amazon EKS クラスターにデプロイするには、GitHub の「aws-file-cache-csi-driver」を参照してください。

Mountpoint for Amazon S3 CSI ドライバーを使用して Amazon S3 オブジェクトにアクセスする

Mountpoint for Amazon S3 Container Storage Interface (CSI) ドライバーを使用すると、Kubernetes アプリケーションはファイルシステムインターフェイスを介して Amazon S3 オブジェクトにアクセスできるため、アプリケーションコードを変更せずに高い総スループットを実現できます。Mountpoint for Amazon S3 上に構築された CSI ドライバーは、Amazon S3 バケットを Amazon EKS 内のコンテナやセルフマネージド型 Kubernetes クラスターからアクセスできるボリュームとして提供します。

考慮事項

- Mountpoint for Amazon S3 CSI ドライバーは、Windows ベースのコンテナイメージと現時点では互換性がありません。
- Mountpoint for Amazon S3 CSI ドライバーは現在、Amazon EKS Hybrid Nodes と互換性がありません。
- Mountpoint for Amazon S3 CSI ドライバーは、AWS Fargate をサポートしていません。ただし、Amazon EC2 で実行されているコンテナ (Amazon EKS またはカスタム Kubernetes インストール) はサポートされています。
- Mountpoint for Amazon S3 CSI ドライバーは、静的プロビジョニングのみをサポートします。動的プロビジョニング、つまり新しいバケットの作成はサポートされていません。

> **Note**
>
> 静的プロビジョニングとは、`PersistentVolume` オブジェクトの `volumeAttributes` で `bucketName` として指定されている既存の Amazon S3 バケットを使用することです。詳細については、GitHub の「Static Provisioning」を参照してください。

- Mountpoint for Amazon S3 CSI ドライバーでマウントされたボリュームは、すべての POSIX ファイルシステム機能をサポートしているわけではありません。ファイルシステムの動作の詳細については、GitHub の「Mountpoint for Amazon S3 file system behavior」を参照してください。

ドライバーのデプロイ方法の詳細は、「the section called “ドライバーをデプロイする”」を参照してください。ドライバーの削除方法の詳細は、「the section called “ドライバーを削除する”」を参照してください。

Mountpoint for Amazon S3 ドライバーをデプロイする

Mountpoint for Amazon S3 Container Storage Interface (CSI) ドライバー を使用すると、Kubernetes アプリケーションはファイルシステムインターフェイスを介して Amazon S3 オブジェクトにアクセスできるため、アプリケーションコードを変更せずに高い総スループットを実現できます。

こちらの手順では、Mountpoint for Amazon S3 CSI Amazon EKS ドライバーをデプロイする方法を解説します。続行する前に、考慮事項をご確認ください。

前提条件

- クラスターの既存の AWS Identity and Access Management (IAM) OpenID Connect (OIDC) プロバイダー。既に存在しているかどうかを確認する、または作成するには、the section called “IAM OIDC プロバイダー” を参照してください。
- デバイスまたは AWS CloudShell で、AWS CLI のバージョン `2.12.3` 以降がインストールおよび設定されていること。
- デバイスまたは AWS CloudShell に、`kubectl` コマンドラインツールがインストールされていること。バージョンはご使用のクラスターの Kubernetes バージョンと同じか、1 つ前のマイナーバージョン以前、あるいはそれより新しいバージョンが使用できます。例えば、クラスターのバージョンが `1.29` である場合、`kubectl` のバージョン `1.28`、`1.29`、または `1.30` が使用できます。`kubectl` をインストールまたはアップグレードする方法については、the section called “kubectl と eksctl のセットアップ” を参照してください。

ステップ 1: IAM ポリシーを作成する

Mountpoint for Amazon S3 CSI ドライバーには、ファイルシステムとやり取りするための Amazon S3 アクセス許可が必要です。このセクションでは、必要な権限を付与する IAM ポリシーを作成する方法を説明します。

以下のポリシー例は、Mountpoint の IAM アクセス許可に関する推奨事項に従っています。別の方法として、AWS マネージドポリシー AmazonS3FullAccess を使用することもできますが、このマネージドポリシーでは Mountpoint に必要以上のアクセス権限が付与されます。

Mountpoint の推奨アクセス許可の詳細については、GitHub の「Mountpoint IAM permissions」を参照してください。

1. IAM コンソール (https://console.aws.amazon.com/iam/) を開きます。
2. 左のナビゲーションペインで、 [ポリシー] を選択してください。

3. [ポリシー] ページで、[ポリシーの作成] を選択します。
4. [ポリシーエディター] には [JSON] を選択します。
5. 以下の [ポリシー] をコピーして、エディタに貼り付けます。

> **Important**
>
> `amzn-s3-demo-bucket1` を Amazon S3 バケットの名前に置き換えます。

```
{
   "Version":"2012-10-17",
   "Statement": [
        {
            "Sid": "MountpointFullBucketAccess",
            "Effect": "Allow",
            "Action": [
                "s3:ListBucket"
            ],
            "Resource": [
                "arn:aws:s3:::amzn-s3-demo-bucket1"
            ]
        },
        {
            "Sid": "MountpointFullObjectAccess",
            "Effect": "Allow",
            "Action": [
                "s3:GetObject",
                "s3:PutObject",
                "s3:AbortMultipartUpload",
                "s3:DeleteObject"
            ],
            "Resource": [
                "arn:aws:s3:::amzn-s3-demo-bucket1/*"
            ]
        }
   ]
}
```

Amazon S3 Express One Zone ストレージクラスで導入されたディレクトリバケットは、汎用バケットとは異なる認証メカニズムを使用します。`s3:*` アクションではな

く、s3express:CreateSession アクションを使用する必要があります。ディレクトリバケットの詳細については、「Amazon S3 ユーザーガイド」の「ディレクトリバケット」を参照してください。

以下は、ディレクトリバケットに使用する最小特権ポリシーの例です。

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "s3express:CreateSession",
            "Resource": "arn:aws:s3express:us-west-2:111122223333:bucket/amzn-s3-demo-bucket1--usw2-az1--x-s3"
        }
    ]
}
```

6. [次へ] を選択します。
7. [確認と作成] ページで、ポリシーに名前を付けます。このウォークスルーでは、名前に AmazonS3CSIDriverPolicy を使用します。
8. [Create policy] (ポリシーの作成) を選択します。

ステップ 2: IAM ロールを作成する

Mountpoint for Amazon S3 CSI ドライバーには、ファイルシステムとやり取りするための Amazon S3 アクセス許可が必要です。このセクションでは、IAM ロールを作成して、これらの権限を委任する方法を示します。このロールを作成するには、次のいずれかのツールを使用できます。

- the section called “eksctl”
- the section called “AWS マネジメントコンソール”
- the section called “AWS CLI”

> **Note**
>
> IAM ポリシー AmazonS3CSIDriverPolicy は、前のセクションで作成しました。

eksctl

を使用して Mountpoint for Amazon S3 CSI ドライバーの IAM ロールを作成するには**eksctl**

IAM ロールと Kubernetes サービスアカウントを作成するには、以下のコマンドを実行します。これらのコマンドはまた、AmazonS3CSIDriverPolicy IAM ポリシーをロールにアタッチし、Kubernetes サービスアカウント (s3-csi-controller-sa) に IAM ロールの Amazon リソースネーム (ARN) の注釈を付け、Kubernetes サービスアカウント名を IAMロールの信頼ポリシーに追加します。

```
CLUSTER_NAME=my-cluster
REGION=region-code
ROLE_NAME=AmazonEKS_S3_CSI_DriverRole
POLICY_ARN=AmazonEKS_S3_CSI_DriverRole_ARN
eksctl create iamserviceaccount \
    --name s3-csi-driver-sa \
    --namespace kube-system \
    --cluster $CLUSTER_NAME \
    --attach-policy-arn $POLICY_ARN \
    --approve \
    --role-name $ROLE_NAME \
    --region $REGION \
    --role-only
```

AWS マネジメントコンソール

1. IAM コンソール (https://console.aws.amazon.com/iam/) を開きます。
2. 左のナビゲーションペインで、[ロール] を選択してください。
3. [ロール] ページで、[ロールの作成] を選択してください。
4. [信頼されたエンティティを選択] ページで、以下の操作を実行します。
   a. [信頼されたエンティティの種類] セクションで、[ウェブ アイデンティティ] を選択します。
   b. [ID プロバイダー] で、(Amazon EKS の [概要] に示されているように) クラスターに [OpenID Connect プロバイダーの URL] を選択します。

      URL が表示されない場合は、「前提条件」を確認してください。
   c. [対象者] で [sts.amazonaws.com] を選択します。
   d. [次へ] を選択します。
5. [アクセス許可を追加] ページで、以下を実行します。

a. [フィルターポリシー] ボックスに AmazonS3CSIDriverPolicy と入力します。

> **Note**
>
> このポリシーは、前のセクションで作成されました。

b. 検索で返された AmazonS3CSIDriverPolicy 結果の左にあるチェックボックスを選択します。

c. [次へ] を選択します。

6. [名前を付けて、レビューし、作成する] ページで、以下の操作を実行します。

a. [ロール名] に、AmazonEKS_S3_CSI_DriverRole などロールの一意の名前を入力します。

b. [タグの追加 (オプション)] で、タグをキーバリューのペアとして添付して、メタデータをロールに追加します。IAM でのタグの使用に関する詳細については、IAM ユーザーガイドの「IAM リソースにタグを付ける」を参照してください。

c. [ロールの作成] を選択してください。

7. ロールが作成されたら、コンソールでロールを選択して編集用に開きます。

8. [信頼関係] タブを選択し、続いて [信頼ポリシーの編集] を選択します。

9. 次のように表示されます。

```
"oidc.eks.region-code.amazonaws.com/id/EXAMPLED539D4633E53DE1B71EXAMPLE:aud":
 "sts.amazonaws.com"
```

前の行の末尾にカンマを追加し、前の行の後に次の行を追加します。*region-code* を、クラスターのある AWS リージョンに置き換えます。*EXAMPLED539D4633E53DE1B71EXAMPLE* をクラスターの OIDC プロバイダー ID に置き換えます。

```
"oidc.eks.region-code.amazonaws.com/id/EXAMPLED539D4633E53DE1B71EXAMPLE:sub":
 "system:serviceaccount:kube-system:s3-csi-driver-sa"
```

10. Condition 演算子が "StringEquals" に設定されていることを確認します。

11. [ポリシーの更新] を選択して終了します。

AWS CLI

1. クラスターの OIDC プロバイダーの URL を表示します。*my-cluster* の部分は自分のクラスター名に置き換えます。コマンドの出力が None の場合は、「前提条件」を確認してください。

```
aws eks describe-cluster --name my-cluster --query "cluster.identity.oidc.issuer" --
output text
```

出力例は次のとおりです。

```
https://oidc.eks.region-code.amazonaws.com/id/EXAMPLED539D4633E53DE1B71EXAMPLE
```

2. IAM ロールを作成し、Kubernetes サービスアカウントに AssumeRoleWithWebIdentity アクションを付与します。

   a. 以下の内容を aws-s3-csi-driver-trust-policy.json というファイルにコピーします。*111122223333* は、ご自分のアカウント ID に置き換えます。*EXAMPLED539D4633E53DE1B71EXAMPLE* および *region-code* を、前のステップで返された値にそれぞれ置き換えます。

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Federated": "arn:aws:iam::111122223333:oidc-provider/oidc.eks.us-
east-1.amazonaws.com/id/EXAMPLED539D4633E53DE1B71EXAMPLE"
      },
      "Action": "sts:AssumeRoleWithWebIdentity",
      "Condition": {
        "StringEquals": {
          "oidc.eks.us-east-1.amazonaws.com/id/
EXAMPLED539D4633E53DE1B71EXAMPLE:sub": "system:serviceaccount:kube-system:s3-csi-
driver-sa",
          "oidc.eks.us-east-1.amazonaws.com/id/
EXAMPLED539D4633E53DE1B71EXAMPLE:aud": "sts.amazonaws.com"
        }
      }
    }
  ]
}
```

   b. ロールを作成します。*AmazonEKS_S3_CSI_DriverRole* を別の名前に変更できますが、変更する場合は、後の手順でも変更してください。

```
aws iam create-role \
  --role-name AmazonEKS_S3_CSI_DriverRole \
  --assume-role-policy-document file://"aws-s3-csi-driver-trust-policy.json"
```

3. 次のコマンドを使用して IAM ロールを作成し、そのロールに IAM ポリシーをアタッチします。

```
aws iam attach-role-policy \
  --policy-arn arn:aws:iam::aws:policy/AmazonS3CSIDriverPolicy \
  --role-name AmazonEKS_S3_CSI_DriverRole
```

> **Note**
>
> IAM ポリシー AmazonS3CSIDriverPolicy は、前のセクションで作成しました。

4. ドライバーを Amazon EKS アドオンとしてインストールする場合は、この手順をスキップしてください。ドライバーのセルフマネージド型インストールでは、作成した IAM ロールの ARN の注釈が付けられた Kubernetes サービスアカウントを作成します。

   a. 次の内容を mountpoint-s3-service-account.yaml という名前のファイルに保存します。*111122223333* は、ご自分のアカウント ID に置き換えます。

```
---
apiVersion: v1
kind: ServiceAccount
metadata:
  labels:
    app.kubernetes.io/name: aws-mountpoint-s3-csi-driver
  name: s3-csi-driver-sa
  namespace: kube-system
  annotations:
    eks.amazonaws.com/role-arn: arn:aws:iam::111122223333:role/
AmazonEKS_S3_CSI_DriverRole
```

   b. クラスター上で Kubernetes サービスアカウントを作成します。Kubernetes サービスアカウント (s3-csi-driver-sa) は、*AmazonEKS_S3_CSI_DriverRole* という名前で作成した IAM ロールの注釈が付けられています。

```
kubectl apply -f mountpoint-s3-service-account.yaml
```

> **Note**
>
> その手順でプラグインがデプロイされると、`s3-csi-driver-sa` という名前のサービスアカウントを作成して使用するように設定されます。

ステップ 3: Mountpoint for Amazon S3 CSI ドライバーをインストールする

Mountpoint for Amazon S3 CSI ドライバーは、Amazon EKS アドオンを使用してインストールできます。次のツールを使用して、アドオンをクラスターに追加できます。

- the section called “eksctl”
- the section called “AWS マネジメントコンソール”
- the section called “AWS CLI”

または、セルフマネージド型インストールとして Mountpoint for Amazon S3 CSI ドライバーをインストールできます。セルフマネージド型インストールを実行する方法については、GitHub の「Installation」を参照してください。

`v1.8.0` 以降、CSI ドライバーの Pod を許容するようにテイントを設定できます。これを行うには、`node.tolerations` を使用して許容するカスタムのテイントセットを指定するか、`node.tolerateAllTaints` を使用してすべてのテイントを許容します。詳細については、Kubernetes ドキュメントの「テイントと容認」を参照してください。

eksctl

を使用して Amazon S3 CSI アドオンを追加するには**`eksctl`**

以下のコマンドを実行してください。*`my-cluster`* をクラスターの名前に、*`111122223333`* をアカウント ID に、*`AmazonEKS_S3_CSI_DriverRole`* を 前に作成した IAM ロール の名前に置き換えます。

```
eksctl create addon --name aws-mountpoint-s3-csi-driver --cluster my-cluster \
  --service-account-role-arn arn:aws:iam::111122223333:role/AmazonEKS_S3_CSI_DriverRole
 --force
```

`--force` オプションを削除した状態で Amazon EKS アドオンを更新する際に、そのアドオン設定のいずれかが現状の設定と競合している場合、その更新は失敗します。この場合、競合を解決するため

のエラーメッセージが表示されます。このオプションを指定する前に、自分が管理する必要がある設定を Amazon EKS アドオンが管理していないことを確認してください。これらの設定は、このオプションの指定により上書きされます。この設定のその他のオプションの詳細については、eksctl ドキュメントの「アドオン」を参照してください。Amazon EKS Kubernetes フィールド管理の詳細については、the section called “カスタマイズ可能なフィールド” を参照してください。

設定ファイルをによって eksctl をカスタマイズできます。詳細については、eksctl ドキュメントの「Working with configuration values」を参照してください。次の例は、すべてのテイントを許容する方法を示しています。

```
# config.yaml
...

addons:
- name: aws-mountpoint-s3-csi-driver
  serviceAccountRoleARN: arn:aws:iam::111122223333:role/AmazonEKS_S3_CSI_DriverRole
  configurationValues: |-
    node:
      tolerateAllTaints: true
```

AWS マネジメントコンソール

1. Amazon EKS コンソール を開きます。
2. 左のナビゲーションペインで [クラスター] を選択します。
3. Mountpoint for Amazon S3 CSI アドオンを設定するクラスターの名前を選択します。
4. [アドオン] タブを選択してください。
5. [その他のアドオンを入手] を選択してください。
6. [アドオンを選択] ページで、次の操作を行います。
   a. [Amazon EKS アドオン] セクションで、[Mountpoint for Amazon S3 CSI ドライバー] チェックボックスを選択します。
   b. [次へ] を選択します。
7. [選択したアドオンセッティングの設定] ページで、次の操作を行います。
   a. 使用する [バージョン] を選択します。
   b. [IAM ロールを選択] で、Mountpoint for Amazon S3 CSI ドライバーの IAM ポリシーをアタッチした IAM ロールの名前を選択します。
   c. (オプション) [オプション設定] を展開した後、[競合解決方法] を更新します。[上書きする] を選択すると、既存のアドオンでの 1 つ以上の設定が、Amazon EKS アドオンの設定で上書きさ

れます。このオプションが有効でない状態で既存の設定との競合が発生する場合は、オペレーションが失敗します。表示されたエラーメッセージを使用して、競合をトラブルシューティングできます。このオプションを選択する前に、自分で管理する必要のある設定が、Amazon EKS アドオンにより管理されていないことを確認してください。

d. (オプション) [オプション設定] を展開した後、[設定値] フィールドで許容値を設定します。

e. [次へ] を選択します。

8. [確認と追加] ページで、[作成] を選択します。アドオンのインストールが完了した後、インストールしたアドオンが表示されます。

AWS CLI

AWS CLI を使用して Mountpoint for Amazon S3 CSI アドオンを追加するには

以下のコマンドを実行してください。*my-cluster* をクラスターの名前に、*111122223333* をアカウント ID に、*AmazonEKS_S3_CSI_DriverRole* を前に作成したロールの名前に置き換えます。

```
aws eks create-addon --cluster-name my-cluster --addon-name aws-mountpoint-s3-csi-
driver \
  --service-account-role-arn arn:aws:iam::111122223333:role/AmazonEKS_S3_CSI_DriverRole
```

`--configuration-values` フラグを使用してコマンドをカスタマイズできます。次の代替例は、すべてのテイントを許容する方法を示しています。

```
aws eks create-addon --cluster-name my-cluster --addon-name aws-mountpoint-s3-csi-
driver \
  --service-account-role-arn arn:aws:iam::111122223333:role/AmazonEKS_S3_CSI_DriverRole
 \
  --configuration-values '{"node":{"tolerateAllTaints":true}}'
```

ステップ 4: Mountpoint for Amazon S3 を設定する

ほとんどの場合、Mountpoint for Amazon S3 はバケット名のみで設定できます。Mountpoint for Amazon S3 を設定する手順については、GitHub の「Configuring Mountpoint for Amazon S3」を参照してください。

ステップ 5: サンプルアプリケーションをデプロイする

既存の Amazon S3 バケットのドライバーにデプロイできます。詳細については、GitHub の「Static provisioning」を参照してください。

Mountpoint for Amazon S3 Amazon EKS アドオンを削除する

Mountpoint for Amazon S3 CSI ドライバーを削除する方法は 2 通りあります。

- クラスター上のアドオンソフトウェアを保持する — このオプションではすべての設定の Amazon EKS 管理が削除されます。また、Amazon EKS による更新の通知機能や、更新の開始後の Amazon EKS アドオンの自動更新機能も削除されます。ただし、クラスター上のアドオンソフトウェアは保持されます。このオプションを選択すると、アドオンは Amazon EKS アドオンではなく、セルフマネージド型インストールになります。このオプションを使用すると、アドオンのダウンタイムは発生しません。この手順のコマンドには、このオプションを使用します。
- クラスターからアドオンソフトウェアを完全に削除する — クラスターから Amazon EKS アドオンを削除するのはアドオンに依存するリソースがクラスター上にない場合のみとすることをお勧めします。このオプションを実行するには、この手順で使用するコマンドから `--preserve` を削除します。

アドオンに IAM アカウントが関連付けられている場合、その IAM アカウントは削除されません。

次のツールを使用して、Amazon S3 CSI アドオンを削除できます。

- the section called “eksctl”
- the section called “AWS マネジメントコンソール”
- the section called “AWS CLI”

eksctl

`eksctl` を使用して Amazon S3 CSI アドオンを削除するには

`my-cluster` を自分のクラスター名に置き換えた上で、次のコマンドを実行します。

```
eksctl delete addon --cluster my-cluster --name aws-mountpoint-s3-csi-driver --preserve
```

AWS マネジメントコンソール

1. Amazon EKS コンソールを開きます。
2. 左のナビゲーションペインで、 [クラスター] を選択してください。
3. Mountpoint for Amazon S3 CSI アドオンを削除するクラスターの名前を選択します。

4. [アドオン] タブを選択してください。
5. [Mountpoint for Amazon S3 CSI ドライバー] を選択します。
6. [を削除] を選択します。
7. [削除: aws-mountpoint-s3-csi-driver] 確認ダイアログボックスで、以下の操作を行います。
   a. Amazon EKS でのアドオンの設定の管理を停止する場合は[クラスターで保持する] を選択してください。クラスターでアドオンソフトウェアを保持する場合はこの操作を実行してください。これはアドオンのすべての設定を自分で管理できるようにするためです。
   b. `aws-mountpoint-s3-csi-driver` と入力します。
   c. [削除] を選択します。

AWS CLI

AWS CLI を使用して Amazon S3 CSI アドオンを削除するには

`my-cluster` を自分のクラスター名に置き換えた上で、次のコマンドを実行します。

```
aws eks delete-addon --cluster-name my-cluster --addon-name aws-mountpoint-s3-csi-
driver --preserve
```

CSI ボリュームのためにスナップショット機能を有効にする

スナップショット機能を使用すると、データのポイントインタイムコピーが可能になります。Kubernetes でこの機能を使用するには、スナップショットをサポートする CSI ドライバー (Amazon EBS CSI ドライバーなど) と CSI スナップショットコントローラーの両方が必要です。スナップショットコントローラーは、Amazon EKS マネージドアドオンまたはセルフマネージドインストールのいずれかとして使用できます。

CSI スナップショットコントローラーを使用する場合の考慮事項を以下に示します。

- スナップショットコントローラーは、スナップショット機能を備えた CSI ドライバーと一緒にインストールする必要があります。Amazon EBS CSI ドライバーのインストール手順については、「the section called “Amazon EBS”」を参照してください。
- Kubernetes では、プロビジョナー `kubernetes.io/aws-ebs` で `StorageClass` を使用する Amazon EBS ボリュームなど、CSI 移行経由で提供されるボリュームのスナップショットはサポートされていません。ボリュームは CSI ドライバープロビジョナー、`ebs.csi.aws.com` を参照する `StorageClass` を使用して作成する必要があります。

- Amazon EKS Auto Mode にはスナップショットコントローラーは含まれません。EKS Auto Mode のストレージ機能はスナップショットコントローラーと互換性があります。

CSI スナップショットコントローラーは Amazon EKS マネージドアドオンを使用してインストールすることをお勧めします。このアドオンには、Amazon EKS でスナップショットを作成および管理するために必要なカスタムリソース定義 (CRD) が含まれています。Amazon EKS アドオンをクラスターに追加するには、「the section called “アドオンを作成する”」を参照してください。アドオンの詳細については、「the section called “Amazon EKS アドオン”」を参照してください。

または、CSI スナップショットコントローラーの自己管理型インストールを希望する場合は、GitHub でのアップストリーム Kubernetes `external-snapshotter` での「Usage」を参照してください。

Amazon EKS クラスターのネットワーキングを設定する

Amazon EKS クラスターは VPC 内に作成されます。AWS インフラストラクチャ上で実行されるノードに対しては、ポッドネットワークは Amazon VPC Container Network Interface (CNI) プラグインによって提供されます。独自のインフラストラクチャでノードを実行している場合は、「the section called “CNI を設定する”」を参照してください。この章では、クラスターのネットワークの詳細に関する以下のトピックについて説明します。

トピック

- 管理コンソールから既存の VPC サブネットを Amazon EKS クラスターに追加する
- VPC とサブネットの Amazon EKS ネットワーキング要件を表示する
- Amazon EKS クラスターの Amazon VPC を作成する
- クラスターの Amazon EKS セキュリティグループ要件を表示する
- Amazon EKS クラスターのネットワーキングアドオンを管理する

管理コンソールから既存の VPC サブネットを Amazon EKS クラスターに追加する

1. 管理コンソールでクラスターに移動します。
2. [ネットワーキング] タブから [VPC リソースを管理] を選択します。
3. [サブネット] ドロップダウンから、クラスターの VPC からのサブネットを追加で選択します。

新しい VPC サブネットを作成するには:

- EKS サブネットの要件を確認します。
- 「Amazon Virtual Private Cloud ユーザーガイド」の「Create a Subnet」を参照してください。

VPC とサブネットの Amazon EKS ネットワーキング要件を表示する

クラスターを作成する際には、VPC と、異なるアベイラビリティーゾーンに存在する 2 つ以上のサブネットを指定します。このトピックでは、クラスターで使用する VPC およびサブネットに関する Amazon EKS 固有の要件と考慮事項の概要について説明します。Amazon EKS で使用する VPC が

ない場合は、「the section called “「VPC を作成する」”」を参照してください。AWS Outposts でローカルクラスターまたは拡張クラスターを作成する場合は、このトピックの代わりに「the section called “VPC とサブネットを作成する”」を参照してください。このトピックのコンテンツは、ハイブリッドノードを持つ Amazon EKS クラスターに適用されます。ハイブリッドノードのその他のネットワーク要件については、「the section called “ネットワークを準備する”」を参照してください。

VPC の要件と考慮事項

クラスターを作成する際には、指定する VPC が次の要件と考慮事項を満たす必要があります。

- VPC には、作成するクラスター、ノード、およびその他の Kubernetes リソースで利用できる十分な数の IP アドレスが必要です。使用する VPC に十分な数の IP アドレスがない場合は、使用可能な IP アドレスの数を増やしてみてください。

  これを行うには、クラスター設定を更新して、クラスターが使用するサブネットとセキュリティグループを変更します。AWS マネジメントコンソール、AWS CLI の最新バージョン、AWS CloudFormation、および `eksctl` のバージョン `v0.164.0-rc.0` 以降から更新できます。クラスターバージョンを正常にアップグレードするにはこれを実行して、サブネットに利用可能な IP アドレスを増やす必要がある場合があります。

  > **Important**
  >
  > 追加するサブネットはすべて、クラスターの作成時に最初に提供したのと同じ一連の AZ 内にある必要があります。新しいサブネットは、その他のすべての要件 (例えば、十分な IP アドレスを持っている必要がある) を満たす必要があります。
  > 例えば、1 つのクラスターを作成し、4 つのサブネットを指定したとします。指定した順序では、1 番目のサブネットは `us-west-2a` アベイラビリティーゾーンにあり、2 番目と 3 番目のサブネットは `us-west-2b` アベイラビリティーゾーンにあり、4 番目のサブネットは `us-west-2c` アベイラビリティーゾーンにあります。サブネットを変更する場合は、3 つのアベイラビリティーゾーンのそれぞれに少なくとも 1 つのサブネットを指定する必要があります。また、サブネットは元のサブネットと同じ VPC 内にある必要があります。

  VPC 内の CIDR ブロックよりも多くの IP アドレスが必要な場合は、VPC に追加の Classless Inter-Domain Routing (CIDR) ブロックを関連付けることで CIDR ブロックを追加できます。クラスターの作成前または作成後に、プライベート (RFC 1918) とパブリック (非 RFC 1918) の CIDR ブロックを VPC に関連付けることができます。

新しい CIDR ブロックを追加するとすぐにそのブロックを使用するノードを追加することができます。ただし、コントロールプレーンでは調整が完了した後にのみ新しい CIDR ブロックが認識されるため、VPC に関連付けられた CIDR ブロックがクラスターで認識されるまでに最大で 1 時間かかることがあります。その後、新しい CIDR ブロック内のノードとポッドに対して、`kubectl cp`、`kubectl exec`、`kubectl attach`、`kubectl logs`、および `kubectl port-forward` コマンドを実行できます (これらのコマンドは `kubelet API` を使用します)。また、ウェブフックバックエンドとして機能するポッドがある場合は、コントロールプレーンの調整が完了するまで待機する必要があります。

- トランジットゲートウェイ、VPC ピアリング、またはその他のネットワーク設定を介して EKS クラスターを他の VPC に接続するときは、IP アドレス範囲が重複しないようにします。CIDR 競合は、EKS クラスターのサービス CIDR が接続された VPC の CIDR と重複するときに発生します。これらのシナリオでは、Service IP アドレスは同じ IP アドレスを持つ接続された VPC 内のリソースよりも優先されますが、トラフィックルーティングが予測不能になり、アプリケーションが目的のリソースに接続できなくなる可能性があります。

  CIDR の競合を回避するには、EKS サービス CIDR が接続された VPC CIDR と重複しないようにし、すべての CIDR 割り当ての一元化された記録を維持するようにします。CIDR の重複が発生した場合は、共有サービス VPC でトランジットゲートウェイを使用できます。詳細については、「共有サービスによる分離された VPC」および「ハイブリッドネットワークにおける Amazon EKS VPC ルーティング可能な IP アドレス保全パターン」を参照してください。また、EKS ベストプラクティスガイドの「VPC とサブネットに関する考慮事項」ページの「VPC 間の通信」セクションを参照してください。

- Kubernetes で `IPv6` アドレスを Pod およびサービスに割り当てる場合は、`IPv6` CIDR ブロックを VPC に関連付けます。詳細については、Amazon VPC ユーザーガイドの「IPv6 CIDR ブロックと VPC の関連付け」を参照してください。ハイブリッドノードで実行されているポッドやサービスで `IPv6` アドレスを使用することはできません。また、`IPv6` IP アドレスファミリーで設定されたクラスターでハイブリッドノードを使用することはできません。

- VPC は、`DNS` ホスト名と `DNS` 解決がサポートされている必要があります。そうではない場合、ノードはクラスターに登録されません。詳細については、Amazon VPC ユーザーガイドの「DNS attributes for your VPC」(VPC の DNS 属性) を参照してください。

- VPC では、AWS PrivateLink を使用する VPC エンドポイントが必要になる場合があります。詳細については、「the section called “サブネットの要件と考慮事項”」を参照してください。

- クラスターがカスタマールーティングのコントロールプレーンエグレス (`controlPlaneEgressMode=CUSTOMER_ROUTED`) を使用する場合、クラスターで使用されるサブネットは、アドミッションウェブフックサーバーや OIDC プロバイダーなど、コントロールプ

レーンが呼び出すエンドポイントにルーティングできる必要があります。これには、エグレスパス (NAT ゲートウェイ、NAT インスタンス、ファイアウォール、一元化された送信 VPC へのトランジットゲートウェイなど) に加えて、トラフィックを許可するルートテーブル、セキュリティグループ、ネットワーク ACL ルールが必要です。詳細については、「the section called “コントロールプレーンエグレス”」を参照してください。

Kubernetes 1.14 以前でクラスターを作成した場合、Amazon EKS は次のタグを VPC に追加しています。

キー	値
kubernetes.io/cluster/ *my-cluster*	owned

このタグは Amazon EKS でのみ使用されています。サービスに影響を与えずに、このタグを削除できます。このタグは、バージョン 1.15 以降のクラスターでは使用されません。

サブネットの要件と考慮事項

クラスターを作成すると、Amazon EKS は、指定したサブネットに 2～4 つの Elastic Network Interface を作成します。これらのネットワークインターフェイスは、クラスターと VPC 間の通信を可能にします。これらのネットワークインターフェイスでは、kubelet API を使用する Kubernetes の機能も有効化されます。kubelet API への接続は、kubectl attach、kubectl cp、kubectl exec、kubectl logs、kubectl port-forward の各コマンドで使用されます。Amazon EKS が作成した各ネットワークインターフェイスの説明には、テキスト Amazon EKS *cluster-name* が書き込まれます。

これらのリクエスタマネージドネットワークインターフェイスでは、コントロールプレーンとワーカーノードの間のデータプレーントラフィックも発生します。これらのネットワークインターフェイスのコントロールプレーン側のデータ転送コストは、Amazon EKS のコストに含まれています。お客様側のトラフィックには AWS の標準のデータ転送料金がかかります。具体的には、コントロールプレーンからワーカーノードへのイングレスのトラフィックとワーカーノードからコントロールプレーンへのエグレスのトラフィックです。詳細については、「the section called “Amazon EKS の料金”」を参照してください。

Amazon EKS は、クラスターの作成時に指定した任意のサブネットにネットワークインターフェイスを作成することができます。クラスターの作成後に、Amazon EKS がネットワークインターフェイスを作成するサブネットを変更できます。クラスターの Kubernetes バージョンを更新する

と、Amazon EKS は作成した元のネットワークインターフェイスを削除し、新しいネットワークインターフェイスを作成します。これらのネットワークインターフェイスは、元のネットワークインターフェイスと同じサブネット内に作成することも、元のネットワークインターフェイスとは異なるサブネット内に作成することもできます。ネットワークインターフェイスを作成するサブネットを制御する場合は、クラスターを作成するとき、またはクラスターの作成後にサブネットを更新するときに、指定するサブネットの数を 2 つだけに制限します。

クラスターのサブネットの要件

クラスターの作成または更新時に指定する サブネットは、次の要件を満たす必要があります。

- サブネットには、Amazon EKS での使用のために、それぞれ 6 個以上の IP アドレスが必要です。ただし、IP アドレスは 16 個以上を推奨します。
- サブネットは、少なくとも 2 つの異なるアベイラビリティーゾーンに存在している必要があります。
- サブネットは AWS Outposts または AWS Wavelength 内に存在できません。ただし、VPC 内にサブネットが存在する場合は、セルフマネージド型ノードおよび Kubernetes のリソースをこれらのタイプのサブネットにデプロイできます。セルフマネージド型ノードの詳細については、「the section called “セルフマネージド型ノード”」を参照してください。
- サブネットは、パブリックでもプライベートでもかまいません。ただし、可能であれば、プライベートサブネットを指定することをお勧めします。パブリックサブネットは、インターネットゲートウェイへのルートが含まれているルートテーブルを含むサブネットです。一方、プライベートサブネットは、インターネットゲートウェイへのルートが含まれていないルートテーブルを含むサブネットです。
- サブネットは以下のアベイラビリティーゾーンには配置できません。

AWS リージョン	リージョン名	拒否されたアベイラビリティーゾーン ID
us-east-1	米国東部 (バージニア北部)	use1-az3
us-west-1	米国西部 (北カリフォルニア)	usw1-az2
ca-central-1	カナダ (中部)	cac1-az3

各コンポーネントの IP アドレスファミリー使用状況

以下の表に、Amazon EKS の各コンポーネントで使用される IP アドレスファミリーを示します。ネットワークアドレス変換 (NAT) またはその他の互換システムを使用すると、表の値が「No」のファミリーに属する送信元 IP アドレスから、これらのコンポーネントに接続できます。

機能は、クラスターの IP ファミリー (`ipFamily`) 設定によって異なる場合があります。この設定は、Kubernetes がサービスに割り当てる CIDR ブロックに使用される IP アドレスのタイプを変更します。設定値が IPv4 のクラスターは IPv4 クラスターと呼ばれ、設定値が IPv6 のクラスターは IPv6 クラスターと呼ばれます。

コンポーネント	IPv4 アドレス	IPv6 アドレス	デュアルスタックアドレス
EKS API パブリックエンドポイント	はい [1、3]	はい [1、3]	はい [1、3]
EKS API VPC エンドポイント	はい	いいえ	いいえ
EKS Auth API パブリックエンドポイント (EKS Pod Identity)	はい [1]	はい [1]	はい [1]
EKS Auth API VPC エンドポイント (EKS Pod Identity)	はい [1]	はい [1]	はい [1]
IPv4 Kubernetes クラスターパブリックエンドポイント[2]	はい	いいえ	いいえ
IPv4 Kubernetes クラスタープライベートエンドポイント[2]	はい	いいえ	いいえ
IPv6 Kubernetes クラスターパブリックエンドポイント[2]	はい [1、4]	はい [1、4]	はい [4]

コンポーネント	IPv4 アドレス	IPv6 アドレス	デュアルスタックアドレス
IPv6 Kubernetes クラスタープライベートエンドポイント[2]	はい [1、4]	はい [1、4]	はい [4]
Kubernetes クラスターサブネット	はい [2]	いいえ	はい [2]
ノードのプライマリ IP アドレス	はい [2]	いいえ	はい [2]
サービス IP アドレスのクラスター CIDR 範囲	はい [2]	はい [2]	いいえ
VPC CNI による Pod IP アドレス	はい [2]	はい [2]	いいえ
IRSA OIDC 発行者 URL	はい [1、3]	はい [1、3]	はい [1、3]

Note

[1] エンドポイントは、IPv4 アドレスと IPv6 アドレスの両方を持つデュアルスタックです。AWS 外部のアプリケーション、クラスターのノード、およびクラスター内のポッドは、IPv4 または IPv6 のいずれかによってこのエンドポイントに到達できます。
[2] クラスターの作成時に、クラスターの IP ファミリー (ipFamily) 設定で IPv4 クラスターと IPv6 クラスターのどちらかを選択します。これは後から変更できません。代わりに、他のクラスターを作成してワークロードを移行する際は、別の設定を選択する必要があります。
[3] デュアルスタックエンドポイントは、2024 年 8 月に導入されました。AWS CLI でデュアルスタックエンドポイントを使用するには、「AWS SDK およびツールリファレンスガイド」の「デュアルスタックと FIPS エンドポイント」の設定を参照してください。新しいエンドポイントを次に示します:

EKS API パブリックエンドポイント

```
eks.region.api.aws
```

IRSA OIDC 発行者 URL

```
oidc-eks.region.api.aws
```

[4] デュアルスタッククラスターエンドポイントは、2024 年 10 月に導入されました。EKS は、この日より後に作成され、クラスターの IP ファミリー (ipFamily) 設定で IPv6 を選択した新しいクラスターに次のエンドポイントを作成します。

EKS クラスターパブリック / プライベートエンドポイント

```
eks-cluster.region.api.aws
```

ノードのサブネットの要件

クラスターの作成時に指定するのと同じサブネットに、ノードと Kubernetes リソースをデプロイできます。ただし、これは必須ではありません。これは、クラスターの作成時に指定しなかったサブネットにもノードと Kubernetes リソースをデプロイできるためです。ノードを異なるサブネットにデプロイする場合、Amazon EKS は、それらのサブネットにクラスターネットワークインターフェイスを作成しません。ノードと Kubernetes リソースをデプロイするサブネットは、次の要件を満たす必要があります。

- サブネットには、すべてのノードと Kubernetes リソースをデプロイするのに十分な数の使用可能な IP アドレスが必要です。
- Kubernetes で IPv6 アドレスを Pod およびサービスに割り当てる場合は、サブネットに関連付けられている IPv6 CIDR ブロックと IPv4 CIDR ブロックをそれぞれ 1 つずつ指定する必要があります。詳細については、Amazon VPC ユーザーガイドの「IPv6 CIDR ブロックをサブネットに関連付ける」を参照してください。サブネットに関連付けられているルートテーブルには、IPv4 および IPv6 のアドレスへのルートを含める必要があります。詳細については、Amazon VPC ユーザーガイドの「ルート」を参照してください。ポッドに割り当てられるのは IPv6 アドレスのみです。ただし、Amazon EKS がクラスターとノード用に作成するネットワークインターフェイスには、IPv4 および IPv6 のアドレスが割り当てられます。
- インターネットから Pod へのインバウンドアクセスが必要な場合は、ロードバランサーのデプロイおよびインバウンドトラフィック (イングレス) に十分な数の IP アドレスを利用できるパブリッ

クサブネットが少なくとも 1 つあることを必ず確認してください。パブリックサブネットにロードバランサーをデプロイできます。ロードバランサーは、プライベートサブネットまたはパブリックサブネット内の Pod への負荷分散を行います。可能であれば、プライベートサブネットにノードをデプロイすることをお勧めします。

- ノードをパブリックサブネットにデプロイする場合は、サブネットが IPv4 パブリックアドレスまたは IPv6 アドレスを自動割り当てする必要があります。ノードを IPv6 CIDR ブロックが関連付けられているプライベートサブネットにデプロイする場合、プライベートサブネットも IPv6 アドレスを自動割り当てする必要があります。2020 年 3 月 26 日以降に Amazon EKS が提供する AWS CloudFormation テンプレートを使用して VPC をデプロイした場合、この設定は有効になっています。テンプレートを使用してこの日付より前に VPC をデプロイした場合、または独自の VPC を使用した場合は、この設定を手動で有効にする必要があります。テンプレートについては、「the section called “「VPC を作成する」”」を参照してください。詳細については、Amazon VPC ユーザーガイドの「サブネットのパブリック IPv4 アドレス指定属性を変更する」および「サブネットの IPv6 アドレス指定属性を変更する」を参照してください。
- ノードをデプロイするサブネットがプライベートサブネットであり、そのルートテーブルにネットワークアドレス変換 (NAT) デバイス (IPv4) または Egress-Only ゲートウェイ (IPv6) へのルートが含まれていない場合は、AWS PrivateLink を使用して VPC エンドポイントを VPC に追加します。ノードとポッドは、通信するすべての AWS サービスに対して VPC エンドポイントが必要です。例としては、Amazon ECR、Elastic Load Balancing、Amazon CloudWatch、AWS Security Token Service、Amazon Simple Storage Service (Amazon S3) などがあります。サービスアカウント (IRSA) に IAM ロールを使用する場合、`com.amazonaws.region-code.oidc-eks` エンドポイントを使用してクラスター OIDC 検出/JWKS エンドポイントにプライベートに到達することもできます。詳細については、「the section called “AWS PrivateLink を使用してクラスター OIDC エンドポイントにアクセスする”」を参照してください。すべての AWS で VPC エンドポイントがサポートされているわけではありません。詳細については、「AWS PrivateLink とは」および「AWS PrivateLink と統合する AWS サービス」を参照してください。Amazon EKS のその他の要件のリストについては、「the section called “プライベートクラスター”」を参照してください。
- ロードバランサーをサブネットにデプロイする場合は、サブネットに次のタグが必要です。
  - プライベートサブネット

キー	値
`kubernetes.io/role/internal-elb`	`1`

  - パブリックサブネット

キー	値
kubernetes.io/role/elb	1

バージョン 1.18 以前の Kubernetes クラスターが作成された場合、Amazon EKS は指定したすべてのサブネットに次のタグを追加します。

キー	値
kubernetes.io/cluster/ *my-cluster*	shared

ここで新しい Kubernetes クラスターを作成した場合、Amazon EKS はサブネットにタグを追加しません。1.19 より前のバージョンのクラスターが使用していたサブネットにタグがあった場合、クラスターが新しいバージョンに更新されても、タグはサブネットから自動的に削除されませんでした。AWS ロードバランサーコントローラーのバージョン 2.1.1 以前では、このタグが必要です。新しいバージョンのロードバランサーコントローラーを使用している場合は、サービスを中断することなくタグを削除できます。コントローラーの詳細については、「the section called “ AWS ロードバランサーコントローラー”」を参照してください。

eksctl または Amazon EKS AWS CloudFormation VPC テンプレートのいずれかを使用して VPC をデプロイした場合、次が適用されます。

- 2020 年 3 月 26 日以降 – パブリック IPv4 アドレスは、パブリックサブネットにより、パブリックサブネットにデプロイした新しいノードに自動的に割り当てられます。
- 2020 年 3 月 26 日より前 – パブリック IPv4 アドレスは、パブリックサブネットにより、パブリックサブネットにデプロイした新しいノードに自動的に割り当てられません。

この変更は、パブリックサブネットにデプロイされた新しいノードグループに、次のような影響を与えます。

- マネージドノードグループ – 2020 年 4 月 22 日以降にノードグループをパブリックサブネットにデプロイした場合は、パブリックサブネットでパブリック IP アドレスの自動割り当てを有効にする必要があります。詳細については、「サブネットの IPv4 アドレス指定属性の変更」を参照してください。

- Linux、Windows、または Arm のセルフマネージドノードグループ – 2020 年 3 月 26 日以降にノードグループをパブリックサブネットにデプロイした場合は、パブリックサブネットでパブリック IP アドレスの自動割り当てを有効にする必要があります。それ以外の場合は、代わりにパブリック IP アドレスを使用してノードを起動する必要があります。詳細については、「サブネットのパブリック IPv4 アドレス指定属性の変更」または「インスタンス起動時のパブリック IPv4 アドレスの割り当て」を参照してください。

共有サブネットの要件と考慮事項

VPC 共有を使用して、同じ AWS 組織内でサブネットを他の AWS アカウントと共有できます。共有サブネットに Amazon EKS クラスターを作成できますが、以下の考慮事項に注意してください。

- VPC サブネットの所有者は、参加者アカウントで Amazon EKS クラスターを作成する前に、そのアカウントとサブネットを共有する必要があります。
- VPC のデフォルトセキュリティグループは所有者に属しているため、デフォルトのセキュリティグループを使用してリソースを起動することはできません。さらに、参加者は、他の参加者または所有者が所有するセキュリティグループを使用してリソースを起動することはできません。
- 共有サブネットでは、参加者と所有者がそれぞれのアカウント内のセキュリティグループを個別に管理します。サブネットの所有者は、参加者が作成したセキュリティグループを表示できますが、これらのグループに対してアクションを実行することはできません。サブネットの所有者がこれらのセキュリティグループの削除または変更を希望する場合は、セキュリティグループを作成した参加者がそのアクションを実行する必要があります。
- 参加者がクラスターを作成する場合、以下の考慮事項が適用されます。
  - クラスター IAM ロールとノード IAM ロールは、そのアカウントで作成する必要があります。詳細については、the section called “クラスター IAM ロール” および the section called “ノード の IAM ロール” を参照してください。
  - 管理対象ノードグループを含め、すべてのノードは同じ参加者が作成する必要があります。
- 共有 VPC の所有者は、参加者が共有サブネット内に作成したクラスターを表示、更新、削除することはできません。これは、アカウントごとに異なるアクセス権を持つ VPC リソースに加えて適用されます。詳細については、Amazon VPC ユーザーガイドの「所有者および参加者の責任と権限」を参照してください。
- Amazon VPC CNI plugin for Kubernetes のカスタムネットワーク機能を使用する場合、所有者アカウントに記載されているアベイラビリティーゾーン ID マッピングを使用して、それぞれの `ENIConfig` を作成する必要があります。詳細については、the section called “カスタムネットワーキング” を参照してください。

VPC サブネット共有の詳細については、Amazon VPC ユーザーガイドの「VPC を他のアカウントと共有する」を参照してください。

Amazon EKS クラスターの Amazon VPC を作成する

Amazon Virtual Private Cloud (Amazon VPC) を使用すると、定義した仮想ネットワーク内で AWS リソースを起動できます。この仮想ネットワークは、お客様自身のデータセンターで運用されている従来のネットワークによく似ています。ただし、Amazon Web Services のスケーラブルなインフラストラクチャを使用できるというメリットがあります。本番用に使う Amazon EKS クラスターをデプロイする前に、Amazon VPC サービスを十分に理解しておくことをお勧めします。詳細については、Amazon VPC ユーザーガイドを参照してください。

Amazon EKS クラスター、ノード、および Kubernetes リソースが VPC にデプロイされます。Amazon EKS で既存の VPC を使用する場合は、その VPC が「the section called “VPC およびサブネットの要件”」に記載されている要件を満たしている必要があります。このトピックでは、Amazon EKS に用意されている AWS CloudFormation テンプレートを使用して Amazon EKS 要件を満たす VPC を作成する方法について説明します。テンプレートをデプロイすると、テンプレートによって作成されたリソースを表示して、作成されたリソースとそれらのリソースの設定を正確に把握できます。ハイブリッドノードを使用している場合、VPC のルートテーブルにオンプレミスネットワークへのルートが含まれている必要があります。ハイブリッドノードのネットワーク要件の詳細については、「the section called “ネットワークを準備する”」を参照してください。

前提条件

Amazon EKS 用の VPC を作成するには、Amazon VPC リソースを作成するのに必要な IAM アクセス許可が必要です。これらのリソースは、VPC、サブネット、セキュリティグループ、ルートテーブルとルート、およびインターネットと NAT ゲートウェイです。詳細については、「Amazon VPC ユーザーガイド」の「パブリックサブネットを持つ VPC を作成するポリシー例」を参照してください。完全なリストについては、「サービス認可リファレンス」の「アクション」を参照してください。

パブリックサブネットとプライベートサブネット、パブリックサブネットのみ、またはプライベートサブネットのみで、VPC を作成できます。

パブリックサブネットおよびプライベートサブネット

この VPC には、2 つのパブリックサブネットと 2 つのプライベート サブネットがあります。パブリックサブネットの関連付けられているルートテーブルには、インターネットゲートウェイへのルートが含まれています。一方、プライベートサブネットのルートテーブルには、インターネットゲー

トウェイへのルートがありません。1 つのパブリックサブネットと 1 つのプライベートサブネットが同じアベイラビリティーゾーンにデプロイされます。他のパブリックサブネットとプライベートサブネットは、同じ AWS リージョン内の 2 番目のアベイラビリティーゾーンにデプロイされます。ほとんどのデプロイにこのオプションをお勧めします。

このオプションを使用すると、プライベートサブネットにノードをデプロイできます。このオプションでは、Kubernetes がパブリックサブネットにロードバランサーをデプロイできるようにします。これにより、プライベートサブネットのノードで実行されている Pod へのトラフィックの負荷を分散できます。パブリック IPv4 アドレスは、パブリックサブネットにデプロイされたノードに自動的に割り当てられます。一方、プライベートサブネットにデプロイされたノードに対しては、パブリック IPv4 アドレスは割り当てられません。

また、パブリックサブネットおよびプライベートサブネットに置かれたノードに対し、IPv6 アドレスを割り当てることもできます。プライベートサブネット内のノードは、クラスターや他の AWS サービスとの通信が可能です。Pod は、各アベイラビリティーゾーンにデプロイされた NAT ゲートウェイ (IPv4 アドレスを使用) または送信専用インターネットゲートウェイ (IPv6 アドレスを使用) を介して、インターネットへの通信が可能です。クラスターまたはノード以外のソースからのすべてのインバウンドトラフィックを拒否するが、すべてのアウトバウンドトラフィックを許可するルールを持つセキュリティグループがデプロイされます。サブネットには、Kubernetes がロードバランサーをデプロイできるようにタグが付けられています。

a. AWS CloudFormation コンソールを開きます。

b. ナビゲーションバーで、Amazon EKS をサポートする AWS リージョンを選択します。

c. [スタックの作成] を選択し、[新しいリソースの使用 (標準)] を選択します。

d. [Prerequisite - Prepare template] (前提条件 – テンプレートの準備) の下で、[Template is ready] (テンプレートの準備完了) がオンになっていることを確認した上で、[Specify template] (テンプレートの指定) の下で、[Amazon S3 URL] を選択します。

e. IPv4 のみをサポートする VPC、または IPv4 と IPv6 をサポートする VPC を作成できます。下記の URL の 1 つを [Amazon S3 URL] の下にあるテキスト領域に貼り付けて、[Next] (次へ) をクリックします。

   - IPv4

```
https://s3.us-west-2.amazonaws.com/amazon-eks/cloudformation/2020-10-29/amazon-eks-vpc-private-subnets.yaml
```

- IPv4 および IPv6

```
https://s3.us-west-2.amazonaws.com/amazon-eks/cloudformation/2020-10-29/amazon-eks-ipv6-vpc-public-private-subnets.yaml
```

a. [Specify stack details] (スタック詳細の指定) ページで、パラメータを入力し、[Next] (次へ) を選択します。

- スタック名: AWS CloudFormation スタックのスタック名を選択します。例えば、前のステップで使用されたテンプレート名を使用できます。この名前には英数字 (大文字と小文字が区別されます) とハイフンのみを使用できます。先頭の文字は英数字である必要があります。また、100 文字より長くすることはできません。名前は、クラスターを作成する AWS リージョンおよび AWS アカウント内で一意である必要があります。
- [VpcBlock]: VPC の IPv4 CIDR 範囲を選択します。デプロイする各ノード、ポッド、およびロードバランサーには、このブロックの IPv4 アドレスが割り当てられます。デフォルトの IPv4 値で、ほとんどの実装において十分な IP アドレスを取得できますが、不足する場合はこの値を変更できます。詳細については、「Amazon VPC ユーザーガイド」の「VPC とサブネットのサイズ設定」を参照してください。VPC を作成したら、追加の CIDR ブロックを VPC に追加することもできます。IPv6 VPC を作成する場合、IPv6 CIDR 範囲は Amazon のグローバルユニキャストアドレススペースから自動的に割り当てられます。
- [PublicSubnet01Block]: パブリックサブネット 1 の IPv4 CIDR ブロックを指定します。デフォルト値は、ほとんどの実装で十分な IP アドレスを提供しますが、そうでない場合は、変更することができます。IPv6 VPC を作成する場合、テンプレートで指定済みブロックを使用できます。
- [PublicSubnet02Block]: パブリックサブネット 2 の IPv4 CIDR ブロックを指定します。デフォルト値は、ほとんどの実装で十分な IP アドレスを提供しますが、そうでない場合は、変更することができます。IPv6 VPC を作成する場合、テンプレートで指定済みブロックを使用できます。
- [PrivateSubnet01Block]: プライベートサブネット 1 の IPv4 CIDR ブロックを指定します。デフォルト値は、ほとんどの実装で十分な IP アドレスを提供しますが、そうでない場合は、変更することができます。IPv6 VPC を作成する場合、テンプレートで指定済みブロックを使用できます。
- [PrivateSubnet02Block]: プライベートサブネット 2 の IPv4 CIDR ブロックを指定します。デフォルト値は、ほとんどの実装で十分な IP アドレスを提供しますが、そうでない場合は、変更することができます。IPv6 VPC を作成する場合、テンプレートで指定済みブロックを使用できます。

b. (オプション) [Configure stack options] (スタックオプションの設定) ページで、スタックリソースにタグ付けを行った上で、[Next] (次へ) をクリックします。

c. [確認] ページで [スタックの作成] を選択します。

d. スタックが作成されたら、コンソールで選択し、[Outputs] (出力) を選択します。

e. 作成された VPC の [VpcId] を記録します。これは、クラスターとノードを作成するときに必要です。

f. 作成されたサブネットの SubnetIds と、それらをパブリックサブネットとプライベートサブネットのどちらとして作成したかを記録します。クラスターとノードを作成するときは、これらのうち少なくとも 2 つが必要です。

g. IPv4 VPC を作成している場合、この手順をスキップしてください。IPv6 VPC を作成している場合は、テンプレートによって作成されたパブリックサブネットにおいて、IPv6 アドレスの自動割り当てオプションが有効化されている必要があります。この設定は、プライベートサブネットに対してすでに有効になっています。次の手順を完了して、設定を有効にします。

   i. Amazon VPC コンソールの https://console.aws.amazon.com/vpc/ を開いてください。

   ii. 左のナビゲーションペインで [Subnets] (サブネット) を選択します。

   iii. パブリックサブネット (***stack-name***/SubnetPublic01 または ***stack-name***/SubnetPublic02 には、public という語が含まれています) の 1 つを選択し、[Actions] (アクション)、[Edit subnet settings] (サブネット設定の編集) の順に選択します。

   iv. [Enable auto-assign IPv6 address] (IPv6 アドレスの自動割り当てを有効にする) チェックボックスをオンにし、[Save] (保存) をクリックします。

   v. 他のパブリックサブネットに対しても、ここまでの手順を再度実行します。

パブリックサブネットのみ

この VPC には、AWS リージョン内の異なるアベイラビリティーゾーンにデプロイされる 3 つのパブリックサブネットがあります。すべてのノードには自動的にパブリック IPv4 アドレスが割り当てられ、インターネットゲートウェイを介してインターネットトラフィックを送受信できます。すべてのインバウンドトラフィックを拒否し、すべてのアウトバウンドトラフィックを許可するセキュリティグループがデプロイされます。サブネットには、Kubernetes がロードバランサーをデプロイできるようにタグが付けられています。

a. AWS CloudFormation コンソールを開きます。

b. ナビゲーションバーで、Amazon EKS をサポートする AWS リージョンを選択します。

c. [Create stack] (スタックの作成) を選択し、[With new resources (standard)] (新しいリソースの使用 (標準)) を選択します。

d. [テンプレートの準備] の下で、[テンプレートの準備完了] が選択されていることを確認し、[テンプレートソース] の下で、[Amazon S3 URL] を選択します。

e. 次の URL を [Amazon S3 URL] の下のテキスト領域に貼り付けて、[次へ] を選択します。

```
https://s3.us-west-2.amazonaws.com/amazon-eks/cloudformation/2020-10-29/amazon-eks-vpc-sample.yaml
```

a. [Specify Details] (詳細の指定) ページで、パラメータを入力し、[Next] (次へ) を選択します。

- スタック名: AWS CloudFormation スタックのスタック名を選択します。例えば、*amazon-eks-vpc-sample* という名前にすることができます。この名前には英数字 (大文字と小文字が区別されます) とハイフンのみを使用できます。先頭の文字は英数字である必要があります。また、100 文字より長くすることはできません。名前は、クラスターを作成する AWS リージョンおよび AWS アカウント内で一意である必要があります。
- [VpcBlock]: VPC の CIDR ブロックを選択します。デプロイする各ノード、ポッド、およびロードバランサーには、このブロックの IPv4 アドレスが割り当てられます。デフォルトの IPv4 値で、ほとんどの実装において十分な IP アドレスを取得できますが、不足する場合はこの値を変更できます。詳細については、「Amazon VPC ユーザーガイド」の「VPC とサブネットのサイズ設定」を参照してください。VPC を作成したら、追加の CIDR ブロックを VPC に追加することもできます。
- Subnet01Block: サブネット 1 の CIDR ブロックを指定します。デフォルト値は、ほとんどの実装で十分な IP アドレスを提供しますが、そうでない場合は、変更することができます。
- Subnet02Block: サブネット 2 の CIDR ブロックを指定します。デフォルト値は、ほとんどの実装で十分な IP アドレスを提供しますが、そうでない場合は、変更することができます。
- Subnet03Block: サブネット 3 の CIDR ブロックを指定します。デフォルト値は、ほとんどの実装で十分な IP アドレスを提供しますが、そうでない場合は、変更することができます。

b. (オプション) [オプション] ページで、スタックリソースをタグ付けします。[次へ] を選択します。

c. [Review] ページで、[Create] を選択します。

d. スタックが作成されたら、コンソールで選択し、[Outputs] (出力) を選択します。

e. 作成された VPC の [VpcId] を記録します。これは、クラスターとノードを作成するときに必要です。

f. 作成されたサブネットの [SubnetIds] を記録します。クラスターとノードを作成するときは、これらのうち少なくとも 2 つが必要です。

g. (オプション) この VPC にデプロイするクラスターは、Pod とサービスにプライベート IPv4 アドレスを割り当てることができます。この VPC にクラスターをデプロイし、Pod とサービスにプライベート IPv6 アドレスを割り当てる場合は、VPC、サブネット、ルートテーブル、およびセキュリティグループを更新します。詳細については、「Amazon VPC ユーザーガイド」の「既存の VPC を IPv4 から IPv6 に移行する」を参照してください。Amazon EKS では、サブネットにおいて、IPv6 アドレスの Auto-assign オプションが有効化されている必要があります。デフォルトでは、このオプションは無効となっています。

プライベートサブネットのみ

この VPC には、AWS リージョン内の異なるアベイラビリティーゾーンにデプロイされる 3 つのプライベートサブネットがあります。サブネットにデプロイされたリソースはインターネットにアクセスできず、インターネットからサブネット内のリソースにアクセスすることもできません。ノードが通常アクセスする必要があるいくつかの AWS サービスの AWS PrivateLink を使用して、VPC エンドポイントがテンプレートによって作成されます。ノードにアウトバウンドインターネットアクセスが必要な場合は、VPC が作成された後で、各サブネットのアベイラビリティーゾーン内に、パブリック NAT ゲートウェイを追加できます。セキュリティグループは、サブネットにデプロイされたリソースからのトラフィックを除き、すべてのインバウンドトラフィックを拒否するように作成されます。また、セキュリティグループは、すべてのアウトバウンドトラフィックを許可します。サブネットには、Kubernetes が内部ロードバランサーをデプロイできるようにタグが付けられています。この設定で VPC を作成する場合は、「the section called “プライベートクラスター”」を参照してその他の要件と考慮事項を確認してください。

a. AWS CloudFormation コンソールを開きます。

b. ナビゲーションバーで、Amazon EKS をサポートする AWS リージョンを選択します。

c. [Create stack] (スタックの作成) を選択し、[With new resources (standard)] (新しいリソースの使用 (標準)) を選択します。

d. [テンプレートの準備] の下で、[テンプレートの準備完了] が選択されていることを確認し、[テンプレートソース] の下で、[Amazon S3 URL] を選択します。

e. 次の URL を [Amazon S3 URL] の下のテキスト領域に貼り付けて、[次へ] を選択します。

```
https://s3.us-west-2.amazonaws.com/amazon-eks/cloudformation/2020-10-29/amazon-eks-fully-private-vpc.yaml
```

a. [詳細の指定] ページでパラメータを入力し、[次へ] を選択します。

- スタック名: AWS CloudFormation スタックのスタック名を選択します。例えば、*amazon-eks-fully-private-vpc* という名前にすることができます。この名前には英数字 (大文字と小文字が区別されます) とハイフンのみを使用できます。先頭の文字は英数字である必要があります。また、100 文字より長くすることはできません。名前は、クラスターを作成する AWS リージョンおよび AWS アカウント内で一意である必要があります。
- [VpcBlock]: VPC の CIDR ブロックを選択します。デプロイする各ノード、ポッド、およびロードバランサーには、このブロックの IPv4 アドレスが割り当てられます。デフォルトの IPv4 値で、ほとんどの実装において十分な IP アドレスを取得できますが、不足する場合はこの値を変更できます。詳細については、「Amazon VPC ユーザーガイド」の「VPC とサブネットのサイズ設定」を参照してください。VPC を作成したら、追加の CIDR ブロックを VPC に追加することもできます。
- PrivateSubnet01Block: サブネット 1 の CIDR ブロックを指定します。デフォルト値は、ほとんどの実装で十分な IP アドレスを提供しますが、そうでない場合は、変更することができます。
- PrivateSubnet02Block: サブネット 2 の CIDR ブロックを指定します。デフォルト値は、ほとんどの実装で十分な IP アドレスを提供しますが、そうでない場合は、変更することができます。
- PrivateSubnet03Block: サブネット 3 の CIDR ブロックを指定します。デフォルト値は、ほとんどの実装で十分な IP アドレスを提供しますが、そうでない場合は、変更することができます。

b. (オプション) [オプション] ページで、スタックリソースをタグ付けします。[次へ] を選択します。

c. [Review] ページで、[Create] を選択します。

d. スタックが作成されたら、コンソールで選択し、[Outputs] (出力) を選択します。

e. 作成された VPC の [VpcId] を記録します。これは、クラスターとノードを作成するときに必要です。

f. 作成されたサブネットの [SubnetIds] を記録します。クラスターとノードを作成するときは、これらのうち少なくとも 2 つが必要です。

g. (オプション) この VPC にデプロイするクラスターは、Pod とサービスにプライベート IPv4 アドレスを割り当てることができます。この VPC にクラスターをデプロイし、Pod とサービスにプライベート IPv6 アドレスを割り当てる場合は、VPC、サブネット、ルートテーブル、およびセキュリティグループを更新します。詳細については、「Amazon VPC ユーザーガイド」の「既存の VPC を IPv4 から IPv6 に移行する」を参照してください。Amazon EKS は、サブネットで Auto-assign IPv6 アドレスオプションが有効になっている必要があります (デフォルトでは無効)。

クラスターの Amazon EKS セキュリティグループ要件を表示する

このトピックでは、Amazon EKS クラスターのセキュリティグループの要件について説明します。

デフォルトのクラスターセキュリティグループ

クラスターを作成すると、Amazon EKS により `eks-cluster-sg-my-cluster-uniqueID` という名前のセキュリティグループが作成されます。セキュリティグループには、デフォルトで次のルールがあります。

ルールタイプ	プロトコル	ポート	送信元	目的地
インバウンド	すべて	すべて	自分	
アウトバウンド	すべて	すべて		0.0.0.0/0 (IPv4) または ::/0 (IPv6)
アウトバウンド	すべて	すべて		自分 (EFA トラフィック用)

デフォルトのセキュリティグループには、同じセキュリティグループの宛先に対して Elastic Fabric Adapter (EFA) トラフィックを許可するアウトバウンドルールが含まれます。これにより、クラスター内の EFA トラフィックが有効になり、AI/ML およびハイパフォーマンスコンピューティング (HPC) ワークロードに役立てることができます。詳細については、「Amazon Elastic Compute Cloud ユーザーガイド」の「Amazon EC2 の AI/ML および HPC ワークロード用の Elastic Fabric Adapter」を参照してください。

> **Important**
>
> クラスターに `0.0.0.0/0` (IPv4)、`::/0` (IPv6) へのアウトバウンドルールが必要ない場合は、削除できます。削除する場合でも、「クラスタートラフィックの制限」に記載されている最低限のルールが適用されます。クラスターセキュリティグループ自体との間のトラフィックを許可するインバウンドルールまたはアウトバウンドルールを削除すると、クラスターが更新されるたびに Amazon EKS によって再作成されます。

Amazon EKS は次のタグをセキュリティグループに追加します。タグを削除した場合、Amazon EKS はクラスターが更新されるたびにこれらのタグをセキュリティグループに追加します。

キー	値
kubernetes.io/cluster/ *my-cluster*	owned
aws:eks:cluster-name	*my-cluster*
Name	eks-cluster-sg- *my-cluste r* -*uniqueid*

Amazon EKS は、同様に作成される次のリソースに、セキュリティグループを自動的に関連付けます。

- 2—4 エラスティックネットワークインターフェイス (これ以降、ネットワークインターフェイス) は、クラスターの作成時に作成されます。
- 作成したマネージドノードグループ内のノードのネットワークインターフェイス。

デフォルトのルールでは、すべてのトラフィックがクラスターとノード間で自由に行き来することができます。また、任意の送信先へのすべてのアウトバウンドトラフィックが許可されています。クラスターを作成すると、オプションで独自のセキュリティグループを指定できます。その場合、Amazon EKS は、指定したセキュリティグループをクラスター用に作成するネットワークインターフェイスにも関連付けます。ただし、作成したノードグループには関連付けられません。

クラスターのセキュリティグループの ID は、AWS マネジメントコンソール のクラスターの [Networking] (ネットワーキング) セクションで確認できます。もしくは、次の AWS CLI コマンドを実行して確認できます。

```
aws eks describe-cluster --name my-cluster --query
 cluster.resourcesVpcConfig.clusterSecurityGroupId
```

クラスタートラフィックの制限

EKS コントロールプレーンとノード間で開いているポートの数を制限する必要がある場合は、デフォルトのアウトバウンドルール `0.0.0.0/0` (IPv4)/`::/0` (IPv6) を削除し、クラスターに必要な以下の最小限のルールを追加できます。

送信元がセルフとなるすべてのトラフィック (クラスターセキュリティグループからのトラフィック) を許可するデフォルトのインバウンドルールを削除した場合、クラスターが更新されると Amazon EKS によって再作成されます。

送信先がセルフとなるすべてのトラフィック (クラスターセキュリティグループへのトラフィック) を許可するデフォルトのアウトバウンドルールを削除した場合、クラスターが更新されると Amazon EKS によって再作成されます。

ルールタイプ	プロトコル	ポート	目的地
アウトバウンド	TCP	443	クラスターセキュリティグループ
アウトバウンド	TCP	10250	クラスターセキュリティグループ
アウトバウンド (DNS)	TCP と UDP	53	クラスターセキュリティグループ

次のトラフィックにもルールを追加する必要があります。

- ノード間通信にノードが使用するプロトコルおよびポート。
- アウトバウンドのインターネットアクセス。これによりノードが Amazon EKS API にアクセス可能になり、起動時のノードの登録やクラスターの詳細分析が行えます。ノードがインターネットにアクセスできない場合は、「インターネットアクセスが制限されたプライベートクラスターをデプロイする」でその他の考慮事項を参照してください。
- Amazon ECR や、イメージをプルする必要があるその他のコンテナレジストリ (DockerHub など) からコンテナイメージをプルするためのノードのアクセス。詳細については、「AWS 全般のリファレンス」の「AWS IP アドレス範囲」を参照してください。
- Amazon S3 へのノードのアクセス。
- IPv4 や IPv6 アドレスが必要な個別のルール。
- ハイブリッドノードを使用している場合は、クラスターに追加のセキュリティグループを追加して、オンプレミスのノードやポッドとの通信を許可する必要があります。詳細については、「the section called “ネットワークを準備する”」を参照してください。

ルールの制限を検討している場合は、変更したルールを本番稼働用のクラスターに適用する前に、すべての Pod を徹底的にテストすることをお勧めします。

最初に Kubernetes `1.14` およびプラットフォームバージョン `eks.3` またはそれ以前でクラスターを最初にデプロイした場合、次の点を考慮してください。

- コントロールプレーンおよびノードセキュリティグループがある場合もあります。これらのグループの作成時、前の表に示した制限付きルールが含まれていました。これらのセキュリティグループは不要で、削除できます。ただし、それらのグループに含まれるルールがクラスターセキュリティグループに含まれていることを確認する必要があります。
- API を使用してクラスターを直接デプロイした場合、または AWS CLI や AWS CloudFormation などのツールを使用してクラスターを作成しており、クラスター作成時にセキュリティグループを指定しなかった場合、Amazon EKS が作成したクラスターネットワークインターフェイスに VPC のデフォルトのセキュリティグループが適用されます。

共有セキュリティグループ

Amazon EKS は、共有セキュリティグループをサポートしています。

- [セキュリティグループの VPC 関連付け] により、セキュリティグループは同じアカウントとリージョン内の複数の VPC に関連付けられます。
  - 「Amazon VPC ユーザーガイド」で、セキュリティグループを複数の VPC に関連付ける方法をご覧ください。
- [共有セキュリティグループ] を使用すると、セキュリティグループを他の AWS アカウントと共有できます。アカウントは同じ AWS 組織内にある必要があります。
  - 「Amazon VPC ユーザーガイド」で、セキュリティグループを組織と共有する方法をご覧ください。
- セキュリティグループは常に 1 つの AWS リージョンに制限されます。

Amazon EKS の考慮事項

- EKS には、標準セキュリティグループと同じ共有セキュリティグループまたはマルチ VPC セキュリティグループの要件があります。

Amazon EKS クラスターのネットワーキングアドオンを管理する

Amazon EKS クラスターで使用できるネットワーキングアドオンがいくつかあります。

組み込みアドオン

> **Note**
>
> EKS クラスターを作成するとき:
>
> - AWS コンソールの使用: 組み込みアドオン (CoreDNS や kube-proxy など) は Amazon EKS アドオンとして自動的にインストールされます。これらは、AWS コンソール、CLI、SDK を介して簡単に設定および更新できます。
> - 他の方法の使用 (CLI や SDK など): 通常の Kubernetes デプロイとして実行されるセルフマネージドバージョンと同じ組み込みアドオンがインストールされます。AWS ツールで管理できないため、手動で設定および更新が必要です。
>
> アドオン管理を簡素化し、AWS サービスを通じて一元化された設定および更新を有効にするには、セルフマネージドバージョンではなく、Amazon EKS アドオンを使用することをお勧めします。

Amazon VPC CNI plugin for Kubernetes

この CNI アドオンは、Elastic Network Interface を作成し、Amazon EC2 ノードにアタッチします。このアドオンは、VPC からプライベート `IPv4` または `IPv6` アドレスを各 Pod およびサービスにも割り当てます。このアドオンはデフォルトでクラスターにインストールされます。詳細については、「the section called “Amazon VPC CNI”」を参照してください。ハイブリッドノードを使用している場合、VPC CNI はデフォルトでインストールされますが、アンチアフィニティルールによりハイブリッドノード上での実行が防止されます。ハイブリッドノードの CNI オプションの詳細については、「the section called “CNI を設定する”」を参照してください。

CoreDNS

CoreDNS は、Kubernetes クラスター DNS として機能する、フレキシブルで拡張可能な DNS サーバーです。CoreDNS により、クラスター内にあるすべての Pod の名前が解決されます。このアドオンはデフォルトでクラスターにインストールされます。詳細については、「the section called “CoreDNS”」を参照してください。

kube-proxy

このアドオンは、Amazon EC2 ノード上のネットワークルールを維持し、Pod へのネットワーク通信を可能にします。このアドオンはデフォルトでクラスターにインストールされます。詳細については、「the section called “kube-proxy”」を参照してください。

オプションの AWS ネットワークアドオン

AWS Load Balancer Controller

`loadbalancer` タイプの Kubernetes サービスオブジェクトをデプロイするとき、コントローラーは AWS Network Load Balancer を作成します。Kubernetes 入力オブジェクトを作成するとき、コントローラーは AWS Application Load Balancer を作成します。Network Load Balancer のプロビジョニングには、Kubernetes に組み込まれている従来のクラウドプロバイダーコントローラーではなく、このコントローラーを使用することをお勧めします。詳細については、AWS Load Balancer Controller ドキュメントを参照してください。

AWS ゲートウェイ API コントローラー

このコントローラーを使用すると、Kubernetes ゲートウェイ API を使用して複数の Kubernetes クラスターを通してサービスに接続できます。コントローラーは、Amazon VPC Lattice サービスを使用して Amazon EC2 インスタンス、コンテナ、サーバーレス機能で実行されている Kubernetes サービスを接続します。詳細については、「AWS ゲートウェイ API コントローラー」ドキュメントを参照してください。

アドオンの詳細については、「the section called “Amazon EKS アドオン”」を参照してください。

Amazon VPC CNI を使用して Pod に IP を割り当てる

> **Tip**
>
> 今後開催予定の Amazon EKS ワークショップに登録してください。

> **Tip**
>
> Amazon EKS Auto Mode ではネットワーク形成のアドオンをインストールまたはアップグレードする必要はありません。Auto Mode にはポッドのネットワーク形成とロードバランシング機能が含まれています。

詳細については、EKS Auto Mode を参照してください。

Amazon VPC CNI plugin for Kubernetes アドオンは、Amazon EKS クラスター内の各 Amazon EC2 ノードにデプロイされます。アドオンは エラスティックネットワークインターフェイス を作成し、Amazon EC2 ノードにアタッチします。また、アドオンは VPC のプライベート IPv4 アドレスまたは IPv6 アドレスを各 Pod に割り当てます。

アドオンのバージョンはクラスター内の各 Fargate ノードにデプロイされますが、Fargate ノードでは更新しません。Amazon EKS クラスターではその他の互換性のある CNI プラグインも使用できますが、これは AWS インフラストラクチャで動作するノード用の Amazon EKS がサポートする唯一の CNI プラグインです。互換性のある他の CNI プラグインの詳細については「the section called “代替 CNI プラグイン”」を参照してください。VPC CNI はハイブリッドノードでの使用はサポートされていません。ハイブリッドノードの CNI オプションの詳細については、the section called “CNI を設定する” を参照してください。

次の表は、Kubernetes バージョンごとに Amazon EKS アドオンタイプの最新バージョンを示しています。

Amazon VPC CNI のバージョン

Kubernetes バージョン	Amazon EKS タイプの VPC CNI バージョン
1.37	v1.23.1-eksbuild.1
1.36	v1.23.1-eksbuild.1
1.35	v1.23.1-eksbuild.1
1.34	v1.23.1-eksbuild.1
1.33	v1.23.1-eksbuild.1
1.32	v1.23.1-eksbuild.1
1.31	v1.23.1-eksbuild.1

> **Important**
>
> このアドオンを自己管理している場合、表のバージョンは利用可能なセルフマネージドバージョンと同じではない可能性があります。このアドオンのセルフマネージドタイプの更新の詳細については「the section called “更新（自己管理）”」を参照してください。

> **Important**
>
> VPC CNI v1.12.0 以降にアップグレードするにはまず VPC CNI v1.7.0 にアップグレードする必要があります。一度に 1 つのマイナーバージョンを更新することをお勧めします。

考慮事項

この機能を使用する際の考慮事項を次に示します。

- バージョンは `major-version.minor-version.patch-version-eksbuild.build-number` として指定されます。
- 各機能のバージョン互換性を確認します。Amazon VPC CNI plugin for Kubernetes の各リリースに含まれる機能によっては、特定の Kubernetes バージョンが必要になることがあります。異なる Amazon EKS の機能を使用する際、アドオンの特定のバージョンが必要な場合については機能に関するドキュメントに記載されています。以前のバージョンを実行する特定の理由がある場合を除いて、最新のバージョンを実行することをお勧めします。

Amazon VPC CNI Amazon EKS アドオンの追加します。

次のステップを使用して、Amazon VPC CNI Plugin for Kubernetes の Amazon EKS アドオンを作成します。

開始する前に、考慮事項を確認してください。詳細については、「the section called “考慮事項”」を参照してください。

前提条件

以下は、Amazon VPC CNI Plugin for Kubernetes の Amazon EKS アドオンの前提条件です。

- 既存の Amazon EKS クラスター。デプロイするには「はじめに」を参照してください。

- クラスターの既存の AWS Identity and Access Management (IAM) OpenID Connect (OIDC) プロバイダー。既に存在しているかどうかを確認する、または作成するには「the section called “IAM OIDC プロバイダー”」を参照してください。
- AmazonEKS_CNI_Policy IAM ポリシー (クラスターが IPv4 ファミリーを使用している場合)、または IPv6 ポリシー (クラスターが IPv6 ファミリーを使用している場合) がアタッチされた IAM 役割です。VPC CNI 役割の詳細については、「the section called “IRSA 用の設定”」を参照してください。IPv6 ポリシーの詳細については、「the section called “IPv6 ファミリーを使用するクラスター用に IAM ポリシーを作成します”」を参照してください。

> **Important**
>
> Amazon VPC CNI Plugin for Kubernetes バージョンv1.16.0 から v1.16.1 では、CNI 仕様バージョン v1.0.0 が実装されています。バージョン v1.0.0 のCNI 仕様の詳細については、GitHub の「Container Network Interface (CNI) Specification」を参照してください。

手順

前提条件を満たしたら、次のステップを使用してアドオンを作成します。

1. クラスターにインストールされているアドオンのバージョンを確認します。

   ```
   kubectl describe daemonset aws-node --namespace kube-system | grep amazon-k8s-cni: |
    cut -d : -f 3
   ```

   出力例は次のとおりです。

   ```
   v1.16.4-eksbuild.2
   ```

2. クラスターにインストールされているアドオンのタイプを確認します。クラスターを作成するために使用したツールによっては、現在クラスターに Amazon EKS アドオンタイプがインストールされていない場合があります。`#######` の部分は、自分のクラスター名に置き換えます。

   ```
   aws eks describe-addon --cluster-name my-cluster --addon-name vpc-cni --query
    addon.addonVersion --output text
   ```

   バージョン番号が返された場合、Amazon EKS タイプのアドオンがクラスターにインストールされているため、このステップの残りのステップを完了する必要はありません。エラーが返された

場合、クラスターに Amazon EKS タイプのアドオンがインストールされていません。インストールするには、このステップの残りのステップを完了します。

3. 現在インストールされているアドオンの設定を保存します。

```
kubectl get daemonset aws-node -n kube-system -o yaml > aws-k8s-cni-old.yaml
```

4. AWS CLI を使用してアドオンを作成します。AWS マネジメントコンソール または eksctl を使用してアドオンを作成する場合は、「the section called “アドオンを作成する”」を参照して、アドオン名の vpc-cni を指定します。デバイスに沿ったコマンドをコピーします。必要に応じてコマンドに次の変更を加え、変更したコマンドを実行してください。
   - *#######* の部分は、自分のクラスター名に置き換えます。
   - *v1.20.3-eksbuild.1* を、使用しているクラスターバージョンに対して最新バージョンの表に記載されている最新バージョンに置き換えます。最新バージョンの表については、「the section called “Amazon VPC CNI のバージョン”」を参照してください。
   - *111122223333* を、アカウント ID に置き換えて、*AmazonEKSVPCCNIRole* を作成した既存の IAM 役割の名前に置き換えます。ロールを指定するには、クラスター用に IAM OpenID Connect (OIDC) プロバイダーが必要です。クラスター用に持っているかどうかを確認、あるいは作成するには、「the section called “IAM OIDC プロバイダー”」を参照してください。

```
aws eks create-addon --cluster-name my-cluster --addon-name vpc-cni --addon-version
 v1.20.3-eksbuild.1 \
    --service-account-role-arn arn:aws:iam::111122223333:role/AmazonEKSVPCCNIRole
```

   Amazon EKS アドオンのデフォルト設定と競合するカスタム設定を現在のアドオンに適用した場合、作成が失敗する可能性があります。作成に失敗した場合、問題解決に役立つエラーが表示されます。または、前のコマンドに --resolve-conflicts OVERWRITE を追加することもできます。これにより、アドオンは既存のカスタム設定を上書きできます。アドオンを作成したら、カスタム設定で更新できます。

5. クラスターの Kubernetes バージョン用のアドオンの最新バージョンがクラスターに追加されたことを確認します。*#######* の部分は、自分のクラスター名に置き換えます。

```
aws eks describe-addon --cluster-name my-cluster --addon-name vpc-cni --query
 addon.addonVersion --output text
```

   アドオンの作成が完了するまでに数秒かかる場合があります。

   出力例は次のとおりです。

```
v1.20.3-eksbuild.1
```

6. 元のアドオンに対してカスタム設定を行った場合は、Amazon EKS アドオンを作成する前に、前のステップで保存した設定を使用して、カスタム設定で EKS アドオンを更新します。「the section called “更新 (EKS アドオン)”」のステップを実行してください。
7. (オプション)cni-metrics-helper をクラスターにインストールします。メトリクスヘルパーは、ネットワークインターフェイスと IP アドレス情報を収集し、クラスターレベルでメトリクスを集計し、メトリクスを Amazon CloudWatch に発行するために使用できるツールです。詳細についてはGitHub の CNI を参照してください。

Amazon VPC CNI を更新する (Amazon EKS アドオン)

Amazon EKS タイプの Amazon VPC CNI Plugin for Kubernetes アドオンを更新します。Amazon EKS タイプのアドオンをクラスターに追加していない場合は「the section called “作成”」に従ってインストールできます。または「the section called “更新 (自己管理) ”」に従って他のタイプの VPC CNI インストールを更新します。

1. クラスターにインストールされているアドオンのバージョンを確認します。####### の部分は自分のクラスター名に置き換えます。

   ```
   aws eks describe-addon --cluster-name my-cluster --addon-name vpc-cni --query
    "addon.addonVersion" --output text
   ```

   出力例は次のとおりです。

   ```
   v1.20.0-eksbuild.1
   ```

   バージョンを the section called “Amazon VPC CNI のバージョン” の最新バージョンの表と比較します。返されたバージョンが、最新バージョンの表にあるクラスターの Kubernetes バージョンのバージョンと同じである場合は既に最新バージョンがクラスターにインストールされているため、この手順の残りを完了する必要はありません。出力にバージョン番号ではなくエラーが表示される場合はAmazon EKS タイプのアドオンがクラスターにインストールされていません。この手順でアドオンを更新する前に、アドオンを作成する必要があります。VPC CNI アドオンの Amazon EKS タイプを作成するには「the section called “作成”」に従います。
2. 現在インストールされているアドオンの設定を保存します。

```
kubectl get daemonset aws-node -n kube-system -o yaml > aws-k8s-cni-old.yaml
```

3. AWS CLI を使用してアドオンを更新します。AWS マネジメントコンソール または eksctl を使用してアドオンを更新する場合は「the section called “アドオンを更新する”」を参照してください。デバイスに沿ったコマンドをコピーします。必要に応じてコマンドに次の変更を加え、変更したコマンドを実行してください。
   - *#######* の部分は、自分のクラスター名に置き換えます。
   - *v1.20.0-eksbuild.1* を、使用しているクラスターバージョンに対して最新バージョンの表に記載されている最新バージョンに置き換えます。
   - *111122223333* を、アカウント ID に置き換えます。また、*AmazonEKSVPCCNIRole* を、作成した既存の IAM 役割の名前に置き換えます。VPC CNI の IAM 役割を作成するには「the section called “ステップ 1: Amazon VPC CNI plugin for Kubernetes の IAM ロールを作成する”」を参照してください。ロールを指定するには、クラスター用に IAM OpenID Connect (OIDC) プロバイダーが必要です。クラスター用に持っているかどうかを確認、あるいは作成するには「the section called “IAM OIDC プロバイダー”」を参照してください。
   - --resolve-conflicts PRESERVE オプションはアドオンの既存の設定値を保存します。アドオン設定にカスタム値を設定していて、このオプションを使用しない場合、Amazon EKS は値をデフォルト値で上書きします。このオプションを使用する場合、実稼働クラスターのアドオンを更新する前に、非稼動クラスターのフィールドおよび値変更をテストすることをお勧めします。この値を OVERWRITE に変更する場合、すべての設定が Amazon EKS のデフォルト値に変更されます。いずれかの設定にカスタム値を設定した場合、Amazon EKS のデフォルト値で上書きされる可能性があります。この値を none に変更した場合、Amazon EKS は設定の値を一切変更しませんが、更新が失敗する可能性があります。更新に失敗した場合、競合の解決に役立つエラーメッセージが返されます。
   - 構成設定を更新しない場合はコマンドから --configuration-values '{*"env":{"AWS_VPC_K8S_CNI_EXTERNALSNAT":"true"}*}' を削除します。構成設定を更新する場合は*"env":{"AWS_VPC_K8S_CNI_EXTERNALSNAT":"true"}* を、希望する設定に置き換えます。この例ではAWS_VPC_K8S_CNI_EXTERNALSNAT 環境変数は true に設定されています。指定する値は設定スキーマに対して有効である必要があります。設定スキーマが不明である場合は aws eks describe-addon-configuration --addon-name vpc-cni --addon-version *v1.20.0-eksbuild.1* を実行して、*v1.20.0-eksbuild.1* を、設定を表示するアドオンのバージョン番号に置き換えます。出力でスキーマが返されます。既存のカスタム設定があり、それをすべて削除してすべての設定の値を Amazon EKS のデフォルトに戻したい場合はコマンドから *"env":{"AWS_VPC_K8S_CNI_EXTERNALSNAT":"true"}* を削除

して、{} が空になるようにします。各設定の説明についてはGitHub の「CNI 設定変数」を参照してください。

```
aws eks update-addon --cluster-name my-cluster --addon-name vpc-cni --addon-version
 v1.20.3-eksbuild.1 \
    --service-account-role-arn arn:aws:iam::111122223333:role/AmazonEKSVPCCNIRole \
    --resolve-conflicts PRESERVE --configuration-values '{"env":
{"AWS_VPC_K8S_CNI_EXTERNALSNAT":"true"}}'
```

更新が完了するまでに数秒かかる場合があります。

4. アドオンのバージョンが更新されたことを確認します。####### の部分は自分のクラスター名に置き換えます。

```
aws eks describe-addon --cluster-name my-cluster --addon-name vpc-cni
```

更新が完了するまでに数秒かかる場合があります。

出力例は次のとおりです。

```
{
    "addon": {
        "addonName": "vpc-cni",
        "clusterName": "my-cluster",
        "status": "ACTIVE",
        "addonVersion": "v1.20.3-eksbuild.1",
        "health": {
            "issues": []
        },
        "addonArn": "arn:aws:eks:region:111122223333:addon/my-cluster/vpc-
cni/74c33d2f-b4dc-8718-56e7-9fdfa65d14a9",
        "createdAt": "2023-04-12T18:25:19.319000+00:00",
        "modifiedAt": "2023-04-12T18:40:28.683000+00:00",
        "serviceAccountRoleArn": "arn:aws:iam::111122223333:role/
AmazonEKSVPCCNIRole",
        "tags": {},
        "configurationValues": "{\"env\":{\"AWS_VPC_K8S_CNI_EXTERNALSNAT\":\"true
\"}}"
    }
}
```

トラブルシューティング

VPC CNI を v1.13.2 より前のバージョンからアップグレードする場合は、更新後にクラスター内のすべてのノードを置き換える必要があります。v1.13.2 より前のバージョンでは、iptables-legacy バックエンドを使用して、ソース NAT (SNAT) などの適切な機能に必要な iptables ルールを挿入します。

バージョン v1.13.2 は、iptables-wrapper を導入した重要なリリースであり、チェーンとルールを挿入するための適切な iptables バックエンド (iptables-legacy または iptables-nft) を自動的に検出します。この変更は、パフォーマンスの制限によりレガシーバックエンドから離れるというアップストリーム Kubernetes の決定方針と一致しています。

VPC CNI を v1.13.2 より前のバージョンからアップグレードした場合、ノードを置き換える必要があります。これは、iptables-legacy バックエンドと iptables-nft バックエンドの両方にルールを導入すると、プライマリ以外の ENI から発生するトラフィックに予期しない動作が生じる可能性があるためです。

Amazon VPC CNI の更新 (セルフマネージド型アドオン)

> **Important**
>
> セルフマネージド型のアドオンを使用する代わりに、Amazon EKS タイプのアドオンをクラスターに追加することをお勧めします。タイプの違いがよくわからない場合は「the section called “Amazon EKS アドオン”」を参照してください。Amazon EKS アドオンをクラスターに追加する方法については「the section called “アドオンを作成する”」を参照してください。Amazon EKS アドオンを使用できない場合はその理由に関する問題をコンテナロードマップ GitHub リポジトリに送信することをお勧めします。

1. Amazon EKS タイプのアドオンがクラスターにインストールされていないことを確認します。`#####` の部分は自分のクラスター名に置き換えます。

   ```
   aws eks describe-addon --cluster-name my-cluster --addon-name vpc-cni --query
    addon.addonVersion --output text
   ```

   エラーメッセージが返された場合、クラスターに Amazon EKS タイプのアドオンがインストールされていません。アドオンを自己管理するにはこの手順の残りのステップを完了してアドオンを更新します。バージョン番号が返された場合、クラスターに Amazon EKS タイプのアドオンがインストールされています。更新するにはこの手順を使用するのではなく、「the section called “ア

ドオンを更新する”」の手順を使用してください。アドオンタイプの違いがよくわからない場合は「the section called “Amazon EKS アドオン”」を参照してください。

2. クラスターに現在インストールされているコンテナイメージのバージョンを確認します。

```
kubectl describe daemonset aws-node --namespace kube-system | grep amazon-k8s-cni: |
 cut -d : -f 3
```

出力例は次のとおりです。

```
v1.20.0-eksbuild.1
```

出力にビルド番号が含まれていない場合があります。

3. 現在の設定をバックアップして、バージョンを更新した後も同じ設定を構成できるようにします。

```
kubectl get daemonset aws-node -n kube-system -o yaml > aws-k8s-cni-old.yaml
```

利用可能なバージョンを確認し、更新するバージョンの変更について理解するには、GitHub の「Releases」を参照してください。新しいバージョンが GitHub で利用できる場合でも、利用可能な最新バージョンの表に記載されている同じ major.minor.patch バージョンに更新することをお勧めします。利用可能な最新バージョンの表については「the section called “Amazon VPC CNI のバージョン”」を参照してください。表に記載されているビルドバージョンはGitHub に記載されているセルフマネージドバージョンでは指定されていません。次のオプションのいずれかでタスクを完了して、バージョンを更新します。

- アドオンのカスタム設定がない場合は更新先のリリースの GitHub の To apply this release: の見出しの下にあるコマンドを実行してください。
- カスタム設定がある場合は次のコマンドを実行してマニフェストファイルをダウンロードします。*https://raw.githubusercontent.com/aws/amazon-vpc-cni-k8s/v1.20.0/config/master/aws-k8s-cni.yaml* を、更新先の GitHub 上のリリースの URL に変更します。

  ```
  curl -O https://raw.githubusercontent.com/aws/amazon-vpc-cni-k8s/v1.20.3/config/
  master/aws-k8s-cni.yaml
  ```

  必要に応じて、前のステップで作成したバックアップのカスタム設定でマニフェストを変更し、変更したマニフェストをクラスターに適用します。ノードがイメージの取得元のプラ

イベート Amazon EKS Amazon ECR リポジトリにアクセスできない場合 (マニフェストの `image:` で始まる行を参照)、イメージをダウンロードして自分のリポジトリにコピーし、リポジトリからイメージを取得するようにマニフェストを変更する必要があります。詳細については「the section called “リポジトリにイメージをコピーする”」を参照してください。

```
kubectl apply -f aws-k8s-cni.yaml
```

4. 新しいバージョンがクラスターにインストールされたことを確認します。

```
kubectl describe daemonset aws-node --namespace kube-system | grep amazon-k8s-cni: |
 cut -d : -f 3
```

出力例は次のとおりです。

```
v1.20.3
```

5. (オプション)`cni-metrics-helper` をクラスターにインストールします。メトリクスヘルパーは、ネットワークインターフェイスと IP アドレス情報を収集し、クラスターレベルでメトリクスを集計し、メトリクスを Amazon CloudWatch に発行するために使用できるツールです。詳細については[GitHub の CNI](#) を参照してください。

IRSA を使用するように Amazon VPC CNI プラグインを設定する

[Amazon VPC CNI plugin for Kubernetes](#) は、Amazon EKS クラスター内の Pod ネットワーキング用のネットワークプラグインです。プラグインは Kubernetes ポッドに VPC IP アドレスを割り当て、各ノードのポッドに必要なネットワークを設定する役割を果たしています。

> **Note**
>
> Amazon VPC CNI プラグインは、Amazon EKS Pod Identity もサポートしています。詳細については、the section called “IAM ロールを割り当てる” を参照してください。

プラグイン:

- AWS アイデンティティとアクセス管理 (IAM) のアクセス許可が必要です。クラスターが `IPv4` ファミリーを使用する場合、このアクセス許可は、`AmazonEKS_CNI_Policy` AWS 管理ポリシーで指定されます。クラスターが `IPv6` ファミリーを使用する場合には、作成した IAM ポリシー

にアクセス許可を追加する必要があります。手順については、「the section called “IPv6 ファミリーを使用するクラスター用に IAM ポリシーを作成します”」を参照してください。このポリシーはAmazon EKS ノード IAM ロール または 個別の IAM ロールにアタッチすることができます。Amazon EKS ノード IAM ロールにポリシーをアタッチする手順については、the section called “ノード の IAM ロール” を参照してください。このトピックで詳細に説明するように、別のロールに割り当てることをお勧めします。

- デプロイ時に作成すると、aws-node という名前の Kubernetes サービスアカウントを使用するように設定されます。サービスアカウントは aws-node という名前の Kubernetes clusterrole にバインドされます。これには、必要な Kubernetes アクセス許可が割り当てられています。

> **Note**
>
> IMDS へのアクセスをブロックする場合を除き、Amazon VPC CNI plugin for Kubernetes 用の Pod には、Amazon EKS ノード IAM ロールに割り当てられたパーミッションへのアクセス権があります。詳細については「ワーカーノードに割り当てられたインスタンスプロファイルへのアクセスを制限する」を参照してください。

- 既存の Amazon EKS クラスターが必要です。デプロイするには、はじめに を参照してください。
- クラスターに、既存の AWS Identity and Access Management (IAM) OpenID Connect (OIDC) プロバイダーが必要です。既に存在しているかどうかを確認する、または作成するには、the section called “IAM OIDC プロバイダー” を参照してください。

ステップ 1: Amazon VPC CNI plugin for Kubernetes の IAM ロールを作成する

1. クラスターで使用する IP ファミリを決定します。

```
aws eks describe-cluster --name my-cluster | grep ipFamily
```

出力例は次のとおりです。

```
"ipFamily": "ipv4"
```

この出力では代わりに ipv6 が返されることがあります。

2. IAM ロールを作成します。IAM ロールを作成するにはeksctl または kubectl および AWS CLI を使用してます。

eksctl

- クラスターの IP ファミリーに適合するコマンドを使用して IAM ロールを作成し、そのロールに IAM ポリシーをアタッチします。このコマンドでは、IAM ロールを作成する AWS CloudFormation スタックを作成およびデプロイし、そのために指定したポリシーをアタッチします。さらに、既存の aws-node Kubernetes サービスアカウントに対し、作成された IAM ロールの ARN をアノテーションします。

  - IPv4

    ####### を独自の値に置き換えます。

    ```
    eksctl create iamserviceaccount \
        --name aws-node \
        --namespace kube-system \
        --cluster my-cluster \
        --role-name AmazonEKSVPCCNIRole \
        --attach-policy-arn arn:aws:iam::aws:policy/AmazonEKS_CNI_Policy \
        --override-existing-serviceaccounts \
        --approve
    ```

  - IPv6

    ####### を独自の値に置き換えます。*111122223333* を、ご自身のアカウント ID に置き換えます。また、*AmazonEKS_CNI_IPv6_Policy* を、IPv6 ポリシー名に置き換えます。IPv6 ポリシーがない場合はthe section called “IPv6 ファミリーを使用するクラスター用に IAM ポリシーを作成します” を参照して作成します。クラスターで IPv6 を使用するにはいくつかの要件を満たす必要があります。詳細については、the section called “IPv6” を参照してください。

    ```
    eksctl create iamserviceaccount \
        --name aws-node \
        --namespace kube-system \
        --cluster my-cluster \
        --role-name AmazonEKSVPCCNIRole \
        --attach-policy-arn arn:aws:iam::111122223333:policy/
    AmazonEKS_CNI_IPv6_Policy \
        --override-existing-serviceaccounts \
        --approve
    ```

kubectl と AWS CLI

i. クラスターの OIDC プロバイダーの URL を表示します。

```
aws eks describe-cluster --name my-cluster --query
 "cluster.identity.oidc.issuer" --output text
```

出力例は次のとおりです。

```
https://oidc.eks.region-code.amazonaws.com/id/EXAMPLED539D4633E53DE1B71EXAMPLE
```

出力が返されない場合は[クラスター用の IAM OIDC プロバイダーを作成する]必要があります。

ii. 次の内容を *vpc-cni-trust-policy.json* という名前のファイルにコピーします。*111122223333* を、ご自身のアカウント ID および前のステップで返された出力 *EXAMPLED539D4633E53DE1B71EXAMPLE* に置き換えます。*#####* を、クラスターのある AWS リージョンに置き換えます。

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "Federated": "arn:aws:iam::111122223333:oidc-provider/
oidc.eks.us-east-1.amazonaws.com/id/EXAMPLED539D4633E53DE1B71EXAMPLE"
            },
            "Action": "sts:AssumeRoleWithWebIdentity",
            "Condition": {
                "StringEquals": {
                    "oidc.eks.us-east-1.amazonaws.com/id/
EXAMPLED539D4633E53DE1B71EXAMPLE:aud": "sts.amazonaws.com",
                    "oidc.eks.us-east-1.amazonaws.com/id/
EXAMPLED539D4633E53DE1B71EXAMPLE:sub": "system:serviceaccount:kube-system:aws-
node"
                }
            }
        }
    ]
}
```

iii. ロールを作成します。*AmazonEKSVPCCNIRole* は任意の名前に置き換えることができます。

```
aws iam create-role \
  --role-name AmazonEKSVPCCNIRole \
  --assume-role-policy-document file://"vpc-cni-trust-policy.json"
```

iv. 必要な IAM ポリシーをロールにアタッチします。クラスターの IP ファミリに適合したコマンドを実行してください。

- IPv4

```
aws iam attach-role-policy \
  --policy-arn arn:aws:iam::aws:policy/AmazonEKS_CNI_Policy \
  --role-name AmazonEKSVPCCNIRole
```

- IPv6

111122223333 はご自身のアカウント ID に置き換え、*AmazonEKS_CNI_IPv6_Policy* はご使用の IPv6 ポリシーの名前に置き換えます。IPv6 ポリシーがない場合はthe section called “IPv6 ファミリーを使用するクラスター用に IAM ポリシーを作成します” を参照して作成します。クラスターで IPv6 を使用するにはいくつかの要件を満たす必要があります。詳細については、the section called “IPv6” を参照してください。

```
aws iam attach-role-policy \
  --policy-arn arn:aws:iam::111122223333:policy/AmazonEKS_CNI_IPv6_Policy \
  --role-name AmazonEKSVPCCNIRole
```

v. 次のコマンドを実行し、先に作成した IAM ロールの ARN で aws-node サービスアカウントをアノテーションします。example の値は、独自の値に置き換えます。

```
kubectl annotate serviceaccount \
    -n kube-system aws-node \
    eks.amazonaws.com/role-arn=arn:aws:iam::111122223333:role/
AmazonEKSVPCCNIRole
```

3. (オプション) Kubernetes サービスアカウントで使用されている AWS Security Token Service エンドポイントタイプを設定します。詳細については、the section called “STS エンドポイント” を参照してください。

ステップ 2: Amazon VPC CNI plugin for Kubernetes の Pod を再デプロイする

1. 認証情報環境変数を適用するために、サービスアカウントに関連付けられている既存の Pod を削除して再作成します。現在アノテーションなしで実行されている Pod には、アノテーションは適用されません。次のコマンドは、既存の aws-node DaemonSet Pod を削除し、サービスアカウントのアノテーションを使用してデプロイします。

```
kubectl delete Pods -n kube-system -l k8s-app=aws-node
```

2. Pod がすべて再起動したことを確認します。

```
kubectl get pods -n kube-system -l k8s-app=aws-node
```

3. Pod の 1 つについて説明し、AWS_WEB_IDENTITY_TOKEN_FILE および AWS_ROLE_ARN 環境変数が存在することを確認します。*cpjw7* を、前のステップの出力で返された、いずれかの Pod の名前に置き換えます。

```
kubectl describe pod -n kube-system aws-node-cpjw7 | grep 'AWS_ROLE_ARN:\|
AWS_WEB_IDENTITY_TOKEN_FILE:'
```

出力例は次のとおりです。

```
AWS_ROLE_ARN:                 arn:aws:iam::111122223333:role/AmazonEKSVPCCNIRole
      AWS_WEB_IDENTITY_TOKEN_FILE:  /var/run/secrets/eks.amazonaws.com/
serviceaccount/token
      AWS_ROLE_ARN:                           arn:aws:iam::111122223333:role/
AmazonEKSVPCCNIRole
      AWS_WEB_IDENTITY_TOKEN_FILE:            /var/run/secrets/eks.amazonaws.com/
serviceaccount/token
```

Pod には 2 つのコンテナが含まれているため、重複した結果の 2 つのセットが返されます。両方のコンテナの値は同じです。

Pod で AWS リージョンエンドポイントを使用している場合、前の出力では下記の行も返されています。

```
AWS_STS_REGIONAL_ENDPOINTS=regional
```

ステップ 3: ノードの IAM ロールから CNI ポリシーを削除する

現在、Amazon EKS ノード IAM ロール に AmazonEKS_CNI_Policy IAM (IPv4) ポリシーまたは IPv6 ポリシー がアタッチされており、さらに、別の IAM ロールを作成し、このロールに対し前出のポリシーをアタッチし、そのロールを aws-node Kubernetes サービスアカウントに割り当てている場合、クラスターの IP ファミリーに適合する AWS CLI コマンドを使用して、ノードのロールからポリシーを削除することをお勧めします。*AmazonEKSNodeRole* を、ノードのロールの名前に置き換えます。

- IPv4

```
aws iam detach-role-policy --role-name AmazonEKSNodeRole --policy-arn
 arn:aws:iam::aws:policy/AmazonEKS_CNI_Policy
```

- IPv6

  111122223333 はご自身のアカウント ID に置き換え、*AmazonEKS_CNI_IPv6_Policy* はご使用の IPv6 ポリシーの名前に置き換えます。

```
aws iam detach-role-policy --role-name AmazonEKSNodeRole --policy-arn
 arn:aws:iam::111122223333:policy/AmazonEKS_CNI_IPv6_Policy
```

IPv6 ファミリーを使用するクラスター用に IAM ポリシーを作成します

IPv6 ファミリーを使用するクラスターを作成し、そのクラスターでバージョン 1.10.1 以降の Amazon VPC CNI plugin for Kubernetes アドオンが設定されている場合は、IAM ロールに割り当てることができる IAM ポリシーを作成する必要があります。作成時に IPv6 ファミリーの使用を設定していない、既存のクラスターにおいて、IPv6 を使用する場合には新しいクラスターを作成する必要があります。クラスターでの IPv6 使用の詳細については「the section called “IPv6”」を参照してください。

1. 次のテキストをコピーし、vpc-cni-ipv6-policy.json という名前のファイルに保存します。

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "ec2:AssignIpv6Addresses",
```

```
                "ec2:DescribeInstances",
                "ec2:DescribeTags",
                "ec2:DescribeNetworkInterfaces",
                "ec2:DescribeInstanceTypes",
                "ec2:DescribeSubnets"
            ],
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "ec2:CreateTags"
            ],
            "Resource": [
                "arn:aws:ec2:*:*:network-interface/*"
            ]
        }
    ]
}
```

2. IAM ポリシーを作成します。

```
aws iam create-policy --policy-name AmazonEKS_CNI_IPv6_Policy --policy-document
 file://vpc-cni-ipv6-policy.json
```

VPC CNI モードと設定について説明します。

Amazon VPC CNI plugin for Kubernetes は、Pod のネットワーキングを提供します。使用可能なネットワーク機能の詳細については次の表を参照してください。

ネットワーキング機能	詳細情報
クラスター、Pod、サービスに IPv6 アドレスを割り当てるようにクラスターを設定する	the section called “IPv6”
Pod に IPv4 Source Network Address Translation を使用する	the section called “アウトバウンドトラフィック”
Pod との間のネットワークトラフィックを制限する	the section called “トラフィックを制限する”

ネットワーキング機能	詳細情報
Pod IP アドレスのサブネット選択を設定する	the section called “サブネットの選択”
ノードでセカンダリネットワークインターフェイスをカスタマイズする	the section called “カスタムネットワーキング”
ノードの IP アドレスを増やす	the section called “IP アドレスを増やす”
Pod ネットワークトラフィックにセキュリティグループを使用する	the section called “ポッドのセキュリティグループ”
Pod に複数のネットワークインターフェイスを使用する	the section called “複数のインターフェイス”

クラスター、Pod、サービスに対する IPv6 アドレスの説明

適用対象: Amazon EC2 インスタンスを持つ Pod と Fargate Pod

Kubernetes のデフォルトでは、IPv4 アドレスが Pod とサービスに割り当てられます。Pod とサービスに IPv4 アドレスを割り当てる代わりに、IPv6 アドレスを割り当てるようにクラスターを設定できます。Amazon EKS は、Kubernetes がサポートしていても、デュアルスタックのポッドまたはサービスをサポートしません。つまり、Pod とサービスに IPv4 アドレスと IPv6 アドレスの両方を割り当てることはできません。

そのクラスターに使用する IP ファミリーは、クラスターの作成時に選択します。クラスターの作成後は、ファミリーを変更できません。

Amazon EKS IPv6 クラスターをデプロイするチュートリアルについては、「the section called “デプロイ”」を参照してください。

この機能を使用する際の考慮事項を次に示します。

IPv6 の機能のサポート

- Windows サポートなし: Windows の Pod とサービスはサポートされていません。
- Nitro ベースの EC2 ノードが必要: IPv6 は、AWS Nitro ベースの Amazon EC2 または Fargate ノードでのみ使用が可能です。
- EC2 ノードおよび Fargate ノードがサポートされる:IPv6 は、Amazon EC2 ノードと Fargate ノードで the section called “ポッドのセキュリティグループ” とともに使用できます。

- Outposts はサポートされない: IPv6 は Amazon EKS on AWS Outposts とともに使用できません。
- FSx for Lustre はサポートされない: the section called “Amazon FSx for Lustre” はサポートされていません。
- カスタムネットワーキングはサポートされない: IP アドレスの枯渇を緩和するために、これまで使用していた the section called “カスタムネットワーキング” の代わりとして、IPv6 を使用することができます。IPv6 では、カスタムネットワーキングを使用することはできません。クラスターにおいて、ネットワークの分離用としてカスタムネットワーキングを使用する場合は、引き続き、IPv4 ファミリーによるカスタムネットワーキングを使用する必要があります。

IP アドレスの割り当て

- Kubernetes サービス: Kubernetes サービスには IPv6 アドレスのみが割り当てられます。これらには、IPv4 アドレスは割り当てられません。
- ポッド: ポッドには IPv6 アドレスとホストローカル IPv4 アドレスが割り当てられます。ホストローカル IPv4 アドレスは、VPC CNI と連鎖されたホストローカル CNI プラグインを使用して割り当てられ、そのアドレスは Kubernetes コントロールプレーンに報告されません。これは、ポッドが別の Amazon VPC またはインターネット内の外部 IPv4 リソースと通信する必要がある場合にのみ使用されます。ホストローカル IPv4 アドレスは、ワーカーノードのプライマリ ENI のプライマリ IPv4 アドレスに (VPC CNI によって) SNAT されます。
- ポッドとサービス: ポッドおよびサービスは IPv4 アドレスではなく、IPv6 アドレスのみを受け取ります。ポッドが外部 IPv4 エンドポイントと通信する必要がある場合、ノード自体で NAT が使用されます。この組み込み NAT 機能により、DNS64 および NAT64 が不要になります。パブリックインターネットアクセスを必要とするトラフィックの場合、ポッドのトラフィックはパブリック IP アドレスに変換されたソースネットワークアドレスになります。
- ルーティングアドレス: ポッドが VPC の外部で通信する場合、元の IPv6 アドレスは保持されます (ノードの IPv6 アドレスに変換されない)。このトラフィックは、インターネットゲートウェイまたは Egress-Only のインターネットゲートウェイを介して直接ルーティングされます。
- ノード: すべてのノードには、IPv4 と IPv6 のアドレスが割り当てられています。
- Fargate ポッド: 各 Fargate Pod は、デプロイ先のサブネット用に指定された CIDR から、IPv6 アドレスを受け取ります。Fargate Pod を実行するための基盤ハードウェアユニットは、そのユニットがデプロイされているサブネットに割り当てられている CIDR から、一意の IPv4 および IPv6 アドレスを取得します。

EKS で **IPv6** を使用する方法

- 新しいクラスターを作成する: 新しいクラスターを作成し、そのクラスターで IPv6 ファミリーの使用を指定する必要があります。これより前のバージョンからクラスターを更新して、IPv6 ファミリーを有効化することはできません。新しいクラスターを作成する手順については、「the section called “デプロイ”」を参照してください。
- 最新の VPC CNI を使用する: Amazon VPC CNI バージョン `1.10.1` 以降をデプロイします。このバージョン以降がデフォルトでデプロイされます。アドオンのデプロイ後は、クラスター内のすべてのノードグループ内にあるすべてのノードを削除しない限り、Amazon VPC CNI アドオンを `1.10.1` より前のバージョンにダウングレードすることはできません。
- **IPv6** 用の VPC CNI を設定する: Amazon EC2 ノードを使用する場合は、IP プレフィックスの委任と IPv6 を使用して Amazon VPC CNI アドオンを設定する必要があります。クラスターの作成時に IPv6 ファミリーを選択した場合は、バージョン `1.10.1` のアドオンが、この設定のデフォルトになります。これは、セルフマネージド型のアドオン、および Amazon EKS アドオンの両方に当てはまります。IP プレフィックス委任の詳細については、「the section called “IP アドレスを増やす”」を参照してください。
- **IPv4** アドレスと **IPv6** アドレスを設定する: クラスターを作成する際に指定する VPC とサブネットには、それらに割り当てられた IPv6 CIDR ブロックが必要です。同時に、IPv4 CIDR ブロックも割り当てられている必要があります。これは、IPv6 のみを使用する場合でも、VPC が機能するには IPv4 CIDR ブロックが必要になるためです。詳細については、「Amazon VPC ユーザーガイド」の「IPv6 CIDR ブロックと VPC の関連付け」を参照してください。
- IPv6 アドレスをノードに自動割り当てする: ノードを作成する際は、IPv6 アドレスを自動割り当てするように設定されたサブネットを指定する必要があります。指定していない場合、ノードをデプロイできません。自動割り当ての設定はデフォルトで無効になっています。詳細については、「Amazon VPC ユーザーガイド」の「サブネットのパブリック IPv6 アドレス属性を変更する」を参照してください。
- ルートテーブルを設定して **IPv6** を使用する: サブネットに割り当てられるルートテーブルには、IPv6 アドレス用のルートが必要です。詳細については、「Amazon VPC ユーザーガイド」の「既存の VPC を IPv4 から IPv6 に移行する」を参照してください。
- **IPv6** のセキュリティグループを設定する: セキュリティグループは IPv6 アドレスを許可する必要があります。詳細については、「Amazon VPC ユーザーガイド」の「既存の VPC を IPv4 から IPv6 に移行する」を参照してください。
- ロードバランサーを設定する: AWS Load Balancer Controller のバージョン `2.3.1` 以降を使用して、「the section called “アプリケーション負荷分散”」を使用した HTTP アプリケーションのロードバランシング、または、「the section called “ネットワーク負荷分散”」を使用したネットワー

クトラフィックの IPv6 Pod へのロードバランシングを行います。いずれの場合もインスタンスモードではなく IP モードを使用してください。詳細については、「the section called “ AWS ロードバランサーコントローラー”」を参照してください。

- **IPv6** IAM ポリシーを追加する: ノード IAM または CNI IAM ロールには、IPv6 のIAM ポリシーをアタッチする必要があります。上記 2 つの中では、CNI IAM ロールへのアタッチが推奨されます。詳細については、「the section called “IPv6 ファミリーを使用するクラスター用に IAM ポリシーを作成します”」および「the section called “ステップ 1: Amazon VPC CNI plugin for Kubernetes の IAM ロールを作成する”」を参照してください。
- すべてのコンポーネントを評価する: アプリケーション、Amazon EKS アドオン、および、IPv6 クラスターのデプロイ前に統合した AWS サービスに関しては、包括的な評価を実施します。これにより、IPv6 を使用した場合も、すべてが想定どおりに動作することを保証できます。

Amazon EKS **IPv6** クラスターとマネージド Amazon Linux ノードをデプロイする

このチュートリアルでは、IPv6 ファミリーを使用する IPv6 Amazon VPC と Amazon EKS クラスターのデプロイ、および Amazon EC2 Amazon Linux ノードを使用するマネージド型ノードグループのデプロイ方法を説明します。IPv6 クラスター内では、Amazon EC2 Windows のノードをデプロイすることはできません。Fargate ノードをクラスターにデプロイすることもできますが、理解しやすくするため、これらの手順はこのトピックでは説明していません。

前提条件

このチュートリアルを開始する前に以下を完了してください。

Amazon EKS クラスターの作成と管理に必要な次のツールおよびリソースをインストールおよび設定します。

- すべての設定を習熟し、要件を満たす設定でクラスターをデプロイすることをお勧めします。詳細については、the section called “クラスターを作成する”、the section called “マネージド型ノードグループ”、ならびにこのトピックの「考慮事項」を参照してください。一部の設定の有効化は、クラスターの作成時にのみ行えます。
- デバイスまたは AWS CloudShell に、kubectl コマンドラインツールがインストールされていること。バージョンはご使用のクラスターの Kubernetes バージョンと同じか、1 つ前のマイナーバージョン以前、あるいはそれより新しいバージョンが使用できます。例えば、クラスターのバージョンが 1.29 である場合、kubectl のバージョン 1.28、1.29、または 1.30 が使用できます。kubectl をインストールまたはアップグレードする方法については、the section called “kubectl と eksctl のセットアップ” を参照してください。

- 使用している IAM セキュリティプリンシパルは、Amazon EKS の IAM ロール、サービスにリンクされたロール、AWS CloudFormation、VPC、関連リソースを使用するために許可が必要です。詳細については、「IAM ユーザーガイド」の「アクション」および「サービスにリンクされたロールの使用」を参照してください。
- `eksctl` を使用する場合は、コンピュータにバージョン `0.215.0` 以降をインストールしてください。それをインストールまたはアップグレードするには、`eksctl` ドキュメントの「インストール」を参照してください。
- ご使用のデバイスまたは AWS CloudShell で、バージョン `2.12.3` 以降、または AWS コマンドラインインターフェイス (AWS CLI) のバージョン `1.27.160` 以降がインストールおよび設定されていること。現在のバージョンを確認するには、`aws --version | cut -d / -f2 | cut -d ' ' -f1` を参照してください。`yum`、`apt-get`、macOS 用の Homebrew などのパッケージマネージャーは、多くの場合 AWS CLI の最新バージョンより数バージョン古くなっています。最新バージョンをインストールするには、「 AWS コマンドラインインターフェイスユーザーガイド」の「インストール」および「aws configure を使用したクイック設定」を参照してください。AWS クラウドシェル にインストールされている AWS CLI バージョンも最新バージョンより数バージョン遅れることがあります。更新するには、「AWS CloudShell ユーザーガイド」の「ホームディレクトリへの AWS CLI のインストール」を参照してください。AWS CloudShell を使用する場合は、バージョン 2.12.3 以降または 1.27.160 以降の AWS CLI をインストールする必要があります。AWS CloudShell には、デフォルトとして AWS CLI の古いバージョンがインストールされている可能性があります。

eksctl または CLI を使用して `IPv6` クラスターをデプロイできます。

eksctl を使用して IPv6 クラスターをデプロイする

a. `ipv6-cluster.yaml` ファイルを作成します。デバイスに沿ったコマンドをコピーします。必要に応じてコマンドに次の変更を加え、変更したコマンドを実行してください:

- *`my-cluster`* の部分は、自分のクラスター名に置き換えます。この名前には英数字 (大文字と小文字が区別されます) とハイフンのみを使用できます。先頭の文字は英数字である必要があります。また、100 文字より長くすることはできません。名前は、クラスターを作成する AWS リージョンおよび AWS アカウント内で一意である必要があります。
- *`region-code`* は、Amazon EKS がサポートする任意の AWS リージョンに置き換えます。AWS リージョンの一覧については、AWS の全般的なリファレンスガイドの Amazon EKS エンドポイントとクォータを参照してください。
- `version` の値には、クラスターのバージョンを設定する必要があります。詳細については、「Amazon EKS supported versions」を参照してください。

- *my-nodegroup* を、ノードグループの名前に置き換えます。ノードグループ名は 63 文字以下である必要があります。先頭は文字または数字でなければなりませんが、残りの文字にはハイフンおよびアンダースコアを含めることもできます。
- *t3.medium* の部分は、任意の AWS Nitro System インスタンスタイプに置き換えます。

```
cat >ipv6-cluster.yaml <<EOF
---
apiVersion: eksctl.io/v1alpha5
kind: ClusterConfig

metadata:
  name: my-cluster
  region: region-code
  version: "X.XX"

kubernetesNetworkConfig:
  ipFamily: IPv6

addons:
  - name: vpc-cni
    version: latest
  - name: coredns
    version: latest
  - name: kube-proxy
    version: latest

iam:
  withOIDC: true

managedNodeGroups:
  - name: my-nodegroup
    instanceType: t3.medium
EOF
```

b. クラスターを作成します。

```
eksctl create cluster -f ipv6-cluster.yaml
```

クラスターの作成には数分かかります。出力の最後の行が表示されるまで、次の手順には進まないでください。この行は、下記のような出力になります。

```
[...]
[#]  EKS cluster "my-cluster" in "region-code" region is ready
```

c. デフォルトの Pod に、IPv6 アドレスが割り当てられていることを確認します。

```
kubectl get pods -n kube-system -o wide
```

出力例は次のとおりです。

```
NAME                       READY   STATUS    RESTARTS   AGE     IP
                    NODE                                              NOMINATED NODE
 READINESS GATES
aws-node-rslts             1/1     Running   1          5m36s
 2600:1f13:b66:8200:11a5:ade0:c590:6ac8   ip-192-168-34-75.region-
code.compute.internal   <none>           <none>
aws-node-t74jh             1/1     Running   0          5m32s
 2600:1f13:b66:8203:4516:2080:8ced:1ca9   ip-192-168-253-70.region-
code.compute.internal  <none>           <none>
coredns-85d5b4454c-cw7w2   1/1     Running   0          56m
 2600:1f13:b66:8203:34e5::                ip-192-168-253-70.region-
code.compute.internal  <none>           <none>
coredns-85d5b4454c-tx6n8   1/1     Running   0          56m
 2600:1f13:b66:8203:34e5::1               ip-192-168-253-70.region-
code.compute.internal  <none>           <none>
kube-proxy-btpbk           1/1     Running   0          5m36s
 2600:1f13:b66:8200:11a5:ade0:c590:6ac8   ip-192-168-34-75.region-
code.compute.internal   <none>           <none>
kube-proxy-jjk2g           1/1     Running   0          5m33s
 2600:1f13:b66:8203:4516:2080:8ced:1ca9   ip-192-168-253-70.region-
code.compute.internal  <none>           <none>
```

d. デフォルトのサービスに、IPv6 アドレスが割り当てられていることを確認します。

```
kubectl get services -n kube-system -o wide
```

出力例は次のとおりです。

```
NAME       TYPE        CLUSTER-IP          EXTERNAL-IP   PORT(S)         AGE
 SELECTOR
```

```
kube-dns   ClusterIP   fd30:3087:b6c2::a   <none>        53/UDP,53/TCP   57m   k8s-
app=kube-dns
```

e. (オプション) サンプルアプリケーションをデプロイするか、AWS ロードバランサーコントローラーとサンプルアプリケーションをデプロイして、「the section called “アプリケーション負荷分散”」に従い HTTP アプリケーションをロードバランシングするか、「the section called “ネットワーク負荷分散”」に従いネットワークトラフィックを IPv6 Pod にロードバランシングします。

f. このチュートリアルのために作成したクラスターとノードの使用が終了したら、それらのリソースは、次のコマンドにより削除しておきます。

```
eksctl delete cluster my-cluster
```

AWS CLI を使用して IPv6 クラスターをデプロイする

> **Important**
>
> - この手順のすべてのステップは、同一のユーザーとして実行する必要があります。現在のユーザーを確認するには、次のコマンドを実行します。
>
>   ```
>   aws sts get-caller-identity
>   ```
>
> - この手順のすべてのステップは、同一のシェル内で実行する必要があります。いくつかのステップには、これより前のステップで設定した変数を使用します。変数の値が異なるシェル内で設定されていると、その変数を使用するステップは正しく機能しません。AWS CloudShell を使用して次の手順を実行する際、約 20 ~ 30 分の間キーボードまたはポインタによる操作がないと、シェルセッションが終了することを銘記しておいてください。実行中のプロセスは、操作数としてカウントされません。
> - 各命令は Bash シェル用に記述されているので、他のシェルでは修正が必要な場合があります。

この手順の各ステップにおいて、すべての example values は、ご自分が使用する値に置き換える必要があります。

a. 以下のコマンドを実行して、後のステップで使用するいくつかの変数を設定します。*region-code* は、リソースをデプロイする AWS リージョンに置き換えます。この値には、Amazon EKS でサポートされている任意の AWS リージョンが設定できます。AWS リージョンの一覧につい

ては、AWS の全般的なリファレンスガイドの Amazon EKS エンドポイントとクォータを参照してください。*my-cluster* の部分は、自分のクラスター名に置き換えます。この名前には英数字 (大文字と小文字が区別されます) とハイフンのみを使用できます。先頭の文字は英数字である必要があります。また、100 文字より長くすることはできません。名前は、クラスターを作成する AWS リージョンおよび AWS アカウント内で一意である必要があります。*my-nodegroup* を、ノードグループの名前に置き換えます。ノードグループ名は 63 文字以下である必要があります。先頭は文字または数字でなければなりませんが、残りの文字にはハイフンおよびアンダースコアを含めることもできます。*111122223333* は、ご自分のアカウント ID に置き換えます。

```
export region_code=region-code
export cluster_name=my-cluster
export nodegroup_name=my-nodegroup
export account_id=111122223333
```

b. パブリックサブネットとプライベートサブネットを持ち、Amazon EKS と IPv6 の要件を満たす Amazon VPC を作成します。

i. 次のコマンドを実行して、AWS CloudFormation スタック名のための変数を設定します。*my-eks-ipv6-vpc* は、任意の名前に置き換えることができます。

```
export vpc_stack_name=my-eks-ipv6-vpc
```

ii. AWS CloudFormation のテンプレートを使用して IPv6 VPC を作成します。

```
aws cloudformation create-stack --region $region_code --stack-name $vpc_stack_name
 \
  --template-url https://s3.us-west-2.amazonaws.com/amazon-eks/
cloudformation/2020-10-29/amazon-eks-ipv6-vpc-public-private-subnets.yaml
```

スタックが作成されるまで数分かかります。以下のコマンドを実行してください。コマンドの出力が「CREATE_COMPLETE」になるまで、次のステップに進まないでください。

```
aws cloudformation describe-stacks --region $region_code --stack-name
 $vpc_stack_name --query Stacks[].StackStatus --output text
```

iii. 作成されたパブリックサブネットの ID を取得します。

```
aws cloudformation describe-stacks --region $region_code --stack-name
 $vpc_stack_name \
```

```
    --query='Stacks[].Outputs[?OutputKey==`SubnetsPublic`].OutputValue' --output
 text
```

出力例は次のとおりです。

```
subnet-0a1a56c486EXAMPLE,subnet-099e6ca77aEXAMPLE
```

iv. 作成されたパブリックサブネットで、IPv6 アドレスの自動割り当てオプションを有効にします。

```
aws ec2 modify-subnet-attribute --region $region_code --subnet-id
 subnet-0a1a56c486EXAMPLE --assign-ipv6-address-on-creation
aws ec2 modify-subnet-attribute --region $region_code --subnet-id
 subnet-099e6ca77aEXAMPLE --assign-ipv6-address-on-creation
```

v. テンプレートによって作成されたサブネットおよびセキュリティグループの名前を、デプロイされた AWS CloudFormation スタックから取得します。これらの名前は、後のステップで使用するために変数に格納しておきます。

```
security_groups=$(aws cloudformation describe-stacks --region $region_code --
stack-name $vpc_stack_name \
    --query='Stacks[].Outputs[?OutputKey==`SecurityGroups`].OutputValue' --output
 text)

public_subnets=$(aws cloudformation describe-stacks --region $region_code --stack-
name $vpc_stack_name \
    --query='Stacks[].Outputs[?OutputKey==`SubnetsPublic`].OutputValue' --output
 text)

private_subnets=$(aws cloudformation describe-stacks --region $region_code --
stack-name $vpc_stack_name \
    --query='Stacks[].Outputs[?OutputKey==`SubnetsPrivate`].OutputValue' --output
 text)

subnets=${public_subnets},${private_subnets}
```

c. クラスターの IAM ロールを作成して、そのロールに必要な Amazon EKS IAM 管理ポリシーをアタッチします。Amazon EKS によって管理される Kubernetes クラスターは、そのサービスで使用するリソースを管理するために、ユーザーに代わって他の AWS のサービスを呼び出します。

i. 次のコマンドを実行して、eks-cluster-role-trust-policy.json ファイルを作成します。

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "eks.amazonaws.com"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
```

ii. 次のコマンドを実行して、ロール名の変数を設定します。*myAmazonEKSClusterRole* は、任意の名前に置き換えることができます。

```
export cluster_role_name=myAmazonEKSClusterRole
```

iii. ロールを作成します。

```
aws iam create-role --role-name $cluster_role_name --assume-role-policy-document
 file://"eks-cluster-role-trust-policy.json"
```

iv. IAM ロールの ARN を取得し、後のステップで使用するために変数に格納します。

```
CLUSTER_IAM_ROLE=$(aws iam get-role --role-name $cluster_role_name --
query="Role.Arn" --output text)
```

v. このロールに、必要な Amazon EKS 管理の IAM ポリシーをアタッチします。

```
aws iam attach-role-policy --policy-arn arn:aws:iam::aws:policy/
AmazonEKSClusterPolicy --role-name $cluster_role_name
```

d. クラスターを作成します。

```
aws eks create-cluster --region $region_code --name $cluster_name --kubernetes-
version 1.XX \
   --role-arn $CLUSTER_IAM_ROLE --resources-vpc-config subnetIds=
$subnets,securityGroupIds=$security_groups \
   --kubernetes-network-config ipFamily=ipv6
```

i. 注意: リクエストで指定したアベイラビリティーゾーンのいずれかに、Amazon EKS クラスターの作成に十分な容量がない場合には、エラーが表示されることがあります。このエラー出力には新しいクラスターをサポートできるアベイラビリティーゾーンが表示されます。アカウント向けにサポートされているアベイラビリティーゾーンにある 2 つ以上のサブネットを使用して、クラスターを作成します。詳細については、the section called “容量不足” を参照してください。

クラスターが作成されるまでに数分かかります。以下のコマンドを実行してください。コマンドの出力が「ACTIVE」になるまで、次のステップに進まないでください。

```
aws eks describe-cluster --region $region_code --name $cluster_name --query
 cluster.status
```

e. クラスターの kubeconfig ファイルを作成または更新し、そのクラスターと通信できるようにします。

```
aws eks update-kubeconfig --region $region_code --name $cluster_name
```

デフォルトでは、config ファイルが ~/.kube に作成されるか、config ファイルが既に ~/.kube に存在する場合には、その中に新しいクラスター設定が追加されます。

f. ノードの IAM ロールを作成します。

i. 次のコマンドを実行して、vpc-cni-ipv6-policy.json ファイルを作成します。

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "ec2:AssignIpv6Addresses",
                "ec2:DescribeInstances",
                "ec2:DescribeTags",
                "ec2:DescribeNetworkInterfaces",
                "ec2:DescribeInstanceTypes"
            ],
            "Resource": "*"
        },
        {
            "Effect": "Allow",
```

```
            "Action": [
                "ec2:CreateTags"
            ],
            "Resource": [
                "arn:aws:ec2:*:*:network-interface/*"
            ]
        }
    ]
}
```

ii. IAM ポリシーを作成します。

```
aws iam create-policy --policy-name AmazonEKS_CNI_IPv6_Policy --policy-document
 file://vpc-cni-ipv6-policy.json
```

iii. 次のコマンドを実行して、node-role-trust-relationship.json ファイルを作成します。

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "ec2.amazonaws.com"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
```

iv. 次のコマンドを実行して、ロール名の変数を設定します。*AmazonEKSNodeRole* は、任意の名前に置き換えることができます。

```
export node_role_name=AmazonEKSNodeRole
```

v. IAM ロールを作成します。

```
aws iam create-role --role-name $node_role_name --assume-role-policy-document
 file://"node-role-trust-relationship.json"
```

vi. IAM ロールに IAM ポリシーをアタッチします。

```
aws iam attach-role-policy --policy-arn arn:aws:iam::$account_id:policy/
AmazonEKS_CNI_IPv6_Policy \
    --role-name $node_role_name
```

> **Important**
>
> 簡単にするため、このチュートリアルでは、ポリシーが以下の IAM ロールにアタッチされています。ただし、本番向けクラスターの場合は、異なる IAM ロールにポリシーをアタッチすることをお勧めします。詳細については、the section called “IRSA 用の設定” を参照してください。

vii. IAM ロールに、2 つの必須なマネージド IAM ポリシーをアタッチします。

```
aws iam attach-role-policy --policy-arn arn:aws:iam::aws:policy/
AmazonEKSWorkerNodePolicy \
  --role-name $node_role_name
aws iam attach-role-policy --policy-arn arn:aws:iam::aws:policy/
AmazonEC2ContainerRegistryReadOnly \
  --role-name $node_role_name
```

viii. IAM ロールの ARN を取得し、後のステップで使用するために変数に格納します。

```
node_iam_role=$(aws iam get-role --role-name $node_role_name --query="Role.Arn" --
output text)
```

g. マネージド型ノードグループを作成する

i. 前のステップで作成したサブネットの ID を表示します。

```
echo $subnets
```

出力例は次のとおりです。

```
subnet-0a1a56c486EXAMPLE,subnet-099e6ca77aEXAMPLE,subnet-0377963d69EXAMPLE,subnet-0c05f819
```

ii. ノードグループを作成します。*0a1a56c486EXAMPLE*、*099e6ca77aEXAMPLE*、*0377963d69EXAMPLE*、および *0c05f819d5EXAMPLE* は、前のステップの出力で返された値に置き換えます。次のコマンドで

は、前の出力の中でサブネット ID の間にあるカンマを必ず削除して使用します。*t3.medium* の部分は、任意の AWS Nitro System インスタンスタイプ に置き換えることができます。

```
aws eks create-nodegroup --region $region_code --cluster-name $cluster_name --
nodegroup-name $nodegroup_name \
    --subnets subnet-0a1a56c486EXAMPLE subnet-099e6ca77aEXAMPLE
 subnet-0377963d69EXAMPLE subnet-0c05f819d5EXAMPLE \
    --instance-types t3.medium --node-role $node_iam_role
```

ノードグループの作成には数分かかります。以下のコマンドを実行してください。出力が「ACTIVE」を返すまで、次のステップに進まないでください。

```
aws eks describe-nodegroup --region $region_code --cluster-name $cluster_name --
nodegroup-name $nodegroup_name \
    --query nodegroup.status --output text
```

h. デフォルトの Pod に IPv6 アドレスが割り当てられていることを、IP 列で確認します。

```
kubectl get pods -n kube-system -o wide
```

出力例は次のとおりです。

```
NAME                       READY   STATUS    RESTARTS   AGE     IP
                     NODE                                                NOMINATED NODE
 READINESS GATES
aws-node-rslts             1/1     Running   1          5m36s
 2600:1f13:b66:8200:11a5:ade0:c590:6ac8   ip-192-168-34-75.region-
code.compute.internal   <none>           <none>
aws-node-t74jh             1/1     Running   0          5m32s
 2600:1f13:b66:8203:4516:2080:8ced:1ca9   ip-192-168-253-70.region-
code.compute.internal  <none>           <none>
coredns-85d5b4454c-cw7w2   1/1     Running   0          56m
 2600:1f13:b66:8203:34e5::                ip-192-168-253-70.region-
code.compute.internal  <none>           <none>
coredns-85d5b4454c-tx6n8   1/1     Running   0          56m
 2600:1f13:b66:8203:34e5::1               ip-192-168-253-70.region-
code.compute.internal  <none>           <none>
kube-proxy-btpbk           1/1     Running   0          5m36s
 2600:1f13:b66:8200:11a5:ade0:c590:6ac8   ip-192-168-34-75.region-
code.compute.internal   <none>           <none>
```

```
kube-proxy-jjk2g            1/1     Running   0          5m33s
 2600:1f13:b66:8203:4516:2080:8ced:1ca9   ip-192-168-253-70.region-
code.compute.internal   <none>           <none>
```

i. デフォルトのサービスに IPv6 アドレスが割り当てられていることを、IP 列で確認します。

```
kubectl get services -n kube-system -o wide
```

出力例は次のとおりです。

```
NAME       TYPE        CLUSTER-IP          EXTERNAL-IP   PORT(S)         AGE
 SELECTOR
kube-dns   ClusterIP   fd30:3087:b6c2::a   <none>        53/UDP,53/TCP   57m   k8s-
app=kube-dns
```

j. (オプション) サンプルアプリケーションをデプロイするか、AWS ロードバランサーコントローラーとサンプルアプリケーションをデプロイして、「the section called “アプリケーション負荷分散”」に従い HTTP アプリケーションをロードバランシングするか、「the section called “ネットワーク負荷分散”」に従いネットワークトラフィックを IPv6 Pod にロードバランシングします。

k. このチュートリアルのために作成したクラスターとノードの使用が終了したら、以下のコマンドにより、作成したリソースのクリーンアップを行う必要があります。削除を行う前に、これらのリソースが、このチュートリアルの外部で使用されていないことを確認します。

i. ここまでの手順を実行したのとは異なるシェルで、このステップを完了する場合は、サンプル値を前の手順を実行した際に指定した値に置き換えながら、すべての変数を前の手順で使用した値に設定します。前の手順を実行したのと同じシェルでこのステップを完了する場合は、このまま次のステップに進みます。

```
export region_code=region-code
export vpc_stack_name=my-eks-ipv6-vpc
export cluster_name=my-cluster
export nodegroup_name=my-nodegroup
export account_id=111122223333
export node_role_name=AmazonEKSNodeRole
export cluster_role_name=myAmazonEKSClusterRole
```

ii. ノードグループを削除します。

```
aws eks delete-nodegroup --region $region_code --cluster-name $cluster_name --
nodegroup-name $nodegroup_name
```

削除には数分かかります。以下のコマンドを実行してください。何らかの出力が返された場合は、次のステップに進まないでください。

```
aws eks list-nodegroups --region $region_code --cluster-name $cluster_name --query
 nodegroups --output text
```

iii. クラスターを削除します。

```
aws eks delete-cluster --region $region_code --name $cluster_name
```

クラスターの削除には数分かかります。次に進む前に、下記のコマンドを使用してクラスターが削除されていることを確認します。

```
aws eks describe-cluster --region $region_code --name $cluster_name
```

下記のような出力が返されるまで、次のステップに進まないでください。

```
An error occurred (ResourceNotFoundException) when calling the DescribeCluster
 operation: No cluster found for name: my-cluster.
```

iv. 作成した IAM リソースを削除します。前の手順で使用した名前とは異なる名前を選択した場合は、その名で *AmazonEKS_CNI_IPv6_Policy* を置き換えます。

```
aws iam detach-role-policy --role-name $cluster_role_name --policy-arn
 arn:aws:iam::aws:policy/AmazonEKSClusterPolicy
aws iam detach-role-policy --role-name $node_role_name --policy-arn
 arn:aws:iam::aws:policy/AmazonEKSWorkerNodePolicy
aws iam detach-role-policy --role-name $node_role_name --policy-arn
 arn:aws:iam::aws:policy/AmazonEC2ContainerRegistryReadOnly
aws iam detach-role-policy --role-name $node_role_name --policy-arn arn:aws:iam::
$account_id:policy/AmazonEKS_CNI_IPv6_Policy
aws iam delete-policy --policy-arn arn:aws:iam::$account_id:policy/
AmazonEKS_CNI_IPv6_Policy
aws iam delete-role --role-name $cluster_role_name
aws iam delete-role --role-name $node_role_name
```

v. VPC を作成した AWS CloudFormation スタックを削除します。

```
aws cloudformation delete-stack --region $region_code --stack-name $vpc_stack_name
```

Pod のアウトバウンドインターネットアクセスを有効にする

適用対象: Linux IPv4 Fargate ノード、Amazon EC2 インスタンスを持つ Linux ノード

IPv6 ファミリーを使用してクラスターをデプロイした場合、このトピックの情報はクラスターに適用されません。IPv6 アドレスがネットワーク変換されないからです。クラスターでの IPv6 使用の詳細については、「the section called “IPv6”」を参照してください。

デフォルトでは、クラスター内の各 Pod に、Pod がデプロイされた VPC に関連付けられた Classless Inter-Domain Routing (CIDR) ブロックから、プライベート IPv4 アドレスが割り当てられます。同じ VPC 内の Pod は、これらのプライベート IP アドレスをエンドポイントとして使用して相互に通信します。VPC に関連付けられている CIDR ブロック外の IPv4 アドレスと Pod が通信する場合、(Linux と Windows 両用の) Amazon VPC CNI プラグインが Pod の IPv4 アドレスを、Pod が実行されているノードのプライマリ Elastic Network Interface のプライマリプライベート IPv4 アドレスに変換します (デフォルトでは *)。

> **Note**
>
> Windows ノードについては、他にも考慮すべき事項があります。デフォルトでは、Windows の VPC CNI プラグインは、同じ VPC 内の宛先へのトラフィックを SNAT から除外するネットワーク設定で定義されます。つまり、内部 VPC 通信では SNAT が無効になっており、Pod に割り当てられた IP アドレスは VPC 内でルーティング可能です。ただし、VPC 外の宛先へのトラフィックでは、送信元 Pod IP がインスタンスの ENI のプライマリ IP アドレスに SNAT されます。Windows のこのデフォルト設定により、Pod はホストインスタンスと同じ方法で VPC 外部のネットワークにアクセスできます。

この動作によって、次の現象が起こります。

- Pod は、実行中のノードにパブリック IP アドレスまたは Elastic IP アドレスがあり、そのノードがルートテーブルにインターネットゲートウェイへのルートがあるパブリックサブネットにある場合、またはルートテーブルに NAT ゲートウェイへのルートがあるプライベートサブネットにある場合、インターネットリソースと通信できます。可能であれば、プライベートサブネットにノードをデプロイすることをお勧めします。
- 1.8.0 よりも古いバージョンのプラグインの場合、VPC ピアリング、トランジット VPC、または AWS Direct Connect を使用してクラスター VPC に接続されているネットワークまたは VPC 内にあるリソースは、セカンダリ Elastic Network Interface の背後にある Pod との通信を開始できません。ただし、Pod はこれらのリソースとの通信を開始し、リソースから応答を受け取ることができます。

ご使用の環境で次のいずれかが当てはまる場合は、次のコマンドを使用してデフォルト設定を変更してください。

- VPC ピアリング、トランジット VPC、または AWS Direct Connect を使用してクラスター VPC に接続されているネットワークまたは VPC 内に、IPv4 アドレスを使用して Pod との通信を開始する必要があるリソースがあり、プラグインのバージョンが 1.8.0 よりも前である。
- トラフィックがノードから離れるときに Pod の送信元 IP アドレスが保持され、ノードの代わりに NAT ゲートウェイ によって SNAT が実行されるようにします。例えば、ネットワークアプライアンスやログが Pod の IP アドレスを観察できるようにするためです。Pod が送信元のトラフィックはインターネットゲートウェイ経由でルーティングできないため、ノードは NAT ゲートウェイへのルートを持つプライベートサブネットにある必要があります。

```
kubectl set env daemonset -n kube-system aws-node AWS_VPC_K8S_CNI_EXTERNALSNAT=true
```

> **Note**
>
> AWS_VPC_K8S_CNI_EXTERNALSNAT および AWS_VPC_K8S_CNI_EXCLUDE_SNAT_CIDRS CNI の設定変数は Windows ノードに適用されません。SNAT の無効化は Windows でサポートされていません。IPv4 CIDR のリストを SNAT から除外する場合は、Windows ブートストラップスクリプトで ExcludedSnatCIDRs パラメータを指定して定義できます。このパラメータの使用に関する詳細については、「the section called “ブートストラップスクリプトの設定パラメータ”」を参照してください。

ホストネットワーキング

* Pod の仕様に hostNetwork=true が含まれている場合 (デフォルトは false)、その IP アドレスは別のアドレスに変換されません。これは、デフォルトでクラスターで実行される kube-proxy および Amazon VPC CNI plugin for Kubernetes Pod の場合に当てはまります。これらの Pod の場合、IP アドレスはノードのプライマリ IP アドレスと同じであるため、Pod の IP アドレスは変換されません。Pod の hostNetwork 設定の詳細については、Kubernetes API リファレンスの「PodSpec v1 core」を参照してください。

Kubernetes ネットワークポリシーにより Pod トラフィックを制限する

概要

デフォルトでは、Kubernetes の IP アドレス、ポート、クラスター内の Pod 間の接続、またはポッドと他のネットワークのリソースの接続に制限はありません。Kubernetes ネットワークポリシーを使用して、自分の Pod 間で送受信されるネットワークトラフィックを制限できます。詳細については、Kubernetes ドキュメントの「ネットワークポリシー」を参照してください。

標準のネットワークポリシー

標準の NetworkPolicy を使用すると、クラスター内のポッド間トラフィックをセグメント化できます。OSI ネットワークモデルのレイヤー 3 と 4 で動作するネットワークポリシーであり、Amazon EKS クラスター内の IP アドレスまたはポートレベルでトラフィックフローを制御できます。標準のネットワークポリシーは、範囲が名前空間レベルに限定されます。

ユースケース

- ワークロード間でネットワークトラフィックをセグメント化すると、関連するアプリケーションのみが相互に通信できるようになります。
- ポリシーを使用して名前空間レベルでテナントを分離すると、ネットワークを分離できます。

例

以下のポリシーでは、sun 名前空間の webapp ポッドからのエグレストラフィックが制限されています。

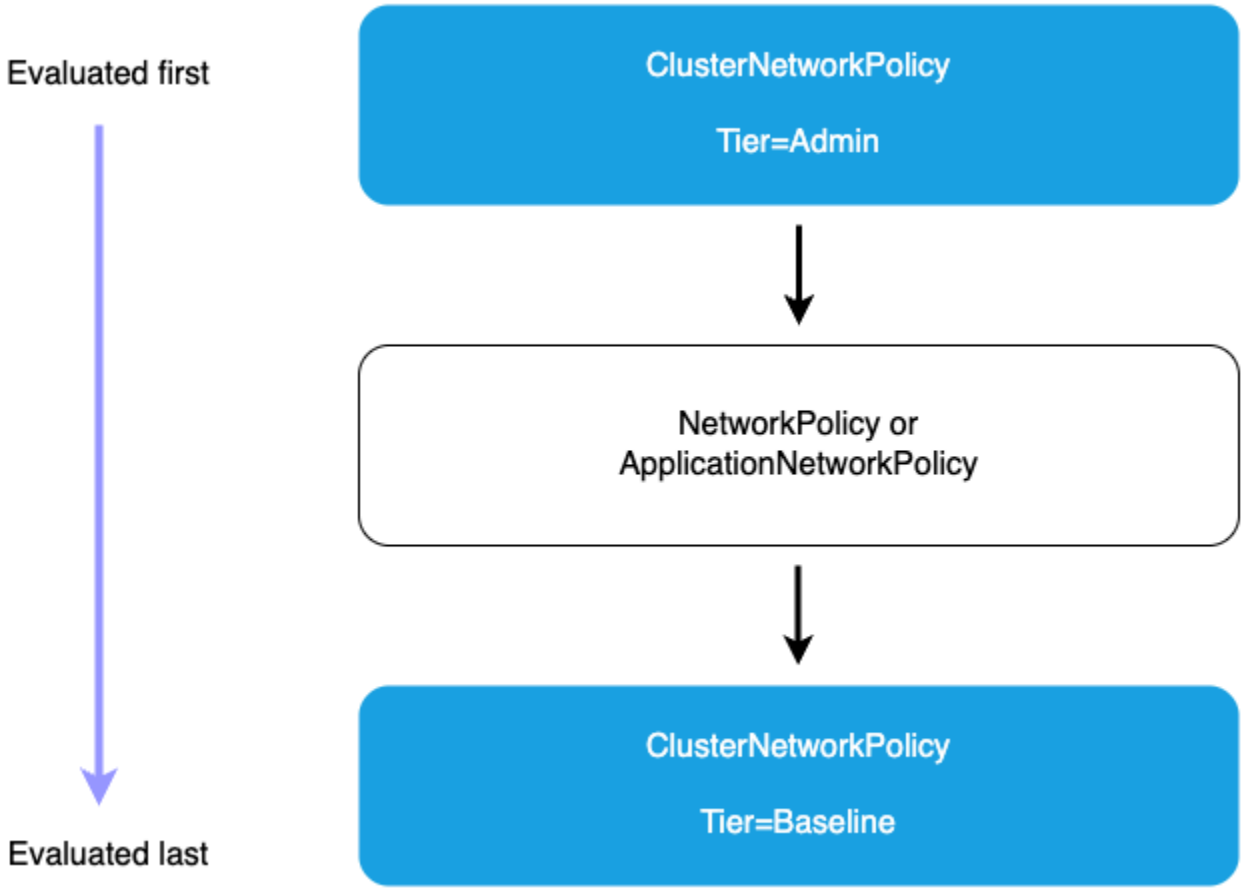
```
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: webapp-egress-policy
  namespace: sun
spec:
  podSelector:
    matchLabels:
      role: webapp
  policyTypes:
  - Egress
  egress:
  - to:
```

```
    - namespaceSelector:
        matchLabels:
          name: moon
      podSelector:
        matchLabels:
          role: frontend
    ports:
    - protocol: TCP
      port: 8080
  - to:
    - namespaceSelector:
        matchLabels:
          name: stars
      podSelector:
        matchLabels:
          role: frontend
    ports:
    - protocol: TCP
      port: 8080
```

ポリシーは、sun 名前空間で role: webapp というラベルが付いたポッドに適用されます。

- 許可されるトラフィック: TCP ポート 8080 の moon 名前空間で role: frontend というラベルが付いたポッド。
- 許可されるトラフィック: TCP ポート 8080 の stars 名前空間で role: frontend というラベルが付いたポッド。
- ブロックされるトラフィック: webapp ポッドからの他のすべてのアウトバウンドトラフィックは暗黙的に拒否されます。

管理者 (あるいはクラスター) ネットワークポリシー



`ClusterNetworkPolicy` を使用すると、ネットワークセキュリティ標準をクラスター全体に適用できます。名前空間ごとにポリシーを繰り返し定義して維持するのではなく、単一のポリシーを使用して、名前空間に関係なく、クラスター内のさまざまなワークロードに対するネットワークアクセスコントロールを一元管理できます。

ユースケース

- EKS クラスター内のすべての (または一部の) ワークロードに対するネットワークアクセスコントロールを一元管理します。
- クラスター全体にわたるデフォルトのネットワークセキュリティ体制を定義します。
- 組織のセキュリティ標準を運用効率の高い方法でクラスターの範囲へと拡張します。

例

以下のポリシーでは、他の名前空間からのクラスタートラフィックを明示的にブロックして、機密性の高いワークロード名前空間へのネットワークアクセスを防ぐことができます。

```
apiVersion: networking.k8s.aws/v1alpha1
kind: ClusterNetworkPolicy
metadata:
  name: protect-sensitive-workload
spec:
  tier: Admin
  priority: 10
  subject:
    namespaces:
      matchLabels:
        kubernetes.io/metadata.name: earth
  ingress:
    - action: Deny
      from:
      - namespaces:
          matchLabels: {} # Match all namespaces.
      name: select-all-deny-all
```

重要な注意事項

Amazon VPC CNI plugin for Kubernetes のネットワークポリシーは、以下に示した構成でサポートされています。

- 標準と管理者のどちらのネットワークポリシーでも、Amazon VPC CNI プラグインのバージョンが 1.21.0 (またはそれ以降) です。
- IPv4 または IPv6 アドレス用に設定されたクラスター。
- ポッド用のセキュリティグループでネットワークポリシーを使用できます。ネットワークポリシーを使用すると、クラスター内の通信をすべて制御できます。Pod 用のセキュリティグループを使用すると、Pod 内のアプリケーションから AWS サービスへのアクセスを制御できます。
- カスタムネットワークおよびプレフィックス委任でネットワークポリシーを使用できます。

考慮事項

アーキテクチャ

- Amazon VPC CNI plugin for Kubernetes を含むクラスターに Amazon VPC CNI plugin for Kubernetes ネットワークポリシーを適用する場合、Amazon EC2 Linux ノードにのみポリシーを適用できます。Fargate または Windows ノードにはポリシーを適用できません。
- ネットワークポリシーは IPv4 または IPv6 アドレスのいずれかのみを適用しますが、両方は適用しません。IPv4 クラスターでは VPC CNI は IPv4 アドレスをポッドに割り当て、IPv4 ポリシーを適用します。IPv6 クラスターでは VPC CNI は IPv6 アドレスをポッドに割り当て、IPv6 ポリシーを適用します。IPv6 クラスターに適用された IPv4 ネットワークポリシールールは無視されます。IPv4 クラスターに適用された IPv6 ネットワークポリシールールは無視されます。

ネットワークポリシー

- Amazon EKS は、metadata.ownerReferences フィールドが設定されている Pod のネットワークポリシーの適用を最適化します。これには、Deployments、StatefulSets、DaemonSets、Jobs、CronJobs などのコントローラーによって管理される Pod が含まれます。コントローラーなしで直接作成されたスタンドアロン Pod には metadata.ownerReferences が設定されておらず、これらの Pod ではネットワークポリシーの適用が確実には機能しない場合があります。
- 同じ Pod に複数のネットワークポリシーを適用できます。同じ Pod を選択するポリシーが 2 つ以上設定されている場合、すべてのポリシーが Pod に適用されます。
- ある IP アドレス範囲 (CIDR) に許可されるポートとプロトコルの組み合わせの最大数は、ネットワークポリシー全体で 24 です。namespaceSelector などのセレクターは、1 つ以上の CIDR に解決されます。複数のセレクターが 1 つの CIDR に解決される場合や、同一または異なるネットワークポリシーに同じ直接 CIDR を複数回指定した場合に、この制限が考慮されます。
- どの Kubernetes サービスでも、サービスポートはコンテナポートと同じでなければなりません。名前付きポートを使用している場合はサービス仕様でも同じ名前を使用してください。

管理者ネットワークポリシー

1. 管理者層ポリシー (最初に評価): 管理者階層のすべての ClusterNetworkPolicy が他のポリシーよりも先に評価されます。管理者階層内では、優先順位に従って (優先順位番号の最も小さいものから) ポリシーが処理されます。次にどうなるかは、アクションタイプによって決まります。
   - 拒否アクション (最も高い優先順位): 拒否アクションを定義した管理者ポリシーがトラフィックと一致すると、そのトラフィックは他のポリシーとは関係なくすぐにブロックされます。ClusterNetworkPolicy や NetworkPolicy ルールはそれ以上処理されません。これにより、組

織全体のセキュリティコントロールを名前空間レベルのポリシーで上書きできないようにしています。

- 許可アクション: 拒否ルールが評価されると、許可アクションを定義した管理者ポリシーが優先順位に従って (優先順位番号の最も小さいものから) 処理されます。許可アクションに一致すると、トラフィックは受け入れられ、ポリシーはそれ以上評価されません。これらのポリシーは、ラベルセレクターに基づいて複数の名前空間に対するアクセスを許可して、特定のリソースにどのワークロードがアクセスできるかを一元的に制御できます。
- パスアクション: 管理階層ポリシーにパスアクションを定義すると、意思決定が下位の階層に委任されます。トラフィックがパスルールに一致すると、そのトラフィックの残りの管理者階層ルールの評価がスキップされ、NetworkPolicy 階層に直接進みます。これにより、管理者は特定のトラフィックパターンに対する制御をアプリケーションチームに明示的に委任できます。例えば、パスルールを使用すると、外部アクセスを厳密に制御しつつ、名前空間内のトラフィック管理を名前空間管理者に委任できます。

2. ネットワークポリシー階層: 拒否または許可に一致する管理者階層ポリシーがない場合や、パスアクションが一致した場合、次は名前空間範囲の NetworkPolicy リソースが評価されます。これらは個々の名前空間内できめ細かく制御できるポリシーで、アプリケーションチームによって管理されます。名前空間範囲のポリシーでは、管理者ポリシーよりも厳しく制限を課すことができます。管理者ポリシーによる拒否の判断を上書きすることはできませんが、管理者ポリシーによって許可またはパスされたトラフィックをさらに制限できます。
3. ベースライン階層管理ポリシー: トラフィックに一致する管理者ポリシーや名前空間範囲のポリシーがない場合、ベースライン階層の ClusterNetworkPolicy が評価されます。これにより、デフォルトのセキュリティ体制を名前空間範囲のポリシーで上書きできるため、管理者が組織全体のデフォルトを設定する一方で、チームが必要に応じて柔軟にカスタマイズできます。ベースラインポリシーは、優先順位に従って (優先順位番号の最も小さいものから) 評価されます。
4. デフォルト拒否 (一致するポリシーがない場合): この deny-by-default 動作により、明示的に認められた接続のみが許可されるため、強力なセキュリティ体制を維持できます。

移行

- クラスターが現在サードパーティーソリューションを使用して Kubernetes ネットワークポリシーを管理している場合、同じポリシーを Amazon VPC CNI plugin for Kubernetes で使用できます。ただし、同じポリシーを管理しないように、既存のソリューションを削除する必要があります。

> **Warning**
>
> ネットワークポリシーソリューションを削除したら、そのソリューションが適用されていたすべてのノードを置き換えることをお勧めします。ソリューションのポッドが突然終了した場合に、トラフィックルールが残ってしまう可能性があるためです。

インストール

- ネットワークポリシー機能では`policyendpoints.networking.k8s.aws` と呼ばれる `PolicyEndpoint` カスタムリソース定義 (CRD が作成され、必要になります。カスタムリソースの `PolicyEndpoint` オブジェクトは Amazon EKS によって管理されます。これらのリソースを変更または削除しないでください。
- インスタンスロールの IAM 認証情報を使用するポッドを実行するか、EC2 IMDS に接続するポッドを実行する場合はEC2 IMDS へのアクセスをブロックするネットワークポリシーがないか慎重に確認してください。EC2 IMDS へのアクセスを許可するネットワークポリシーを追加する必要がある場合があります。詳細については「Amazon EC2 ユーザーガイド」の「インスタンスメタデータとユーザーデータ」を参照してください。

  サービスアカウントの IAM ロールまたは EKS Pod Identity を使用するポッドはEC2 IMDS にアクセスしません。
- Amazon VPC CNI plugin for Kubernetes は、各ポッドの追加のネットワークインターフェイスにはネットワークポリシーを適用せず、各ポッドのプライマリインターフェイス (`eth0`) のみにネットワークポリシーを適用します。これは以下のアーキテクチャに影響します：
  - `ENABLE_V4_EGRESS` 変数が `true` に設定された `IPv6` ポッド。この変数により、`IPv4` エグレス機能が IPv6 ポッドをクラスター外のエンドポイントなどの `IPv4` エンドポイントに接続できるようになります。`IPv4` エグレス機能は、ローカルループバック IPv4 アドレスを持つ追加のネットワークインターフェースを作成することで機能します。
  - Multus などのチェーンネットワークプラグインを使用する場合。これらのプラグインは各ポッドにネットワークインターフェースを追加するため、ネットワークポリシーはチェーンネットワークプラグインには適用されません。

Kubernetes ネットワークポリシーを使用して Pod ネットワークトラフィックを制限する

Kubernetes ネットワークポリシーを使用して、自分の Pod 間で送受信されるネットワークトラフィックを制限できます。詳細については、Kubernetes ドキュメントの「ネットワークポリシー」を参照してください。

この機能を使用するには以下を設定する必要があります：

1. Pod 起動時のポリシーの適用を設定します。これは VPC CNI `DaemonSet` の `aws-node` コンテナでできます。
2. アドオンのネットワークポリシーパラメータを有効にします。
3. Kubernetes ネットワークポリシーを使用するようにクラスターを設定します。

開始する前に、考慮事項を確認してください。詳細については「the section called “考慮事項”」を参照してください。

前提条件

この機能の前提条件は次のとおりです。

最小クラスターバージョン

既存の Amazon EKS クラスター。デプロイするには、はじめに を参照してください。クラスターは、次の表に示す Kubernetes バージョンとプラットフォームバージョンのいずれかを実行している必要があります。一覧にあるバージョンより後の Kubernetes とプラットフォームのバージョンもサポートされることにご注意ください。現在の Kubernetes バージョンを確認するには、次のコマンドの *my-cluster* をクラスターの名前に置き換えて、変更したコマンドを実行します。

```
aws eks describe-cluster --name my-cluster --query cluster.version --output text
```

Kubernetes バージョン	プラットフォームバージョン
1.27.4	eks.5
1.26.7	eks.6

最小 VPC CNI バージョン

標準 Kubernetes ネットワークポリシーと管理者ネットワークポリシーの両方を作成するには、VPC CNI プラグインのバージョン 1.21 を実行する必要があります。現在のバージョンは、次のコマンドで確認できます。

```
kubectl describe daemonset aws-node --namespace kube-system | grep amazon-k8s-cni: |
 cut -d : -f 3
```

バージョンが 1.21 より前の場合は、the section called “更新 (EKS アドオン)” を確認してバージョン 1.21 以降にアップグレードしてください。

最小 Linux カーネルバージョン

ノードにはバージョン 5.10 以降の Linux カーネルが必要です。カーネルのバージョンはuname -r で確認できます。Amazon EKS 最適化 Amazon Linux、Amazon EKS 最適化高速 Amazon Linux AMI、と Bottlerocket AMI の最新バージョンを使用している場合、既に必要なカーネルバージョンがインストールされています。

Amazon EKS 最適化高速 Amazon Linux AMI v20231116 バージョン以降には、カーネルバージョン 5.10 があります。

ステップ 1: Pod 起動時のポリシーの適用を設定する

Amazon VPC CNI plugin for Kubernetes はポッドのプロビジョニングと並行して、ポッドのネットワークポリシーを設定します。新しいポッドにすべてのポリシーが設定されるまで、新しいポッドのコンテナはデフォルトの許可ポリシーで起動します。これは標準モードと呼ばれます。デフォルトの許可ポリシーはすべての ingress トラフィックと egress トラフィックが新しいポッドとの間で許可されることを意味します。例えば、新しいポッドがアクティブなポリシーで更新されるまで、ポッドにはファイアウォールルールが適用されません (すべてのトラフィックが許可されます)。

変数 NETWORK_POLICY_ENFORCING_MODE を strict に設定すると、VPC CNI を使用するポッドはデフォルトの拒否ポリシーで起動し、その後ポリシーが設定されます。これは Strict モードと呼ばれます。Strict モードではポッドがクラスター内でアクセスする必要があるすべてのエンドポイントにネットワークポリシーが必要です。この要件は CoreDNS ポッドに適用されることに注意してください。デフォルトの拒否ポリシーはホストネットワークを使用するポッドには設定されていません。

このデフォルトのネットワークポリシーはVPC CNI DaemonSet の aws-node コンテナで環境変数 NETWORK_POLICY_ENFORCING_MODE を strict に設定することで変更できます。

```
env:
  - name: NETWORK_POLICY_ENFORCING_MODE
    value: "strict"
```

ステップ 2: アドオンのネットワークポリシーパラメータを有効にする

ネットワークポリシー機能はデフォルトでノード上のポート 8162 をメトリクスに使用します。また、この機能はヘルスプローブにポート 8163 を使用します。そのノード上、またはこれらのポート

を使用する必要があるポッド内で他のアプリケーションを実行すると、そのアプリは実行できません。VPC CNI バージョン v1.14.1 以降で、これらのポートを変更できます。

アドオンのネットワークポリシーパラメータを有効にするには次の手順に従います。

AWS マネジメントコンソール

1. Amazon EKS コンソールを開きます。
2. 左のナビゲーションペインで、[クラスター] を選択し、 Amazon VPC CNI アドオンを設定するクラスターの名前を選択してください。
3. [アドオン] タブを選択してください。
4. アドオンボックスの右上にあるボックスを選択し、次に [編集] を選択してください。
5. [**Amazon VPC CNI** を設定] ページで、次の操作を行います：
   a. [バージョン] リストで v1.14.0-eksbuild.3 以降のバージョンを選択してください。
   b. [オプションの構成設定] を展開します。
   c. [設定値] に JSON キー "enableNetworkPolicy": と値 "true" を入力します。結果のテキストは有効な JSON オブジェクトでなければなりません。このキーと値だけがテキストボックス内のデータである場合はキーと値を中括弧 { } で囲みます。

      次の例ではネットワークポリシー機能が有効になっており、メトリクスとヘルスプローブがデフォルトのポート番号に設定されています：

```
{
    "enableNetworkPolicy": "true",
    "nodeAgent": {
        "healthProbeBindAddr": "8163",
        "metricsBindAddr": "8162"
    }
}
```

Helm

helm を通して Amazon VPC CNI plugin for Kubernetes をインストールしている場合、設定を更新してポートを変更できます。

1. 次に、以下のコマンドを実行して、ポートを変更します。キー nodeAgent.metricsBindAddr または キー nodeAgent.healthProbeBindAddr の値にそれぞれポート番号を設定します。

```
helm upgrade --set nodeAgent.metricsBindAddr=8162 --set
 nodeAgent.healthProbeBindAddr=8163 aws-vpc-cni --namespace kube-system eks/aws-vpc-
cni
```

kubectl

1. エディターで aws-nodeDaemonSet を開きます。

```
kubectl edit daemonset -n kube-system aws-node
```

2. VPC CNI aws-node デーモンセットマニフェストの aws-network-policy-agent コンテナで、以下の args: のコマンド引数でポート番号を置き換えます。

```
    - args:
            - --metrics-bind-addr=:8162
            - --health-probe-bind-addr=:8163
```

ステップ 3: Kubernetes ネットワークポリシーを使用するようにクラスターを設定する

これはAmazon EKS アドオンまたはセルフマネージドアドオンに対して設定できます。

Amazon EKS アドオン

AWS CLI を使用すると、次のコマンドを実行して、Kubernetes ネットワークポリシーを使用するようにクラスターを設定できます。my-cluster をクラスターの名前に置き換え、IAM ロール ARN を使用するロールに置き換えます。

```
aws eks update-addon --cluster-name my-cluster --addon-name vpc-cni --addon-version
 v1.14.0-eksbuild.3 \
    --service-account-role-arn arn:aws:iam::123456789012:role/AmazonEKSVPCCNIRole \
    --resolve-conflicts PRESERVE --configuration-values '{"enableNetworkPolicy":
 "true"}'
```

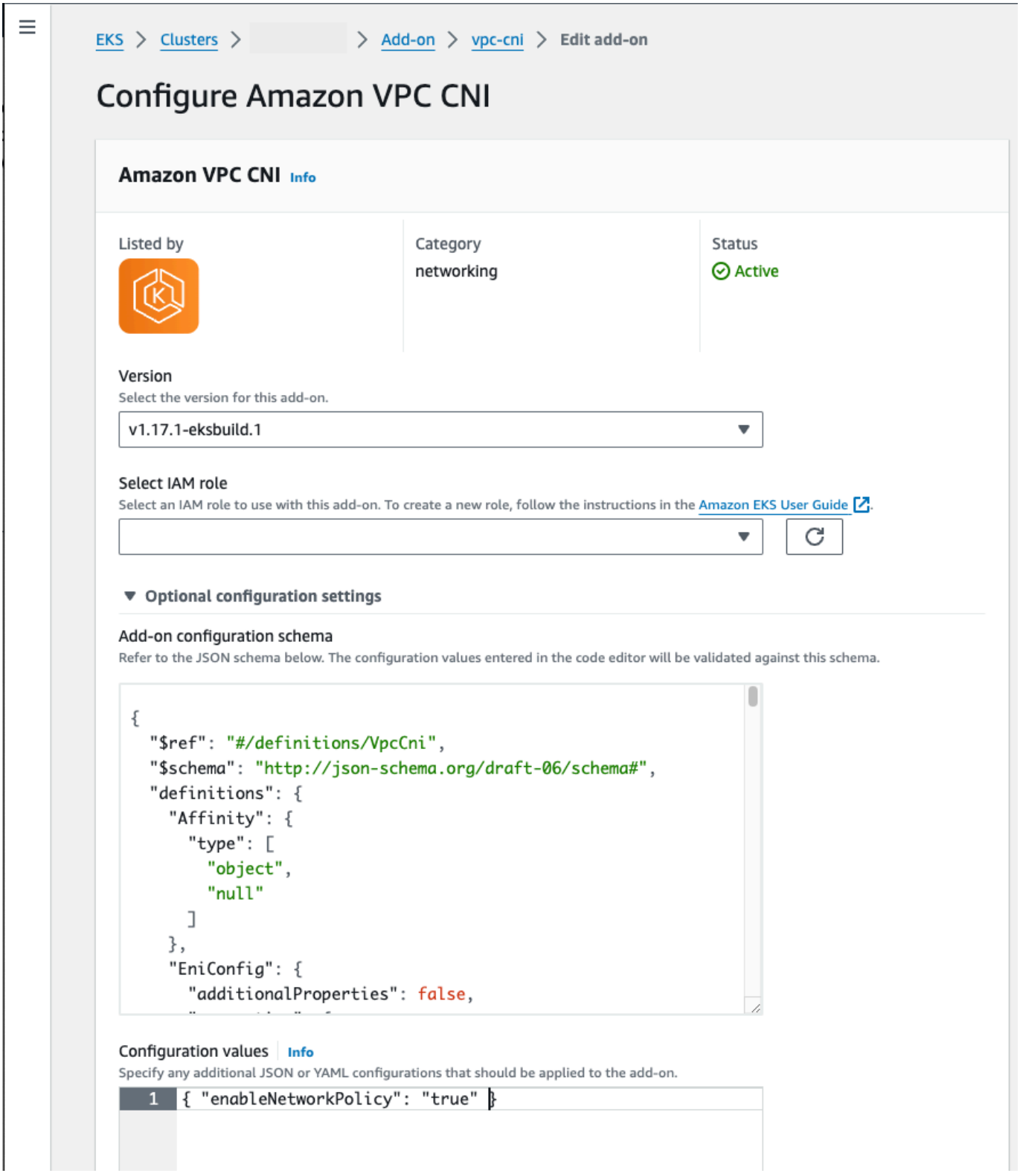
AWS マネジメントコンソールを使用してこれを設定するには、以下の手順に従います。

1. Amazon EKS コンソールを開きます。
2. 左のナビゲーションペインで、[クラスター] を選択し、 Amazon VPC CNI アドオンを設定するクラスターの名前を選択してください。

3. [アドオン] タブを選択してください。
4. アドオンボックスの右上にあるボックスを選択し、次に [編集] を選択してください。
5. [**Amazon VPC CNI** を設定] ページで、次の操作を行います：
   a. [バージョン] リストで v1.14.0-eksbuild.3 以降のバージョンを選択してください。
   b. [オプションの構成設定] を展開します。
   c. [設定値] に JSON キー "enableNetworkPolicy": と値 "true" を入力します。結果のテキストは有効な JSON オブジェクトでなければなりません。このキーと値だけがテキストボックス内のデータである場合はキーと値を中括弧 { } で囲みます。次の例はネットワークポリシーが有効になっていることを示しています：

      ```
      { "enableNetworkPolicy": "true" }
      ```

      次のスクリーンショットはこのシナリオの例を示しています。


EKS > Clusters > > Add-on > vpc-cni > Edit add-on
Configure Amazon VPC CNI
Amazon VPC CNI Info
Listed by
Category
networking
Status
Active
Version
Select the version for this add-on.
v1.17.1-eksbuild.1
Select IAM role
Select an IAM role to use with this add-on. To create a new role, follow the instructions in the Amazon EKS User Guide.
Optional configuration settings
Add-on configuration schema
Refer to the JSON schema below. The configuration values entered in the code editor will be validated against this schema.
{
"$ref": "#/definitions/VpcCni",
"$schema": "http://json-schema.org/draft-06/schema#",
"definitions": {
"Affinity": {
"type": [
"object",
"null"
]
},
"EniConfig": {
"additionalProperties": false,
Configuration values Info
Specify any additional JSON or YAML configurations that should be applied to the add-on.
1 { "enableNetworkPolicy": "true" }

セルフマネージド型アドオン

Helm

helm を通して Amazon VPC CNI plugin for Kubernetes をインストールしている場合、設定を更新してネットワークポリシーを有効にできます。

1. 次のコマンドを実行してネットワークポリシーを有効にします。

```
helm upgrade --set enableNetworkPolicy=true aws-vpc-cni --namespace kube-system eks/
aws-vpc-cni
```

kubectl

1. エディターで amazon-vpc-cniConfigMap を開きます。

```
kubectl edit configmap -n kube-system amazon-vpc-cni -o yaml
```

2. 次の行を ConfigMap のdataに追加します。

```
enable-network-policy-controller: "true"
```

   行を追加すると、ConfigMap は次の例のようになるはずです。

```
apiVersion: v1
 kind: ConfigMap
 metadata:
  name: amazon-vpc-cni
  namespace: kube-system
 data:
  enable-network-policy-controller: "true"
```

3. エディターで aws-nodeDaemonSet を開きます。

```
kubectl edit daemonset -n kube-system aws-node
```

   a. VPC CNI aws-node デーモンセットマニフェストの aws-network-policy-agent コンテナで、args: のコマンド引数 --enable-network-policy=false の false を true に置き換えます。

```
    - args:
       - --enable-network-policy=true
```

ステップ 4. 次のステップ

設定が完了したら、aws-node ポッドがクラスターで実行されていることを確認します。

```
kubectl get pods -n kube-system | grep 'aws-node\|amazon'
```

出力例は次のとおりです。

```
aws-node-gmqp7                                          2/2     Running   1 (24h ago)
 24h
aws-node-prnsh                                          2/2     Running   1 (24h ago)
 24h
```

バージョン 1.14 以降の aws-node ポッドには 2 つのコンテナがあります。以前のバージョンではネットワークポリシーが無効になっている場合、aws-node ポッドにコンテナは 1 つしか存在しません。

これで、Kubernetes ネットワークポリシーをクラスターにデプロイできます。

Kubernetes ネットワークポリシーを実装するために、Kubernetes の NetworkPolicy オブジェクトまたは ClusterNetworkPolicy オブジェクトを作成してクラスターにデプロイできます。NetworkPolicy オブジェクトの範囲は名前空間に限定されますが、ClusterNetworkPolicy オブジェクトの範囲はクラスター全体または複数の名前空間に限定できます。ラベルセレクター、名前空間、および IP アドレス範囲に基づいて、Pod 間のトラフィックを許可または拒否するポリシーを実装します。NetworkPolicy オブジェクトの作成の詳細については Kubernetes ドキュメントの「ネットワークポリシー」を参照してください。

Kubernetes NetworkPolicy オブジェクトの適用は、Extended Berkeley Packet Filter (eBPF) を使用して実装されます。iptables をベースにした実装に関連して、CPU 使用率の低下やシーケンシャルルックアップの回避など、低レイテンシーを実現し、パフォーマンス特性を発揮します。さらに、eBPF プローブは、複雑なカーネルレベルの問題のデバッグやオブザーバビリティの向上に役立つ、コンテキスト豊富なデータへのアクセスを提供します。Amazon EKS は、プローブを利用して各ノードのポリシー結果をログに記録し、そのデータを外部のログコレクターにエクスポートしてデバッグに役立てる eBPF ベースのエクスポーターをサポートしています。詳細については「eBPF ドキュメント」を参照してください。

Amazon EKS Pod ネットワークトラフィックの Kubernetes ネットワークポリシーを無効にする

Kubernetes ネットワークポリシーを無効にして Amazon EKS ポッドネットワークトラフィックの制限を停止する

1. すべての Kubernetes ネットワークポリシーを一覧表示します。

```
kubectl get netpol -A
```

2. 各 Kubernetes ネットワークポリシーを削除します。ネットワークポリシーを無効にする前に、すべてのネットワークポリシーを削除する必要があります。

```
kubectl delete netpol <policy-name>
```

3. エディタで aws-node DaemonSet を開きます。

```
kubectl edit daemonset -n kube-system aws-node
```

4. VPC CNI `aws-node` デーモンセットマニフェストの `aws-network-policy-agent` コンテナで、`args:` のコマンド引数 `--enable-network-policy` の `true` を `false` に置き換えます。

```
    - args:
       - --enable-network-policy=false
```

Amazon EKS の Kubernetes ネットワークポリシーのトラブルシューティング

これは、Amazon VPC CNI のネットワークポリシー機能のトラブルシューティングガイドです。

このガイドでは、以下について説明します。

- インストール情報、CRD、RBAC アクセス許可 the section called “新しい `policyendpoints` CRD とアクセス許可”
- ネットワークポリシーの問題の診断時に調べるログ the section called “ネットワークポリシーのログ”
- トラブルシューティング用の eBPF SDK ツール群の実行
- 既知の問題と解決策 the section called “既知の問題と解決策”

> **Note**
>
> ネットワークポリシーは、Kubernetes デプロイメントによって作成されたポッドにのみ適用されることに注意してください。VPC CNI のネットワークポリシーにおける他の制限事項については、「the section called “考慮事項”」を参照してください。

ネットワークポリシーのログを読み、eBPF SDK のツールを実行することにより、ネットワークポリシーを使用するネットワーク接続をトラブルシューティングおよび調査できます。

新しい **policyendpoints** CRD とアクセス許可

- CRD: `policyendpoints.networking.k8s.aws`
- Kubernetes API: `v1.networking.k8s.io` と呼ばれる `apiservice`
- Kubernetes リソース: `Kind: NetworkPolicy`
- RBAC: `aws-node` と呼ばれる `ClusterRole` (VPC CNI)、`eks:network-policy-controller` と呼ばれる `ClusterRole` (EKS クラスターコントロールプレーンのネットワークポリシーコントローラー)

ネットワークポリシーの場合、VPC CNI は `policyendpoints.networking.k8s.aws` と呼ばれる新しい `CustomResourceDefinition` (CRD) を作成します。VPC CNI には CRD を作成し、この CRD および VPC CNI (`eniconfigs.crd.k8s.amazonaws.com`) によってインストールされたその他の CRD の CustomResources (CR) を作成するアクセス許可が必要です。両方の CRD は GitHub の `crds.yaml` ファイルで利用できます。具体的には、VPC CNI には `policyendpoints` の「get」、「list」、「watch」動詞のアクセス許可が必要です。

Kubernetes ネットワークポリシーは、`v1.networking.k8s.io` と呼ばれる `apiservice` の一部であり、これはポリシー YAML ファイル内では `apiversion: networking.k8s.io/v1` となります。VPC CNI `DaemonSet` には、Kubernetes API のこの部分を使用するアクセス許可が必要です。

VPC CNI アクセス許可は、`aws-node` と呼ばれる `ClusterRole` にあります。`ClusterRole` オブジェクトは、名前空間にグループ化されないことに注意してください。次の内容では、クラスターの `aws-node` が示されます。

```
kubectl get clusterrole aws-node -o yaml
```

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  labels:
    app.kubernetes.io/instance: aws-vpc-cni
    app.kubernetes.io/managed-by: Helm
    app.kubernetes.io/name: aws-node
    app.kubernetes.io/version: v1.19.4
    helm.sh/chart: aws-vpc-cni-1.19.4
    k8s-app: aws-node
  name: aws-node
rules:
- apiGroups:
  - crd.k8s.amazonaws.com
  resources:
  - eniconfigs
  verbs:
  - list
  - watch
  - get
- apiGroups:
  - ""
  resources:
  - namespaces
  verbs:
  - list
  - watch
  - get
- apiGroups:
  - ""
  resources:
  - pods
  verbs:
  - list
  - watch
  - get
- apiGroups:
  - ""
  resources:
  - nodes
  verbs:
  - list
  - watch
```

```
  - get
- apiGroups:
  - ""
  - events.k8s.io
  resources:
  - events
  verbs:
  - create
  - patch
  - list
- apiGroups:
  - networking.k8s.aws
  resources:
  - policyendpoints
  verbs:
  - get
  - list
  - watch
- apiGroups:
  - networking.k8s.aws
  resources:
  - policyendpoints/status
  verbs:
  - get
- apiGroups:
  - vpcresources.k8s.aws
  resources:
  - cninodes
  verbs:
  - get
  - list
  - watch
  - patch
```

また、新しいコントローラーは各 EKS クラスターのコントロールプレーンで実行されます。コントローラーは、eks:network-policy-controller と呼ばれる ClusterRole のアクセス許可を使用します。次の内容では、クラスターの eks:network-policy-controller が示されます。

```
kubectl get clusterrole eks:network-policy-controller -o yaml
```

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
```

```
metadata:
  labels:
    app.kubernetes.io/name: amazon-network-policy-controller-k8s
  name: eks:network-policy-controller
rules:
- apiGroups:
  - ""
  resources:
  - namespaces
  verbs:
  - get
  - list
  - watch
- apiGroups:
  - ""
  resources:
  - pods
  verbs:
  - get
  - list
  - watch
- apiGroups:
  - ""
  resources:
  - services
  verbs:
  - get
  - list
  - watch
- apiGroups:
  - networking.k8s.aws
  resources:
  - policyendpoints
  verbs:
  - create
  - delete
  - get
  - list
  - patch
  - update
  - watch
- apiGroups:
  - networking.k8s.aws
  resources:
```

```
  - policyendpoints/finalizers
  verbs:
  - update
- apiGroups:
  - networking.k8s.aws
  resources:
  - policyendpoints/status
  verbs:
  - get
  - patch
  - update
- apiGroups:
  - networking.k8s.io
  resources:
  - networkpolicies
  verbs:
  - get
  - list
  - patch
  - update
  - watch
```

ネットワークポリシーのログ

ネットワークポリシーによって接続が許可されるか拒否されるかについての VPC CNI による各判定は、フローログに記録されます。各ノードのネットワークポリシーログにはネットワークポリシーが設定されているすべてのポッドのフローログが含まれます。ネットワークポリシーログは /var/log/aws-routed-eni/network-policy-agent.log に保存されます。次の例は network-policy-agent.log ファイルからのものです。

```
{"level":"info","timestamp":"2023-05-30T16:05:32.573Z","logger":"ebpf-
client","msg":"Flow Info: ","Src
IP":"192.168.87.155","Src Port":38971,"Dest IP":"64.6.160","Dest
Port":53,"Proto":"UDP","Verdict":"ACCEPT"}
```

ネットワークポリシーログはデフォルトで無効になっています。ネットワークポリシーログを有効にするには次の手順に従います。

> **Note**
>
> ネットワークポリシーログには、VPC CNI `aws-node` DaemonSet マニフェストの `aws-network-policy-agent` コンテナ用に vCPU をさらに 1 つ追加する必要があります。

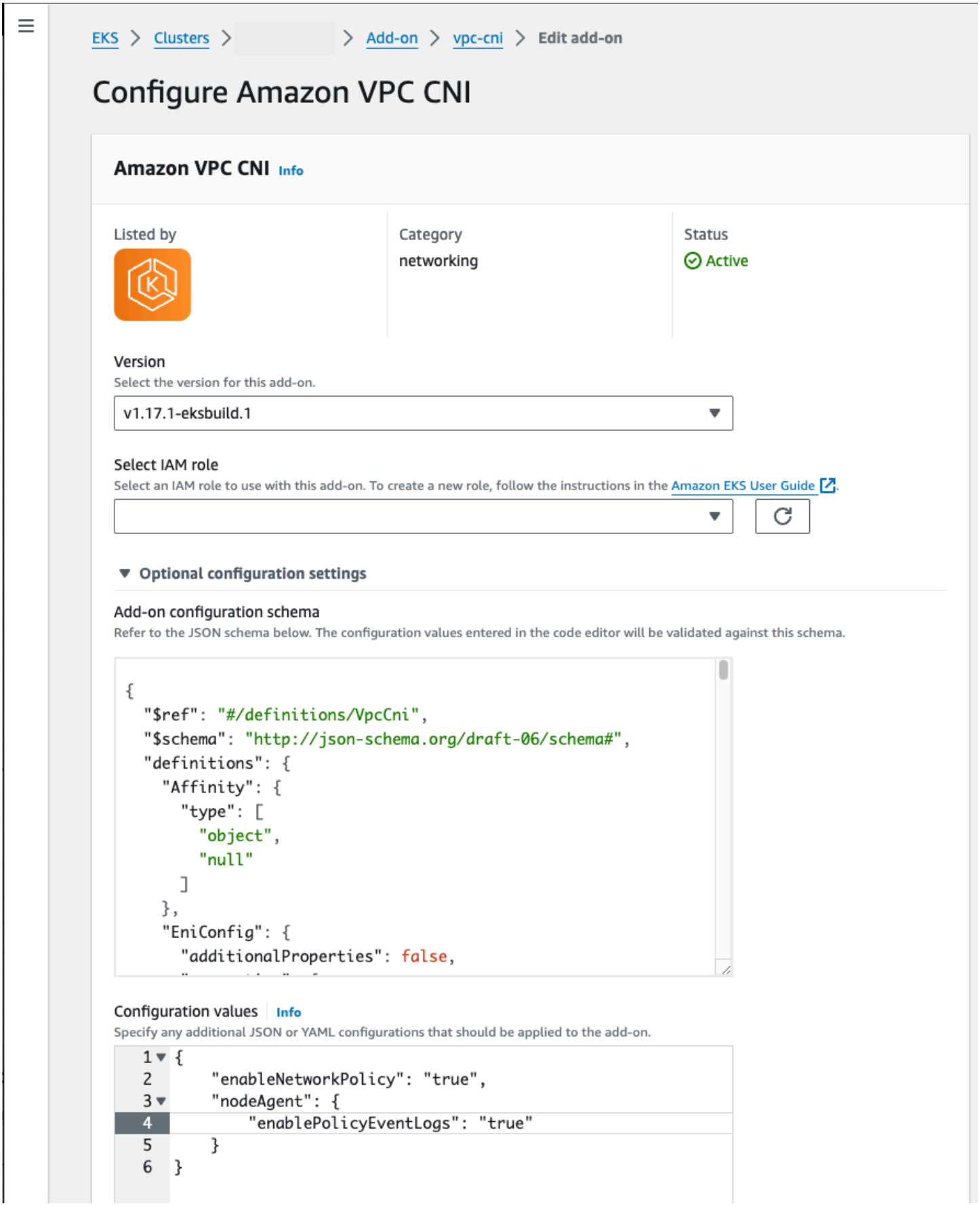
Amazon EKS アドオン

AWS マネジメントコンソール

a. Amazon EKS コンソールを開きます。
b. 左のナビゲーションペインで、[クラスター] を選択し、 Amazon VPC CNI アドオンを設定するクラスターの名前を選択してください。
c. [アドオン] タブを選択してください。
d. アドオンボックスの右上にあるボックスを選択し、次に [編集] を選択してください。
e. ***Amazon VPC CNI*** の設定ページで、次の手順に従います。
   i. [バージョン] ドロップダウンリストで `v1.14.0-eksbuild.3` 以降のバージョンを選択してください。
   ii. [オプションの構成設定] を展開します。
   iii. 最上位の JSON キー `"nodeAgent":` を入力してください。値は [設定値] にキー `"enablePolicyEventLogs":` と `"true"` の値を持つオブジェクトです。結果のテキストは有効な JSON オブジェクトでなければなりません。次の例はネットワークポリシーとネットワークポリシーログが有効になっており、ネットワークポリシーログが CloudWatch Logs に送信されていることを示しています。

```
{
    "enableNetworkPolicy": "true",
    "nodeAgent": {
        "enablePolicyEventLogs": "true"
    }
}
```

次のスクリーンショットはこのシナリオの例を示しています。


EKS > Clusters > > Add-on > vpc-cni > Edit add-on
Configure Amazon VPC CNI
Amazon VPC CNI Info
Listed by
Category
networking
Status
Active
Version
Select the version for this add-on.
v1.17.1-eksbuild.1
Select IAM role
Select an IAM role to use with this add-on. To create a new role, follow the instructions in the Amazon EKS User Guide.
Optional configuration settings
Add-on configuration schema
Refer to the JSON schema below. The configuration values entered in the code editor will be validated against this schema.
{
"$ref": "#/definitions/VpcCni",
"$schema": "http://json-schema.org/draft-06/schema#",
"definitions": {
"Affinity": {
"type": [
"object",
"null"
]
},
"EniConfig": {
"additionalProperties": false,
Configuration values Info
Specify any additional JSON or YAML configurations that should be applied to the add-on.
1 {
2 "enableNetworkPolicy": "true",
3 "nodeAgent": {
4 "enablePolicyEventLogs": "true"
5 }
6 }

AWS CLI

a. 次の AWS CLI コマンドを実行してください。my-cluster をクラスターの名前に置き換え、IAM 役割 ARN を使用する役割に置き換えます。

```
aws eks update-addon --cluster-name my-cluster --addon-name vpc-cni --addon-
version v1.14.0-eksbuild.3 \
    --service-account-role-arn arn:aws:iam::123456789012:role/AmazonEKSVPCCNIRole
 \
    --resolve-conflicts PRESERVE --configuration-values '{"nodeAgent":
 {"enablePolicyEventLogs": "true"}}'
```

セルフマネージド型アドオン

Helm

helm を通して Amazon VPC CNI plugin for Kubernetes をインストールしている場合、設定を更新してネットワークポリシーログを記述できます。

a. 次のコマンドを実行してネットワークポリシーを有効にします。

```
helm upgrade --set nodeAgent.enablePolicyEventLogs=true aws-vpc-cni --namespace
 kube-system eks/aws-vpc-cni
```

kubectl

kubectl を通して Amazon VPC CNI plugin for Kubernetes をインストールしている場合、設定を更新してネットワークポリシーログを記述できます。

a. エディターで aws-nodeDaemonSet を開きます。

```
kubectl edit daemonset -n kube-system aws-node
```

b. VPC CNI aws-node DaemonSet マニフェストの aws-network-policy-agent コンテナで、args: のコマンド引数 --enable-policy-event-logs=false で false を true に置き換えます。

```
    - args:
       - --enable-policy-event-logs=true
```

ネットワークポリシーログを Amazon CloudWatch Logs に送信する

Amazon CloudWatch Logs などのサービスを使用して、ネットワークポリシーログをモニタリングできます。次の方法を使用して、ネットワークポリシーログを CloudWatch Logs に送信できます。

EKS クラスターの場合、ポリシー ログは `/aws/eks/cluster-name/cluster/` の下に配置され、セルフマネージド K8S クラスターの場合、ログは `/aws/k8s-cluster/cluster/` の下に配置されます。

Amazon VPC CNI plugin for Kubernetes によるネットワークポリシーログの送信

ネットワークポリシーを有効にすると、2 つ目のコンテナがノードエージェントの `aws-node` ポッドに追加されます。このノードエージェントはネットワークポリシーログを CloudWatch Logs に送信できます。

> **Note**
>
> ノードエージェントはネットワークポリシーログのみを送信します。VPC CNI によって作成された他のログは含まれません。

前提条件

- VPC CNI に使用している IAM 役割に、次の権限をスタンザまたは個別のポリシーとして追加します。

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "VisualEditor0",
            "Effect": "Allow",
            "Action": [
                "logs:DescribeLogGroups",
                "logs:CreateLogGroup",
                "logs:CreateLogStream",
                "logs:PutLogEvents"
            ],
            "Resource": "*"
        }
    ]
```

```
}
```

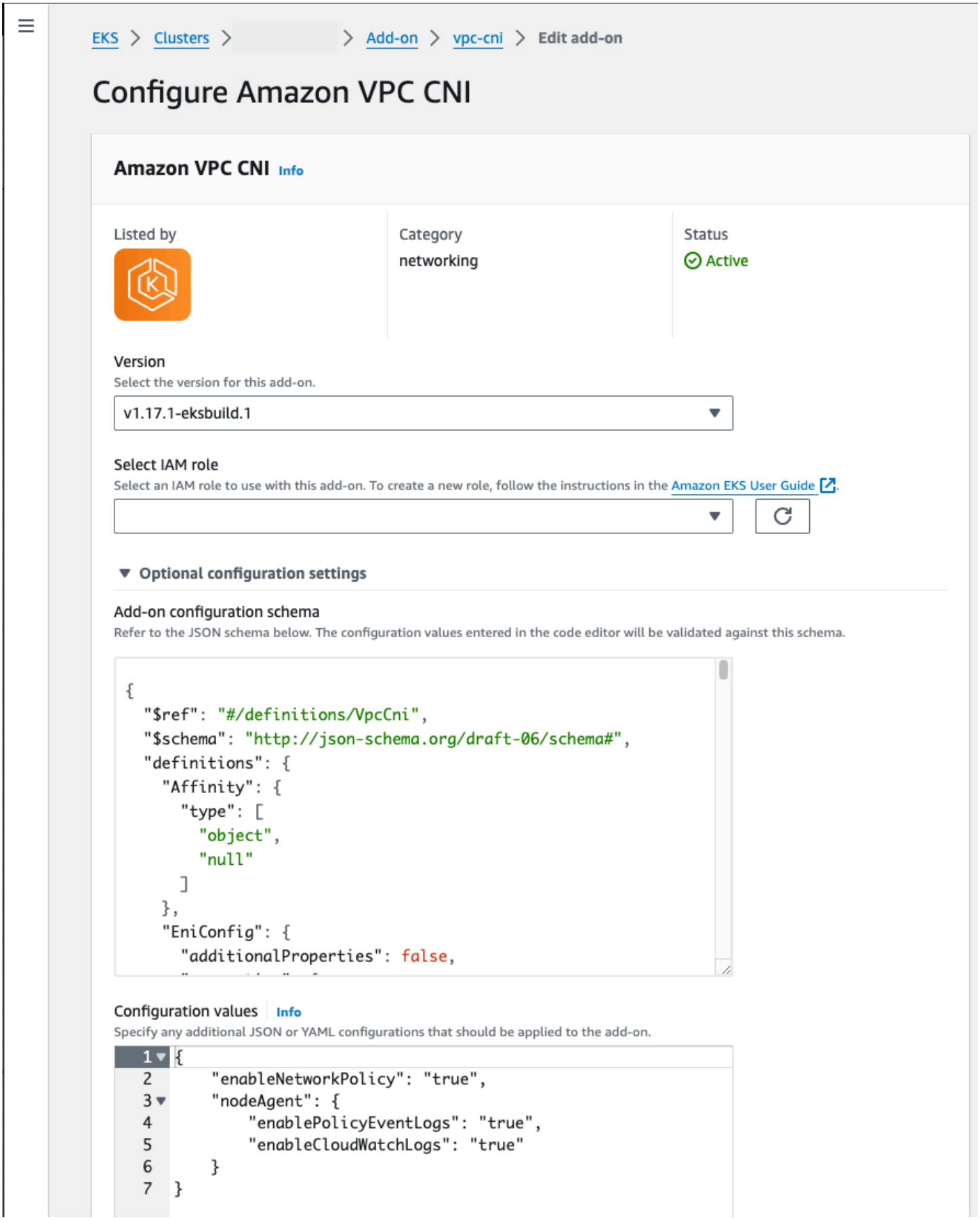
Amazon EKS アドオン

AWS マネジメントコンソール

a. Amazon EKS コンソールを開きます。
b. 左のナビゲーションペインで、[クラスター] を選択し、 Amazon VPC CNI アドオンを設定するクラスターの名前を選択してください。
c. [アドオン] タブを選択してください。
d. アドオンボックスの右上にあるボックスを選択し、次に [編集] を選択してください。
e. ***Amazon VPC CNI*** の設定ページで、次の手順に従います。
   i. [バージョン] ドロップダウンリストで `v1.14.0-eksbuild.3` 以降のバージョンを選択してください。
   ii. [オプションの構成設定] を展開します。
   iii. 最上位の JSON キー `"nodeAgent":` を入力してください。値は [設定値] にキー `"enableCloudWatchLogs":` と `"true"` の値を持つオブジェクトです。結果のテキストは有効な JSON オブジェクトでなければなりません。次の例はネットワークポリシーとネットワークポリシーログが有効になっており、ログが CloudWatch Logs に送信されていることを示しています。

```
{
    "enableNetworkPolicy": "true",
    "nodeAgent": {
        "enablePolicyEventLogs": "true",
        "enableCloudWatchLogs": "true",
    }
}
```

次のスクリーンショットはこのシナリオの例を示しています。


EKS > Clusters > > Add-on > vpc-cni > Edit add-on
Configure Amazon VPC CNI
Amazon VPC CNI Info
Listed by
Category
networking
Status
Active
Version
Select the version for this add-on.
v1.17.1-eksbuild.1
Select IAM role
Select an IAM role to use with this add-on. To create a new role, follow the instructions in the Amazon EKS User Guide.
Optional configuration settings
Add-on configuration schema
Refer to the JSON schema below. The configuration values entered in the code editor will be validated against this schema.
{
"$ref": "#/definitions/VpcCni",
"$schema": "http://json-schema.org/draft-06/schema#",
"definitions": {
"Affinity": {
"type": [
"object",
"null"
]
},
"EniConfig": {
"additionalProperties": false,
Configuration values Info
Specify any additional JSON or YAML configurations that should be applied to the add-on.
1 {
2 "enableNetworkPolicy": "true",
3 "nodeAgent": {
4 "enablePolicyEventLogs": "true",
5 "enableCloudWatchLogs": "true"
6 }
7 }

AWS CLI

a. 次の AWS CLI コマンドを実行してください。my-cluster をクラスターの名前に置き換え、IAM 役割 ARN を使用する役割に置き換えます。

```
aws eks update-addon --cluster-name my-cluster --addon-name vpc-cni --addon-
version v1.14.0-eksbuild.3 \
    --service-account-role-arn arn:aws:iam::123456789012:role/AmazonEKSVPCCNIRole
 \
    --resolve-conflicts PRESERVE --configuration-values '{"nodeAgent":
 {"enablePolicyEventLogs": "true", "enableCloudWatchLogs": "true"}}'
```

セルフマネージド型アドオン

Helm

helm を通して Amazon VPC CNI plugin for Kubernetes をインストールしている場合、設定を更新してネットワークポリシーログを CloudWatch Logs に送信できます。

a. 次のコマンドを実行してネットワークポリシーログを有効にし、CloudWatch Logs に送信します。

```
helm upgrade --set nodeAgent.enablePolicyEventLogs=true --set
 nodeAgent.enableCloudWatchLogs=true aws-vpc-cni --namespace kube-system eks/aws-
vpc-cni
```

kubectl

a. エディターで aws-nodeDaemonSet を開きます。

```
kubectl edit daemonset -n kube-system aws-node
```

b. VPC CNI aws-node DaemonSet マニフェストの aws-network-policy-agent コンテナで、args: の 2 つのコマンド引数 --enable-policy-event-logs=false および --enable-cloudwatch-logs=false の false を true に置き換えます。

```
    - args:
       - --enable-policy-event-logs=true
       - --enable-cloudwatch-logs=true
```

Fluent Bit **DaemonSet**と一緒にネットワークポリシーログの送信

DaemonSet で Fluent Bit を使用してノードからログを送信する場合、ネットワークポリシーのネットワークポリシーログを含めるように設定を追加できます。次の設定例を使用できます。

```
    [INPUT]
        Name              tail
        Tag               eksnp.*
        Path              /var/log/aws-routed-eni/network-policy-agent*.log
        Parser            json
        DB                /var/log/aws-routed-eni/flb_npagent.db
        Mem_Buf_Limit     5MB
        Skip_Long_Lines   On
        Refresh_Interval  10
```

eBPF SDK の内容

Amazon VPC CNI plugin for Kubernetes は、複数のツールをまとめた eBPF SDK コレクションをノードにインストールします。eBPF SDK ツールを使用して、ネットワークポリシーの問題を特定できます。例えば、次のコマンドはノードで実行されているプログラムを一覧表示します。

```
sudo /opt/cni/bin/aws-eks-na-cli ebpf progs
```

このコマンドを実行するために、任意の方法を使用してノードに接続できます。

既知の問題と解決策

次のセクションでは、Amazon VPC CNI ネットワークポリシーの機能に関する既知の問題とその解決策について説明します。

enable-policy-event-logs が「false」に設定されているにも関わらず、ネットワークポリシーログが生成される

問題: enable-policy-event-logs 設定が false に設定されている場合でも、EKS VPC CNI によってネットワークポリシーログが生成される。

解決策: enable-policy-event-logs 設定はポリシー「決定」ログのみを無効にしますが、すべてのネットワークポリシーエージェントのログ記録が無効になるわけではありません。この動作については、GitHub の「aws-network-policy-agent README」に記載されています。ログ記録を完全に無効にするには、他のログ記録設定の調整が必要な場合があります。

ネットワークポリシーマップのクリーンアップ問題

問題: ポッドが削除された後も、ネットワーク policyendpoint が残り続けてクリーンアップされない問題。

解決策: この問題は、VPC CNI アドオンバージョン 1.19.3-eksbuild.1 による問題で発生しました。この問題を解決するには、VPC CNI アドオンの新しいバージョンに更新してください。

ネットワークポリシーが適用されない

問題: Amazon VPC CNI プラグインでネットワークポリシー機能が有効になっているが、ネットワークポリシーが正しく適用されない。

ネットワークポリシー kind: NetworkPolicy を作成してもポッドに影響を与えない場合、policyendpoint オブジェクトがポッドと同じ名前空間で作成されていることを確認します。名前空間に policyendpoint オブジェクトがない場合、ネットワークポリシーコントローラー (EKS クラスターの一部) が、ネットワークポリシーエージェント (VPC CNI の一部) が適用するためのネットワークポリシールールを作成できなかったということです。

解決策: この解決策は、VPC CNI (ClusterRole: aws-node) およびネットワークポリシーコントローラー (ClusterRole: eks:network-policy-controller) のアクセス許可を修正し、Kyverno などのポリシー実施ツールでこれらのアクションを許可することです。Kyverno ポリシーが policyendpoint オブジェクトの作成をブロックしていないことを確認してください。必要なアクセス許可については、「the section called “新しい policyendpoints CRD とアクセス許可”」で前のセクションを参照してください。

strict モードでポリシーを削除してもポッドがデフォルトの拒否状態に戻らない

問題: ネットワークポリシーが strict モードで有効になっている場合、ポッドはデフォルト拒否ポリシーで開始されます。ポリシーが適用されると、指定されたエンドポイントへのトラフィックが許可されます。しかし、ポリシーが削除されても、ポッドがデフォルト拒否状態に戻らず、代わりにデフォルト許可状態になってしまいます。

解決策: この問題は、ネットワークポリシーエージェント 1.2.0 のリリースを含む VPC CNI リリース 1.19.3 で修正されました。修正後は、strict モードを有効にすると、ポリシーが削除されるとポッドは期待どおりにデフォルト拒否状態に戻ります。

ポッドの起動レイテンシーのセキュリティグループ

問題: EKS でポッド機能のセキュリティグループを使用すると、ポッドの起動レイテンシーが増加する。

解決策: レイテンシーは、VPC リソースコントローラーがポッドのブランチ ENI を作成するために使用する CreateNetworkInterface API の API スロットリングによるリソースコントローラーのレート制限によるものです。このオペレーションのアカウントの API 制限を確認し、必要に応じて上限の引き上げをリクエストすることを検討してください。

vpc.amazonaws.com/pod-eni の不足による FailedScheduling

問題: 以下のエラーにより、ポッドのスケジューリングが失敗する: `FailedScheduling 2m53s (x28 over 137m) default-scheduler 0/5 nodes are available: 5 Insufficient vpc.amazonaws.com/pod-eni. preemption: 0/5 nodes are available: 5 No preemption victims found for incoming pod.`

解決策: 前の問題と同様に、ポッドにセキュリティグループを割り当てると、ポッドスケジューリングのレイテンシーが増加し、各 ENI を追加する時間の CNI しきい値を超えて増加するため、ポッド起動が失敗する原因になります。これは、Podのセキュリティグループを使用するときに予期される動作です。ワークロードアーキテクチャを設計する際、スケジューリングの影響を考慮してください。

IPAM の接続問題とセグメンテーション障害

問題: IPAM の接続問題、スロットリングのリクエスト、セグメンテーションの障害など、複数のエラーが発生する。

- `Checking for IPAM connectivity …`
- `Throttling request took 1.047064274s`
- `Retrying waiting for IPAM-D`
- `panic: runtime error: invalid memory address or nil pointer dereference`

解決策: AL2023 で `systemd-udev` をインストールすると、破損を引き起こすポリシーでファイルが書き換えられるため、この問題が発生します。パッケージが更新された別の `releasever` に更新する場合、またはパッケージ自体を手動で更新する場合に発生する可能性があります。AL2023 ノードでは、`systemd-udev` をインストールまたは更新しないでください。

デバイス名による検索失敗エラー

問題: エラーメッセージ:
`{"level":"error","ts":"2025-02-05T20:27:18.669Z","caller":"ebpf/bpf_client.go:578","msg":"failed to find device by name eni9ea69618bf0: %!w(netlink.LinkNotFoundError={0xc000115310})"}`

解決策: この問題は、Amazon VPC CNI ネットワークポリシーエージェント (v1.2.0) の最新バージョンで特定されて修正されています。この問題を解決するには、VPC CNI の最新バージョンに更新してください。

Multus CNI イメージの CVE 脆弱性

問題: 拡張された EKS ImageScan CVE レポートで、Multus CNI イメージバージョン v4.1.4-eksbuild.2_thick の脆弱性が特定されている。

解決策: 脆弱性のない Multus CNI イメージの新しいバージョンおよび新しい Network Policy Controller イメージに更新します。スキャナーを更新することで、以前のバージョンで見つかった脆弱性に対処できます。

ログのフロー情報の「DENY」判定

問題: ネットワークポリシーログに次の DENY 判定が表示される:

```
{"level":"info","ts":"2024-11-25T13:34:24.808Z","logger":"ebpf-
client","caller":"events/events.go:193","msg":"Flow
Info: ","Src IP":"","Src Port":9096,"Dest IP":"","Dest
Port":56830,"Proto":"TCP","Verdict":"DENY"}
```

解決策: この問題は、ネットワークポリシーコントローラーの新しいバージョンで解決されています。ログ記録の問題を解決するには、最新の EKS プラットフォームバージョンに更新してください。

Calico から移行後のポッド間の通信問題

問題: EKS クラスターをバージョン 1.30 にアップグレードし、ネットワークポリシーを Calico から Amazon VPC CNI に切り替えた後、ネットワークポリシーが適用されるとポッド間の通信が失敗する。ネットワークポリシーを削除すると通信は回復する。

解決策: VPC CNI のネットワークポリシーエージェントには、Calico ほど多くのポートを指定することができません。代わりに、ネットワークポリシーでポート範囲を使用します。ネットワークポリシーの各 `ingress:` または `egress:` セレクターの各プロトコルにおけるポートの一意の組み合わせの最大数は 24 です。ポート範囲を使用して一意のポート数を減らし、この制限を回避します。

ネットワークポリシーエージェントがスタンドアロンポッドをサポートしない

問題: スタンドアロンポッドに適用されるネットワークポリシーは、一貫性のない動作を示す可能性がある。

解決策: 現在、ネットワークポリシーエージェントは、デプロイメント/レプリカセットの一部としてデプロイされたポッドのみをサポートしています。ネットワークポリシーがスタンドアロンポッドに適用された場合、一貫性のない動作を示す可能性があります。これは、このページの上部の「the section called “考慮事項”」および GitHub の「aws-network-policy-agent の GitHub 問題 #327」に記載されています。整合性のあるネットワークポリシーの動作のため、デプロイまたはレプリカセットの一部としてポッドをデプロイします。

Amazon EKS のネットワークポリシーの星型デモ

このデモでは、Amazon EKS クラスターにフロントエンド、バックエンド、およびクライアントサービスを作成します。また、このデモでは、各サービス間で利用可能な入力および出力のパスを示す管理グラフィカルユーザーインターフェースが作成されます。実稼働ワークロードを実行しないクラスターでデモを完了することをお勧めします。

ネットワークポリシーを作成する前に、サービスはすべて、双方向に通信できます。ネットワークポリシーを適用すると、クライアントはフロントエンドサービスとのみ通信でき、バックエンドはフロントエンドからのみトラフィックを受け付けます。

1. フロントエンド、バックエンド、クライアント、および管理ユーザーインターフェイスサービスを適用します。

```
kubectl apply -f https://raw.githubusercontent.com/aws-samples/eks-
workshop/2f9d29ed3f82ed6b083649e975a0e574fb8a4058/content/beginner/120_network-
policies/calico/stars_policy_demo/create_resources.files/namespace.yaml
kubectl apply -f https://raw.githubusercontent.com/aws-samples/eks-
workshop/2f9d29ed3f82ed6b083649e975a0e574fb8a4058/content/beginner/120_network-
policies/calico/stars_policy_demo/create_resources.files/management-ui.yaml
kubectl apply -f https://raw.githubusercontent.com/aws-samples/eks-
workshop/2f9d29ed3f82ed6b083649e975a0e574fb8a4058/content/beginner/120_network-
policies/calico/stars_policy_demo/create_resources.files/backend.yaml
kubectl apply -f https://raw.githubusercontent.com/aws-samples/eks-
workshop/2f9d29ed3f82ed6b083649e975a0e574fb8a4058/content/beginner/120_network-
policies/calico/stars_policy_demo/create_resources.files/frontend.yaml
kubectl apply -f https://raw.githubusercontent.com/aws-samples/eks-
workshop/2f9d29ed3f82ed6b083649e975a0e574fb8a4058/content/beginner/120_network-
policies/calico/stars_policy_demo/create_resources.files/client.yaml
```

2. クラスター上の Pod をすべて表示します。

```
kubectl get pods -A
```

出力例は次のとおりです。

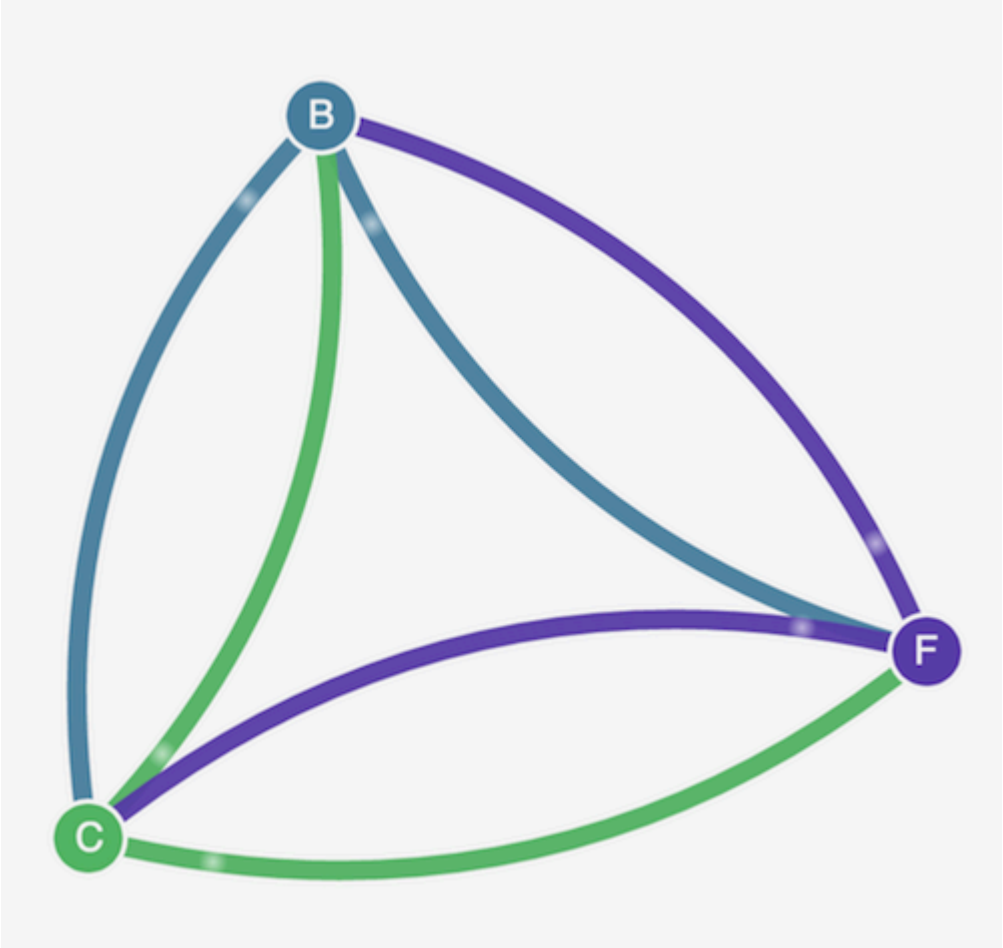
出力では、次の出力に示す名前空間にポッドが表示されます。READY 列内のポッドの *NAMES* とポッド数は、次の出力とは異なります。似たような名前のポッドが見えて、それらがすべて STATUS 列内に Running を持つまで続けないでください。

```
NAMESPACE         NAME                                  READY   STATUS
 RESTARTS   AGE
[...]
client            client-xlffc                          1/1     Running   0
     5m19s
[...]
management-ui     management-ui-qrb2g                   1/1     Running   0
     5m24s
stars             backend-sz87q                         1/1     Running   0
     5m23s
stars             frontend-cscnf                        1/1     Running   0
     5m21s
[...]
```

3. 管理ユーザーインターフェイスに接続するには、クラスターで実行されているサービスの EXTERNAL-IP に接続します。

```
kubectl get service/management-ui -n management-ui
```

4. ブラウザを開いて、前のステップの場所に移動します。管理ユーザーインターフェイスが表示されます。[C] ノードはクライアントサービス、[F] ノードはフロントエンドサービス、[B] ノードはバックエンドサービスを表します。各ノードには、太字の色付きの行で示されるように、他のすべてのノードへの完全な通信アクセスが含まれます。



5. 相互にサービスを分離するには、次のネットワークポリシーを stars および client の名前空間の両方に適用します。

```
kind: NetworkPolicy
apiVersion: networking.k8s.io/v1
metadata:
  name: default-deny
spec:
  podSelector:
    matchLabels: {}
```

次のコマンドを使用して、ポリシーを両方の名前空間に適用できます。

```
kubectl apply -n stars -f https://raw.githubusercontent.com/aws-samples/eks-
workshop/2f9d29ed3f82ed6b083649e975a0e574fb8a4058/content/beginner/120_network-
policies/calico/stars_policy_demo/apply_network_policies.files/default-deny.yaml
```

```
kubectl apply -n client -f https://raw.githubusercontent.com/aws-samples/eks-
workshop/2f9d29ed3f82ed6b083649e975a0e574fb8a4058/content/beginner/120_network-
policies/calico/stars_policy_demo/apply_network_policies.files/default-deny.yaml
```

6. ブラウザを更新します。管理ユーザーインターフェイスはノードに到達できなくなるため、ユーザーインターフェイスに表示されないことがわかります。
7. 管理ユーザーインターフェイスがサービスにアクセスできるように、次の別のネットワークポリシーを適用します。このポリシーを適用して UI を許可します。

```
kind: NetworkPolicy
apiVersion: networking.k8s.io/v1
metadata:
  namespace: stars
  name: allow-ui
spec:
  podSelector:
    matchLabels: {}
  ingress:
    - from:
        - namespaceSelector:
            matchLabels:
              role: management-ui
```

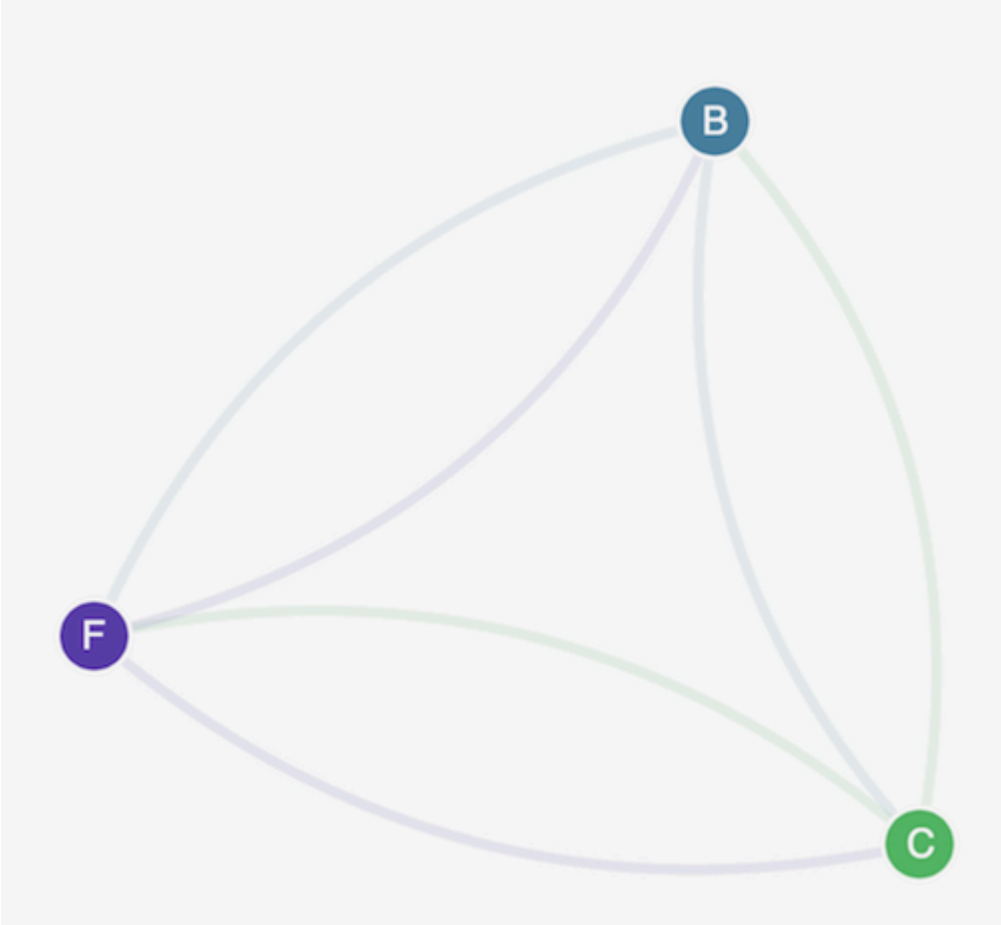
このポリシーを適用してクライアントを許可します。

```
kind: NetworkPolicy
apiVersion: networking.k8s.io/v1
metadata:
  namespace: client
  name: allow-ui
spec:
  podSelector:
    matchLabels: {}
  ingress:
    - from:
        - namespaceSelector:
            matchLabels:
              role: management-ui
```

次のコマンドを使用して両方のポリシーを適用できます。

```
kubectl apply -f https://raw.githubusercontent.com/aws-samples/eks-
workshop/2f9d29ed3f82ed6b083649e975a0e574fb8a4058/content/beginner/120_network-
policies/calico/stars_policy_demo/apply_network_policies.files/allow-ui.yaml
kubectl apply -f https://raw.githubusercontent.com/aws-samples/eks-
workshop/2f9d29ed3f82ed6b083649e975a0e574fb8a4058/content/beginner/120_network-
policies/calico/stars_policy_demo/apply_network_policies.files/allow-ui-client.yaml
```

8. ブラウザを更新します。管理ユーザーインターフェイスは再びノードにアクセスできますが、ノードは相互に通信できないことがわかります。

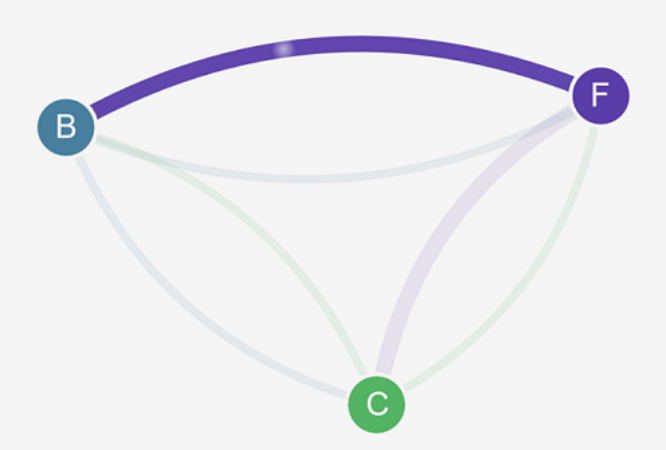


9. フロントエンドサービスからバックエンドサービスへのトラフィックを許可するには、次のネットワークポリシーを適用します。

```
kind: NetworkPolicy
apiVersion: networking.k8s.io/v1
metadata:
  namespace: stars
  name: backend-policy
spec:
```

```
  podSelector:
    matchLabels:
      role: backend
  ingress:
    - from:
        - podSelector:
            matchLabels:
              role: frontend
      ports:
        - protocol: TCP
          port: 6379
```

10.ブラウザを更新します。フロントエンドがバックエンドと通信できることがわかります。

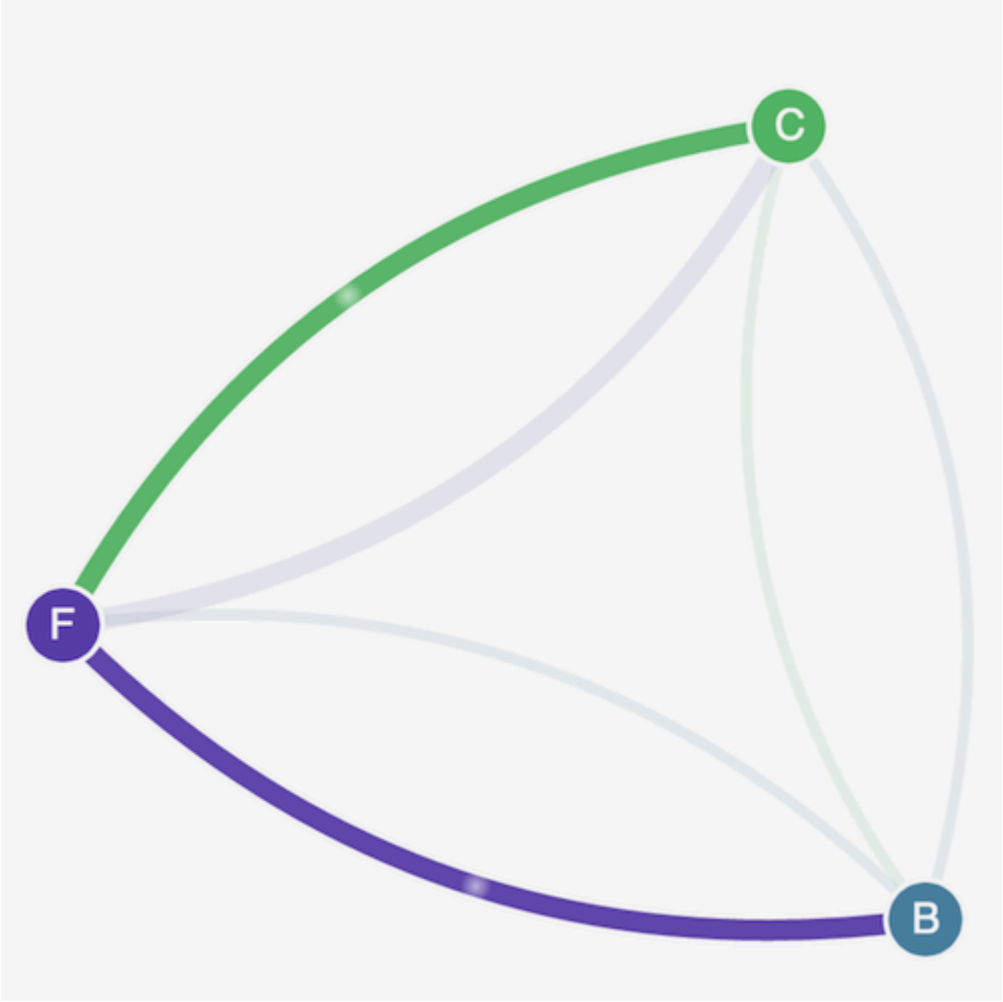


11.クライアントからフロントエンドサービスへのトラフィックを許可するには、次のネットワークポリシーを適用します。

```
kind: NetworkPolicy
apiVersion: networking.k8s.io/v1
metadata:
  namespace: stars
  name: frontend-policy
spec:
  podSelector:
    matchLabels:
      role: frontend
  ingress:
    - from:
        - namespaceSelector:
            matchLabels:
              role: client
      ports:
```

```
        - protocol: TCP
          port: 80
```

12.ブラウザを更新します。クライアントがフロントエンドサービスと通信できることがわかります。フロントエンドサービスは、バックエンドサービスと引き続き通信できます。



13(オプション) デモが完了したら、そのリソースを削除することができます。

```
kubectl delete -f https://raw.githubusercontent.com/aws-samples/eks-
workshop/2f9d29ed3f82ed6b083649e975a0e574fb8a4058/content/beginner/120_network-
policies/calico/stars_policy_demo/create_resources.files/client.yaml
kubectl delete -f https://raw.githubusercontent.com/aws-samples/eks-
workshop/2f9d29ed3f82ed6b083649e975a0e574fb8a4058/content/beginner/120_network-
policies/calico/stars_policy_demo/create_resources.files/frontend.yaml
kubectl delete -f https://raw.githubusercontent.com/aws-samples/eks-
workshop/2f9d29ed3f82ed6b083649e975a0e574fb8a4058/content/beginner/120_network-
policies/calico/stars_policy_demo/create_resources.files/backend.yaml
```

```
kubectl delete -f https://raw.githubusercontent.com/aws-samples/eks-
workshop/2f9d29ed3f82ed6b083649e975a0e574fb8a4058/content/beginner/120_network-
policies/calico/stars_policy_demo/create_resources.files/management-ui.yaml
kubectl delete -f https://raw.githubusercontent.com/aws-samples/eks-
workshop/2f9d29ed3f82ed6b083649e975a0e574fb8a4058/content/beginner/120_network-
policies/calico/stars_policy_demo/create_resources.files/namespace.yaml
```

リソースを削除した後でも、ノード上にネットワークポリシーエンドポイントが残っている場合があり、それがクラスター内のネットワークを予期しない方法で妨げることがあります。これらのルールを削除する唯一の確実な方法は、ノードを再起動するか、すべてのノードを終了して、リサイクルすることです。すべてのノードを終了するには、Auto Scaling グループの必要数を 0 に設定してから目的数にバックアップするか、単にノードを終了します。

Pod IP アドレスのサブネット選択を設定する

適用対象: Amazon EC2 インスタンスを持つ Linux ノード

Amazon VPC CNI plugin for Kubernetes はセカンダリ Elastic Network Interface (ENI)をノード上に作成し、それらの ENI から Pod へ IP アドレスを割り当てます。デフォルトでは、VPC CNI はノードのプライマリネットワークインターフェイスと同じサブネットにセカンダリ ENI を作成します。VPC CNI が Pod IP アドレスに使用するサブネットは、次の方法で制御できます:

- 拡張サブネット検出 – VPC CNI は、同じ VPC とアベイラビリティーゾーン内の `kubernetes.io/role/cni` でタグ付けされたサブネットを自動的に検出して使用します。VPC CNI バージョン 1.18.0 以降が必要です。ほとんどのユースケースでは、この方法をお勧めします。
- カスタムネットワーク – `ENIConfig` カスタムリソースを使用して、アベイラビリティーゾーンごとにサブネットとセキュリティグループを手動で指定します。詳細については、「the section called “カスタムネットワーキング”」を参照してください。

> **Note**
>
> 両方の機能が有効になっている場合は、カスタムネットワーキングが優先されます。

拡張サブネット検出

VPC CNI バージョン 1.18.0 以降では、デフォルトで拡張サブネット検出が有効になります (ENABLE_SUBNET_DISCOVERY=true)。VPC CNI は、ノードと同じ VPC とアベイラビリティーゾーン内のサブネットを自動的に検出し、それらを使用してセカンダリ ENI を作成し、Pod IP アドレスを割り当てます。これにより、手動の ENIConfig 設定なしで使用可能な IP アドレス領域が拡張されます。

この機能が有効になっていることを確認するには:

```
kubectl describe ds aws-node -n kube-system | grep ENABLE_SUBNET_DISCOVERY
```

この機能を無効にするには、aws-node DaemonSet で ENABLE_SUBNET_DISCOVERY=false を設定します。

サブネットタグの動作 (**kubernetes.io/role/cni**)

kubernetes.io/role/cni タグは、VPC CNI が ENI オペレーションと Pod IP 割り当てのサブネットを処理する方法を制御します。タグの効果は、VPC CNI が新しい ENI を作成するか、既存の ENI を照合するかによって異なります。

> **Important**
>
> クラスタースコープおよび除外タグ付けメカニズムは、VPC CNI のバージョン v1.22.2 以降でのみ使用できます。このバージョンより前のバージョンでは、標準のオプトイン動作のみがタグ kubernetes.io/role/cni=1 を介して使用できます。

タグ値

次の表は、各タグ値がサブネットの動作にどのように影響するかをまとめたものです。

タグ値	新しい ENI の作成	既存の ENI の調整
1	オプトイン: VPC CNI はこのサブネットに新しい ENI を作成します。	ENI は Pod IP 割り当てに引き続き使用できます。
0	除外: VPC CNI は、このサブネットに新しい ENI を作成しません。	除外: VPC CNI は、このサブネット内の既存の ENI を Pod IP 割り当てから除外

タグ値	新しい ENI の作成	既存の ENI の調整
		します。この ENI から新しい Pod IP は割り当てられません。
なし (タグなし)	新しい ENI には使用されない: VPC CNI は、新しい ENI の作成用のタグのないセカンダリサブネットを選択しません。プライマリサブネット (ノードが起動したサブネット) は、下位互換性のために、タグがなくても新しい ENI の作成に使用されます。	中断なし: タグなしサブネットの既存の ENI は、Pod IP 割り当てに引き続き使用できます。VPC CNI は、これらの ENI を削除または除外しません。

作成と調整の動作

VPC CNI は、新しい ENI を作成する場合と既存の ENI を調整する場合とで、意図的に異なるポリシーを適用します。

- 作成 (セカンダリサブネットではフェイルクローズ): VPC CNI が新しいセカンダリ ENI を作成する必要がある場合、`kubernetes.io/role/cni=1` で明示的にタグ付けされたセカンダリサブネットのみを使用します。タグ付けされていないセカンダリサブネットは、新しい ENI の作成には選択されません。これにより、新しいネットワークインターフェイスは、管理者が明示的に承認したサブネットにのみ配置されます。
- 調整 (タグのないサブネットの場合はフェイルオープン): VPC CNI がノードに既にアタッチされている既存の ENI を起動または調整するときに、サブネットに `kubernetes.io/role/cni` タグがないことを理由に ENI を除外しません。これにより、これらの ENI の IP アドレスを既に使用している実行中の Pod が中断されるのを防ぎます。

VPC CNI はこの設計を意図的に使用します。タグ付けされていないサブネットから既にアタッチされている ENI を強制的に除外すると、その ENI から現在 IP アドレスを使用している Pod が破壊されます。

> **Important**
>
> サブネットが既存の ENI を含む新しい Pod IP を提供できないようにするには、`kubernetes.io/role/cni=0` でサブネットにタグ付けします。タグがないと、その

サブネットへの新しい ENI の作成のみが防止されます。既存の ENI を割り当てから除外しません。

プライマリサブネットの処理

VPC CNI には、kubernetes.io/role/cni タグがない場合でも、ENI 作成用のノードのプライマリサブネット (ノードが起動したサブネット) が常に含まれます。これにより、既存のクラスターとの下位互換性が維持されます。プライマリサブネットの動作:

- タグの有無に関係なく、ENI 作成に含まれます (0 がタグ付けされている場合を除く)。
- kubernetes.io/role/cni=0 でタグ付けされている場合、VPC CNI は新しい ENI 作成と既存の ENI 割り当ての両方からプライマリサブネットを除外します。

クラススコープのサブネットフィルタリング

kubernetes.io/role/cni=1 でサブネットにタグ付けされている場合、VPC CNI はさらにキー形式 cni.networking.k8s.aws/cluster/<cluster-name> を使用してクラスター固有のタグをチェックします。サブネットにこの形式のクラスタータグがある場合、名前が一致するクラスターのみがそのサブネットを使用します。kubernetes.io/role/cni=1 を持ち、クラスター固有のタグを持たないサブネットは、VPC 内のすべてのクラスターで使用できます。

例えば、サブネットを特定のクラスターに制限するには:

```
aws ec2 create-tags --resources subnet-example \
  --tags Key=kubernetes.io/role/cni,Value=1 Key=cni.networking.k8s.aws/cluster/my-
cluster,Value=shared
```

これは、複数の EKS クラスターが VPC を共有し、各クラスターで Pod IP アドレスに異なるサブネットを使用する場合に便利です。

推奨されるワークフロー

Pod IP アドレスに新しいサブネットを追加するには:

1. ノードと同じ VPC とアベイラビリティーゾーンに新しいサブネットを作成します。
2. kubernetes.io/role/cni=1 でサブネットにタグ付けします。
3. サブネットに適切なルートテーブルとネットワーク ACL があることを確認します。
4. VPC CNI が新しいサブネットを検出し、使用を開始することを確認します。

Pod IP 割り当てからサブネットを削除するには:

1. `kubernetes.io/role/cni=0` でサブネットにタグ付けします。
2. そのサブネットの IP を使用しているポッドが自然に終了または再スケジュールされるまで待ちます。
3. VPC CNI がそのサブネット内の ENI からの新しい Pod IP の割り当てを停止することを確認します。

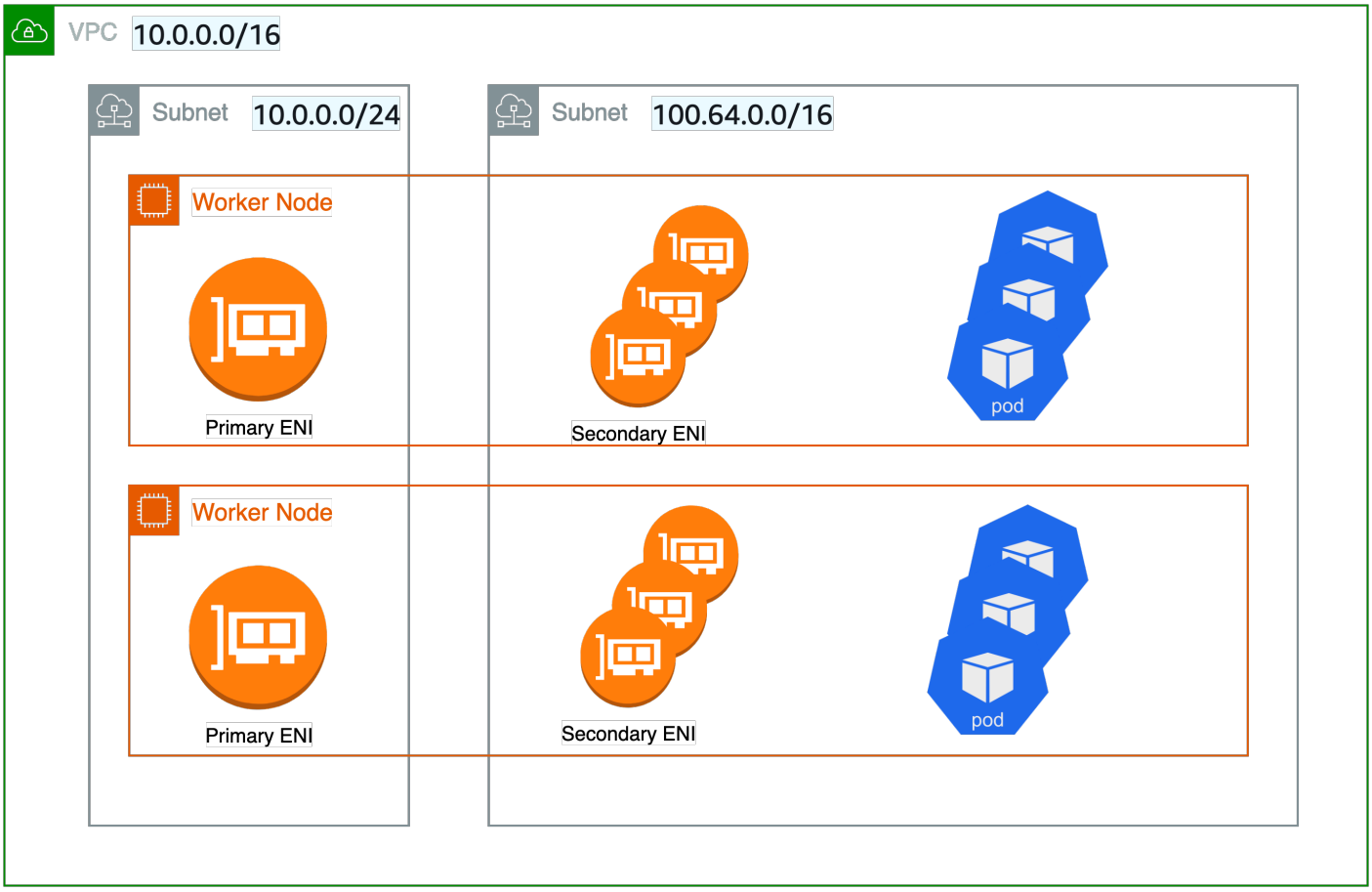
> **Important**
>
> サブネットの使用を停止するには、`kubernetes.io/role/cni` タグを削除しないでください。タグを削除すると新しい ENI は作成されなくなりますが、既存の ENI は割り当てから除外されません。サブネットをアクティブに除外するには、`kubernetes.io/role/cni=0` でタグ付けします。

考慮事項

- 拡張サブネット検出には、Amazon VPC CNI バージョン 1.18.0 以降が必要です。
- この機能には、VPC CNI IAM ロールの `ec2:DescribeSubnets` アクセス許可が必要です。`AmazonEKS_CNI_Policy` マネージドポリシーにそのアクセス許可が含まれています。IPv6 セルフマネージド IAM ポリシーにはそれは含まれていません。セルフマネージド IAM ポリシー (IPv6 クラスターなど) を使用する場合は、`ec2:DescribeSubnets` を手動で追加してサブネット検出を有効にします。
- この機能は、セカンダリ IP アドレスモードとプレフィックス委任モードの両方で機能します。
- 検出されたすべてのサブネットは、ノードと同じ VPC にある必要があります。
- VPC CNI は、ノードと同じアベイラビリティーゾーンにあるサブネットにのみ ENI を作成します。
- VPC CNI は、タグの変更に関係なく、Pod に割り当てられた IP アドレスがまだある ENI の割り当てを解除しません。
- 共有 VPC (クロスアカウントサブネット) を使用する場合は、クラスターが起動される参加者アカウントのサブネットにタグ付けします。
- 拡張サブネット検出は、Pod のセキュリティグループ、ネットワークポリシー、プレフィックス委任、および SNAT とともに使用できます。

カスタムネットワーキングを使用して代替サブネットに Pod をデプロイする

適用対象: Linux IPv4 Fargate ノード、Amazon EC2 インスタンスを持つ Linux ノード



Amazon VPC CNI plugin for Kubernetes が、Amazon EC2 ノード用にセカンダリ Elastic Network Interface (ネットワークインターフェイス) を作成する際には、デフォルトで、ノードのプライマリネットワークインターフェイスと同じサブネットがその格納先になります。また、プライマリネットワークインターフェイスに関連付けられているものと同じセキュリティグループが、セカンダリネットワークインターフェイスにも関連付けられます。以下の 1 つ以上の理由により、プラグインを使用して異なるサブネットにセカンダリネットワークインターフェイスを作成させたり、セカンダリネットワークインターフェイスに異なるセキュリティグループを関連付けたり、また、その両方を行ったりする必要性が生じることがあります。

- プライマリネットワークインターフェイスが存在するサブネットで使用可能な IPv4 アドレスには、数の上で制限があります。これにより、サブネット内に作成できる Pod の数に制限が生じる可能性があります。セカンダリネットワークインターフェイス用に別のサブネットを指定すると、Pod のために使用可能な IPv4 アドレスの数を増やすことができます。

- セキュリティ上の理由からも、ノードのプライマリネットワークインターフェイスで使用するものとは異なるセキュリティグループまたはサブネットが、Pod に必要となる場合があります。
- ノードをパブリックサブネット内に構成した場合、Pod はプライベートサブネット内に配置します。パブリックサブネットに関連付けられているルートテーブルには、インターネットゲートウェイへのルートが含まれています。プライベートサブネットに関連付けられているルートテーブルには、インターネットゲートウェイへのルートは含まれません。

> **Tip**
>
> また、カスタムネットワーキングを使用することなく Amazon EKS クラスターに直接新規または既存のサブネットを追加することもできます。詳細については、「the section called “管理コンソールから既存の VPC サブネットを Amazon EKS クラスターに追加する”」を参照してください。

考慮事項

この機能を使用する際の考慮事項を次に示します。

- カスタムネットワーキングを有効にすると、プライマリネットワークインターフェイスに割り当てられた IP アドレスが、Pod に割り当てられることはなくなります。Pod には、セカンダリネットワークインターフェイスからの IP アドレスのみが割り当てられます。
- クラスターで IPv6 ファミリーを使用している場合は、カスタムネットワーキングを使用することはできません。
- IPv4 アドレスの枯渇を軽減するために、カスタムネットワーキングを使用する予定の場合は、IPv6 ファミリーを使用してクラスターを作成することもできます。詳細については、「the section called “IPv6”」を参照してください。
- サブネットにデプロイされたセカンダリネットワークインターフェイス用の Pod が、ノードのプライマリネットワークインターフェイスとは異なるサブネットおよびセキュリティグループを使用できるように設定されていても、そのサブネットとセキュリティグループは、ノードと同じ VPC 内に配置される必要があります。
- Fargate の場合、サブネットは Fargate プロファイルを介して制御されます。詳細については、「the section called “プロファイルを定義する”」を参照してください。

Amazon EKS ノードのセカンダリネットワークインターフェイスをカスタマイズする

このチュートリアルを開始する前に以下を完了してください。

- 考慮事項を確認する
- Amazon VPC CNI plugin for Kubernetes によるセカンダリネットワークインターフェイスの作成と、Pod への IP アドレス割り当て方法に関する知識。詳細については、GitHub の「ENI Allocation」(ENI 割り当て) を参照してください。
- ご使用のデバイスまたは AWS CloudShell で、バージョン `2.12.3` 以降、または AWS コマンドラインインターフェイス (AWS CLI) のバージョン `1.27.160` 以降がインストールおよび設定されていること。現在のバージョンを確認するには、`aws --version | cut -d / -f2 | cut -d ' ' -f1` を参照してください。`yum`、`apt-get`、macOS 用の Homebrew などのパッケージマネージャーは、多くの場合 AWS CLI の最新バージョンより数バージョン古くなっています。最新バージョンをインストールするには、 AWS コマンドラインインターフェイスユーザーガイドの「インストール」および「aws configure を使用したクイック設定」を参照してください。AWS CloudShell にインストールされている AWS CLI バージョンも最新バージョンより数バージョン遅れることがあります。更新するには、「AWS CloudShell ユーザーガイド」の「ホームディレクトリへの AWS CLI のインストール」を参照してください。
- デバイスまたは AWS CloudShell に、`kubectl` コマンドラインツールがインストールされていること。`kubectl` をインストールまたはアップグレードする方法については、the section called “kubectl と eksctl のセットアップ” を参照してください。
- このトピック内のステップはBash シェル内で実行することが推奨されます。Bash シェルを使用していない場合、行継続文字や、変数の設定と使用に関する方法など、一部のスクリプトコマンドのためにシェルの調整が必要となります。さらに、シェルの引用規則とエスケープ規則は異なる場合があります。詳細については「AWS コマンドラインインターフェイスのユーザーガイド」の「AWS CLI での文字列への引用符の使用」を参照してください。

このチュートリアルでは置き換えが必要だと明記されている場合を除き、サンプル値をそのまま使用することをお勧めします。本番向けクラスター向けにステップを完了する際には例値 を任意の値に置き換えることができます。すべてのステップは同一のターミナルで実行することをお勧めします。設定された変数はステップ全体で使用され、また、これらは異なるターミナルには存在しないためです。

このトピックのコマンドは「AWS CLI 使用例を使用する」に記載されている規則に従ってフォーマットされています。使用している AWS CLI プロファイルで定義されているデフォルトの AWS リージョンとは異なる AWS リージョンにあるリソースに対してコマンドラインからコマンドを実行

する場合はコマンドに --region us-west-2 を追加する必要があります。us-west-2 は自分の AWS リージョンに置き換えます。

カスタムネットワーキングを本番用クラスターにデプロイする場合は、the section called “ステップ 2: VPC を設定する” にスキップします。

ステップ 1: テスト VPC およびクラスターを作成する

次の手順により、テスト VPC とクラスターを作成し、そのクラスターのためにカスタムネットワークを構成できます。本番向けのクラスターで必要になるいくつかの機能で、このトピックに関連のないものについては説明していません。そのため、本番ワークロード向けに、このテストクラスターを使用することはお勧めしません。詳細については、the section called “クラスターを作成する” を参照してください。

1. 以下のコマンドを実行して、account_id 変数を定義します。

```
account_id=$(aws sts get-caller-identity --query Account --output text)
```

2. VPC を作成します。

   a. テストシステムにデプロイする場合は Amazon EKS AWS クラウドフォーメーション テンプレートを使用して VPC を作成します。

```
aws cloudformation create-stack --stack-name my-eks-custom-networking-vpc \
  --template-url https://s3.us-west-2.amazonaws.com/amazon-eks/
cloudformation/2020-10-29/amazon-eks-vpc-private-subnets.yaml \
  --parameters ParameterKey=VpcBlock,ParameterValue=192.168.0.0/24 \
  ParameterKey=PrivateSubnet01Block,ParameterValue=192.168.0.64/27 \
  ParameterKey=PrivateSubnet02Block,ParameterValue=192.168.0.96/27 \
  ParameterKey=PublicSubnet01Block,ParameterValue=192.168.0.0/27 \
  ParameterKey=PublicSubnet02Block,ParameterValue=192.168.0.32/27
```

   b. AWS クラウドFormation スタックが作成されるまで数分かかります。次のコマンドを実行して、スタックのデプロイステータスを確認します。

```
aws cloudformation describe-stacks --stack-name my-eks-custom-networking-vpc --
query Stacks\[\].StackStatus  --output text
```

      コマンドの出力が「CREATE_COMPLETE」になるまで、次のステップに進まないでください。

   c. テンプレートによって作成された、プライベートサブネット ID の値により変数を定義します。

```
subnet_id_1=$(aws cloudformation describe-stack-resources --stack-name my-eks-
custom-networking-vpc \
    --query "StackResources[?
LogicalResourceId=='PrivateSubnet01'].PhysicalResourceId" --output text)
subnet_id_2=$(aws cloudformation describe-stack-resources --stack-name my-eks-
custom-networking-vpc \
    --query "StackResources[?
LogicalResourceId=='PrivateSubnet02'].PhysicalResourceId" --output text)
```

d. 前のステップで取得した、サブネットのアベイラビリティーゾーンを使用して変数を定義します。

```
az_1=$(aws ec2 describe-subnets --subnet-ids $subnet_id_1 --query
 'Subnets[*].AvailabilityZone' --output text)
az_2=$(aws ec2 describe-subnets --subnet-ids $subnet_id_2 --query
 'Subnets[*].AvailabilityZone' --output text)
```

3. クラスターの IAM ロールを作成します。

a. IAM 信頼ポリシー用の JSON ファイルを作成するには次のコマンドを実行してください。

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "eks.amazonaws.com"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
```

b. Amazon EKS クラスター の IAM ロールを作成します。必要に応じて、`eks-cluster-role-trust-policy.json` の前に、ステップでファイルの書き込み先となったコンピュータ上のパスを追加します。このコマンドは前のステップで作成した信頼ポリシーをロールに関連付けます。IAM ロールを作成するには、ロールを作成する IAM プリンシパル に `iam:CreateRole` アクション (許可) を割り当てる必要があります。

```
aws iam create-role --role-name myCustomNetworkingAmazonEKSClusterRole --assume-
role-policy-document file://"eks-cluster-role-trust-policy.json"
```

c. AmazonEKSClusterPolicy という名前の Amazon EKS マネージド型ポリシーをロールにアタッチします。IAM ポリシーを IAM プリンシパルにアタッチするにはポリシーのアタッチを行っているプリンシパルに、次のいずれかの IAM アクション (許可) を割り当てる必要があります: `iam:AttachUserPolicy` または `iam:AttachRolePolicy`

```
aws iam attach-role-policy --policy-arn arn:aws:iam::aws:policy/
AmazonEKSClusterPolicy --role-name myCustomNetworkingAmazonEKSClusterRole
```

4. Amazon EKS クラスターを作成し、このクラスターとデバイスの間の通信を設定します。

a. クラスターを作成する。

```
aws eks create-cluster --name my-custom-networking-cluster \
   --role-arn arn:aws:iam::$account_id:role/myCustomNetworkingAmazonEKSClusterRole
 \
   --resources-vpc-config subnetIds="$subnet_id_1","$subnet_id_2"
```

> **Note**
>
> リクエストで指定したアベイラビリティーゾーンのいずれかに、Amazon EKS クラスターの作成に十分な容量がない場合にはエラーが表示されることがあります。このエラー出力には新しいクラスターをサポートできるアベイラビリティーゾーンが表示されます。アカウント向けにサポートされているアベイラビリティーゾーンにある 2 つ以上のサブネットを使用して、クラスターを作成します。詳細については、the section called “容量不足” を参照してください。

b. クラスターが作成されるまでに数分かかります。次のコマンドを実行して、クラスターのデプロイステータスを確認します。

```
aws eks describe-cluster --name my-custom-networking-cluster --query
 cluster.status
```

コマンドの出力が「`"ACTIVE"`」になるまで、次のステップに進まないでください。

c. クラスターと通信するために `kubectl` を設定します。

```
aws eks update-kubeconfig --name my-custom-networking-cluster
```

ステップ 2: VPC を設定する

このチュートリアルでは、the section called “ステップ 1: テスト VPC およびクラスターを作成する” で作成した VPC が必要です。本番向けクラスターの場合は、example values をすべて実際の値に置き換えて、VPC に合うように各ステップを調整します。

1. 現在インストールされている Amazon VPC CNI plugin for Kubernetes が最新バージョンであることを確認します。Amazon EKS アドオンタイプの最新バージョンを確認し、そのバージョンに更新するには「the section called “アドオンを更新する”」を参照してください。セルフマネージドアドオンタイプの最新バージョンを確認し、そのバージョンに更新するには「the section called “Amazon VPC CNI”」を参照してください。
2. クラスター VPC の ID を取得し、後のステップで使用するために変数に格納します。

```
vpc_id=$(aws eks describe-cluster --name my-custom-networking-cluster --query
 "cluster.resourcesVpcConfig.vpcId" --output text)
```

3. 追加の Classless Inter-Domain Routing (CIDR ブロックを、クラスターの VPC に関連付けます。CIDR ブロックは既存の関連付けられた CIDR ブロックと重複することはできません。

   a. 現在 VPC に関連付けられている CIDR ブロックを表示します。

```
aws ec2 describe-vpcs --vpc-ids $vpc_id \
    --query 'Vpcs[*].CidrBlockAssociationSet[*].{CIDRBlock: CidrBlock, State:
 CidrBlockState.State}' --out table
```

   出力例は次のとおりです。

```
---------------------------------
|          DescribeVpcs         |
+-----------------+-------------+
|    CIDRBlock    |    State    |
+-----------------+-------------+
|  192.168.0.0/24 |  associated |
+-----------------+-------------+
```

b. 追加の CIDR ブロックを VPC に関連付けます。次のコマンドで CIDR ブロック値を置き換えます。詳細については「Amazon VPC ユーザーガイド」の「追加の IPv4 CIDR ブロックと VPC の関連付け」を参照してください。

```
aws ec2 associate-vpc-cidr-block --vpc-id $vpc_id --cidr-block 192.168.1.0/24
```

c. 新しいブロックが関連付けられていることを確認します。

```
aws ec2 describe-vpcs --vpc-ids $vpc_id --query
 'Vpcs[*].CidrBlockAssociationSet[*].{CIDRBlock: CidrBlock, State:
 CidrBlockState.State}' --out table
```

出力例は次のとおりです。

```
----------------------------------
|          DescribeVpcs          |
+-----------------+--------------+
|    CIDRBlock    |    State     |
+-----------------+--------------+
|  192.168.0.0/24 |  associated  |
|  192.168.1.0/24 |  associated  |
+-----------------+--------------+
```

新しい CIDR ブロックの State が associated に遷移するまで、次のステップには進まないでください。

4. 既存のサブネットが存在する各アベイラビリティーゾーン内で、使用するサブネットを必要な数だけ作成します。前のステップで VPC に関連付けた、CIDR ブロック内にある CIDR ブロックを指定します。

a. 新しいサブネットを作成します。次のコマンドで CIDR ブロック値を置き換えます。サブネットは既存のサブネットが置かれているものとは異なる VPC CIDR ブロックの中に作成する必要があり、作成するアベイラビリティーゾーンは既存のサブネットと同じにする必要があります。この例では現在のプライベートサブネットが存在する各アベイラビリティーゾーンにある新しい CIDR ブロックの中に、1 つのサブネットが作成されます。作成されたサブネットの ID は後のステップで使用するために変数に格納されます。Name の値は前のステップで Amazon EKS VPC テンプレートを使用して作成されたサブネットに、割り当てられている値と一致します。名前は必須ではありません。名前には異なる値を使用できます。

```
new_subnet_id_1=$(aws ec2 create-subnet --vpc-id $vpc_id --availability-zone $az_1
 --cidr-block 192.168.1.0/27 \
    --tag-specifications 'ResourceType=subnet,Tags=[{Key=Name,Value=my-eks-custom-
networking-vpc-PrivateSubnet01},{Key=kubernetes.io/role/internal-elb,Value=1}]' \
    --query Subnet.SubnetId --output text)
new_subnet_id_2=$(aws ec2 create-subnet --vpc-id $vpc_id --availability-zone $az_2
 --cidr-block 192.168.1.32/27 \
    --tag-specifications 'ResourceType=subnet,Tags=[{Key=Name,Value=my-eks-custom-
networking-vpc-PrivateSubnet02},{Key=kubernetes.io/role/internal-elb,Value=1}]' \
    --query Subnet.SubnetId --output text)
```

Important

デフォルトで新しいサブネットはVPC のメインルートテーブルに暗黙的に関連付けられます。このルートテーブルにより、その VPC にデプロイされているすべてのリソース間の通信が可能になります。ただし、VPC に関連付けられた CIDR ブロックの外部にある IP アドレスが割り当てられたリソースとの通信は許可されません。この動作を変更するには独自のルートテーブルをサブネットに関連付けます。詳細については「Amazon VPC ユーザーガイド」の「サブネットルートテーブル」を参照してください。

b. 使用している VPC の現在のサブネットを表示します。

```
aws ec2 describe-subnets --filters "Name=vpc-id,Values=$vpc_id" \
    --query 'Subnets[*].{SubnetId: SubnetId,AvailabilityZone:
 AvailabilityZone,CidrBlock: CidrBlock}' \
    --output table
```

出力例は次のとおりです。

```
-------------------------------------------------------------------
|                          DescribeSubnets                        |
+------------------+--------------------+---------------------------+
| AvailabilityZone |     CidrBlock      |         SubnetId          |
+------------------+--------------------+---------------------------+
|  us-west-2d      |  192.168.0.0/27    |     subnet-example1       |
|  us-west-2a      |  192.168.0.32/27   |     subnet-example2       |
|  us-west-2a      |  192.168.0.64/27   |     subnet-example3       |
|  us-west-2d      |  192.168.0.96/27   |     subnet-example4       |
```

```
|  us-west-2a      |  192.168.1.0/27    |     subnet-example5        |
|  us-west-2d      |  192.168.1.32/27   |     subnet-example6        |
+------------------+--------------------+----------------------------+
```

作成した 192.168.1.0 CIDR ブロック内のサブネットが、192.168.0.0 CIDR ブロック内のサブネットと同じアベイラビリティーゾーンに置かれていることが確認できます。

ステップ 3: Kubernetes リソースを設定する

1. aws-node DaemonSet で AWS_VPC_K8S_CNI_CUSTOM_NETWORK_CFG 環境変数を true に設定します。

```
kubectl set env daemonset aws-node -n kube-system
 AWS_VPC_K8S_CNI_CUSTOM_NETWORK_CFG=true
```

2. クラスターのセキュリティグループ の ID を取得し、次のステップで使用するために変数に格納します。このセキュリティグループはクラスターの作成時に Amazon EKS により自動的に作成されます。

```
cluster_security_group_id=$(aws eks describe-cluster --name my-custom-networking-
cluster --query cluster.resourcesVpcConfig.clusterSecurityGroupId --output text)
```

3. Pod をデプロイするサブネットごとに ENIConfig カスタムリソースを作成します。

   a. 各ネットワークインターフェイス設定に固有のファイルを作成します。

   次のコマンドでは、前のステップで作成した 2 つのサブネットのそれぞれに、個別の ENIConfig ファイルが作成されます。name の値は一意である必要があります。これらの名前はサブネットが存在するアベイラビリティーゾーンと同じものが付けられます。クラスターセキュリティグループは、ENIConfig に割り当てられます。

```
cat >$az_1.yaml <<EOF
apiVersion: crd.k8s.amazonaws.com/v1alpha1
kind: ENIConfig
metadata:
  name: $az_1
spec:
  securityGroups:
    - $cluster_security_group_id
  subnet: $new_subnet_id_1
```

```
EOF
```

```
cat >$az_2.yaml <<EOF
apiVersion: crd.k8s.amazonaws.com/v1alpha1
kind: ENIConfig
metadata:
  name: $az_2
spec:
  securityGroups:
    - $cluster_security_group_id
  subnet: $new_subnet_id_2
EOF
```

本番向けのクラスターでは、前のコマンドに次の変更を行います。

- $cluster_security_group_id は各 ENIConfig で使用する既存のセキュリティグループの ID に置き換えます。
- 可能な限り、ENIConfig が使用されるアベイラビリティーゾーンと同じ名前を、ENIConfigs に付けることをお勧めします。いくつかの理由から、ENIConfigs に対して、アベイラビリティーゾーンとは異なる名前を使用する必要が生じることがあります。例えば、同じアベイラビリティーゾーンに 3 つ以上のサブネットがあり、そのすべてでカスタムネットワーキングを使用したい場合は同一のアベイラビリティーゾーンに複数の ENIConfigs が必要になります。各 ENIConfig には一意の名前が必要であるため、複数の ENIConfigs に対しアベイラビリティーゾーンの名前を使用することはできません。

  ENIConfig の中で、アベイラビリティーゾーンと異なる名前を持つものについては前のコマンドで $az_1 と $az_2 を独自の名前に置き換えます。さらに、この後のチュートリアルで ENIConfig でノードをアノテーションします。

  > **Note**
  >
  > 本番向けクラスターで使用するために有効なセキュリティグループを指定しておらず、かつ:

- Amazon VPC CNI plugin for Kubernetes のバージョン 1.8.0 以降を使用する場合は、ノードのプライマリ Elastic Network Interface に関連付けられたセキュリティグループが使用されます。

- 1.8.0 より前のバージョンの Amazon VPC CNI plugin for Kubernetes を使用している場合は、VPC のデフォルトのセキュリティグループが、セカンダリネットワークインターフェイスに割り当てられます。

> **Important**
>
> - AWS_VPC_K8S_CNI_EXTERNALSNAT=false は、Kubernetes 用 Amazon VPC CNI プラグインのデフォルトの設定です。デフォルト設定を使用している場合、VPC に関連付けられた CIDR ブロックのいずれにもない IP アドレスに向けて送信されたトラフィックにはノードのプライマリネットワークインターフェイスでの、セキュリティグループとサブネットが使用されます。セカンダリネットワークインターフェイスを作成するために使用された、ENIConfigs で定義されているサブネットとセキュリティグループはこのトラフィックでは使用されません。この設定の詳細については、the section called “アウトバウンドトラフィック” を参照してください。
> - また、Pod でもセキュリティグループを使用する場合、そのセキュリティグループは SecurityGroupPolicy で指定されたものとなり、ENIConfigs で指定されたものは使用されません。詳細については、the section called “ポッドのセキュリティグループ” を参照してください。

b. 以下のコマンドを使用して、作成したカスタムリソースファイルをそれぞれクラスターに適用します。

```
kubectl apply -f $az_1.yaml
kubectl apply -f $az_2.yaml
```

4. ENIConfigs が作成されたことを確認します。

```
kubectl get ENIConfigs
```

出力例は次のとおりです。

```
NAME         AGE
us-west-2a   117s
us-west-2d   105s
```

5. 本番向けのクラスターで、カスタムネットワーキングを有効にしており、ENIConfigs の名前を使用されているアベイラビリティーゾーンとは異なるものにした場合は次のステップに移り、Amazon EC2 ノードをデプロイします。

クラスターで作成された任意の新しい Amazon EC2 ノードに対し、Kubernetes がアベイラビリティーゾーン用の ENIConfig を自動的に適用することを許可します。

a. このチュートリアルでのテスト用クラスターの場合は、このまま 次のステップ に移動します。

実稼働クラスターの場合は、 ENI_CONFIG_ANNOTATION_DEF 環境変数用のキー aws-node を使用するアノテーションが、k8s.amazonaws.com/eniConfig DaemonSet のためのコンテナ仕様内に存在することを確認します。

```
kubectl describe daemonset aws-node -n kube-system | grep
 ENI_CONFIG_ANNOTATION_DEF
```

出力が返される場合はアノテーションが存在します。出力が返されない場合、その変数は設定されていません。本番向けクラスターの場合にはここでの設定と、以降のステップで示す設定のどちらも使用可能です。ここでの設定を使用すると、以降のステップの設定は上書きされます。このチュートリアルではこの後のステップでの設定を使用します。

b. クラスター内に作成された任意の新しい Amazon EC2 ノードに対し、自動的にアベイラビリティーゾーンの ENIConfig を適用するように、aws-node DaemonSet を更新します。

```
kubectl set env daemonset aws-node -n kube-system
 ENI_CONFIG_LABEL_DEF=topology.kubernetes.io/zone
```

ステップ 4: Amazon EC2 ノードをデプロイする

1. ノードの IAM ロールを作成します。

a. IAM 信頼ポリシー用の JSON ファイルを作成するには次のコマンドを実行してください。

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "ec2.amazonaws.com"
      },
      "Action": "sts:AssumeRole"
    }
  ]
```

```
}
```

b. IAM ロールを作成し、返された Amazon リソースネーム (ARN) を、後のステップで使用するために変数に格納します。

```
node_role_arn=$(aws iam create-role --role-name myCustomNetworkingNodeRole --
assume-role-policy-document file://"node-role-trust-relationship.json" \
    --query Role.Arn --output text)
```

c. IAM ロールに、3 つの必須な IAM マネージドポリシーをアタッチします。

```
aws iam attach-role-policy \
  --policy-arn arn:aws:iam::aws:policy/AmazonEKSWorkerNodePolicy \
  --role-name myCustomNetworkingNodeRole
aws iam attach-role-policy \
  --policy-arn arn:aws:iam::aws:policy/AmazonEC2ContainerRegistryReadOnly \
  --role-name myCustomNetworkingNodeRole
aws iam attach-role-policy \
    --policy-arn arn:aws:iam::aws:policy/AmazonEKS_CNI_Policy \
    --role-name myCustomNetworkingNodeRole
```

> **Important**
>
> 分かりやすくするため、このチュートリアルではAmazonEKS_CNI_Policy ポリシーはノードの IAM ロールにアタッチされています。ただし、本番向けクラスターの場合は、Amazon VPC CNI plugin for Kubernetes のみで使用される IAM ロールに、個別にポリシーをアタッチすることをお勧めします。詳細については、the section called “IRSA 用の設定” を参照してください。

2. 次のいずれかのタイプのノードグループを作成します。デプロイするインスタンスタイプを確認するには、the section called “Amazon EC2 インスタンスタイプ” を参照してください。このチュートリアルのために、マネージド型、および 起動テンプレートを使用しない、または AMI ID が指定されていない起動テンプレートを使用する オプションを完了します。本番ワークロードでノードグループを使用する場合はそのグループをデプロイする前に、マネージドノードグループおよびセルフマネージドノードグループのすべてのオプションについて理解しておくことをお勧めします。

   - マネージド型 — 次のいずれかのオプションを使用して、ノードグループをデプロイします：
     - 起動テンプレートを使用しない、または AMI ID が指定されていない起動テンプレートを使用する — 次のコマンドを実行してください。このチュートリアルではサンプルの値を使用しま

す。本番向けのノードグループの場合はすべてのサンプルの値を、実際の値に置き換えてください。ノードグループ名は 63 文字以下である必要があります。先頭は文字または数字でなければなりませんが、残りの文字にはハイフンおよびアンダースコアを含めることもできます。

```
aws eks create-nodegroup --cluster-name my-custom-networking-cluster --nodegroup-
name my-nodegroup \
    --subnets $subnet_id_1 $subnet_id_2 --instance-types t3.medium --node-role
 $node_role_arn
```

- 指定された AMI ID を持つ起動テンプレートを使用::

  A. インスタンスタイプに基づいて、ノードの最大 Pod 数を決定します。詳細については、the section called “maxPods の決定方法” を参照してください。次のステップで使用するために、値をメモします。

  B. 起動テンプレートで、Amazon EKS 最適化 AMI ID を指定するか、Amazon EKS 最適化 AMI から構築されたカスタム AMI を指定します。その後、起動テンプレートを使用してノードグループをデプロイし、さらに、起動テンプレートにある次のユーザーデータを指定します。このユーザーデータは、引数を NodeConfig 仕様に渡します。NodeConfig の詳細については、「NodeConfig API リファレンス」を参照してください。20 は、前のステップで使用した値 (推奨) か、または独自の値に置き換えることができます。

  ```
  ---
  MIME-Version: 1.0
  Content-Type: multipart/mixed; boundary="BOUNDARY"
  --BOUNDARY
  Content-Type: application/node.eks.aws

  ---
  apiVersion: node.eks.aws/v1alpha1
  kind: NodeConfig
  spec:
    cluster:
      name: my-cluster
      ...
      kubelet:
        config:
          maxPods: 20
  ```

  Amazon EKS 最適化 AMI から構築されていないカスタム AMI を作成した場合は自分で設定をカスタム作成する必要があります。

- セルフマネージド型::

  i. インスタンスタイプに基づいて、ノードの最大 Pod 数を決定します。詳細については、the section called “maxPods の決定方法” を参照してください。次のステップで使用するために、値をメモします。

  ii. the section called “Amazon Linux” の手順に従ってノードグループをデプロイします。

> **Note**
>
> 本番クラスター内のノードで、非常に大量の Pod をサポートさせる場合には、プレフィックス委任を有効にできます。例えば、m5.large インスタンスタイプの場合は 110 が返されます。この機能を有効化する方法については、the section called “IP アドレスを増やす” を参照してください。この機能はカスタムネットワーキングで使用できます。

3. ノードグループの作成には数分かかります。次のコマンドを使用すると、マネージド型のノードグループ作成のステータスを確認できます。

```
aws eks describe-nodegroup --cluster-name my-custom-networking-cluster --nodegroup-
name my-nodegroup --query nodegroup.status --output text
```

出力が「ACTIVE」を返すまで、次のステップに進まないでください。

4. このチュートリアルではこのステップをスキップできます。

本番向けのクラスターにおいて、ENIConfigs の名前を、それが使用されているアベイラビリティーゾーンと同じにしなかった場合にはノードに対し、そのノードで使用すべき ENIConfig の名前をアノテーションします。各アベイラビリティーゾーンにサブネットが 1 つしかなく、ENIConfigs にアベイラビリティーゾーンと同じ名前を付けている場合にはこのステップは必要ありません。前のステップでその設定を有効にしている場合、Amazon VPC CNI plugin for Kubernetes が適切な ENIConfig をノードに対し自動的に関連付けるためです。

a. クラスター内のノードのリストを取得します。

```
kubectl get nodes
```

出力例は次のとおりです。

```
NAME                                           STATUS   ROLES    AGE     VERSION
ip-192-168-0-126.us-west-2.compute.internal    Ready    <none>   8m49s   v1.22.9-
eks-810597c
```

```
ip-192-168-0-92.us-west-2.compute.internal    Ready    <none>   8m34s   v1.22.9-
eks-810597c
```

b. 各ノードが属するアベイラビリティーゾーンを決定します。前の出力に基づいて IP アドレスを置き換えて、前のステップで返されたノードごとに次のコマンドを実行します。

```
aws ec2 describe-instances --filters Name=network-interface.private-dns-
name,Values=ip-192-168-0-126.us-west-2.compute.internal \
--query 'Reservations[].Instances[].{AvailabilityZone: Placement.AvailabilityZone,
 SubnetId: SubnetId}'
```

出力例は次のとおりです。

```
[
    {
        "AvailabilityZone": "us-west-2d",
        "SubnetId": "subnet-Example5"
    }
]
```

c. 各ノードを、サブネット ID とアベイラビリティーゾーンのために作成した ENIConfig でアノテーションします。各ノードのアノテーションに使用できる ENIConfig は 1 つだけです。ただし、複数のノードに対し同じ ENIConfig でアノテーションすることは可能です。値の例は、実際に使用する値に置き換えます。

```
kubectl annotate node ip-192-168-0-126.us-west-2.compute.internal
 k8s.amazonaws.com/eniConfig=EniConfigName1
kubectl annotate node ip-192-168-0-92.us-west-2.compute.internal
 k8s.amazonaws.com/eniConfig=EniConfigName2
```

5. カスタムネットワーキング機能の使用に切り替える前に、Pod を実行している本稼働クラスターにノードがあった場合は、以下のタスクを完了してください。

a. カスタムネットワーク機能を使用しているノードが、使用可能であることを確認します。

b. ノードを遮断およびドレインして、Pod をスムーズにシャットダウンします。詳細については、Kubernetes ドキュメントの「Safely Drain a Node」(ノードを安全にドレインする) を参照してください。

c. ノードを終了します。ノードが既存のマネージド型ノードグループ内にある場合はそのノードグループを削除できます。以下のコマンドを実行してください。

```
aws eks delete-nodegroup --cluster-name my-custom-networking-cluster --nodegroup-
name my-nodegroup
```

カスタムネットワーク機能はk8s.amazonaws.com/eniConfig ラベルで登録されている新しいノードでのみ使用されます。

6. Pod に、前の手順で作成したサブネットの 1 つに関連付けられた CIDR ブロックから、IP アドレスが割り当てられていることを確認します。

```
kubectl get pods -A -o wide
```

出力例は次のとおりです。

```
NAMESPACE     NAME                          READY   STATUS    RESTARTS   AGE     IP
           NODE                                            NOMINATED NODE   READINESS
 GATES
kube-system   aws-node-2rkn4                1/1     Running   0          7m19s
 192.168.0.92    ip-192-168-0-92.us-west-2.compute.internal    <none>
 <none>
kube-system   aws-node-k96wp                1/1     Running   0          7m15s
 192.168.0.126   ip-192-168-0-126.us-west-2.compute.internal   <none>
 <none>
kube-system   coredns-657694c6f4-smcgr      1/1     Running   0          56m
 192.168.1.23    ip-192-168-0-92.us-west-2.compute.internal    <none>
 <none>
kube-system   coredns-657694c6f4-stwv9      1/1     Running   0          56m
 192.168.1.28    ip-192-168-0-92.us-west-2.compute.internal    <none>
 <none>
kube-system   kube-proxy-jgshq              1/1     Running   0          7m19s
 192.168.0.92    ip-192-168-0-92.us-west-2.compute.internal    <none>
 <none>
kube-system   kube-proxy-wx9vk              1/1     Running   0          7m15s
 192.168.0.126   ip-192-168-0-126.us-west-2.compute.internal   <none>
 <none>
```

VPC に追加した 192.168.1.0 CIDR ブロックからの IP アドレスが、coredns Pod に割り当てられていることが確認できます。カスタムネットワーキングを使用していない場合はここに 192.168.0.0 CIDR ブロックからのアドレスが割り当てられています。この CIDR ブロックが、元々 VPC に関連付けられている唯一のブロックであるためです。

Pod の spec が hostNetwork=true を含む場合には、ノードのプライマリ IP アドレスが割り当てられます。追加したサブネットのアドレスは割り当てられません。デフォルトではこの値は false に設定されます。クラスターで実行される kube-proxy および Amazon VPC CNI plugin for Kubernetes (aws-node) の Pod に対して、この値は true に設定されます。これが、kube-proxy とプラグインの aws-node の Pod に、前の出力にある 192.168.1.x アドレスが割り当てられない理由です。Pod の hostNetwork 設定の詳細については、Kubernetes API リファレンスの「PodSpec v1 core」を参照してください。

ステップ 5: チュートリアルでのリソースを削除する

このチュートリアル完了後は作成したリソースを削除することをお勧めします。その後、ここでのステップを調整して、本番向けクラスターのカスタムネットワーキングを有効化することができます。

1. 作成したノードグループが完全にテスト向けである場合はそれを削除してください。

```
aws eks delete-nodegroup --cluster-name my-custom-networking-cluster --nodegroup-name
 my-nodegroup
```

2. AWS CLI から、クラスターが削除済みだと出力された場合でも、実際にはその削除プロセスが完了していない場合があります。削除のプロセスには数分かかります。次のコマンドを実行して、処理の完了を確認します。

```
aws eks describe-nodegroup --cluster-name my-custom-networking-cluster --nodegroup-
name my-nodegroup --query nodegroup.status --output text
```

次のように出力されるまで、次に進まないでください。

```
An error occurred (ResourceNotFoundException) when calling the DescribeNodegroup
 operation: No node group found for name: my-nodegroup.
```

3. 作成したノードグループが完全にテスト向けである場合はノードの IAM ロールを削除します。

a. ロールからポリシーをデタッチします。

```
aws iam detach-role-policy --role-name myCustomNetworkingNodeRole --policy-arn
 arn:aws:iam::aws:policy/AmazonEKSWorkerNodePolicy
aws iam detach-role-policy --role-name myCustomNetworkingNodeRole --policy-arn
 arn:aws:iam::aws:policy/AmazonEC2ContainerRegistryReadOnly
```

```
aws iam detach-role-policy --role-name myCustomNetworkingNodeRole --policy-arn
 arn:aws:iam::aws:policy/AmazonEKS_CNI_Policy
```

b. ロールを削除します。

```
aws iam delete-role --role-name myCustomNetworkingNodeRole
```

4. クラスターを削除します。

```
aws eks delete-cluster --name my-custom-networking-cluster
```

以下のコマンドを使用して、クラスターが削除されたことを確認します。

```
aws eks describe-cluster --name my-custom-networking-cluster --query cluster.status
 --output text
```

次のような出力が返された場合はクラスターが正常に削除されています。

```
An error occurred (ResourceNotFoundException) when calling the DescribeCluster
 operation: No cluster found for name: my-custom-networking-cluster.
```

5. クラスターの IAM ロールを削除します。

a. ロールからポリシーをデタッチします。

```
aws iam detach-role-policy --role-name myCustomNetworkingAmazonEKSClusterRole --
policy-arn arn:aws:iam::aws:policy/AmazonEKSClusterPolicy
```

b. ロールを削除します。

```
aws iam delete-role --role-name myCustomNetworkingAmazonEKSClusterRole
```

6. 前のステップで作成したサブネットを削除します。

```
aws ec2 delete-subnet --subnet-id $new_subnet_id_1
aws ec2 delete-subnet --subnet-id $new_subnet_id_2
```

7. 作成した VPC を削除します。

```
aws cloudformation delete-stack --stack-name my-eks-custom-networking-vpc
```

プレフィックスを使用して Amazon EKS ノードに割り当てる IP アドレスを増やす

適用対象: Amazon EC2 インスタンスを持つ Linux および Windows ノード

適用対象: パブリックサブネットおよびプライベートサブネット

各 Amazon EC2 インスタンスではエラスティックネットワークインターフェイス の最大数と、各ネットワークインターフェイスに割り当て可能な IP アドレスの最大数がサポートされています。各ノードにはネットワークインターフェースごとに 1 つの IP アドレスが必要です。その他の使用可能な IP アドレスはすべて Pods に割り当てることができます。Pod それぞれに固有の IP アドレスが必要です。その結果、使用可能なコンピューティングリソースとメモリリソースはあるが、Pods に割り当てる IP アドレスが不足しているために追加の Pods に対応できないノードが存在する可能性があります。

個別のセカンダリ IP アドレスではなく、IP プレフィックスをノードに割り当てて、ノードでPodsに割り当て可能な IP アドレスの数を増やす方法について説明します。各プレフィックスには複数の IP アドレスが含まれます。IP プレフィックスが割り当てられるようにクラスターを設定しない場合、クラスターは Pod 接続に必要なネットワークインターフェイスと IP アドレスを設定するために Amazon EC2 アプリケーションプログラミングインターフェイス (API) の呼び出しをさらに実行する必要があります。クラスターのサイズが大きくなるにつれて、これらの API コールの頻度が高くなるため、Pod とインスタンスの起動時間が長くなる場合があります。これにより、大規模でスパイク的なワークロードの需要を満たすためにスケーリングの遅延が発生します。また、スケーリング要件を満たすためには追加のクラスターと VPC をプロビジョニングする必要があるため、コストと管理オーバーヘッドも増加します。詳細については「GitHub」の「Kubernetes スケーラビリティ閾値」を参照してください。

Amazon VPC CNI plugin for Kubernetes 機能との互換性

次の機能で IP プレフィックスを使用できます：

- IPv4 ソースネットワークアドレス変換 - 詳細については「the section called “アウトバウンドトラフィック”」を参照してください。
- クラスター、ポッド、サービスへの IPv6 アドレス - 詳細については「the section called “IPv6”」を参照してください。
- Kubernetes ネットワークポリシーを使用したトラフィックの制限 - 詳細については、「the section called “Kubernetes ポリシー”」を参照してください。

次のリストは適用される Amazon VPC CNI プラグイン設定に関する情報を示しています。各設定の詳細については、GitHub の「amazon-vpc-cni-k8s」を参照してください。

- `WARM_IP_TARGET`
- `MINIMUM_IP_TARGET`
- `WARM_PREFIX_TARGET`

考慮事項

この機能を使用する場合は以下を考慮してください：

- 各 Amazon EC2 インスタンスタイプでは、Pod の最大数がサポートされています。マネージドノードグループが複数のインスタンスタイプで構成されている場合、クラスターのインスタンスで指定する Pod の最大数の内、最小の値がクラスター内のすべてのノードに適用されます。
- デフォルトでは1 つのノードで実行できる Pods の最大数は 110 ですが、この値は変更できます。数値を変更し、既存のマネージド型ノードグループがある場合、ノードグループの AMI または起動テンプレートを次に更新するときに、新しいノードで変更済みの値が適用されるようになります。
- IP アドレスの割り当てから IP プレフィックスの割り当てに移行する場合は既存のノードにローリング置換を実行するのではなく、新しいノードグループを作成して使用可能な IP アドレスの数を増やすことをお勧めします。IP アドレスと IP プレフィックスの両方が割り当てられているノードで Pod を実行すると、アドバタイズされた IP アドレスの容量に不一致が生じ、ノード上の将来のワークロードに影響する可能性があります。推奨される移行方法については、「Amazon EKS Best Practices Guide」の「Prefix Delegation mode for Linux」を参照してください。
- セキュリティグループのスコープはノードレベル - 詳細については「セキュリティグループ」を参照してください。
- ネットワークインターフェイスに割り当てられた IP プレフィックスはノードあたりの高い Pod 密度をサポートし、最適な起動時間になっています。
- IP プレフィックスと IP アドレスは標準の Amazon EC2 Elastic ネットワークインターフェイスに関連付けられています。特定のセキュリティグループを必要とする Pod には、ブランチネットワークインターフェイスのプライマリ IP アドレスが割り当てられます。IP アドレスを取得するポッド、または IP プレフィックスからの IP アドレスを、同じノード上のブランチネットワークインターフェイスを取得する Pod と混在させることができます。
- Linux ノードのみを含むクラスターの場合。
  - ネットワークインターフェイスにプレフィックスを割り当てるようにアドオンを設定すると、クラスター内のすべてのノードグループにあるすべてのノードを削除しない限り、Amazon VPC CNI plugin for Kubernetes アドオンを `1.9.0` (または `1.10.1`) より前のバージョンにダウングレードすることはできません。

- Pod のセキュリティグループ (POD_SECURITY_GROUP_ENFORCING_MODE=standard および AWS_VPC_K8S_CNI_EXTERNALSNAT=false に設定) も使用している場合、Pod が VPC の外部のエンドポイントと通信するときに、Pod に割り当てた任意のセキュリティグループではなく、ノードのセキュリティグループが使用されます。

  セキュリティグループ (POD_SECURITY_GROUP_ENFORCING_MODE=strict に設定) も使用している場合、Pods が VPC の外部のエンドポイントと通信するときに、Pod's セキュリティグループが使用されます。

Amazon EKS ノードで使用可能な IP アドレスを増やす

個別のセカンダリ IP アドレスではなく、IP プレフィックスをノードに割り当てて、ノードで Pod に割り当て可能な IP アドレスの数を増やす方法について説明します。

前提条件

- 既存のクラスターが必要です。デプロイするには、the section called "クラスターを作成する" を参照してください。
- Amazon EKS ノードが配置されているサブネットでは、/28 個の連続ブロック (IPv4 クラスターの場合)、または /80 個の Classless Inter-Domain Routing (CIDR) ブロック (IPv6 クラスターの場合) が十分な数として必要になります。IPv6 クラスターに含めることができるのは Linux ノードだけです。IP アドレスがサブネット CIDR 全体に分散している場合、IP プレフィックスを使用すると失敗する可能性があります。次の構成を推奨します。
  - サブネット CIDR 予約を使用すると、予約された範囲内の IP アドレスがまだ使用されている場合でも、解放時に IP アドレスが再割り当てされません。これにより、セグメンテーションなしでプレフィックスを割り当てることができます。
  - IP プレフィックスが割り当てられているワークロードの実行に特に使用される新しいサブネットを使用してください。IP プレフィックスを割り当てると、Windows と Linux 両方のワークロードを同じサブネットで実行できます。
- ノードに IP プレフィックスを割り当てるには、ノードが AWS Nitro ベースである必要があります。Nitro ベースではないインスタンスでは、引き続き個別のセカンダリ IP アドレスが割り当てられますが、Pod に割り当てることのできる IP アドレスの数は Nitro ベースのインスタンスよりも大幅に少なくなります。
- Linux ノードのみのクラスターの場合 – クラスターが IPv4 ファミリーで設定されている場合は、バージョン 1.9.0 以降の Amazon VPC CNI plugin for Kubernetes アドオンがインストールされている必要があります。現在のバージョンは、次のコマンドで確認できます。

```
kubectl describe daemonset aws-node --namespace kube-system | grep Image | cut -d "/"
 -f 2
```

クラスターが IPv6 ファミリーで設定されている場合は、バージョン 1.10.1 のアドオンがインストールされている必要があります。プラグインバージョンが必要なバージョンよりも古い場合は、更新する必要があります。詳細については、「Amazon VPC CNI を使用してポッドに IP を割り当てる」の更新セクションを参照してください。

- Windows ノードのみのクラスターの場合
  - クラスターで Windows サポートを有効にする必要があります。詳細については、the section called “Windows サポートを有効にする” を参照してください。

ノードに IP アドレスプレフィックスを割り当てる

ノードに IP アドレスプレフィックスを割り当てるようにクラスターを設定します。ノードのオペレーティングシステムに対応する手順を完了します。

Linux

1. パラメータを有効にして、Amazon VPC CNI DaemonSet のネットワークインターフェイスにプレフィックスを割り当てます。クラスターをデプロイすると、バージョン 1.10.1 以降の Amazon VPC CNI plugin for Kubernetes アドオンも同時にデプロイされます。IPv6 ファミリーを使用してクラスターを作成している場合、この設定はデフォルトで true に設定されています。IPv4 ファミリーを使用してクラスターを作成している場合、この設定はデフォルトで false に設定されています。

   ```
   kubectl set env daemonset aws-node -n kube-system ENABLE_PREFIX_DELEGATION=true
   ```

   Important

   サブネットに使用可能な IP アドレスがある場合でも、サブネットに使用可能な /28 個の連続したブロックがない場合には、Amazon VPC CNI plugin for Kubernetes ログに次のエラーが記述されます。

   ```
   InsufficientCidrBlocks: The specified subnet does not have enough free cidr
    blocks to satisfy the request
   ```

これは、サブネット全体に広がる既存のセカンダリ IP アドレスの断片化が原因で発生する可能性があります。このエラーを解決するには、新しいサブネットを作成してそこに Pod を起動するか、Amazon EC2 サブネットの CIDR 予約を使用して、プレフィックス割り当てで使用するサブネット内の領域を予約します。詳細については、Amazon VPC ユーザーガイドの「サブネット CIDR の予約」を参照してください。

2. 前提条件のリストに記載されたバージョン以降の Amazon VPC CNI plugin for Kubernetes を使用しながら、起動テンプレートなしでマネージドノードグループをデプロイする場合、または AMI ID を指定していない起動テンプレートを使用してデプロイする場合には、このまま次のステップに進みます。マネージド型ノードグループは、Pod の最大数を自動的に計算します。

   セルフマネージド型ノードグループ、または AMI ID を指定した起動テンプレートを使用したマネージド型ノードグループをデプロイする場合は、ノードの最大 Pod 数を設定する必要があります。適切な値を決定する方法の詳細については、the section called “maxPods の決定方法” を参照してください。

   Important

   マネージド型ノードグループは、maxPods の値に最大数を適用します。vCPUs が 30 未満のインスタンスの場合、最大数は 110 で、その他すべてのインスタンスの最大数は 250 です。この最大数は、プレフィックス委任が有効かどうかにかかわらず適用されます。

3. IPv6 用に設定されたクラスターを使用している場合は、このまま次のステップに進みます。

   以下のオプションの内の 1 つでパラメータを指定します。どのオプションが適切で、どの値を提供するかを決定するには、GitHub の「WARM_PREFIX_TARGET, WARM_IP_TARGET, and MINIMUM_IP_TARGET」を参照してください。

   example values は、ゼロより大きい値で置き換えます。

   - WARM_PREFIX_TARGET

     ```
     kubectl set env ds aws-node -n kube-system WARM_PREFIX_TARGET=1
     ```

   - WARM_IP_TARGET または MINIMUM_IP_TARGET – この値を設定した場合、WARM_PREFIX_TARGET に対し設定されている値はすべて上書きされます。

     ```
     kubectl set env ds aws-node -n kube-system WARM_IP_TARGET=5
     ```

```
kubectl set env ds aws-node -n kube-system MINIMUM_IP_TARGET=2
```

4. 少なくとも 1 つの Amazon EC2 Nitro Amazon Linux 2023 インスタンスタイプを使用して、次のいずれかのタイプのノードグループを作成します。Nitro インスタンスタイプのリストについては、「Amazon EC2 ユーザーガイド」の「Instances built on the Nitro System」を参照してください。この機能は Windows ではサポートされていません。*110* が含まれているオプションの場合は、ステップ 3 の値 (推奨) または独自の値に置き換えてください。

   - セルフマネージド – 「セルフマネージド Amazon Linux ノードを作成する」の手順に従い、ノードグループをデプロイします。CloudFormation スタックを作成する前に、テンプレートファイルを開き、次のように NodeLaunchTemplate の UserData を調整します。

     ```
     ...
                 apiVersion: node.eks.aws/v1alpha1
                 kind: NodeConfig
                 spec:
                   cluster:
                     name: ${ClusterName}
                     apiServerEndpoint: ${ApiServerEndpoint}
                     certificateAuthority: ${CertificateAuthorityData}
                     cidr: ${ServiceCidr}
                   kubelet:
                     config:
                       maxPods: 110
     ...
     ```

     eksctl を使用してノードグループを作成する場合は、次のコマンドを使用できます。

     ```
     eksctl create nodegroup --cluster my-cluster --managed=false --max-pods-per-node
      110
     ```

   - マネージド型 — 次のいずれかのオプションを使用して、ノードグループをデプロイします：

     - 起動テンプレートがない場合、または AMI ID が指定されていない起動テンプレートを使用する場合 – 「クラスターのマネージドノードグループを作成する」の手順を完了します。マネージド型ノードグループは、Amazon EKS で推奨される max-pods の値を自動的に計算します。
     - 指定した AMI ID を持つ起動テンプレート — 起動テンプレートで Amazon EKS 最適化 AMI ID を指定するか、Amazon EKS 最適化 AMI から構築されたカスタム AMI を指定してから、起動テンプレートを使用してノードグループをデプロイ 起動テンプレートでは、次の

ユーザーデータを指定します。このユーザーデータが NodeConfig オブジェクトを渡してノードの nodeadm ツールによって読み取られます。nodeadm の詳細については、nodeadm のドキュメントを参照してください。

```
MIME-Version: 1.0
Content-Type: multipart/mixed; boundary="//"

--//
Content-Type: application/node.eks.aws

---
apiVersion: node.eks.aws/v1alpha1
kind: NodeConfig
spec:
 cluster:
   apiServerEndpoint: my-cluster
   certificateAuthority: LS0t...
   cidr: 10.100.0.0/16
   name: my-cluster
 kubelet:
   config:
     maxPods: 110
--//--
```

eksctl を使用してノードグループを作成する場合は、次のコマンドを使用できます。

```
eksctl create nodegroup --cluster my-cluster --max-pods-per-node 110
```

Amazon EKS 最適化 AMI から構築されていないカスタム AMI を作成した場合は、自分で設定をカスタム作成する必要があります。

> **Note**
>
> また、インスタンスとは異なるサブネットの Pod に IP アドレスを割り当てる場合は、このステップで機能を有効化する必要があります。詳細については、the section called “カスタムネットワーキング” を参照してください。

Windows

1. IP プレフィックスの割り当てを有効にします。

   a. 編集する amazon-vpc-cni ConfigMap を開きます。

   ```
   kubectl edit configmap -n kube-system amazon-vpc-cni -o yaml
   ```

   b. data セクションに次の行を追加します。

   ```
     enable-windows-prefix-delegation: "true"
   ```

   c. ファイルを保存し、エディタを閉じます。

   d. ConfigMap に行が追加されたことを確認します。

   ```
   kubectl get configmap -n kube-system amazon-vpc-cni -o "jsonpath={.data.enable-
   windows-prefix-delegation}"
   ```

   返された出力が true でない場合は、エラーが発生した可能性があります。ステップをもう一度実行してみてください。

   > **Important**
   >
   > サブネットに使用可能な IP アドレスがある場合でも、サブネットに使用可能な /28 個の連続したブロックがない場合には、Amazon VPC CNI plugin for Kubernetes ログに次のエラーが記述されます。
   >
   > ```
   > InsufficientCidrBlocks: The specified subnet does not have enough free cidr
   >  blocks to satisfy the request
   > ```
   >
   > これは、サブネット全体に広がる既存のセカンダリ IP アドレスの断片化が原因で発生する可能性があります。このエラーを解決するには、新しいサブネットを作成してそこに Pod を起動するか、Amazon EC2 サブネットの CIDR 予約を使用して、プレフィックス割り当てで使用するサブネット内の領域を予約します。詳細については、Amazon VPC ユーザーガイドの「サブネット CIDR の予約」を参照してください。

2. (オプション) クラスターの事前スケーリング動作と動的スケーリング動作を制御するための追加設定を指定します。詳細については、「GitHub」の「Windows でのプレフィックス委任モードの設定オプション」を参照してください。

a. 編集する amazon-vpc-cni ConfigMap を開きます。

```
kubectl edit configmap -n kube-system amazon-vpc-cni -o yaml
```

b. サンプル値を 0 より大きい値に置き換え、必要なエントリを ConfigMap の data セクションに追加します。warm-ip-target または minimum-ip-target のいずれかに値を設定した場合、その値は warm-prefix-target に設定された値をオーバーライドします。

```
  warm-prefix-target: "1"
  warm-ip-target: "5"
  minimum-ip-target: "2"
```

c. ファイルを保存し、エディタを閉じます。

3. 少なくとも 1 つの Amazon EC2 Nitro インスタンスタイプで Windows ノードグループを作成します。Nitro インスタンスタイプのリストについては、「Amazon EC2 ユーザーガイド」の「Instances built on the Nitro System」を参照してください。デフォルトでは、1 つのノードにデプロイできる Pod の最大数は 110 です。この数値を増減させる場合は、ブートストラップ設定のユーザーデータに以下を指定します。*max-pods-quantity* を最大ポッド値に置き換えます。

```
-KubeletExtraArgs '--max-pods=max-pods-quantity'
```

マネージド型ノードグループをデプロイする場合は、この設定を起動テンプレートに追加する必要があります。詳細については、the section called “起動テンプレート” を参照してください。Windows ブートストラップスクリプトの設定パラメータの詳細については、「the section called “ブートストラップスクリプトの設定パラメータ”」を参照してください。

最大 Pod 数と使用可能な IP アドレスを決定する

1. ノードがデプロイされると、クラスター内のノードの表示が可能になります。

```
kubectl get nodes
```

出力例は次のとおりです。

```
NAME                                              STATUS     ROLES    AGE   VERSION
ip-192-168-22-103.region-code.compute.internal    Ready      <none>   19m   v1.XX.X-
eks-6b7464
```

```
ip-192-168-97-94.region-code.compute.internal    Ready      <none>   19m   v1.XX.X-
eks-6b7464
```

2. いずれかのノードを記述して、そのノードの max-pods 値と使用可能な IP アドレスの数を決定します。*192.168.30.193* を、前の出力で返されたいずれかのノードの名前の IPv4 アドレスに置き換えます。

```
kubectl describe node ip-192-168-30-193.region-code.compute.internal | grep 'pods\|
PrivateIPv4Address'
```

出力例は次のとおりです。

```
pods:                                  110
vpc.amazonaws.com/PrivateIPv4Address:  144
```

前の出力にある 110 は、*144* 個の IP アドレスが使用可能であるにもかかわらず、Kubernetes がノードにデプロイする Pod の最大数です。

セキュリティグループを個別の Pod に割り当てる

適用対象: Amazon EC2 インスタンスを持つ Linux ノード

適用対象: プライベートサブネット

Pod のセキュリティグループは、Amazon EC2 セキュリティグループを Kubernetes Pod と統合します。Amazon EC2 セキュリティグループを使用して、多くの Amazon EC2 インスタンスタイプと Fargate で実行されているノードにデプロイする Pod との間で送受信されるインバウンドおよびアウトバウンドネットワークトラフィックを許可するルールを定義できます。この機能の詳細な説明については「ポッドのセキュリティグループの紹介」の投稿を参照してください。

Amazon VPC CNI plugin for Kubernetes 機能との互換性

Pod のセキュリティグループは、以下の機能で使用できます。

- IPv4 ソースネットワークアドレス変換 - 詳細については「the section called “アウトバウンドトラフィック”」を参照してください。
- クラスター、ポッド、サービスへの IPv6 アドレス - 詳細については「the section called “IPv6”」を参照してください。

- Kubernetes ネットワークポリシーを使用したトラフィックの制限 - 詳細については、「the section called “Kubernetes ポリシー”」を参照してください。

考慮事項

Pod のセキュリティグループをデプロイする前に、次の制限事項と条件を考慮してください。

- Pod のセキュリティグループは、Windows ノードまたは EKS Auto Mode では使用できません。
- Pod のセキュリティグループは、バージョン 1.16.0 以降の Amazon VPC CNI プラグインを使用して、Amazon EC2 ノードを含む IPv6 ファミリー用に設定されたクラスターで使用できます。バージョン 1.7.7 以降の Amazon VPC CNI プラグインを使用すると、Fargate ノードのみを含む IPv6 ファミリー用に設定されたクラスターで Pod のセキュリティグループを使用できます。詳細については、「the section called “IPv6”」を参照してください。
- Pod のセキュリティグループはほとんどの Nitro ベースの Amazon EC2 インスタンスファミリーでサポートされていますが、ファミリーの全世代がサポートしているわけではありません。例えば、m5、c5、r5、m6g、、c6g、r6g インスタンスファミリーと世代ではサポートされています。t ファミリーのインスタンスタイプは一切サポートされていません。サポートされているインスタンスタイプの詳細なリストについては、GitHub で limits.go ファイルを参照してください。ノードはそのファイルで一覧表示されているインスタンスタイプのうち、IsTrunkingCompatible: true を含むものである必要があります。
- カスタムネットワークと Pod のセキュリティグループを組み合わせて使用している場合、ENIConfig で指定されたセキュリティグループではなく、Pod のセキュリティグループによって指定されたセキュリティグループが使用されます。
- Amazon VPC CNI プラグインのバージョン 1.10.2 以前を使用していて、Pod の仕様に terminationGracePeriodSeconds 設定を含める場合は、設定の値を「0」にすることはできません。
- Amazon VPC CNI プラグインのバージョン 1.10 以前、またはデフォルト設定が POD_SECURITY_GROUP_ENFORCING_MODE=strict のバージョン 1.11 を使用している場合、externalTrafficPolicy が Local に設定されたインスタンスターゲットを使用するタイプ NodePort および LoadBalancer の Kubernetes サービスは、セキュリティグループを割り当てる Pod ではサポートされていません。インスタンスターゲットでのロードバランサーの使用の詳細については「the section called “ネットワーク負荷分散”」を参照してください。
- Amazon VPC CNI プラグインのバージョン 1.10 以前、またはデフォルト設定が POD_SECURITY_GROUP_ENFORCING_MODE=strict のバージョン 1.11 を使用している場合、セキュリティグループが割り当てられた Pod からのアウトバウンドトラフィックに対してソース NAT が無効になり、その結果アウトバウンドセキュリティグループルールが適用されます。イン

ターネットにアクセスするには、セキュリティグループが割り当てられた Pod を、NAT ゲートウェイまたはインスタンスで構成されたプライベートサブネットにデプロイされたノードで起動する必要があります。パブリックサブネットにデプロイされたセキュリティグループが割り当てられたポッドは、インターネットにアクセスできません。

`POD_SECURITY_GROUP_ENFORCING_MODE=standard` に設定されたプラグインのバージョン `1.11` 以降を使用している場合、VPC の外部を送信先とする Pod トラフィックはインスタンスのプライマリネットワークインターフェイスの IP アドレスに変換されます。このトラフィックには Pod のセキュリティグループ内のルールではなく、プライマリネットワークインターフェイスのセキュリティグループ内のルールが使用されます。

- セキュリティグループが関連付けられている Pod で Calico ネットワークポリシーを使用するには、Amazon VPC CNI プラグインのバージョン `1.11.0` 以降を使用し、かつ `POD_SECURITY_GROUP_ENFORCING_MODE=standard` に設定することが必要になります。それ以外の場合は、セキュリティグループが関連付けられている Pod との間のトラフィックフローに Calico ネットワークポリシーは適用されず、Amazon EC2 セキュリティグループの適用のみに限定されます。Amazon VPC CNI バージョンを更新するには、「the section called “Amazon VPC CNI”」を参照してください。
- NodeLocal DNSCache を使用するクラスターのセキュリティグループを使用する Amazon EC2 ノードで実行されている Pod は、Amazon VPC CNI プラグインのバージョンが `1.11.0` 以降で `POD_SECURITY_GROUP_ENFORCING_MODE=standard` に設定されている場合にのみサポートされます。Amazon VPC CNI プラグインのバージョンを更新するには、「the section called “Amazon VPC CNI”」を参照してください。
- Pod のセキュリティグループは、解約率が高い Pod の場合、Pod の起動レイテンシーがより高くなる可能性があります。これはリソースコントローラーのレート制限によるものです。
- EC2 セキュリティグループのスコープは Pod レベル - 詳細については「セキュリティグループ」を参照してください。

  `POD_SECURITY_GROUP_ENFORCING_MODE=standard` および `AWS_VPC_K8S_CNI_EXTERNALSNAT=false` を設定した場合、VPC 外部のエンドポイント宛てのトラフィックは、Pod のセキュリティグループではなくノードのセキュリティグループを使用します。

Amazon EKS Pod のセキュリティグループ用に Amazon VPC CNI Plugin for Kubernetes を設定する

Amazon EC2 インスタンスで Pod を使用する場合は、セキュリティグループ用に Amazon VPC CNI Plugin for Kubernetes を設定する必要があります。

Fargate Pod のみを使用し、クラスターに Amazon EC2 ノードがない場合は、「the section called “SecurityGroupPolicy”」を参照してください。

1. 次のコマンドを使用して、現在の Amazon VPC CNI Plugin for Kubernetes のバージョンをチェックします。

   ```
   kubectl describe daemonset aws-node --namespace kube-system | grep amazon-k8s-cni: |
    cut -d : -f 3
   ```

   出力例は次のとおりです。

   ```
   v1.7.6
   ```

   Amazon VPC CNI Plugin for Kubernetes のバージョンが 1.7.7 より前の場合は、プラグインをバージョン 1.7.7 以降に更新してください。詳細については、「the section called “Amazon VPC CNI”」を参照してください。

2. AmazonEKSVPCResourceController マネージド IAM ポリシーを Amazon EKS クラスターに関連付けられているクラスターロールに追加します。ポリシーにより、ロールはネットワークインターフェイスとそのプライベート IP アドレスを管理し、ネットワークインターフェイスをインスタンスにアタッチおよびデタッチできます。

   a. クラスター IAM ロールの名前を取得し、変数に格納します。*my-cluster* の部分は自分のクラスター名に置き換えます。

      ```
      cluster_role=$(aws eks describe-cluster --name my-cluster --query cluster.roleArn
       --output text | cut -d / -f 2)
      ```

   b. ロールへのポリシーの付与

      ```
      aws iam attach-role-policy --policy-arn arn:aws:iam::aws:policy/
      AmazonEKSVPCResourceController --role-name $cluster_role
      ```

3. Amazon VPC CNI アドオンを有効にして Pod のネットワークインターフェイスを管理するには、aws-node DaemonSet で ENABLE_POD_ENI 変数を true に設定します。この設定が true になると、クラスター内の各ノードについて、アドオンにより cninode カスタムリソースが作成されます。VPC リソースコントローラーは、1 つの特別なネットワークインターフェイスを作成してアタッチします。これは、トランクネットワークインターフェイスと呼ばれ、説明は aws-k8s-trunk-eni です。

```
kubectl set env daemonset aws-node -n kube-system ENABLE_POD_ENI=true
```

> **Note**
>
> トランクネットワークインターフェイスは、インスタンスタイプでサポートされているネットワークインターフェイスの最大数に含まれます。各インスタンスタイプによりサポートされるネットワークインターフェイスの最大数のリストについては、Amazon EC2 ユーザーガイドの「各インスタンスタイプのネットワークインターフェイスあたりの IP アドレス数」を参照してください。ノードにすでに最大数の標準ネットワークインターフェイスがアタッチされている場合、VPC リソースコントローラーはスペースを予約します。コントローラーが標準ネットワークインターフェイスをデタッチして削除し、トランクネットワークインターフェイスを作成し、インスタンスにアタッチできるように、実行中の Pod をスケールダウンする必要があります。

4. CNINode カスタムリソースがどのノードにあるか、次のコマンドで確認できます。[No resources found] が返された場合、数秒待ってから、もう一度試してください。前のステップで、Amazon VPC CNI Plugin for Kubernetes Pod を再起動する必要があります。再起動には数秒かかります。

```
kubectl get cninode -A
     NAME FEATURES
     ip-192-168-64-141.us-west-2.compute.internal [{"name":"SecurityGroupsForPods"}]
     ip-192-168-7-203.us-west-2.compute.internal [{"name":"SecurityGroupsForPods"}]
```

1.15 より古いバージョンの VPC CNI バージョンを使用している場合は、CNINode カスタムリソースの代わりにノードラベルが使用されていました。true に設定されているノードラベル aws-k8s-trunk-eni がどのノードにあるか、次のコマンドで確認できます。[No resources found] が返された場合、数秒待ってから、もう一度試してください。前のステップで、Amazon VPC CNI Plugin for Kubernetes Pod を再起動する必要があります。再起動には数秒かかります。

```
kubectl get nodes -o wide -l vpc.amazonaws.com/has-trunk-attached=true
```

トランクネットワークインターフェイスが作成されると、Pod にはトランクネットワークインターフェイスまたは標準ネットワークインターフェイスのセカンダリ IP アドレスが割り当てられます。ノードが削除されると、トランクインターフェイスは自動的に削除されます。

後のステップで Pod のセキュリティグループをデプロイすると、VPC リソースコントローラーはブランチネットワークインターフェイスと呼ばれる特別なインターフェイスを aws-k8s-branch-eni の説明と共に作成し、セキュリティグループを関連付けます。ノードにアタッチされた標準ネットワークインターフェイスとトランクネットワークインターフェイスに加えて、ブランチネットワークインターフェイスが作成されます。

Liveness プローブまたは Readiness プローブを使用している場合は、TCP Early Demux も無効にする必要があります。これにより、kubelet は TCP を使用してブランチネットワークインターフェイス上の Pod に接続できます。TCP early demux を無効にするには、次のコマンドを実行します。

```
kubectl patch daemonset aws-node -n kube-system \
  -p '{"spec": {"template": {"spec": {"initContainers": [{"env":
[{"name":"DISABLE_TCP_EARLY_DEMUX","value":"true"}],"name":"aws-vpc-cni-init"}]}}}}'
```

> **Note**
>
> Amazon VPC CNI Plugin for Kubernetes アドオンのバージョン 1.11.0 以降を使用し、かつ POD_SECURITY_GROUP_ENFORCING_MODE=standard に設定している場合、次のステップで説明されているように、前のコマンドを実行する必要はありません。

5. クラスターで NodeLocal DNSCache が使用されている場合、独自のセキュリティグループを持つ Pod で Calico ネットワークポリシーを使用する場合、またはセキュリティグループを割り当てる Pod に対して externalTrafficPolicy を Local に設定したインスタンスターゲットを使用するタイプ NodePort および LoadBalancer の Kubernetes サービスがある場合、バージョン 1.11.0 以降の Amazon VPC CNI Plugin for Kubernetes アドオンを使用して次の設定を有効にする必要があります。

```
kubectl set env daemonset aws-node -n kube-system
 POD_SECURITY_GROUP_ENFORCING_MODE=standard
```

重要: Pod のセキュリティグループルールは、**kubelet** や **nodeLocalDNS** など、同じノードにある Pod とサービスの間のトラフィックまたは Pod 間のトラフィックには適用されません。同じノードで異なるセキュリティグループを使用するポッドは、異なるサブネットに設定されているため通信できません。また、これらのサブネット間のルーティングは無効になっています。Pod から VPC の外部のアドレスへのアウトバウンドトラフィックは、インスタンスのプライマリネットワークインターフェイスの IP アドレスに変換されたネットワークアドレスです

(AWS_VPC_K8S_CNI_EXTERNALSNAT=true に設定していない場合)。このトラフィックには Pod のセキュリティグループ内のルールではなく、プライマリネットワークインターフェイスのセキュリティグループ内のルールが使用されます。** この設定を既存のPod に適用するには、Pod または Pod が実行されているノードを再起動する必要があります。

6. Pod のセキュリティグループポリシーの使用方法については、「the section called "SecurityGroupPolicy"」を参照してください。

Amazon EKS Pod のセキュリティグループポリシーを使用する

Pod のセキュリティグループを使用するには、既存のセキュリティグループが必要です。次のステップは、Pod に対してセキュリティグループポリシーを使用する方法を示しています。次のステップではターミナル間で保持されない変数が使用されるため、特に明記されていない限り、同じターミナルからすべてのステップを完了してください。

Amazon EC2 インスタンスを持つ Pod がある場合は、この手順を使用する前にプラグインを設定する必要があります。詳細については、「the section called "設定"」を参照してください。

1. リソースをデプロイする Kubernetes 名前空間を作成します。*my-namespace* は使用する名前空間の名前に置き換えることができます。

```
kubectl create namespace my-namespace
```

2. アマゾン EKS SecurityGroupPolicy をクラスターにデプロイします。

   a. 次のコンテンツをデバイスにコピーします。サービスアカウントラベルに基づいて Pod を選択したい場合は、*podSelector* を serviceAccountSelector で置き換えることができます。セレクターをどちらか 1 つ指定する必要があります。podSelector が空 (例: podSelector: {}) であると、名前空間内のすべての Pod が選択されます。*my-role* は自分の役割名に変更できます。serviceAccountSelector が空であると、名前空間内のすべてのサービスアカウントが選択されます。*my-security-group-policy* を自分の SecurityGroupPolicy の名前に置き換え、*my-namespace* は SecurityGroupPolicy を作成する名前空間に置き換えることができます。

   my_pod_security_group_id を既存のセキュリティグループの ID に置き換える必要があります。既存のセキュリティグループがない場合は作成する必要があります。詳細については「アマゾン EC2 ユーザーガイド」の「アマゾン EC2 security groups for リナックス instances」(リナックス インスタンス用の アマゾン EC2 セキュリティグループ) を参照してください。1~5 個のセキュリティグループ ID を指定できます。複数の ID を指定した場合、すべ

てのセキュリティグループ内のすべてのルールの組み合わせが、選択した Pod に対して有効になります。

```
cat >my-security-group-policy.yaml <<EOF
apiVersion: vpcresources.k8s.aws/v1beta1
kind: SecurityGroupPolicy
metadata:
  name: my-security-group-policy
  namespace: my-namespace
spec:
  podSelector:
    matchLabels:
      role: my-role
  securityGroups:
    groupIds:
      - my_pod_security_group_id
EOF
```

Important

Pod に対して指定した 1 つまたは複数のセキュリティグループは次の基準を満たす必要があります。

- 存在している必要があります。存在しない場合、セレクターに一致する Pod をデプロイすると、Pod は作成プロセスでスタックしたままになります。Pod の説明を書き込むと、次のようなエラーメッセージが表示されます: `An error occurred (InvalidSecurityGroupID.NotFound) when calling the CreateNetworkInterface operation: The securityGroup ID 'sg-05b1d815d1EXAMPLE' does not exist`。
- プローブを設定した任意のポートを介して、ノードに適用されたセキュリティ グループ (kubelet の場合) からのインバウンド通信を許可する必要があります。
- TCP および UDP ポート 53 番経由のアウトバウンド通信を、CoreDNS を実行しているPod (または Pod が実行されるノード) に割り当てられたセキュリティグループへ許可する必要があります。CoreDNS Pod のセキュリティグループは、指定したセキュリティグループからのインバウンド TCP および UDP ポート 53 番トラフィックを許可する必要があります。
- セキュリティグループには、通信を行う必要がある他の Pod との通信に必要なインバウンドルールとアウトバウンドルールを含める必要があります。

- Fargate でセキュリティグループを使用している場合は、セキュリティグループに、Pod と Kubernetes コントロールプレーンとの通信を許可するルールを含める必要があります。最も簡単な方法はクラスターセキュリティグループをセキュリティグループの 1 つとして指定することです。

セキュリティグループポリシーは、新しくスケジュールされた Pod にのみ適用されます。実行中の Pod には影響しません。

b. ポリシーをデプロイします。

```
kubectl apply -f my-security-group-policy.yaml
```

3. 前のステップで指定した *podSelector* の *my-role* の値に一致するラベルを持つサンプルアプリケーションをデプロイします。

a. 次のコンテンツをデバイスにコピーします。サンプル値を独自のものに置き換えて、変更したコマンドを実行してください。*my-role* を置き換える場合は前のステップでセレクターに指定した値と同じであることを確認してください。

```
cat >sample-application.yaml <<EOF
apiVersion: apps/v1
kind: Deployment
metadata:
  name: my-deployment
  namespace: my-namespace
  labels:
    app: my-app
spec:
  replicas: 4
  selector:
    matchLabels:
      app: my-app
  template:
    metadata:
      labels:
        app: my-app
        role: my-role
    spec:
      terminationGracePeriodSeconds: 120
      containers:
      - name: nginx
        image: public.ecr.aws/nginx/nginx:1.23
        ports:
```

```
        - containerPort: 80
---
apiVersion: v1
kind: Service
metadata:
  name: my-app
  namespace: my-namespace
  labels:
    app: my-app
spec:
  selector:
    app: my-app
  ports:
    - protocol: TCP
      port: 80
      targetPort: 80
EOF
```

b. 次のコマンドを使用して、アプリケーションをデプロイします。アプリケーションをデプロイすると、Amazon VPC CNI Plugin for Kubernetes が `role` ラベルと一致し、前のステップで指定したセキュリティグループが Pod に適用されます。

```
kubectl apply -f sample-application.yaml
```

4. サンプルアプリケーションでデプロイされた Pod を表示します。このトピックの以降の説明ではこのターミナルを `TerminalA` と呼びます。

```
kubectl get pods -n my-namespace -o wide
```

出力例は次のとおりです。

```
NAME                             READY   STATUS    RESTARTS   AGE     IP
  NODE                                             NOMINATED NODE   READINESS GATES
my-deployment-5df6f7687b-4fbjm   1/1     Running   0          7m51s   192.168.53.48
  ip-192-168-33-28.region-code.compute.internal   <none>           <none>
my-deployment-5df6f7687b-j9fl4   1/1     Running   0          7m51s   192.168.70.145
  ip-192-168-92-33.region-code.compute.internal   <none>           <none>
my-deployment-5df6f7687b-rjxcz   1/1     Running   0          7m51s   192.168.73.207
  ip-192-168-92-33.region-code.compute.internal   <none>           <none>
my-deployment-5df6f7687b-zmb42   1/1     Running   0          7m51s   192.168.63.27
  ip-192-168-33-28.region-code.compute.internal   <none>           <none>
```

> **Note**
>
> Pod がスタックしている場合は次のヒントをお試しください。
>
> - Waiting 状態でスタックした Pod があれば、`kubectl describe pod my-deployment-xxxxxxxxxx-xxxxx -n my-namespace` を実行してください。`Insufficient permissions: Unable to create Elastic Network Interface.` が表示された場合、前のステップで IAM クラスター役割に IAM ポリシーを追加したことを確認します。
> - Pod が Pending 状態でスタックした場合、ノードのインスタンスタイプが limits.go のリストに含まれていることを確認します。そのうえで、インスタンスタイプでサポートされるブランチネットワークインターフェイスの最大製品数とノードグループ内のノード数を乗算した数にまだ達していないことを確認します。例えば、`m5.large` インスタンスでは9 つのブランチネットワークインターフェイスがサポートされています。ノードグループに 5 つのノードがある場合、ノードグループに対して最大 45 のブランチネットワークインターフェイスを作成できます。46 番目の Pod をデプロイしようとすると、セキュリティグループに関連付けられた別の Pod が削除されるまで Pending 状態のままになります。

`kubectl describe pod my-deployment-xxxxxxxxxx-xxxxx -n my-namespace` を実行したときに次のようなメッセージが表示されている場合は、無視しても問題ありません。このメッセージは、Amazon VPC CNI Plugin for Kubernetes がホストネットワークの設定を試み、ネットワークインターフェイスの作成中に失敗したときに表示される場合があります。プラグインはネットワークインターフェイスが作成されるまで、このイベントをログに記録します。

```
Failed to create Pod sandbox: rpc error: code = Unknown desc = failed to set up
 sandbox container "e24268322e55c8185721f52df6493684f6c2c3bf4fd59c9c121fd4cdc894579f"
 network for Pod "my-deployment-5df6f7687b-4fbjm": networkPlugin
cni failed to set up Pod "my-deployment-5df6f7687b-4fbjm-c89wx_my-namespace" network:
 add cmd: failed to assign an IP address to container
```

インスタンスタイプで実行できる Pod の最大数を超えることはできません。各インスタンスタイプで実行できるポッドの最大数の一覧については、GitHub の「eni-max-pods.txt」を参照してください。セキュリティグループが関連付けられているポッドを削除するか、ポッドが実行されているノードを削除すると、VPC リソースコントローラーによってブランチネットワークインターフェイスが削除されます。セキュリティグループ用に Pod を含むクラスターを Pod で削除する

場合、コントローラーではブランチネットワークインターフェイスを削除しないため、自分で削除する必要があります。ネットワークインターフェイスの削除方法の詳細については「Amazon EC2 ユーザーガイド」の「ネットワークインターフェイスの削除」を参照してください。

5. 別のターミナルで、Pod のいずれかをシェルで操作します。このトピックの以降の説明ではこのターミナルを TerminalB と呼びます。*5df6f7687b-4fbjm* を、前の手順の出力で返された Pod のいずれかの ID に置き換えます。

```
kubectl exec -it -n my-namespace my-deployment-5df6f7687b-4fbjm -- /bin/bash
```

6. TerminalB のシェルから、サンプルアプリケーションが動作することを確認します。

```
curl my-app
```

出力例は次のとおりです。

```
<!DOCTYPE html>
<html>
<head>
<title>Welcome to nginx!</title>
[...]
```

アプリケーションを実行しているすべての Pod が作成したセキュリティグループに関連付けられているため、出力を受信できます。そのグループにはセキュリティグループが関連付けられているすべての Pod 間のすべてのトラフィックを許可するルールが含まれています。DNS トラフィックはそのセキュリティグループからノードに関連付けられているクラスターセキュリティグループへのアウトバウンドで許可されます。ノードは、Pod が名前のルックアップを実行した CoreDNS Pod を実行しています。

7. TerminalA で、クラスターセキュリティグループへの DNS 通信を許可するセキュリティグループルールを、セキュリティグループから削除します。前のステップで DNS ルールをクラスターセキュリティグループに追加しなかった場合は*$my_cluster_security_group_id* を、ルールを作成したセキュリティグループの ID で置き換えます。

```
aws ec2 revoke-security-group-ingress --group-id $my_cluster_security_group_id --
security-group-rule-ids $my_tcp_rule_id
aws ec2 revoke-security-group-ingress --group-id $my_cluster_security_group_id --
security-group-rule-ids $my_udp_rule_id
```

8. TerminalB で、アプリケーションにもう一度アクセスしてみます。

```
curl my-app
```

出力例は次のとおりです。

```
curl: (6) Could not resolve host: my-app
```

クラスターセキュリティグループが関連付けられている CoreDNS Pod に Pod がアクセスできなくなったため、この試行は失敗します。クラスターセキュリティグループから、Pod に関連付けられたセキュリティグループからの DNS 通信を許可するセキュリティグループルールがなくなりました。

前のステップで、いずれかの Pod に返された IP アドレスを使用してアプリケーションにアクセスしようとすると、セキュリティグループが関連付けられており、名前のルックアップが不要な Pod 間ですべてのポートが許可されているため、引き続き応答を受信します。

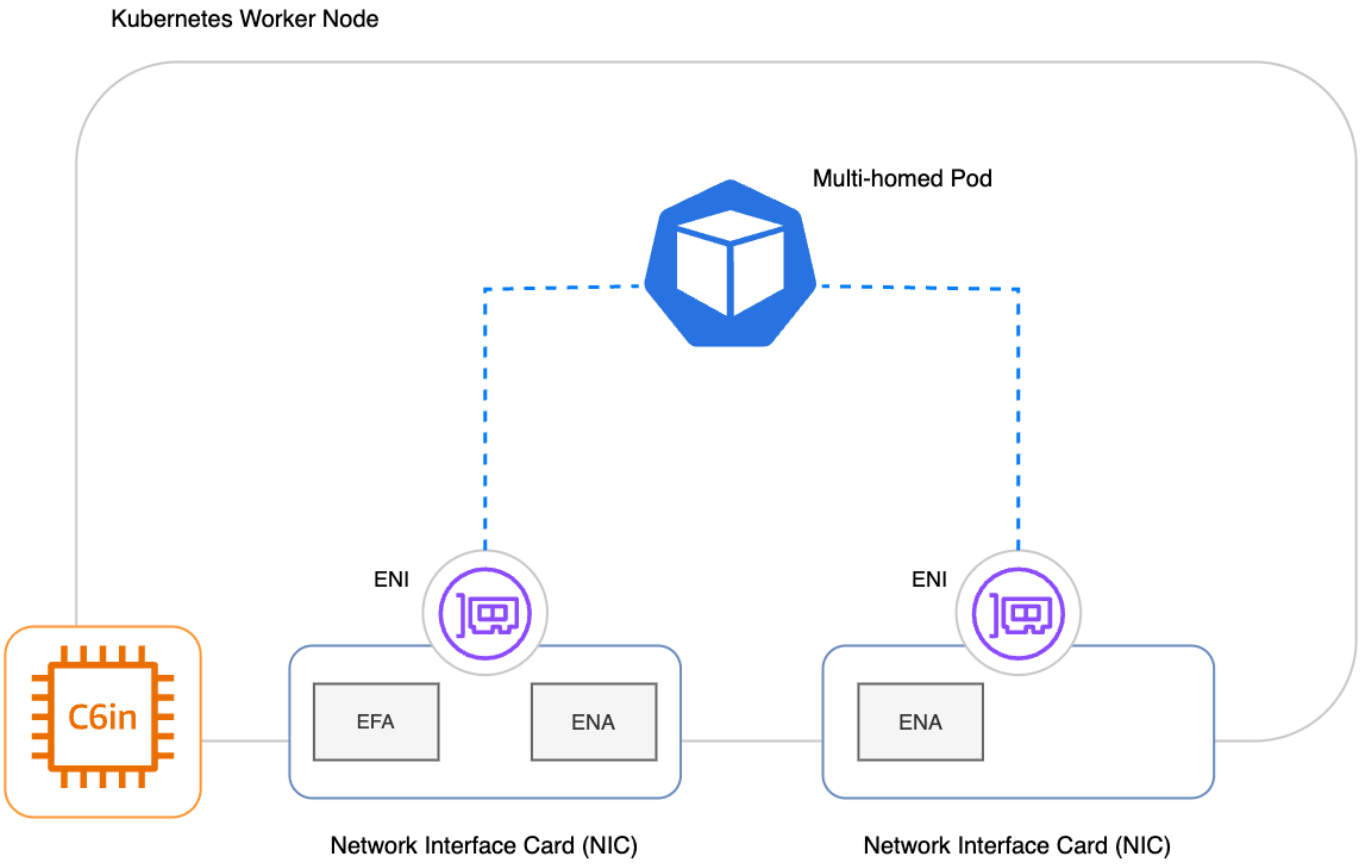
9. 試行し終えたら、作成したサンプルのセキュリティグループポリシー、アプリケーション、およびセキュリティグループを削除できます。TerminalA から、次のコマンドを実行してください。

```
kubectl delete namespace my-namespace
aws ec2 revoke-security-group-ingress --group-id $my_pod_security_group_id --
security-group-rule-ids $my_inbound_self_rule_id
wait
sleep 45s
aws ec2 delete-security-group --group-id $my_pod_security_group_id
```

複数のネットワークインターフェイスをポッドにアタッチする

デフォルトでは、Amazon VPC CNI プラグインは各ポッドに 1 つの IP アドレスを割り当てます。この IP アドレスは、ポッドのすべての送受信トラフィックを処理する Elastic Network Interface にアタッチされます。帯域幅とパケット/秒レートのパフォーマンスを向上させるには、VPC CNI のマルチ NIC 機能を使用してマルチホームポッドを設定できます。マルチホームポッドは、複数のネットワークインターフェイス (および複数の IP アドレス) を使用する単一の Kubernetes ポッドです。マルチホームポッドを実行すると、同時接続を使用してアプリケーショントラフィックを複数のネットワークインターフェイスに分散できます。これは、人工知能 (AI)、機械学習 (ML)、ハイパフォーマンスコンピューティング (HPC) のユースケースに特に役立ちます。

次の図は、複数のネットワークインターフェイスカード (NIC) が使用されているワーカーノードで実行されているマルチホームポッドを示しています。



背景

Amazon EC2 では、Elastic Network Interface は仮想ネットワークカードを表す VPC 内の論理ネットワーキングコンポーネントです。多くの EC2 インスタンスタイプでは、ネットワークインターフェイスはハードウェア内で単一のネットワークインターフェイスカード (NIC) を共有します。この単一の NIC には、最大帯域幅とパケット毎秒レートがあります。

マルチ NIC 機能が有効になっている場合、VPC CNI は、デフォルトでは行われる IP アドレスの一括割り当てを行いません。代わりに、VPC CNI は、新しいポッドの起動時に、各ネットワークカードのネットワークインターフェイスに 1 つの IP アドレスをオンデマンドで割り当てます。この動作により、マルチホームポッドを使用することで増加する IP アドレスの枯渇率が低下します。VPC CNI はオンデマンドで IP アドレスを割り当てるため、マルチ NIC 機能が有効になっているインスタンスではポッドが起動するまでに時間がかかる場合があります。

考慮事項

- Kubernetes クラスターで VPC CNI バージョン `1.20.0` 以降が実行されていることを確認します。マルチ NIC 機能は、VPC CNI のバージョン `1.20.0` 以降でのみ使用できます。
- VPC CNI プラグインで `ENABLE_MULTI_NIC` 環境変数を有効にします。次のコマンドを実行して変数を設定し、DaemonSet のデプロイを開始できます。
  - `kubectl set env daemonset aws-node -n kube-system ENABLE_MULTI_NIC=true`
- 複数のネットワークインターフェイスカード (NIC) を持つワーカーノードを作成してください。複数のネットワークインターフェイスカードを持つ EC2 インスタンスのリストについては、「Amazon EC2 ユーザーガイド」の「ネットワークカード」を参照してください。
- マルチ NIC 機能が有効になっている場合、VPC CNI は、デフォルトでは行われる IP アドレスの一括割り当てを行いません。VPC CNI はオンデマンドで IP アドレスを割り当てるため、マルチ NIC 機能が有効になっているインスタンスではポッドが起動するまでに時間がかかる場合があります。詳細については、前のセクションを参照してくださいthe section called “背景”
- マルチ NIC 機能を有効にしても、デフォルトではポッドは複数のネットワークインターフェイスを持ちません。マルチ NIC を使用するように各ワークロードを設定する必要があります。複数のネットワークインターフェイスを持つ必要があるワークロードに `k8s.amazonaws.com/nicConfig: multi-nic-attachment` 注釈を追加します。

IPv6 に関する考慮事項

- カスタム IAM ポリシー - IPv6 クラスターの場合は、VPC CNI に次のカスタム IAM ポリシーを作成して使用します。このポリシーはマルチ NIC に固有です。IPv6 クラスターでの VPC CNI の使用に関する一般的な情報については、「the section called “IPv6”」を参照してください。

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "AmazonEKSCNIPolicyIPv6MultiNIC",
            "Effect": "Allow",
            "Action": [
                "ec2:CreateNetworkInterface",
                "ec2:DescribeInstances",
                "ec2:AssignIpv6Addresses",
                "ec2:DetachNetworkInterface",
                "ec2:DescribeNetworkInterfaces",
                "ec2:DescribeTags",
```

```
                "ec2:ModifyNetworkInterfaceAttribute",
                "ec2:DeleteNetworkInterface",
                "ec2:DescribeInstanceTypes",
                "ec2:UnassignIpv6Addresses",
                "ec2:AttachNetworkInterface",
                "ec2:DescribeSubnets"
            ],
            "Resource": "*"
        },
        {
            "Sid": "AmazonEKSCNIPolicyENITagIPv6MultiNIC",
            "Effect": "Allow",
            "Action": "ec2:CreateTags",
            "Resource": "arn:aws:ec2:*:*:network-interface/*"
        }
    ]
}
```

- IPv6 移行メカニズムは使用できません - マルチ NIC 機能を使用する場合、VPC CNI はIPv6 クラスター上のポッドに IPv4 アドレスを割り当てません。それ以外の場合、VPC CNI はホストローカル IPv4 アドレスを各ポッドに割り当て、ポッドが別の Amazon VPC またはインターネットの外部 IPv4 リソースと通信できるようにします。

Usage

VPC CNI でマルチ NIC 機能を有効にし、aws-node ポッドを再起動すると、各ワークロードをマルチホームに設定できます。以下は、必要な注釈が付けられた YAML 設定の例です。

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: orders-deployment
  namespace: ecommerce
  labels:
    app: orders
spec:
  replicas: 3
  selector:
    matchLabels:
      app: orders
  template:
    metadata:
```

```
      annotations:
         k8s.amazonaws.com/nicConfig: multi-nic-attachment
      labels:
        app: orders
    spec:
...
```

よくある質問

1. ネットワークインターフェイスカード (NIC) とは

ネットワークインターフェイスカード (NIC) は、単にネットワークカードとも呼ばれ、基盤となるクラウドコンピューティングハードウェアのネットワーク接続を可能にする物理デバイスです。最新の EC2 サーバーでは、これは Nitro ネットワークカードを指します。Elastic Network Interface (ENI) は、この基盤となるネットワークカードの仮想表現です。

一部の EC2 インスタンスタイプには、帯域幅とパケットレートのパフォーマンスを向上させるために複数の NIC があります。このようなインスタンスでは、追加のネットワークカードにセカンダリ ENI を割り当てることができます。例えば、ENI #1 はネットワークカードインデックス 0 にアタッチされた NIC のインターフェイスとして機能しますが、ENI #2 は別のネットワークカードインデックスにアタッチされた NIC のインターフェイスとして機能します。

2. マルチホームポッドとは

マルチホームポッドは、複数のネットワークインターフェイス (および複数の IP アドレスを暗示) を持つ単一の Kubernetes ポッドです。各ポッドネットワークインターフェイスは Elastic Network Interface (ENI) に関連付けられており、これらの ENI は基盤となるワーカーノード上の個別の NIC の論理表現です。複数のネットワークインターフェイスを使用すると、マルチホームポッドのデータ転送容量が増え、データ転送速度も上がります。

> **Important**
>
> VPC CNI は、複数の NIC を持つインスタンスタイプでのみマルチホームポッドを設定できます。

3. この機能を使用する理由は何ですか?

Kubernetes ベースのワークロードでネットワークパフォーマンスをスケールする必要がある場合は、マルチ NIC 機能を使用して、ENA デバイスがアタッチされているすべての基盤となる NIC とイ

ンターフェイスで通信するマルチホームポッドを実行できます。追加のネットワークカードを使用すると、複数の同時接続にアプリケーショントラフィックを分散することで、アプリケーションの帯域幅容量とパケットレートのパフォーマンスが向上します。これは、人工知能 (AI)、機械学習 (ML)、ハイパフォーマンスコンピューティング (HPC) のユースケースに特に役立ちます。

4. この機能の使用方法を教えてください。

1. まず、Kubernetes クラスターが VPC CNI バージョン 1.20 以降を使用していることを確認する必要があります。VPC CNI を EKS アドオンとして更新する手順については、「the section called "更新 (EKS アドオン)"」を参照してください。
2. 次に、ENABLE_MULTI_NIC 環境変数を使用して、VPC CNI でマルチ NIC サポートを有効にする必要があります。
3. 次に、複数のネットワークカードを持つノードを作成して結合する必要があります。複数のネットワークカードを持つ EC2 インスタンスタイプのリストについては、「Amazon EC2 ユーザーガイド」の「ネットワークカード」を参照してください。
4. 最後に、複数のネットワークインターフェイス (マルチホームポッド) を使用するか、単一のネットワークインターフェイスを使用するかのいずれかに各ワークロードを設定します。

5. サポートされているワーカーノードで複数の NIC を使用するようにワークロードを設定するにはどうすればよいですか?

マルチホームポッドを使用するには、注釈 k8s.amazonaws.com/nicConfig: multi-nic-attachment を追加する必要があります。これにより、基盤となるインスタンス内のすべての NIC からポッドに ENI がアタッチされます (ポッドと NIC 間の 1 対多マッピング)。

この注釈がない場合、VPC CNI はポッドに必要なネットワークインターフェイスが 1 つだけであると見なし、使用可能な任意の NIC の ENI から IP を 1 つ割り当てます。

6. この機能ではどのネットワークインターフェイスアダプターがサポートされていますか?

IP トラフィックの基盤となるネットワークカードに少なくとも 1 つの ENA がアタッチされている場合は、任意のネットワークインターフェイスアダプターを使用できます。ENA に関する詳細については、「Amazon EC2 ユーザーガイド」の「Elastic Network Adapter (ENA)」を参照してください。

サポートされているネットワークデバイス設定:

- ENA インターフェイスはVPC の IP ネットワークをサポートするために必要なすべての従来の IP ネットワークとルーティング機能を提供します。詳細については、「EC2 インスタンスで ENA による拡張ネットワーキングを有効にする」を参照してください。

- EFA (EFA と ENA の組み合わせ) インターフェイスは、IP ネットワーク用の ENA デバイスと、低レイテンシーで高スループット通信用の EFA デバイスの両方を提供します。

> **Important**
>
> ネットワークカードに EFA 専用アダプターのみがアタッチされている場合、VPC CNI はマルチホームポッドのネットワーク接続をプロビジョニングするときにそのネットワークカードをスキップします。ただし、EFA 専用アダプターをネットワークカード上の ENA アダプターと組み合わせると、VPC CNI はこのデバイス上の ENI も管理します。EKS クラスターで EFA 専用インターフェイスを使用するには、「the section called “EFA を使用したクラスターのトレーニングの設定”」を参照してください。

7. クラスター内のノードが ENA をサポートしているかどうかを確認できますか?

はい。AWS CLI または EC2 API を使用して、クラスター内の EC2 インスタンスに関するネットワーク情報を取得できます。これにより、インスタンスに ENA サポートがあるかどうかの詳細が表示されます。次の例で、<your-instance-id> をノードの EC2 インスタンス ID に置き換えます。

AWS CLI の例:

```
aws ec2 describe-instances --instance-ids <your-instance-id> --query
 "Reservations[].Instances[].EnaSupport"
```

出力例:

```
[ true ]
```

8. ポッドに関連付けられているさまざまな IP アドレスを確認できますか?

いいえ、簡単にはできません。ただし、ノードから nsenter を使用して、ip route show のような一般的なネットワークツールを実行すれば、追加の IP アドレスとインターフェイスを確認できます。

9. ポッドのネットワークインターフェイスの数を制御できますか?

いいえ。ワークロードがサポートされているインスタンスで複数の NIC を使用するように設定されている場合、単一のポッドにはインスタンス上のすべてのネットワークカードから IP アドレスが自

動的に割り当てられます。または、シングルホームポッドには、インスタンス上の 1 つの NIC に 1 つのネットワークインターフェイスがアタッチされます。

> **Important**
>
> EFA 専用デバイスのみがアタッチされているネットワークカードは、VPC CNI によってスキップされます。

10. 特定の NIC を使用するようにポッドを設定できますか?

いいえ、それはサポートされません。ポッドに関連する注釈がある場合、VPC CNI はワーカーノード上の ENA アダプターですべての NIC を使用するように自動的に設定します。

11. この機能は、他の VPC CNI ネットワーク機能と連携しますか?

はい。VPC CNI のマルチ NIC 機能は、カスタムネットワークと拡張サブネット検出の両方と連携します。ただし、マルチホームポッドではカスタムサブネットやセキュリティグループを使用しません。代わりに、VPC CNI はノードと同じサブネットとセキュリティグループ設定を使用して、マルチホームポッドに IP アドレスとネットワークインターフェイスを割り当てます。カスタムネットワークの詳細については、「the section called “カスタムネットワーキング”」を参照してください。

VPC CNI のマルチ NIC 機能はポッドのセキュリティグループと組み合わせて動作することはできません。

12. この機能でネットワークポリシーを使用できますか?

はい。マルチ NIC で Kubernetes ネットワークポリシーを使用できます。Kubernetes ネットワークポリシーにより、Pod 間で送受信されるネットワークトラフィックが制限されます。VPC CNI でネットワークポリシーを適用する詳細については、「the section called “Kubernetes ポリシー”」を参照してください。

13. マルチ NIC サポートは EKS オートモードで有効になっていますか?

マルチ NIC は EKS オートモードクラスターではサポートされていません。

Amazon EKS クラスターの代替 CNI プラグイン

Amazon VPC CNI plugin for Kubernetes は、Amazon EC2 ノードにおいて Amazon EKS によりサポートされている唯一の CNI プラグインです。Amazon EKS は、Amazon EKS Hybrid Nodes 用の Cilium および Calico のコア機能をサポートしています。Amazon EKS では、アップストリーム

Kubernetes が実行されており、クラスター内の Amazon EC2 ノードに互換性のある CNI プラグインをインストールできます。クラスターに Fargate ノードがある場合、Amazon VPC CNI plugin for Kubernetes は既に Fargate ノードに存在します。これは Fargate ノードで使用できる唯一の CNI プラグインです。Fargate ノードに代替 CNI プラグインをインストールしようとすると失敗します。

Amazon EC2 ノードで代替 CNI プラグインを使用する予定であれば、当該プラグインの商用サポートを受けるか、社内のエキスパートによるトラブルシューティングを行い、解決策を CNI プラグインプロジェクトに提供することをお勧めします。

Amazon EKS は、互換性のある代替 CNI プラグインのサポートを提供するパートナーネットワークと連携しています。バージョンと認定、および実行されたテストの詳細については、次のパートナーのドキュメントを参照してください。

Partner	製品	ドキュメント
Tigera	Calico	インストール手順
Isovalent	Cilium	インストール手順
ジュニパー	クラウドネイティブ Contrail Networking (CN2)	インストール手順
VMware	Antrea	インストール手順

Amazon EKS ではすべてのユースケースをカバーする幅広いオプションの提供を目指しています。

互換性のある代替ネットワークポリシープラグイン

Calico は、コンテナネットワークとセキュリティのために広く採用されているソリューションです。Calico on EKS を使用すると、EKS クラスターに完全準拠のネットワークポリシーが適用されます。さらに、基盤となる VPC の IP アドレスを節約する Calico のネットワークを使用することもできます。Calico Cloud は Calico Open Source の機能を強化すると共に、高度なセキュリティとオブザーバビリティ機能を提供します。

関連付けられたセキュリティグループを持つポッドとの間のトラフィックフローに、Calico ネットワークポリシーは適用されず、Amazon VPC セキュリティグループの適用のみに限定されます。

Calico ネットワークポリシーの適用を使用する場合は、Kubernetes の既知の問題を回避するために、環境変数 `ANNOTATE_POD_IP` を `true` に設定することをお勧めします。この機能を使用するに

は、ポッドの `patch` のアクセス許可を `aws-node` ClusterRole に追加する必要があります。`aws-node` DaemonSet にパッチのアクセス許可を追加すると、プラグインのセキュリティ範囲が広がることに注意してください。詳細については、GitHub の VPC CNI リポジトリの ANNOTATE_POD_IP を参照してください。

Amazon EKS Auto Mode の考慮事項

Amazon EKS Auto Mode は、代替 CNI プラグインおよびネットワークポリシープラグインをサポートしていません。詳細については、「EKS Auto Mode」を参照してください。

Multus を使用して複数のネットワークインターフェイスを Pod にアタッチする

Multus CNI は、Amazon EKS 用のコンテナネットワークインターフェイス (CNI) プラグインです。これにより、Pod に複数のネットワークインターフェイスをアタッチできるようになります。詳細については、GitHub の Multus-CNI を参照してください。

Amazon EKS では、Amazon VPC CNI プラグインによって割り当てられたネットワークインターフェイスが Pod ごとに 1 つあります。Multus を使用すると、複数のインターフェイスを備えたマルチホーム Pod を作成できます。これは、Multusが「meta-plugin」(他の複数の CNI プラグインを呼び出すことができる CNI プラグイン) として機能することによって達成されます。AWS では、Amazon VPC CNI プラグインをデフォルトのデリゲートプラグインとして設定することで、Multus をサポートしています。

- Amazon EKS では、シングルルートの I/O 仮想化 (SR-IOV)、およびデータプレーン開発キット (DPDK) のCNI プラグインを構築および公開することはありません。ただし、Multus で管理されたホストデバイスおよび `ipvlan` プラグインを介して、Amazon EC2 Elastic Network Adapters (ENA) と直接接続することで、パケットアクセラレーションを実現することができます。
- Amazon EKS では Multus をサポートすることで、追加の CNI プラグインの簡単な連鎖を可能にする汎用プロセスを提供します。AWS は、Multus とその連鎖プロセスをサポートしています。ただし、連鎖可能な互換性のある CNI プラグインすべてについてのサポートや、連鎖設定とは無関係な CNI プラグインで発生する可能性のある問題のサポートは行っていません。
- Amazon EKS は Multus プラグインのサポートとライフサイクル管理を提供していますが、追加のネットワークインターフェイスに関連付けられた IP アドレスや他の管理については責任を負いません。Amazon VPC CNI プラグインを使用する、デフォルトのネットワークインターフェイスの IP アドレスと管理に変更はありません。
- デフォルトのデリゲートプラグインとして正式にサポートされるのは、Amazon VPC CNI プラグインのみです。Amazon VPC CNI プラグインをプライマリネットワークに使用しないことを選択

した場合は、公開されている Multus インストールマニフェストを変更して、デフォルトのデリゲートプラグインを代替の CNI に再設定する必要があります。

- Multus は、Amazon VPC CNI をプライマリ CNI として使用する場合にのみサポートされます。セカンダリなど、高次のインターフェイスに使用する場合、Amazon VPC CNI はサポートされません。
- Pod に追加で割り当てたネットワークインターフェイスが Amazon VPC CNI プラグインの管理対象にならないようにするには、そのネットワークインターフェイスに次のタグを追加します。

  key

  : `node.k8s.amazonaws.com/no_manage`

  値

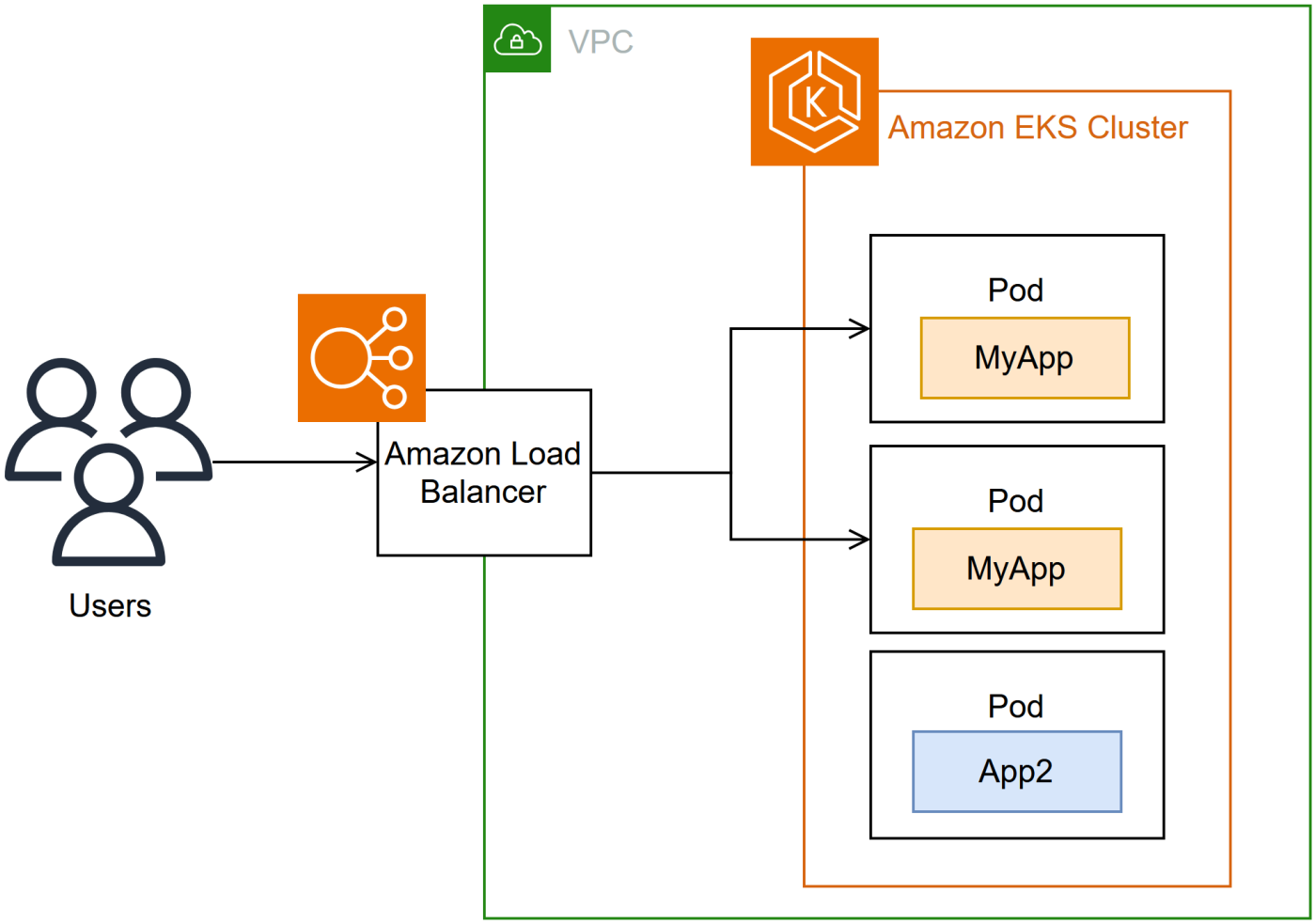
  : `true`
- Multus にはネットワークポリシーとの互換性がありますが、Pod に追加でアタッチされるネットワークインターフェイスの一部であるポートと IP アドレスを含めるように、ポリシーを強化する必要があります。

実装のウォークスルーについては、GitHub の「Multus セットアップガイド」を参照してください。

AWS Load Balancer Controller を使用してインターネットトラフィックをルーティングする

> **Tip**
>
> 今後開催予定の Amazon EKS ワークショップに登録してください。

AWS Load Balancer Controller により、Kubernetes クラスター向けの AWS Elastic Load Balancer を管理できます。コントローラーを使用すると、クラスターアプリケーションをインターネットに公開できます。コントローラーは、クラスターサービスまたは Ingress リソースを指す AWS ロードバランサーをプロビジョニングします。つまり、コントローラーはクラスター内の複数のポッドを指す単一の IP アドレスまたは DNS 名を作成します。



コントローラーは、Kubernetes Ingress または Service のリソースを監視します。これに応じて、適切な AWS Elastic Load Balancing リソースが作成されます。Kubernetes リソースに注釈を適用することで、ロードバランサーの特定の動作を設定できます。例えば、注釈を使用してロードバランサーに AWS セキュリティグループをアタッチできます。

コントローラは、以下のリソースをプロビジョニングします。

Kubernetes **`Ingress`**

Kubernetes `Ingress` を作成すると、LBC は AWS Application Load Balancer (ALB) を作成します。Ingress リソースに適用できる注釈を確認してください。

`LoadBalancer` タイプの Kubernetes サービス

`LoadBalancer` タイプの Kubernetes サービスを作成すると、LBC は AWS Network Load Balancer (NLB) を作成します。サービスリソースに適用できる注釈を確認してください。

以前は、インスタンスターゲットには Kubernetes Network Load Balancer が使用されていましたが、IP ターゲットには LBC が使用されていました。AWS Load Balancer Controller バージョン `2.3.0` 以降では、いずれかのターゲットタイプを使用して NLBs を作成できます。NLB ターゲットタイプの詳細については、Network Load Balancer のユーザーガイドの「ターゲットタイプ」を参照してください。

コントローラーは GitHub で管理されるオープンソースプロジェクトです。

コントローラーをデプロイする前に、「Application Load Balancers を使用したアプリケーションおよび HTTP トラフィックのルーティング」および the section called “ネットワーク負荷分散” で前提条件と考慮事項を確認しておくことをお勧めします。これらのトピックでは、AWS ロードバランサーを含むサンプルアプリケーションをデプロイしています。

Kubernetes **`Gateway`** API

AWS Load Balancer Controller バージョン `2.14.0` 以降では、Kubernetes `Gateway` の作成時に AWS Application Load Balancer (ALB) が作成されます。Kubernetes Gateway は、イングレスよりも多くの設定を標準化します (イングレスでは、よく使用される多くのオプションにカスタム注釈が必要でした)。Gateway リソースに適用可能な設定を確認します。`Gateway` API の詳細については、Kubernetes ドキュメントの「Gateway API」を参照してください。

コントローラーのインストール

次のどちらかの手順を使用して AWS Load Balancer Controller をインストールできます。

- Amazon EKS を初めて使用する場合は、AWS Load Balancer Controller のインストールが簡素化されるため、Helm を使用してインストールすることをお勧めします。詳細については、「the section called “Helm を使用してインストールする”」を参照してください。
- パブリックコンテナレジストリへのネットワークアクセスが制限されているクラスターなどの高度な設定には、Kubernetes マニフェストを使用します。詳細については、「the section called “マニフェストを使用して インストールする”」を参照してください。

非推奨のコントローラーバージョンから移行する

- 非推奨バージョンの AWS Load Balancer Controller がインストールされている場合は、「the section called “非推奨からの移行”」を参照してください。

- 非推奨バージョンはアップグレードできません。このバージョンを削除し、AWS Load Balancer Controller の最新バージョンをインストールする必要があります。
- 非推奨バージョンには以下が含まれます。
  - AWS Load Balancer Controller の前身である AWS ALB Ingress Controller for Kubernetes (「Ingress Controller」)。
  - AWS Load Balancer Controller の任意の `0.1.x` バージョン

レガシークラウドプロバイダー

Kubernetes には、AWS のレガシークラウドプロバイダーが含まれています。レガシークラウドプロバイダーは AWS Load Balancer Controller と同様に、AWS ロードバランサーをプロビジョニングできます。レガシークラウドプロバイダーは Classic Load Balancer を作成します。AWS Load Balancer Controller をインストールしない場合、Kubernetes はデフォルトでレガシークラウドプロバイダーを使用します。AWS Load Balancer Controller をインストールして、レガシークラウドプロバイダーを使用しないようにしてください。

> **Important**
>
> バージョン 2.5 以降では、AWS Load Balancer Controller は、`type: LoadBalancer` を持つ Kubernetes サービスリソースのデフォルトコントローラーとなり、サービスごとに AWS Network Load Balancer (NLB) を作成します。これは、サービスの変更ウェブフックを変化することで実行され、これにより `type: LoadBalancer` の新しいサービスの `spec.loadBalancerClass` フィールドが `service.k8s.aws/nlb` に設定されます。Helm チャートの値 `enableServiceMutatorWebhook` を `false` に設定すると、この機能をオフにして、レガシークラウドプロバイダーをデフォルトのコントローラーとして使用するように戻すことができます。この機能をオフにしない限り、クラスターはサービスに新しい Classic Load Balancer をプロビジョニングしません。既存の Classic Load Balancer は従来どおり機能します。

Helm による AWS Load Balancer Controller のインストール

> **Tip**
>
> 今後開催予定の Amazon EKS ワークショップに登録してください。

> **Tip**
>
> Amazon EKS 自動モード ではネットワーク形成のアドオンをインストールまたはアップグレードする必要はありません。自動モード にはポッドのネットワーク形成とロードバランシング機能が含まれています。
> 詳細については、「EKS Auto Mode」を参照してください。

このトピックでは、Kubernetes のパッケージマネージャーである Helm と `eksctl` を使用して AWS Load Balancer Controller をインストールする方法について説明します。コントローラーはデフォルトのオプションでインストールされます。注釈を使用した設定などコントローラーの詳細については、GitHub で「AWS Load Balancer Controller ドキュメント」を参照してください。

以下のステップでは、サンプル値を独自の値に置き換えます。

前提条件

このチュートリアルを開始する前に、以下のステップを完了する必要があります。

- Amazon EKS クラスターを作成します。作成する場合は、はじめに を参照してください。
- Amazon EKS クラスターで Gateway API CRD をインストールします。
- ローカルマシンに Helm をインストールします。
- Amazon VPC CNI plugin for Kubernetes、`kube-proxy`、および CoreDNS アドオンが、サービスアカウントトークンに記載されている最小のバージョンであることを確認してください。
- AWS Elastic Load Balancing の概念を確認します。詳細については、「Elastic Load Balancing ユーザーガイド」を参照してください。
- Kubernetes サービスと Ingress リソースについて確認します。

考慮事項

このページの設定手順に進む前に、以下の点について考慮してください。

- IAM ポリシーとロール (`AmazonEKSLoadBalancerControllerRole`) は、同じ AWS アカウントの複数の EKS クラスターで再利用できます。
- ロール (`AmazonEKSLoadBalancerControllerRole`) が最初に作成されたのと同じクラスターにコントローラーをインストールする場合は、ロールが存在することを確認した後、「ステップ 2: Load Balancer Controller のインストール」に進みます。

- サービスアカウントの IAM ロール (IRSA) を使用している場合、IRSA はクラスターごとに設定する必要があり、ロールの信頼ポリシー内の OpenID Connect (OIDC) プロバイダー ARN は各 EKS クラスターに固有です。さらに、既存の `AmazonEKSLoadBalancerControllerRole` を使用して新しいクラスターにコントローラーをインストールする場合は、そのロールの信頼ポリシーを更新して新しいクラスターの OIDC プロバイダーを追加し、適切なロール注釈を持つ新しいサービスアカウントを作成します。OIDC プロバイダーが既に存在しているかどうかを確認する、または OIDC プロバイダーを作成するには、「the section called “IAM OIDC プロバイダー”」を参照してください。

ステップ 1: **`eksctl`** を使用して IAM ロールを作成する

次の手順は、AWS Load Balancer Controller v2.14.1 リリースバージョンに関するものです。すべてのリリースの詳細については、GitHub で「AWS Load Balancer Controller Release Page」を参照してください。

1. ユーザーに代わって AWS API を呼び出すことを許可する、AWS Load Balancer Controller 用の IAM ポリシーをダウンロードします。

```
curl -O https://raw.githubusercontent.com/kubernetes-sigs/aws-load-balancer-
controller/v2.14.1/docs/install/iam_policy.json
```

   - AWS パーティションが政府リージョンや中国リージョンなど標準以外のものである場合は、GitHub でポリシーを確認し、ご使用のリージョンに適切なポリシーをダウンロードします。
2. 前のステップでダウンロードしたポリシー を使用して、IAM ポリシーを作成します。

```
aws iam create-policy \
    --policy-name AWSLoadBalancerControllerIAMPolicy \
    --policy-document file://iam_policy.json
```

   > **Note**
   >
   > AWS マネジメントコンソール でポリシーを確認すると、コンソールには [ELB] サービスに関する警告が表示されますが、[ELB v2] サービスに関する警告は表示されません。これは、ポリシー内のアクションの一部が [ELB v2] には存在するが、[ELB] には存在しないために起こります。[ELB] に関する警告は無視できます。
3. クラスター名、リージョンコード、およびアカウント ID の値を置き換えます。

```
eksctl create iamserviceaccount \
    --cluster=<cluster-name> \
    --namespace=kube-system \
    --name=aws-load-balancer-controller \
    --attach-policy-arn=arn:aws:iam::<AWS_ACCOUNT_ID>:policy/
AWSLoadBalancerControllerIAMPolicy \
    --override-existing-serviceaccounts \
    --region <aws-region-code> \
    --approve
```

ステップ 2: AWS Load Balancer Controller をインストールする

1. eks-charts Helm チャートリポジトリを追加します。AWS はこのリポジトリを GitHub で管理しています。

```
helm repo add eks https://aws.github.io/eks-charts
```

2. ローカルリポジトリを更新して、最新のグラフがあることを確認します。

```
helm repo update eks
```

3. AWS Load Balancer コントローラをインストールします。

   Amazon EC2 インスタンスメタデータサービス (IMDS) に対するアクセスが制限されている
   Amazon EC2 ノードにコントローラーをデプロイする場合、または Fargate もしくは Amazon EKS Hybrid Nodes にデプロイする場合には、次の helm コマンドに次のフラグを追加します。

   - --set region=*region-code*
   - --set vpcId=*vpc-xxxxxxxx*

   ******* の部分は自分のクラスター名に置き換えます。次のコマンドの中の aws-load-balancer-controllerは、前のステップで作成した Kubernetes サービスアカウントです。

   Helm チャートの設定の詳細については、GitHub の「values.yaml」を参照してください。

```
helm install aws-load-balancer-controller eks/aws-load-balancer-controller \
  -n kube-system \
  --set clusterName=my-cluster \
  --set serviceAccount.create=false \
  --set serviceAccount.name=aws-load-balancer-controller \
```

```
  --version 1.14.0
```

> **Important**
>
> デプロイされたグラフは、セキュリティに関する更新を自動的に受信しません。この更新が利用可能になったら、手動で新しいグラフにアップグレードする必要があります。アップグレードする場合は、前のコマンドで *install* を upgrade に変更します。

helm install コマンドは、コントローラーのカスタムリソース定義 (CRD) を自動的にインストールします。一方、helm upgrade コマンドでは自動的にインストールされません。helm upgrade, コマンドを使用する場合は、CRD を手動でインストールする必要があります。次のコマンドを実行して、CRD をインストールします。

```
wget https://raw.githubusercontent.com/aws/eks-charts/master/stable/aws-load-balancer-controller/crds/crds.yaml
kubectl apply -f crds.yaml
```

ステップ 3: コントローラーがインストールされていることを確認する

1. コントローラがインストールされていることを確認します。

   ```
   kubectl get deployment -n kube-system aws-load-balancer-controller
   ```

   出力例は次のとおりです。

   ```
   NAME                           READY   UP-TO-DATE   AVAILABLE   AGE
   aws-load-balancer-controller   2/2     2            2           84s
   ```

   Helm を使用してデプロイした場合は、前の出力を受け取ります。Kubernetes マニフェストを使用してデプロイした場合、レプリカは 1 つしかありません。

2. コントローラーを使用して AWS リソースをプロビジョニングする場合には、クラスターは特定の要件を満たしている必要があります。詳細については、「the section called “アプリケーション負荷分散”」および「the section called “ネットワーク負荷分散”」を参照してください。

マニフェストによる AWS Load Balancer Controller のインストール

> **Tip**
>
> Amazon EKS Auto Mode では、ネットワーク形成のアドオンをインストールまたはアップグレードする必要はありません。自動モード にはポッドのネットワーク形成とロードバランシング機能が含まれています。
> 詳細については、「EKS Auto Mode」を参照してください。

このトピックでは、Kubernetes マニフェストをダウンロードし適用することでコントローラーをインストールする方法について説明します。コントローラについての完全な ドキュメントは、GitHub でご覧になれます。

以下のステップでは、サンプル値を独自の値に置き換えます。

前提条件

このチュートリアルを開始する前に、以下のステップを完了する必要があります。

- Amazon EKS クラスターを作成します。作成する場合は「はじめに」を参照してください。
- ローカルマシンに Helm をインストールします。
- Amazon VPC CNI plugin for Kubernetes、`kube-proxy`、および CoreDNS アドオンが、サービスアカウントトークンに記載されている最小のバージョンであることを確認してください。
- AWS Elastic Load Balancing の概念を確認します。詳細については、「Elastic Load Balancing ユーザーガイド」を参照してください。
- Kubernetes サービスと Ingress リソースについて確認します。

考慮事項

このページの設定手順に進む前に、以下の点について考慮してください。

- IAM ポリシーとロール (`AmazonEKSLoadBalancerControllerRole`) は、同じ AWS アカウントの複数の EKS クラスターで再利用できます。
- ロール (`AmazonEKSLoadBalancerControllerRole`) が最初に作成されたのと同じクラスターにコントローラーをインストールする場合は、ロールが存在することを確認した後、「ステップ 2: cert-manager のインストール」に進みます。

- サービスアカウントの IAM ロール (IRSA) を使用している場合、IRSA はクラスターごとに設定する必要があり、ロールの信頼ポリシー内の OpenID Connect (OIDC) プロバイダー ARN は各 EKS クラスターに固有です。さらに、既存の `AmazonEKSLoadBalancerControllerRole` を使用して新しいクラスターにコントローラーをインストールする場合は、そのロールの信頼ポリシーを更新して新しいクラスターの OIDC プロバイダーを追加し、適切なロール注釈を持つ新しいサービスアカウントを作成します。OIDC プロバイダーが既に存在しているかどうかを確認する、または OIDC プロバイダーを作成するには、「the section called “IAM OIDC プロバイダー”」を参照してください。

ステップ 1: IAM を設定する

次の手順は、AWS Load Balancer Controller v2.14.1 リリースバージョンに関するものです。すべてのリリースの詳細については、GitHub で「AWS Load Balancer Controller Release Page」を参照してください。

1. ユーザーに代わって AWS API を呼び出すことを許可する、AWS Load Balancer Controller 用の IAM ポリシーをダウンロードします。

   Example

   AWS

   ```
   curl -O https://raw.githubusercontent.com/kubernetes-sigs/aws-load-balancer-controller/v2.14.1/docs/install/iam_policy.json
   ```

   AWS GovCloud (US)

   ```
   curl -O https://raw.githubusercontent.com/kubernetes-sigs/aws-load-balancer-controller/v2.14.1/docs/install/iam_policy_us-gov.json
   ```

   ```
   mv iam_policy_us-gov.json iam_policy.json
   ```

2. 前のステップでダウンロードしたポリシー を使用して、IAM ポリシーを作成します。

   ```
   aws iam create-policy \
       --policy-name AWSLoadBalancerControllerIAMPolicy \
       --policy-document file://iam_policy.json
   ```

> **Note**
>
> AWS マネジメントコンソール でポリシーを確認すると、コンソールには [ELB] サービスに関する警告が表示されますが、[ELB v2] サービスに関する警告は表示されません。これは、ポリシー内のアクションの一部が [ELB v2] には存在するが、[ELB] には存在しないために起こります。[ELB] に関する警告は無視できます。

Example

eksctl

a. *my-cluster* はご自分のクラスター名に、*111122223333* はご自分のアカウント ID に置き換えた上で、コマンドを実行します。

```
eksctl create iamserviceaccount \
  --cluster=my-cluster \
  --namespace=kube-system \
  --name=aws-load-balancer-controller \
  --role-name AmazonEKSLoadBalancerControllerRole \
  --attach-policy-arn=arn:aws:iam::111122223333:policy/
AWSLoadBalancerControllerIAMPolicy \
  --approve
```

AWS CLI and kubectl

a. クラスターの OIDC プロバイダー ID を取得して、変数に格納します。

```
oidc_id=$(aws eks describe-cluster --name my-cluster --query
 "cluster.identity.oidc.issuer" --output text | cut -d '/' -f 5)
```

b. クラスターの ID を持つ IAM OIDC プロバイダーが既にアカウントにあるかどうかを確認します。クラスターと IAM の両方に OIDC を設定する必要があります。

```
aws iam list-open-id-connect-providers | grep $oidc_id | cut -d "/" -f4
```

出力が返された場合は、クラスター用の IAM OIDC プロバイダーが既に存在します。出力が返されない場合はクラスター用の IAM OIDC プロバイダーを作成する必要があります。詳細については、「the section called “IAM OIDC プロバイダー”」を参照してください。

c. 次のコンテンツをデバイスにコピーします。*111122223333* は、ご自分のアカウント ID に置き換えます。*region-code* を、クラスターのある AWS リージョンに置き換えます。*EXAMPLED539D4633E53DE1B71EXAMPLE* を、前のステップで返された出力に置き換えます。

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "Federated": "arn:aws:iam::111122223333:oidc-provider/oidc.eks.us-
east-1.amazonaws.com/id/EXAMPLED539D4633E53DE1B71EXAMPLE"
            },
            "Action": "sts:AssumeRoleWithWebIdentity",
            "Condition": {
                "StringEquals": {
                    "oidc.eks.us-east-1.amazonaws.com/id/
EXAMPLED539D4633E53DE1B71EXAMPLE:aud": "sts.amazonaws.com",
                    "oidc.eks.us-east-1.amazonaws.com/id/
EXAMPLED539D4633E53DE1B71EXAMPLE:sub": "system:serviceaccount:kube-system:aws-
load-balancer-controller"
                }
            }
        }
    ]
}
```

d. IAM ロールの作成

```
aws iam create-role \
  --role-name AmazonEKSLoadBalancerControllerRole \
  --assume-role-policy-document file://"load-balancer-role-trust-policy.json"
```

e. IAM ロールに、必要な Amazon EKS 管理の IAM ポリシーをアタッチします。*111122223333* は、ご自分のアカウント ID に置き換えます。

```
aws iam attach-role-policy \
```

```
  --policy-arn arn:aws:iam::111122223333:policy/AWSLoadBalancerControllerIAMPolicy
\
  --role-name AmazonEKSLoadBalancerControllerRole
```

f. 次のコンテンツをデバイスにコピーします。*111122223333* は、ご自分のアカウント ID に置き換えます。テキストを置き換えたら、変更したコマンドを実行して aws-load-balancer-controller-service-account.yaml ファイルを作成します。

```
cat >aws-load-balancer-controller-service-account.yaml <<EOF
apiVersion: v1
kind: ServiceAccount
metadata:
  labels:
    app.kubernetes.io/component: controller
    app.kubernetes.io/name: aws-load-balancer-controller
  name: aws-load-balancer-controller
  namespace: kube-system
  annotations:
    eks.amazonaws.com/role-arn: arn:aws:iam::111122223333:role/
AmazonEKSLoadBalancerControllerRole
EOF
```

g. クラスター上で Kubernetes サービスアカウントを作成します。aws-load-balancer-controller という名前の Kubernetes サービスアカウントは、作成した IAM ロール (名前: *AmazonEKSLoadBalancerControllerRole*) でアノテーションされています。

```
kubectl apply -f aws-load-balancer-controller-service-account.yaml
```

ステップ 2: **cert-manager** をインストールする

以下のいずれかの方法で cert-manager をインストールして、Webhook に証明書設定を導入します。詳細については、「cert-manager ドキュメント」の「開始方法」を参照してください。

cert-manager をインストールするには、quay.io コンテナレジストリを使用することをお勧めします。ノードが quay.io コンテナレジストリにアクセスできない場合は、Amazon ECR を使用して cert-manager をインストールします (以下を参照)。

Example

Quay.io

a. ノードが quay.io コンテナレジストリにアクセスできる場合は、cert-manager をインストールして、Webhook に証明書設定を挿入します。

```
kubectl apply \
    --validate=false \
    -f https://github.com/jetstack/cert-manager/releases/download/v1.13.5/cert-manager.yaml
```

Amazon ECR

a. 以下のいずれかの方法で cert-manager をインストールして、Webhook に証明書設定を導入します。詳細については、「cert-manager ドキュメント」の「開始方法」を参照してください。

b. マニフェストをダウンロードします。

```
curl -Lo cert-manager.yaml https://github.com/jetstack/cert-manager/releases/download/v1.13.5/cert-manager.yaml
```

c. 次のイメージをプルして、ノードがアクセスできるリポジトリにプッシュします。イメージをプルし、タグ付けして独自のリポジトリにプッシュする方法の詳細については、the section called “リポジトリにイメージをコピーする” を参照してください。

```
quay.io/jetstack/cert-manager-cainjector:v1.13.5
quay.io/jetstack/cert-manager-controller:v1.13.5
quay.io/jetstack/cert-manager-webhook:v1.13.5
```

d. 三つのイメージのマニフェストの quay.io を、独自のレジストリ名に置き換えます。次のコマンドは、プライベートリポジトリの名前がソースリポジトリと同じであることを前提としています。*111122223333.dkr.ecr.region-code.amazonaws.com* をプライベートレジストリに置き換えます。

```
sed -i.bak -e 's|quay.io|111122223333.dkr.ecr.region-code.amazonaws.com|' ./cert-manager.yaml
```

e. マニフェストを適用します。

```
kubectl apply \
    --validate=false \
    -f ./cert-manager.yaml
```

ステップ 3: AWS Load Balancer Controller をインストールする

1. Controller の詳細をダウンロードします。Controller の詳細については、GitHub のドキュメントを参照してください。

```
curl -Lo v2_14_1_full.yaml https://github.com/kubernetes-sigs/aws-load-balancer-controller/releases/download/v2.14.1/v2_14_1_full.yaml
```

2. ファイルに以下の編集を行います。

   a. v2_14_1_full.yaml ファイルをダウンロードした場合は、次のコマンドを実行してマニフェストの ServiceAccount セクションを削除します。このセクションを削除しないと、前のステップでサービスアカウントに作成した必須の注釈が上書きされます。コントローラーを削除した場合、このセクションの削除により、前のステップで作成したサービスアカウントも保持されます。

   ```
   sed -i.bak -e '764,772d' ./v2_14_1_full.yaml
   ```

   別のバージョンのファイルをダウンロードした場合は、エディタでファイルを開き、次の行を削除します。

   ```
   apiVersion: v1
   kind: ServiceAccount
   metadata:
     labels:
       app.kubernetes.io/component: controller
       app.kubernetes.io/name: aws-load-balancer-controller
     name: aws-load-balancer-controller
     namespace: kube-system
   ---
   ```

   b. *My-cluster* をユーザーのクラスター名に置き換えることにより、ファイルのDeployment spec セクションの your-cluster-name をクラスターの名前に置き換えます。

```
sed -i.bak -e 's|your-cluster-name|my-cluster|' ./v2_14_1_full.yaml
```

c. ノードが Amazon EKS Amazon ECR イメージリポジトリにアクセスできない場合は、次のイメージをプルして、ノードがアクセスできるリポジトリにプッシュする必要があります。イメージをプルし、タグ付けして独自のリポジトリにプッシュする方法の詳細については、the section called “リポジトリにイメージをコピーする” を参照してください。

```
public.ecr.aws/eks/aws-load-balancer-controller:v2.14.1
```

マニフェストにレジストリの名前を追加します。次のコマンドは、プライベートリポジトリの名前がソースリポジトリと同じであると仮定し、プライベートレジストリの名前をファイルに追加します。*111122223333.dkr.ecr.region-code.amazonaws.com* をレジストリに置き換えます。この行は、プライベートリポジトリにソースリポジトリと同じ名前を付けたことを前提としています。そうでない場合は、プライベートレジストリ名の後の `eks/aws-load-balancer-controller` テキストを、リポジトリ名に変更します。

```
sed -i.bak -e 's|public.ecr.aws/eks/aws-load-balancer-controller|
111122223333.dkr.ecr.region-code.amazonaws.com/eks/aws-load-balancer-
controller|' ./v2_14_1_full.yaml
```

d. (Fargate または制限付き IMDS の場合にのみ必須)

Amazon EC2 インスタンスメタデータサービス (IMDS) に対するアクセスが制限されている Amazon EC2 ノードにコントローラをデプロイする場合、または Fargate もしくは Amazon EKS Hybrid Nodes にデプロイする場合には、`- args:` の下に `following parameters` を追加します。

```
[...]
spec:
      containers:
        - args:
            - --cluster-name=your-cluster-name
            - --ingress-class=alb
            - --aws-vpc-id=vpc-xxxxxxxx
            - --aws-region=region-code


[...]
```

3. ファイルを適用します。

```
kubectl apply -f v2_14_1_full.yaml
```

4. IngressClass および IngressClassParams マニフェストをクラスターにダウンロードします。

```
curl -Lo v2.14.1_ingclass.yaml https://github.com/kubernetes-sigs/aws-load-balancer-controller/releases/download/v2.14.1/v2_14_1_ingclass.yaml
```

5. マニフェストをクラスターに適用します。

```
kubectl apply -f v2.14.1_ingclass.yaml
```

ステップ 4: コントローラーがインストールされていることを確認する

1. コントローラがインストールされていることを確認します。

```
kubectl get deployment -n kube-system aws-load-balancer-controller
```

出力例は次のとおりです。

```
NAME                           READY   UP-TO-DATE   AVAILABLE   AGE
aws-load-balancer-controller   2/2     2            2           84s
```

Helm を使用してデプロイした場合は、前の出力を受け取ります。Kubernetes マニフェストを使用してデプロイした場合、レプリカは 1 つしかありません。

2. コントローラーを使用して AWS リソースをプロビジョニングする場合には、クラスターは特定の要件を満たしている必要があります。詳細については、「the section called “アプリケーション負荷分散”」および「the section called “ネットワーク負荷分散”」を参照してください。

非推奨になった ALB Ingress Controller からのアプリの移行

このトピックでは非推奨のコントローラーバージョンから移行する方法を説明します。具体的には、AWS Load Balancer Controller の非推奨バージョンを削除する方法について説明します。

- 非推奨バージョンはアップグレードできません。まずそれらを削除してから、最新バージョンをインストールする必要があります。

- 非推奨バージョンには以下が含まれます。
  - AWS Load Balancer Controller の前身である AWS ALB Ingress Controller for Kubernetes (「Ingress Controller」)。
  - AWS Load Balancer Controller の任意の `0.1.x` バージョン

非推奨のコントローラーバージョンを削除する

> **Note**
>
> 非推奨バージョンは、Helm を使用したか、Kubernetes マニフェストに従って手動でインストールされた可能性があります。この手順は元々インストールしてあるツールを使用して実行してください。

1. `incubator/aws-alb-ingress-controller` Helm チャートをインストールしてある場合はこれをアンインストールします。

   ```
   helm delete aws-alb-ingress-controller -n kube-system
   ```

2. `eks-charts/aws-load-balancer-controller` のバージョン `0.1.x` をインストールしている場合はこれをアンインストールします。`0.1.x` からバージョン `1.0.0` へのアップグレードはWebhook API のバージョンとの互換性がないため動作しません。

   ```
   helm delete aws-load-balancer-controller -n kube-system
   ```

3. イングレス・コントローラー がインストール済みであるかどうかを確認します。

   ```
   kubectl get deployment -n kube-system alb-ingress-controller
   ```

   これはコントローラが取り付けられていない場合の出力です。

   ```
   Error from server (NotFound): deployments.apps "alb-ingress-controller" not found
   ```

   これは、コントローラが取り付けられている場合の出力です。

   ```
   NAME                     READY UP-TO-DATE AVAILABLE AGE
   alb-ingress-controller 1/1   1          1         122d
   ```

4. 次のコマンドを入力してコントローラを削除します。

```
kubectl delete -f https://raw.githubusercontent.com/kubernetes-sigs/aws-alb-ingress-controller/v1.1.8/docs/examples/alb-ingress-controller.yaml
kubectl delete -f https://raw.githubusercontent.com/kubernetes-sigs/aws-alb-ingress-controller/v1.1.8/docs/examples/rbac-role.yaml
```

AWS Load Balancer Controller への移行

ALB Ingress Controller for Kubernetes から AWS Load Balancer Controller に移行するには、以下を実行する必要があります。

1. ALB イングレス・コントローラー を削除します (上記を参照)。
2. AWS ロードバランサー コントローラーをインストールします。
3. AWS Load Balancer Controller で使用される IAM ロールにポリシーを追加します。このポリシーにより、ALB Ingress Controller for Kubernetes によって作成されたリソースを LBC が管理できるようになります。
4. IAM ポリシーをダウンロードします。このポリシーにより、ALB Ingress Controller for Kubernetes によって作成されたリソースを AWS Load Balancer Controller が管理できるようになります。ポリシーを表示することもできます。

```
curl -O https://raw.githubusercontent.com/kubernetes-sigs/aws-load-balancer-controller/v2.14.1/docs/install/iam_policy_v1_to_v2_additional.json
```

5. クラスターが AWS GovCloud (米国東部) または AWS GovCloud (米国西部) の AWS リージョンにある場合は `arn:aws:` を `arn:aws-us-gov:` に置き換えます。

```
sed -i.bak -e 's|arn:aws:|arn:aws-us-gov:|' iam_policy_v1_to_v2_additional.json
```

6. IAM ポリシーを作成し、返された ARN を書き留めます。

```
aws iam create-policy \
  --policy-name AWSLoadBalancerControllerAdditionalIAMPolicy \
  --policy-document file://iam_policy_v1_to_v2_additional.json
```

7. AWS Load Balancer Controller によって使用される IAM ロールに IAM ポリシーをアタッチします。`####` を役割の名前 (例: `AmazonEKSLoadBalancerControllerRole`) に置き換えます。

`eksctl` を使用して役割を作成している場合、作成された役割名を見つけるにはAWS クラウドフォーメーション コンソールを開き、eksctl-*#######*-addon-iamserviceaccount-kube-system-aws-load-balancer-controller スタックを選択してください。[Resources (リソース)] タブを選択してください。役割名は[Physical ID (物理 ID)] 列で見つかります。

```
aws iam attach-role-policy \
  --role-name your-role-name \
  --policy-arn arn:aws:iam::111122223333:policy/
AWSLoadBalancerControllerAdditionalIAMPolicy
```

Amazon EKS クラスターで DNS の CoreDNS を管理する

> **Tip**
>
> Amazon EKS Auto Mode ではネットワーク形成のアドオンをインストールまたはアップグレードする必要はありません。Auto Mode にはポッドのネットワーク形成とロードバランシング機能が含まれています。
> 詳細については、EKS Auto Mode を参照してください。

CoreDNS は、Kubernetes クラスター DNS として機能する、フレキシブルで拡張可能な DNS サーバーです。少なくとも 1 つのノードで Amazon EKS クラスターを起動すると、クラスターにデプロイされたノード数に関係なく、デフォルトで CoreDNS イメージの 2 つのレプリカがデプロイされます。これらの CoreDNS ポッドは、クラスター内のすべてのポッドの名前解決を行います。クラスターに名前空間が CoreDNS `deployment` の名前空間と一致する名前空間を持つ Fargate プロファイルが含まれいる場合、CoreDNS ポッドを Fargate ノードにデプロイできます。CoreDNS の詳細については、Kubernetes ドキュメント の「Using CoreDNS for Service Discovery」を参照してください。

CoreDNS のバージョン

次の表は、Kubernetes バージョンごとに Amazon EKS アドオンタイプの最新バージョンを示しています。

Kubernetes バージョン	CoreDNS のバージョン
1.37	v1.14.7-eksbuild.10
1.36	v1.14.7-eksbuild.10
1.35	v1.14.7-eksbuild.10
1.34	v1.13.2-eksbuild.50
1.33	v1.12.4-eksbuild.57
1.32	v1.11.4-eksbuild.79
1.31	v1.11.4-eksbuild.79

Important

このアドオンを自己管理している場合、表のバージョンは利用可能なセルフマネージドバージョンと同じではない可能性があります。このアドオンのセルフマネージドタイプの更新の詳細については、the section called “更新 (セルフマネージド)” を参照してください。

CoreDNS アップグレードに関する重要な考慮事項

- CoreDNS 更新では PodDisruptionBudget が使用され、更新プロセス中に DNS サービスの可用性を維持します。
- CoreDNS デプロイの安定性と可用性を高めるには、`PodDisruptionBudget` を使用してバージョン `v1.9.3-eksbuild.6` 以降および `v1.10.1-eksbuild.3` をデプロイします。既存の `PodDisruptionBudget` をデプロイした場合、これらのバージョンへのアップグレードは失敗する可能性があります。アップグレードが失敗した場合は次のいずれかのタスクを実行することで問題が解決します。
  - Amazon EKS アドオンをアップグレードする際は競合解決のオプションとして既存の設定を上書きすることを選択してください。デプロイに他のカスタム設定を行った場合は、アップグレード後に他のカスタム設定を再適用できるように、アップグレード前に必ず設定をバックアップしてください。
  - 既存の `PodDisruptionBudget` を削除して、アップグレードを再試行してください。

- EKS アドオンバージョン v1.9.3-eksbuild.3 以降および v1.10.1-eksbuild.6 以降では、CoreDNS デプロイは /ready エンドポイントを使用するように readinessProbe を設定します。このエンドポイントは、CoreDNS の Corefile 設定ファイルで有効になっています。

  カスタム Corefile を使用する場合は、ready プラグインを設定に追加して、プローブが使用できるように /ready エンドポイントを CoreDNS でアクティブにする必要があります。

- EKS アドオンバージョン v1.9.3-eksbuild.7 以降および v1.10.1-eksbuild.4 以降ではPodDisruptionBudget を変更できます。次の例ではフィールドを使用して、アドオンを編集したり、[任意の設定項目]でこれらの設定を変更したりできます。この例はデフォルトの PodDisruptionBudget を示しています。

  ```
  {
      "podDisruptionBudget": {
          "enabled": true,
          "maxUnavailable": 1
          }
  }
  ```

  maxUnavailable または minAvailable を設定できますが、両方を単一の PodDisruptionBudget で設定することはできません。PodDisruptionBudgets の詳細についてはKubernetes ドキュメントの「PodDisruptionBudgetの指定」を参照してください。

  enabled を false に設定しても、PodDisruptionBudget は削除されないことに注意してください。このフィールドを false に設定後、PodDisruptionBudget オブジェクトを削除する必要があります。同様に、PodDisruptionBudget のあるバージョンにアップグレードした後、古いバージョンのアドオンを使用するようにアドオンを編集した場合 (アドオンをダウングレード)、PodDisruptionBudget は削除されません。次のコマンドを実行して、PodDisruptionBudget を削除します。

  ```
  kubectl delete poddisruptionbudget coredns -n kube-system
  ```

- EKS アドオンバージョン v1.10.1-eksbuild.5 以降ではデフォルトの許容値を KEP 2067 に準拠するように node-role.kubernetes.io/master:NoSchedule から node-role.kubernetes.io/control-plane:NoSchedule に変更してください。KEP 2067 の詳細については、GitHub で「Kubernetes Enhancement Proposals (KEPs)」の「KEP-2067: Rename the kubeadm "master" label and taint」を参照してください。

EKS アドオンバージョン v1.8.7-eksbuild.8 以降、および v1.9.3-eksbuild.9 以降では、どちらの許容範囲もすべての Kubernetes バージョンと互換性を維持できるように設定されています。

- EKS アドオンバージョン v1.9.3-eksbuild.11 および v1.10.1-eksbuild.7 以降では、CoreDNS デプロイは topologySpreadConstraints のデフォルト値を設定します。このデフォルト値により、複数のアベイラビリティーゾーンに使用可能なノードがある場合には、CoreDNS Pod がアベイラビリティーゾーン全体に分散します。デフォルト値の代わりに使用するカスタム値を設定できます。デフォルト値は次のとおりです。

```
topologySpreadConstraints:
  - maxSkew: 1
    topologyKey: topology.kubernetes.io/zone
    whenUnsatisfiable: ScheduleAnyway
    labelSelector:
      matchLabels:
        k8s-app: kube-dns
```

- カスタム Corefile 設定とアドオンの更新 – Amazon EKS アドオンは CoreDNS ConfigMap (data.Corefile) を完全に管理します。Corefile を直接変更する場合 (例えば、オンプレミス DNS 解決の条件付きフォワーダーを追加する場合)、アドオンは次の更新中に変更を上書きします。アドオンの更新全体でカスタム Corefile 設定を保持するには、アドオンの configurationValues パラメータで corefile キーを使用します。詳細については、「the section called “カスタマイズ可能なフィールド”」を参照してください。

CoreDNS **v1.11** アップグレードに関する考慮事項

- EKS アドオン バージョン v1.11.1-eksbuild.4 以降ではコンテナイメージは Amazon EKS Distro によって維持される最小ベースイメージに基づいています。これには最小限のパッケージが含まれ、シェルはありません。詳細については、「Amazon EKS Distro」を参照してください。CoreDNS イメージの使用方法とトラブルシューティングは変わりません。

CoreDNS Amazon EKS アドオンを作成する

CoreDNS Amazon EKS アドオンを作成します。アドオンを作成する前に、クラスターを用意する必要があります。詳細については、「the section called “クラスターを作成する”」を参照してください。

1. クラスターにインストールされているアドオンのバージョンを確認します。

```
kubectl describe deployment coredns --namespace kube-system | grep coredns: | cut -
d : -f 3
```

出力例は次のとおりです。

```
v1.10.1-eksbuild.13
```

2. クラスターにインストールされているアドオンのタイプを確認します。クラスターを作成するために使用したツールによっては、現在クラスターに アマゾン EKS アドオンタイプがインストールされていない場合があります。*#######* の部分は、自分のクラスター名に置き換えます。

```
aws eks describe-addon --cluster-name my-cluster --addon-name coredns --query
 addon.addonVersion --output text
```

バージョン番号が返された場合、アマゾン EKS タイプのアドオンがクラスターにインストールされているため、このステップの残りのステップを完了する必要はありません。エラーが返された場合、クラスターに アマゾン EKS タイプのアドオンがインストールされていません。インストールするには、このステップの残りのステップを完了します。

3. 現在インストールされているアドオンの設定を保存します。

```
kubectl get deployment coredns -n kube-system -o yaml > aws-k8s-coredns-old.yaml
```

4. AWS CLI を使用してアドオンを作成します。AWS マネジメントコンソール または eksctl を使用してアドオンを作成する場合は、「the section called “アドオンを作成する”」を参照して、アドオン名の coredns を指定します。デバイスに沿ったコマンドをコピーします。必要に応じてコマンドに次の変更を加え、変更したコマンドを実行してください。
   - *#######* の部分は自分のクラスター名に置き換えます。
   - *v1.11.3-eksbuild.1* を、使用しているクラスターバージョンに対して最新バージョンの表に記載されている最新バージョンに置き換えます。

```
aws eks create-addon --cluster-name my-cluster --addon-name coredns --addon-version
 v1.11.3-eksbuild.1
```

Amazon EKS アドオンのデフォルト設定と競合するカスタム設定を現在のアドオンに適用した場合、作成が失敗する可能性があります。作成に失敗した場合、問題解決に役立つエラーが表

示されます。または、前のコマンドに `--resolve-conflicts OVERWRITE` を追加することもできます。これにより、アドオンは既存のカスタム設定を上書きできます。アドオンを作成したら、カスタム設定で更新できます。

5. クラスターの Kubernetes バージョン用のアドオンの最新バージョンがクラスターに追加されたことを確認します。`#######` の部分は、自分のクラスター名に置き換えます。

```
aws eks describe-addon --cluster-name my-cluster --addon-name coredns --query
 addon.addonVersion --output text
```

アドオンの作成が完了するまでに数秒かかる場合があります。

出力例は次のとおりです。

```
v1.11.3-eksbuild.1
```

6. 元のアドオンに対してカスタム設定を行った場合は、Amazon EKS アドオンを作成する前に、前のステップで保存した設定を使用して、カスタム設定で Amazon EKS アドオンを更新します。アドオンの更新方法については「the section called “更新 (EKS アドオン)”」を参照してください。

CoreDNS Amazon EKS アドオンを更新する

Amazon EKS タイプの アドオンを更新します。Amazon EKS アドオンをクラスターに追加していない場合はアドオンを追加するか、「the section called “更新 (セルフマネージド)”」を参照してください。

開始する前に、アップグレードに関する考慮事項を確認してください。詳細については「the section called “CoreDNS アップグレードに関する重要な考慮事項”」を参照してください。

1. クラスターにインストールされているアドオンのバージョンを確認します。`#######` の部分は自分のクラスター名に置き換えます。

```
aws eks describe-addon --cluster-name my-cluster --addon-name coredns --query
 "addon.addonVersion" --output text
```

出力例は次のとおりです。

```
v1.10.1-eksbuild.13
```

返されたバージョンが、最新バージョンの表にあるクラスターの Kubernetes バージョンのバージョンと同じである場合は既に最新バージョンがクラスターにインストールされているため、この手順の残りを完了する必要はありません。出力にバージョン番号ではなくエラーが表示される場合はアマゾン EKS タイプのアドオンがクラスターにインストールされていません。この手順でアドオンを更新する前に、アドオンを作成する必要があります。

2. 現在インストールされているアドオンの設定を保存します。

```
kubectl get deployment coredns -n kube-system -o yaml > aws-k8s-coredns-old.yaml
```

3. AWS CLI を使用してアドオンを更新します。AWS マネジメントコンソール または eksctl を使用してアドオンを更新する場合は「the section called “アドオンを更新する”」を参照してください。デバイスに沿ったコマンドをコピーします。必要に応じてコマンドに次の変更を加え、変更したコマンドを実行してください。
   - *`#######`* の部分は自分のクラスター名に置き換えます。
   - *`v1.11.3-eksbuild.1`* を、使用しているクラスターバージョンに対して最新バージョンの表に記載されている最新バージョンに置き換えます。
   - `--resolve-conflicts`*`PRESERVE`* オプションはアドオンの既存の設定値を保存します。アドオン設定にカスタム値を設定していて、このオプションを使用しない場合、アマゾン EKS は値をデフォルト値で上書きします。このオプションを使用する場合、実稼働クラスターのアドオンを更新する前に、非稼動クラスターのフィールドおよび値変更をテストすることをお勧めします。この値を OVERWRITE に変更する場合、すべての設定が アマゾン EKS のデフォルト値に変更されます。いずれかの設定にカスタム値を設定した場合、アマゾン EKS のデフォルト値で上書きされる可能性があります。この値を none に変更した場合、アマゾン EKS は設定の値を一切変更しませんが、更新が失敗する可能性があります。更新に失敗した場合、競合の解決に役立つエラーメッセージが返されます。
   - 構成設定を更新しない場合はコマンドから `--configuration-values '{`*`"replicaCount":3`*`}'` を削除します。構成設定を更新する場合は*`"replicaCount":3`* を設定したい設定に置き換えてください。この例では、CoreDNS のレプリカの数は `3` に設定されています。指定する値は設定スキーマに対して有効である必要があります。設定スキーマが不明である場合は`aws eks describe-addon-configuration --addon-name coredns --addon-version` *`v1.11.3-eksbuild.1`* を実行して、*`v1.11.3-eksbuild.1`* を、設定を表示するアドオンのバージョン番号に置き換えます。出力でスキーマが返されます。既存のカスタム設定があり、それをすべて削除してすべての設定の値を Amazon EKS のデフォルトに戻したい場合はコマンドから *`"replicaCount":3`* を削除して、{} が空になるようにします。

設定の詳細については、Kubernetes ドキュメントの「Customizing DNS Service」を参照してください。

```
aws eks update-addon --cluster-name my-cluster --addon-name coredns --addon-version
 v1.11.3-eksbuild.1 \
    --resolve-conflicts PRESERVE --configuration-values '{"replicaCount":3}'
```

更新が完了するまでに数秒かかる場合があります。

4. アドオンのバージョンが更新されたことを確認します。`#######` の部分は自分のクラスター名に置き換えます。

```
aws eks describe-addon --cluster-name my-cluster --addon-name coredns
```

更新が完了するまでに数秒かかる場合があります。

出力例は次のとおりです。

```
{
    "addon": {
        "addonName": "coredns",
        "clusterName": "my-cluster",
        "status": "ACTIVE",
        "addonVersion": "v1.11.3-eksbuild.1",
        "health": {
            "issues": []
        },
        "addonArn": "arn:aws:eks:region:111122223333:addon/my-cluster/coredns/
d2c34f06-1111-2222-1eb0-24f64ce37fa4",
        "createdAt": "2023-03-01T16:41:32.442000+00:00",
        "modifiedAt": "2023-03-01T18:16:54.332000+00:00",
        "tags": {},
        "configurationValues": "{\"replicaCount\":3}"
    }
}
```

CoreDNS Amazon EKS セルフマネージドアドオンを更新する

> **Important**
>
> セルフマネージド型のアドオンを使用する代わりに、アマゾン EKS タイプのアドオンをクラスターに追加することをお勧めします。タイプの違いがよくわからない場合は「the section called “Amazon EKS アドオン”」を参照してください。アマゾン EKS アドオンをクラスターに追加する方法については「the section called “アドオンを作成する”」を参照してください。アマゾン EKS アドオンを使用できない場合はその理由に関する問題をコンテナロードマップ GitHub リポジトリに送信することをお勧めします。

開始する前に、アップグレードに関する考慮事項を確認してください。詳細については「the section called “CoreDNS アップグレードに関する重要な考慮事項”」を参照してください。

1. クラスターにインストールされているアドオンがセルフマネージド型であることを確認します。`######` の部分は自分のクラスター名に置き換えます。

   ```
   aws eks describe-addon --cluster-name my-cluster --addon-name coredns --query
    addon.addonVersion --output text
   ```

   エラーメッセージが返された場合、クラスターにセルフマネージド型のアドオンがインストールされています。インストールするにはこのステップの残りのステップを完了します。バージョン番号が返された場合、クラスターに アマゾン EKS タイプのアドオンがインストールされています。Amazon EKS タイプのアドオンを更新するにはこの手順を使用するのではなく、「CoreDNS Amazon EKS アドオンを更新する」の手順を使用してください。アドオンタイプの違いがよくわからない場合は「the section called “Amazon EKS アドオン”」を参照してください。
2. クラスターに現在インストールされているコンテナイメージのバージョンを確認します。

   ```
   kubectl describe deployment coredns -n kube-system | grep Image | cut -d ":" -f 3
   ```

   出力例は次のとおりです。

   ```
   v1.8.7-eksbuild.2
   ```
3. 現在の CoreDNS のバージョンが `v1.5.0` 以降で、かつ CoreDNS バージョン表に記載されるバージョンよりも前の場合は、このステップをスキップしてください。現在のバージョンが

1.5.0 より前の場合、プロキシアドオンではなく進んだアドオンを使用するためには、CoreDNS の ConfigMap を修正する必要があります。

a. 次のコマンドを使用して ConfigMap を開きます。

```
kubectl edit configmap coredns -n kube-system
```

b. 次の行の proxy を forward に置き換えます。ファイルを保存し、エディタを終了します。

```
proxy . /etc/resolv.conf
```

4. Kubernetes 1.17 以前にクラスターを最初にデプロイした場合は、CoreDNS マニフェストから廃止された行を削除する必要があります。

> **Important**
>
> CoreDNS バージョン 1.7.0 に更新する前に、この手順を完了する必要があります。以前のバージョンに更新する場合でも、この手順を完了することをお勧めします。

a. CoreDNS マニフェストにその行があるかどうか確認します。

```
kubectl get configmap coredns -n kube-system -o jsonpath='{$.data.Corefile}' |
 grep upstream
```

出力が返されない場合、マニフェストにその行がないため、次のステップに進み、CoreDNS を更新することができます。出力が返された場合はその行を削除します。

b. 以下のコマンドを使用して ConfigMapを編集し、ファイル内の upstream という単語がある行を削除します。このファイル内の他の部分は変更しないでください。行を削除したら、変更を保存します。

```
kubectl edit configmap coredns -n kube-system -o yaml
```

5. 現在の CoreDNS イメージバージョンを取得します。

```
kubectl describe deployment coredns -n kube-system | grep Image
```

出力例は次のとおりです。

```
602401143452.dkr.ecr.region-code.amazonaws.com/eks/coredns:v1.8.7-eksbuild.2
```

6. CoreDNS 1.8.3 以降に更新する場合は、endpointslices のアクセス許可を system:coredns Kubernetes clusterrole に追加する必要があります。

   ```
   kubectl edit clusterrole system:coredns -n kube-system
   ```

   ファイルの rules セクション内の既存の権限行の下に次の行を追加します。

   ```
   [...]
   - apiGroups:
     - discovery.k8s.io
     resources:
     - endpointslices
     verbs:
     - list
     - watch
   [...]
   ```

7. *602401143452* および *region-code* を、前のステップで返された出力の値に置き換えて、CoreDNS アドオンを更新します。*v1.11.3-eksbuild.1* を、使用している Kubernetes バージョンについて最新バージョンの表に記載されている CoreDNS バージョンに置き換えます。

   ```
   kubectl set image deployment.apps/coredns -n kube-system
    coredns=602401143452.dkr.ecr.region-code.amazonaws.com/eks/coredns:v1.11.3-
   eksbuild.1
   ```

   出力例は次のとおりです。

   ```
   deployment.apps/coredns image updated
   ```

8. コンテナイメージのバージョンをもう一度チェックして、前のステップで指定したバージョンに更新されたことを確認します。

   ```
   kubectl describe deployment coredns -n kube-system | grep Image | cut -d ":" -f 3
   ```

   出力例は次のとおりです。

```
v1.11.3-eksbuild.1
```

CoreDNS ポッドを高い DNS トラフィックにスケールする

1 つ以上のノードで Amazon EKS クラスターを起動すると、クラスターにデプロイされたノード数に関係なく、デフォルトで CoreDNS イメージの 2 つのレプリカのデプロイがデプロイされます。これらの CoreDNS ポッドは、クラスター内のすべてのポッドの名前解決を行います。アプリケーションは名前解決を使用して、クラスター内のポッドとサービスに接続し、同様にクラスター外のサービスに接続します。ポッドからの名前解決 (クエリ) のリクエスト数が増えると、CoreDNS ポッドが処理しきれなくなって速度が低下し、ポッドが処理できないリクエストが拒否される可能性があります。

CoreDNS ポッドの負荷の増加を処理するには、CoreDNS の自動スケーリングシステムを検討してください。Amazon EKS は、CoreDNS の EKS アドオンバージョンで CoreDNS デプロイの自動スケーリングを管理できます。この CoreDNS オートスケーラーは、ノード数や CPU コア数など、クラスターの状態を継続的にモニタリングします。この情報に基づいて、コントローラーは EKS クラスター内の CoreDNS デプロイのレプリカ数を動的に調整します。この機能は CoreDNS `v1.9` 以降で動作します。CoreDNS 自動スケーリングと互換性のあるバージョンの詳細については、次のセクションを参照してください。

デフォルトでは、システムはクラスター内のノード数と CPU コア数の両方に基づく動的な式を使用して、CoreDNS レプリカを自動的に管理します。式では、(`numberOfNodes` を 16 で割った値) と (`numberOfCPUCores` を 256 で割った値) の大きい方を計算します。システムは 10 分間のピーク期間にわたって需要を評価し、DNS クエリ負荷の増加に対応するために必要な場合は即座にスケールアップします。システムの安定性を維持し中断を回避するために、システムは 3 分ごとにレプリカを 33% 削減することで徐々にスケールダウンします。

EKS アドオン設定を使用して、`nodesPerReplica` および `cpuCoresPerReplica` スケーリングパラメータを調整できます。

- `nodesPerReplica`: CoreDNS レプリカあたりのノード数。デフォルトは 16 です。値が小さいほど、特定の数のノードに対して自動スケーリングが維持するレプリカの数が増えます。
- `cpuCoresPerReplica`: CoreDNS レプリカあたりの CPU コア数。デフォルトは 256 です。値が小さいほど、特定の数の CPU コアに対して自動スケーリングが維持するレプリカの数が増えます。

自動スケーリングは、計算された 2 つのレプリカ数のうち大きい方を使用し、結果を minReplicas と maxReplicas の間にクランプします。nodesPerReplica または cpuCoresPerReplica を使用するには、CoreDNS アドオンがクラスターの Kubernetes バージョンの最小バージョンである必要があります。詳細については、「高度なスケーリングパラメータに対する EKS アドオンの最小バージョン」を参照してください。

この機能はアプリケーションの全体的な可用性とクラスターのスケーラビリティを向上させるために、他の EKS クラスターの自動スケーリングのベストプラクティスと組み合わせて使用することをお勧めします。

前提条件

Amazon EKS が CoreDNS デプロイをスケールするには、次の前提条件を満たす必要があります。

- CoreDNS の EKS アドオンバージョンを使用する必要があります。
- nodesPerReplica または cpuCoresPerReplica を設定するには、次のセクションに示すように、クラスターの Kubernetes バージョンに基づいて最小 CoreDNS アドオンバージョンを使用する必要があります。

高度なスケーリングパラメータに対する EKS アドオンの最小バージョン

次の表に、nodesPerReplica および cpuCoresPerReplica スケーリングパラメータを使用するために各 Kubernetes バージョンに必要な CoreDNS アドオンの最小バージョンを示します。

Kubernetes バージョン	CoreDNS のバージョン
1.36	v1.14.3-eksbuild.16
1.35	v1.14.3-eksbuild.16
1.34	v1.13.2-eksbuild.24
1.33	v1.12.4-eksbuild.31
1.32	v1.11.4-eksbuild.53
1.31	v1.11.4-eksbuild.53

AWS マネジメントコンソールで CoreDNS 自動スケーリングを設定する

1. セルフマネージド型の CoreDNS デプロイではなく、CoreDNS 用の EKS アドオンがあることを確認します。

   クラスターを作成するために使用したツールによっては現在クラスターに Amazon EKS アドオンタイプがインストールされていない場合があります。クラスターにインストールされているアドオンのタイプを確認するには次のコマンドを実行してください。`my-cluster` を自分のクラスター名に置き換えます。

   ```
   aws eks describe-addon --cluster-name my-cluster --addon-name coredns --query
    addon.addonVersion --output text
   ```

   バージョン番号が返された場合、クラスターに Amazon EKS タイプのアドオンがインストールされ、次のステップに進むことができます。エラーが返された場合、クラスターに Amazon EKS タイプのアドオンがインストールされていません。ステップ「CoreDNS Amazon EKS を作成する」の残りのステップを完了して、セルフマネージドバージョンを Amazon EKS アドオンに置き換えます。

2. CoreDNS の EKS アドオンが、最小 EKS アドオンバージョンと同じかそれ以上のバージョンであることを確認します。

   クラスターにインストールされているアドオンのバージョンを確認します。AWS マネジメントコンソールで確認するか、以下のコマンドを実行できます：

   ```
   kubectl describe deployment coredns --namespace kube-system | grep coredns: | cut -
   d : -f 3
   ```

   出力例は次のとおりです。

   ```
   v1.10.1-eksbuild.13
   ```

   このバージョンを前のセクションの最小 EKS アドオンバージョンと比較します。必要に応じて、「CoreDNS Amazon EKS アドオンを更新する」の手順に従って EKS アドオンを上位のバージョンにアップグレードします。

3. 自動スケーリング設定を EKS アドオンの [オプションの設定] に追加します。

   a. Amazon EKS コンソールを開きます。

b. 左のナビゲーションペインで、[クラスター] を選択し、次にアドオンを設定するクラスター名を選択してください。

c. [アドオン] タブを選択してください。

d. CoreDNS アドオンボックスの右上にあるボックスを選択し、次に [編集] を選択してください。

e. [CoreDNS の設定] ページで、次の操作を行います。

i. 使用する [バージョン] を選択してください。前のステップと同じバージョンを保持し、別のアクションでバージョンと設定を更新することをお勧めします。

ii. [オプションの構成設定] を展開します。

iii. [設定値] で JSON キー `"autoscaling":` とネストした JSON オブジェクトの値に、キー `"enabled":` と値 `true` を指定します。結果のテキストは有効な JSON オブジェクトでなければなりません。このキーと値だけがテキストボックス内のデータである場合はキーと値を中括弧 { } で囲みます。次の例は自動スケーリングが有効になっていることを示しています：

```
{
  "autoScaling": {
    "enabled": true
  }
}
```

次の例は、最小レプリカ値と最大レプリカ値を使用して自動スケーリングが有効になっている場合を示しています。クラスター内の DNS サービスに回復力を提供できるように、CoreDNS ポッドの最小数は常に 2 より大きくすることをお勧めします。

```
{
  "autoScaling": {
    "enabled": true,
    "minReplicas": 2,
    "maxReplicas": 10
  }
}
```

次の例では、`nodesPerReplica` および `cpuCoresPerReplica` スケーリングパラメータも設定します。これらのキーには、CoreDNS アドオンの最小バージョンが必要です。詳細については、「高度なスケーリングパラメータに対する EKS アドオンの最小バージョン」を参照してください。

```
{
  "autoScaling": {
    "enabled": true,
    "minReplicas": 2,
    "maxReplicas": 10,
    "nodesPerReplica": 5,
    "cpuCoresPerReplica": 256
  }
}
```

f. CoreDNS ポッドを置き換えて新しい設定を適用するには、[変更を保存] を選択します。

Amazon EKS は CoreDNS の Kubernetes デプロイのロールアウトを使用して EKS アドオンに変更を適用します。ロールアウトのステータスはAWS マネジメントコンソール のアドオン [更新履歴] と `kubectl rollout status deployment/coredns --namespace kube-system` で追跡できます。

`kubectl rollout` は以下のコマンドを実行してください：

```
kubectl rollout

history  -- View rollout history
pause    -- Mark the provided resource as paused
restart  -- Restart a resource
resume   -- Resume a paused resource
status   -- Show the status of the rollout
undo     -- Undo a previous rollout
```

ロールアウトに時間がかかりすぎる場合、Amazon EKS はロールアウトを取り消し、[アドオン更新] のタイプと [失敗] のステータスのメッセージがアドオンの [更新履歴] に追加されます。問題を調査するにはロールアウトの履歴から開始し、CoreDNS のポッドで `kubectl logs` を実行して CoreDNS のログを確認します。

4. [更新履歴] の新しいエントリのステータスが [成功] の場合、ロールアウトが完了し、アドオンはすべての CoreDNS のポッドで新しい設定を使用していることを意味します。クラスター内のノード数とノードの CPU コア数を変更すると、Amazon EKS は CoreDNS デプロイのレプリカ数をスケールします。

AWS コマンドラインインターフェイスで CoreDNS 自動スケーリングを設定する

1. セルフマネージド型の CoreDNS デプロイではなく、CoreDNS 用の EKS アドオンがあることを確認します。

   クラスターを作成するために使用したツールによっては現在クラスターに Amazon EKS アドオンタイプがインストールされていない場合があります。クラスターにインストールされているアドオンのタイプを確認するには次のコマンドを実行してください。`my-cluster` を自分のクラスター名に置き換えます。

   ```
   aws eks describe-addon --cluster-name my-cluster --addon-name coredns --query
    addon.addonVersion --output text
   ```

   バージョン番号が返された場合、クラスターに Amazon EKS タイプのアドオンがインストールされています。エラーが返された場合、クラスターに Amazon EKS タイプのアドオンがインストールされていません。ステップ「CoreDNS Amazon EKS を作成する」の残りのステップを完了して、セルフマネージドバージョンを Amazon EKS アドオンに置き換えます。

2. CoreDNS の EKS アドオンが、最小 EKS アドオンバージョンと同じかそれ以上のバージョンであることを確認します。

   クラスターにインストールされているアドオンのバージョンを確認します。AWS マネジメントコンソールで確認するか、以下のコマンドを実行できます：

   ```
   kubectl describe deployment coredns --namespace kube-system | grep coredns: | cut -
   d : -f 3
   ```

   出力例は次のとおりです。

   ```
   v1.10.1-eksbuild.13
   ```

   このバージョンを前のセクションの最小 EKS アドオンバージョンと比較します。必要に応じて、「CoreDNS Amazon EKS アドオンを更新する」の手順に従って EKS アドオンを上位のバージョンにアップグレードします。

3. 自動スケーリング設定を EKS アドオンの [オプションの設定] に追加します。

   次の AWS CLI コマンドを実行してください。`my-cluster` をクラスターの名前に置き換え、IAM ロール ARN を使用するロールに置き換えます。

```
aws eks update-addon --cluster-name my-cluster --addon-name coredns \
    --resolve-conflicts PRESERVE --configuration-values '{"autoScaling":
{"enabled":true}}'
```

Amazon EKS は CoreDNS の Kubernetes デプロイのロールアウトを使用して EKS アドオンに変更を適用します。ロールアウトのステータスはAWS マネジメントコンソール のアドオン [更新履歴] と `kubectl rollout status deployment/coredns --namespace kube-system` で追跡できます。

`kubectl rollout` は以下のコマンドを実行してください：

```
kubectl rollout

history  -- View rollout history
pause    -- Mark the provided resource as paused
restart  -- Restart a resource
resume   -- Resume a paused resource
status   -- Show the status of the rollout
undo     -- Undo a previous rollout
```

ロールアウトに時間がかかりすぎる場合、Amazon EKS はロールアウトを取り消し、[アドオン更新] のタイプと [失敗] のステータスのメッセージがアドオンの [更新履歴] に追加されます。問題を調査するにはロールアウトの履歴から開始し、CoreDNS のポッドで `kubectl logs` を実行して CoreDNS のログを確認します。

4. (オプション) 自動スケーリングが CoreDNS ポッド数をスケールできる最小値と最大値を指定できます。

   次の例は、最小レプリカ値と最大レプリカ値を使用して自動スケーリングが有効になっている場合を示しています。クラスター内の DNS サービスに回復力を提供できるように、CoreDNS ポッドの最小数は常に 2 より大きくすることをお勧めします。

   ```
   aws eks update-addon --cluster-name my-cluster --addon-name coredns \
       --resolve-conflicts PRESERVE --configuration-values '{"autoScaling":
   {"enabled":true,"minReplicas":2,"maxReplicas":10}}'
   ```

5. (オプション) `nodesPerReplica` および `cpuCoresPerReplica` スケーリングパラメータを設定することもできます。これらのキーには、CoreDNS アドオンの最小バージョンが必要です。詳細

については、「高度なスケーリングパラメータに対する EKS アドオンの最小バージョン」を参照してください。

```
aws eks update-addon --cluster-name my-cluster --addon-name coredns \
    --resolve-conflicts PRESERVE --configuration-values '{"autoScaling":
{"enabled":true,"minReplicas":2,"maxReplicas":10,"nodesPerReplica":5,"cpuCoresPerReplica":256
```

6. 次のコマンドを実行して、アドオンの更新のステータスを確認します：

```
aws eks describe-addon --cluster-name my-cluster --addon-name coredns
```

`"status": "ACTIVE"` という行が表示された場合、ロールアウトが完了し、アドオンはすべての CoreDNS ポッドで新しい設定を使用していることを意味します。クラスター内のノード数とノードの CPU コア数を変更すると、Amazon EKS は CoreDNS デプロイのレプリカ数をスケールします。

CoreDNS メトリクスを使用して Kubernetes DNS 解決をモニタリングする

CoreDNS は、EKS アドオンとして、9153 ポートの CoreDNS からのメトリクスを `kube-dns` サービス内の Prometheus 形式で公開します。Prometheus、Amazon CloudWatch エージェント、またはその他の互換性のあるシステムを使用して、これらのメトリックスをスクレイピング (収集 できます。

Prometheus と CloudWatch エージェントの両方と互換性のあるスクレイプ設定の例については Amazon CloudWatch ユーザーガイドの「Prometheus 用の CloudWatch エージェント設定」を参照してください。

Amazon EKS クラスターで **kube-proxy** を管理する

> **Tip**
>
> Amazon EKS Auto Mode ではネットワーク形成のアドオンをインストールまたはアップグレードする必要はありません。Auto Mode にはポッドのネットワーク形成とロードバランシング機能が含まれています。
> 詳細については、EKS Auto Mode を参照してください。

セルフマネージド型のアドオンを使用する代わりに、Amazon EKS タイプのアドオンをクラスターに追加することをお勧めします。タイプの違いがよくわからない場合は「the section called “Amazon EKS アドオン”」を参照してください。Amazon EKS アドオンをクラスターに追加する方法については、「the section called “アドオンを作成する”」を参照してください。Amazon EKS アドオンを使用できない場合はその理由に関する問題をコンテナロードマップ GitHub リポジトリに送信することをお勧めします。

kube-proxy アドオンは Amazon EKS クラスター内の各 Amazon EC2 ノードにデプロイされます。これは、ノード上のネットワークルールを維持し、Pod へのネットワーク通信を有効にします。アドオンはクラスター内の Fargate ノードにはデプロイされません。詳細については、Kubernetes ドキュメントの「kube-proxy」を参照してください。

Amazon EKS アドオンとしてインストールする

kube-proxy バージョン

次の表は、Kubernetes バージョンごとに Amazon EKS アドオンタイプの最新バージョンを示しています。

Kubernetes バージョン	kube-proxy バージョン
1.37	v1.37.0-eksbuild.4
1.36	v1.36.0-eksbuild.21
1.35	v1.35.3-eksbuild.25
1.34	v1.34.6-eksbuild.25
1.33	v1.33.10-eksbuild.25
1.32	v1.32.13-eksbuild.28
1.31	v1.31.14-eksbuild.32

> **Note**
>
> このアドオンを自己管理している場合、表のバージョンは利用可能なセルフマネージドバージョンと同じではない可能性があります。

kube-proxy コンテナイメージ

kube-proxy コンテナイメージは Amazon EKS Distro によって維持される最小ベースイメージに基づいています。これには最小限のパッケージが含まれ、シェルはありません。詳細については、「Amazon EKS Distro」を参照してください。

次の表に、各 Amazon EKS クラスターバージョンで利用可能な最新のセルフマネージド kube-proxy コンテナイメージのバージョンを示します。

バージョン	kube-proxy
1.37	v1.37.0-eksbuild.4
1.36	v1.36.0-eksbuild.21
1.35	v1.35.3-eksbuild.25
1.34	v1.34.6-eksbuild.25
1.33	v1.33.10-minimal-eksbuild.25
1.32	v1.32.13-minimal-eksbuild.28
1.31	v1.31.14-minimal-eksbuild.32

Amazon EKS アドオンタイプを更新するときは有効な Amazon EKS アドオンバージョンを指定しますが、この表に記載されているバージョンではない可能性があります。これはAmazon EKS アドオンのバージョンが、このアドオンのセルフマネージドタイプを更新するときに指定されるコンテナイメージのバージョンと常に一致するとは限らないためです。このアドオンのセルフマネージドタイプを更新するときはこの表に記載されている有効なコンテナイメージのバージョンを指定します。

Kubernetes kube-proxy セルフマネージドアドオンの更新

> **Important**
>
> セルフマネージド型のアドオンを使用する代わりに、アマゾン EKS タイプのアドオンをクラスターに追加することをお勧めします。タイプの違いがよくわからない場合は「the section called “Amazon EKS アドオン”」を参照してください。アマゾン EKS アドオンをクラスターに追加する方法については「the section called “アドオンを作成する”」を参照して

ください。Amazon EKS アドオンを使用できない場合は、その理由に関する問題をコンテナロードマップ GitHub リポジトリに送信することをお勧めします。

前提条件

- 既存の Amazon EKS クラスター。デプロイするには「はじめに」を参照してください。

考慮事項

- Amazon EKS クラスターの Kube-proxy は、Kubernetes と同じ互換性およびスキューポリシーが適用されます。Amazon EKS アドオンバージョンのクラスターとの互換性の検証について説明します。

1. クラスターにインストールされているアドオンがセルフマネージド型であることを確認します。*#######* の部分は自分のクラスター名に置き換えます。

```
aws eks describe-addon --cluster-name my-cluster --addon-name kube-proxy --query
 addon.addonVersion --output text
```

エラーメッセージが返された場合、クラスターにセルフマネージド型のアドオンがインストールされています。このトピックの残りの手順は、セルフマネージド型のアドオンを更新することです。バージョン番号が返された場合、クラスターに Amazon EKS タイプのアドオンがインストールされています。Amazon EKS タイプのアドオンを更新するには、このトピックの手順を使用するのではなく、「Amazon EKS アドオンの更新」の手順を使用してください。アドオンタイプの違いがよくわからない場合は、「the section called “Amazon EKS アドオン”」を参照してください。

2. クラスターに現在インストールされているコンテナイメージのバージョンを確認します。

```
kubectl describe daemonset kube-proxy -n kube-system | grep Image
```

出力例は次のとおりです。

```
Image:    602401143452.dkr.ecr.region-code.amazonaws.com/eks/kube-proxy:v1.29.1-
eksbuild.2
```

出力例では、*v1.29.1-eksbuild.2* がクラスターにインストールされているバージョンです。

3. *602401143452* および *region-code* を、前のステップの出力の値に置き換えて、kube-proxy アドオンを更新します。*v1.30.6-eksbuild.3* を、「Amazon EKS クラスターの各バージョンで使用可能な最新のセルフマネージド kube-proxy コンテナイメージバージョン」の表に記載されている kube-proxy バージョンに置き換えます。

   > **Important**
   >
   > 各イメージタイプのマニフェストは異なり、デフォルトまたは最小のイメージタイプ間で互換性がありません。エントリポイントと引数が一致するように、前のイメージと同じイメージタイプを使用する必要があります。

   ```
   kubectl set image daemonset.apps/kube-proxy -n kube-system kube-
   proxy=602401143452.dkr.ecr.region-code.amazonaws.com/eks/kube-proxy:v1.30.6-
   eksbuild.3
   ```

   出力例は次のとおりです。

   ```
   daemonset.apps/kube-proxy image updated
   ```

4. 新しいバージョンがクラスターにインストールされたことを確認します。

   ```
   kubectl describe daemonset kube-proxy -n kube-system | grep Image | cut -d ":" -f 3
   ```

   出力例は次のとおりです。

   ```
   v1.30.0-eksbuild.3
   ```

5. 同じクラスターで x86 ノードと Arm ノードを使用しており、クラスターが 2020 年 8 月 17 日より前にデプロイされている場合。以下のコマンドにより kube-proxy マニフェストを編集して、複数のハードウェアアーキテクチャにノードセレクターを含めます。このオペレーションを行うのは 1 回限りです。マニフェストにセレクタを追加した後、アドオンを更新するたびに追加する必要はありません。クラスターが 2020 年 8 月 17 日以降にデプロイされている場合、kube-proxy は既にマルチアーキテクチャに対応しています。

   ```
   kubectl edit -n kube-system daemonset/kube-proxy
   ```

エディタを使用して、次のノードセレクターをファイルに追加し、その内容を保存します。エディタ上で、このテキストを挿入する場所の例については、GitHub のCNI マニフェストファイルをご覧ください。これにより、Kubernetes はノードのハードウェアアーキテクチャに基づいて、正しいハードウェアイメージを取得できるようになります。

```
- key: "kubernetes.io/arch"
  operator: In
  values:
  - amd64
  - arm64
```

6. クラスターを最初に作成したのが Kubernetes バージョン 1.14 以降である場合は、既にこの Affinity Rule が kube-proxy に含まれているため、この手順をスキップできます。Amazon EKS クラスターを最初に作成したのが Kubernetes 1.13 以前のバージョンであり、今後そのクラスターで Fargate ノードを使用する場合は、kube-proxy マニフェストを編集して NodeAffinity ルールを含め、kube-proxy Pod が Fargate ノードでスケジュールしないようにしてください。この編集を行うのは 1 回限りです。Affinity Rule をマニフェストに一度追加したら、アドオンを更新するたびに追加する必要はありません。kube-proxy DaemonSet を編集します。

```
kubectl edit -n kube-system daemonset/kube-proxy
```

エディタで、次の Affinity Rule をファイルの DaemonSet spec セクションに追加して、ファイルを保存します。エディタ上で、このテキストを挿入する場所の例については、GitHub のCNI マニフェストファイルをご覧ください。

```
- key: eks.amazonaws.com/compute-type
  operator: NotIn
  values:
  - fargate
```

Amazon EKS にワークロードとアドオンをデプロイする方法について説明します。

ワークロードはコンテナにデプロイされ、コンテナは Kubernetes の Pod にデプロイされます。Pod には、1 つまたは複数のコンテナが含まれます。通常、Kubernetes サービスには、同じサービスを提供する 1 つまたは複数の Pod がデプロイされます。同じサービスを提供する複数の Pod をデプロイすると、次の操作を実行できます。

- AWS マネジメントコンソールを使用して、各クラスターでワークロードに関する情報の表示を実行できます。
- Kubernetes Vertical Pod Autoscaler を使用して、Pod を垂直方向にスケールアップまたはスケールダウンします。
- Kubernetes Horizontal Pod Autoscaler で需要を満たすために必要な Pod の数を水平方向にスケーリングします。
- 外部 (インターネットにアクセス可能な Podの場合) または内部 (プライベート Pod の場合) ネットワークロードバランサーを作成して、Pod 間でネットワークトラフィックを分散させることができます。ロードバランサーは、OSI モデルのレイヤ 4 でトラフィックをルーティングします。
- Application Load Balancer を作成して、Pod 間でアプリケーショントラフィックを分散します。アプリケーションロードバランサーは、OSI モデルのレイヤ 7 でトラフィックをルーティングします。
- Kubernetes を初めて利用する場合、このトピックでは、サンプルアプリケーションをデプロイします。
- `externalIPs` を使用して、サービスに割り当てることができる IP アドレスを制限できます。

Linux サンプルアプリケーションをデプロイする

このトピックでは、サンプルアプリケーションを Linux ノードのクラスターにデプロイします。

前提条件

- 少なくとも 1 つのノードがある既存の Kubernetes クラスター。既存の Amazon EKS クラスターがない場合は、はじめに のガイドの 1 つを使用して Amazon EKS クラスターをデプロイできます。

- Kubectlコンピュータに がインストールされている。詳細については、the section called “kubectl と eksctl のセットアップ” を参照してください。
- Kubectlクラスターと通信できるように が設定されている。詳細については、the section called “kubectl を使用してクラスターにアクセスする” を参照してください。
- サンプルワークロードを Fargate にデプロイする予定の場合は、名前を変更しない限り、このチュートリアルで作成したのと同じ名前空間 eks-sample-app を含む既存の Fargate プロファイル が必要です。はじめに のいずれかを使用してクラスターを作成した場合は、新しいプロファイルを作成するか、既存のプロファイルに名前空間を追加する必要があります。これは、入門ガイドで作成されたプロファイルでは、このチュートリアルで使用される名前空間が指定されていないためです。VPC には、少なくとも 1 つのプライベートサブネットも必要です。

多くの変数は次のステップで変更できますが、変数値は、指定された場合のみ変更することをお勧めします。Kubernetes Pod、デプロイ、およびサービスへの理解を深めたら、他の値の変更を試すことができます。

名前空間を作成します。

名前空間を使用すると、Kubernetes 内のリソースをグループ化できます。詳細については、Kubernetes ドキュメントの 名前空間 を参照してください。AWS Fargate でコンピューティング管理を簡素化するためにサンプルアプリケーションをデプロイする場合は、起動時にどのポッドが AWS Fargate を使用するかを定義するの namespace の値が eks-sample-app であることを確認します。

```
kubectl create namespace eks-sample-app
```

Kubernetes デプロイメントを作成する

Kubernetes デプロイメントを作成します。このサンプルデプロイでは、パブリックリポジトリからコンテナイメージをプルし、コンテナイメージの 3 つのレプリカ (個別の Pod) をクラスターにデプロイします。詳細については、Kubernetes ドキュメントの デプロイ を参照してください。

1. 次の内容を eks-sample-deployment.yaml という名前のファイルに保存します。サンプルアプリケーションのコンテナではネットワークストレージを使用しませんが、それを使用する必要があるアプリケーションが存在する場合があります。詳細については、アプリデータストレージ を参照してください。
   - kubernetes.io/arch キーの下の amd64 または arm64 values は、アプリケーションをいずれかのハードウェアアーキテクチャにデプロイできることを意味します (クラスター内に両方

がある場合)。これは、このイメージがマルチアーキテクチャイメージであるため可能ですが、すべてがそうであるわけではありません。イメージを取得するリポジトリ内のイメージの詳細を表示することで、イメージがサポートされているハードウェアアーキテクチャを判別できます。ハードウェアアーキテクチャのタイプをサポートしないイメージをデプロイする場合、またはイメージをデプロイしない場合は、そのタイプをマニフェストから削除します。詳細については、Kubernetes ドキュメントの よく知られているラベル、注釈、テイント を参照してください。

- kubernetes.io/os: linux nodeSelector は、クラスターに、例えば、Linux ノードと Windows ノードがある場合、イメージは Linux ノードにのみデプロイされることを意味します。詳細については、Kubernetes ドキュメントの よく知られているラベル、注釈、テイント を参照してください。

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: eks-sample-linux-deployment
  namespace: eks-sample-app
  labels:
    app: eks-sample-linux-app
spec:
  replicas: 3
  selector:
    matchLabels:
      app: eks-sample-linux-app
  template:
    metadata:
      labels:
        app: eks-sample-linux-app
    spec:
      affinity:
        nodeAffinity:
          requiredDuringSchedulingIgnoredDuringExecution:
            nodeSelectorTerms:
            - matchExpressions:
              - key: kubernetes.io/arch
                operator: In
                values:
                - amd64
                - arm64
      containers:
      - name: nginx
```

```
        image: public.ecr.aws/nginx/nginx:1.23
        ports:
        - name: http
          containerPort: 80
        imagePullPolicy: IfNotPresent
      nodeSelector:
        kubernetes.io/os: linux
```

2. デプロイマニフェストをクラスターに適用します。

```
kubectl apply -f eks-sample-deployment.yaml
```

サービスを作成する

サービスを利用すると、単一の IP アドレスまたは名前を使用して、すべてのレプリカにアクセスできます。詳細については、Kubernetes ドキュメントの サービス を参照してください。サンプルアプリケーションには実装されていませんが、他の AWS サービスとやり取りする必要があるアプリケーションがある場合、Pod の Kubernetes サービスアカウントを作成し、AWS IAM アカウントに関連付けることをお勧めします。サービスアカウントを指定すると、ポッドには、他のサービスとやり取りするために指定した最小限のアクセス許可だけが与えられます。詳細については、the section called “IRSA による認証情報” を参照してください。

1. 次の内容を eks-sample-service.yaml という名前のファイルに保存します。Kubernetes は、クラスター内からのみアクセスできる独自の IP アドレスをサービスに割り当てます。クラスターの外部からサービスにアクセスするには、AWS ロードバランサーコントローラー をデプロイして、アプリケーション または ネットワーク トラフィックをサービスに負荷分散します。

```
apiVersion: v1
kind: Service
metadata:
  name: eks-sample-linux-service
  namespace: eks-sample-app
  labels:
    app: eks-sample-linux-app
spec:
  selector:
    app: eks-sample-linux-app
  ports:
    - protocol: TCP
      port: 80
```

```
    targetPort: 80
```

2. サービスマニフェストをクラスターに適用します。

```
kubectl apply -f eks-sample-service.yaml
```

作成されたリソースを確認する

1. eks-sample-app 名前空間内に存在するすべてのリソースを表示します。

```
kubectl get all -n eks-sample-app
```

出力例は次のとおりです。

```
NAME                                                 READY   STATUS    RESTARTS   AGE
pod/eks-sample-linux-deployment-65b7669776-m6qxz     1/1     Running   0          27m
pod/eks-sample-linux-deployment-65b7669776-mmxvd     1/1     Running   0          27m
pod/eks-sample-linux-deployment-65b7669776-qzn22     1/1     Running   0          27m

NAME                                 TYPE          CLUSTER-IP      EXTERNAL-IP   PORT(S)
   AGE
service/eks-sample-linux-service     ClusterIP     10.100.74.8     <none>        80/TCP
   32m

NAME                                            READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/eks-sample-linux-deployment     3/3     3            3           27m

NAME                                                        DESIRED   CURRENT   READY
 AGE
replicaset.apps/eks-sample-linux-deployment-776d8f8fd8      3         3         3
 27m
```

出力では、前の手順でデプロイしたサンプルマニフェストで指定されたサービスとデプロイが表示されます。また、3 つのポッドも表示されます。これは、サンプルマニフェストで 3 replicas が指定されたためです。ポッドの詳細については、Kubernetes ドキュメントの Pods (ポッド) を参照してください。Kubernetes では、サンプルマニフェストで指定されていなくても自動的に replicaset リソースが作成されます。ReplicaSets の詳細については、「Kubernetes ドキュメント」の「ReplicaSet」を参照してください。

> **Note**
>
> Kubernetes では、マニフェストで指定したレプリカの数を維持できます。本稼働デプロイで、Kubernetes を使用してレプリカ数を水平にスケーリングしたり、Pod のコンピューティングリソースを垂直にスケーリングしたりする場合は、「Horizontal Pod Autoscaler を使用してポッドデプロイをスケールする」と「Vertical Pod Autoscaler を使用してポッドリソースを調整する」を使用します。

2. デプロイされたサービスの詳細を表示します。

```
kubectl -n eks-sample-app describe service eks-sample-linux-service
```

出力例は次のとおりです。

```
Name:              eks-sample-linux-service
Namespace:         eks-sample-app
Labels:            app=eks-sample-linux-app
Annotations:       <none>
Selector:          app=eks-sample-linux-app
Type:              ClusterIP
IP Families:       <none>
IP:                10.100.74.8
IPs:               10.100.74.8
Port:              <unset>  80/TCP
TargetPort:        80/TCP
Endpoints:         192.168.24.212:80,192.168.50.185:80,192.168.63.93:80
Session Affinity:  None
Events:            <none>
```

前の出力では、IP: の値は、クラスター内のどのノードまたは Pod からもアクセスできる一意の IP アドレスですが、クラスターの外からはアクセスできません。Endpoints の値は、VPC 内からサービスの一部である Pod に割り当てられる IP アドレスです。

3. 前の手順で 名前空間を表示 したときに、出力にリストされたポッドのうちの 1 つの詳細を表示します。*65b7669776-m6qxz* を、Pod の 1 つに対して返された値に置き換えます。

```
kubectl -n eks-sample-app describe pod eks-sample-linux-deployment-65b7669776-m6qxz
```

簡略化された出力例

```
Name:         eks-sample-linux-deployment-65b7669776-m6qxz
Namespace:    eks-sample-app
Priority:     0
Node:         ip-192-168-45-132.us-west-2.compute.internal/192.168.45.132
[...]
IP:           192.168.63.93
IPs:
  IP:           192.168.63.93
Controlled By:  ReplicaSet/eks-sample-linux-deployment-65b7669776
[...]
Conditions:
  Type              Status
  Initialized       True
  Ready             True
  ContainersReady   True
  PodScheduled      True
[...]
Events:
  Type    Reason     Age    From
 Message
  ----    ------     ----   ----
 -------
  Normal  Scheduled  3m20s  default-scheduler
 Successfully assigned eks-sample-app/eks-sample-linux-deployment-65b7669776-m6qxz to
 ip-192-168-45-132.us-west-2.compute.internal
[...]
```

前の出力では、IP: の値は CIDR ブロックから Pod に割り当てられる一意の IP です。CIDR ブロックは、ノードが存在するサブネットに割り当てられています。異なる CIDR ブロックからのポッド IP アドレスを割り当てたい場合は、デフォルトの動作を変更できます。詳細については、the section called “カスタムネットワーキング” を参照してください。また、Kubernetes スケジューラが IP アドレス *192.168.45.132* を使用してNode上の Pod をスケジュールしたことも確認できます。

> **Tip**
>
> コマンドラインを使用する代わりに、Pod、サービス、デプロイ、その他の Kubernetes リソースに関する多くの詳細を AWS マネジメントコンソール に表示できます。詳細については、the section called “クラスターリソースにアクセスする” を参照してください。

ポッドでシェルを実行する

1. 前のステップで説明したポッドでシェルを実行し、*65b7669776-m6qxz* をいずれかのポッドの ID で置き換えます。

```
kubectl exec -it eks-sample-linux-deployment-65b7669776-m6qxz -n eks-sample-app -- /
bin/bash
```

2. Pod シェルから、前の手順でデプロイと共にインストールされたウェブサーバーからの出力を表示します。サービス名のみ指定する必要があります。デフォルトでは、Amazon EKS クラスターと共にデプロイされる CoreDNS によって、サービスの IP アドレスに解決されます。

```
curl eks-sample-linux-service
```

   出力例は次のとおりです。

```
<!DOCTYPE html>
<html>
<head>
<title>Welcome to nginx!</title>
[...]
```

3. ポッドシェルから、ポッドの DNS サーバーを表示します。

```
cat /etc/resolv.conf
```

   出力例は次のとおりです。

```
nameserver 10.100.0.10
search eks-sample-app.svc.cluster.local svc.cluster.local cluster.local us-
west-2.compute.internal
options ndots:5
```

   前の出力では、クラスターにデプロイされたすべてのポッドに対して `10.100.0.10` が `nameserver` として自動的に割り当てられています。

4. `exit` を入力して、ポッドの接続を切断します。
5. サンプルアプリケーションの使用が終了したら、次のコマンドを使用して、サンプルの名前空間、サービス、デプロイを削除できます。

```
kubectl delete namespace eks-sample-app
```

次のステップ

サンプルアプリケーションをデプロイしたら、次の演習の一部を試すことができます。

- Application Load Balancer を使用してアプリケーションと HTTP トラフィックをルーティングする
- Network Load Balancer を使用して TCP および UDP トラフィックをルーティングする

Windows サンプルアプリケーションをデプロイする

このトピックでは、サンプルアプリケーションを Windows ノードのクラスターにデプロイします。

前提条件

- 少なくとも 1 つのノードがある既存の Kubernetes クラスター。既存の Amazon EKS クラスターがない場合は、はじめに のガイドの 1 つを使用して Amazon EKS クラスターをデプロイできます。クラスターと少なくとも 1 つの Amazon EC2 Windows ノードで Windows サポートを有効にする必要があります。
- Kubectlコンピュータに がインストールされている。詳細については、the section called “kubectl と eksctl のセットアップ” を参照してください。
- Kubectlクラスターと通信できるように が設定されている。詳細については、the section called “kubectl を使用してクラスターにアクセスする” を参照してください。
- サンプルワークロードを Fargate にデプロイする予定の場合は、名前を変更しない限り、このチュートリアルで作成したのと同じ名前空間 eks-sample-app を含む既存の Fargate プロファイル が必要です。はじめに のいずれかを使用してクラスターを作成した場合は、新しいプロファイルを作成するか、既存のプロファイルに名前空間を追加する必要があります。これは、入門ガイドで作成されたプロファイルでは、このチュートリアルで使用される名前空間が指定されていないためです。VPC には、少なくとも 1 つのプライベートサブネットも必要です。

多くの変数は次のステップで変更できますが、変数値は、指定された場合のみ変更することをお勧めします。Kubernetes Pod、デプロイ、およびサービスへの理解を深めたら、他の値の変更を試すことができます。

名前空間を作成します。

名前空間を使用すると、Kubernetes 内のリソースをグループ化できます。詳細については、Kubernetes ドキュメントの 名前空間 を参照してください。AWS Fargate でコンピューティング管理を簡素化するためにサンプルアプリケーションをデプロイする場合は、起動時にどのポッドが AWS Fargate を使用するかを定義するの namespace の値が eks-sample-app であることを確認します。

```
kubectl create namespace eks-sample-app
```

Kubernetes デプロイメントを作成する

このサンプルデプロイでは、パブリックリポジトリからコンテナイメージをプルし、コンテナイメージの 3 つのレプリカ (個別の Pod) をクラスターにデプロイします。詳細については、Kubernetes ドキュメントの デプロイ を参照してください。

1. 次の内容を eks-sample-deployment.yaml という名前のファイルに保存します。サンプルアプリケーションのコンテナではネットワークストレージを使用しませんが、それを使用する必要があるアプリケーションが存在する場合があります。詳細については、アプリデータストレージを参照してください。
   - kubernetes.io/os: windows nodeSelector は、クラスターに、例えば、Windows ノードと Linux ノードがある場合、イメージは Windows ノードにのみデプロイされることを意味します。詳細については、Kubernetes ドキュメントの よく知られているラベル、注釈、テイント を参照してください。

   ```
   apiVersion: apps/v1
   kind: Deployment
   metadata:
     name: eks-sample-windows-deployment
     namespace: eks-sample-app
     labels:
       app: eks-sample-windows-app
   spec:
     replicas: 3
     selector:
       matchLabels:
         app: eks-sample-windows-app
     template:
       metadata:
   ```

```
      labels:
        app: eks-sample-windows-app
    spec:
      affinity:
        nodeAffinity:
          requiredDuringSchedulingIgnoredDuringExecution:
            nodeSelectorTerms:
            - matchExpressions:
              - key: kubernetes.io/arch
                operator: In
                values:
                - amd64
      containers:
      - name: windows-server-iis
        image: mcr.microsoft.com/windows/servercore:ltsc2019
        ports:
        - name: http
          containerPort: 80
        imagePullPolicy: IfNotPresent
        command:
        - powershell.exe
        - -command
        - "Add-WindowsFeature Web-Server; Invoke-WebRequest -UseBasicParsing
 -Uri 'https://dotnetbinaries.blob.core.windows.net/servicemonitor/2.0.1.6/
ServiceMonitor.exe' -OutFile 'C:\\ServiceMonitor.exe'; echo '<html><body><br/
><br/><marquee><H1>Hello EKS!!!<H1><marquee></body><html>' > C:\\inetpub\\wwwroot\
\default.html; C:\\ServiceMonitor.exe 'w3svc'; "
      nodeSelector:
        kubernetes.io/os: windows
```

2. デプロイマニフェストをクラスターに適用します。

```
kubectl apply -f eks-sample-deployment.yaml
```

サービスを作成する

サービスを利用すると、単一の IP アドレスまたは名前を使用して、すべてのレプリカにアクセスできます。詳細については、Kubernetes ドキュメントの サービス を参照してください。サンプルアプリケーションには実装されていませんが、他の AWS サービスとやり取りする必要があるアプリケーションがある場合、Pod の Kubernetes サービスアカウントを作成し、AWS IAM アカウントに関連付けることをお勧めします。サービスアカウントを指定すると、ポッドには、他のサービスとやり

取りするために指定した最小限のアクセス許可だけが与えられます。詳細については、the section called “IRSA による認証情報” を参照してください。

1. 次の内容を eks-sample-service.yaml という名前のファイルに保存します。Kubernetes は、クラスター内からのみアクセスできる独自の IP アドレスをサービスに割り当てます。クラスターの外部からサービスにアクセスするには、AWS ロードバランサーコントローラー をデプロイして、アプリケーション または ネットワーク トラフィックをサービスに負荷分散します。

```
apiVersion: v1
kind: Service
metadata:
  name: eks-sample-windows-service
  namespace: eks-sample-app
  labels:
    app: eks-sample-windows-app
spec:
  selector:
    app: eks-sample-windows-app
  ports:
    - protocol: TCP
      port: 80
      targetPort: 80
```

2. サービスマニフェストをクラスターに適用します。

```
kubectl apply -f eks-sample-service.yaml
```

作成されたリソースを確認する

1. eks-sample-app 名前空間内に存在するすべてのリソースを表示します。

```
kubectl get all -n eks-sample-app
```

出力例は次のとおりです。

```
NAME                                             READY   STATUS    RESTARTS   AGE
pod/eks-sample-windows-deployment-65b7669776-m6qxz   1/1     Running   0          27m
pod/eks-sample-windows-deployment-65b7669776-mmxvd   1/1     Running   0          27m
pod/eks-sample-windows-deployment-65b7669776-qzn22   1/1     Running   0          27m
```

```
NAME                                    TYPE          CLUSTER-IP      EXTERNAL-IP   PORT(S)
   AGE
service/eks-sample-windows-service   ClusterIP    10.100.74.8     <none>        80/
TCP    32m

NAME                                            READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/eks-sample-windows-deployment 3/3     3            3           27m

NAME                                                        DESIRED   CURRENT   READY
 AGE
replicaset.apps/eks-sample-windows-deployment-776d8f8fd8    3         3         3
   27m
```

出力では、前の手順でデプロイしたサンプルマニフェストで指定されたサービスとデプロイが表示されます。また、3 つのポッドも表示されます。これは、サンプルマニフェストで 3 replicas が指定されたためです。ポッドの詳細については、Kubernetes ドキュメントの Pods (ポッド) を参照してください。Kubernetes では、サンプルマニフェストで指定されていなくても自動的に replicaset リソースが作成されます。ReplicaSets の詳細については、「Kubernetes ドキュメント」の「ReplicaSet」を参照してください。

> **Note**
>
> Kubernetes では、マニフェストで指定したレプリカの数を維持できます。本稼働デプロイで、Kubernetes を使用してレプリカ数を水平にスケーリングしたり、Pod のコンピューティングリソースを垂直にスケーリングしたりする場合は、「Horizontal Pod Autoscaler を使用してポッドデプロイをスケールする」と「Vertical Pod Autoscaler を使用してポッドリソースを調整する」を使用します。

2. デプロイされたサービスの詳細を表示します。

```
kubectl -n eks-sample-app describe service eks-sample-windows-service
```

出力例は次のとおりです。

```
Name:              eks-sample-windows-service
Namespace:         eks-sample-app
Labels:            app=eks-sample-windows-app
Annotations:       <none>
Selector:          app=eks-sample-windows-app
Type:              ClusterIP
```

```
IP Families:       <none>
IP:                10.100.74.8
IPs:               10.100.74.8
Port:              <unset>  80/TCP
TargetPort:        80/TCP
Endpoints:         192.168.24.212:80,192.168.50.185:80,192.168.63.93:80
Session Affinity:  None
Events:            <none>
```

前の出力では、IP: の値は、クラスター内のどのノードまたは Pod からもアクセスできる一意の IP アドレスですが、クラスターの外からはアクセスできません。Endpoints の値は、VPC 内からサービスの一部である Pod に割り当てられる IP アドレスです。

3. 前の手順で 名前空間を表示 したときに、出力にリストされたポッドのうちの 1 つの詳細を表示します。*65b7669776-m6qxz* を、Pod の 1 つに対して返された値に置き換えます。

```
kubectl -n eks-sample-app describe pod eks-sample-windows-deployment-65b7669776-m6qxz
```

簡略化された出力例

```
Name:         eks-sample-windows-deployment-65b7669776-m6qxz
Namespace:    eks-sample-app
Priority:     0
Node:         ip-192-168-45-132.us-west-2.compute.internal/192.168.45.132
[...]
IP:           192.168.63.93
IPs:
  IP:           192.168.63.93
Controlled By:  ReplicaSet/eks-sample-windows-deployment-65b7669776
[...]
Conditions:
  Type              Status
  Initialized       True
  Ready             True
  ContainersReady   True
  PodScheduled      True
[...]
Events:
  Type    Reason     Age    From
 Message
  ----    ------     ----   ----
 -------
```

```
  Normal  Scheduled  3m20s  default-scheduler
 Successfully assigned eks-sample-app/eks-sample-windows-deployment-65b7669776-m6qxz
 to ip-192-168-45-132.us-west-2.compute.internal
[...]
```

前の出力では、IP: の値は CIDR ブロックから Pod に割り当てられる一意の IP です。CIDR ブロックは、ノードが存在するサブネットに割り当てられています。異なる CIDR ブロックからのポッド IP アドレスを割り当てたい場合は、デフォルトの動作を変更できます。詳細については、the section called “カスタムネットワーキング” を参照してください。また、Kubernetes スケジューラが IP アドレス *192.168.45.132* を使用してNode上の Pod をスケジュールしたことも確認できます。

> **Tip**
>
> コマンドラインを使用する代わりに、Pod、サービス、デプロイ、その他の Kubernetes リソースに関する多くの詳細を AWS マネジメントコンソール に表示できます。詳細については、the section called “クラスターリソースにアクセスする” を参照してください。

ポッドでシェルを実行する

1. 前のステップで説明したポッドでシェルを実行し、*65b7669776-m6qxz* をいずれかのポッドの ID で置き換えます。

```
kubectl exec -it eks-sample-windows-deployment-65b7669776-m6qxz -n eks-sample-app --
 powershell.exe
```

2. Pod シェルから、前の手順でデプロイと共にインストールされたウェブサーバーからの出力を表示します。サービス名のみ指定する必要があります。デフォルトでは、Amazon EKS クラスターと共にデプロイされる CoreDNS によって、サービスの IP アドレスに解決されます。

```
Invoke-WebRequest -uri eks-sample-windows-service/default.html -UseBasicParsing
```

出力例は次のとおりです。

```
StatusCode        : 200
StatusDescription : OK
Content           : < h t m l > < b o d y > < b r / > < b r / > < m a r q u e e > < H
 1 > H e l l o
```

```
                    E K S ! ! ! < H 1 > < m a r q u e e > < / b o d y > < h t m l >
```

3. ポッドシェルから、ポッドの DNS サーバーを表示します。

```
Get-NetIPConfiguration
```

省略された出力

```
InterfaceAlias       : vEthernet
[...]
IPv4Address          : 192.168.63.14
[...]
DNSServer            : 10.100.0.10
```

前の出力では、クラスターにデプロイされたすべてのポッドに対して `10.100.0.10` が DNS サーバーとして自動的に割り当てられます。

4. `exit` を入力して、ポッドの接続を切断します。
5. サンプルアプリケーションの使用が終了したら、次のコマンドを使用して、サンプルの名前空間、サービス、デプロイを削除できます。

```
kubectl delete namespace eks-sample-app
```

次のステップ

サンプルアプリケーションをデプロイしたら、次の演習の一部を試すことができます。

- Application Load Balancer を使用してアプリケーションと HTTP トラフィックをルーティングする
- Network Load Balancer を使用して TCP および UDP トラフィックをルーティングする

Vertical Pod Autoscaler を使用してポッドリソースを調整する

Kubernetes Vertical Pod Autoscaler は、Pod の CPU とメモリの予約を自動的に調整し、アプリケーションを「適切なサイズ」にします。この調整により、クラスターリソースの使用率が向上し、他の Pod の CPU とメモリが解放されます。このトピックでは、Vertical Pod Autoscaler をクラスターにデプロイし、動作していることを確認する方法について説明します。

- 既存の Amazon EKS クラスターがあります。そうでない場合は、「はじめに」を参照してください。
- Kubernetes メトリクスサーバーがインストールされています。詳細については、「」を参照してくださいthe section called “メトリクスサーバー”
- Amazon EKS クラスターと通信するように設定された kubectl クライアントを使用しています。
- デバイスに、OpenSSL 1.1.1 以降がインストールされています。

Vertical Pod Autoscaler をデプロイする

このセクションでは、Vertical Pod Autoscaler をクラスターにデプロイします。

1. ターミナルウィンドウを開き、Vertical Pod Autoscaler ソースコードをダウンロードするディレクトリに移動します。
2. kubernetes/autoscaler GitHub リポジトリのクローンを作成します。

```
git clone https://github.com/kubernetes/autoscaler.git
```

3. vertical-pod-autoscaler ディレクトリを変更します。

```
cd autoscaler/vertical-pod-autoscaler/
```

4. (オプション) Vertical Pod Autoscaler の別のバージョンをすでにデプロイしている場合は、次のコマンドを使用して削除します。

```
./hack/vpa-down.sh
```

5. ノードが registry.k8s.io コンテナレジストリに対してインターネットからアクセスできない場合は、次のイメージをプルして、独自のプライベートリポジトリにプッシュする必要があります。イメージをプルして独自のプライベートリポジトリにプッシュする方法の詳細については、「the section called “リポジトリにイメージをコピーする”」 を参照してください。

```
registry.k8s.io/autoscaling/vpa-admission-controller:0.10.0
registry.k8s.io/autoscaling/vpa-recommender:0.10.0
registry.k8s.io/autoscaling/vpa-updater:0.10.0
```

プライベートの Amazon ECR リポジトリにイメージをプッシュする場合は、マニフェストの registry.k8s.io をレジストリに置き換えます。*111122223333* は、ご自分のアカウント ID に置き換えます。*region-code* を、クラスターのある AWS リージョンに置き換えます。次のコ

マンドは、お使いのリポジトリにマニフェストのリポジトリ名と同じ名前を付けていることを前提としています。リポジトリに別名を付けた場合も、同様に変更する必要があります。

```
sed -i.bak -e 's/registry.k8s.io/111122223333.dkr.ecr.region-code.amazonaws.com/' ./
deploy/admission-controller-deployment.yaml
sed -i.bak -e 's/registry.k8s.io/111122223333.dkr.ecr.region-code.amazonaws.com/' ./
deploy/recommender-deployment.yaml
sed -i.bak -e 's/registry.k8s.io/111122223333.dkr.ecr.region-code.amazonaws.com/' ./
deploy/updater-deployment.yaml
```

6. 次のコマンドを使用して、Vertical Pod Autoscaler をクラスターにデプロイします。

```
./hack/vpa-up.sh
```

7. Vertical Pod Autoscaler Pod が正常に作成されたことを確認します。

```
kubectl get pods -n kube-system
```

出力例は次のとおりです。

```
NAME                                        READY   STATUS    RESTARTS   AGE
[...]
metrics-server-8459fc497-kfj8w              1/1     Running   0          83m
vpa-admission-controller-68c748777d-ppspd   1/1     Running   0          7s
vpa-recommender-6fc8c67d85-gljpl            1/1     Running   0          8s
vpa-updater-786b96955c-bgp9d                1/1     Running   0          8s
```

Vertical Pod Autoscaler のインストールをテストします

このセクションでは、サンプルアプリケーションをデプロイして、Vertical Pod Autoscaler が動作していることを確認します。

1. 次のコマンドを使用して `hamster.yaml` Vertical Pod Autoscaler の例をデプロイします。

```
kubectl apply -f examples/hamster.yaml
```

2. `hamster` サンプルアプリケーションから Pod を取得します。

```
kubectl get pods -l app=hamster
```

出力例は次のとおりです。

```
hamster-c7d89d6db-rglf5   1/1     Running   0          48s
hamster-c7d89d6db-znvz5   1/1     Running   0          48s
```

3. cpu と memory の予約を表示する Pod の 1 つを説明します。*c7d89d6db-rglf5* を前のステップの出力で返された ID の 1 つに置き換えます。

```
kubectl describe pod hamster-c7d89d6db-rglf5
```

出力例は次のとおりです。

```
[...]
Containers:
  hamster:
    Container ID:  docker://
e76c2413fc720ac395c33b64588c82094fc8e5d590e373d5f818f3978f577e24
    Image:         registry.k8s.io/ubuntu-slim:0.1
    Image ID:      docker-pullable://registry.k8s.io/ubuntu-
slim@sha256:b6f8c3885f5880a4f1a7cf717c07242eb4858fdd5a84b5ffe35b1cf680ea17b1
    Port:          <none>
    Host Port:     <none>
    Command:
      /bin/sh
    Args:
      -c
      while true; do timeout 0.5s yes >/dev/null; sleep 0.5s; done
    State:          Running
      Started:      Fri, 27 Sep 2019 10:35:16 -0700
    Ready:          True
    Restart Count:  0
    Requests:
      cpu:        100m
      memory:     50Mi
[...]
```

元の Pod は 100 millicpu の CPU と 50 メビバイトのメモリを予約していることがわかります。このサンプルアプリケーションでは、100 millicpu は Pod の実行に必要な数よりも少ないため、CPU に制約があります。また、必要とするよりもはるかに少ないメモリを予約します。Vertical Pod Autoscaler `vpa-recommender` のデプロイでは、hamster Pod を分析し

て、CPU とメモリの要件が適切かどうかを確認します。調整が必要な場合、vpa-updater は更新された値で Pod を再起動します。

4. vpa-updater が新しい hamster ポッドを起動するまで待ちます。これには 1~2 分かかります。次のコマンドを使用して、Pod をモニタリングできます。

> **Note**
>
> 新しい Pod が起動されたかどうかが不明な場合は、Pod 名を前のリストと比較します。新しい Pod が起動すると、新しい Pod 名が表示されます。

```
kubectl get --watch Pods -l app=hamster
```

5. 新しい hamster ポッドが開始されたら、それについて説明し、更新された CPU とメモリの予約を表示します。

```
kubectl describe pod hamster-c7d89d6db-jxgfv
```

出力例は次のとおりです。

```
[...]
Containers:
  hamster:
    Container ID:
 docker://2c3e7b6fb7ce0d8c86444334df654af6fb3fc88aad4c5d710eac3b1e7c58f7db
    Image:         registry.k8s.io/ubuntu-slim:0.1
    Image ID:      docker-pullable://registry.k8s.io/ubuntu-
slim@sha256:b6f8c3885f5880a4f1a7cf717c07242eb4858fdd5a84b5ffe35b1cf680ea17b1
    Port:          <none>
    Host Port:     <none>
    Command:
      /bin/sh
    Args:
      -c
      while true; do timeout 0.5s yes >/dev/null; sleep 0.5s; done
    State:          Running
      Started:      Fri, 27 Sep 2019 10:37:08 -0700
    Ready:          True
    Restart Count:  0
    Requests:
```

```
      cpu:        587m
      memory:     262144k
[...]
```

以前の出力では、cpu 予約が元の値の 5 倍以上である 587 millicpu に増加したことがわかります。memory は 262,144 キロバイト (約 250 メビバイト、つまり元の値の 5 倍) に増加しました。この Pod はリソース不足だったため、Vertical Pod Autoscaler で見積りをより適切な値で修正しました。

6. hamster-vpa リソースの詳細を表示して、新しい推奨事項を表示します。

```
kubectl describe vpa/hamster-vpa
```

出力例は次のとおりです。

```
Name:         hamster-vpa
Namespace:    default
Labels:       <none>
Annotations:  kubectl.kubernetes.io/last-applied-configuration:
                {"apiVersion":"autoscaling.k8s.io/
v1beta2","kind":"VerticalPodAutoscaler","metadata":{"annotations":{},"name":"hamster-
vpa","namespace":"d...
API Version:  autoscaling.k8s.io/v1beta2
Kind:         VerticalPodAutoscaler
Metadata:
  Creation Timestamp:  2019-09-27T18:22:51Z
  Generation:          23
  Resource Version:    14411
  Self Link:           /apis/autoscaling.k8s.io/v1beta2/namespaces/default/
verticalpodautoscalers/hamster-vpa
  UID:                 d0d85fb9-e153-11e9-ae53-0205785d75b0
Spec:
  Target Ref:
    API Version:  apps/v1
    Kind:         Deployment
    Name:         hamster
Status:
  Conditions:
    Last Transition Time:  2019-09-27T18:23:28Z
    Status:                True
    Type:                  RecommendationProvided
  Recommendation:
```

```
    Container Recommendations:
      Container Name:  hamster
      Lower Bound:
        Cpu:     550m
        Memory:  262144k
      Target:
        Cpu:     587m
        Memory:  262144k
      Uncapped Target:
        Cpu:     587m
        Memory:  262144k
      Upper Bound:
        Cpu:     21147m
        Memory:  387863636
Events:          <none>
```

7. アプリケーション例の試用が終了したら、次のコマンドで削除します。

```
kubectl delete -f examples/hamster.yaml
```

Horizontal Pod Autoscaler を使用してポッドデプロイをスケールする

Kubernetes Horizontal Pod Autoscaler は、そのリソースの CPU 使用率に基づいて、デプロイ、レプリケーションコントローラー、またはレプリカセット内の Pod の数を自動的にスケーリングします。これにより、アプリケーションは需要の増加に合わせてスケールアウトしたり、リソースが不要になったときにスケールインしたりできるため、ノードを他のアプリケーションに解放できます。ターゲットの CPU 使用率を設定すると、Horizontal Pod Autoscaler はターゲットを満たすようにアプリケーションをスケールインまたはスケールアウトします。

Horizontal Pod Autoscaler は、Kubernetes の標準の API リソースで、Amazon EKS クラスターにメトリクスソース (Kubernetes メトリクスサーバーなど) がインストールされているだけで動作します。アプリケーションのスケーリングを開始するために、クラスターに Horizontal Pod Autoscaler をデプロイまたはインストールする必要はありません。詳細については、Kubernetes ドキュメントの「Horizontal Pod Autoscaler」を参照してください。

このトピックでは、Amazon EKS クラスターの Horizontal Pod Autoscaler の準備や、サンプルアプリケーションで動作することの確認方法を説明しています。

> **Note**
>
> このトピックは、Kubernetes ドキュメントの「Horizontal Pod autoscaler walkthrough」に基づいています。

- 既存の Amazon EKS クラスターがあります。そうでない場合は、「はじめに」を参照してください。
- Kubernetes メトリクスサーバーがインストールされています。詳細については、「」を参照してくださいthe section called “メトリクスサーバー”
- Amazon EKS クラスターと通信するように設定された `kubectl` クライアントを使用しています。

Horizontal Pod Autoscaler テストアプリケーションを実行する

このセクションでは、サンプルアプリケーションをデプロイして、Horizontal Pod Autoscaler が動作していることを確認します。

> **Note**
>
> この例は、Kubernetes ドキュメントの「Horizontal Pod autoscaler walkthrough」に基づいています。

1. 次のコマンドを使用して、シンプルな Apache ウェブサーバーアプリケーションをデプロイします。

   ```
   kubectl apply -f https://k8s.io/examples/application/php-apache.yaml
   ```

   この Apache ウェブサーバー Pod には 500 millicpu の CPU 制限が与えられ、サービスはポート 80 で提供されています。
2. `php-apache` デプロイ用の Horizontal Pod Autoscaler リソースを作成します。

   ```
   kubectl autoscale deployment php-apache --cpu-percent=50 --min=1 --max=10
   ```

   このコマンドは、デプロイの CPU 使用率の目標が 50%、Pod の数が最低 1 つ、最大 10 個の Autoscaler を作成します。CPU の平均負荷が 50% を下回ると、Autoscaler はデプロイの Pod 数

を最小の 1 に減らそうとします。負荷が 50 % を超えると、Autoscaler はデプロイの Pod 数を最大の 10 まで増やそうとします。詳細については、Kubernetes ドキュメントの「How does a HorizontalPodAutoscaler work?」を参照してください。

3. 次のコマンドで Autoscaler の詳細を表示します。

```
kubectl get hpa
```

出力例は次のとおりです。

```
NAME         REFERENCE                TARGETS   MINPODS   MAXPODS   REPLICAS   AGE
php-apache   Deployment/php-apache   0%/50%    1         10        1          51s
```

ご覧のように、現在の CPU 負荷は 0% です。サーバーにまだ負荷がないためです。Pod 数は既に最低の境界 (1) にあるため、スケールインすることはできません。

4. コンテナを実行して、ウェブサーバーのロードを作成します。

```
kubectl run -i \
    --tty load-generator \
    --rm --image=busybox \
    --restart=Never \
    -- /bin/sh -c "while sleep 0.01; do wget -q -O- http://php-apache; done"
```

5. デプロイのスケールアウトを監視するには、前のステップを実行したターミナルとは別のターミナルで次のコマンドを定期的に実行します。

```
kubectl get hpa php-apache
```

出力例は次のとおりです。

```
NAME         REFERENCE                TARGETS    MINPODS   MAXPODS   REPLICAS   AGE
php-apache   Deployment/php-apache   250%/50%   1         10        5          4m44s
```

レプリカ数が増えるまで、1 分以上かかる場合があります。実際の CPU のパーセンテージが目標のパーセンテージよりも高い限り、レプリカの数は最大 10 まで増加します。この場合は 250% のため、REPLICAS の数は増え続けます。

> **Note**
>
> レプリカの数が最大値に達するまで、数分かかる場合があります。例えば、CPU 負荷が 50% 以下に保つために必要なレプリカの数が 6 だけの場合、負荷はレプリカの数 6 を超えてスケールすることはありません。

6. 負荷を停止します。負荷を生成しているターミナルウィンドウで、Ctrl+C キーを押して負荷を停止します。スケールインを監視しているターミナルで次のコマンドを再度実行することで、レプリカの数が 1 にスケールバックされるのを確認できます。

```
kubectl get hpa
```

出力例は次のとおりです。

```
NAME         REFERENCE                TARGETS   MINPODS   MAXPODS   REPLICAS   AGE
php-apache   Deployment/php-apache    0%/50%    1         10        1          25m
```

> **Note**
>
> 現在の CPU の割合が 0% であっても、スケールダウンのデフォルトの時間枠は 5 分であるため、レプリカの数が 1 に再び達するまでには時間がかかります。時間枠は変更可能です。詳細については、Kubernetes ドキュメントの「Horizontal Pod Autoscaler」を参照してください。

7. サンプルアプリケーションのテストが終了したら、php-apache リソースを削除します。

```
kubectl delete deployment.apps/php-apache service/php-apache
 horizontalpodautoscaler.autoscaling/php-apache
```

Network Load Balancer を使用して TCP および UDP トラフィックをルーティングする

> **Note**
>
> 新規: Amazon EKS Auto Mode は、ロードバランシングの定型作業を自動化します。詳細については、以下を参照してください。
>
> - the section called “ロードバランサーをデプロイする”
> - the section called “サービスの作成”

ネットワークトラフィックは、OSI モデルの L4 で負荷分散されます。L7 でアプリケーショントラフィックの負荷を分散するには、Kubernetes `ingress` をデプロイし、これによって AWS Application Load Balancer をプロビジョニングします。詳細については、「the section called “アプリケーション負荷分散”」を参照してください。2 種類の負荷分散の違いについては、AWS ウェブサイトの「Elastic Load Balancing の特徴」を参照してください。

タイプ `LoadBalancer` の Kubernetes `Service` を作成する際、デフォルトでは、AWS クラウドプロバイダーロードバランサーコントローラーにより AWS Classic Load Balancer が作成されますが、AWS Network Load Balancer も作成できます。このコントローラーは、将来の重大なバグ修正のみを受けています。AWS クラウドプロバイダーのロードバランサーコントローラーの使い方については、Kubernetes ドキュメントの「AWS cloud provider load balancer controller」を参照してください。使用方法については、このトピックでは説明しません。

AWS クラウドプロバイダーロードバランサーコントローラーの代わりに、AWS ロードバランサーコントローラー のバージョン `2.7.2` 以降を使用することをお勧めします。AWS Load Balancer Controller では、AWS Network Load Balancer は作成できますが、AWS Classic Load Balancer は作成できません。このトピックの残りの部分では、AWS Load Balancer Controller の使用について説明します。

AWS Network Load Balancer では、Amazon EC2 IP およびインスタンスターゲット、AWS Fargate IP ターゲット、または IP ターゲットとして Amazon EKS Hybrid Nodes にデプロイされた Pod にネットワークトラフィックを負荷分散できます。詳細については、GitHub の AWS Load Balancer Controller を参照してください。

前提条件

AWS Load Balancer Controller を使用してネットワークトラフィックの負荷分散を行うには、次の要件を満たす必要があります。

- 既存のクラスターがある。既存のクラスターがない場合は、「はじめに」を参照してください。既存のクラスターのバージョンを更新する必要がある場合は、the section called “Kubernetes バージョンを更新する” を参照して下さい。
- クラスターに AWS Load Balancer Controller がデプロイされている。詳細については、「the section called “ AWS ロードバランサーコントローラー”」を参照してください。バージョン `2.7.2` 以降をお勧めします。
- 少なくとも 1 つのサブネット。アベイラビリティーゾーンで複数のタグ付きサブネットが見つかった場合、コントローラーは、サブネット ID が辞書式順序で最初に来る最初のサブネットを選択します。サブネットには、利用可能な IP アドレスが最低 8 個必要です。
- AWS Load Balancer Controller バージョン `2.1.1` 以前を使用している場合は、次のようにサブネットにタグ付けする必要があります。バージョン `2.1.2` 以降を使用している場合、このタグはオプションです。同じ VPC 内で複数のクラスターを実行していたり、VPC 内で複数の AWS のサービスがサブネットを共有していたりして、クラスターごとにロードバランサーがプロビジョニングされる場所をより詳細に制御したい場合は、サブネットにタグを付けることができます。サービスオブジェクトのアノテーションとしてサブネット ID を明示的に指定すると、Kubernetes と AWS Load Balancer Controller は、これらのサブネットを直接使用してロードバランサーを作成します。ロードバランサーのプロビジョニングにこの方法を使用することを選択した場合、サブネットのタグ付けは不要です。次のプライベートサブネットおよびパブリックサブネットのタグ付け要件は省略できます。`#######` の部分は自分のクラスター名に置き換えます。
  - キー – `kubernetes.io/cluster/<my-cluster>`
  - 値 – `shared` または `owned`
- サービスオブジェクトまたは Ingress オブジェクトのアノテーションとしてサブネット ID を明示的に指定しない場合、パブリックサブネットとプライベートサブネットは次の要件を満たしている必要があります。サービスオブジェクトやイングレスオブジェクトのアノテーションとしてサブネット ID を明示的に指定してロードバランサーをプロビジョニングする場合、Kubernetes と AWS Load Balancer Controller はこれらのサブネットを直接使用してロードバランサーを作成します。次のタグは必要ありません。
  - プライベートサブネット — 次の形式でタグ付けする必要があります。これは、Kubernetes と AWS Load Balancer Controller が、サブネットを内部ロードバランサーに使用できることを認識できるようするためです。`eksctl` または Amazon EKS AWS CloudFormation テンプレートを

使用して、2020 年 3 月 26 日以降に VPC を作成する場合、サブネットは、作成時に適切にタグ付けされます。Amazon EKS AWS CloudFormation VPC テンプレートの詳細については、「the section called “「VPC を作成する」”」を参照してください。

- キー – kubernetes.io/role/internal-elb
- 値 – 1

- パブリックサブネット — 次の形式でタグ付けする必要があります。これは、各アベイラビリティーゾーンでパブリックサブネットを選択するのではなく (サブネット ID の辞書順)、Kubernetes がそれらのサブネットのみを外部ロードバランサーに使用することを認識できるようにするためです。eksctl または Amazon EKS AWS CloudFormation テンプレートを使用して、2020 年 3 月 26 日以降に VPC を作成する場合、サブネットは、作成時に適切にタグ付けされます。Amazon EKS AWS CloudFormation VPC テンプレートの詳細については、「the section called “「VPC を作成する」”」を参照してください。
  - キー – kubernetes.io/role/elb
  - 値 – 1

サブネットロールタグが明示的に追加されていない場合、Kubernetes サービスコントローラーはクラスター VPC サブネットのルートテーブルを調べて、サブネットがプライベートであるかパブリックであるかを判断します。この動作に頼らず、プライベートまたはパブリックロールタグを明示的に追加することをお勧めします。AWS Load Balancer Controller では、ルートテーブルを検査しないため、自動検出を正常に実行するにはプライベートタグとパブリックタグが必要です。

考慮事項

- ロードバランサーの設定は、サービス用のマニフェストに追加された注釈により制御されます。AWS Load Balancer Controller を使用する場合のサービスアノテーションは、AWS クラウドプロバイダーロードバランサーコントローラーを使用する場合のサービスアノテーションとは異なります。サービスをデプロイする前に、必ず AWS Load Balancer Controller のアノテーションを確認してください。
- Amazon VPC CNI plugin for Kubernetes を使用する場合、AWS Load Balancer Controller は Amazon EC2 の IP ターゲットかインスタンスターゲット、および Fargate の IP ターゲットに負荷分散できます。互換性のある代替 CNI プラグインを使用する場合、コントローラーはインスタンスターゲットにのみ負荷分散できます (Amazon EKS Hybrid Nodes に負荷分散している場合を除く)。ハイブリッドノードの場合、コントローラーは IP ターゲットを負荷分散できます。Network Load Balancer のターゲットタイプの詳細については、「Network Load Balancer のユーザーガイド」の「Target type」(ターゲットタイプ) を参照してください。

- ロードバランサーの作成時 (または作成後) にタグを追加する場合は、サービス仕様に次のアノテーションを追加します。詳細については、AWS Load Balancer Controller ドキュメントのAWS リソースタグを参照してください。

```
service.beta.kubernetes.io/aws-load-balancer-additional-resource-tags
```

- Elastic IP アドレスを Network Load Balancer に割り当てるには、次のアノテーションを追加します。サンプル値を Elastic IP アドレスの `Allocation IDs` に置き換えます。`Allocation IDs` の数は、ロードバランサーに使用されるサブネットの数に一致する必要があります。詳細については、AWS Load Balancer Controller ドキュメントを参照してください。

```
service.beta.kubernetes.io/aws-load-balancer-eip-allocations: eipalloc-
xxxxxxxxxxxxxxxxx,eipalloc-yyyyyyyyyyyyyyyyy
```

- Amazon EKS は、クライアントトラフィック用のインバウンドルールをノードのセキュリティグループに 1 つ追加し、VPC 内のロードバランサーのサブネットごとに、作成する各 Network Load Balancer のヘルスチェック用のルールを 1 つずつ追加します。タイプ `LoadBalancer` のサービスの展開は、Amazon EKS が、セキュリティグループに許可されているルールの最大数に対するクォータを超えるルールを作成しようとした場合に失敗することがあります。詳細については、Amazon VPC ユーザーガイドの「セキュリティグループ」を参照してください。セキュリティグループのルールの最大数を超える可能性を最小限に抑えるために、次のオプションを検討します。
  - セキュリティグループクォータあたりのルールの増加をリクエストする。詳細については、Service Quotas ユーザーガイドの「Requesting a quota increase (クォータの引き上げのリクエスト)」を参照してください。
  - インスタンスターゲットではなく IP ターゲットを使用する。IP ターゲットを使用すると、同じターゲットポートでルールを共有できます。ロードバランサーのサブネットは、アノテーションを使用して手動で指定できます。詳細については、GitHub の 「アノテーション」 を参照してください。
  - タイプ `LoadBalancer` のサービスの代わりにイングレスを使用して、サービスにトラフィックを送信します。AWS Application Load Balancer では、Network Load Balancer よりも必要なルールは少なくなります。複数のイングレス間で ALB を共有できます。詳細については、「the section called “アプリケーション負荷分散”」を参照してください。複数のサービス間で Network Load Balancer を共有することはできません。
  - クラスターを複数のアカウントにデプロイする。

- Pod が Amazon EKS クラスター内の Windows で実行されている場合、ロードバランサーを使用する 1 つのサービスで最大 1,024 個のバックエンド Pod をサポートできます。各ポッドには固有の IP アドレスがあります。
- 新しい Network Load Balancer の作成には、AWS Load Balancer Controller を使用することをおすすめします。AWS クラウドプロバイダーロードバランサーコントローラーで作成された既存の Network Load Balancer を置き換えようとすると、複数の Network Load Balancer でアプリケーションのダウンタイムを引き起こす可能性があります。

ネットワークロードバランサーを作成する

IP またはインスタンスのターゲットを含むネットワークロードバランサーを作成できます。

Network Load Balancer の作成 — IP ターゲット

- Amazon EC2 ノード、Fargate、または Amazon EKS Hybrid Nodes にデプロイされた Pod で IP ターゲットを使用できます。Kubernetes サービスはタイプ `LoadBalancer` として作成する必要があります。詳細については、Kubernetes ドキュメントの「Garbage Collecting Load Balancers (ガベージコレクションを行うロードバランサー」を参照してください。

  IP ターゲットを使用するロードバランサーを作成するには、次のアノテーションをサービスマニフェストに追加し、サービスをデプロイします。`aws-load-balancer-type` の `external` 値により、AWS クラウドプロバイダーロードバランサーコントローラーではなく、AWS Load Balancer Controller によって Network Load Balancer が作成されます。アノテーション付きのサンプルサービスマニフェストを表示できます。

```
service.beta.kubernetes.io/aws-load-balancer-type: "external"
service.beta.kubernetes.io/aws-load-balancer-nlb-target-type: "ip"
```

> **Note**
>
> IPv6 Pod に負荷分散する場合は、次のアノテーションを追加します。IPv6 を使用して負荷分散を行えるのは、IP ターゲットにのみです。インスタンスターゲットには行えません。このアノテーションを使用しない場合、負荷分散には IPv4 が使用されます。

```
service.beta.kubernetes.io/aws-load-balancer-ip-address-type: dualstack
```

デフォルトでは、Network Load Balancer は `internal` `aws-load-balancer-scheme` を使用して作成されます。Network Load Balancer は、クラスターの VPC 内の任意のサブネットで起動できます。これには、クラスターの作成時に指定されていないサブネットも含まれます。

Kubernetes は、サブネットがパブリックかプライベートかどうかをルートテーブルで調べます。パブリックサブネットには、インターネットゲートウェイを使ったインターネットへの直接の経路がありますが、プライベートサブネットにはありません。

パブリックサブネットに Network Load Balancer を作成して Amazon EC2 ノードへの負荷分散を行う場合 (Fargate はプライベートのみ可能)、次のアノテーションを付けて `internet-facing` を指定します。

```
service.beta.kubernetes.io/aws-load-balancer-scheme: "internet-facing"
```

> **Note**
>
> `service.beta.kubernetes.io/aws-load-balancer-type: "nlb-ip"` アノテーションは、下位互換性のために引き続きサポートされています。ただし、新しいロードバランサーには、`service.beta.kubernetes.io/aws-load-balancer-type: "nlb-ip"` の代わりに以前のアノテーションを使用することをお勧めします。

> **Important**
>
> サービスの作成後は、このアノテーションを編集しないでください。変更する必要がある場合は、サービスオブジェクトを削除し、このアノテーションを希望する値にして再度作成します。

Network Load Balancer の作成 — インスタンスターゲット

- AWS クラウドプロバイダーロードバランサーコントローラーでは、インスタンスターゲットのみを使用して Network Load Balancer を作成します。バージョン `2.2.0` 以降の AWS Load Balancer Controller でも同様に、インスタンスターゲットを使用して Network Load Balancer を作成します。新しい Network Load Balancer の作成には、AWS クラウドプロバイダーロードバランサーコントローラーではなく、こちらを使用することをお勧めします。Network Load Balancer のインスタンスターゲットは、Amazon EC2 ノードにデプロイされた Pod では使用できますが、Fargate

にデプロイされた Pod では使用できません。Fargate にデプロイされたポッド間でネットワークトラフィックを負荷分散するには、IP ターゲットを使用する必要があります。

Network Load Balancer をプライベートサブネットにデプロイするには、サービスの仕様に次のアノテーションが必要です。アノテーション付きのサンプルサービスマニフェストを表示できます。`aws-load-balancer-type` の `external` 値により、AWS クラウドプロバイダーロードバランサーコントローラーではなく、AWS Load Balancer Controller によって Network Load Balancer が作成されます。

```
service.beta.kubernetes.io/aws-load-balancer-type: "external"
service.beta.kubernetes.io/aws-load-balancer-nlb-target-type: "instance"
```

デフォルトでは、Network Load Balancer は `internal` `aws-load-balancer-scheme` を使用して作成されます。内部 Network Load Balancer の場合、Amazon EKS クラスターは、VPC 内で少なくとも 1 つのプライベートサブネットを使用するように設定されている必要があります。Kubernetes は、サブネットがパブリックかプライベートかどうかをルートテーブルで調べます。パブリックサブネットには、インターネットゲートウェイを使ったインターネットへの直接の経路がありますが、プライベートサブネットにはありません。

パブリックサブネットに Network Load Balancer を作成して Amazon EC2 ノードへの負荷分散を行う場合、以下のアノテーションを付けて `internet-facing` を指定します。

```
service.beta.kubernetes.io/aws-load-balancer-scheme: "internet-facing"
```

> **Important**
>
> サービスの作成後は、このアノテーションを編集しないでください。変更する必要がある場合は、サービスオブジェクトを削除し、このアノテーションを希望する値にして再度作成します。

(オプション) サンプルアプリケーションをデプロイする

- クラスター VPC に少なくとも 1 つのパブリックサブネットまたはプライベートサブネットが存在する。

- クラスターに AWS Load Balancer Controller がデプロイされている。詳細については、「the section called “ AWS ロードバランサーコントローラー”」を参照してください。バージョン 2.7.2 以降をお勧めします。

1. Fargate にデプロイする場合は、VPC 内に利用可能なプライベートサブネットがあることを確認し、Fargate プロファイルを作成します。Fargate にデプロイしない場合は、このステップを省略してください。次のコマンドを実行するか、コマンドにある AWS マネジメントコンソール と name に同じ値を使用して、namespace でプロファイルを作成できます。example values は実際に使用する値に置き換えます。

```
eksctl create fargateprofile \
    --cluster my-cluster \
    --region region-code \
    --name nlb-sample-app \
    --namespace nlb-sample-app
```

2. サンプルアプリケーションをデプロイします。

   a. アプリケーションの名前空間を作成します。

```
kubectl create namespace nlb-sample-app
```

   b. 次の内容をコンピュータ上の sample-deployment.yaml という名前のファイルに保存します。

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nlb-sample-app
  namespace: nlb-sample-app
spec:
  replicas: 3
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
        - name: nginx
          image: public.ecr.aws/nginx/nginx:1.23
```

```
          ports:
            - name: tcp
              containerPort: 80
```

c. マニフェストをクラスターに適用します。

```
kubectl apply -f sample-deployment.yaml
```

3. IP ターゲットに負荷分散を行うインターネット向け Network Load Balancer を使用して、サービスを作成します。

a. 次の内容をコンピュータ上の sample-service.yaml という名前のファイルに保存します。Fargate ノードにデプロイする場合は、service.beta.kubernetes.io/aws-load-balancer-scheme: internet-facing 行を削除します。

```
apiVersion: v1
kind: Service
metadata:
  name: nlb-sample-service
  namespace: nlb-sample-app
  annotations:
    service.beta.kubernetes.io/aws-load-balancer-type: external
    service.beta.kubernetes.io/aws-load-balancer-nlb-target-type: ip
    service.beta.kubernetes.io/aws-load-balancer-scheme: internet-facing
spec:
  ports:
    - port: 80
      targetPort: 80
      protocol: TCP
  type: LoadBalancer
  selector:
    app: nginx
```

b. マニフェストをクラスターに適用します。

```
kubectl apply -f sample-service.yaml
```

4. サービスがデプロイされたことを確認します。

```
kubectl get svc nlb-sample-service -n nlb-sample-app
```

出力例は次のとおりです。

```
NAME             TYPE           CLUSTER-IP         EXTERNAL-IP
                                                 PORT(S)        AGE
sample-service   LoadBalancer   10.100.240.137   k8s-nlbsampl-nlbsampl-xxxxxxxxxx-
xxxxxxxxxxxxxxxx.elb.region-code.amazonaws.com  80:32400/TCP   16h
```

> **Note**
>
> *10.100.240.137* および *xxxxxxxxxx-xxxxxxxxxxxxxxxx* の値は出力例とは異なります (ロードバランサーに固有のものになります)。*us-west-2* は、クラスターがどの AWS リージョンにあるかによって異なる場合があります。

5. Amazon EC2 AWS マネジメントコンソール を開きます。左側のナビゲーションペインで、([Load Balancing] (ロードバランシング) の下の) [Target Groups] (ターゲットグループ) を選択します。[名前] 列で、前のステップの出力である EXTERNAL-IP 列に入っている名前の一部と [ロードバランサー] 列の値が一致するターゲットグループの名前を選択します 。例えば、出力が以前の出力と同じ場合は、k8s-default-samplese-*xxxxxxxxxx* という名前のターゲットグループを選択します。[Target type] (ターゲットタイプ) は IP です。これは、サービスのサンプルマニフェストで指定されているためです。
6. [ターゲットグループ] を選択し、[ターゲット] タブを選択します。[登録済みターゲット] の下に、前のステップでデプロイされた 3 つのレプリカの 3 つの IP アドレスが表示されます。すべてのターゲットのステータスが [正常] になってから次に進みます。すべてのターゲットが healthy になるまで数分かかることがあります。ターゲットは、healthy 状態に変更される前の unhealthy 状態である可能性があります。
7. *xxxxxxxxxx-xxxxxxxxxxxxxxxx* および *us-west-2* を EXTERNAL-IP の前のステップの出力で返された値に置き換え、サービスにトラフィックを送信します。プライベートサブネットにデプロイした場合は、踏み台ホストなどの VPC 内のデバイスからページを表示する必要があります。詳細については、AWS での Linux 踏み台ホストを参照してください。

```
curl k8s-default-samplese-xxxxxxxxxx-xxxxxxxxxxxxxxxx.elb.region-code.amazonaws.com
```

出力例は次のとおりです。

```
<!DOCTYPE html>
<html>
<head>
<title>Welcome to nginx!</title>
```

```
[...]
```

8. サンプルのデプロイ、サービス、および名前空間の使用が終了したら、それらを削除します。

```
kubectl delete namespace nlb-sample-app
```

Application Load Balancer を使用してアプリケーションと HTTP トラフィックをルーティングする

> **Note**
>
> 新規: Amazon EKS Auto Mode は、ロードバランシングの定型作業を自動化します。詳細については、以下を参照してください。
>
> - the section called “ロードバランサーをデプロイする”
> - the section called “ingress class の作成”

Kubernetes `ingress` を作成するとき、アプリケーショントラフィックの負荷分散を行う AWS Application Load Balancer (ALB) がプロビジョニングされます。詳細については、「Application Load Balancer ユーザーガイド」の「Application Load Balancer とは？」および Kubernetes ドキュメントの「Ingress」を参照してください。ALB は、ノードまたは AWS Fargate にデプロイされたポッドで使用できます。ALB をパブリックサブネットまたはプライベートサブネットにデプロイできます。

アプリケーショントラフィックは、OSI モデルの L7 で負荷分散されます。L4 でネットワークトラフィックの負荷を分散するには、`LoadBalancer` タイプの Kubernetes `service` をデプロイします このタイプは、 AWS Network Load Balancer をプロビジョニングします。詳細については、「the section called “ネットワーク負荷分散”」を参照してください。2 種類の負荷分散の違いについては、AWS ウェブサイトの「Elastic Load Balancing の特徴」を参照してください。

前提条件

あるアプリケーションに対してアプリケーショントラフィックの負荷分散を行うには、次の要件を満たす必要があります。

- 既存のクラスターがある。既存のクラスターがない場合は、「はじめに」を参照してください。既存のクラスターのバージョンを更新する必要がある場合は、the section called “Kubernetes バージョンを更新する” を参照して下さい。
- クラスターに AWS Load Balancer Controller がデプロイされている。詳細については、「the section called “ AWS ロードバランサーコントローラー”」を参照してください。バージョン 2.7.2 以降をお勧めします。
- 少なくとも 2 つのサブネットが異なるアベイラビリティーゾーンに存在している。AWS Load Balancer Controller は、各アベイラビリティーゾーンから 1 つずつサブネットを選択します。タグ付けされたサブネットがアベイラビリティーゾーンで複数見つかった場合、コントローラーは、サブネット ID が辞書式順序で最初に来るサブネットを選択します。各サブネットには最低 8 個の利用可能な IP アドレスが必要です。

  ワーカーノードにアタッチされた複数のセキュリティグループを使用している場合は、次のように 1 つのセキュリティグループにタグを付ける必要があります。`#######` の部分は自分のクラスター名に置き換えます。

  - キー – `kubernetes.io/cluster/<my-cluster>`
  - 値 – `shared` または `owned`
- AWS Load Balancer Controller バージョン 2.1.1 以前を使用している場合、サブネットに次のようにタグ付けする必要があります。バージョン 2.1.2 以降を使用している場合、このタグ付けはオプションです。ただし、次のいずれかの場合は、サブネットにタグ付けることをお勧めします。同じVPCで実行されている複数のクラスターがあるか、VPCでサブネットを共有する複数の AWS サービスがあります。または、各クラスターに対してロードバランサーをプロビジョニングする場所を詳細に制御する必要があります。`#######` の部分は自分のクラスター名に置き換えます。

  - キー – `kubernetes.io/cluster/<my-cluster>`
  - 値 – `shared` または `owned`
- パブリックサブネットとプライベートサブネットは次の要件を満たしている必要があります。サービスオブジェクトまたは Ingress オブジェクトのアノテーションとしてサブネット ID を明示的に指定しない限り、これは例外です。サービスオブジェクトまたは Ingress オブジェクトのアノテーションとしてサブネット ID を明示的に指定して、ロードバランサーをプロビジョニングすることを想定しています。このような状況では、Kubernetesと AWS Load Balancer Controller は、これらのサブネットを直接使用してロードバランサーを作成します。次のタグは必要ありません。

  - プライベートサブネット — 次の形式でタグ付けする必要があります。これは、Kubernetes と AWS ロードバランサーコントローラーが、サブネットを内部ロードバランサーに使用できることを認識できるようにするためです。`eksctl` または Amazon EKS AWS CloudFormation テンプレートを使用して、2020 年 3 月 26 日以降に VPC を作成する場合、サブネットは、作成時に

適切にタグ付けされます。Amazon EKS AWS CloudFormation VPC テンプレートの詳細については、「the section called “「VPC を作成する」”」を参照してください。

- キー – kubernetes.io/role/internal-elb
- 値 – 1

- パブリックサブネット — 次の形式でタグ付けする必要があります。これは、Kubernetes は、外部ロードバランサー用に指定されたサブネットのみを使用できることを認識できるようにするためです。この方法では、Kubernetes は各アベイラビリティーゾーンでパブリックサブネットを (サブネット ID に基づいて辞書順に) 選択しません eksctl または Amazon EKS AWS CloudFormation テンプレートを使用して、2020 年 3 月 26 日以降に VPC を作成する場合、サブネットは、作成時に適切にタグ付けされます。Amazon EKS AWS CloudFormation VPC テンプレートの詳細については、「the section called “「VPC を作成する」”」を参照してください。
  - キー – kubernetes.io/role/elb
  - 値 – 1

サブネットロールタグが明示的に追加されていない場合、Kubernetes サービスコントローラーはクラスター VPC サブネットのルートテーブルを調べます。これは、サブネットがプライベートかパブリックかを判断するためです。この動作に頼らないことをお勧めします。代わりに、プライベートまたはパブリックロールタグを明示的に追加します。AWS Load Balancer Controller は、ルートテーブルを検査しません。また、自動検出を正常に実行するには、プライベートタグとパブリックタグが必要です。

- Kubernetes のイングレスリソースが kubernetes.io/ingress.class: alb アノテーションを使用してクラスターで作成されるたびに、AWS Load Balancer Controller により ALB およびサポートされる必要な AWS リソースが作成されます。イングレスリソースは、HTTP または HTTPS トラフィックをクラスター内の別の Pod にルーティングするように ALB を設定します。イングレスオブジェクトで AWS Load Balancer Controller を使用できるようにするには、Kubernetes イングレス仕様に次のアノテーションを追加します。詳細については、GitHub の「Ingress specification (Ingress 仕様)」を参照してください。

```
annotations:
    kubernetes.io/ingress.class: alb
```

> **Note**
>
> IPv6 Pod への負荷分散を行う場合は、イングレス仕様に次のアノテーションを追加します。IPv6 を使用して負荷分散を行えるのは、IP ターゲットにのみです。インスタンス

ターゲットには行えません。このアノテーションを使用しない場合、負荷分散には IPv4 が使用されます。

```
alb.ingress.kubernetes.io/ip-address-type: dualstack
```

- AWS Load Balancer Controller は、次のトラフィックモードをサポートしています。
  - インスタンス – クラスター内のノードを ALB のターゲットとして登録します。ALB に到達するトラフィックは、サービスの NodePort にルーティングされてから、Pod にプロキシされます。これがデフォルトのトラフィックモードです。`alb.ingress.kubernetes.io/target-type: instance` の注釈を使用して明示的に指定することもできます。

    Note

    Kubernetes サービスでは、このトラフィックモードを使用する NodePort または LoadBalancer タイプを指定する必要があります。

  - IP - ポッドを ALB のターゲットとして登録します。ALB に到達するトラフィックは、サービスの Pod に直接ルーティングされます。このトラフィックモードを使用するには、`alb.ingress.kubernetes.io/target-type: ip` の注釈を指定する必要があります。IP ターゲットタイプは、ターゲット Pod が Fargate または Amazon EKS Hybrid Nodes で実行されている場合には必須です。
- コントローラーによって作成された ALB にタグを付けるには、次のアノテーションをコントローラーに追加します: `alb.ingress.kubernetes.io/tags`。AWS Load Balancer Controller でサポートされるすべての使用可能なアノテーションのリストについては、GitHub の「Ingress アノテーション」を参照してください。
- ALB コントローラーのバージョンをアップグレードまたはダウングレードすると、ALB コントローラーのバージョンに依存する機能が大きく変更される可能性があります。各リリースで導入された新しい変更の詳細については、GitHub で ALB Controller リリースノートを参照してください。

イングレスグループで ALB を再利用する

IngressGroups を使用して複数のサービスのリソース間で Application Load Balancer を共有できます。

イングレスをグループに入れるには、Kubernetes のイングレスリソース仕様に次のアノテーションを追加します。

```
alb.ingress.kubernetes.io/group.name: my-group
```

グループ名は、次のようにする必要があります。

- 長さが 63 文字以下。
- 名前は、小文字の英文字、-、および . で構成されます。
- 数字または文字で始めます。

コントローラーは、同じイングレスグループ内のすべてのイングレスのイングレスルールを自動的にマージします。これは、単一のALBでそれらをサポートしています。イングレスで定義されたほとんどのアノテーションは、そのイングレスで定義されたパスにのみ適用されます。デフォルトでは、イングレスリソースはどのイングレスグループにも属していません。

> **Warning**
>
> 潜在的なセキュリティリスク
> イングレスリソースを作成または変更する RBAC 権限を持つすべての Kubernetes ユーザーが同じ信頼境界内にいる場合にのみ、イングレスにイングレスグループを指定します。アノテーションをグループ名で追加すると、他の Kubernetes ユーザーが、同じイングレスグループに属するイングレスを作成または変更する可能性があります。これにより、優先度の高いルールで既存のルールを上書きするなど、望ましくない動作が発生する可能性があります。

イングレスリソースの順序番号を追加できます。

```
alb.ingress.kubernetes.io/group.order: '10'
```

番号は 1 ~ 1000 の範囲で指定できます。同じイングレスグループ内のすべてのイングレスについて、最小の番号を持つものが最初に評価されます。このアノテーションがないイングレスはすべて、値 0 で評価されます。番号が小さいルールと番号が大きいルールが重複すると、番号が小さいルールが上書きされる可能性があります。デフォルトでは、同じイングレスグループ内のイングレス間のルールの順序は、名前空間と名前に基づき辞書式順序で決定されます。

> **Important**
>
> 同じイングレスグループの各イングレスに、一意のプライオリティ番号があることを確認します。　イングレス間で重複する順序番号を持つことはできません。

(オプション) サンプルアプリケーションをデプロイする

- クラスター VPC に少なくとも 1 つのパブリックサブネットまたはプライベートサブネットが存在する。
- クラスターに AWS Load Balancer Controller がデプロイされている。詳細については、「the section called “ AWS ロードバランサーコントローラー”」を参照してください。バージョン 2.7.2 以降をお勧めします。

Amazon EC2 ノード、Fargate ポッド、またはその両方を持つクラスターで、サンプルアプリケーションを実行できます。

1. Fargate にデプロイしない場合は、このステップを省略してください Fargate にデプロイする場合は、Fargate プロファイルを作成します。次のコマンドを実行するか、コマンドにある AWS マネジメントコンソール と name に同じ値を使用して、namespace でプロファイルを作成できます。example values は実際に使用する値に置き換えます。

   ```
   eksctl create fargateprofile \
       --cluster my-cluster \
       --region region-code \
       --name alb-sample-app \
       --namespace game-2048
   ```

2. サンプルアプリケーションとしてゲーム 2048 をデプロイし、AWS Load Balancer Controller がイングレスオブジェクトの結果として AWS ALB を作成することを確認します。デプロイ先のサブネットのタイプに関する手順を実行します。

   a. IPv6 ファミリーを使用して作成したクラスター内の Pod にデプロイしている場合には、次のステップに進みます。

      - Public::

      ```
      kubectl apply -f https://raw.githubusercontent.com/kubernetes-sigs/aws-load-balancer-controller/v2.14.1/docs/examples/2048/2048_full.yaml
      ```

- プライベート::

  A. マニフェストをダウンロードします。

  ```
  curl -O https://raw.githubusercontent.com/kubernetes-sigs/aws-load-balancer-controller/v2.14.1/docs/examples/2048/2048_full.yaml
  ```

  B. ファイルを編集して、alb.ingress.kubernetes.io/scheme: internet-facing と記述された行を探します。

  C. *########*を internal に変更し、ファイルを保存します。

  D. マニフェストをクラスターに適用します。

  ```
  kubectl apply -f 2048_full.yaml
  ```

b. IPv6 ファミリーを使用して作成したクラスター内の Pod にデプロイする場合、次の手順を完了します。

  i. マニフェストをダウンロードします。

  ```
  curl -O https://raw.githubusercontent.com/kubernetes-sigs/aws-load-balancer-controller/v2.14.1/docs/examples/2048/2048_full.yaml
  ```

  ii. エディタでファイルを開き、イングレス仕様のアノテーションに次の行を追加します。

  ```
  alb.ingress.kubernetes.io/ip-address-type: dualstack
  ```

  iii. インターネット向けの Pod ではなく、内部の Pod への負荷分散を行う場合は、alb.ingress.kubernetes.io/scheme: *internet-facing* という行を alb.ingress.kubernetes.io/scheme: internal に変更します。

  iv. ファイルを保存します。

  v. マニフェストをクラスターに適用します。

  ```
  kubectl apply -f 2048_full.yaml
  ```

3. 数分後、次のコマンドを使用して、イングレスリソースが作成されたことを確認します。

```
kubectl get ingress/ingress-2048 -n game-2048
```

出力例は次のとおりです。

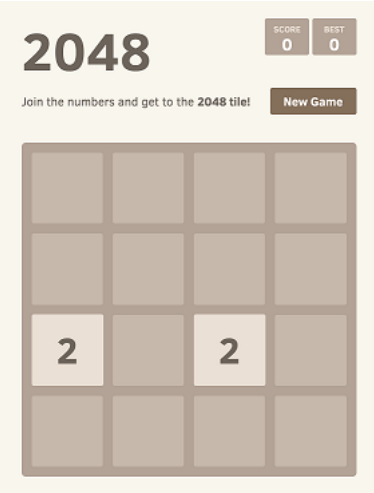
```
NAME           CLASS    HOSTS   ADDRESS
                     PORTS   AGE
ingress-2048   <none>   *       k8s-game2048-ingress2-xxxxxxxxxx-yyyyyyyyyy.region-
code.elb.amazonaws.com   80      2m32s
```

> **Note**
>
> プライベートサブネットでロードバランサーを作成した場合、前の出力の ADDRESS の下の値の前に `internal-` が付けられます。

イングレスが数分後に作成されていない場合は、次のコマンドを実行して AWS Load Balancer Controller のログを表示します。これらのログには、デプロイに関する問題の診断に使用できるエラーメッセージが含まれている場合があります。

```
kubectl logs -f -n kube-system -l app.kubernetes.io/instance=aws-load-balancer-
controller
```

1. パブリックサブネットにデプロイした場合は、ブラウザを開き、前のコマンドの出力から ADDRESS URL に移動してサンプルアプリケーションを表示します。何も表示されない場合は、ブラウザを更新してからもう一度試してください。プライベートサブネットにデプロイした場合は、踏み台ホストなどの VPC 内のデバイスからページを表示する必要があります。詳細については、AWS での Linux 踏み台ホストを参照してください。



2. サンプルアプリケーションの試用が終了したら、次のコマンドのいずれかを実行して削除します。

- ダウンロードしたコピーを適用するのではなくマニフェストを適用した場合は、次のコマンドを使用します。

```
kubectl delete -f https://raw.githubusercontent.com/kubernetes-sigs/aws-load-balancer-controller/v2.14.1/docs/examples/2048/2048_full.yaml
```

- マニフェストをダウンロードして編集した場合は、次のコマンドを使用します。

```
kubectl delete -f 2048_full.yaml
```

サービスに割り当てることができる外部IPアドレスを制限する

Kubernetes サービスには、クラスターの内部から次を経由することで到達できます。

- Kubernetes によって自動的に割り当てられるクラスター IP アドレス
- サービス仕様で externalIPs プロパティに指定した任意の IP アドレス 外部 IP アドレスは Kubernetes によって管理されるのではなく、クラスターの管理者が管理します。externalIPs で指定される外部 IP アドレスは、クラウドプロバイダーによって提供される LoadBalancer タイプのサービスに割り当てられる外部 IP アドレスとは異なります。

Kubernetes サービスの詳細については、Kubernetes ドキュメントの「サービス」を参照してください。サービス仕様で externalIPs に指定できる IP アドレスを制限できます。

1. webhook の証明書を管理するにはcert-manager をデプロイします。詳細については「cert-manager」(証明書マネージャー) ドキュメントを参照してください。

```
kubectl apply -f https://github.com/jetstack/cert-manager/releases/download/v1.5.4/cert-manager.yaml
```

2. cert-manager Pod が実行中であることを確認します。

```
kubectl get pods -n cert-manager
```

出力例は次のとおりです。

```
NAME                                       READY   STATUS    RESTARTS   AGE
cert-manager-58c8844bb8-nlx7q              1/1     Running   0          15s
cert-manager-cainjector-745768f6ff-696h5   1/1     Running   0          15s
```

```
cert-manager-webhook-67cc76975b-4v4nk      1/1     Running   0          14s
```

3. 既存のサービスを確認して、アドレスを制限する CIDR ブロックに含まれない外部 IP アドレスが割り当てられていないことを確認します。

```
kubectl get services -A
```

出力例は次のとおりです。

```
NAMESPACE                   NAME                                    TYPE
 CLUSTER-IP       EXTERNAL-IP     PORT(S)         AGE
cert-manager                cert-manager                            ClusterIP
 10.100.102.137   <none>          9402/TCP        20m
cert-manager                cert-manager-webhook                    ClusterIP
 10.100.6.136     <none>          443/TCP         20m
default                     kubernetes                              ClusterIP
 10.100.0.1       <none>          443/TCP         2d1h
externalip-validation-system   externalip-validation-webhook-service   ClusterIP
 10.100.234.179   <none>          443/TCP         16s
kube-system                 kube-dns                                ClusterIP
 10.100.0.10      <none>          53/UDP,53/TCP   2d1h
my-namespace                my-service                              ClusterIP
 10.100.128.10    192.168.1.1     80/TCP          149m
```

いずれかの値が、アクセスを制限するブロック内にない IP アドレスである場合はブロック内に存在するアドレスを変更し、サービスを再びデプロイする必要があります。例えば、前の出力の `my-service` サービスにはステップ 5 の CIDR ブロックの例に含まれていない外部 IP アドレスが割り当てられています。

4. 外部 IP のウェブフックマニフェストをダウンロードします。また、GitHub でウェブフックのソースコードを確認できます。

```
curl -O https://s3.us-west-2.amazonaws.com/amazon-eks/docs/externalip-webhook.yaml
```

5. CIDR ブロックを指定します。ダウンロードしたファイルをエディタで開き、次の行の最初の `\#` を削除します。

```
#args:
#- --allowed-external-ip-cidrs=10.0.0.0/8
```

10.0.0.0/8 を独自の CIDR ブロックに置き換えます。ブロックはいくつでも指定できます。複数のブロックを指定する場合は、ブロック間にコンマを追加します。

6. クラスターが us-west-2 AWS に存在しない場合、ファイル内の us-west-2、602401143452、および amazonaws.com を次のコマンドに置き換えます。コマンドを実行する前に、*#####* および *111122223333* を「Amazon EKS アドオンの Amazon コンテナイメージレジストリを表示する」のリストから AWS リージョンの値に置き換えます。

```
sed -i.bak -e 's|602401143452|111122223333|' externalip-webhook.yaml
sed -i.bak -e 's|us-west-2|region-code|' externalip-webhook.yaml
sed -i.bak -e 's|amazonaws.com||' externalip-webhook.yaml
```

7. マニフェストをクラスターに適用します。

```
kubectl apply -f externalip-webhook.yaml
```

「CIDR ブロックを指定します」のステップで指定したブロックに含まれていない、externalIPs に指定された IP アドレスを使用して、サービスをクラスターにデプロイしようとすると、失敗します。

あるリポジトリから別のリポジトリにコンテナイメージをコピーする

このトピックでは、ノードがアクセスできないリポジトリからコンテナイメージをプルし、ノードがアクセスできるリポジトリにイメージをプッシュする方法について説明します。イメージを、Amazon ECR またはノードがアクセスできる代替リポジトリにプッシュできます。

- コンピュータにインストールされ設定されている Docker エンジン。手順については、Docker ドキュメントの「Docker Engine のインストール」を参照してください。
- ご使用のデバイスまたは AWS CloudShell で、バージョン 2.12.3 以降、または AWS コマンドラインインターフェイス (AWS CLI) のバージョン 1.27.160 以降がインストールおよび設定されていること。現在のバージョンを確認するには、aws --version | cut -d / -f2 | cut -d ' ' -f1 を参照してください。yum、apt-get、macOS 用の Homebrew などのパッケージマネージャーは、多くの場合 AWS CLI の最新バージョンより数バージョン古くなっています。最新バージョンをインストールするには、「 AWS コマンドラインインターフェイスユーザーガイド」の「インストール」および「aws configure を使用したクイック設定」を参照してください。AWS クラウドシェル にインストールされている AWS CLI バージョンも最新バージョンより数バージョ

ン遅れることがあります。更新するには、「AWS CloudShell ユーザーガイド」の「ホームディレクトリへの AWS CLI のインストール」を参照してください。

- ノードで Amazon のネットワークを介してプライベート Amazon ECR リポジトリからコンテナイメージのプルを行ったり、コンテナイメージをプッシュする場合は、Amazon ECR のインターフェイス VPC エンドポイント。詳細については、Amazon Elastic コンテナレジストリ ユーザーガイドの 「Amazon ECR 用の VPC エンドポイントを作成する」 を参照してください。

リポジトリからコンテナイメージをプルし、独自のリポジトリにプッシュする場合は、次の手順を完了してください。このトピックで紹介する次の例では、Amazon VPC CNI Plugin for Kubernetes メトリクスヘルパーのイメージがプルされます。これらのステップを実行する場合は、必ずexample values を置き換えてください。

1. Amazon ECR リポジトリまたは別のリポジトリがまだ作成されていない場合は、ノードがアクセスできるリポジトリを作成します。次のコマンドは、Amazon ECR プライベートリポジトリを作成します。Amazon ECR プライベートリポジトリ名は文字で始まる必要があります。小文字、数字、ハイフン (-)、アンダースコア (_)、フォワードスラッシュ (/) のみ含むことができます。詳細については、Amazon Elastic Container Registry ユーザーガイドの「プライベートリポジトリの作成」を参照してください。

   cni-metrics-helper を選択したものに置き換えることができます。ベストプラクティスとして、各イメージに個別のリポジトリを作成します。イメージタグはリポジトリ内で一意である必要があるため、この方法をお勧めします。*region-code* を Amazon ECR でサポートされている AWS リージョンに置き換えます。

   ```
   aws ecr create-repository --region region-code --repository-name cni-metrics-helper
   ```

2. ノードがプルする必要があるイメージのレジストリ、リポジトリ、およびタグ (オプション) を決めます。この情報は `registry/repository[:tag]` 形式です。

   Amazon EKS のイメージのインストールに関するトピックの多くは、マニフェストファイルを適用するか、Helm チャートを使用してイメージをインストールする必要があります。ただし、マニフェストファイルの適用や、Helm チャートをインストールする前に、まずマニフェストやチャートの `values.yaml` ファイルの内容を確認してください そうすることで、プルするレジストリ、リポジトリ、およびタグを決定することができます。

   例えば、Amazon VPC CNI plugin for Kubernetes メトリクスヘルパーのマニフェストファイルに次の行があります。`602401143452.dkr.ecr.us-west-2.amazonaws.com` がレジストリで、

これは Amazon ECR のプライベートレジストリです。cni-metrics-helper が、レポジトリです。

```
image: "602401143452.dkr.ecr.us-west-2.amazonaws.com/cni-metrics-helper:v1.12.6"
```

イメージの場所として、次のバリエーションがあります:

- repository-name:tag のみ。この場合、通常 docker.io はレジストリですが、レジストリが指定されていない場合は、Kubernetes がデフォルトでリポジトリ名の前にレジストリをプリペンドするため、指定されていません。
- repository-name/repository-namespace/repository:tag. リポジトリの名前空間は任意ですが、イメージを分類するためにリポジトリ所有者が指定される場合があります。例えば、Amazon ECR Public Gallery にあるすべての Amazon EC2 イメージは、aws-ec2 名前空間を使用します。

  Helm を使用してイメージをインストールする前に、Helm values.yaml ファイルを表示してイメージの場所を決定します。例えば、Amazon VPC CNI plugin for Kubernetes メトリクスヘルパーの values.yaml ファイルに次の行があります。

  ```
  image:
    region: us-west-2
    tag: v1.12.6
    account: "602401143452"
    domain: "amazonaws.com"
  ```

3. マニフェストファイルで指定されたコンテナイメージをプルします。

   a. Amazon ECR Public Gallery など、パブリックレジストリからプルする場合は、認証が不要なため、次のサブステップにスキップできます。この例では、CNI メトリクスヘルパーイメージのリポジトリを含む Amazon ECR プライベートレジストリに対して認証します。Amazon EKS は、「Amazon EKS アドオンの Amazon コンテナイメージレジストリを表示する」に記載されている各レジストリにイメージを保持します。*602401143452* と *region-code* を別のレジストリの情報に置き換えることで、任意のレジストリに対して認証が可能です。Amazon EKS がサポートされている AWS リージョン には、それぞれに個別のレジストリが存在します。

   ```
   aws ecr get-login-password --region region-code | docker login --username AWS --
   password-stdin 602401143452.dkr.ecr.region-code.amazonaws.com
   ```

   b. イメージをプルします。この例では、前のサブステップで認証したレジストリからプルします。*602401143452* と *region-code* を前のサブステップで指定した情報に置き換えます。

```
docker pull 602401143452.dkr.ecr.region-code.amazonaws.com/cni-metrics-
helper:v1.12.6
```

4. プルしたイメージに、レジストリ、リポジトリ、タグでタグ付けします。次の例では、マニフェストファイルからイメージをプルし、最初のステップで作成した Amazon ECR プライベートリポジトリにイメージをプッシュすることを前提としています。*111122223333* は、ご自分のアカウント ID に置き換えます。*region-code* を、Amazon ECR プライベートリポジトリを作成した AWS リージョンに置き換えます。

```
docker tag cni-metrics-helper:v1.12.6 111122223333.dkr.ecr.region-code.amazonaws.com/
cni-metrics-helper:v1.12.6
```

5. レジストリに対して認証します。この例では、最初のステップで作成した Amazon ECR プライベートレジストリに対して認証します。詳細については、Amazon Elastic Container Registry ユーザーガイドの「レジストリの認証」を参照してください。

```
aws ecr get-login-password --region region-code | docker login --username AWS --
password-stdin 111122223333.dkr.ecr.region-code.amazonaws.com
```

6. リポジトリにプッシュするイメージを付けます。この例では、最初のステップで作成した Amazon ECR プライベートリポジトリにイメージをプッシュします。詳細については、Amazon Elastic Container Registry ユーザーガイドの「Docker イメージをプッシュする」を参照してください。

```
docker push 111122223333.dkr.ecr.region-code.amazonaws.com/cni-metrics-helper:v1.12.6
```

7. プッシュしたイメージ用の `registry/repository:tag` を使用して前のステップでイメージの決定に使用したマニフェストファイルを更新します。Helm チャートを使用してインストールする場合、`registry/repository:tag` を指定するオプションが多くあります。チャートをインストールする場合、リポジトリにプッシュしたイメージの `registry/repository:tag` を指定します。

Amazon EKS アドオンの Amazon コンテナイメージレジストリを表示する

AWS Amazon EKS アドオンをクラスターにデプロイすると、ノードはアドオンのインストールメカニズムで指定されたレジストリ (インストールマニフェストや Helm `values.yaml` ファイルな

ど から必要なコンテナイメージを取得します。画像はAmazon EKS Amazon ECR プライベートリポジトリからプルします。Amazon EKS は Amazon EKSがサポートする各 AWS リージョンのリポジトリにイメージをレプリケートします。ノードは次のレジストリのいずれかからインターネット経由でコンテナイメージをプルできます。またはVPC 内に Amazon ECR (AWS プライベートリンク 用のインターフェイス VPC エンドポイントを作成した場合、ノードは Amazon のネットワーク経由でイメージをプルできます。レジストリにはAWS IAM アカウントでの認証が必要です。ノードは Amazon EKS ノードの IAM ロールを使用して認証します。このロールには関連付けられた AmazonEC2ContainerRegistryReadOnly で管理される IAM ポリシーのアクセス許可があります。

AWS リージョン	レジストリ
af-south-1	877085696533.dkr.ecr.af-south-1.amazonaws.com
ap-east-1	800184023465.dkr.ecr.ap-east-1.amazonaws.com
ap-east-2	533267051163.dkr.ecr.ap-east-2.amazonaws.com
ap-southeast-3	296578399912.dkr.ecr.ap-southeast-3.amazonaws.com
ap-south-2	900889452093.dkr.ecr.ap-south-2.amazonaws.com
ap-southeast-4	491585149902.dkr.ecr.ap-southeast-4.amazonaws.com
ap-southeast-5	151610086707.dkr.ecr.ap-southeast-5.amazonaws.com
ap-southeast-6	333609536671.dkr.ecr.ap-southeast-6.amazonaws.com
ap-south-1	602401143452.dkr.ecr.ap-south-1.amazonaws.com

AWS リージョン	レジストリ
ap-northeast-3	602401143452.dkr.ecr.ap-northeast-3.amazonaws.com
ap-northeast-2	602401143452.dkr.ecr.ap-northeast-2.amazonaws.com
ap-southeast-1	602401143452.dkr.ecr.ap-southeast-1.amazonaws.com
ap-southeast-2	602401143452.dkr.ecr.ap-southeast-2.amazonaws.com
ap-southeast-7	121268973566.dkr.ecr.ap-southeast-7.amazonaws.com
ap-northeast-1	602401143452.dkr.ecr.ap-northeast-1.amazonaws.com
cn-north-1	918309763551.dkr.ecr.cn-north-1.amazonaws.com.cn
cn-northwest-1	961992271922.dkr.ecr.cn-northwest-1.amazonaws.com.cn
eu-central-1	602401143452.dkr.ecr.eu-central-1.amazonaws.com
eu-west-1	602401143452.dkr.ecr.eu-west-1.amazonaws.com
eu-west-2	602401143452.dkr.ecr.eu-west-2.amazonaws.com
eu-south-1	590381155156.dkr.ecr.eu-south-1.amazonaws.com
eu-west-3	602401143452.dkr.ecr.eu-west-3.amazonaws.com

AWS リージョン	レジストリ
eu-south-2	455263428931.dkr.ecr.eu-south-2.amazonaws.com
eu-north-1	602401143452.dkr.ecr.eu-north-1.amazonaws.com
eu-central-2	900612956339.dkr.ecr.eu-central-2.amazonaws.com
il-central-1	066635153087.dkr.ecr.il-central-1.amazonaws.com
mx-central-1	730335286997.dkr.ecr.mx-central-1.amazonaws.com
me-south-1	558608220178.dkr.ecr.me-south-1.amazonaws.com
me-central-1	759879836304.dkr.ecr.me-central-1.amazonaws.com
us-east-1	602401143452.dkr.ecr.us-east-1.amazonaws.com
us-east-2	602401143452.dkr.ecr.us-east-2.amazonaws.com
us-west-1	602401143452.dkr.ecr.us-west-1.amazonaws.com
us-west-2	602401143452.dkr.ecr.us-west-2.amazonaws.com
ca-central-1	602401143452.dkr.ecr.ca-central-1.amazonaws.com
ca-west-1	761377655185.dkr.ecr.ca-west-1.amazonaws.com

AWS リージョン	レジストリ
sa-east-1	602401143452.dkr.ecr.sa-east-1.amazonaws.com
us-gov-east-1	151742754352.dkr.ecr.us-gov-east-1.amazonaws.com
us-gov-west-1	013241004608.dkr.ecr.us-gov-west-1.amazonaws.com
eusc-de-east-1	877088126301.dkr.ecr.eusc-de-east-1.amazonaws.eu

Amazon EKS アドオン

アドオンは、Kubernetes アプリケーションをサポートする操作機能を提供するソフトウェアですが、アプリケーションに固有のものではありません。これには、監視エージェントや Kubernetes ドライバなどのソフトウェアが含まれます。これらのソフトウェアにより、クラスターは、ネットワーク、コンピューティング、およびストレージ用の基盤となる AWS リソースと相互作用できるようになります。アドオンソフトウェアは、通常、Kubernetes コミュニティや AWS のようなクラウドプロバイダ、またはサードパーティーベンダーによって構築、保守されます。Amazon EKS は、Amazon VPC CNI plugin for Kubernetes、`kube-proxy`、および、CoreDNS などのセルフマネージド型アドオンを、すべてのクラスターで自動的にインストールします。VPC CNI アドオンは Amazon EKS Hybrid Nodes と互換性がなく、ハイブリッドノードにデプロイされないことに注意してください。必要に応じて、アドオンのデフォルト設定は変更と更新が可能です。

Amazon EKS アドオンはAmazon EKS クラスター用に厳選されたアドオンセットのインストールと管理を実行してください。Amazon EKS アドオンには最新のセキュリティパッチ、バグ修正が含まれており、また Amazon EKS と連携するために AWS により検証されています。Amazon EKS アドオンを使用すると、Amazon EKS クラスターの安全性と安定性を一貫して保証でき、アドオンのインストール、設定、および更新に必要となる作業量を削減できます。セルフマネージド型のアドオン (`kube-proxy` など) がすでにクラスターで実行されており、Amazon EKS アドオンとして利用可能な場合、`kube-proxy` Amazon EKS アドオンをインストールすれば、その機能からのメリットを活用できるようになります。

Amazon EKS API を使用することで、Amazon EKS アドオンのための、Amazon EKS 管理の特定の設定フィールドを更新できます。Amazon EKS で管理されていない設定フィールドは、アドオンの

起動後に Kubernetes クラスター内で直接変更することもできます。これには、アドオン向けの特定の設定フィールドの定義が含まれます (該当する場合)。これらの変更は一度実行されると Amazon EKS によって上書きされることはありません。これは、Kubernetes によるサーバー側の適用機能を使用して実現しています。詳細については、the section called “カスタマイズ可能なフィールド” を参照してください。

Amazon EKS アドオンはどの Amazon EKS ノードタイプでも使用できます。詳細については、コンピューティングを管理する を参照してください。

Amazon EKS アドオンはAmazon EKS API、AWS マネジメントコンソール、AWS CLI、および `eksctl` を使用して、追加、更新、または削除ができます。Amazon EKS アドオンはAWS クラウドフォーメーション を使用しても作成できます。

考慮事項

Amazon EKS アドオンを使用する場合は次の点を考慮してください:

- クラスターでアドオンの設定を行うにはアドオンを操作するための IAM 許可を、IAM プリンシパルが持っている必要があります。詳細については「Amazon Elastic Kubernetes Service で定義されるアクション」で、Addon の名前ごとにアクションを参照してください。
- Amazon EKS アドオンはクラスター用にプロビジョニングまたは設定するノードで実行されます。ノードタイプには Amazon EC2 インスタンス、Fargate、ハイブリッドノードがあります。
- Amazon EKS で管理されていないフィールドを変更して、Amazon EKS アドオンのインストールをカスタマイズできます。詳細については、「the section called “カスタマイズ可能なフィールド”」を参照してください。
- AWS マネジメントコンソール を使用してクラスターを作成すると、AWS マネジメントコンソール は Amazon EKS `kube-proxy`、Amazon VPC CNI plugin for Kubernetes、および CoreDNS アドオンをクラスターに自動的に追加します。`config` ファイルなしで `eksctl` を使用してクラスターを作成すると、`eksctl` はバージョン 0.184.0 以降の Amazon EKS アドオンとして `kube-proxy`、Amazon VPC CNI plugin for Kubernetes、CoreDNS、および metrics-server をインストールします。以前のバージョンの `eksctl` の場合、または他のツールを使用する場合は、代わりにセルフマネージド `kube-proxy`、Amazon VPC CNI plugin for Kubernetes、および CoreDNS アドオンがインストールされます。これらは自分で管理することが可能で、また、クラスターの作成後に Amazon EKS アドオンを手動で追加することもできます。クラスターの作成にどの方法を使用するかにかかわらず、Amazon VPC CNI plugin for Kubernetes アドオンはハイブリッドノードにインストールされません。

- eks:addon-cluster-admin ClusterRoleBinding は cluster-admin ClusterRole を eks:addon-manager Kubernetes の ID にバインドします。このロールには eks:addon-manager ID が Kubernetes 名前空間を作成し、アドオンを名前空間にインストールするために必要なアクセス許可があります。eks:addon-cluster-admin ClusterRoleBinding が削除されても、Amazon EKS クラスターは引き続き機能しますが、Amazon EKS はアドオンを管理できなくなります。次のプラットフォームバージョンから始まるすべてのクラスターは新しい ClusterRoleBinding バージョンを使用します。
- AWS からの EKS アドオンのサブセットは Amazon EKS Hybrid Nodes との互換性が検証されています。詳細については、the section called “ AWS アドオン” の互換性の表を参照してください。

アドオン用のカスタム名前空間

コミュニティアドオンと AWS アドオンの場合、アドオンの作成時に必要に応じてカスタム名前空間を指定できます。特定の名前空間にアドオンをインストールした後で、その名前空間を変更するには、いったんアドオンを削除してから再作成する必要があります。

名前空間を指定しないと、アドオンには事前定義された名前空間が使用されます。

カスタム名前空間を使用すると、EKS クラスター内でアドオンオブジェクトを効率よく分類整理できるようになります。この柔軟性により、アドオンを運用上のニーズや既存の名前空間戦略に合わせて整理できます。

アドオンを作成するときに、カスタム名前空間を設定できます。詳細については、the section called “アドオンを作成する” を参照してください。

アドオン向けに事前定義された名前空間を取得する

アドオン向けに事前定義された名前空間は、名前空間を指定しない場合にアドオンがインストールされる名前空間です。

アドオン向けに事前定義された名前空間を取得するには、次のコマンドを使用します。

```
aws eks describe-addon-versions --addon-name <addon-name> --query
 "addons[].defaultNamespace"
```

出力の例:

```
[
    "kube-system"
```

```
]
```

Amazon EKS 自動モードの考慮事項

Amazon EKS 自動モードには以下のような重要なクラスター機能を提供する機能が含まれています:

- ポッドネットワーキング
- サービスネットワーキング
- クラスター DNS
- 自動スケーリング
- ブロックストレージ
- ロードバランサーコントローラー
- Pod Identity エージェント
- ノードモニタリングエージェント

自動モードコンピューティングでは以下に示すような、一般的に使用される多くの EKS アドオンが冗長になります：

- Amazon VPC CNI
- kube-proxy
- CoreDNS
- Amazon EBS CSI ドライバー
- EKS Pod Identity エージェント

ただし、クラスターが自動モードをセルフマネージド EC2 インスタンス、マネージド型ノードグループ、AWS Fargate などの他のコンピューティングオプションと組み合わせる場合、これらのアドオンは引き続き必要です。AWS はアンチアフィニティルールを用いて EKS アドオンを強化し、アドオンポッドがサポートされているコンピュートタイプのみに自動的にスケジュールされるようにしています。さらに、ユーザーは EKS アドオン `DescribeAddonVersions` API を活用して、各アドオンとその特定のバージョンでサポートされている computeTypes を検証できるようになりました。加えて EKS 自動モードでは上記のコントローラーは AWS の所有するインフラストラクチャで実行されます。そのため、EKS 自動モードを他のタイプのコンピューティングと組み合わせて使用している場合をのぞいて、これらのコントローラーをアカウントで見ることすらないかもしれません。組み合わせて使用している場合はクラスターにインストールしたコントローラーが表示されます。

既存のクラスターで EKS 自動モードを有効にする予定がある場合は、特定のアドオンのバージョンアップグレードが必要になる場合があります。詳細については「EKS 自動モードの the section called “必要なアドオンバージョン”」を参照してください。

サポート

AWS はサポートレベルが異なる複数のタイプのアドオンを発行します。

- AWS アドオン： これらのアドオンは によって構築され、完全にサポートされていますAWS。
  - AWS アドオンを使用して、Amazon EFS などのAWS 他のサービスと連携します。
  - 詳細については、the section called “ AWS アドオン” を参照してください。
- AWS マーケットプレイス アドオン： これらのアドオンは独立したAWSパートナーによってスキャンAWSされ、サポートされます。
  - マーケットプレイス アドオンを使用して、Splunk によるモニタリングなど、クラスターに価値のある高度な機能を追加します。
  - 詳細については、the section called “マーケットプレイス アドオン” を参照してください。
- コミュニティアドオン: これらのアドオンは によってスキャンされますAWSが、オープンソースコミュニティによってサポートされます。
  - コミュニティアドオンを使用すると、Kubernetes メトリクスサーバーなどの一般的なオープンソースソフトウェアのインストールの複雑さを軽減できます。
  - コミュニティアドオンは AWS によってソースからパッケージ化され、AWS はバージョンの互換性についてのみコミュニティアドオンを検証します。
  - 詳細については、the section called “コミュニティアドオン” を参照してください。

次の表は各アドオンタイプのサポート範囲の詳細を示しています：

Category	機能	AWSアドオン	AWS マーケットプレイス アドオン	コミュニティアドオン
開発	AWS によって構築されました	はい	いいえ	はい
開発	AWS によって検証済み	はい	いいえ	はい*

Category	機能	AWSアドオン	AWS マーケットプレイス アドオン	コミュニティアドオン
開発	AWS パートナーによって検証済み	いいえ	はい	いいえ
メンテナンス	AWS によってスキャンされました	はい	はい	はい
メンテナンス	AWS によって修正されました	はい	いいえ	はい
メンテナンス	AWS パートナーによってパッチが適用されました	いいえ	はい	いいえ
ディストリビューション	AWS によって公開	はい	いいえ	はい
ディストリビューション	AWS パートナーによって公開	いいえ	はい	いいえ
サポート	AWSによる基本インストールサポート	はい	はい	はい
サポート	フルAWSサポート	はい	いいえ	いいえ
サポート	完全なAWSパートナーサポート	いいえ	はい	いいえ

*: コミュニティアドオンの検証には、Kubernetes バージョンの互換性のみが含まれます。例えば、クラスターにコミュニティアドオンをインストールすると、AWS はクラスターの Kubernetes バージョンと互換性があるかどうかを確認します。

AWS マーケットプレイス アドオンは AWS の外部ソースから追加のソフトウェア依存関係をダウンロードできます。これらの外部依存関係は によってスキャンまたは検証されませんAWS。外部の依存関係を取得する AWS マーケットプレイス アドオンをデプロイするときはセキュリティ要件を考慮してください。

AWS アドオン

以下の Amazon EKS アドオンをクラスターで作成できます。`eksctl`、AWS マネジメントコンソール、または AWS CLI を使用して、使用可能なアドオンの最新リストを表示できます。使用可能なすべてのアドオンを確認したり、アドオンをインストールしたりするにはthe section called “アドオンを作成する” を参照してください。アドオンが IAM アクセス権限を要求する場合、クラスター用に IAM OpenID Connect (OIDC) プロバイダーが必要です。既に存在しているかどうかを確認する、または作成するには、the section called “IAM OIDC プロバイダー” を参照してください。アドオンはインストール後に、作成または削除できます。詳細については、the section called “アドオンを更新する” または the section called “アドオンを削除する” を参照してください。Amazon EKS Hybrid Nodes で EKS アドオンを実行する際の考慮事項の詳細については、the section called “CNI アドオンの設定” を参照してください。

次の Amazon EKS アドオンのいずれかを使用できます。

説明	詳細情報	互換性のあるコンピューティングタイプ
クラスターのためにネイティブ VPC ネットワークを提供する	the section called “Amazon VPC CNI Kubernetes用プラグイン”	EC2
Kubernetes クラスター DNS として機能する、フレキシブルで拡張可能な DNS サーバー	the section called “CoreDNS”	EC2、Fargate、EKS 自動モード、EKS Hybrid Nodes
各 Amazon EC2 ノードのネットワークルールを管理する	the section called “`Kube-proxy` ”	EC2、EKS Hybrid Nodes

説明	詳細情報	互換性のあるコンピューティングタイプ
クラスターのために Amazon EBS ストレージを提供する	the section called “Amazon EBS CSI ドライバー”	EC2
クラスターのために Amazon EC2 インスタンスストレージを提供する	the section called “Amazon EC2 Instance Store CSI ドライバー”	EC2
クラスターのために Amazon EFS ストレージを提供する	the section called “Amazon EFS CSI ドライバー”	EC2、EKS 自動モード
クラスターのために Amazon S3 Filesストレージを提供する	the section called “Amazon EFS CSI ドライバー”	EC2、EKS 自動モード
クラスターに Amazon FSx for Lustre ストレージを提供する	the section called “Amazon FSx CSI ドライバー”	EC2、EKS 自動モード
クラスターのために Amazon S3 ストレージを提供する	the section called “Mountpoint for Amazon S3 CSI ドライバー”	EC2、EKS 自動モード
追加ノードのヘルス問題を検出する	the section called “ノードモニタリングエージェント”	EC2、EKS Hybrid Nodes
Amazon EBS CSI ドライバーなどの互換性のある CSI ドライバーでスナップショット機能の使用を有効にする	the section called “CSI スナップショットコントローラー”	EC2、Fargate、EKS 自動モード、EKS Hybrid Nodes
SageMaker HyperPod タスクガバナンスは Amazon EKS クラスター内のチーム間のコンピューティングリソースの割り当てと使用状況を最適化し、タスクの優先順位付けとリソース共有の非効率性に対処します。	the section called “Amazon SageMaker HyperPod タスクガバナンス”	EC2、EKS 自動モード

説明	詳細情報	互換性のあるコンピューティングタイプ
Amazon SageMaker HyperPod オブザーバビリティアドオンは、HyperPod クラスターの包括的なモニタリングおよびオブザーバビリティ機能を実現します。	the section called “Amazon SageMaker HyperPod オブザーバビリティアドオン”	EC2、EKS 自動モード
Amazon SageMaker HyperPod トレーニングオペレーターは、高度なスケジューリング機能とリソース管理機能を備えており、Amazon EKS クラスターで効率よく分散トレーニングを実施できます。	the section called “Amazon SageMaker HyperPod トレーニングオペレーター”	EC2、EKS 自動モード
Amazon SageMaker HyperPod 推論演算子は、リソース使用率とコスト効率を最適化して、高性能 AI 推論ワークロードのデプロイと管理を可能にします。	the section called “Amazon SageMaker HyperPod 推論演算子”	EC2、EKS 自動モード
ネットワークフローデータを収集して Amazon CloudWatch に報告する Kubernetes エージェント。クラスターノード間の TCP 接続を包括的にモニタリングできます。	the section called “AWSネットワーク・フロー・モニター の”	EC2、EKS 自動モード

説明	詳細情報	互換性のあるコンピューティングタイプ
オープンテレメトリー プロジェクトの、安全かつ本番に対応した、AWS によってサポートされているディストリビューション	the section called “AWS オープンテレメトリー用ディストロ”	EC2、Fargate、EKS 自動モード、EKS Hybrid Nodes
AWS CloudTrail 管理イベントや Amazon VPC フローログなどの基本的なデータソースを分析および処理するセキュリティモニタリングサービス。Amazon GuardDuty は、Kubernetes 監査ログやランタイムモニタリングなどの機能も処理します	the section called “Amazon GuardDuty エージェント”	EC2、EKS 自動モード
AWS によって提供されるモニタリングおよびオブザーバビリティサービス。このアドオンは CloudWatch エージェントをインストールし、Amazon EKS のオブザーバビリティが強化された CloudWatch アプリケーションシグナルと CloudWatch コンテナインサイトの両方を有効にします	the section called “Amazon CloudWatch 観測可能エージェント”	EC2、EKS 自動モード、EKS Hybrid Nodes
EC2 インスタンスプロファイルから EC2 インスタンスに認証情報を提供する場合と同じような方法で、アプリケーションの認証情報を管理する機能。	the section called “EKS Pod Identity エージェント”	EC2、EKS Hybrid Nodes

説明	詳細情報	互換性のあるコンピューティングタイプ
cert-manager を有効にして、AWS プライベート CA から X.509 証明書を発行します。cert-manager が必要です。	the section called “AWS プライベート CA の Connector for Kubernetes”	EC2、Fargate、EKS 自動モード、EKS Hybrid Nodes
SR-IOV ネットワークデバイスのパフォーマンスに関する Prometheus メトリクスを生成する	the section called “SR-IOV ネットワークメトリクスエクスポーター”	EC2
AWS Secrets Manager からシークレットを取得し、AWS Systems Manager Parameter Store からパラメータを取得して、Kubernetes ポッドにファイルとしてマウントします。	the section called “AWS Secrets Store CSI Driver プロバイダー”	EC2、EKS 自動モード、EKS Hybrid Nodes
Spaces を使用すると、JupyterLab および Code Editor アプリケーションを作成および管理して、インタラクティブな ML ワークロードを実行できます。	the section called “Amazon SageMaker Spaces”	HyperPod

Amazon VPC CNI Kubernetes用プラグイン

Amazon VPC CNI Plugin for Kubernetes の Amazon EKS アドオンは、クラスターにネイティブ VPC ネットワークを提供する Kubernetes コンテナネットワークインターフェイス (CNI) プラグインです。このアドオンのセルフマネージドタイプまたはマネージドタイプが、デフォルトで各 Amazon EC2 ノードにインストールされます。詳細については「Kubernetesコンテナネットワークインターフェース (CNI) プラグイン」を参照してください。

> **Note**
>
> このアドオンを Amazon EKS 自動モード クラスターにインストールする必要はありません。詳細については、the section called “Amazon EKS 自動モードの考慮事項” を参照してください。

Amazon EKS アドオン名は `vpc-cni` です。

必要な IAM 許可

このアドオンはAmazon EKS のサービスアカウント用の IAM ロール機能を使用します。詳細については、the section called “IRSA による認証情報” を参照してください。

クラスターが IPv4 ファミリーを使用する場合はAmazonEKS_CNI_Policy のアクセス権限が必要です。クラスターが IPv6 ファミリーを使用する場合は IPv6 モード のアクセス許可を持つ IAM ポリシー を作成する必要があります。次のコマンドで、IAM ロールを作成し、そのロールにポリシーの 1 つをアタッチして、アドオンが使用する Kubernetes サービスアカウントに注釈を付けることができます。

`#######` をクラスターの名前に置き換え、*`AmazonEKSVPCCNIRole`* を役割の名前に置き換えます。クラスターが IPv6 ファミリを使用する場合は*`AmazonEKS_CNI_Policy`* を作成したポリシーの名前に置き換えてください。このコマンドを使用するにはデバイスに eksctl がインストールされている必要があります。別のツールを使用してロールを作成し、ポリシーをアタッチして、Kubernetes サービスアカウントに注釈を付ける必要がある場合は、「the section called “IAM ロールを割り当てる”」を参照してください。

```
eksctl create iamserviceaccount --name aws-node --namespace kube-system --cluster my-
cluster --role-name AmazonEKSVPCCNIRole \
    --role-only --attach-policy-arn arn:aws:iam::aws:policy/AmazonEKS_CNI_Policy --
approve
```

情報を更新する

一度に更新できるマイナーバージョンは 1 つのみです。例えば、現在のバージョンが `1.28.`*`x`*`-eksbuild.`*`y`* で、これを `1.30.`*`x`*`-eksbuild.`*`y`* に更新する場合は現在のバージョンを `1.29.`*`x`*`-eksbuild.`*`y`* に更新してから、再度 `1.30.`*`x`*`-eksbuild.`*`y`* に更新する必要があります。アドオンの更新の詳細については「the section called “更新 (EKS アドオン)”」を参照してください。

CoreDNS

CoreDNS の Amazon EKS アドオンは、Kubernetes クラスター DNS として機能できる柔軟で拡張可能な DNS サーバーです。このアドオンのセルフマネージドタイプまたはマネージドタイプが、クラスターの作成時にデフォルトでインストールされました。少なくとも 1 つのノードで Amazon EKS クラスターを起動すると、クラスターにデプロイされたノード数に関係なく、デフォルトで CoreDNS イメージの 2 つのレプリカがデプロイされます。これらの CoreDNS ポッドは、クラスター内のすべてのポッドの名前解決を行います。クラスターに Fargate プロファイルが含まれ、名前空間が CoreDNS デプロイの名前空間と一致している場合、CoreDNS Pod を Fargate ノードにデプロイできます。詳細については、「the section called “プロファイルを定義する”」を参照してください。

> **Note**
>
> このアドオンを Amazon EKS 自動モード クラスターにインストールする必要はありません。詳細については、the section called “Amazon EKS 自動モードの考慮事項” を参照してください。

Amazon EKS アドオン名は `coredns` です。

必要な IAM 許可

このアドオンには許可は必要ありません。

追加情報

CoreDNS の詳細については、Kubernetes ドキュメントの「サービスディスカバリーに CoreDNS を使用する」および「Customizing DNS Service」を参照してください。

Kube-proxy

Kube-proxy Amazon EKS アドオンは各 Amazon EC2 ノードのネットワークルールを維持します。これにより、Pod とのネットワーク通信が可能になります。このアドオンのセルフマネージドタイプまたはマネージドタイプが、デフォルトでクラスター内の各 Amazon EC2 ノードにインストールされます。

> **Note**
>
> このアドオンを Amazon EKS 自動モード クラスターにインストールする必要はありません。詳細については、the section called “Amazon EKS 自動モードの考慮事項” を参照してください。

Amazon EKS アドオン名は `kube-proxy` です。

必要な IAM 許可

このアドオンには許可は必要ありません。

情報を更新する

現在のバージョンを更新する前に、次の要件を考慮してください:

- Amazon EKS クラスターの `Kube-proxy` はKubernetes と同じ互換性およびスキューポリシーが適用されます。

追加情報

`kube-proxy` の詳細については、Kubernetes ドキュメントの「kube-proxy」を参照してください。

Amazon EBS CSI ドライバー

Amazon EBS CSI ドライバーの Amazon EKS アドオンは、クラスター用に Amazon EBS ストレージを提供する Kubernetes Container Storage Interface (CSI) プラグインです。

> **Note**
>
> このアドオンを Amazon EKS 自動モード クラスターにインストールする必要はありません。自動モード にはブロックストレージ機能があります。詳細については、the section called “ステートフルワークロードをデプロイする” を参照してください。

Amazon EKS アドオン名は `aws-ebs-csi-driver` です。

必要な IAM 許可

このアドオンはAmazon EKS のサービスアカウント用の IAM ロール機能を使用します。詳細については、the section called “IRSA による認証情報” を参照してください。このアドオンには、タグベースのスコープ用の AmazonEBSCSIDriverPolicyV2、クラススコープ分離用の AmazonEBSCSIDriverEKSClusterScopedPolicy、またはタグベースの制限が必要ない場合は AmazonEBSCSIDriverPolicy のいずれかの AWS マネージドポリシーが必要です。次のコマンドで IAM ロールを作成して、それにマネージドポリシーをアタッチできます。*#######* をクラスターの名前に置き換え、*AmazonEKS_EBS_CSI_DriverRole* を役割の名前に置き換えます。このコマンドを使用するにはデバイスに eksctl がインストールされている必要があります。別のツールを使用する必要がある場合や、暗号化にカスタム KMS キー を使用する必要がある場合はthe section called “ステップ 1: IAM ロールを作成する” を参照してください。

`AmazonEBSCSIDriverPolicy` から移行する場合は、EBS CSI ドライバーポリシーの移行 を参照してください。

```
eksctl create iamserviceaccount \
    --name ebs-csi-controller-sa \
    --namespace kube-system \
    --cluster my-cluster \
    --role-name AmazonEKS_EBS_CSI_DriverRole \
    --role-only \
    --attach-policy-arn arn:aws:iam::aws:policy/service-role/AmazonEBSCSIDriverPolicyV2
 \
    --approve
```

追加情報

アドオンの詳細については、the section called “Amazon EBS” を参照してください。

Amazon EC2 Instance Store CSI ドライバー

Amazon EC2 Instance Store CSI ドライバー Amazon EKS アドオンは、クラスター用に Amazon EC2 インスタンスストア NVMe ストレージを提供する Kubernetes Container Storage Interface (CSI) プラグインです。

Amazon EKS アドオン名は `aws-ec2-local-instance-store-csi-driver` です。

必要な IAM 許可

このアドオンには許可は必要ありません。ドライバーはローカルの Linux オペレーションを通じてバックエンドの NVMe デバイスとやり取りし、AWS API コールは行いません。

追加情報

アドオンの詳細については、the section called “Amazon EC2 インスタンスストア” を参照してください。

Amazon EFS CSI ドライバー

Amazon EFS CSI ドライバー Amazon EKS アドオンは、クラスター用に Amazon EFS と Amazon S3 Files ストレージを提供する Kubernetes Container Storage Interface (CSI) プラグインです。

Amazon EKS アドオン名は `aws-efs-csi-driver` です。

必要な IAM 許可

このアドオンはAmazon EKS のサービスアカウント用の IAM ロール機能を使用します。詳細については、the section called “IRSA による認証情報” を参照してください。

必要な特定の AWS マネージドポリシーは、使用するファイルシステムタイプによって異なります。

- Amazon EFS ファイルシステムのみ: AmazonEFSCSIDriverPolicy マネージドポリシーをアタッチします。
- Amazon S3 Filesシステムのみ: `AmazonS3FilesCSIDriverPolicy` マネージドポリシーをアタッチします。
- Amazon EFS と Amazon S3 Filesシステムの両方の場合: `AmazonEFSCSIDriverPolicy` と `AmazonS3FilesCSIDriverPolicy` の両方のマネージドポリシーをアタッチします。

次のコマンドで IAM ロールを作成して、その役割にマネージドポリシーをアタッチできます。*`######`* をクラスターの名前に置き換え、*`AmazonEKS_EFS_CSI_DriverRole`* を役割の名前に置き換えます。次の例では、Amazon EFS ファイルシステムの `AmazonEFSCSIDriverPolicy` をアタッチします。Amazon S3 Filesシステムを使用している場合は、ポリシー ARN を `arn:aws:iam::aws:policy/service-role/AmazonS3FilesCSIDriverPolicy` に置き換えます。両方のファイルシステムタイプを使用している場合は、2 番目のポリシー ARN で `--attach-policy-arn` フラグを追加します。これらのコマンドを使用するにはデバイスに eksctl がインストールされている必要があります。別のツールを使用する必要がある場合は、Amazon EFS の場合は the section called “ステップ 1: IAM ロールを作成する”、Amazon S3 Filesの場合は the section called “ステップ 1: IAM ロールを作成する” を参照してください。

```
export cluster_name=my-cluster
export role_name=AmazonEKS_EFS_CSI_DriverRole
```

```
eksctl create iamserviceaccount \
    --name efs-csi-controller-sa \
    --namespace kube-system \
    --cluster $cluster_name \
    --role-name $role_name \
    --role-only \
    --attach-policy-arn arn:aws:iam::aws:policy/service-role/AmazonEFSCSIDriverPolicy \
    --approve
TRUST_POLICY=$(aws iam get-role --output json --role-name $role_name --query
 'Role.AssumeRolePolicyDocument' | \
    sed -e 's/efs-csi-controller-sa/efs-csi-*/' -e 's/StringEquals/StringLike/')
aws iam update-assume-role-policy --role-name $role_name --policy-document
 "$TRUST_POLICY"
```

> **Note**
>
> 上記の例では、`efs-csi-controller-sa` のみを設定します。Amazon S3 Filesシステムを使用している場合は、`efs-csi-node-sa` も設定する必要があります。S3 ファイル IAM のセットアップの詳細については、the section called “ステップ 1: IAM ロールを作成する” を参照してください。

追加情報

アドオンの詳細については、the section called “Amazon EFS” を参照してください。

Amazon FSx CSI ドライバー

Amazon FSx CSI ドライバー Amazon EKS アドオンは、クラスター用に Amazon FSx for Lustre ストレージを提供する Kubernetes Container Storage Interface (CSI) プラグインです。

Amazon EKS アドオン名は `aws-fsx-csi-driver` です。

> **Note**
>
> - クラスターに既存の Amazon FSx CSI ドライバーをインストールすると、アドオンのインストールに失敗する可能性があります。EKS 以外の FSx CSI ドライバーが存在するところに Amazon EKS のアドオンバージョンをインストールしようとすると、リソースの競合によりインストールが失敗します。この問題を解決するには、インストール時に `OVERWRITE` フラグを使用します。

```
aws eks create-addon --addon-name aws-fsx-csi-driver --cluster-name my-cluster
 --resolve-conflicts OVERWRITE
```

- Amazon FSx CSI Driver EKS アドオンは、EKS Pod Identity またはサービスアカウントの IAM ロール (IRSA) による認証をサポートしています。EKS Pod Identity を使用するには、FSx CSI Driver アドオンをデプロイする前後に Pod Identity エージェントをインストールします。詳細については、the section called “エージェントのセットアップ” を参照してください。代わりに IRSA を使用するには、the section called “IAM OIDC プロバイダー” を参照してください。

必要な IAM 許可

このアドオンはAmazon EKS のサービスアカウント用の IAM ロール機能を使用します。詳細については、the section called “IRSA による認証情報” を参照してください。AmazonFSxFullAccess AWS マネージドポリシーの許可が必要です。次のコマンドで IAM ロールを作成して、それにマネージドポリシーをアタッチできます。*#######* をクラスターの名前に置き換え、*AmazonEKS_FSx_CSI_* をロールの名前に置き換えます。このコマンドを使用するにはデバイスに eksctl がインストールされている必要があります。

```
eksctl create iamserviceaccount \
    --name fsx-csi-controller-sa \
    --namespace kube-system \
    --cluster my-cluster \
    --role-name AmazonEKS_FSx_CSI_DriverRole \
    --role-only \
    --attach-policy-arn arn:aws:iam::aws:policy/AmazonFSxFullAccess \
    --approve
```

追加情報

アドオンの詳細については、the section called “Amazon FSx for Lustre” を参照してください。

Mountpoint for Amazon S3 CSI ドライバー

Mountpoint for Amazon S3 CSI ドライバーの Amazon EKS アドオンは、クラスター用に Amazon S3 ストレージを提供する Kubernetes Container Storage Interface (CSI) プラグインです。

Amazon EKS アドオン名は `aws-mountpoint-s3-csi-driver` です。

必要な IAM 許可

このアドオンはAmazon EKS のサービスアカウント用の IAM ロール機能を使用します。詳細については、the section called “IRSA による認証情報” を参照してください。

作成される IAM ロールには S3 へのアクセスを許可するポリシーが必要です。ポリシーを作成するときは Mountpoint IAM アクセス許可の推奨事項 に従ってください。別の方法として、AWS マネージドポリシー AmazonS3FullAccess を使用することもできますが、このマネージドポリシーでは Mountpoint に必要以上のアクセス権限が付与されます。

次のコマンドで IAM ロールを作成して、その役割にマネージドポリシーをアタッチできます。*my-cluster* をクラスターの名前に、*region-code* を正しい AWS リージョンコードに、*AmazonEKS_S3_CSI_DriverRole* を役割の名前に、*AmazonEKS_S3_CSI_DriverRole_ARN* を役割 ARN に置き換えます。これらのコマンドを使用するにはデバイスに eksctl がインストールされている必要があります。IAM コンソールまたは AWS CLI を使用する手順については「the section called “ステップ 2: IAM ロールを作成する”」を参照してください。

```
CLUSTER_NAME=my-cluster
REGION=region-code
ROLE_NAME=AmazonEKS_S3_CSI_DriverRole
POLICY_ARN=AmazonEKS_S3_CSI_DriverRole_ARN
eksctl create iamserviceaccount \
    --name s3-csi-driver-sa \
    --namespace kube-system \
    --cluster $CLUSTER_NAME \
    --attach-policy-arn $POLICY_ARN \
    --approve \
    --role-name $ROLE_NAME \
    --region $REGION \
    --role-only
```

追加情報

アドオンの詳細については、the section called “Mountpoint for Amazon S3” を参照してください。

CSI スナップショットコントローラー

コンテナ・ストレージ・インターフェース (CSI) スナップショットコントローラーを使用すると、Amazon EBS CSI ドライバーなどの互換性のある CSI ドライバーのスナップショット機能を使用できます。

Amazon EKS アドオン名は `snapshot-controller` です。

必要な IAM 許可

このアドオンには許可は必要ありません。

追加情報

アドオンの詳細については、the section called “CSI スナップショットコントローラー” を参照してください。

Amazon SageMaker HyperPod タスクガバナンス

SageMaker HyperPod タスクガバナンスはリソースの割り当てを合理化し、Amazon EKS クラスターのチームやプロジェクト全体でコンピューティングリソースを効率的に活用できるように設計されています。これにより、管理者は次の設定を行うことができます。

- さまざまなタスクの優先度
- 各チームのコンピューティング割り当て
- 各チームがアイドル状態のコンピューティングリソースを貸し借りする方法
- チームが自らのタスクを先取りした場合

HyperPod タスクガバナンスは Amazon EKS クラスターオブザーバビリティも提供し、クラスター容量をリアルタイムで可視化します。これにはコンピューティングの可用性と使用状況、チームの割り当てと使用状況、タスクの実行と待機時間に関する情報が含まれ、情報に基づいた意思決定とプロアクティブなリソース管理のための設定を行います。

Amazon EKS アドオン名は `amazon-sagemaker-hyperpod-taskgovernance` です。

必要な IAM 許可

このアドオンには許可は必要ありません。

追加情報

アドオンの詳細については「SageMaker HyperPod タスクガバナンス」を参照してください。

Amazon SageMaker HyperPod オブザーバビリティアドオン

Amazon SageMaker HyperPod Observability アドオンは、HyperPod クラスターの包括的なモニタリングおよびオブザーバビリティ機能を実現します。このアドオンは、ノードエクスポー

ター、DCGM エクスポーター、kube-state-metrics、EFA エクスポーターなどの重要なモニタリングコンポーネントを自動的にデプロイおよび管理します。メトリクスを収集して顧客指定の Amazon Managed Prometheus (AMP) インスタンスに転送し、顧客のトレーニングジョブからカスタムメトリクスおよびイベント取り込み用の OTLP エンドポイントを公開します。

アドオンは、HyperPod Task Governance アドオン、HyperPod Training Operator、Kubeflow、KEDA などのさまざまなコンポーネントからメトリクスをスクレイピングすることで、より広範な HyperPod エコシステムに統合されます。収集されたすべてのメトリクスは Amazon Managed Prometheus で一元化され、顧客は Amazon Managed Grafana ダッシュボードを通じて統一されたオブザーバビリティビューを実現できます。HyperPod 環境全体でクラスターの状態、リソース使用率、トレーニングジョブのパフォーマンスをエンドツーエンドで可視化できるようになります。

Amazon EKS アドオン名は `amazon-sagemaker-hyperpod-observability` です。

必要な IAM 許可

このアドオンはAmazon EKS のサービスアカウント用の IAM ロール機能を使用します。詳細については、the section called “IRSA による認証情報” を参照してください。次のマネージドポリシーが必要です。

- `AmazonPrometheusRemoteWriteAccess` – クラスターから AMP へのリモート書き込みメトリクスの場合
- `CloudWatchAgentServerPolicy` – クラスターから CloudWatch へのログのリモート書き込みの場合

追加情報

アドオンおよびその機能の詳細については、「SageMaker HyperPod のオブザーバビリティ」を参照してください。

Amazon SageMaker HyperPod トレーニングオペレーター

Amazon SageMaker HyperPod トレーニングオペレーターでは、大規模な GPU クラスター全体で分散トレーニングを効率的に管理することで、生成 AI モデルを迅速に開発できます。インテリジェントな障害復旧、ハングしたジョブの検出、プロセスレベルの管理機能が導入されており、トレーニングの中断を最小限に抑え、コストを削減できます。障害発生時にジョブを完全に再起動しなければならない従来のトレーニングインフラストラクチャとは異なり、このオペレーターは外科的なプロセス復旧を実装してトレーニングジョブを円滑に実行し続けます。

また、HyperPod のヘルスモニタリング機能およびオブザーバビリティ機能と連携して、トレーニングの実行と重要なメトリクス (損失の急増やスループットの低下など) の自動モニタリングをリアルタイムに可視化します。コード変更が不要なシンプルな YAML 設定を利用して復旧ポリシーを定義できるため、回復不能なトレーニング状態にすばやく対応して復旧させることができます。こうしたモニタリング機能と復旧機能が連携して、運用上のオーバーヘッドを最小限に抑えながら、最適なトレーニングパフォーマンスを維持します。

Amazon EKS アドオン名は `amazon-sagemaker-hyperpod-training-operator` です。

詳細については、「Amazon SageMaker デベロッパーガイド」の「HyperPod トレーニングオペレーターの使用」を参照してください。

必要な IAM 許可

このアドオンでは、IAM アクセス許可が必須であり、EKS Pod Identity を使用します。

AWS では、`AmazonSageMakerHyperPodTrainingOperatorAccess`マネージドポリシー を推奨しています。

詳細については、「Amazon SageMaker 開発者ガイド」の「Installing the training operator」を参照してください。

追加情報

アドオンの詳細については、「SageMaker HyperPod training operator」を参照してください。

Amazon SageMaker HyperPod 推論演算子

Amazon SageMaker HyperPod は、インタラクティブな実験やトレーニングから推論やトレーニング後のワークフローまで、AI 開発のライフサイクル全体をサポートするエンドツーエンドのエクスペリエンスを実現します。Kubernetes の柔軟性とマネージドエクスペリエンスの運用上の優位性を組み合わせた包括的な推論プラットフォームを提供します。モデルライフサイクル全体で同じ HyperPod コンピューティングを使用して、エンタープライズグレードの信頼性で生成 AI モデルをデプロイ、スケーリング、最適化できます。

Amazon SageMaker HyperPod は、kubectl、Python SDK、Amazon SageMaker Studio UI、HyperPod CLI など、複数の方法でモデルをデプロイできる柔軟なデプロイインターフェイスを提供します。この機能は、需要に基づいて自動的に調整される動的リソース割り当てを備えた高度なオートスケーリング機能を提供します。さらに、パフォーマンスを最適化するために time-to-first-

token、GPU 使用率などの重要なメトリクスを追跡する包括的なオブザーバビリティとモニタリング機能を提供しています。

Amazon EKS アドオン名は `amazon-sagemaker-hyperpod-inference` です。

インストール方法

このアドオンをインストールするには、次のいずれかの方法を使用します。

- SageMaker コンソール (推奨): ガイド付き設定で効率的なインストールエクスペリエンスを提供します。
- EKS アドオンコンソールまたは CLI: 推論演算子をインストールする前に、依存関係アドオンを手動でインストールする必要があります。必要な依存関係については、以下の前提条件セクションを参照してください。

前提条件

EKS アドオンコンソールまたは CLI を使用して推論演算子アドオンをインストールする前に、次の依存関係がインストールされていることを確認してください。

必要な EKS アドオン:

- Amazon S3 Mountpoint CSI ドライバー (最小バージョン: v1.14.1-eksbuild.1)
- Metrics Server (最小バージョン: v0.7.2-eksbuild.4)
- Amazon FSx CSI ドライバー (最小バージョン: v1.6.0-eksbuild.1)
- Cert Manager (最小バージョン: v1.18.2-eksbuild.2)

各依存関係のインストール手順の詳細については、「Installing the inference operator」を参照してください。

必要な IAM 許可

このアドオンは IAM アクセス許可を必要とし、OIDC/IRSA を使用します。

以下のマネージドポリシーは、最小範囲のアクセス許可を提供するため、使用が推奨されます。

- `AmazonSageMakerHyperPodInferenceAccess` – 推論演算子の設定に必要な管理者権限を提供します
- `AmazonSageMakerHyperPodGatedModelAccess` – SageMaker Jumpstart (Meta Llama、GPT-Neo など) でゲート付きモデルへのアクセスを SageMaker HyperPod に提供します

詳細については、「Installing the inference operator」を参照してください。

追加情報

Amazon SageMaker HyperPod 推論演算子の詳細については、「SageMaker HyperPod inference operator」を参照してください。

トラブルシューティングの詳細については、「Troubleshooting SageMaker HyperPod model deployment」を参照してください。

AWSネットワーク・フロー・モニター の

Amazon CloudWatch ネットワーク・フロー・モニター エージェントはクラスター内のすべてのノードから TCP 接続統計を収集し、ネットワークフローレポートを Amazon CloudWatch ネットワーク・フロー・モニター 摂取 APIs。

Amazon EKS アドオン名は `aws-network-flow-monitoring-agent` です。

必要な IAM 許可

このアドオンには IAM 権限が必要です。

アドオンに `CloudWatchNetworkFlowMonitorAgentPublishPolicy` マネージドポリシーを添付する必要があります。

必要な IAM セットアップの詳細については Amazon CloudWatch ネットワーク・フロー・モニター エージェント GitHub リポジトリの「IAM ポリシー」を参照してください。

マネージドポリシーの詳細については、Amazon CloudWatch ユーザーガイドの CloudWatchNetworkFlowMonitorAgentPublishPolicy を参照してください。

追加情報

アドオンの詳細については Amazon CloudWatch ネットワーク・フロー・モニター エージェント GitHub リポジトリを参照してください。

ノードモニタリングエージェント

ノードモニタリングエージェントの Amazon EKS アドオンは追加ノードヘルスの問題を検出できます。これらの追加のヘルスシグナルはオプションのノード自動修復機能でも活用でき、必要に応じてノードを自動的に置き換えることができます。

> **Note**
>
> このアドオンを Amazon EKS 自動モード クラスターにインストールする必要はありません。詳細については、the section called “Amazon EKS 自動モードの考慮事項” を参照してください。

Amazon EKS アドオン名は `eks-node-monitoring-agent` です。

必要な IAM 許可

このアドオンに追加の許可は必要ありません。

追加情報

詳細については、the section called “ノードのヘルスと修復” を参照してください。

AWS オープンテレメトリー用ディストロ

AWS オープンテレメトリー用ディストロ Amazon EKS アドオンは安全な本番環境対応の、AWS によってサポートされている オープンテレメトリー プロジェクトのディストリビューションです。詳細についてはGitHub で「AWS オープンテレメトリー用ディストロ」を参照してください。

Amazon EKS アドオン名は `adot` です。

必要な IAM 許可

このアドオンには詳細設定でオプトインできる事前設定されたカスタムリソースの 1 つを使用している場合のみ IAM 許可が必要です。

追加情報

詳細については、AWS Distro for OpenTelemetry ドキュメントの「Getting Started with AWS Distro for OpenTelemetry using EKS Add-Ons」を参照してください。

ADOT では前提条件として `cert-manager` アドオンがクラスターにデプロイされている必要があります。そうでなければ、https://registry.terraform.io/modules/terraform-aws-modules/eks/aws/latest `cluster_addons` プロパティを使用して直接デプロイした場合、このアドオンは機能しません。その他の要件については、AWS Distro for OpenTelemetry ドキュメントの「Requirements for Getting Started with AWS Distro for OpenTelemetry using EKS Add-Ons」を参照してください。

Amazon GuardDuty エージェント

Amazon GuardDuty エージェント Amazon EKS アドオンでは、GuardDuty ランタイムモニタリングによる分析のために、EKS クラスターノードからランタイムイベント (ファイルアクセス、プロセス実行、ネットワーク接続) を収集します。GuardDuty 自体 (エージェントではない) は、AWS CloudTrail 管理イベントや Amazon VPC フローログなどの基本的なデータソースと、Kubernetes 監査ログやランタイムモニタリングなどの機能を分析して処理するセキュリティモニタリングサービスです。

Amazon EKS アドオン名は `aws-guardduty-agent` です。

必要な IAM 許可

このアドオンには許可は必要ありません。

追加情報

詳細については「Amazon GuardDuty における Amazon EKSクラスタのランタイムモニタリング」を参照してください。

- Amazon EKS クラスターで潜在的なセキュリティの脅威を検出するには Amazon GuardDuty ランタイムモニタリングを有効にし、ガードデューティ セキュリティエージェントを Amazon EKS クラスターにデプロイします。

Amazon CloudWatch 観測可能エージェント

Amazon CloudWatch Observability エージェント Amazon EKS アドオンは AWS が提供するモニタリングおよびオブザーバビリティサービスです。このアドオンは CloudWatch エージェントをインストールし、Amazon EKS のオブザーバビリティが強化された CloudWatch アプリケーションシグナルと CloudWatch コンテナインサイトの両方を有効にします。詳細については「Amazon CloudWatch エージェント」を参照してください。

Amazon EKS アドオン名は `amazon-cloudwatch-observability` です。

必要な IAM 許可

このアドオンはAmazon EKS のサービスアカウント用の IAM ロール機能を使用します。詳細については、the section called “IRSA による認証情報” を参照してください。AWSXrayWriteOnlyAccess と CloudWatchAgentServerPolicy AWS のマネージドポリシーのパーミッションが必要です。次のコマンドで IAM ロールを作成し、それにマネージドポリシーをアタッチして、アドオンが使用する Kubernetes サービスアカウントに注釈を付けることができます。`#######` をクラスターの名前に置

き換え、*AmazonEKS_Observability_role* を役割の名前に置き換えます。このコマンドを使用するにはデバイスに eksctl がインストールされている必要があります。別のツールを使用してロールを作成し、ポリシーをアタッチして、Kubernetes サービスアカウントに注釈を付ける必要がある場合は、「the section called “IAM ロールを割り当てる”」を参照してください。

```
eksctl create iamserviceaccount \
    --name cloudwatch-agent \
    --namespace amazon-cloudwatch \
    --cluster my-cluster \
    --role-name AmazonEKS_Observability_Role \
    --role-only \
    --attach-policy-arn arn:aws:iam::aws:policy/AWSXrayWriteOnlyAccess \
    --attach-policy-arn arn:aws:iam::aws:policy/CloudWatchAgentServerPolicy \
    --approve
```

追加情報

詳細については「CloudWatch エージェントのインストール」を参照してください。

AWS プライベート CA の Connector for Kubernetes

Kubernetes 用 AWS プライベート CA Connector は、ユーザーが AWS プライベート Certificate Authority (AWS プライベート CA) から証明書を取得できるようにする cert-manager のアドオンです。

- Amazon EKS アドオン名は `aws-privateca-connector-for-kubernetes` です。
- アドオン名前空間は `aws-privateca-issuer` です。

このアドオンには `cert-manager` が必要です。`cert-manager` はコミュニティアドオンとして Amazon EKS で使用できます。このアドオンの詳細については、「the section called “Cert Manager”」を参照してください。アドオンのインストールに関する詳細については、the section called “アドオンを作成する” を参照してください。

必要な IAM 許可

このアドオンには IAM 権限が必要です。

EKS Pod Identities を使用して `AWSPrivateCAConnectorForKubernetesPolicy` IAM ポリシーを `aws-privateca-issuer` Kubernetes サービスアカウントにアタッチします。詳細については、the section called “Pod Identity を使用する” を参照してください。

必要な権限の詳細については、「AWS マネージドポリシーのリファレンス」の「AWSPrivateCAConnectorForKubernetesPolicy」を参照してください。

追加情報

詳細については、AWS Private CA Issuer for Kubernetes GitHub リポジトリを参照してください。

アドオン設定の詳細については、aws-privateca-issuer GitHub リポジトリの「values.yaml」を参照してください。values.yaml のバージョンが、クラスターにインストールされているアドオンのバージョンと一致していることを確認します。

このアドオンは、EKS Auto Mode の system NodePool で使用される CriticalAddonsOnly テイントを許容します。詳細については、the section called “重要なアドオンを実行する” を参照してください。

EKS Pod Identity エージェント

Amazon EKS Pod Identity Agent Amazon EKS アドオンはEC2 インスタンスプロファイルから EC2 インスタンスに認証情報を提供する場合と同じような方法で、アプリケーションの認証情報を管理する機能を提供します。

> **Note**
>
> このアドオンを Amazon EKS 自動モード クラスターにインストールする必要はありません。Amazon EKS 自動モード は EKS Pod Identity と統合されています。詳細については、the section called “Amazon EKS 自動モードの考慮事項” を参照してください。

Amazon EKS アドオン名は eks-pod-identity-agent です。

必要な IAM 許可

Pod Identity Agent アドオン自体には IAM ロールは必要ありません。Amazon EKS ノードの IAM ロールからの許可を使用して機能しますが、アドオン専用の IAM ロールは必要ありません。

情報を更新する

一度に更新できるマイナーバージョンは 1 つのみです。例えば、現在のバージョンが 1.28.x-eksbuild.y で、これを 1.30.x-eksbuild.y に更新する場合は現在のバージョンを 1.29.x-eksbuild.y に更新してから、再度 1.30.x-eksbuild.y に更新する必要があります。アドオンの更新の詳細については「the section called “アドオンを更新する”」を参照してください。

SR-IOV ネットワークメトリクスエクスポーター

SR-IOV ネットワークメトリクスエクスポーターの Amazon EKS アドオンは、SR-IOV ネットワークデバイスに関するメトリクスを Prometheus 形式で収集して公開します。これにより、EKS ベアメタルノードの SR-IOV ネットワークパフォーマンスをモニタリングできます。エクスポーターは、SR-IOV 対応のネットワークインターフェイスを持つノードで DaemonSet として動作し、Prometheus でスクレイピングできるメトリクスをエクスポートします。

> **Note**
>
> このアドオンには、SR-IOV 対応のネットワークインターフェイスを持つノードが必要です。

プロパティ	値
アドオン名	`sriov-network-metrics-exporter`
名前空間	`monitoring`
ドキュメント	SR-IOV ネットワークメトリクスエクスポーター GitHub リポジトリ
サービスアカウント名	なし
マネージド IAM ポリシー	なし
カスタム IAM 許可	なし

AWS Secrets Store CSI Driver プロバイダー

Secrets Store CSI Driver 用の AWS プロバイダーは、AWS Secrets Manager からシークレットを、AWS Systems Manager Parameter Store からパラメータを取得し、Kubernetes ポッドにファイルとしてマウントできるようにするアドオンです。

必要な IAM 許可

このアドオンには IAM 許可は必要ありません。ただし、アプリケーションポッドには、AWS Secrets Manager からシークレットを、AWS Systems Manager Parameter Store からパラメータを

取得するための IAM 許可が必要です。アドオンをインストールしたら、サービスアカウントの IAM ロール (IRSA) または EKS Pod Identity を介してアクセスを設定する必要があります。IRSA を使用するには、Secrets Manager の IRSA セットアップドキュメントを参照してください。EKS Pod Identity を使用するには、Secrets Manager の Pod Identity セットアップドキュメントを参照してください。

AWS では、`AWSSecretsManagerClientReadOnlyAccess`マネージドポリシー を推奨しています。

必要な許可の詳細については、「AWS マネージドポリシーのリファレンス」の「`AWSSecretsManagerClientReadOnlyAccess`」を参照してください。

追加情報

詳細については、secrets-store-csi-driver-provider-aws GitHub リポジトリを参照してください。

アドオンの詳細については、アドオンに関する AWS Secrets Manager ドキュメントを参照してください。

Amazon SageMaker Spaces

Amazon SageMaker Spaces アドオンは、EKS クラスターまたは HyperPod-EKS クラスターで IDE とノートブックを実行する機能を提供します。管理者は EKS コンソールを使用してクラスターにアドオンをインストールし、イメージ、コンピューティングリソース、ノートブック設定用のローカルストレージ (スペースにアタッチされる追加ストレージ)、ファイルシステム、初期化スクリプトなどのデフォルトのスペース設定を定義できます。

AI 開発者は kubectl を使用してスペースを作成、更新、削除できます。管理者が提供するデフォルト設定を使用することも、設定をカスタマイズすることもできます。AI 開発者は、ローカルの VS Code IDE や、管理者が設定したカスタム DNS ドメインで JupyterLab または CodeEditor IDE をホストするウェブブラウザを使用して、EKS または HyperPod-EKS 上のスペースにアクセスできます。また、kubernetes のポート転送機能を使用して、ウェブブラウザからスペースにアクセスすることもできます。

Amazon EKS アドオン名は `amazon-sagemaker-spaces` です。

必要な IAM 許可

このアドオンには IAM 権限が必要です。必要な IAM セットアップの詳細については、「Amazon SageMaker デベロッパーガイド」の「IAM 許可のセットアップ」を参照してください。

追加情報

アドオンとその機能の詳細については、「Amazon SageMaker デベロッパーガイド」の「HyperPod 上の SageMaker AI ノートブック」を参照してください。

コミュニティアドオン

AWS API を使用して、Kubernetes Metrics Server などのコミュニティ アドオンをインストールできます。複数のクラスターでソフトウェアを保守する際の複雑さを軽減するために、コミュニティアドオンを Amazon EKS アドオンとしてインストールすることもできます。

例えば、AWS API、CLI、または管理コンソールを使用してコミュニティ アドオンをインストールできます。クラスターの作成時にコミュニティアドオンをインストールできます。

コミュニティアドオンは既存の Amazon EKS アドオンと同様に管理します。コミュニティアドオンは独自のサポート範囲があるという点で、既存のアドオンとは異なります。

> **Note**
>
> コミュニティアドオンの使用はユーザーが自由に決定できます。お客様と AWS 間の責任共有モデルの一環として、これらのサードパーティープラグインに関して、何をクラスターにインストールするかを理解しておく必要があります。また、お客様はクラスターのセキュリティニーズを満たすコミュニティアドオンについても責任を負います。詳細については、the section called “EKS にデプロイされたソフトウェアのサポート” を参照してください。

コミュニティアドオンは AWS によって構築されるものではありません。AWS は、バージョンの互換性に関するコミュニティアドオンの検証のみ行います。例えば、クラスターにコミュニティアドオンをインストールすると、AWS はクラスターの Kubernetes バージョンと互換性があるかどうかを確認します。

重要なのは AWS はコミュニティアドオンを完全にサポートしていないことです。AWS はアドオンのインストールやアドオンの削除など、AWS API を使用して行われたライフサイクルオペレーションのみをサポートします。

Sコミュニティアドオンのサポートが必要な場合は、既存のプロジェクトリソースを使用します。例えば、プロジェクトのリポジトリに GitHub の問題を作成できます。

アドオンタイプを決定する

AWS CLI を使用して、Amazon EKS アドオンのタイプを判断できます。

アドオンに関する情報を取得するには、次の CLI コマンドを使用します。metrics-server を任意のアドオンの名前に置き換えることができます。

```
aws eks describe-addon-versions --addon-name metrics-server
```

owner フィールドの CLI 出力を確認します。

```
{
    "addons": [
        {
            "addonName": "metrics-server",
            "type": "observability",
            "owner": "community",
            "addonVersions": [
```

owner の値が community の場合、アドオンはコミュニティ アドオンです。AWS は、アドオンのインストール、更新、および削除のサポートのみを提供します。アドオン自体の機能とオペレーションについて質問がある場合は、GitHub の問題などのコミュニティリソースを使用してください。

コミュニティアドオンをインストールまたは更新する

コミュニティアドオンは他の Amazon EKS アドオンと同じ方法でインストールまたは更新します。

- the section called “アドオンを作成する”
- the section called “アドオンを更新する”
- the section called “アドオンを削除する”

利用可能なコミュニティアドオン

Amazon EKS で利用できるコミュニティアドオンは次のとおりです。

- the section called “Kubernetes メトリクスサーバー”
- the section called “kube-state-metrics”
- the section called “Prometheus Node Exporter”
- the section called “Cert Manager”
- the section called “外部 DNS”
- the section called “Fluent Bit”

Kubernetes メトリクスサーバー

Kubernetes Metrics Server は、Kubernetes の組み込み自動スケーリングパイプラインのコンテナリソースメトリクスのスケーラブルで効率的なソースです。Kubelets からリソースメトリクスを収集し、Horizontal Pod Autoscaler と Vertical Pod Autoscaler で使用するメトリクス API を通じて Kubernetes apiserver で公開します。

プロパティ	値
アドオン名	`metrics-server`
名前空間	`kube-system`
ドキュメント	GitHub Readme
サービスアカウント名	なし
マネージド IAM ポリシー	なし
カスタム IAM 許可	なし

kube-state-metrics

クラスターレベルのメトリクスを生成して公開するアドオンエージェント。

Kubernetes API の Kubernetes オブジェクトの状態は、メトリクスとして公開できます。kube-state-metrics というアドオンエージェントは、Kubernetes API サーバーに接続し、クラスター内の個々のオブジェクトの状態から生成されたメトリクスを使用して HTTP エンドポイントを公開することができます。ラベルと注釈、起動時間と終了時間、オブジェクトの現在のステータスまたはフェーズなど、オブジェクトの状態に関するさまざまな情報が公開されます。

プロパティ	値
アドオン名	`kube-state-metrics`
名前空間	`kube-state-metrics`
ドキュメント	Kubernetes ドキュメントの「Metrics for Kubernetes Object States」

プロパティ	値
サービスアカウント名	なし
マネージド IAM ポリシー	なし
カスタム IAM 許可	なし

Prometheus Node Exporter

プラガブルメトリクスコレクターを使用して Go で記述された *NIX カーネルによって公開される、ハードウェアおよび OS メトリクス用の Prometheus エクスポーター。Prometheus Node Exporter は、さまざまなハードウェアおよびカーネル関連のメトリクスを公開します。

プロパティ	値
アドオン名	`prometheus-node-exporter`
名前空間	`prometheus-node-exporter`
ドキュメント	Prometheus ドキュメントの「Monitoring Linux host metrics with the Node Exporter」
サービスアカウント名	なし
マネージド IAM ポリシー	なし
カスタム IAM 許可	なし

Cert Manager

Cert Manager を使用すると、証明書の作成と更新を管理できます。

プロパティ	値
アドオン名	`cert-manager`
名前空間	`cert-manager`

プロパティ	値
ドキュメント	Cert Manager のドキュメント
サービスアカウント名	なし
マネージド IAM ポリシー	なし
カスタム IAM 許可	なし

外部 DNS

外部 DNS EKS アドオンを使用すると、Kubernetes リソースを介して Route53 DNS レコードを管理できます。

外部 DNS アクセス許可を、管理対象とするホストゾーンの `route53:ChangeResourceRecordSets`、`route53:ListHostedZones`、`route53:ListResourceRe` に縮小できます。

プロパティ	値
アドオン名	`external-dns`
名前空間	`external-dns`
ドキュメント	GitHub Readme
サービスアカウント名	`external-dns`
マネージド IAM ポリシー	`arn:aws:iam::aws:policy/AmazonRoute53FullAccess`
カスタム IAM 許可	なし

Fluent Bit

Fluent Bit は、軽量で高性能なログプロセッサおよびフォワーダーです。さまざまなソースからデータや口グを収集して統合し、Amazon CloudWatch Logs、Amazon S3、Amazon Data Firehose など

の複数の送信先に送信できるようにします。Fluent Bit はパフォーマンスおよびリソースの効率を配慮して設計され、Kubernetes 環境に最適です。

このアドオンには、デフォルト設定で IAM アクセス許可は必要ありません。ただし、AWS 出力口ケーションを設定する場合、このアドオンに IAM アクセス許可を付与する必要がある場合があります。詳細については、the section called “Pod Identity を使用する” を参照してください。

プロパティ	値
アドオン名	`fluent-bit`
名前空間	`fluent-bit`
ドキュメント	Fluent Bit のドキュメント
サービスアカウント名	`fluent-bit`
マネージド IAM ポリシー	なし
カスタム IAM 許可	なし

属性を表示する

コミュニティアドオンのオープンソース属性とライセンス情報をダウンロードできます。

1. 属性をダウンロードするアドオンの名前とバージョンを決定します。
2. その名前とバージョンで次のコマンドを更新します。

```
curl -O https://amazon-eks-docs.s3.amazonaws.com/attributions/<add-on-name>/<add-on-version>/attributions.zip
```

例えば、次のようになります。

```
curl -O https://amazon-eks-docs.s3.amazonaws.com/attributions/kube-state-metrics/v2.14.0-eksbuild.1/attributions.zip
```

3. コマンドを使用して、ファイルをダウンロードします。

この zip ファイルを使用して、ライセンス属性に関する情報を表示します。

AWS マーケットプレイス アドオン

前述の Amazon EKS アドオンのリストに加えて、独立系ソフトウェアベンダーのさまざまな運用ソフトウェア Amazon EKS アドオンを追加することもできます。アドオンを選択すると、そのアドオンとそのインストール要件について詳しく知ることができます。

Accuknox

アドオン名は `accuknox_kubearmor` で、名前空間は `kubearmor` です。アドオンの公開は、Accuknox で行います。

アドオンの詳細については、KubeArmor ドキュメントの「Getting Started with KubeArmor」を参照してください。

サービスアカウント名

サービスアカウントはこのアドオンでは使用されません。

AWS マネージド IAM ポリシー

このアドオンではマネージドポリシーは使用されません。

カスタム IAM 許可

このアドオンではカスタム許可は使用されません。

Akuity

アドオン名は `akuity_agent` で、名前空間は `akuity` です。アドオンの公開は、Akuity で行います。

アドオンの詳細については、Akuity Platform ドキュメントの「Installing the Akuity Agent on Amazon EKS with the Akuity EKS add-on」を参照してください。

サービスアカウント名

サービスアカウントはこのアドオンでは使用されません。

AWS マネージド IAM ポリシー

このアドオンではマネージドポリシーは使用されません。

カスタム IAM 許可

このアドオンではカスタム許可は使用されません。

Calyptia

アドオン名は `calyptia_fluent-bit` で、名前空間は `calyptia-fluentbit` です。アドオンの公開は、Calyptia で行います。

アドオンの詳細については、Calyptia ドキュメントウェブサイトの「Getting Started with Calyptia Core Agent」を参照してください。

サービスアカウント名

サービスアカウント名は `calyptia-fluentbit` です。

AWS マネージド IAM ポリシー

このアドオンは `AWSMarketplaceMeteringRegisterUsage` マネージドポリシーを使用します。詳細については、AWS マネージドポリシーのリファレンスガイドの「AWSMarketplaceMeteringRegisterUsage」を参照してください。

必要な IAM ロールを作成するコマンド

次のコマンドではクラスターの IAM OpenID Connect (OIDC) プロバイダーが必要です。既に存在しているかどうかを確認する、または作成するには、the section called “IAM OIDC プロバイダー” を参照してください。*my-cluster* をクラスターの名前に置き換え、*my-calyptia-role* をロールの名前に置き換えます。このコマンドを使用するにはデバイスに eksctl がインストールされている必要があります。別のツールを使用してロールを作成し、Kubernetes サービスアカウントに注釈を付ける必要がある場合は「the section called “IAM ロールを割り当てる”」を参照してください。

```
eksctl create iamserviceaccount --name service-account-name  --namespace calyptia-
fluentbit --cluster my-cluster --role-name my-calyptia-role \
    --role-only --attach-policy-arn arn:aws:iam::aws:policy/
AWSMarketplaceMeteringRegisterUsage --approve
```

Cisco Observability Collector

アドオン名は `cisco_cisco-cloud-observability-collectors` で、名前空間は `appdynamics` です。アドオンの公開は、Cisco で行います。

アドオンの詳細については、Cisco AppDynamics ドキュメントの「Use the Cisco Cloud Observability AWS Marketplace Add-Ons」を参照してください。

サービスアカウント名

サービスアカウントはこのアドオンでは使用されません。

AWS マネージド IAM ポリシー

このアドオンではマネージドポリシーは使用されません。

カスタム IAM 許可

このアドオンではカスタム許可は使用されません。

Cisco Observability Operator

アドオン名は `cisco_cisco-cloud-observability-operators` で、名前空間は `appdynamics` です。アドオンの公開は、Cisco で行います。

アドオンの詳細については、Cisco AppDynamics ドキュメントの「Use the Cisco Cloud Observability AWS Marketplace Add-Ons」を参照してください。

サービスアカウント名

サービスアカウントはこのアドオンでは使用されません。

AWS マネージド IAM ポリシー

このアドオンではマネージドポリシーは使用されません。

カスタム IAM 許可

このアドオンではカスタム許可は使用されません。

CLOUDSOFT

アドオン名は `cloudsoft_cloudsoft-amp` で、名前空間は `cloudsoft-amp` です。アドオンの公開は、CLOUDSOFT で行います。

アドオンの詳細については、CLOUDSOFT ドキュメントの「Amazon EKS ADDON」を参照してください。

サービスアカウント名

サービスアカウントはこのアドオンでは使用されません。

AWS マネージド IAM ポリシー

このアドオンではマネージドポリシーは使用されません。

カスタム IAM 許可

このアドオンではカスタム許可は使用されません。

Cribl

アドオン名は `cribl_cribledge` で、名前空間は `cribledge` です。アドオンの公開は、Cribl で行います。

アドオンの詳細については、Cribl ドキュメントの「Installing the Cribl Amazon EKS Add-on for Edge」を参照してください。

サービスアカウント名

サービスアカウントはこのアドオンでは使用されません。

AWS マネージド IAM ポリシー

このアドオンではマネージドポリシーは使用されません。

カスタム IAM 許可

このアドオンではカスタム許可は使用されません。

Dynatrace

アドオン名は dynatrace_dynatrace-operator で、名前空間は dynatrace です。アドオンの公開は、Dynatrace で行います。

アドオンの詳細については、Dynatrace ドキュメントの「Kubernetes monitoring」を参照してください。

サービスアカウント名

サービスアカウントはこのアドオンでは使用されません。

AWS マネージド IAM ポリシー

このアドオンではマネージドポリシーは使用されません。

カスタム IAM 許可

このアドオンではカスタム許可は使用されません。

Datree

アドオン名は datree_engine-pro で、名前空間は datree です。アドオンの公開は、Datree で行います。

アドオンの詳細については、Datree ドキュメントの「Amazon EKS-intergration」を参照してください。

サービスアカウント名

サービスアカウント名はdatree-webhook-server-awsmp です。

AWS マネージド IAM ポリシー

マネージドポリシーは AWSLicenseManagerConsumptionPolicy です。詳細については、AWS マネージドポリシーのリファレンスガイドの「AWSLicenseManagerConsumptionPolicy」を参照してください。

必要な IAM ロールを作成するコマンド

次のコマンドではクラスターの IAM OpenID Connect (OIDC) プロバイダーが必要です。既に存在しているかどうかを確認する、または作成するには、the section called “IAM OIDC プロバイダー” を参照してください。*my-cluster* をクラスターの名前に置き換え、*my-datree-role* をロールの名前に置き換えます。このコマンドを使用するにはデバイスに eksctl がインストールされている必要があります。別のツールを使用してロールを作成し、Kubernetes サービスアカウントに注釈を付ける必要がある場合は、the section called “IAM ロールを割り当てる” を参照してください。

```
eksctl create iamserviceaccount --name datree-webhook-server-awsmp --namespace datree
 --cluster my-cluster --role-name my-datree-role \
    --role-only --attach-policy-arn arn:aws:iam::aws:policy/service-role/
AWSLicenseManagerConsumptionPolicy --approve
```

カスタム許可

このアドオンではカスタム許可は使用されません。

Datadog

アドオン名は `datadog_operator` で、名前空間は `datadog-agent` です。アドオンの公開は、Datadog で行います。

アドオンの詳細については、Datadog ドキュメントの「Installing the Datadog Agent on Amazon EKS with the Datadog Operator Add-on」を参照してください。

サービスアカウント名

サービスアカウントはこのアドオンでは使用されません。

AWS マネージド IAM ポリシー

このアドオンではマネージドポリシーは使用されません。

カスタム IAM 許可

このアドオンではカスタム許可は使用されません。

Groundcover

アドオン名は `groundcover_agent` で、名前空間は `groundcover` です。アドオンの公開は、Groundcover で行います。

アドオンの詳細については、groundcover ドキュメントの「Installing the groundcover Amazon EKS Add-on」を参照してください。

サービスアカウント名

サービスアカウントはこのアドオンでは使用されません。

AWS マネージド IAM ポリシー

このアドオンではマネージドポリシーは使用されません。

カスタム IAM 許可

このアドオンではカスタム許可は使用されません。

IBM Instana

アドオン名は `instana-agent` で、名前空間は `instana-agent` です。IBM がアドオンを公開しています。

アドオンの詳細については、AWS ブログの「Implement observability for Amazon EKS workloads using the Instana Amazon EKS add-on」と「Monitor and optimize Amazon EKS costs with IBM Instana and Kubecost」を参照してください。

Instana Observability (Instana) は、Instana エージェントを Amazon EKS クラスターにデプロイする Amazon EKS アドオンを提供しています。このアドオンを使用すると、リアルタイムのパフォーマンスデータを収集および分析して、コンテナ化されたアプリケーションに関するインサイトを得ることができます。Instana Amazon EKS アドオンでは、Kubernetes 環境全体を可視化できます。デプロイすると、Instana エージェントがノード、名前空間、デプロイ、サービス、ポッドなど、Amazon EKS クラスター内のコンポーネントを自動的に検出します。

サービスアカウント名

サービスアカウントはこのアドオンでは使用されません。

AWS マネージド IAM ポリシー

このアドオンではマネージドポリシーは使用されません。

カスタム IAM 許可

このアドオンではカスタム許可は使用されません。

Grafana Labs

アドオン名は `grafana-labs_kubernetes-monitoring` で、名前空間は `monitoring` です。アドオンの公開は、Grafana Labs で行います。

アドオンの詳細については、Grafana Labs ドキュメントの「Configure Kubernetes Monitoring as an Add-on with Amazon EKS」を参照してください。

サービスアカウント名

サービスアカウントはこのアドオンでは使用されません。

AWS マネージド IAM ポリシー

このアドオンではマネージドポリシーは使用されません。

カスタム IAM 許可

このアドオンではカスタム許可は使用されません。

Guance

- パブリッシャー – Guance
- 名前 – `guance_datakit`
- 名前空間 – `datakit`
- サービスアカウント名 – サービスアカウントはこのアドオンでは使用されません。
- AWS マネージド IAM ポリシー – このアドオンではマネージドポリシーは使用されません。
- カスタム IAM アクセス許可 – このアドオンではカスタムアクセス許可は使用されません。
- セットアップと使用に関する指示 – Guance ドキュメントの「Using Amazon EKS add-on」を参照してください。

HA Proxy

名前は `haproxy-technologies_kubernetes-ingress-ee` で、名前空間は `haproxy-controller` です。アドオンの公開は、HA Proxy で行います。

アドオンの詳細については、HA Proxy のドキュメントの「Installing on AWS」を参照してください。

サービスアカウント名

サービスアカウント名は `customer defined` です。

AWS マネージド IAM ポリシー

マネージドポリシーは AWSLicenseManagerConsumptionPolicy です。詳細については、AWS マネージドポリシーのリファレンスガイドの「AWSLicenseManagerConsumptionPolicy」を参照してください。

必要な IAM ロールを作成するコマンド

次のコマンドではクラスターの IAM OpenID Connect (OIDC) プロバイダーが必要です。既に存在しているかどうかを確認する、または作成するには、the section called “IAM OIDC プロバイダー” を参照してください。*`my-cluster`* をクラスターの名前に置き換え、*`my-haproxy-role`* をロールの名前に置き換えます。このコマンドを使用するにはデバイスに eksctl がインストールされている必要があります。別のツールを使用してロールを作成し、Kubernetes サービスアカウントに注釈を付ける必要がある場合は、the section called “IAM ロールを割り当てる” を参照してください。

```
eksctl create iamserviceaccount --name service-account-name  --namespace haproxy-
controller --cluster my-cluster --role-name my-haproxy-role \
    --role-only --attach-policy-arn arn:aws:iam::aws:policy/service-role/
AWSLicenseManagerConsumptionPolicy --approve
```

カスタム権限

このアドオンではカスタム許可は使用されません。

Kpow

アドオン名は `factorhouse_kpow` で、名前空間は `factorhouse` です。アドオンの公開は、Factorhouse で行います。

アドオンの詳細については、Kpow ドキュメントの「AWS Marketplace LM」を参照してください。

サービスアカウント名

サービスアカウント名は `kpow` です。

AWS マネージド IAM ポリシー

マネージドポリシーは AWSLicenseManagerConsumptionPolicy です。詳細については、AWS マネージドポリシーのリファレンスガイドの「AWSLicenseManagerConsumptionPolicy」を参照してください。

必要な IAM ロールを作成するコマンド

次のコマンドではクラスターの IAM OpenID Connect (OIDC) プロバイダーが必要です。既に存在しているかどうかを確認する、または作成するには、the section called “IAM OIDC プロバイダー” を参照してください。*my-cluster* をクラスターの名前に置き換え、*my-kpow-role* を IAM ロールの名前に置き換えます。このコマンドを使用するにはデバイスに eksctl がインストールされている必要があります。別のツールを使用してロールを作成し、Kubernetes サービスアカウントに注釈を付ける必要がある場合は、the section called “IAM ロールを割り当てる” を参照してください。

```
eksctl create iamserviceaccount --name kpow --namespace factorhouse --cluster my-
cluster --role-name my-kpow-role \
    --role-only --attach-policy-arn arn:aws:iam::aws:policy/service-role/
AWSLicenseManagerConsumptionPolicy --approve
```

カスタム権限

このアドオンではカスタム許可は使用されません。

Kubecost

アドオン名は kubecost_kubecost で、名前空間は kubecost です。アドオンの公開は、Kubecost で行います。

アドオンの詳細については、Kubecost ドキュメントの「AWS Cloud Billing Integration」を参照してください。

クラスターに「Amazon EBS で Kubernetes ボリュームを保存する」をインストールしておく必要があります。インストールしていない場合は、エラーが発生します。

サービスアカウント名

サービスアカウントはこのアドオンでは使用されません。

AWS マネージド IAM ポリシー

このアドオンではマネージドポリシーは使用されません。

カスタム IAM 許可

このアドオンではカスタム許可は使用されません。

Kasten

アドオン名は `kasten_k10` で、名前空間は `kasten-io` です。アドオンの公開は、Kasten by Veeam で行います。

アドオンの詳細については、Kasten ドキュメントの「Installing K10 on AWS using Amazon EKS Add-on」を参照してください。

デフォルトが `StorageClass` のクラスターに Amazon EBS CSI ドライバーがインストールされている必要があります。

サービスアカウント名

サービスアカウント名は `k10-k10` です。

AWS マネージド IAM ポリシー

マネージドポリシーは AWSLicenseManagerConsumptionPolicy です。詳細については、AWS マネージドポリシーのリファレンスガイドの「AWSLicenseManagerConsumptionPolicy」を参照してください。

必要な IAM ロールを作成するコマンド

次のコマンドではクラスターの IAM OpenID Connect (OIDC) プロバイダーが必要です。既に存在しているかどうかを確認する、または作成するには、「the section called “IAM OIDC プロバイダー”」を参照してください。*my-cluster* をクラスターの名前に置き換え、*my-kasten-role* をロールの名前に置き換えます。このコマンドを使用するにはデバイスに eksctl がインストールされている必要があります。別のツールを使用してロールを作成し、Kubernetes サービスアカウントに注釈を付ける必要がある場合は、the section called “IAM ロールを割り当てる” を参照してください。

```
eksctl create iamserviceaccount --name k10-k10 --namespace kasten-io --cluster my-
cluster --role-name my-kasten-role \
    --role-only --attach-policy-arn arn:aws:iam::aws:policy/service-role/
AWSLicenseManagerConsumptionPolicy --approve
```

カスタム権限

このアドオンではカスタム許可は使用されません。

Kong

アドオン名は kong_konnect-ri で、名前空間は kong です。アドオンの公開は、Kong で行います。

アドオンの詳細については、Kong ドキュメントの「Installing the Kong Gateway EKS Add-on」を参照してください。

クラスターに「Amazon EBS で Kubernetes ボリュームを保存する」をインストールしておく必要があります。インストールしていない場合は、エラーが発生します。

サービスアカウント名

サービスアカウントはこのアドオンでは使用されません。

AWS マネージド IAM ポリシー

このアドオンではマネージドポリシーは使用されません。

カスタム IAM 許可

このアドオンではカスタム許可は使用されません。

LeakSignal

アドオン名は leaksignal_leakagent で、名前空間は leakagent です。アドオンの公開は、LeakSignal で行います。

アドオンの詳細については、LeakSignal ドキュメントの https://www.leaksignal.com/docs/LeakAgent/Deployment/AWS%20EKS%20Addon/[Install the LeakAgent add-on] を参照してください。

クラスターに「Amazon EBS で Kubernetes ボリュームを保存する」をインストールしておく必要があります。インストールしていない場合は、エラーが発生します。

サービスアカウント名

サービスアカウントはこのアドオンでは使用されません。

AWS マネージド IAM ポリシー

このアドオンではマネージドポリシーは使用されません。

カスタム IAM 許可

このアドオンではカスタム許可は使用されません。

NetApp

アドオン名は `netapp_trident-operator` で、名前空間は `trident` です。アドオンの公開は、NetApp で行います。

アドオンの詳細についてはNetApp ドキュメントの「Configure the Trident EKS add-on」を参照してください。

サービスアカウント名

サービスアカウントはこのアドオンでは使用されません。

AWS マネージド IAM ポリシー

このアドオンではマネージドポリシーは使用されません。

カスタム IAM 許可

このアドオンではカスタム許可は使用されません。

New Relic

アドオン名は `new-relic_kubernetes-operator` で、名前空間は `newrelic` です。アドオンの公開は、New Relic で行います。

アドオンの詳細については、New Relic ドキュメントの「Installing the New Relic Add-on for EKS」を参照してください。

サービスアカウント名

サービスアカウントはこのアドオンでは使用されません。

AWS マネージド IAM ポリシー

このアドオンではマネージドポリシーは使用されません。

カスタム IAM 許可

このアドオンではカスタム許可は使用されません。

Rafay

アドオン名は `rafay-systems_rafay-operator` で、名前空間は `rafay-system` です。アドオンの公開は、Rafay で行います。

アドオンの詳細についてはRafay ドキュメントの「Installing the Rafay Amazon EKS Add-on」を参照してください。

サービスアカウント名

サービスアカウントはこのアドオンでは使用されません。

AWS マネージド IAM ポリシー

このアドオンではマネージドポリシーは使用されません。

カスタム IAM 許可

このアドオンではカスタム許可は使用されません。

Rad Security

- パブリッシャー – RAD SECURITY
- 名前 – `rad-security_rad-security`
- 名前空間 – `ksoc`
- サービスアカウント名 – サービスアカウントはこのアドオンでは使用されません。
- AWS マネージド IAM ポリシー – このアドオンではマネージドポリシーは使用されません。
- カスタム IAM アクセス許可 – このアドオンではカスタムアクセス許可は使用されません。
- セットアップと使用に関する指示 – Rad Security ドキュメントの「AWS マーケットプレイス経由でのRADのインストール」を参照してください。

SolarWinds

- パブリッシャー – SOLARWINDS
- 名前 – `solarwinds_swo-k8s-collector-addon`
- 名前空間 – `solarwinds`
- サービスアカウント名 – サービスアカウントはこのアドオンでは使用されません。
- AWS マネージド IAM ポリシー – このアドオンではマネージドポリシーは使用されません。

- カスタム IAM アクセス許可 – このアドオンではカスタムアクセス許可は使用されません。
- セットアップと使用に関する指示 – SolarWinds ドキュメントの「Amazon EKSクラスターを監視する」を参照してください。

Solo

アドオン名は `solo-io_istio-distro` で、名前空間は `istio-system` です。アドオンの公開は、Solo で行います。

アドオンの詳細については、Solo.io ドキュメントの「Installing Istio」を参照してください。

サービスアカウント名

サービスアカウントはこのアドオンでは使用されません。

AWS マネージド IAM ポリシー

このアドオンではマネージドポリシーは使用されません。

カスタム IAM 許可

このアドオンではカスタム許可は使用されません。

Snyk

- パブリッシャー – SNYK
- 名前 – `snyk_runtime-sensor`
- 名前空間 – `snyk_runtime-sensor`
- サービスアカウント名 – サービスアカウントはこのアドオンでは使用されません。
- AWS マネージド IAM ポリシー – このアドオンではマネージドポリシーは使用されません。
- カスタム IAM アクセス許可 – このアドオンではカスタムアクセス許可は使用されません。
- セットアップと使用に関する指示 – Snyk ユーザードキュメントの「Snyk runtime sensor」を参照してください。

Stormforge

アドオン名は `stormforge_optimize-Live` で、名前空間は `stormforge-system` です。アドオンの公開は、Stormforge で行います。

アドオンの詳細については、StormForge ドキュメントの「Installing the StormForge Agent」を参照してください。

サービスアカウント名

サービスアカウントはこのアドオンでは使用されません。

AWS マネージド IAM ポリシー

このアドオンではマネージドポリシーは使用されません。

カスタム IAM 許可

このアドオンではカスタム許可は使用されません。

SUSE

- パブリッシャー – SUSE
- 名前 – `suse_observability-agent`
- 名前空間 – `suse-observability`
- サービスアカウント名 – サービスアカウントはこのアドオンでは使用されません。
- AWS マネージド IAM ポリシー – このアドオンではマネージドポリシーは使用されません。
- カスタム IAM アクセス許可 – このアドオンではカスタムアクセス許可は使用されません。
- セットアップと使用手順 – SUSE のドキュメントの「Quick Start」を参照してください。

Splunk

アドオン名は `splunk_splunk-otel-collector-chart` で、名前空間は `splunk-monitoring` です。アドオンの公開は、Splunk で行います。

アドオンの詳細については、Splunk ドキュメントの「Install the Splunk add-on for Amazon EKS」を参照してください。

サービスアカウント名

サービスアカウントはこのアドオンでは使用されません。

AWS マネージド IAM ポリシー

このアドオンではマネージドポリシーは使用されません。

カスタム IAM 許可

このアドオンではカスタム許可は使用されません。

Teleport

アドオン名は `teleport_teleport` で、名前空間は `teleport` です。アドオンの公開は、Teleport で行います。

アドオンの詳細については、Teleport ドキュメントの「How Teleport Works」を参照してください。

サービスアカウント名

サービスアカウントはこのアドオンでは使用されません。

AWS マネージド IAM ポリシー

このアドオンではマネージドポリシーは使用されません。

カスタム IAM 許可

このアドオンではカスタム許可は使用されません。

Tetrate

アドオン名は `tetrate-io_istio-distro` で、名前空間は `istio-system` です。アドオンの公開は、Tetrate で行います。

アドオンの詳細については、Tetrate Istio Distro ウェブサイトを参照してください。

サービスアカウント名

サービスアカウントはこのアドオンでは使用されません。

AWS マネージド IAM ポリシー

このアドオンではマネージドポリシーは使用されません。

カスタム IAM 許可

このアドオンではカスタム許可は使用されません。

Upbound Universal Crossplane

アドオン名は `upbound_universal-crossplane` で、名前空間は `upbound-system` です。アドオンの公開は、Upbound で行います。

アドオンの詳細については、Upbound ドキュメントの「Upbound Universal Crossplane (UXP)」を参照してください。

サービスアカウント名

サービスアカウントはこのアドオンでは使用されません。

AWS マネージド IAM ポリシー

このアドオンではマネージドポリシーは使用されません。

カスタム IAM 許可

このアドオンではカスタム許可は使用されません。

Upwind

アドオン名は upwind で、名前空間は upwind です。アドオンの公開は、Upwind で行います。

アドオンの詳細についてはUpwind のドキュメントを参照してください。

サービスアカウント名

サービスアカウントはこのアドオンでは使用されません。

AWS マネージド IAM ポリシー

このアドオンではマネージドポリシーは使用されません。

カスタム IAM 許可

このアドオンではカスタム許可は使用されません。

Amazon EKS アドオンを作成する

Amazon EKS アドオンはAmazon EKS クラスター用のアドオンソフトウェアです。すべての Amazon EKS アドオン:

- 最新のセキュリティパッチやバグ修正が含まれています。
- AWS によって Amazon EKS で動作することが検証されています。
- アドオンソフトウェアの管理に必要な作業量を削減します。

Amazon EKS アドオンは eksctl、AWS マネジメントコンソール、AWS CLI を使用して作成できます。アドオンに IAM ロールが必要な場合、ロールの作成の詳細については Amazon EKS アドオンの特定のア ドオンの詳細を参照してください。

前提条件

アドオンを作成する前に、次を完了します:

- クラスターのアドオンを作成する前に、そのクラスターが存在している必要があります。詳細については、the section called “クラスターを作成する” を参照してください。
- アドオンに IAM ロールが必要かどうかを確認します。詳細については、the section called “互換性を確認する” を参照してください。
- Amazon EKS アドオンバージョンがクラスターと互換性があることを検証します。詳細については、the section called “互換性を確認する” を参照してください。
- 使用しているコンピュータまたは AWS CloudShell に、eksctl コマンドラインツールのバージョン 0.190.0 以降がインストールされていることを確認します。詳細については、eksctl ウェブサイトの「インストール」を参照してください。

手順

Amazon EKS アドオンは eksctl、AWS マネジメントコンソール、AWS CLI を使用して作成できます。アドオンに IAM ロールが必要な場合、ロールの作成の詳細については、「使用可能な AWS の Amazon EKS アドオン」の特定のアドオンの詳細を参照してください。

アドオンの作成 (eksctl)

1. クラスターバージョンで利用可能なアドオンの名前を表示します。*1.37* をお持ちのクラスターのバージョンに置き換えます。

   ```
   eksctl utils describe-addon-versions --kubernetes-version 1.37 | grep AddonName
   ```

   出力例は次のとおりです。

   ```
   "AddonName": "aws-ebs-csi-driver",
                           "AddonName": "coredns",
                           "AddonName": "kube-proxy",
                           "AddonName": "vpc-cni",
                           "AddonName": "adot",
                           "AddonName": "dynatrace_dynatrace-operator",
   ```

```
                    "AddonName": "upbound_universal-crossplane",
                    "AddonName": "teleport_teleport",
                    "AddonName": "factorhouse_kpow",
                    [...]
```

2. 作成するアドオンで使用できるバージョンを表示します。*1.37* をお持ちのクラスターのバージョンに置き換えます。*name-of-addon* を表示するバージョンのアドオン名に置き換えます。名前は前のステップで返された名前のいずれかである必要があります。

```
eksctl utils describe-addon-versions --kubernetes-version 1.37 --name name-of-addon |
 grep AddonVersion
```

次の出力はvpc-cni という名前のアドオンに対して返される内容の例です。このアドオンには複数のバージョンがあることがわかります。

```
"AddonVersions": [
    "AddonVersion": "v1.12.0-eksbuild.1",
    "AddonVersion": "v1.11.4-eksbuild.1",
    "AddonVersion": "v1.10.4-eksbuild.1",
    "AddonVersion": "v1.9.3-eksbuild.1",
```

a. 作成するアドオンが Amazon EKS か AWS Marketplace アドオンか判断します。AWS Marketplace にはサードパーティ製のアドオンがあり、アドオンを作成するために追加のステップを実行する必要があります。

```
eksctl utils describe-addon-versions --kubernetes-version 1.37 --name name-of-
addon | grep ProductUrl
```

出力が返されない場合、アドオンは Amazon EKS アドオンです。出力が返された場合、アドオンは AWS Marketplace アドオンです。次の出力はteleport_teleport という名前のアドオンのものです。

```
"ProductUrl": "https://aws.amazon.com/marketplace/pp?sku=3bda70bb-566f-4976-806c-
f96faef18b26"
```

返される URL を含んだ AWS Marketplace のアドオンの詳細について説明します。アドオンにサブスクリプションが必要な場合、AWS Marketplace を通じてアドオンにサブスクライブできます。AWS Marketplace からアドオンを作成する場合、アドオンの作成に使用する IAM プリンシパルには AWSServiceRoleForAWSLicenseManagerRole のサービスにリンクされたロー

ルを作成する許可が必要です。IAM エンティティへの許可の割り当てに関する詳細については、IAM ユーザーガイドの「IAM アイデンティティ許可の追加と削除」を参照してください。

3. Amazon EKS アドオンを作成します。コマンドをコピーし、次のように *user-data* を置き換えます:

   - *#######* の部分は自分のクラスター名に置き換えます。
   - *#####*を作成するアドオンの名前に置き換えます。
   - 最新バージョンよりも前のバージョンのアドオンが必要な場合、*##*を使用する前のステップの出力で返されたバージョン番号に置き換えます。
   - アドオンがサービスアカウントロールを使用している場合、*111122223333* をアカウント ID に置き換え、*role-name* をロールの名前に置き換えます。サービスアカウントのロールを作成する手順については、作成するアドオンのドキュメントを参照してください。アドオンのリストについては、「the section called “ AWS アドオン”」を参照してください。サービスアカウントロールを指定するにはクラスター用に IAM OpenID Connect (OIDC) プロバイダーが必要です。クラスター用に持っているかどうかを確認、あるいは作成するには、「the section called “IAM OIDC プロバイダー”」を参照してください。

     アドオンがサービスアカウントロールを使用していない場合、`--service-account-role-arnarn:aws:iam::111122223333:role/role-name` を削除してください。
   - このコマンド例はアドオンの既存のセルフマネージドバージョンがある場合、その設定を上書きします。既存のセルフマネージド型アドオンの設定を上書きしたくない場合、*--force* オプションを削除してください。オプションを削除し、Amazon EKS アドオンが既存のセルフマネージド型アドオンの設定を上書きする必要がある場合、Amazon EKS アドオンの作成が失敗します。その場合、競合を解決するためのエラーメッセージが表示されます。このオプションを指定する前に、自分が管理する必要がある設定を Amazon EKS アドオンが管理していないことを確認してください。これらの設定はこのオプションの指定により上書きされます。

     ```
     eksctl create addon --cluster my-cluster --name name-of-addon --version latest \
         --service-account-role-arn arn:aws:iam::111122223333:role/role-name --force
     ```

     コマンドにすべての利用可能なオプションのリストを確認できます。

     ```
     eksctl create addon --help
     ```

     利用可能なオプションの詳細については、`eksctl` ドキュメントの「アドオン」を参照してください。

アドオンの作成 (AWS コンソール)

1. Amazon EKS コンソールを開きます。
2. 左のナビゲーションペインで [クラスター] を選択してください。
3. アドオンを作成するクラスターの名前を選択してください。
4. [アドオン] タブを選択してください。
5. [その他のアドオンを入手] を選択してください。
6. [アドオンの選択] ページで、クラスターに追加するアドオンを選択してください。[Amazon EKS アドオン] と [AWS Marketplace アドオン] は必要な数だけ追加できます。

   AWS Marketplace アドオンの場合、アドオンを作成するために使用する IAM プリンシパルにはAWS ライセンスマネージャーからアドオンの資格を読み取る許可が必要です。AWSライセンスマネージャー はユーザーに代わってライセンスを管理する AWS リソースを許可する AWSServiceRoleForAWSLicenseManagerRole のサービスリンクロール (SLR が必要になります。SLR はアカウントごとに 1 回限りの要件であり、アドオンやクラスターごとに別々の SLR を作成する必要はありません。IAM プリンシパルへの許可の割り当ての詳細については、IAM ユーザーガイドの「IAM ID のアクセス許可の追加および削除」を参照してください。

   インストールする [AWS Marketplace アドオン] がリストされていない場合は、ページ番号を選択して追加のページ結果を表示するか、または検索ボックスで検索できます。[フィルタリング] オプションで、[カテゴリ]、[ベンダー]、または [料金モデル] によってフィルタリングして、検索結果からアドオンを選択することもできます。インストールするアドオンを選択したら、[次へ] を選択してください。
7. [選択したアドオンセッティングの設定] ページで、次の操作を行います:
   a. [サブスクリプションオプションを表示] を選択し、[サブスクリプションのオプション] フォームを開きます。[料金の詳細] と [法務] のセクションを確認し、[登録] ボタンを選択して続行します。
   b. [バージョン] で、インストールするバージョンを選択してください。作成する個々のアドオンが別のバージョンを推奨している場合を除き、[最新] と表示されたバージョンをお勧めします。アドオンに推奨バージョンがあるかどうかを確認するには作成しているアドオンのドキュメントを参照してください。アドオンのリストについては、the section called “ AWS アドオン” を参照してください。
   c. アドオンのロールを設定するには EKS Pod Identity の IAM ロールおよびサービスアカウント用の IAM ロール (IRSA) の 2 つのオプションがあります。希望するオプションについては以下の適切なステップに従います。選択したすべてのアドオンの [ステータス] で [サブスクリプションが必要] と表示されている場合、[次へ] を選択してください。クラスターが作成された後にサ

ブスクライブするまで、それ以上これらのアドオンを設定することはできません。[ステータス] で [サブスクリプションが必要] と表示されていないアドオンの場合、次を実行してください:

i. [サービスアカウント用の ポッド· アイデンティティー IAM ロール] で、既存の EKS Pod Identity IAM ロールを使用するか、または [推奨役割を作成] ボタンを使用して EKS Pod Identity IAM 役割を作成できます。このフィールドには適切な信頼ポリシーを持つオプションのみが表示されます。選択するロールがない場合は一致する信頼ポリシーを持つ既存のロールが存在しないことになります。選択したアドオンのサービスアカウント用の EKS Pod Identity IAM ロールを設定するには [推奨ロールを作成] を選択してください。ロール作成ウィザードが別のウィンドウで開きます。ウィザードは次のようにロールの情報を自動的に入力してください。EKS Pod Identity IAM ロールを作成するアドオンごとに、IAM ウィザードのステップを次のように実行してください。

- [信頼されたエンティティを選択] ステップでは [EKS] の AWS サービスオプションと [EKS – Pod Identity] のユースケースが事前に選択されており、アドオンのために適切な信頼ポリシーが自動的に入力されます。例えば、ロールは「the section called “EKS Pod Identity の利点”」で詳述されているように、pods.eks.amazonaws.com の IAM プリンシパルを含む適切な信頼ポリシーを使用して作成されます。[次へ] を選択してください。
- [アクセス許可の追加] ステップではロールポリシー用の適切なマネージドポリシーがアドオンのために事前に選択されています。例えば、Amazon VPC CNI アドオンの場合、ロールは「the section called “Amazon VPC CNI Kubernetes用プラグイン”」で詳述されているように、マネージドポリシー `AmazonEKS_CNI_Policy` を使用して作成されます。[次へ] を選択します。
- [名前、確認、作成] のステップにおいて、[ロール名] で、アドオンのためにデフォルトのロール名が自動的に入力されます。例えば、Amazon VPC CNI アドオンの場合、ロールは AmazonEKSPodIdentityAmazonVPCCNIRole という名前で作成されます。[説明] ではアドオンのためにデフォルトの適切な説明が自動的に入力されます。例えば、Amazon VPC CNI アドオンの場合、ロールは AWS リソースにアクセスすることを「Amazon EKS クラスターで実行されているポッドに許可します」という説明で作成されます。[信頼ポリシー] で、アドオンのために事前入力された信頼ポリシーを表示します。[ロールの作成] を選択してください。

  注: デフォルトのロール名を保持すると、EKS は新しいクラスターでアドオン用のロールを事前に選択したり、既存のクラスターにアドオンを追加したりできます。この名前は引き続き上書きでき、ロールはクラスター全体でアドオンのために使用できますが、ロールはドロップダウンから手動で選択する必要があります。

ii. [ステータス] が [サブスクリプションが必要] でなく、IRSA を使用してロールを設定するアドオンについてはIAM ポリシーを作成してロールにアタッチするために作成するアドオン

に関するドキュメントを参照してください。アドオンのリストについては、「the section called “ AWS アドオン”」を参照してください。IAM ロールを選択するにはクラスター用に OpenID Connect (OIDC) プロバイダーが必要です。クラスター用に持っているかどうかを確認、あるいは作成するには、the section called “IAM OIDC プロバイダー” を参照してください。

iii. [オプション設定のセッティング] を選択してください。

iv. アドオンに設定が必要な場合、設定値 ボックスに入力します。アドオンに設定情報が必要かどうかを判断するには作成するアドオンのドキュメントを参照してください。アドオンのリストについては、the section called “ AWS アドオン” を参照してください。

v. [競合解決方法] で利用可能なオプションをいずれか選択してください。[コンフリクト解決方法] で [上書き] を選択すると、既存のアドオンでの 1 つ以上の設定が、Amazon EKS アドオンの設定で上書きされます。このオプションが有効でない状態で既存の設定との競合が発生する場合はオペレーションが失敗します。表示されたエラーメッセージを使用して、競合をトラブルシューティングできます。このオプションを選択する前に、自分で管理する必要のある設定が、Amazon EKS アドオンにより管理されていないことを確認します。

vi. アドオンを特定の名前空間にインストールする場合は、その名前空間を [名前空間] フィールドに入力します。AWS アドオンとコミュニティアドオンでは、アドオンをインストールするカスタム Kubernetes 名前空間を定義できます。詳細については、the section called “アドオン用のカスタム名前空間” を参照してください。

vii.[次へ] を選択します。

8. [確認と追加] ページで、[作成] を選択してください。アドオンのインストールが完了した後、インストールしたアドオンが表示されます。

9. インストールしたアドオンにサブスクリプションが必要な場合、次のステップを実行してください:

   a. アドオンの右下隅にある [サブスクライブ] ボタンを選択してください。AWS Marketplace のアドオンのページが表示されます。[製品概要] や [価格情報] など、アドオンに関する情報をお読みください。

   b. アドオンページの右上にある [サブスクライブを続ける] ボタンを選択してください。

   c. [利用規約] をよくお読みください。利用規約に同意する場合、[利用規約に同意] を選択してください。サブスクリプションの処理に数分かかることがあります。サブスクリプションの処理中、[Amazon EKS コンソールに戻る] ボタンがグレー表示されます。

   d. サブスクリプションの処理が完了すると、[Amazon EKS コンソールに戻る] ボタンがグレー表示されなくなります。ボタンを選択し、クラスターの Amazon EKS コンソールの [アドオン] タブに戻ります。

e. サブスクライブしているアドオンについては [削除して再インストール] を選択し、次に [アドオンの再インストール] を選択してください。アドオンのインストールに数分かかることがあります。インストールが完了すると、アドオンを設定できます。

アドオンの作成 (AWS CLI)

1. ご使用のデバイスまたは AWS CloudShell で、バージョン 2.12.3 以降、または AWS コマンドラインインターフェイス (AWS CLI) のバージョン 1.27.160 以降が必要です。現在のバージョンを確認するには、「aws --version | cut -d / -f2 | cut -d ' ' -f1」を参照してください。yum、apt-get、macOS 用の Homebrew などのパッケージマネージャーは、多くの場合 AWS CLI の最新バージョンより数バージョン古くなっています。最新バージョンをインストールするには、 AWS コマンドラインインターフェイスユーザーガイドの「インストール」および「aws configure を使用したクイック設定」を参照してください。AWS CloudShell にインストールされている AWS CLI バージョンも最新バージョンより数バージョン遅れることがあります。更新するには、「AWS CloudShell ユーザーガイド」の「ホームディレクトリへの AWS CLI のインストール」を参照してください。
2. 利用可能なアドオンを確認します。利用可能なすべてのアドオン、種類、発行元を確認できます。AWS Marketplace を介して利用可能なアドオンの URL も確認できます。*1.37* をお持ちのクラスターのバージョンに置き換えます。

```
aws eks describe-addon-versions --kubernetes-version 1.37 \
    --query 'addons[].{MarketplaceProductUrl: marketplaceInformation.productUrl,
 Name: addonName, Owner: owner, Publisher: publisher, Type: type}' --output table
```

出力例は次のとおりです。

```
-------------------------------------------------------------------------------------------------
|
 DescribeAddonVersions
  |
+-----------------------------------------------------------------
+------------------------------+------------------+--------------
+---------------------+
|                     MarketplaceProductUrl                      |            Name
          |      Owner       |  Publisher   |        Type         |
+-----------------------------------------------------------------
+------------------------------+------------------+--------------
+---------------------+
```

```
|  None                                                                |  aws-ebs-csi-driver
           |  aws             |  eks         |  storage                |
|  None                                                                |  coredns
           |  aws             |  eks         |  networking             |
|  None                                                                |  kube-proxy
           |  aws             |  eks         |  networking             |
|  None                                                                |  vpc-cni
           |  aws             |  eks         |  networking             |
|  None                                                                |  adot
           |  aws             |  eks         |  observability          |
| https://aws.amazon.com/marketplace/pp/prodview-brb73nceicv7u |
 dynatrace_dynatrace-operator |  aws-marketplace |  dynatrace    |  monitoring
 |
| https://aws.amazon.com/marketplace/pp/prodview-uhc2iwi5xysoc |  upbound_universal-
crossplane |  aws-marketplace |  upbound     |  infra-management    |
| https://aws.amazon.com/marketplace/pp/prodview-hd2ydsrgqy4li |  teleport_teleport
          |  aws-marketplace |  teleport    |  policy-management   |
| https://aws.amazon.com/marketplace/pp/prodview-vgghgqdsplhvc |  factorhouse_kpow
          |  aws-marketplace |  factorhouse |  monitoring          |
|  [...]                                                               |  [...]
           |  [...]           |  [...]       |  [...]                  |
+----------------------------------------------------------------
+------------------------------+------------------+--------------
+----------------------+
```

出力が異なる場合があります。この出力例ではタイプ networking で利用可能なアドオンが 3 種類あり、タイプ eks の発行元でアドオンが 5 種類あります。Owner 列に aws-marketplace があるアドオンはインストールする前にサブスクリプションが必要な場合があります。URL にアクセスしてアドオンの詳細を確認したり、アドオンをサブスクライブしたりできます。

3. アドオンごとに利用可能なバージョンを確認できます。*1.37* をクラスターのバージョンに置き換え、*vpc-cni* を前のステップで返されたアドオンの名前に置き換えます。

```
aws eks describe-addon-versions --kubernetes-version 1.37 --addon-name vpc-cni \
    --query 'addons[].addonVersions[].{Version: addonVersion, Defaultversion:
 compatibilities[0].defaultVersion}' --output table
```

出力例は次のとおりです。

```
-----------------------------------------
|          DescribeAddonVersions         |
+-----------------+-----------------------+
```

```
| Defaultversion  |       Version        |
+-----------------+----------------------+
|  False          |  v1.12.0-eksbuild.1  |
|  True           |  v1.11.4-eksbuild.1  |
|  False          |  v1.10.4-eksbuild.1  |
|  False          |  v1.9.3-eksbuild.1   |
+-----------------+----------------------+
```

Defaultversion 列に True と表示されているバージョンはデフォルトでアドオンが作成されたバージョンです。

4. (オプション) 次のコマンドを実行して、選択したアドオンの設定オプションを調べます:

```
aws eks describe-addon-configuration --addon-name vpc-cni --addon-version v1.12.0-
eksbuild.1
```

```
{
    "addonName": "vpc-cni",
    "addonVersion": "v1.12.0-eksbuild.1",
    "configurationSchema": "{\"$ref\":\"#/definitions/VpcCni\",\"$schema\":\"http://
json-schema.org/draft-06/schema#\",\"definitions\":{\"Cri\":{\"additionalProperties
\":false,\"properties\":{\"hostPath\":{\"$ref\":\"#/definitions/HostPath\"}},\"title
\":\"Cri\",\"type\":\"object\"},\"Env\":{\"additionalProperties\":false,\"properties
\":{\"ADDITIONAL_ENI_TAGS\":{\"type\":\"string\"},\"AWS_VPC_CNI_NODE_PORT_SUPPORT
\":{\"format\":\"boolean\",\"type\":\"string\"},\"AWS_VPC_ENI_MTU\":{\"format\":
\"integer\",\"type\":\"string\"},\"AWS_VPC_K8S_CNI_CONFIGURE_RPFILTER\":{\"format
\":\"boolean\",\"type\":\"string\"},\"AWS_VPC_K8S_CNI_CUSTOM_NETWORK_CFG\":{\"format
\":\"boolean\",\"type\":\"string\"},\"AWS_VPC_K8S_CNI_EXTERNALSNAT\":{\"format\":
\"boolean\",\"type\":\"string\"},\"AWS_VPC_K8S_CNI_LOGLEVEL\":{\"type\":\"string\"},
\"AWS_VPC_K8S_CNI_LOG_FILE\":{\"type\":\"string\"},\"AWS_VPC_K8S_CNI_RANDOMIZESNAT
\":{\"type\":\"string\"},\"AWS_VPC_K8S_CNI_VETHPREFIX\":{\"type\":\"string\"},
\"AWS_VPC_K8S_PLUGIN_LOG_FILE\":{\"type\":\"string\"},\"AWS_VPC_K8S_PLUGIN_LOG_LEVEL
\":{\"type\":\"string\"},\"DISABLE_INTROSPECTION\":{\"format\":\"boolean\",\"type
\":\"string\"},\"DISABLE_METRICS\":{\"format\":\"boolean\",\"type\":\"string
\"},\"DISABLE_NETWORK_RESOURCE_PROVISIONING\":{\"format\":\"boolean\",\"type
\":\"string\"},\"ENABLE_POD_ENI\":{\"format\":\"boolean\",\"type\":\"string
\"},\"ENABLE_PREFIX_DELEGATION\":{\"format\":\"boolean\",\"type\":\"string\"},
\"WARM_ENI_TARGET\":{\"format\":\"integer\",\"type\":\"string\"},\"WARM_PREFIX_TARGET
\":{\"format\":\"integer\",\"type\":\"string\"}},\"title\":\"Env\",\"type\":
\"object\"},\"HostPath\":{\"additionalProperties\":false,\"properties\":{\"path
\":{\"type\":\"string\"}},\"title\":\"HostPath\",\"type\":\"object\"},\"Limits
\":{\"additionalProperties\":false,\"properties\":{\"cpu\":{\"type\":\"string
\"},\"memory\":{\"type\":\"string\"}},\"title\":\"Limits\",\"type\":\"object\"},
```

```
\"Resources\":{\"additionalProperties\":false,\"properties\":{\"limits\":{\"$ref\":
\"#/definitions/Limits\"},\"requests\":{\"$ref\":\"#/definitions/Limits\"}},\"title
\":\"Resources\",\"type\":\"object\"},\"VpcCni\":{\"additionalProperties\":false,
\"properties\":{\"cri\":{\"$ref\":\"#/definitions/Cri\"},\"env\":{\"$ref\":\"#/
definitions/Env\"},\"resources\":{\"$ref\":\"#/definitions/Resources\"}},\"title\":
\"VpcCni\",\"type\":\"object\"}}}"
}
```

出力は標準の JSON スキーマです。

上記のスキーマで使用できる有効な設定値の例 (JSON フォーマット) を次に示します。

```
{
  "resources": {
    "limits": {
      "cpu": "100m"
    }
  }
}
```

上記のスキーマで使用できる有効な設定値の例 (YAML フォーマット) を次に示します。

```
  resources:
    limits:
      cpu: 100m
```

5. アドオンに IAM アクセス許可が必要かどうかを確認します。その場合は(1) EKS Pod Identity またはサービスアカウントの IAM ロール (IRSA 使用するかどうかを決定し、(2) アドオンで使用する IAM ロールの ARN を決定し、(3) アドオンで使用される Kubernetes サービスアカウントの名前を決定する必要があります。詳細については、「the section called “IAM情報を取得する”」を参照してください。

   - Amazon EKS ではアドオンが EKS Pod Identity をサポートしている場合はEKS Pod Identity の使用を推奨しています。そのためにはPod Identity エージェントがクラスター にインストールされている必要があります。アドオンで Pod Identity を使用する場合の詳細については、「the section called “IAM ロール”」を参照してください。
   - アドオンまたはクラスターが EKS Pod Identity 用に設定されていない場合は、IRSA を使用します。クラスターで IRSA が設定されていることを確認します。
   - Amazon EKS アドオンのドキュメントを参照して、アドオンに IAM アクセス許可と、関連する Kubernetes サービスアカウントの名前が必要かどうかを確認します。

a. Amazon EKS アドオンを作成します。デバイスに沿ったコマンドをコピーします。必要に応じてコマンドに次の変更を加え、変更したコマンドを実行してください:

- *#######* の部分は自分のクラスター名に置き換えます。
- *vpc-cni* を前のステップの出力で返された作成するアドオン名に置き換えます。
- *version-number* を使用する前のステップの出力で返されたバージョンに置き換えます。
- カスタム Kubernetes 名前空間にアドオンをインストールする場合は、`--namespace-config 'namespace=<my-namespace>'` オプションを追加してください。このオプションは、AWS アドオンとコミュニティアドオンでのみ使用できます。詳細については、the section called “アドオン用のカスタム名前空間” を参照してください。
- アドオンに IAM アクセス許可が必要ない場合は *<service-account-configuration>* を削除してください。
- 次のいずれかを行います:
  - アドオンが (1) IAM 許可が必要で、(2) クラスターが EKS Pod Identity を使用している場合は*<service-account-configuration>* を次の Pod Identity の関連付けに置き換えます。*<service-account-name>* をアドオンで使用されるサービスアカウント名に置き換えます。*<role-arn>* を IAM ロールの ARN に置き換えます。ロールにはEKS Pod Identity に必要な信頼ポリシーが必要です。

    ```
    --pod-identity-associations 'serviceAccount=<service-account-name>,roleArn=<role-arn>'
    ```

  - アドオンが (1) IAM アクセス許可が必要で、(2) クラスターが IRSA を使用している場合は*<service-account-configuration>* を次の IRSA 設定に置き換えます。*111122223333* を、アカウント ID に置き換えます。また、*role-name* を、作成した既存の IAM ロールの名前に置き換えます。ロールの作成手順については作成するアドオンのドキュメントを参照してください。アドオンのリストについては、「the section called “ AWS アドオン”」を参照してください。サービスアカウントロールを指定するにはクラスター用に IAM OpenID Connect (OIDC) プロバイダーが必要です。クラスター用に持っているかどうかを確認、あるいは作成するには、the section called “IAM OIDC プロバイダー” を参照してください。

    ```
    --service-account-role-arn arn:aws:iam::111122223333:role/role-name
    ```

- このコマンド例は、アドオンの既存のセルフマネージドバージョンの `--configuration-values` オプションがあれば上書きします。これを、文字列やファイル入力などの目的の設定値に置き換えます。設定値を指定したくない場合、`--configuration-values` オプション

を削除します。AWS CLI に既存のセルフマネージド型アドオンの設定を上書きされたくない場合、*--resolve-conflicts OVERWRITE* オプションを削除してください。オプションを削除し、Amazon EKS アドオンが既存のセルフマネージド型アドオンの設定を上書きする必要がある場合、Amazon EKS アドオンの作成が失敗します。その場合、競合を解決するためのエラーメッセージが表示されます。このオプションを指定する前に、自分が管理する必要がある設定を Amazon EKS アドオンが管理していないことを確認してください。これらの設定はこのオプションの指定により上書きされます。

```
aws eks create-addon --cluster-name my-cluster --addon-name vpc-cni --addon-version
 version-number \
    <service-account-configuration> --configuration-values '{"resources":
{"limits":{"cpu":"100m"}}}' --resolve-conflicts OVERWRITE
```

```
aws eks create-addon --cluster-name my-cluster --addon-name vpc-cni --addon-version
 version-number \
   <service-account-configuration> --configuration-values 'file://example.yaml' --
resolve-conflicts OVERWRITE
```

使用可能なすべてのオプションのリストについては、Amazon EKS コマンドラインリファレンスの「 create-addon 」を参照してください。作成したアドオンが前のステップの Owner 列に aws-marketplace が一覧表示されている場合、作成が失敗して次のエラーと同様なエラーメッセージが表示されることがあります。

```
{
    "addon": {
        "addonName": "addon-name",
        "clusterName": "my-cluster",
        "status": "CREATE_FAILED",
        "addonVersion": "version",
        "health": {
            "issues": [
                {
                    "code": "AddonSubscriptionNeeded",
                    "message": "You are currently not subscribed to this add-on. To
 subscribe, visit the AWS Marketplace console, agree to the seller EULA, select the
 pricing type if required, then re-install the add-on"
                }
            ]
        }
    }
```

```
}
```

前の出力のエラーと同様なものが表示された場合、前のステップに出力された URL にアクセスしてアドオンをサブスクライブしてください。サブスクライブしたら `create-addon` コマンドを再実行してください。

Amazon EKS アドオンを更新する

Amazon EKS は新しいバージョンがリリースされた場合、あるいはクラスターを新しい Kubernetes マイナーバージョンに更新しても、アドオンを自動的に更新しません。既存のクラスターのアドオンを更新するには更新を開始する必要があります。更新を開始した後、Amazon EKS がアドオンを更新します。アドオンを更新する前に、アドオンの現行のドキュメントを確認してください。利用可能なアドオンのリストについては、「the section called “ AWS アドオン”」を参照してください。アドオンに IAM ロールが必要な場合、ロールの作成の詳細については、「使用可能な AWS の Amazon EKS アドオン」の特定のアドオンの詳細を参照してください。

前提条件

アドオンを更新する前に、次のことを完了してください。

- アドオンに IAM ロールが必要かどうかを確認します。詳細については、the section called “Amazon EKS アドオン” を参照してください。
- Amazon EKS アドオンバージョンがクラスターと互換性があることを検証します。詳細については、the section called “互換性を確認する” を参照してください。

手順

`eksctl`、AWS マネジメントコンソール、AWS CLI のいずれかを使用して Amazon EKS アドオンを更新できます。

アドオンの更新 (eksctl)

1. クラスターにインストールされている現在のアドオンおよびアドオンのバージョンを確認します。`#######` の部分は自分のクラスター名に置き換えます。

```
eksctl get addon --cluster my-cluster
```

出力例は次のとおりです。

```
NAME        VERSION              STATUS  ISSUES  IAMROLE  UPDATE AVAILABLE
coredns     v1.8.7-eksbuild.2    ACTIVE  0
kube-proxy  v1.23.7-eksbuild.1   ACTIVE  0                v1.23.8-eksbuild.2
vpc-cni     v1.10.4-eksbuild.1   ACTIVE  0                v1.12.0-eksbuild.1,v1.11.4-
eksbuild.1,v1.11.3-eksbuild.1,v1.11.2-eksbuild.1,v1.11.0-eksbuild.1
```

クラスターにあるアドオンおよびバージョンによって、出力が異なる場合があります。前述の出力例ではクラスターの既存のアドオン 2 つは UPDATE AVAILABLE 列に新しいバージョンがあることがわかります。

2. アドオンを更新します。

   a. デバイスに沿ったコマンドをコピーします。必要に応じてコマンドに次の変更を加えます。

   - *#######* の部分は自分のクラスター名に置き換えます。
   - *region-code* を、クラスターのある AWS リージョンに置き換えます。
   - *vpc-cni* を更新する前のステップの出力で返されたアドオンの名前に置き換えます。
   - 利用可能な最新バージョンよりも前のバージョンに更新する場合、*latest* を使用する前のステップの出力で返されたバージョン番号に置き換えます。一部のアドオンには推奨バージョンがあります。詳細については、更新するアドオンのドキュメントを参照してください。アドオンのリストについては、the section called “ AWS アドオン” を参照してください。* アドオンが Kubernetes サービスアカウントおよび IAM ロールを使用する場合、*111122223333* をアカウント ID に置き換え、*role-name* を作成した既存の IAM ロールの名前に置き換えます。ロールの作成手順については作成するアドオンのドキュメントを参照してください。アドオンのリストについては、「the section called “ AWS アドオン”」を参照してください。サービスアカウントロールを指定するにはクラスター用に IAM OpenID Connect (OIDC) プロバイダーが必要です。クラスター用に持っているかどうかを確認、あるいは作成するには、「the section called “IAM OIDC プロバイダー”」を参照してください。

     アドオンが Kubernetes サービスアカウントおよび IAM ロールを使用しない場合、`serviceAccountRoleARN: arn:aws:iam::`*111122223333*`:role/`*role-name* 行を削除してください。
   - *##*オプションはアドオンの既存値を保存します。アドオン設定にカスタム値を設定していて、このオプションを使用しない場合、Amazon EKS は値をデフォルト値で上書きします。このオプションを使用する場合、実稼働クラスターのアドオンを更新する前に、非稼動クラスターのフィールドおよび値変更をテストすることをお勧めします。この値を `overwrite` に変更する場合、すべての設定が Amazon EKS のデフォルト値に変更されます。いずれかの設定にカスタム値を設定した場合、Amazon EKS のデフォルト値で上書きされる可能性があ

ります。この値を none に変更した場合、Amazon EKS は設定の値を一切変更しませんが、更新が失敗する可能性があります。更新に失敗した場合、競合の解決に役立つエラーメッセージが返されます。

```
cat >update-addon.yaml <<EOF
apiVersion: eksctl.io/v1alpha5
kind: ClusterConfig
metadata:
  name: my-cluster
  region: region-code

addons:
- name: vpc-cni
  version: latest
  serviceAccountRoleARN: arn:aws:iam::111122223333:role/role-name
  resolveConflicts: preserve
EOF
```

b. 変更コマンドを実行して update-addon.yaml ファイルを作成します。

c. クラスターに設定ファイルを適用します。

```
eksctl update addon -f update-addon.yaml
```

アドオンの更新の詳細については、eksctl ドキュメントの「アドオンの更新」を参照してください。

アドオンの更新 (AWS コンソール)

1. Amazon EKS コンソールを開きます。
2. 左のナビゲーションペインで、 [クラスター] を選択してください。
3. アドオンを更新するクラスターの名前を選択します。
4. [アドオン] タブを選択します。
5. 更新するアドオンを選択します。
6. [編集] を選択します。
7. [*******を設定] ページで、次を実行します:

a. 使用する [バージョン] を選択します。アドオンには推奨バージョンがある場合があります。詳細については、更新するアドオンのドキュメントを参照してください。アドオンのリストについては、「the section called “ AWS アドオン”」を参照してください。

b. アドオンのロールを設定するには EKS Pod Identity の IAM ロールおよびサービスアカウント用の IAM ロール (IRSA) の 2 つのオプションがあります。希望するオプションについては以下の適切なステップに従います。選択したすべてのアドオンの [ステータス] で [サブスクリプションが必要] と表示されている場合、[次へ] を選択してください。[ステータス] で [サブスクリプションが必要] と表示されていないアドオンの場合、次を実行してください。

   i. [サービスアカウント用の ポッド· アイデンティティー IAM ロール] で、既存の EKS Pod Identity IAM ロールを使用するか、または [推奨役割を作成] ボタンを使用して EKS Pod Identity IAM 役割を作成できます。このフィールドには適切な信頼ポリシーを持つオプションのみが表示されます。選択するロールがない場合は一致する信頼ポリシーを持つ既存のロールが存在しないことになります。選択したアドオンのサービスアカウント用の EKS Pod Identity IAM ロールを設定するには [推奨ロールを作成] を選択してください。ロール作成ウィザードが別のウィンドウで開きます。ウィザードは次のようにロールの情報を自動的に入力してください。EKS Pod Identity IAM ロールを作成するアドオンごとに、IAM ウィザードのステップを次のように実行してください。

   - [信頼されたエンティティを選択] ステップでは [EKS] の AWS サービスオプションと [EKS – Pod Identity] のユースケースが事前に選択されており、アドオンのために適切な信頼ポリシーが自動的に入力されます。例えば、ロールは「the section called “EKS Pod Identity の利点”」で詳述されているように、pods.eks.amazonaws.com の IAM プリンシパルを含む適切な信頼ポリシーを使用して作成されます。[次へ] を選択してください。
   - [アクセス許可の追加] ステップではロールポリシー用の適切なマネージドポリシーがアドオンのために事前に選択されています。例えば、Amazon VPC CNI アドオンの場合、ロールは「the section called “Amazon VPC CNI Kubernetes用プラグイン”」で詳述されているように、マネージドポリシー `AmazonEKS_CNI_Policy` を使用して作成されます。[次へ] を選択します。
   - [名前、確認、作成] のステップにおいて、[ロール名] で、アドオンのためにデフォルトのロール名が自動的に入力されます。例えば、Amazon VPC CNI アドオンの場合、ロールは AmazonEKSPodIdentityAmazonVPCCNIRole という名前で作成されます。[説明] ではアドオンのためにデフォルトの適切な説明が自動的に入力されます。例えば、Amazon VPC CNI アドオンの場合、ロールは AWS リソースにアクセスすることを「Amazon EKS クラスターで実行されているポッドに許可します」という説明で作成されます。[信頼ポリシー] で、アドオンのために事前入力された信頼ポリシーを表示します。[ロールの作成] を選択してください。

> **Note**
>
> デフォルトのロール名を保持すると、EKS は新しいクラスターでアドオン用のロールを事前に選択したり、既存のクラスターにアドオンを追加したりできます。この名前は引き続き上書きでき、ロールはクラスター全体でアドオンのために使用できますが、ロールはドロップダウンから手動で選択する必要があります。

ii. [ステータス] が [サブスクリプションが必要] でなく、IRSA を使用してロールを設定するアドオンについてはIAM ポリシーを作成してロールにアタッチするために作成するアドオンに関するドキュメントを参照してください。アドオンのリストについては、「the section called “ AWS アドオン”」を参照してください。IAM ロールを選択するにはクラスター用に OpenID Connect (OIDC) プロバイダーが必要です。クラスター用に持っているかどうかを確認、あるいは作成するには、the section called “IAM OIDC プロバイダー” を参照してください。

c. [オプションの構成設定] を展開します。

d. [設定値] で、アドオン固有の設定情報を入力してください。詳細については、更新するアドオンのドキュメントを参照してください。アドオンのリストについては、the section called “ AWS アドオン” を参照してください。[コンフリクト解決方法] で、いずれかのオプションを選択してください。アドオン設定にカスタム値を設定している場合、[保存] オプションをお勧めします。このオプションを選択しない場合、Amazon EKS は値をデフォルト値で上書きします。このオプションを使用する場合、実稼働クラスターのアドオンを更新する前に、非稼動クラスターのフィールドおよび値変更をテストすることをお勧めします。この値を [上書き] に変更する場合、すべての設定が Amazon EKS のデフォルト値に変更されます。いずれかの設定にカスタム値を設定した場合、Amazon EKS のデフォルト値で上書きされる可能性があります。この値を [なし] に変更した場合、Amazon EKS は設定の値を一切変更しませんが、更新が失敗する可能性があります。更新に失敗した場合、競合の解決に役立つエラーメッセージが返されます。

8. [変更の保存] をクリックします。

アドオンの更新 (AWS CLI)

1. ご使用のデバイスまたは AWS CloudShell で、バージョン `2.12.3` 以降、または AWS コマンドラインインターフェイス (AWS CLI) のバージョン `1.27.160` 以降が必要です。現在のバージョンを確認するには、「`aws --version | cut -d / -f2 | cut -d ' ' -f1`」を参照してく

ださい。yum、apt-get、macOS 用の Homebrew などのパッケージマネージャーは、多くの場合 AWS CLI の最新バージョンより数バージョン古くなっています。最新バージョンをインストールするには、 AWS コマンドラインインターフェイスユーザーガイドの「インストール」および「aws configure を使用したクイック設定」を参照してください。AWS CloudShell にインストールされている AWS CLI バージョンも最新バージョンより数バージョン遅れることがあります。更新するには「AWS CloudShell ユーザーガイド」の「ホームディレクトリへの AWS CLI のインストール」を参照してください。

2. インストールされているアドオンのリストを参照してください。*#######* の部分は自分のクラスター名に置き換えます。

```
aws eks list-addons --cluster-name my-cluster
```

出力例は次のとおりです。

```
{
    "addons": [
        "coredns",
        "kube-proxy",
        "vpc-cni"
    ]
}
```

3. 更新するアドオンの現在のバージョンを表示します。*my-cluster* をクラスターの名前に置き換えて、*vpc-cni* を更新するアドオンの名前に置き換えます。

```
aws eks describe-addon --cluster-name my-cluster --addon-name vpc-cni --query
 "addon.addonVersion" --output text
```

出力例は次のとおりです。

```
v1.10.4-eksbuild.1
```

4. クラスターのバージョンで使用できるアドオンのバージョンを確認します。*1.37* をクラスターのバージョンに置き換えて、*vpc-cni* を更新するアドオンの名前に置き換えます。

```
aws eks describe-addon-versions --kubernetes-version 1.37 --addon-name vpc-cni \
    --query 'addons[].addonVersions[].{Version: addonVersion, Defaultversion:
 compatibilities[0].defaultVersion}' --output table
```

出力例は次のとおりです。

```
-----------------------------------------
|          DescribeAddonVersions         |
+-----------------+----------------------+
| Defaultversion  |       Version        |
+-----------------+----------------------+
|  False          |  v1.12.0-eksbuild.1  |
|  True           |  v1.11.4-eksbuild.1  |
|  False          |  v1.10.4-eksbuild.1  |
|  False          |  v1.9.3-eksbuild.1   |
+-----------------+----------------------+
```

`Defaultversion` 列に `True` と表示されているバージョンはデフォルトでアドオンが作成されたバージョンです。

5. アドオンを更新してください。デバイスに沿ったコマンドをコピーします。必要に応じてコマンドに次の変更を加え、変更したコマンドを実行してください。このコマンドの詳細については、Amazon EKS コマンドラインリファレンスの「update-addon」を参照してください。
   - *`#######`* の部分は自分のクラスター名に置き換えます。
   - *`vpc-cni`* を前のステップの出力で返された更新するアドオンの名前に置き換えます。
   - *`version-number`* を更新する前のステップの出力で返されたバージョンに置き換えます。一部のアドオンには推奨バージョンがあります。詳細については、更新するアドオンのドキュメントを参照してください。アドオンのリストについては、the section called “ AWS アドオン” を参照してください。* アドオンが Kubernetes サービスアカウントおよび IAM ロールを使用する場合、*`111122223333`* をアカウント ID に置き換え、*`role-name`* を作成した既存の IAM ロールの名前に置き換えます。ロールの作成手順については作成するアドオンのドキュメントを参照してください。アドオンのリストについては、「the section called “ AWS アドオン”」を参照してください。サービスアカウントロールを指定するにはクラスター用に IAM OpenID Connect (OIDC) プロバイダーが必要です。クラスター用に持っているかどうかを確認、あるいは作成するには、「the section called “IAM OIDC プロバイダー”」を参照してください。

     アドオンが Kubernetes サービスアカウントおよび IAM ロールを使用しない場合、`serviceAccountRoleARN: arn:aws:iam::`*`111122223333`*`:role/`*`role-name`* 行を削除してください。
   - `--resolve-conflicts PRESERVE` オプションはアドオンの既存値を保存します。アドオン設定にカスタム値を設定していて、このオプションを使用しない場合、Amazon EKS は値をデフォルト値で上書きします。このオプションを使用する場合、実稼働クラスターのアドオン

を更新する前に、非稼動クラスターのフィールドおよび値変更をテストすることをお勧めします。この値を OVERWRITE に変更する場合、すべての設定が Amazon EKS のデフォルト値に変更されます。いずれかの設定にカスタム値を設定した場合、Amazon EKS のデフォルト値で上書きされる可能性があります。この値を NONE に変更した場合、Amazon EKS は設定の値を一切変更しませんが、更新が失敗する可能性があります。更新に失敗した場合、競合の解決に役立つエラーメッセージが返されます。

- すべてのカスタム設定を削除する場合は*--configuration-values '{}'* オプションを使用して更新を実行してください。これにより、すべてのカスタム設定がデフォルト値に設定されます。カスタム設定を変更しない場合、*--configuration-values* フラグを指定しないでください。カスタム設定を調整する場合、*{}* を新しいパラメータに置き換えます。

```
aws eks update-addon --cluster-name my-cluster --addon-name vpc-cni --addon-version
 version-number \
    --service-account-role-arn arn:aws:iam::111122223333:role/role-name --
configuration-values '{}' --resolve-conflicts PRESERVE
```

6. 更新のステータスを確認します。*my-cluster* をクラスターの名前に置き換えて、*vpc-cni* を更新するアドオンの名前に置き換えます。

```
aws eks describe-addon --cluster-name my-cluster --addon-name vpc-cni
```

出力例は次のとおりです。

```
{
    "addon": {
        "addonName": "vpc-cni",
        "clusterName": "my-cluster",
        "status": "UPDATING",
    }
}
```

ステータスが ACTIVE に変わると更新が完了します。

Amazon EKS アドオンバージョンのクラスターとの互換性を検証する

Amazon EKS アドオンを作成する前に、Amazon EKS アドオンのバージョンがクラスターと互換性があることを検証する必要があります。

アドオンバージョンを記述する API を使用して、EKS アドオンの利用可能なバージョンと、各アドオンバージョンがサポートしている Kubernetes バージョンを一覧表示します。

1. AWS CLI がインストールされ、`aws sts get-caller-identity` と連携していることを確認します。このコマンドが機能しない場合は「AWS CLI の使用を開始する」を参照してください。
2. バージョンの互換性情報を取得するアドオンの名前 (`amazon-cloudwatch-observability` など) を特定します。
3. クラスターの Kubernetes バージョンを特定します (`1.37` など)。
4. AWS CLI を使用して、クラスターの Kubernetes バージョンと互換性のあるアドオンのバージョンを取得します。

```
aws eks describe-addon-versions --addon-name amazon-cloudwatch-observability --
kubernetes-version 1.37
```

出力例は次のとおりです。

```
{
    "addons": [
        {
            "addonName": "amazon-cloudwatch-observability",
            "type": "observability",
            "addonVersions": [
                {
                    "addonVersion": "vX.X.X-eksbuild.X",
                    "architecture": [
                        "amd64",
                        "arm64"
                    ],
                    "computeTypes": [
                        "ec2",
                        "auto",
                        "hybrid"
                    ],
                    "compatibilities": [
                        {
                            "clusterVersion": "1.37",
                            "platformVersions": [
                                "*"
                            ],
                            "defaultVersion": true
                        }
```

```
                    ],
                }
            ]
        }
    ]
}
```

この出力はアドオンのバージョン vX.X.X-eksbuild.X が、Kubernetes クラスターのバージョン 1.37 と互換性があることを示しています。

コンピューティングタイプとのアドオン互換性

describe-addon-versions 出力の computeTypes フィールドはアドオンと EKS 自動モード I マネージドノードまたはハイブリッドノードとの互換性を示します。auto とマークされたアドオンはEKS 自動モードI のクラウドベース AWS マネージドインフラストラクチャで動作しますが、hybrid とマークされたアドオンはEKS クラウドコントロールプレーンに接続されたオンプレミスノードで実行できます。

詳細については「the section called “Amazon EKS 自動モードの考慮事項”」を参照してください。

クラスターから Amazon EKS アドオンを削除する

eksctl、AWS マネジメントコンソール、AWS CLI を使用して Amazon EKS アドオンをクラスターから削除できます。

クラスターから Amazon EKS アドオンを削除する場合:

- アドオンが提供する機能にはダウンタイムはありません。
- サービスアカウントの IAM ロール (IRSA) を使用していて、アドオンに IAM ロールが関連付けられている場合、IAM ロールは削除されません。
- Pod Identities を使用している場合、アドオンが所有する Pod Identity の関連付けはすべて削除されます。AWS CLI に --preserve オプションを指定すると、関連付けは保持されます。
- Amazon EKS はアドオンの設定の管理を停止します。
- 新しいバージョンが利用可能になると、コンソールが通知を停止します。
- AWS ツールや API を使用してアドオンを更新することはできません。
- 自己管理できるようにアドオンソフトウェアをクラスターに保持するか、クラスターからアドオンソフトウェアを削除するかを選択できます。クラスターからアドオンソフトウェアを削除するのは、アドオンが提供する機能に依存するリソースがクラスターにない場合のみにしてください。

前提条件

アドオンを削除する前に、以下を完了してください。

- 既存の Amazon EKS クラスター。デプロイするには「はじめに」を参照してください。
- アドオンに IAM ロールが必要かどうかを確認します。詳細については、「the section called “IAM 情報を取得する”」を参照してください。
- デバイスまたは AWS CloudShell にインストールされている `eksctl` コマンドラインツールのバージョン `0.215.0` 以降。`eksctl` をインストールまたはアップグレードするには、`eksctl` ドキュメントの「インストール」を参照してください。

手順

Amazon EKS アドオンを削除する場合、次の 2 つのオプションがあります。

- クラスター上のアドオンソフトウェアを保持する – このオプションではすべての設定の Amazon EKS 管理が削除されます。また、Amazon EKS による更新の通知機能や、更新の開始後の Amazon EKS アドオンの自動更新機能も削除されます。ただし、クラスター上のアドオンソフトウェアは保持されます。このオプションを選択すると、アドオンは Amazon EKS アドオンではなく、セルフマネージド型インストールになります。このオプションを使用すると、アドオンのダウンタイムは発生しません。
- クラスターからアドオンソフトウェアを完全に削除する – クラスターから Amazon EKS アドオンを削除するのはアドオンに依存するリソースがクラスター上にない場合のみとすることをお勧めします。

Amazon EKS アドオンは `eksctl`、AWS マネジメントコンソール、AWS CLI を使用して削除できます。

アドオンの削除 (eksctl)

1. クラスターに現在インストールされているアドオンを確認します。*`my-cluster`* の部分は自分のクラスター名に置き換えます。

```
eksctl get addon --cluster my-cluster
```

   出力例は次のとおりです。

```
NAME        VERSION              STATUS  ISSUES  IAMROLE  UPDATE AVAILABLE
coredns     v1.8.7-eksbuild.2    ACTIVE  0
kube-proxy  v1.23.7-eksbuild.1   ACTIVE  0
vpc-cni     v1.10.4-eksbuild.1   ACTIVE  0
[...]
```

クラスターにあるアドオンおよびバージョンによって、出力が異なる場合があります。

2. アドオンを削除します。*my-cluster* をクラスターの名前に置き換えて、*name-of-add-on* を削除する前のステップの出力で返されたアドオン名に置き換えます。*--preserve* オプションを削除する場合、Amazon EKS がアドオンを管理しなくなるだけではなく、アドオンソフトウェアがクラスターから削除されます。

```
eksctl delete addon --cluster my-cluster --name name-of-addon --preserve
```

アドオンの削除の詳細についてはeksctl ドキュメントの「Deleting addons」を参照してください。

アドオンの削除 (AWS コンソール)

1. Amazon EKS コンソールを開きます。
2. 左のナビゲーションペインで [クラスター] を選択します。
3. Amazon EKS アドオンを削除するクラスターの名前を選択してください。
4. [アドオン] タブを選択してください。
5. 削除するアドオンを選択してください。
6. [削除] を選択してください。
7. [削除: *#######*] 確認ダイアログボックスで、次を実行します:
   a. Amazon EKS でのアドオンの設定の管理を停止する場合は[クラスターで保持する] を選択してください。クラスターでアドオンソフトウェアを保持する場合はこの操作を実行してください。これはアドオンのすべての設定を自分で管理できるようにするためです。
   b. アドオン名を入力してください。
   c. [削除] を選択してください。

アドオンの削除 (AWS CLI)

1. デバイスまたは AWS CloudShell にインストールされている eksctl コマンドラインツールのバージョン 0.215.0 以降が必要です。eksctl をインストールまたはアップグレードするにはeksctl ドキュメントの「インストール」を参照してください。
2. インストールされているアドオンのリストを参照してください。*my-cluster* の部分は自分のクラスター名に置き換えます。

```
aws eks list-addons --cluster-name my-cluster
```

出力例は次のとおりです。

```
{
    "addons": [
        "coredns",
        "kube-proxy",
        "vpc-cni",
        "name-of-addon"
    ]
}
```

3. インストールされているアドオンを削除します。*my-cluster* をクラスターの名前に置き換えて、*name-of-add-on* を削除するアドオン名に置き換えます。*--preserve* を削除することにより、アドオンソフトウェアがクラスターから削除されます。

```
aws eks delete-addon --cluster-name my-cluster --addon-name name-of-addon --preserve
```

簡略化した出力例を次に示します。

```
{
    "addon": {
        "addonName": "name-of-add-on",
        "clusterName": "my-cluster",
        "status": "DELETING",
    }
}
```

4. 削除のステータスを確認します。*my-cluster* をクラスターの名前に置き換えて、*name-of-addon* を削除するアドオン名に置き換えます。

```
aws eks describe-addon --cluster-name my-cluster --addon-name name-of-addon
```

アドオンを削除した後の出力例は次のとおりです。

```
An error occurred (ResourceNotFoundException) when calling the DescribeAddon
 operation: No addon: name-of-addon found in cluster: my-cluster
```

Amazon EKS アドオンの IAM ロール

特定の Amazon EKS アドオンでは、AWS API を呼び出す IAM ロールおよび許可が必要です。例えば、Amazon VPC CNI アドオンは特定の AWS API を呼び出して、アカウント内のネットワークリソースを設定します。これらのアドオンには、 IAM を使用してアクセス許可を付与する必要があります。具体的には、アドオンを実行しているポッドのサービスアカウントを、特定の IAM ポリシーを持つ IAM ロールに関連付ける必要があります。

クラスターワークロードに AWS アクセス許可を付与する際の推奨方法は、Amazon EKS 機能 Pod Identity を使用することです。Pod Identity の関連付けを使用して、アドオンのサービスアカウントを IAM ロールにマッピングできます。ポッドが関連付けられたサービスアカウントを使用する場合、Amazon EKS はポッドのコンテナに環境変数を設定します。環境変数は、AWS CLI を含む AWS SDK が EKS Pod Identity の認証情報を使用するように設定します。詳細については、the section called “Pod Identity”を参照してください。

Amazon EKS アドオンは、アドオンに対応する Pod Identity の関連付けのライフサイクルを管理するのに役立ちます。例えば、Amazon EKS アドオンと必要な Pod Identity の関連付けを 1 回の API コールで作成または更新できます。Amazon EKS には、推奨される IAM ポリシーを取得するための API も用意されています。

1. Amazon EKS Pod Identity エージェントがクラスターに設定されていることを確認します。
2. インストールするアドオンに、`describe-addon-versions` AWS CLI オペレーションを使用して IAM アクセス許可が必要かどうかを確認します。`requiresIamPermissions` フラグが `true` の場合、`describe-addon-configurations` オペレーションを使用してアドオンに必要なアクセス許可を決定する必要があります。レスポンスには、推奨されるマネージド IAM ポリシーのリストが含まれます。
3. `describe-addon-configuration` CLI オペレーションを使用して、Kubernetes サービスアカウントの名前と IAM ポリシーを取得します。提案されたポリシーの範囲をセキュリティ要件に照らして評価します。

4. 提案されたアクセス許可ポリシーと Pod Identity に必要な信頼ポリシーを使用して IAM ロールを作成します。詳細については、「the section called “Pod Identity の関連付けを作成する (AWS コンソール)”」を参照してください。
5. CLI を使用して Amazon EKS アドオンを作成または更新します。少なくとも 1 つの Pod Identity の関連付けを指定します。Pod Identity の関連付けは、Kubernetes サービスアカウントの名前、および IAM ロールの ARN です。
   - アドオン API を使用して作成された Pod Identity の関連付けは、それぞれのアドオンによって所有されます。アドオンを削除すると、Pod Identity の関連付けも削除されます。AWS CLI または API を使用してアドオンを削除するときに、`preserve` オプションを使用してこのカスケード削除を防ぐことができます。必要に応じて、Pod Identity の関連付けを直接更新または削除することもできます。アドオンは、既存の Pod Identity の関連付けの所有権を引き受けることはできません。既存の関連付けを削除し、アドオンの作成または更新オペレーションを使用して再作成する必要があります。
   - Amazon EKS では、Pod Identity の関連付けを使用してアドオンの IAM アクセス許可を管理することをお勧めします。前の方法であるサービスアカウント (IRSA) の IAM ロールは引き続きサポートされています。アドオンの IRSA `serviceAccountRoleArn` と Pod Identity の関連付けの両方を指定できます。EKS Pod Identity エージェントがクラスターにインストールされている場合、`serviceAccountRoleArn` は無視され、EKS は指定された Pod Identity の関連付けを使用します。Pod Identity が有効になっていない場合、`serviceAccountRoleArn` が使用されます。
   - 既存のアドオンの Pod Identity の関連付けを更新すると、Amazon EKS はアドオンポッドのローリング再起動を開始します。

Amazon EKS アドオンに関する IAM 情報を取得する

アドオンを作成する前に、AWS CLI を使用して次を判断します:

- アドオンに IAM 許可が必要かどうか
- 使用する推奨 IAM ポリシー

手順

1. インストールするアドオンの名前とクラスターの Kubernetes バージョンを決定します。アドオンの詳細については「the section called “Amazon EKS アドオン”」を参照してください。
2. AWS CLI を使用してアドオンに IAM アクセス許可が必要かどうかを確認します。

```
aws eks describe-addon-versions \
--addon-name <addon-name> \
--kubernetes-version <kubernetes-version>
```

例えば、次のようになります。

```
aws eks describe-addon-versions \
--addon-name aws-ebs-csi-driver \
--kubernetes-version 1.30
```

以下の出力例を確認します。requiresIamPermissions は true であり、デフォルトのアドオンバージョンであることに注意してください。推奨される IAM ポリシーを取得するときはアドオンバージョンを指定する必要があります。

```
{
    "addons": [
        {
            "addonName": "aws-ebs-csi-driver",
            "type": "storage",
            "addonVersions": [
                {
                    "addonVersion": "v1.31.0-eksbuild.1",
                    "architecture": [
                        "amd64",
                        "arm64"
                    ],
                    "compatibilities": [
                        {
                            "clusterVersion": "1.30",
                            "platformVersions": [
                                "*"
                            ],
                            "defaultVersion": true
                        }
                    ],
                    "requiresConfiguration": false,
                    "requiresIamPermissions": true
                },
[...]
```

3. アドオンに IAM アクセス許可が必要な場合はAWS CLI を使用して推奨される IAM ポリシーを取得します。

```
aws eks describe-addon-configuration \
--query podIdentityConfiguration \
--addon-name <addon-name> \
--addon-version <addon-version>
```

例えば、次のようになります。

```
aws eks describe-addon-configuration \
--query podIdentityConfiguration \
--addon-name aws-ebs-csi-driver \
--addon-version v1.31.0-eksbuild.1
```

以下の出力を確認します。`recommendedManagedPolicies` を書き留めます。

```
[
    {
        "serviceAccount": "ebs-csi-controller-sa",
        "recommendedManagedPolicies": [
            "arn:aws:iam::aws:policy/service-role/AmazonEBSCSIDriverPolicyV2"
        ]
    }
]
```

4. IAM 役割を作成し、管理ポリシーをアタッチします。または管理ポリシーを確認し、必要に応じてアクセス許可の範囲を絞り込みます。詳細については「the section called “Pod Identity の関連付けを作成する (AWS コンソール)”」を参照してください。

Pod Identity サポートリファレンス

次の表は特定の Amazon EKS アドオンが EKS Pod Identity をサポートしているかどうかを示しています。

アドオン名	Pod Identity のサポート	必要な最小バージョン
Amazon EBS CSI ドライバー	はい	v1.26.0-eksbuild.1

アドオン名	Pod Identity のサポート	必要な最小バージョン
Amazon VPC CNI	はい	v1.15.5-eksbuild.1
Amazon EFS CSI ドライバー	はい	v2.0.5-eksbuild.1
AWS オープンテレメトリー用ディストロ	はい	v0.94.1-eksbuild.1
Mountpoint for Amazon S3 CSI ドライバー	いいえ	該当なし
Amazon CloudWatch 観測可能エージェント	はい	v3.1.0-eksbuild.1

この表は 2024 年 10 月 28 日に最終更新されました。

Pod Identity を使用して Amazon EKS アドオンに IAM 役割を割り当てる

特定の Amazon EKS アドオンではIAM 役割および許可が必要です。Pod Identity の関連付けを使用するように Amazon EKS アドオンを更新する前に、使用するロールとポリシーを検証します。詳細については、「the section called “IAM情報を取得する”」を参照してください。

1. 決定事項:
   - `cluster-name` – アドオンをインストールするクラスターの名前。
   - `addon-name` – インストールするアドオンの名前。
   - `service-account-name` – アドオンで使用される Kubernetes サービスアカウントの名前。
   - `iam-role-arn` – アドオンに十分なアクセス許可を持つ IAM 役割の ARN。役割にはEKS Pod Identity に必要な信頼ポリシーが必要です。詳細については「the section called “Pod Identity の関連付けを作成する (AWS コンソール)”」を参照してください。
2. AWS CLI を使用してアドオンを更新します。同じ `--pod-identity-associations` 構文を使用して、アドオンの作成時に Pod Identity の関連付けを指定することもできます。アドオンの更新中に Pod Identity の関連付けを指定すると、以前のすべての Pod Identity の関連付けが上書きされることに注意してください。

```
aws eks update-addon --cluster-name <cluster-name> \
```

```
--addon-name <addon-name> \
--pod-identity-associations '[{"serviceAccount": "<service-account-name>", "roleArn":
 "<role-arn>"}]'
```

例えば、次のようになります。

```
aws eks update-addon --cluster-name mycluster \
--addon-name aws-ebs-csi-driver \
--pod-identity-associations '[{"serviceAccount": "ebs-csi-controller-sa", "roleArn":
 "arn:aws:iam::123456789012:role/StorageDriver"}]'
```

3. Pod Identity の関連付けが作成されたことを確認します:

```
aws eks list-pod-identity-associations --cluster-name <cluster-name>
```

成功すると、次のような出力が表示されます。EKS アドオンの OwnerARN を書き留めます。

```
{
    "associations": [
        {
            "clusterName": "mycluster",
            "namespace": "kube-system",
            "serviceAccount": "ebs-csi-controller-sa",
            "associationArn": "arn:aws:eks:us-
west-2:123456789012:podidentityassociation/mycluster/a-4wvljrezsukshq1bv",
            "associationId": "a-4wvljrezsukshq1bv",
            "ownerArn": "arn:aws:eks:us-west-2:123456789012:addon/mycluster/aws-ebs-
csi-driver/9cc7ce8c-2e15-b0a7-f311-426691cd8546"
        }
    ]
}
```

Amazon EKS アドオンから Pod Identity の関連付けを削除する

Amazon EKS アドオンから Pod Identity の関連付けを削除します。

1. 決定事項:
   - `cluster-name` – アドオンをインストールする EKS クラスターの名前。
   - `addon-name` – インストールする Amazon EKS アドオンの名前。

2. アドオンを更新して、Pod Identity の関連付けの空の配列を指定します。

```
aws eks update-addon --cluster-name <cluster-name> \
--addon-name <addon-name> \
--pod-identity-associations "[]"
```

EKS アドオンの Pod Identity のトラブルシューティング

AWS API、SDK、または CLI オペレーションの試行中にアドオンでエラーが発生した場合は以下を確認します：

- Pod Identity エージェントがクラスター にインストールされている。
  - Pod Identity Agent のインストール方法については「the section called “エージェントのセットアップ”」を参照してください。
- アドオンに有効な Pod Identity の関連付けがある。
  - AWS CLI を使用して、アドオンで使用されるサービスアカウント名の関連付けを取得します。

```
aws eks list-pod-identity-associations --cluster-name <cluster-name>
```

- IAM ロールに、Pod Identity に必要な信頼ポリシーがある。
  - AWS CLI を使用して、アドオンの信頼ポリシーを取得します。

```
aws iam get-role --role-name <role-name> --query Role.AssumeRolePolicyDocument
```

- IAM ロールに、アドオンに必要な許可がある。
  - AWS CloudTrail を使用して `AccessDenied` または `UnauthorizedOperation` イベントを確認します。
- Pod Identity の関連付けのサービスアカウント名はアドオンで使用されるサービスアカウント名と一致している。
  - 利用可能なアドオンについては「the section called “ AWS アドオン”」を参照してください。
- `pod-identity-webhook` という名前の MutatingWebhookConfiguration の設定を確認する。
  - ウェブフックの `admissionReviewVersions` は `v1beta1` である必要があり、`v1` では機能しません。

Amazon EKS アドオンのためにカスタマイズできるフィールドを決定する

Amazon EKS アドオンは標準的なベストプラクティスによる設定を使用して、クラスターにインストールされます。Amazon EKS アドオンをクラスターに追加する方法については「the section called "Amazon EKS アドオン"」を参照してください。

Amazon EKS アドオンの設定をカスタマイズすると、高度な機能を有効にすることが可能です。Amazon EKS は、Kubernetes サーバー側の適用機能を使用して、Amazon EKS で管理されていない (ユーザーによる) 設定の内容を上書きすることなく、Amazon EKS によるアドオンの管理を可能にします。詳細については、Kubernetes ドキュメントの「Server Side Apply」を参照してください。これを実現するために、Amazon EKS は、インストールするアドオンごとに最低限のフィールドセットを管理します。Amazon EKS や別の Kubernetes コントロールプレーンプロセス (`kube-controller-manager` など) によって管理されていないすべてのフィールドは、ユーザーが問題なく管理できます。

> **Important**
>
> Amazon EKS で管理しているフィールドを変更すると、Amazon EKS がアドオンを管理できなくなるため、アドオンの更新時にユーザーによる変更が上書きされる可能性があります。

フィールド管理構文

Kubernetes オブジェクトの詳細を表示すると、マネージド型フィールドとアンマネージド型フィールドの両方が出力に返されます。管理対象フィールドは、次のいずれかになります。

- 完全マネージド型 - フィールドのすべてのキーは Amazon EKS によって管理されます。このフィールドの値を変更すると、競合の原因となります。
- 部分的マネージド型 - フィールドの一部のキーは Amazon EKS によって管理されます。Amazon EKS によって明示的に管理されているキーに対する変更のみが競合の原因となります。

どちらのタイプのフィールドも `manager: eks` でタグづけされています。

各キーはフィールド自体を表す `.` (これは常に空のセットがマッピングされます) であるか、またはサブフィールドまたは項目を表す文字列です。フィールドの管理状況の出力は以下のタイプの宣言で構成されます。

- `f:`*`name`* (*`name`* はリスト内のフィールドの名前)。

- k:*keys* (*keys* はリスト項目のフィールドのマップ)。
- v:*value* (*value* はリスト項目を正確な JSON 形式で記述した値)。
- i:*index* (*index* はリスト内の項目の位置)。

CoreDNS アドオン出力内で次に続く部分には、以前の宣言を表示します。

- フルマネージド型フィールド — マネージド型フィールドに f: (フィールド) が指定されていますが、k:(キー) はありません。この場合は、フィールド全体が管理されています。このフィールドの値を変更すると、競合の原因となります。

  次の出力ではcoredns という名前のコンテナが、eks によって管理されていることがわかります。args、image、および imagePullPolicy サブフィールドも eks によって管理されています。このフィールドの値を変更すると、競合の原因となります。

  ```
  [...]
  f:containers:
    k:{"name":"coredns"}:
    .: {}
    f:args: {}
    f:image: {}
    f:imagePullPolicy: {}
  [...]
  manager: eks
  [...]
  ```

- 部分的マネージド型キー - マネージド型キーに値が指定されている場合、宣言されたキーはそのフィールドで管理されます。指定されたキーを変更すると、競合の原因となります。

  次の出力ではname キーを使用する eks が、config-volume および tmp ボリュームセットを管理していることがわかります。

  ```
  [...]
  f:volumes:
    k:{"name":"config-volume"}:
      .: {}
      f:configMap:
        f:items: {}
        f:name: {}
      f:name: {}
    k:{"name":"tmp"}:
  ```

```
    .: {}
    f:name: {}
[...]
manager: eks
[...]
```

- 部分的マネージド型フィールドへのキーの追加 - 特定のキーバリューのみが管理されている場合であれば、競合を引き起こすことなく、引数などの追加のキーをフィールドに追加できます。キーを追加する場合はまず、そのフィールドがマネージド型でないことを確認してください。マネージド型の値を追加または変更すると、競合の原因となります。

  次の出力ではname キーおよび name フィールドの両方が、マネージド型であることがわかります。コンテナ名を追加または変更すると、このマネージド型のキーとの競合が発生します。

```
[...]
f:containers:
  k:{"name":"coredns"}:
[...]
    f:name: {}
[...]
manager: eks
[...]
```

手順

kubectl を使用すると、任意の Amazon EKS アドオンについて、どのフィールドが Amazon EKS によって管理されているかを知ることができます。

Amazon EKS や別の Kubernetes コントロールプレーンプロセス (kube-controller-manager など) によって管理されていないすべてのフィールドは、ユーザーが問題なく管理できます。

1. 確認するアドオンを決定します。クラスターにデプロイされた deployments と DaemonSet をすべて表示するには、「the section called “クラスターリソースにアクセスする”」を参照してください。
2. アドオンのマネージド型フィールドを表示するには、次のコマンドを実行します。

```
kubectl get type/add-on-name -n add-on-namespace -o yaml
```

たとえば、次のコマンドを使用すると、CoreDNS アドオンのマネージド型フィールドを表示できます。

```
kubectl get deployment/coredns -n kube-system -o yaml
```

フィールドの管理状態は返される出力の中の、次のセクションに記載されています。

```
[...]
managedFields:
  - apiVersion: apps/v1
    fieldsType: FieldsV1
    fieldsV1:
[...]
```

> **Note**
>
> 出力に、managedFields が表示されない場合、--show-managed-fields をコマンドに追加し、もう一度実行してください。使用している kubectl のバージョンによって、管理フィールドがデフォルトで返されるかどうかが決まります。

次のステップ

アドオンのために、AWS が所有していないフィールドをカスタマイズします。

デプロイ中にコンテナイメージの署名を検証する

AWS Signer を使用し、デプロイ時に署名済みのコンテナイメージを検証したい場合は、次のいずれかのソリューションが使用可能です。

- Gatekeeper と Ratify — Gatekeeper をアドミッションコントローラーとして使用し、AWS Signer プラグインで構成された Ratify を署名を検証するためのウェブフックとして使用します。
- Kyverno — 署名を検証するための AWS Signer プラグインで構成された Kubernetes ポリシーエンジン。

> **Note**
>
> コンテナイメージの署名を検証する前に、選択したアドミッションコントローラーの要求に応じて、Notation トラストストアとトラストポリシーを構成します。

EKS 機能

> **Tip**
>
> 使用開始: ACK 機能の作成 | Argo CD 機能の作成 | kro 機能の作成

Amazon EKS 機能は、開発者のベロシティを加速させ、Kubernetes での構築とスケールの複雑さを軽減するのに役立つ、完全マネージド型のクラスター機能のレイヤーセットです。EKS 機能は、宣言型の継続的デプロイ、AWS リソース管理、Kubernetes リソースの作成とオーケストレーションに対応した Kubernetes ネイティブの機能であり、いずれも AWS で完全に管理されます。EKS 機能を使用すると、基盤となるプラットフォームサービスのオペレーションにかかる負担を AWS にオフロードして、ワークロードの構築とスケールに集中できます。これらの機能はクラスター内ではなく EKS 内で実行されるため、重要なプラットフォームコンポーネントをワーカーノードにインストール、メンテナンス、スケールする必要がありません。

使用を開始するには、新規または既存の EKS クラスターに 1 つ以上の EKS 機能を作成します。そのためには、AWS CLI、AWS マネジメントコンソール、EKS API、eksctl、または任意の Infrastructure as Code ツールを使用します。EKS 機能は、互いに連携するように設計されていると共に、ユースケースと要件に基づいて選択可能な独立したクラウドリソースです。

EKS でサポートされている Kubernetes バージョンは、いずれも EKS 機能でサポートされています。

> **Note**
>
> EKS 機能は、Amazon EKS が利用可能なすべての AWS 商用リージョンで利用できます。サポートされているリージョンの一覧については、「AWS 全般的なリファレンス」の「Amazon EKS エンドポイントとクォータ」を参照してください。

利用可能な機能

AWS Controllers for Kubernetes (ACK)

ACK を使用すると、Kubernetes API を使用して AWS リソースを管理できるため、Kubernetes カスタムリソースを使用して S3 バケット、RDS データベース、IAM ロールなどの AWS リソースを作成および管理できます。ACK は、目的の状態と AWS での実際の状態とを継続的に調整して、時間

の経過に伴うドリフトを修正します。これにより、システムの正常性を維持し、リソースを指定どおりに設定できます。Kubernetes ワークロードと共に、同じツールとワークフローを使用して AWSリソースも管理でき、S3、RDS、DynamoDB、Lambda など 50 を超える AWS サービスをサポートしています。ACK は、クロスアカウントおよびクロスリージョンのリソース管理をサポートし、マルチアカウントでマルチクラスターの複雑なシステム管理アーキテクチャを実現します。また、読み取り専用リソースと読み取り専用採用をサポートしているため、他の Infrastructure as Code ツールから Kubernetes ベースのシステムに容易に移行できます。

ACK の詳細はこちら →

Argo CD

Argo CD は、アプリケーションを GitOps ベースで継続的にデプロイする仕組みを実装しており、Git リポジトリをワークロードとシステム状態の信頼できるソースとして使用します。また、アプリケーションリソースを Git リポジトリからクラスターに自動的に同期し、ドリフトを検出および修正して、デプロイしたアプリケーションが目的の状態と一致するようにします。1 つの Argo CD インスタンスから複数のクラスターにわたってアプリケーションをデプロイおよび管理でき、変更がコミットされるたびに Git リポジトリから自動的にデプロイされます。Argo CD と ACK を一緒に使用すると、基盤となる GitOps システムを実現して、ワークロードの依存関係管理をシンプルにし、クラスターやインフラストラクチャを規模に応じて管理するなどシステム全体の設計をサポートできます。Argo CD は、AWS アイデンティティセンターとの統合により認証と認可を実施し、Argo UI のホストによりアプリケーションのヘルスとデプロイのステータスを可視化します。

Argo CD の詳細はこちら →

kro (Kube Resource Orchestrator)

kro を使用すると、複数のリソースをより高レベルの抽象化に組み合わせるカスタム Kubernetes API を作成できます。これにより、プラットフォームチームは、一般的なリソースの組み合わせに対して再利用可能なパターン (クラウドの構成要素) を定義できます。kro では、Kubernetes リソースと AWS リソースの両方を統合抽象化に構成し、シンプルな構文を使用して、動的設定と条件付きロジックを有効にできます。kro を使用すると、プラットフォームチームが適切なガードレールに沿ってセルフサービス機能を提供できるため、開発者は組織の標準とベストプラクティスを維持しながら、シンプルな目的別 API を使用して複雑なインフラストラクチャをプロビジョニングできます。kro リソースは Kubernetes リソースにすぎず、Kubernetes マニフェストに指定して Git に保存できます。あるいは、Amazon ECR などの OCI 互換のレジストリにプッシュして、組織内に広範に分散させることもできます。

kro の詳細はこちら →

EKS 機能のメリット

EKS 機能は、AWS で完全に管理されるため、基礎となるクラスターサービスのインストール、メンテナンス、スケールが不要です。AWS がセキュリティパッチ適用、更新、オペレーション管理を担うため、チームはクラスターオペレーションではなく AWS での構築に集中できます。クラスターリソースを消費する従来の Kubernetes アドオンとは異なり、機能はワーカーノードではなく EKS で動作します。これにより、クラスター内コントローラーやその他のプラットフォームコンポーネントの管理に伴うオペレーションの負担を最小限に抑えながら、ワークロードのクラスターのキャパシティとリソースを解放できます。

EKS 機能により、`kubectl` などネイティブの Kubernetes API とツールを使用して、デプロイ、AWS リソース、カスタム Kubernetes リソース、およびコンポジションを管理できます。すべての機能がクラスターのコンテキストで動作し、アプリケーションリソースとクラウドインフラストラクチャリソースの両方で設定のドリフトを自動的に検出して修正します。1 つのコントロールポイントから複数のクラスター、AWS アカウント、およびリージョンにリソースをデプロイして管理できるため、複雑な分散環境でのオペレーションがシンプルになります。

EKS 機能は、GitOps ワークフロー用に設計されており、インフラストラクチャとアプリケーションのバージョンを宣言型で管理できます。変更が Git からシステムを通じて流れていくため、監査証跡、ロールバック機能、およびコラボレーションワークフローを既存の開発プラクティスと統合できます。この Kubernetes ネイティブアプローチにより、複数のツールを使用したり、クラスターの外部にある Infrastructure as Code システムを管理したりする必要がなく、信頼できる単一のソースを参照すればよくなります。バージョン管理された Kubernetes 宣言型設定に定義された目的の状態は、環境全体に継続的に適用されます。

料金

EKS 機能に前払いのコミットメントや料金はありません。機能リソースごとに Amazon EKS クラスターでアクティブになっている時間単位で課金されます。EKS 機能で管理される特定の Kubernetes リソースも、時間単位で課金されます。

現在の料金については、「Amazon EKS の料金ページ」を参照してください。

> **Tip**
>
> AWS Cost Explorer と、コストと使用状況レポートとを使用すると、機能のコストを他の EKS 料金とは別に追跡できます。機能にコスト配分の目的でクラスター名、機能タイプ、その他の詳細のタグを付与できます。

EKS 機能の仕組み

各機能は、EKS クラスターに作成する AWS リソースです。作成した機能は、EKS で実行され、AWS で完全に管理されます。

> **Note**
>
> 1 つのクラスターのタイプ (Argo CD、ACK、kro) ごとに機能リソースを 1 つ作成できます。同じクラスターで同じタイプに機能リソースを複数作成することはできません。

クラスターの機能とのインタラクションには、標準の Kubernetes API とツールを使用します。

- `kubectl` を使用して Kubernetes カスタムリソースを適用します。
- GitOps ワークフローの信頼できるソースとして Git リポジトリを使用します。

追加のツールがサポートされている機能もあります。例えば、次のようになります。

- Argo CD CLI を使用して、Argo CD 機能のリポジトリおよびクラスターを設定および管理します。
- Argo CD UI を使用して、Argo CD 機能で管理されるアプリケーションを可視化および管理します。

各機能は、互いに連携するように設計されていますが、いずれも独立しており、完全にオプトインされています。必要に応じて 1 つ、2 つ、または 3 つすべての機能を有効にし、要件の進化に伴って設定を更新できます。

EKS コンピューティングのいずれのタイプも、EKS 機能で使用できるようにサポートされています。詳細については、「コンピューティングを管理する」を参照してください。

IAM ロールでのセキュリティ設定と詳細については、「the section called “EKS 機能に関する考慮事項”」を参照してください。マルチクラスターアーキテクチャパターンについては、「the section called “考慮事項”」を参照してください。

一般的なユースケース

アプリケーションとインフラストラクチャの GitOps

Argo CD を使用してアプリケーションとオペレーションコンポーネントをデプロイし、ACK を使用してクラスター設定の管理とインフラストラクチャのプロビジョニングを行います。どちらも、Git リポジトリから実施します。アプリケーション、データベース、ストレージ、ネットワークといったスタック全体がコードとして定義され、自動的にデプロイされます。

例: 開発チームが Git に変更をプッシュします。Argo CD が更新後のアプリケーションをデプロイし、ACK が新規 RDS データベースを適切な設定でプロビジョニングします。すべての変更が監査可能で、可逆性があり、環境間で一貫性が保たれます。

セルフサービスによるプラットフォームエンジニアリング

kro でカスタム API を作成して、ACK リソースと Kubernetes リソースを構成できます。プラットフォームチームは、ガードレールに沿って承認済みのパターンを定義します。アプリケーションチームは、シンプルで高レベルの API を使用して、完全なスタックをプロビジョニングします。

例: プラットフォームチームは、「WebApplication」という API を作成して、デプロイ、サービス、イングレス、S3 バケットをプロビジョニングします。開発者は、この API を使用するにあたって、その基盤となる複雑さや AWS アクセス許可について知る必要がありません。

マルチクラスターアプリケーション管理

Argo CD を使用して、リージョンやアカウントが異なる複数の EKS クラスターにアプリケーションをデプロイします。一貫したポリシーとワークフローに従って、単一の Argo CD インスタンスからすべてのデプロイを管理します。

例: 複数のリージョンにまたがる開発、ステージング、本番稼働用のクラスターに同じアプリケーションをデプロイします。Argo CD により、各環境はその対応する Git ブランチと常に同期するようになります。

マルチクラスター管理

ACK を使用して EKS クラスターを定義およびプロビジョニングし、kro を使用して組織の標準に従ってクラスター設定をカスタマイズし、Argo CD を使用してクラスターのライフサイクルと設定を管理します。これにより、作成から継続的な運用までエンドツーエンドでクラスターを管理できます。

例: ACK と kro により、クラスターインフラストラクチャをプロビジョニングおよび管理するように EKS クラスターを定義して、ネットワーク、セキュリティポリシー、アドオンなどの設定に関して組織全体の標準を定義します。Argo CD を使用すると、一貫した標準と自動ライフサイクル管理を利用してフリート全体でクラスター、設定、Kubernetes バージョン更新を作成し、継続的に管理します。

移行とモダナイズ

ネイティブのクラウドリソースプロビジョニングと GitOps ワークフローに従って、EKS への移行をシンプルにします。ACK を使用して既存の AWS リソースを再作成することなく採用し、Argo CD を使用して Git からワークロードのデプロイを運用できるようにします。

例: EC2 から EKS に移行するチームは、まず ACK を使用して既存の RDS データベースと S3 バケットを採用し、次に Argo CD を使用して Git からコンテナ化されたアプリケーションをデプロイします。移行パスは明確で、オペレーションは最初から標準化されます。

アカウントとリージョンのブートストラッピング

Argo CD と ACK を一緒に使用して、アカウントとリージョン全体にわたるインフラストラクチャのロールアウトを自動化します。Git に Infrastructure as Code を定義し、機能がデプロイと管理を担えるようにします。

例: プラットフォームチームは、VPC、IAM ロール、RDS インスタンス、モニタリングスタックといった標準アカウント設定を定義している Git リポジトリを維持します。Argo CD がこれらの設定を新しいアカウントとリージョンに自動的にデプロイするため、一貫性が確保され、手動セットアップ時間が数日から数分に短縮されます。

機能リソースの使用

Amazon EKS クラスターの機能リソースを一覧表示、記述、更新、削除します。

EKS 機能リソース

EKS 機能は、Amazon EKS クラスターでマネージド機能を有効にする AWS リソースです。機能は EKS で実行されるため、コントローラーやその他の運用コンポーネントをワーカーノードにインストールして維持する必要がありません。機能は、特定の EKS クラスター用に作成され、ライフサイクル全体でそのクラスターと関連付けられます。

各機能リソースには、次のものがあります。

- クラスター内で一意の名前
- 機能タイプ (ACK、ARGOCD、または KRO)
- 名前とタイプの両方を指定する Amazon リソースネーム (ARN)
- 機能 IAM ロール
- 現在の状態を示すステータス

- 汎用の設定と機能タイプ固有の設定

機能ステータスについて

機能リソースには、現在の状態を示すステータスがあります。機能のステータスとヘルスは、EKS コンソールまたは AWS CLI を使用して表示できます。

コンソール:

1. https://console.aws.amazon.com/eks/home#/clusters で Amazon EKS コンソールを開きます。
2. クラスター名を選択します。
3. [機能] タブを選択すると、すべての機能のステータスが表示されます。
4. 詳細なヘルス情報については、[オブザーバビリティ] タブを選択し、[クラスターをモニタリング] を選択してから、[機能] タブを選択します。

AWSCLI:

```
aws eks describe-capability \
  --region region-code \
  --cluster-name my-cluster \
  --capability-name my-capability-name
```

機能ステータス

CREATING: 機能のセットアップ中です。コンソールから移動できます。この機能はバックグラウンドで引き続き作成されます。

ACTIVE: 機能の実行中ですぐに使用できるようになります。リソースが想定どおりに動作しない場合は、リソースのステータスと IAM アクセス許可を確認してください。ガイダンスについては、「the section called “機能をトラブルシューティングする”」を参照してください。

UPDATING: 設定変更の適用中です。ステータスが `ACTIVE` に戻るまで待ちます。

DELETING: クラスターから機能を削除中です。

CREATE_FAILED: セットアップでエラーが発生しました。一般的な原因には、以下が含まれます。

- IAM ロールの信頼ポリシーが正しくないか、見つかりません。
- IAM ロールが存在しないか、アクセスできません。

- クラスターアクセスに問題があります。
- 設定パラメータが無効です。

特定のエラーの詳細については、機能ヘルスセクションを確認してください。

UPDATE_FAILED: 設定の更新に失敗しました。機能ヘルスセクションで詳細を確認し、IAM アクセス許可を検証します。

> **Tip**
>
> トラブルシューティングの詳細なガイダンスについては、以下を参照してください。
>
> - the section called “機能をトラブルシューティングする” - 一般的な機能のトラブルシューティング
> - the section called “トラブルシューティング” - ACK 固有の問題
> - the section called “トラブルシューティング” - Argo CD 固有の問題
> - the section called “トラブルシューティング” - kro 固有の問題

機能を作成する

クラスターに機能を作成するには、以下のトピックを参照してください。

- the section called “ACK 機能を作成する” – Kubernetes API を使用して AWS リソースを管理するための ACK 機能を作成する
- the section called “Argo CD 機能を作成する” – GitOps による継続的デリバリーのための Argo CD 機能を作成する
- the section called “kro 機能の作成” – リソースの構成とオーケストレーションのための kro 機能を作成する

機能を一覧表示する

クラスター上のすべての機能リソースを一覧表示できます。

コンソール

1. https://console.aws.amazon.com/eks/home#/clusters で Amazon EKS コンソールを開きます。

2. クラスター名を選択して、クラスターの詳細ページを開きます。
3. [機能] タブを選択します。
4. [マネージド機能] で機能リソースを表示します。

AWS CLI

`list-capabilities` コマンドを使用して、クラスター上のすべての機能を表示します。*region-code* はクラスターがある AWS リージョンに、*my-cluster* はクラスターの名前に置き換えます。

```
aws eks list-capabilities \
  --region region-code \
  --cluster-name my-cluster
```


```
{
    "capabilities": [
        {
            "capabilityName": "my-ack",
            "arn": "arn:aws:eks:us-west-2:111122223333:capability/my-cluster/ack/my-
ack/abc123",
            "type": "ACK",
            "status": "ACTIVE",
            "createdAt": "2025-11-02T10:30:00.000000-07:00",
            "modifiedAt": "2025-11-02T10:32:15.000000-07:00"
        },
        {
            "capabilityName": "my-kro",
            "arn": "arn:aws:eks:us-west-2:111122223333:capability/my-cluster/kro/my-
kro/abc123",
            "type": "KRO",
            "status": "ACTIVE",
            "version": "v0.6.3",
            "createdAt": "2025-11-02T10:30:00.000000-07:00",
            "modifiedAt": "2025-11-02T10:32:15.000000-07:00"
        },
        {
            "capabilityName": "my-argocd",
            "arn": "arn:aws:eks:us-west-2:111122223333:capability/my-cluster/argocd/my-
argocd/abc123",
            "type": "ARGOCD",
            "status": "ACTIVE",
```

```
            "version": "3.1.8-eks-1",
            "createdAt": "2025-11-21T08:22:28.486000-05:00",
            "modifiedAt": "2025-11-21T08:22:28.486000-05:00"
        }
    ]
}
```

機能の説明

設定やステータスなど、特定の機能に関する詳細情報を取得します。

コンソール

1. https://console.aws.amazon.com/eks/home#/clusters で Amazon EKS コンソールを開きます。
2. クラスター名を選択して、クラスターの詳細ページを開きます。
3. [機能] タブを選択します。
4. [マネージド機能] から表示する機能を選択します。
5. ステータス、設定、作成時刻など、機能の詳細を表示します。

AWS CLI

`describe-capability` コマンドを使用して、詳細情報を表示します。*`region-code`* をクラスターがある AWS リージョンに、*`my-cluster`* をクラスターの名前に、*`capability-name`* を機能の作成時に付けた名前 (`my-ack`、`my-argocd`、`my-kro` など) に置き換えます。

```
aws eks describe-capability \
  --region region-code \
  --cluster-name my-cluster \
  --capability-name capability-name
```

出力の例:

```
{
  "capability": {
    "capabilityName": "my-ack",
    "capabilityArn": "arn:aws:eks:us-west-2:111122223333:capability/my-cluster/ack/my-
ack/abc123",
    "clusterName": "my-cluster",
    "type": "ACK",
```

```
    "roleArn": "arn:aws:iam::111122223333:role/AmazonEKSCapabilityACKRole",
    "status": "ACTIVE",
    "configuration": {},
    "tags": {},
    "health": {
      "issues": []
    },
    "createdAt": "2025-11-19T17:11:30.242000-05:00",
    "modifiedAt": "2025-11-19T17:11:30.242000-05:00",
    "deletePropagationPolicy": "RETAIN"
  }
}
```

機能の設定を更新する

機能の設定の一部は、作成後に更新できます。使用できる設定オプションは、機能タイプによって異なります。

> **Note**
>
> EKS 機能リソースは、パッチ適用やバージョン更新を含め、完全に管理されています。機能を更新すると、リソースの設定が更新されますが、マネージド機能コンポーネントのバージョンは更新されません。

AWS CLI

`update-capability` コマンドを使用して機能を変更します。

```
aws eks update-capability \
  --region region-code \
  --cluster-name my-cluster \
  --capability-name capability-name \
  --role-arn arn:aws:iam::111122223333:role/NewCapabilityRole
```

> **Note**
>
> 作成後にすべての機能プロパティを更新できるわけではありません。変更できる内容の詳細については、機能固有のドキュメントを参照してください。

機能を削除する

クラスター上で機能が不要になった場合は、機能リソースを削除できます。

> **Important**
>
> 機能を削除する前に、クラスターリソースを削除します。
> 機能リソースを削除しても、その機能を使用して作成されたリソースは自動的に削除されません。
>
> - すべての Kubernetes カスタムリソース定義 (CRD) は、クラスターにインストールされたままです。
> - ACK リソースはクラスターに残り、対応する AWS リソースはアカウントに残ります。
> - Argo CD アプリケーションとその Kubernetes リソースはクラスターに残ります。
> - kro ResourceGraphDefinitions とインスタンスはクラスターに残ります。
>
> 孤立したリソースを回避するため、機能を削除する前に、これらのリソースを削除する必要があります。
> 必要に応じて、ACK Kubernetes リソースに関連付けられた AWS リソースを保持することもできます。「ACK に関する考慮事項」を参照してください。

コンソール

1. https://console.aws.amazon.com/eks/home#/clusters で Amazon EKS コンソールを開きます。
2. クラスター名を選択して、クラスターの詳細ページを開きます。
3. [機能] タブを選択します。
4. [マネージド機能] のリストから削除する機能を選択します。
5. [機能を削除] を選択します。
6. 確認ダイアログで、機能の名前を入力して削除を確定します。
7. [削除] を選択します。

AWS CLI

`delete-capability` コマンドを使用して、機能リソースを削除します。

region-code をクラスターがある AWS リージョンに、*my-cluster* をクラスターの名前に、*capacity-name* を削除する機能名に置き換えます。

```
aws eks delete-capability \
  --region region-code \
  --cluster-name my-cluster \
  --capability-name capability-name
```

次のステップ

- the section called “Kubernetes resources” – 各機能タイプによって提供される Kubernetes リソースについて説明する
- the section called “ACK の概念” – ACK の概念およびリソースライフサイクルを理解する
- the section called “Argo CD の使用” – GitOps ワークフローに Argo CD 機能を使用する
- the section called “kro の概念” – kro の概念とリソース構成を理解する

EKS 機能コントローラーログにアクセスする

ACK、kro、Argo CD 用の EKS 機能コントローラーは、クラスター外の AWS マネージドインフラストラクチャで実行されます。Amazon CloudWatch Vended Logs を使用してこれらのコントローラーのログ配信を設定し、モニタリングとトラブルシューティングのためにコントローラーの動作を可視化することができます。

ログは構造化された JSON 形式で配信され、ログレベル、メッセージ、コントローラー名、調整識別子などの運用フィールドが含まれます。内部 AWS メタデータは配信前にフィルタリングされるため、機能コントローラーに関連する運用ログコンテンツのみを受け取れます。

サポートされているログのタイプ

各機能には、CloudWatch Vended Logs 配信ソースとして個別に設定できる 1 つ以上のログタイプがあります。

ACK

- EKS_CAPABILITY_ACK_LOGS

kro

- EKS_CAPABILITY_KRO_LOGS

Argo CD

- EKS_CAPABILITY_ARGOCD_APPLICATION_LOGS
- EKS_CAPABILITY_ARGOCD_APPLICATIONSET_LOGS
- EKS_CAPABILITY_ARGOCD_COMMITSERVER_LOGS
- EKS_CAPABILITY_ARGOCD_REPOSERVER_LOGS
- EKS_CAPABILITY_ARGOCD_SERVER_LOGS

ACK は、すべての ACK サービスコントローラーをカバーする単一のログタイプを使用します。ログレコードには、ログを生成した ACK サービスコントローラーを識別する `controllerGroup` フィールド (`s3.services.k8s.aws`、`rds.services.k8s.aws` など) が含まれます。このフィールドを使用して、CloudWatch Logs Insights または他のクエリツールで特定のサービスコントローラーのログをフィルタリングできます。

Argo CD には、コントローラーコンポーネントごとに 1 つずつの、5 つの異なるログタイプがあります。これにより、必要なコントローラーのみのログ記録を有効にし、異なる送信先にルーティングできます。

ログ配信の設定

機能コントローラーのログ配信を設定するには、AWS マネジメントコンソール または Amazon CloudWatch Logs API を使用します。詳細な設定手順については、「Amazon CloudWatch Logs ユーザーガイド」の「Enabling logging from AWS services」を参照してください。各機能コントローラーは個別の CloudWatch Vended Logs 配信ソースとして設定できるため、受信するログを選択できます。

コンソール

1. https://console.aws.amazon.com/eks/home#/clusters で Amazon EKS コンソールを開きます。
2. クラスター名を選択します。
3. [機能] タブを選択し、機能を選択します。
4. [ログ配信] セクションで、[追加] を選択します。
5. 機能のログタイプを選択し、送信先を選択します。

6. [追加] を選択して配信を作成します。

Amazon CloudWatch API の使用

ログ記録の設定には次の 3 つのステップが必要です。

1. CloudWatch `PutDeliverySource` API を使用して機能の配信ソースを作成する 機能 ARN を `resourceArn` として使用し、目的のログタイプを指定します。
2. `PutDeliveryDestination` を使用して配信先を作成します。CloudWatch ロググループ、S3 バケット、または Amazon Data Firehose 配信ストリームの ARN を指定します。
3. `CreateDelivery` を使用して送信元と送信先を接続する配信を作成する

機能 ARN は `describe-capability` コマンドを使用して取得できます。

```
aws eks describe-capability \
  --region region-code \
  --cluster-name my-cluster \
  --capability-name my-capability \
  --query 'capability.capabilityArn' --output text
```

複数の配信を作成することで、1 つの機能 (配信ソース) から複数の送信先にログを送信するよう設定できます。複数の機能から同じ送信先にログを送信することもできます。

IAM アクセス許可

選択した送信先によっては、ログ配信が成功するように CloudWatch ロググループ、S3 バケット、または Data Firehose 配信ストリームの IAM ポリシーまたはロールを設定する必要がある場合があります。AWS アカウント間でログを送信する場合は、`PutDeliveryDestinationPolicy` API を使用して、送信先への配信を許可する IAM ポリシーを設定します。詳細については、「CloudWatch Vended Logs permissions」のドキュメントを参照してください。

例: CloudWatch ログインサイトでのログのクエリ

すべてのコントローラーのエラーログを表示するには、次のようにします。

```
fields @timestamp, controller, message, error
| filter level = "error"
| sort @timestamp desc
| limit 50
```

サービスコントローラーによるフィルタリング、調整サイクルの追跡、Argo CD アプリケーションのフィルタリングなどの、その他のクエリの例については「トラブルシューティングにコントローラーログを使用する」を参照してください。

ログの表示

ログ配信を設定すると、ログは指定された送信先に配信されます。ログにアクセスする方法は、選択した送信先タイプによって異なります。

- CloudWatch Logs – CloudWatch Logs コンソールでログを表示する、AWS CLI コマンドを使用する、または CloudWatch Logs Insights でクエリを実行する。
- Amazon S3 – S3 コンソール、AWS CLI、または Amazon Athena などの分析ツールを使用して、S3 バケット内のオブジェクトとしてログにアクセスする。
- Amazon Data Firehose – ログは、設定された Firehose ターゲット (S3、OpenSearch Service、Redshift など) にストリーミングされます。

料金

CloudWatch Vended Logs の料金は、選択した送信先に基づいてログの配信とストレージに適用されます。CloudWatch Vended Logs は、標準の CloudWatch Logs と比較して、組み込みの AWS 認証と認可により、信頼性が高く安全なログ配信を低価格で実現します。詳細については、「CloudWatch 料金ページのVended Logs セクション」を参照してください。

機能の Kubernetes リソース

クラスターで機能を有効にしたら、後は通常クラスターで Kubernetes カスタムリソースを作成して管理することでクラスターを操作することになります。機能ごとに独自のカスタムリソース定義 (CRD) セットを策定して、それぞれに固有の機能で Kubernetes API を拡張できます。

Argo CD リソース

Argo CD 機能を有効にすると、次の Kubernetes リソースを作成および管理できます。

アプリケーション

Git リポジトリからターゲットクラスターへのデプロイを定義します。`Application` リソースでは、ソースリポジトリ、ターゲット名前空間、同期ポリシーを指定します。Argo CD 機能インスタンスごとに最大 3,000 個の `Application` リソースを作成できます。

ApplicationSet

テンプレートから複数の Application リソースを生成し、マルチクラスターとマルチ環境のデプロイを可能にします。ApplicationSet リソースは、ジェネレーターを使用して、クラスターリストや Git ディレクトリといったソースに基づいて Application リソースを動的に作成します。

AppProject

Application リソースを論理的にグループ化してアクセスを制御します。AppProject リソースは、Application リソースが使用できるリポジトリ、クラスター、名前空間を定義して、マルチテナンシーとセキュリティ境界を有効にします。

Application リソースの例:

```
apiVersion: argoproj.io/v1alpha1
kind: Application
metadata:
  name: my-app
  namespace: argocd
spec:
  project: default
  source:
    repoURL: https://github.com/org/repo
    targetRevision: main
    path: manifests
  destination:
    server: https://kubernetes.default.svc
    namespace: production
```

Argo CD のリソースと概念の詳細については、「the section called “Argo CD の概念”」を参照してください。

kro リソース

kro 機能を有効にすると、次の Kubernetes リソースを作成および管理できます。

ResourceGraphDefinition (RGD)

複数の Kubernetes と AWS リソースを高次の抽象化に構成するカスタム API を定義します。プラットフォームチームは、ガードレールに沿ってパターンを再利用可能できるように ResourceGraphDefinition リソースを作成します。

カスタムリソースインスタンス

ResourceGraphDefinition リソースを作成したら、ResourceGraphDefinition で定義されたカスタム API のインスタンスを作成できます。kro は、ResourceGraphDefinition に指定されたリソースを自動的に作成および管理します。

ResourceGraphDefinition リソースの例:

```
apiVersion: kro.run/v1alpha1
kind: ResourceGraphDefinition
metadata:
  name: web-application
spec:
  schema:
    apiVersion: v1alpha1
    kind: WebApplication
    spec:
      name: string
      replicas: integer
  resources:
    - id: deployment
      template:
        apiVersion: apps/v1
        kind: Deployment
        # ... deployment spec
    - id: service
      template:
        apiVersion: v1
        kind: Service
        # ... service spec
```

WebApplication インスタンスの例:

```
apiVersion: v1alpha1
kind: WebApplication
metadata:
  name: my-web-app
  namespace: default
spec:
  name: my-web-app
  replicas: 3
```

このインスタンスを適用すると、kro は `ResourceGraphDefinition` に定義された `Deployment` リソースと `Service` リソースを自動的に作成します。

kro のリソースと概念の詳細については、「the section called “kro の概念”」を参照してください。

ACK リソース

ACK 機能を有効にすると、Kubernetes カスタムリソースを使用して AWS リソースを作成および管理できます。ACK では 50 を超える AWS サービスの 200 を超える CRD を策定できるため、Kubernetes ワークロードと共に AWS リソースを定義し、Kubernetes で専用の AWS インフラストラクチャリソースを管理できます。

ACK リソースの例:

S3 バケット

: `Bucket` リソースは、バージョニング、暗号化、ライフサイクルポリシーを備えた Amazon S3 バケットを作成および管理します。

RDS DBInstance

: `DBInstance` リソースは、自動バックアップとメンテナンス期間を備えた Amazon RDS データベースインスタンスをプロビジョニングして管理します。

DynamoDB テーブル

: `Table` リソースは、プロビジョニング済みのキャパシティまたはオンデマンドキャパシティを備えた DynamoDB テーブルを作成および管理します。

IAM ロール

: `Role` リソースは、AWS サービスアクセスの信頼ポリシーとアクセス許可ポリシーを備えた IAM ロールを定義します。

Lambda 関数

: `Function` リソースは、コード、ランタイム、実行ロール設定を備えた Lambda 関数を作成および管理します。

`Bucket` リソースの指定例:

```
apiVersion: s3.services.k8s.aws/v1alpha1
kind: Bucket
metadata:
```

```
  name: my-app-bucket
spec:
  name: my-unique-bucket-name-12345
  versioning:
    status: Enabled
  encryption:
    rules:
      - applyServerSideEncryptionByDefault:
          sseAlgorithm: AES256
```

ACK のリソースと概念の詳細については、「the section called “ACK の概念”」を参照してください。

リソースの制限

EKS 機能には、次のリソース制限があります。

Argo CD の使用制限:

- Argo CD 機能インスタンスあたり最大 3,000 個の `Application` リソース
- Argo CD 機能インスタンスあたり最大 100 個のリモートクラスターを設定可能

リソース設定の制限:

- Argo CD では `Application` リソースあたり最大 150 個の Kubernetes リソース
- kro では `ResourceGraphDefinition` あたり最大 64 個の Kubernetes リソース

> **Note**
>
> これらの制限は、各機能インスタンスで管理されるリソースの数に適用されます。さらに厳しい制限が必要な場合は、複数のクラスターに機能をデプロイできます。

次のステップ

機能に固有のタスクと高度な設定については、以下のトピックを参照してください。

- the section called “ACK の概念” – ACK の概念およびリソースライフサイクルを理解する
- the section called “Argo CD の使用” – GitOps ワークフローに Argo CD 機能を使用する

- the section called “kro の概念” – kro の概念とリソース構成を理解する

EKS 機能と考慮事項

EKS 機能をデプロイする前に、アクセスコントロール設計、マネージドソリューションかセルフマネージドソリューションかの選択、マルチクラスターデプロイ用のアーキテクチャパターン、運用のベストプラクティスなど、重要な考慮事項を確認します。

機能 IAM ロールと Kubernetes RBAC

EKS 機能リソースごとに、機能 IAM ロールが設定されています。機能ロールを使用すると、自分の代わりに EKS 機能を動作させるための AWS サービスアクセス許可を付与できます。例えば、EKS Capability for ACK を使用して Amazon S3 バケットを管理するには、S3 バケットの管理アクセス許可を機能に付与して、バケットを作成および管理できるようにします。

機能を設定すると、クラスター内の Kubernetes カスタムリソースで AWS の S3 リソースを作成および管理できます。Kubernetes RBAC は、こうしたカスタムリソースをどのユーザーとグループが作成および管理できるかを決定するためのクラスター内アクセスコントロールメカニズムです。例えば、特定の Kubernetes RBAC ユーザーとグループに、選択した名前空間で `Bucket` リソースを作成および管理するためのアクセス許可を付与します。

このように、IAM と Kubernetes RBAC は、エンドツーエンドのアクセス管理システムを二等分して、EKS の機能とリソースに関連するアクセス許可を管理しています。ここで重要なのは、ユースケースに適した IAM アクセス許可と RBAC アクセスポリシーを組み合わせることです。

機能 IAM ロールと Kubernetes アクセス許可の詳細については、「the section called “EKS 機能に関する考慮事項”」を参照してください。

マルチクラスターアーキテクチャパターン

複数のクラスターに機能をデプロイする際は、よく使用される次のアーキテクチャパターンを考慮してください。

一元管理でのハブとスポーク

一元管理クラスターで 3 つすべての機能を実行して、ワークロードをオーケストレーションし、複数のワークロードクラスター全体にわたってクラウドインフラストラクチャを管理します。

- 管理クラスター上の Argo CD は、リージョンやアカウントが異なる複数のワークロードクラスターにアプリケーションをデプロイします。

- 管理クラスター上の ACK は、すべてのクラスターの AWS リソース (RDS、S3、IAM) をプロビジョニングします。
- 管理クラスター上の kro は、すべてのクラスターにわたって動作するポータブルプラットフォーム抽象化を作成します。

このパターンにより、ワークロードとクラウドインフラストラクチャの管理が一元化されるため、組織が多数のクラスターを管理する場合の運用をシンプルにできます。

分散型 GitOps

ワークロードとクラウドインフラストラクチャは、ワークロードが実行されているのと同じクラスター上の機能で管理されます。

- Argo CD は、ローカルクラスターのアプリケーションリソースを管理します。
- ACK リソースは、クラスターとワークロードのニーズに応じて使用されます。
- kro プラットフォーム抽象化がインストールされ、ローカルリソースをオーケストレーションします。

このパターンでは運用が分散され、各チームは 1 つ以上のクラスターでそれぞれ独自の専用プラットフォームサービスを管理します。

ハイブリッド ACK デプロイでのハブとスポーク

モデルを一元化および分散すると共に、範囲と所有権に基づいてアプリケーションのデプロイとリソース管理を一元化します。

- ハブクラスター:
  - Argo CD は、ローカルクラスターとすべてのリモートワークロードクラスターへの GitOps デプロイを管理します。
  - ACK は、管理クラスター上で管理者範囲のリソース (本番稼働用データベース、IAM ロール、VPC) を操作します。
  - kro は、管理クラスター上でプラットフォーム抽象化を再利用可能にします。
- スポーククラスター:
  - ワークロードは、一元化されたハブクラスター上で Argo CD によって管理されます。
  - ACK は、ローカルでワークロード範囲のリソース (S3 バケット、ElastiCache インスタンス、SQS キュー) を操作します。

- kro は、ローカルでリソース構成と構成要素パターンを操作します。

このパターンは関心を分離します。プラットフォームチームは管理クラスター上で重要なインフラストラクチャを (必要に応じてワークロードクラスターも) 一元管理し、アプリケーションチームはワークロードと共にクラウドリソースを指定および管理します。

パターンを選択する

アーキテクチャを選択する場合は、次の要因を考慮してください。

- 組織構造: 中央集権型のプラットフォームではハブパターンが好まれますが、分散型のチームではクラスターごとの機能が好まれることがあります。
- リソース範囲: 管理者範囲のリソース (データベース、IAM) は一元管理の方がメリットを得られることが多く、一方ワークロードリソース (バケット、キュー) はローカルで管理することもできます。
- セルフサービス: 中央集権型のプラットフォームチームは、規範的なカスタムリソースを作成および配布して、ワークロードの一般的なニーズに合わせてクラウドリソースの安全なセルフサービスを有効にできます。
- クラスターフリート管理: クラスターを一元管理すると、お客様所有のコントロールプレーンにより、EKS クラスターフリートを他の管理者範囲のリソースと共に管理できます。
- コンプライアンス要件: 組織によっては、監査とガバナンスの一元管理が必要です。
- 運用の複雑さ: 機能インスタンスが少ないほど、運用はシンプルになりますが、ボトルネックが発生する可能性があります。

> **Note**
>
> 1 つのパターンから始めて、プラットフォームの成熟に伴って別のパターンに進化させていくことができます。機能は独立しているため、ニーズに応じてクラスター間でデプロイ方法を変えることができます。

EKS 機能をセルフマネージドソリューションと比較する

EKS 機能では、EKS で実行される一般的な Kubernetes ツールとコントローラーを完全に管理できます。これは、クラスターにインストールして運用するセルフマネージドソリューションとは異なります。

主な違い

デプロイと管理

AWS が EKS 機能を完全に管理するため、コンポーネントソフトウェアのインストール、設定、メンテナンスが必要ありません。また、AWS はクラスターに必要なすべての Kubernetes カスタムリソース定義 (CRD) を自動的にインストールして管理します。

セルフマネージドソリューションでは、Helm チャート、kubectl、またはその他の演算子を使用して、クラスターソフトウェアをインストールおよび設定します。セルフマネージドソリューションのソフトウェアライフサイクルとランタイム設定を完全に制御できるため、ソリューションのどのレイヤーでもカスタマイズを実施できます。

運用とメンテナンス

AWS は、自動更新とセキュリティパッチを使用して、EKS 機能のパッチ適用やその他のソフトウェアライフサイクルの運用を管理します。EKS 機能は、AWS 機能との統合により設定を効率化し、高可用性と耐障害性を備えており、コントローラーワークロードをクラスター内でトラブルシューティングする必要がありません。

セルフマネージドソリューションでは、コンポーネントのヘルスとログのモニタリング、セキュリティパッチとバージョン更新の適用、複数のレプリカとポッド中断の予算による高可用性の設定、コントローラーのワークロードに関する問題のトラブルシューティングと修復、リリースとバージョンの管理を行う必要があります。デプロイを完全に制御できますが、そのためには組織の標準とセキュリティコンプライアンス要件に沿ってプライベートクラスターアクセスやその他の統合を実現するカスタムソリューションが必要です。

リソース消費

EKS 機能はクラスターとは別に EKS で実行されるため、ノードリソースとクラスターリソースが解放されます。クラスターワークロードリソースを使用せず、ワーカーノード上で CPU やメモリを消費しません。また、自動的にスケールして、クラスターキャパシティプランニングへの影響を最小限に抑えます。

セルフマネージドソリューションは、ワーカーノードでコントローラーやその他のコンポーネントを実行するためのワーカーノードリソース、クラスター IP、その他のクラスターリソースを直接消費します。クラスターサービスを管理するには、ワークロードのキャパシティプランニングが必要であり、スケールと高可用性の要件を管理するためにリソースのリクエストと制限を計画して設定する必要があります。

機能のサポート

EKS 機能は完全マネージド型のサービス機能であるため、セルフマネージドソリューションと比べると、本質的に自由度が低くなります。ほとんどの特徴とユースケースをサポートしますが、セルフマネージドソリューションと比べると、カバレッジに違いがあります。

セルフマネージドソリューションを使用すると、ソフトウェアの設定やオプションなど機能性のさまざまな側面を完全に制御できます。独自のカスタムイメージを実行する、設定のあらゆる側面をカスタマイズする、セルフマネージドソリューションの機能性を完全に制御するといったこともできます。

コストに関する考慮事項

EKS 機能のリソースごとに時間単位でコストが伴います。これは機能タイプによって異なります。EKS 機能で管理されるクラスターリソースにも、独自の料金に関連付けられた時間単位のコストが伴います。詳細については「Amazon EKS の料金」を参照してください。

セルフマネージドソリューションには AWS の料金に関連する直接コストはありませんが、コントローラーおよび関連するワークロードでクラスターコンピューティングリソースが使用された場合には料金が発生します。ノードとクラスターのリソースの消費以外にも、セルフマネージドソリューションに伴う総所有コストには、メンテナンス、トラブルシューティング、サポートの運用にかかるオーバーヘッドとコストが含まれます。

EKS 機能かセルフマネージドソリューションかを選択する

EKS 機能 基本要件を満たすためにクラスタープラットフォームを運用することではなく、運用オーバーヘッドを削減し、ソフトウェアとシステムの差別化された価値に集中する場合は、この選択肢を検討してください。セキュリティパッチとソフトウェアライフサイクル管理の運用に伴う負担を最小限に抑える、アプリケーションワークロードのノードとクラスターのリソースを解放する、設定とセキュリティ管理をシンプルにする、AWS サポートカバレッジからメリットが得られるといった場合は、EKS 機能を使用します。EKS 機能は、本番稼働のほとんどのユースケースに最適であり、新規デプロイに推奨されるアプローチです。

セルフマネージドソリューション 特定の Kubernetes リソース API バージョンを必要とする、コントローラーを独自に構築する、既存の自動化とツールがセルフマネージドデプロイを軸に構築されている、コントローラーランタイム設定をきめ細かくカスタマイズする必要があるといったは、この選択肢を検討します。セルフマネージドソリューションでは、特殊なユースケースに柔軟に対処し、デプロイとランタイム設定を完全に制御できます。

> **Note**
>
> EKS 機能をクラスター内でセルフマネージドソリューションと共存させることができるため、段階的に移行を進めることができます。

機能固有の比較

機能に固有の特徴、アップストリームの違い、移行パスなどの詳細な比較については、以下を参照してください。

- the section called “セルフマネージドとの比較”
- the section called “セルフマネージドとの比較”
- the section called “セルフマネージドとの比較”

Kubernetes 用 AWS コントローラー (ACK) で Kubernetes から AWS リソースをデプロイする

Kubernetes 用 AWS コントローラー (ACK) を使用すると、Kubernetes から直接 AWS サービスリソースを定義および管理できます。Kubernetes 用 AWS コントローラー (ACK) により、Kubernetes カスタムリソースを使用してワークロードリソースとクラウドインフラストラクチャを管理できるほか、使い慣れた Kubernetes API とツールを使用してアプリケーションワークロードも管理できます。

AWS では EKS の機能を使用して ACK を完全に管理できるため、クラスターに ACK コントローラーをインストールして保守およびスケールする必要がありません。

ACK の仕組み

ACK は、Kubernetes カスタムリソース仕様を AWS API コールに変換します。AWS サービスリソースを表す Kubernetes カスタムリソースを作成、更新、または削除すると、ACK は AWS リソースを作成、更新、または削除するために必要な AWS API コールを実行します。

ACK でサポートされている AWS リソースごとに独自のカスタムリソース定義 (CRD) を用意して、そのリソース設定を指定するための Kubernetes API スキーマを定義します。例えば、S3 の CRD にバケットやバケットポリシーなどの S3 リソースを含めることができます。

ACK は、Kubernetes カスタムリソースに定義された目的の状態に合わせて AWS リソースの状態を継続的に調整します。リソースが目的の状態からドリフトした場合、ACK はそれを検出し、是正アクションを実行して調整し直します。Kubernetes リソースへの変更は、すぐに AWS リソースの状態に反映されます。一方、アップストリームの AWS リソースに対する変更の場合、ドリフトをパッシブに検出して修復するには 10 時間もかかることがありますが (再同期期間)、通常ははるかに早く終了します。

S3 バケットリソースマニフェストの例

```
apiVersion: s3.services.k8s.aws/v1alpha1
kind: Bucket
metadata:
  name: my-ack-bucket
spec:
  name: my-unique-bucket-name
```

このカスタムリソースをクラスターに適用すると、アカウントに Amazon S3 バケットがまだ存在しない場合には ACK によって作成されます。以後、このリソースに変更を加えた場合、例えばデフォルト以外のストレージ階層を指定したりポリシーを追加したりすると、その変更が AWS 内の S3 リソースに適用されます。このリソースがクラスターから削除されると、デフォルトでは AWS 内の S3 バケットも削除されます。

ACK のメリット

ACK では Kubernetes ネイティブの AWS リソース管理を利用できるため、普段アプリケーションで使用しているのと同じ Kubernetes API とツールで AWS リソースを管理できます。このように統合が図られているため、さまざまなツールを切り替えたり、Infrastructure as Code 管理ワークフローを個別に習得したりする必要がありません。Kubernetes マニフェストで AWS リソースを宣言的に定義して、GitOps ワークフローとインフラストラクチャを、既存の開発プロセスとシームレスに統合するコードプラクティスとして有効にします。

ACK は、AWS リソースの目的の状態を実際の状態で継続的に調整し、ドリフトを修正して、インフラストラクチャ全体の一貫性を確保します。このように継続的に調整することで、AWS リソースに必須の帯域外変更が、宣言した設定に合わせて自動的に元に戻されるため、Infrastructure as Code の整合性が維持されます。複数の AWS アカウントとリージョンにわたるリソースを管理するように ACK を設定することで、他にツールを追加することなく、複雑なマルチアカウントアーキテクチャを実現できます。

組織全体で他のインフラストラクチャ管理ツールから移行するできるように、ACK がリソースの採用をサポートしているため、再作成することなく既存の AWS リソースを ACK の管理下に置くことができます。ACK ではこのほか、読み取り専用リソースにより、変更アクセス権がなくても AWS リソースを監視でき、注釈により、Kubernetes リソースがクラスターから削除された場合でもオプションで AWS リソースを保持できます。

EKS Capability for ACK の詳細と使用開始については、「the section called “ACK の概念”」および「the section called “考慮事項”」を参照してください。

サポートされる AWS サービス

ACK は、次のような幅広い AWS サービスをサポートしています。

- Amazon EC2
- Amazon S3
- Amazon RDS
- Amazon DynamoDB
- Amazon ElastiCache
- Amazon EKS
- Amazon SQS
- Amazon SNS
- AWS Lambda
- AWS IAM

一般提供アップストリームとしてリストされているすべての AWS サービスが、EKS Capability for ACK でサポートされています。サポートされている AWS サービスの完全なリストについては、ACK ウェブサイトの「ACK サービスページ」を参照してください。

他の EKS マネージド機能との統合

ACK は、他の EKS マネージド機能と統合されています。

- Argo CD: Argo CD を使用すると、複数のクラスターにまたがる ACK リソースのデプロイを管理して、AWS インフラストラクチャの GitOps ワークフローを有効にできます。
  - ACK を Argo CD と組み合わせると GitOps のメリットが広がりますが、ACK を git と統合する必要はありません。

- kro (Kube Resource Orchestrator): kro を使用すると、ACK リソースから複雑なリソースを作成して、高次の抽象化を作成してリソース管理をシンプルにできます。
  - kro で Kubernetes リソースと AWS リソースの両方を定義することで、複合カスタムリソースを作成できます。チームメンバーは、こうしたカスタムリソースを使用して、複雑なアプリケーションをすばやくデプロイできます。

ACK の使用を開始する

EKS Capability for ACK の使用を開始するには:

1. 自分の代わりに ACK で AWS リソースを管理できるように、IAM 機能ロールを作成して、必要なアクセス許可を設定します。
2. AWS コンソール、AWS CLI、または任意の Infrastructure as Code ツールを使用して、EKS クラスターに ACK 機能リソースを作成します。
3. Kubernetes カスタムリソースをクラスターに適用して、Kubernetes での AWS リソースの管理を開始します。

ACK 機能を作成する

この章では、Amazon EKS クラスターに ACK 機能を作成する方法について説明します。

前提条件

ACK 機能を作成する前に、以下の点を確認してください。

- Amazon EKS クラスター
- ACK で AWS リソースを管理するためのアクセス許可が IAM 機能ロールに付与されている
- EKS クラスターに機能リソースを作成するための十分な IAM アクセス許可がある
- 適切な CLI ツールがインストールおよび設定されているか、EKS コンソールにアクセスできる

IAM 機能ロールを作成する手順については、「the section called “機能 IAM ロール”」を参照してください。

> **Important**
>
> ACK は、AWS リソースを作成、変更、削除する権限を付与するインフラストラクチャ管理機能です。これは、制御に慎重を要する管理者範囲の機能です。クラスターに Kubernetes リソースを作成するためのアクセス許可があれば誰でも、IAM 機能ロールのアクセス許可に従って、ACK で AWS リソースを効果的に作成できます。ACK でどの AWS リソースを作成および管理できるかは、どの IAM 機能ロールを使用するかによって決まります。最小特権のアクセス許可を付与した適切なロールを作成する際のガイダンスについては、「the section called “機能 IAM ロール”」および「the section called “EKS 機能に関する考慮事項”」を参照してください。

ツールを選択する

AWS マネジメントコンソール、AWS CLI、または eksctl を使用して、ACK 機能を作成できます。

- the section called “コンソール” - 指示に従って操作する場合はコンソールを使用します。
- the section called “ AWS CLI” - スクリプトを作成して自動化を図る場合は AWS CLI を使用します。
- the section called “eksctl” - Kubernetes ネイティブの操作をする場合は eksctl を使用します。

ACK 機能を作成するとどうなるか

ACK 機能を作成すると、次のようになります。

1. EKS が、ACK 機能サービスを作成して、クラスター内のリソースをモニタリングおよび管理するように設定します。
2. カスタムリソース定義 (CRD) がクラスターにインストールされます。
3. Kubernetes のベースラインアクセス許可を付与する機能固有のアクセスエントリポリシーを使用して、IAM 機能ロールのアクセスエントリを自動的に作成します (「the section called “EKS 機能に関する考慮事項”」を参照)。
4. 機能が、指定した IAM 機能ロールを引き受けます。
5. ACK が、クラスター内のカスタムリソースを監視し始めます。
6. 機能のステータスが `CREATING` から `ACTIVE` に変わります。

アクティブになったら、AWS リソースを管理するための ACK カスタムリソースをクラスターに作成できます。

> **Note**
>
> 自動的に作成されたアクセスエントリには、AWS リソースを管理するための ACK アクセス許可を付与する `AmazonEKSACKPolicy` が含まれます。Kubernetes シークレットを参照する一部の ACK リソース (パスワード付きの RDS データベースなど) には、追加のアクセスエントリポリシーが必要です。アクセスエントリの詳細と追加のアクセス許可の設定方法については、「the section called “EKS 機能に関する考慮事項”」を参照してください。

次のステップ

ACK 機能の作成後:

- the section called “ACK の概念” - ACK の概念を理解し、AWS リソースの使用を開始する
- the section called “アクセス許可の設定” - IAM アクセス許可およびマルチアカウントパターンを設定する

コンソールを使用して ACK 機能を作成する

AWS マネジメントコンソールを使用して、Amazon EKS クラスターに ACK 機能を作成します。

ACK 機能を作成する

1. https://console.aws.amazon.com/eks/home#/clusters で Amazon EKS コンソールを開きます。
2. クラスター名を選択して、クラスターの詳細ページを開きます。
3. [機能] タブを選択します。
4. 左側のナビゲーションで、[Kubernetes 用 AWS コントローラー (ACK)] を選択します。
5. [Kubernetes 用 AWS コントローラー (ACK) 機能] を選択します。
6. IAM 機能ロールに対して以下の操作を行います。
   - IAM 機能ロールが既に存在する場合は、ドロップダウンから選択します。
   - ロールを作成する必要がある場合は、[管理者ロールを作成] を選択します。

IAM コンソールが新しいタブに開かれます。信頼ポリシーと AdministratorAccess 管理ポリシーには、値があらかじめ入力されています。このポリシーを選択解除し、必要に応じて他のアクセス許可を追加できます。

ロールを作成して EKS コンソールに戻ると、そのロールが自動的に選択されます。

> **Important**
>
> 提案された AdministratorAccess ポリシーは、幅広いアクセス許可を付与し、使用開始を効率化することを目的としています。本番稼働で使用する場合は、これをカスタムポリシーに置き換えて、そのポリシーでは ACK で管理する特定の AWS サービスに必要なアクセス許可のみを付与します。最小特権ポリシーを作成する際のガイダンスについては、「the section called “アクセス許可の設定”」および「the section called “EKS 機能に関する考慮事項”」を参照してください。

7. [作成] を選択します。

機能作成プロセスが始まります。

機能がアクティブであることを確認する

1. [機能] タブで、ACK 機能のステータスを表示します。
2. ステータスが CREATING から ACTIVE に変わるまで待ちます。
3. アクティブになったら、この機能を使用する準備は完了です。

機能のステータスとトラブルシューティングの詳細については、「the section called “機能の使用”」を参照してください。

カスタムリソースが使用可能であることを確認する

機能がアクティブになったら、ACK カスタムリソースがクラスターで使用可能になっていることを確認します。

コンソールの使用

1. Amazon EKS コントロールでクラスターに移動します。
2. [リソース] タブを選択します。

3. [拡張機能] を選択します。
4. [CustomResourceDefinitions] を選択します。

AWS リソースに使用可能な多数の CRD がリストされます。

kubectl を使用する

```
kubectl api-resources | grep services.k8s.aws
```

AWS リソースに使用可能な多数の API がリストされます。

> **Note**
>
> Kubernetes 用 AWS コントローラーの機能は、さまざまな AWS リソース用に多数の CRD をインストールします。

次のステップ

- the section called “ACK の概念” - ACK の概念を理解して使用を開始する
- the section called “アクセス許可の設定” - 他の AWS サービスに対して IAM アクセス許可を設定する
- the section called “機能の使用” - ACK 機能リソースを管理する

AWS CLI を使用して ACK 機能を作成する

AWS CLI 使用して Amazon EKS クラスターに ACK 機能を作成します。この手順では、IAM ロールの作成、および機能がアクティブであることの確認について説明します。

前提条件

- AWS CLI バージョン `2.12.3` 以降。バージョンを確認するには、`aws --version` を実行します。詳細についてはAWS コマンドラインインターフェイスユーザーガイドの「インストール」を参照してください。
- **`kubectl`** - Kubernetes クラスターを操作するためのコマンドラインツール。詳細については、「the section called “kubectl と eksctl のセットアップ”」を参照してください。

ステップ 1: IAM 機能ロールを作成する

信頼ポリシーファイルを作成します。

```
cat > ack-trust-policy.json << 'EOF'
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "capabilities.eks.amazonaws.com"
      },
      "Action": [
        "sts:AssumeRole",
        "sts:TagSession"
      ]
    }
  ]
}
EOF
```

IAM ロールを作成します。

```
aws iam create-role \
  --role-name ACKCapabilityRole \
  --assume-role-policy-document file://ack-trust-policy.json
```

`AdministratorAccess` マネージドポリシーをロールにアタッチします。

```
aws iam attach-role-policy \
  --role-name ACKCapabilityRole \
  --policy-arn arn:aws:iam::aws:policy/AdministratorAccess
```

> **Important**
>
> 提案された `AdministratorAccess` ポリシーは、幅広いアクセス許可を付与し、使用開始を効率化することを目的としています。本番稼働で使用する場合は、これをカスタムポリシーに置き換えて、そのポリシーでは ACK で管理する特定の AWS サービスに必要なアクセス許可のみを付与します。最小特権ポリシーを作成する際のガイダンスについては、「the

section called “アクセス許可の設定”」および「the section called “EKS 機能に関する考慮事項”」を参照してください。

ステップ 2: ACK 機能を作成する

ACK 機能リソースをクラスターに作成します。*region-code* はクラスターがある AWS リージョンに、*my-cluster* はクラスターの名前に置き換えます。

```
aws eks create-capability \
  --region region-code \
  --cluster-name my-cluster \
  --capability-name my-ack \
  --type ACK \
  --role-arn arn:aws:iam::$(aws sts get-caller-identity --query Account --output
 text):role/ACKCapabilityRole \
  --delete-propagation-policy RETAIN
```

このコマンドはすぐに戻りますが、EKS が必要な機能インフラストラクチャとコンポーネントを作成するため、機能がアクティブになるまでにはしばらく時間がかかります。EKS は、この機能に関連する Kubernetes カスタムリソース定義をその作成時にクラスターにインストールします。

Note

クラスターが存在しないというエラーやアクセス許可がないというエラーが表示された場合は、以下の点を確認します。

- クラスター名が正しいこと
- AWS CLI が正しいリージョンに設定されていること
- 必要な IAM アクセス許可を追加したこと

ステップ 3: 機能がアクティブであることを確認する

機能がアクティブになるまで待機します。*region-code* はクラスターがある AWS リージョンに、*my-cluster* はクラスターの名前に置き換えます。

```
aws eks describe-capability \
  --region region-code \
  --cluster-name my-cluster \
```

```
  --capability-name my-ack \
  --query 'capability.status' \
  --output text
```

ステータスが ACTIVE と表示されたら、機能の準備は完了です。ステータスが ACTIVE になるまで、次のステップに進まないでください。

機能の詳細全体を表示することもできます。

```
aws eks describe-capability \
  --region region-code \
  --cluster-name my-cluster \
  --capability-name my-ack
```

ステップ 4: カスタムリソースが使用可能であることを確認する

機能がアクティブになったら、ACK カスタムリソースがクラスターで使用可能になっていることを確認します。

```
kubectl api-resources | grep services.k8s.aws
```

AWS リソースに使用可能な多数の API がリストされます。

> **Note**
>
> Kubernetes 用 AWS コントローラーの機能は、さまざまな AWS リソース用に多数の CRD をインストールします。

次のステップ

- the section called “ACK の概念” - ACK の概念を理解して使用を開始する
- the section called “アクセス許可の設定” - 他の AWS サービスに対して IAM アクセス許可を設定する
- the section called “機能の使用” - ACK 機能リソースを管理する

eksctl を使用して ACK 機能を作成する

eksctl を使用して Amazon EKS クラスターに ACK 機能を作成します。

> **Note**
>
> 以下のステップを実行するには、eksctl バージョン `0.215.0` 以降が必要です。バージョンを確認するには、`eksctl version` を実行します。

ステップ 1: IAM 機能ロールを作成する

信頼ポリシーファイルを作成します。

```
cat > ack-trust-policy.json << 'EOF'
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "capabilities.eks.amazonaws.com"
      },
      "Action": [
        "sts:AssumeRole",
        "sts:TagSession"
      ]
    }
  ]
}
EOF
```

IAM ロールを作成します。

```
aws iam create-role \
  --role-name ACKCapabilityRole \
  --assume-role-policy-document file://ack-trust-policy.json
```

`AdministratorAccess` マネージドポリシーをロールにアタッチします。

```
aws iam attach-role-policy \
  --role-name ACKCapabilityRole \
  --policy-arn arn:aws:iam::aws:policy/AdministratorAccess
```

> **Important**
>
> 提案された AdministratorAccess ポリシーは、幅広いアクセス許可を付与し、使用開始を効率化することを目的としています。本番稼働で使用する場合は、これをカスタムポリシーに置き換えて、そのポリシーでは ACK で管理する特定の AWS サービスに必要なアクセス許可のみを付与します。最小特権ポリシーを作成する際のガイダンスについては、「the section called “アクセス許可の設定”」および「the section called “EKS 機能に関する考慮事項”」を参照してください。

ステップ 2: ACK 機能を作成する

eksctl を使用して ACK 機能を作成します。*region-code* はクラスターがある AWS リージョンに、*my-cluster* はクラスターの名前に置き換えます。

```
eksctl create capability \
  --cluster my-cluster \
  --region region-code \
  --name ack \
  --type ACK \
  --role-arn arn:aws:iam::$(aws sts get-caller-identity --query Account --output
 text):role/ACKCapabilityRole
```

このコマンドはすぐに戻りますが、機能がアクティブになるまでにはしばらく時間がかかります。

ステップ 3: 機能がアクティブであることを確認する

機能のステータスを確認します。

```
eksctl get capability \
  --cluster my-cluster \
  --region region-code \
  --name ack
```

ステータスが ACTIVE と表示されたら、機能の準備は完了です。

ステップ 4: カスタムリソースが使用可能であることを確認する

機能がアクティブになったら、ACK カスタムリソースがクラスターで使用可能になっていることを確認します。

```
kubectl api-resources | grep services.k8s.aws
```

AWS リソースに使用可能な多数の API がリストされます。

> **Note**
>
> Kubernetes 用 AWS コントローラーの機能は、さまざまな AWS リソース用に多数の CRD をインストールします。

次のステップ

- the section called “ACK の概念” - ACK の概念を理解して使用を開始する
- the section called “アクセス許可の設定” - 他の AWS サービスに対して IAM アクセス許可を設定する
- the section called “機能の使用” - ACK 機能リソースを管理する

ACK の概念

ACK は、AWS の実際の状態に照らしてマニフェストの目的の状態を継続的に調整することで、Kubernetes API を利用して AWS リソースを管理します。Kubernetes カスタムリソースを作成または更新すると、ACK は対応する AWS リソースを作成または変更するために必要な AWS API コールを実行し、ドリフトがないかモニタリングして、現在の状態を反映するように Kubernetes ステータスを更新します。このアプローチにより、クラスターと AWS との一貫性を維持しながら、使い慣れた Kubernetes ツールとワークフローを使用してインフラストラクチャを管理できます。

このトピックでは、ACK が Kubernetes API を利用して AWS リソースを管理する際の土台となる概念について説明します。

ACK の使用を開始する

ACK 機能を作成したら (「the section called “ACK 機能を作成する”」を参照)、クラスターで Kubernetes マニフェストを使用して AWS リソースの管理を開始できます。

例えば、`bucket.yaml` でこの S3 バケットマニフェストを作成して、独自のバケット名を選択します。

```
apiVersion: s3.services.k8s.aws/v1alpha1
```

```
kind: Bucket
metadata:
  name: my-test-bucket
  namespace: default
spec:
  name: my-unique-bucket-name-12345
```

マニフェストを適用します。

```
kubectl apply -f bucket.yaml
```

ステータスを確認します。

```
kubectl get bucket my-test-bucket
kubectl describe bucket my-test-bucket
```

AWS にバケットが作成されたことを確認します。

```
aws s3 ls | grep my-unique-bucket-name-12345
```

Kubernetes リソースを削除します。

```
kubectl delete bucket my-test-bucket
```

AWS からバケットが削除されたことを確認します。

```
aws s3 ls | grep my-unique-bucket-name-12345
```

バケットがリストに表示されなくなるはずです。つまり、ACK で AWS リソースのライフサイクル全体を管理できていることになります。

ACK の使用を開始する方法の詳細については、ACK ウェブサイトの「ACK の使用を開始する」を参照してください。

リソースのライフサイクルと調整

ACK は、継続的な調整ループを使用して、AWS リソースが Kubernetes マニフェストに定義された目的の状態と一致するようにします。

調整の仕組み:

1. ユーザーが Kubernetes カスタムリソース (S3 バケットなど) を作成または更新します。
2. ACK がその変更を検出し、目的の状態を AWS の実際の状態と比較します。
3. 両者が異なる場合、ACK は AWS API コールを実行して差分を調整します。
4. ACK が現在の状態を反映するように Kubernetes のリソースステータスを更新します。
5. このループが通常数時間ごとに継続的に繰り返されます。

調整がトリガーされるのは、Kubernetes リソースを新規に作成したとき、既存のリソースの `spec` を更新したとき、または ACK 外での手動変更によって AWS でドリフトが発生したことを ACK が検出したときです。また、ACK は 10 時間という再同期期間で定期的に調整を行います。Kubernetes リソースを変更した場合は即時調整がトリガーされるのに対して、アップストリームの AWS リソースを変更した場合はパッシブなドリフト検出が定期的な再同期で行われます。

上記のように使用を開始すると、ACK は次のステップを実行します。

1. バケットが AWS に存在するかどうかを確認します。
2. 存在しない場合は、`s3:CreateBucket` を呼び出します。
3. バケットの ARN と状態で Kubernetes ステータスを更新します。
4. ドリフトがないか継続的にモニタリングします。

ACK の仕組みの詳細については、ACK ウェブサイトの「ACK の調整」を参照してください。

ステータス条件

ACK リソースは、ステータス条件を使用してリソースの状態を伝えます。ステータス条件を理解すると、問題のトラブルシューティングとリソース状態の把握に役立ちます。

- Ready: リソースを消費する準備ができていることを示します (標準化された Kubernetes 条件)。
- ACK.ResourceSynced: リソース仕様が AWS リソースの状態と一致することを示します。
- ACK.Terminal: 回復不能なエラーが発生したことを示します。
- ACK.Adopted: リソースが新規に作成されたのではなく既存の AWS リソースから採用されたことを示します。
- ACK.Recoverable: 回復可能なエラーであるものの、仕様を更新することなく解決可能であることを示します。
- ACK.Advisory: リソースのアドバイザリ情報を提供します。

- ACK.LateInitialized: フィールドの遅延初期化が完了しているかどうかを示します。
- ACK.ReferencesResolved: すべての AWSResourceReference フィールドが解決されたかどうかを示します。
- ACK.IAMRoleSelected: このリソースを管理するために IAMRoleSelector が選択されたかどうかを示します。

リソースのステータスを確認する:

```
# Check if resource is ready
kubectl get bucket my-bucket -o jsonpath='{.status.conditions[?
(@.type=="Ready")].status}'

# Check for terminal errors
kubectl get bucket my-bucket -o jsonpath='{.status.conditions[?
(@.type=="ACK.Terminal")]}'
```

ステータスの例:

```
status:
  conditions:
  - type: Ready
    status: "True"
    lastTransitionTime: "2024-01-15T10:30:00Z"
  - type: ACK.ResourceSynced
    status: "True"
    lastTransitionTime: "2024-01-15T10:30:00Z"
  - type: ACK.Terminal
    status: "False"
  ackResourceMetadata:
    arn: arn:aws:s3:::my-unique-bucket-name
    ownerAccountID: "111122223333"
    region: us-west-2
```

ACK のステータスと条件の詳細については、ACK ウェブサイトの「ACK の条件」を参照してください。

削除ポリシー

ACK の削除ポリシーは、Kubernetes AWS リソースを削除したときにリソースがどうなるかを制御します。

削除する (デフォルト)

Kubernetes リソースを削除すると、AWS リソースも削除されます。これがデフォルトの動作です。

```
# No annotation needed - this is the default
apiVersion: s3.services.k8s.aws/v1alpha1
kind: Bucket
metadata:
  name: temp-bucket
spec:
  name: temporary-bucket
```

このリソースを削除すると、AWS の S3 バケットも削除されます。

保持する

AWS リソースは、Kubernetes リソースを削除しても保持されます。

```
apiVersion: s3.services.k8s.aws/v1alpha1
kind: Bucket
metadata:
  name: important-bucket
  annotations:
    services.k8s.aws/deletion-policy: "retain"
spec:
  name: production-data-bucket
```

このリソースを削除すると、Kubernetes から削除されますが、S3 バケットは AWS に残ります。

本番稼働用データベースでは、Kubernetes リソース、複数のアプリケーションで使用される共有リソース、誤って削除されてはならない重要なデータがあるリソースを長く維持する必要があります。また、リソースを採用して設定した後、手動管理に戻すという一時的な ACK 管理も必要です。`retain` ポリシーは、こうした用途に有益です。

ACK 削除ポリシーの詳細については、ACK ウェブサイトの「ACK 削除ポリシー」を参照してください。

リソースの採用

リソースを採用すると、既存の AWS リソースを再作成することなく ACK の管理下に置くことができます。

どのような場合に採用するか:

- 既存のインフラストラクチャを ACK 管理に移行する場合
- Kubernetes で AWS リソースを誤って削除したときに孤立したリソースを復旧する場合
- 他のツール (CloudFormation や Terraform) で作成されたリソースをインポートする場合

採用の仕組み:

```
apiVersion: s3.services.k8s.aws/v1alpha1
kind: Bucket
metadata:
  name: existing-bucket
  annotations:
    services.k8s.aws/adoption-policy: "adopt-or-create"
spec:
  name: my-existing-bucket-name
```

このリソースを作成すると、次のようになります。

1. ACK が、その名前のバケットが AWS に存在するかどうかを確認します。
2. 存在すれば、ACK はそのバケットを採用します (API コールは作成されません)。
3. ACK が AWS から現在の設定を読み取ります。
4. ACK が実際の状態を反映するように Kubernetes ステータスを更新します。
5. 今後の更新でリソースを正常に調整します。

採用されたら、リソースは他の ACK リソースと同様に管理され、`retain` 削除ポリシーを使用しない限り、Kubernetes リソースを削除すると、AWS リソースも削除されます。

リソースを採用するときに、AWS リソースは既に存在している必要があり、それを検出するためには ACK に読み取りアクセス許可が必要です。`adopt-or-create` ポリシーは、リソースが存在すれば採用し、存在しなければ作成します。これは、リソースが存在するかどうかにかかわらず宣言型ワークフローを機能させるときに便利です。

ACK リソースの採用の詳細については、ACK ウェブサイトの「ACK リソースの採用」を参照してください。

クロスアカウントおよびクロスリージョンのリソース

ACK は、1 つのクラスターからさまざまな AWS アカウントとリージョンのリソースを管理できます。

クロスリージョンリソースの注釈

注釈を使用すると、AWS リソースのリージョンを指定できます。

```
apiVersion: s3.services.k8s.aws/v1alpha1
kind: Bucket
metadata:
  name: eu-bucket
  annotations:
    services.k8s.aws/region: eu-west-1
spec:
  name: my-eu-bucket
```

特定の名前空間に作成されたすべての AWS リソースのリージョンを指定することもできます。

名前空間の注釈

名前空間内のすべてのリソースに対してデフォルトのリージョンを設定します。

```
apiVersion: v1
kind: Namespace
metadata:
  name: production
  annotations:
    services.k8s.aws/default-region: us-west-2
```

この名前空間に作成されたリソースは、リソースレベルの注釈で上書きされない限り、このリージョンを使用します。

クロスアカウント

IAM ロールセレクターを使用すると、特定の IAM ロールを名前空間にマッピングできます。

```
apiVersion: services.k8s.aws/v1alpha1
kind: IAMRoleSelector
metadata:
  name: target-account-config
spec:
  arn: arn:aws:iam::444455556666:role/ACKTargetAccountRole
  namespaceSelector:
    names:
      - production
```

マッピングされた名前空間に作成されたリソースは、指定されたロールを自動的に使用します。

IAM ロールセレクターの詳細については、「ACK クロスアカウントリソース管理」を参照してください。クロスアカウント設定の詳細については、「the section called “アクセス許可の設定”」を参照してください。

エラー処理と再試行動作

ACK は、一時的なエラーを自動的に処理し、失敗したオペレーションを再試行します。

再試行戦略:

- 一時的なエラー (レート制限、サービスの一時的な問題、不十分なアクセス許可) が発生したら、自動再試行をトリガーします。
- エクスポネンシャルバックオフにより、AWS API が過剰になるのを防ぎます。
- エラータイプに応じて最大再試行回数を変えます。
- 永続的なエラー (無効なパラメータ、リソース名の競合) は再試行されません。

`kubectl describe` を使用して、エラーの詳しい情報がないかリソースのステータスを確認します。

```
kubectl describe bucket my-bucket
```

エラーメッセージでステータス条件を確認し、最近調整を試みたことを示すイベントがないかチェックし、ステータス条件の `message` フィールドで障害に関する説明を参照します。よくあるエラーには、IAM アクセス許可が十分でない、AWS でリソース名が競合している、`spec` に無効な設定値がある、AWS サービスクォータが超過しているといったことがあります。

よくあるエラーのトラブルシューティングについては、「the section called “トラブルシューティング”」を参照してください。

kro によるリソース構成

複数の ACK リソースをまとめて作成および接続するには、EKS Capability for kro (Kube Resource Orchestrator) を使用します。kro を使用すると、リソースのグループを宣言的に定義し、リソース間で設定を渡して、複雑なインフラストラクチャパターンを容易に管理できます。

ACK リソースでカスタムリソース構成を作成する詳細な例については、「the section called “kro の概念”」を参照してください。

次のステップ

- the section called “考慮事項” - EKS に固有のパターンと統合戦略

ACK アクセス許可を設定する

自分の代わりに ACK で AWS リソースを作成および管理できるようにするには、IAM アクセス許可が必要です。このトピックでは、IAM が ACK でどのように機能するかを説明し、さまざまなユースケースに合わせてアクセス許可を設定する際のガイダンスを示します。

IAM は ACK でどのように機能するか

ACK は、IAM ロールを使用して AWS で認証し、リソースに対してアクションを実行します。ACK にアクセス許可を付与する方法が 2 つあります。

機能ロール: ACK 機能の作成時に指定する IAM ロールです。このロールは、デフォルトではすべての ACK オペレーションに使用されます。

IAM ロールセレクター: 特定の名前空間またはリソースにマッピングできる追加の IAM ロールです。これらのロールの範囲内にあるリソースの機能ロールは上書きされます。

リソースを作成または管理する必要があるときに、ACK はどの IAM ロールを使用するかを決定します。

1. IAMRoleSelector がリソースの名前空間に一致するかどうかを確認します。
2. 一致した場合、その IAM ロールを引き受けます。
3. 一致しない場合、機能ロールを使用します。

このアプローチにより、シンプルな単一ロールセットアップから複雑なマルチアカウントかつマルチチームの設定まで、アクセス許可を柔軟に管理できます。

使用開始: シンプルなアクセス許可セットアップ

開発、テスト、またはシンプルなユースケースの場合、必要なすべてのサービスアクセス許可を機能ロールに直接追加できます。

このアプローチは、次の場合に適しています。

- ACK の使用を開始している

- すべてのリソースが同じ AWS アカウントにある
- 1 つのチームがすべての ACK リソースを管理する
- すべての ACK ユーザーに同じアクセス許可が付与されていると確信している

本番稼働でのベストプラクティス: IAM ロールセレクター

本番稼働環境では、IAM ロールセレクターを使用して、最小特権のアクセスと名前空間レベルでの分離を実装します。

IAM ロールセレクターを使用する場合、機能ロールに必要なアクセス許可はサービス固有のロールを引き受けるための `sts:AssumeRole` および `sts:TagSession` アクセス許可だけです。機能ロール自体に AWS サービスアクセス許可 (S3 や RDS など) を追加する必要はありません。そうしたアクセス許可は、機能ロールが引き受ける個々の IAM ロールに付与します。

アクセス許可モデルを選択する:

次の場合は、直接アクセス許可 (機能ロールにサービスアクセス許可を追加すること) を使用します。

- 使用を開始し、最もシンプルなセットアップにする場合
- すべてのリソースがクラスターと同じアカウントにある場合
- クラスター全体の管理アクセス許可を必要とする場合
- すべてのチームが同じアクセス許可を共有できる場合

次の場合は、IAM ロールセレクターを使用します。

- 複数の AWS アカウント全体のリソースを管理する場合
- チームや名前空間ごとに異なるアクセス許可が必要な場合
- 名前空間ごとにきめ細かなアクセスコントロールが必要な場合
- 最小特権のセキュリティプラクティスに従う場合

直接アクセス許可から開始し、その後、要件の増大に応じて IAM ロールセレクターに移行できます。

なぜ本番稼働で IAM ロールセレクターを使用するのか:

- 最小特権: 名前空間ごとに必要なアクセス許可のみを取得します。

- チーム分離: チーム A が誤ってチーム B のアクセス許可を使用することがないようにします。
- 容易な監査: どの名前空間がどのロールを使用するかを明確にマッピングします。
- クロスアカウントサポート: 複数のアカウントのリソースを管理する場合に必要です。
- 関心の分離: サービスや環境ごとに異なるロールを使用します。

基本的な IAM ロールセレクターのセットアップ

ステップ 1: サービス固有の IAM ロールを作成する

特定の AWS サービスに対するアクセス許可を付与した IAM ロールを作成します。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "s3:*"
      ],
      "Resource": "*"
    }
  ]
}
```

機能ロールに引き受けられるように信頼ポリシーを設定します。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "AWS": "arn:aws:iam::111122223333:role/ACKCapabilityRole"
      },
      "Action": ["sts:AssumeRole", "sts:TagSession"]
    }
  ]
}
```

ステップ 2: AssumeRole アクセス許可を機能ロールに付与する

サービス固有のロールを引き受けるためのアクセス許可を機能ロールに追加します。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": ["sts:AssumeRole", "sts:TagSession"],
      "Resource": "arn:aws:iam::111122223333:role/ACK-S3-Role"
    }
  ]
}
```

ステップ 3: IAMRoleSelector を作成する

名前空間に IAM ロールをマッピングします。

```
apiVersion: services.k8s.aws/v1alpha1
kind: IAMRoleSelector
metadata:
  name: s3-namespace-config
spec:
  arn: arn:aws:iam::111122223333:role/ACK-S3-Role
  namespaceSelector:
    names:
      - s3-resources
```

ステップ 4: マッピングした名前空間にリソースを作成する

`s3-resources` 名前空間内のリソースは、指定されたロールを自動的に使用します。

```
apiVersion: s3.services.k8s.aws/v1alpha1
kind: Bucket
metadata:
  name: my-bucket
  namespace: s3-resources
spec:
  name: my-production-bucket
```

マルチアカウント管理

IAM ロールセレクターを使用すると、複数の AWS アカウントのリソースを管理できます。

ステップ 1: クロスアカウント IAM ロールを作成する

ターゲットアカウント (444455556666) に、ソースアカウントの機能ロールを信頼するロールを作成します。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "AWS": "arn:aws:iam::111122223333:role/ACKCapabilityRole"
      },
      "Action": ["sts:AssumeRole", "sts:TagSession"]
    }
  ]
}
```

このロールにサービス固有のアクセス許可をアタッチします。

ステップ 2: AssumeRole アクセス許可を付与する

ソースアカウント (111122223333) で、機能ロールがターゲットアカウントロールを引き受けられるようにします。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": ["sts:AssumeRole", "sts:TagSession"],
      "Resource": "arn:aws:iam::444455556666:role/ACKTargetAccountRole"
    }
  ]
}
```

ステップ 3: IAMRoleSelector を作成する

名前空間にクロスアカウントロールをマッピングします。

```
apiVersion: services.k8s.aws/v1alpha1
kind: IAMRoleSelector
```

```
metadata:
  name: production-account-config
spec:
  arn: arn:aws:iam::444455556666:role/ACKTargetAccountRole
  namespaceSelector:
    names:
      - production
```

ステップ 4: リソースを作成する

production 名前空間内のリソースは、ターゲットアカウントに作成します。

```
apiVersion: s3.services.k8s.aws/v1alpha1
kind: Bucket
metadata:
  name: my-bucket
  namespace: production
spec:
  name: my-cross-account-bucket
```

セッションタグ

EKS ACK 機能は、すべての AWS API リクエストにセッションタグを自動的に設定します。これらのタグは、各リクエストのソースを特定することで、きめ細かなアクセスコントロールと監査を可能にします。

利用可能なセッションタグ

ACK によって行われるすべての AWS API コールには、次のセッションタグが含まれています。

タグキー	説明
eks:eks-capability-arn	リクエストを行う EKS 機能の ARN
eks:kubernetes-namespace	管理対象のリソースの Kubernetes 名前空間
eks:kubernetes-api-group	リソースの Kubernetes API グループ (例: s3.services.k8s.aws)

セッションタグを使用したアクセス制御

これらのセッションタグを IAM ポリシー条件で使用して、ACK が管理できるリソースを制限できます。これにより、名前空間ベースの IAM ロールセレクターに加えて追加セキュリティレイヤーが設定されます。

例: 名前空間による制限

リクエストが production 名前空間から発信された場合にのみ、ACK が S3 バケットを作成できるようにします。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "s3:CreateBucket",
      "Resource": "*",
      "Condition": {
        "StringEquals": {
          "aws:PrincipalTag/eks:kubernetes-namespace": "production"
        }
      }
    }
  ]
}
```

例: 機能による制限

特定の ACK 機能からのアクションのみを許可します。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "s3:*",
      "Resource": "*",
      "Condition": {
        "StringEquals": {
          "aws:PrincipalTag/eks:eks-capability-arn": "arn:aws:eks:us-
west-2:111122223333:capability/my-cluster/ack/my-ack"
        }
```

```
      }
    }
  ]
}
```

> **Note**
>
> セッションタグは、デフォルトでこれらのタグを設定しないセルフマネージド ACK とは異なります。これにより、マネージド機能によるより詳細なアクセスコントロールが可能になります。

高度な IAM ロールセレクターパターン

クロスアカウントロールマッピング、名前空間スコープロールの選択、およびその他の例を含む高度な設定については、ACK ウェブサイトの「きめ細かな IAM ロール」を参照してください。

次のステップ

- the section called “ACK の概念” - ACK の概念およびリソースライフサイクルを理解する
- the section called “EKS 機能に関する考慮事項” - 機能のセキュリティベストプラクティスを理解する

EKS を利用する場合の ACK の考慮事項

このトピックでは、IAM 設定、マルチアカウントパターン、他の EKS 機能との統合など、EKS Capability for ACK を使用する際の重要な考慮事項について説明します。

IAM 設定のパターン

ACK 機能は、IAM 機能ロールを使用して AWS で認証します。要件に基づいて適切な IAM パターンを選択します。

シンプル: 単一の機能ロール

開発、テスト、またはシンプルなユースケースの場合、必要なすべてのアクセス許可を機能ロールに直接付与します。

どのようなときに使うか:

- ACK の使用を開始する
- シングルアカウントでデプロイする
- 1 つのチームがすべてのリソースを管理する
- 開発環境とテスト環境

例: リソースのタグ付け条件に従って、S3 と RDS のアクセス許可を機能ロールに追加します。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": ["s3:*"],
      "Resource": "*",
      "Condition": {
        "StringEquals": {
          "aws:RequestedRegion": ["us-west-2", "us-east-1"]
        }
      }
    },
    {
      "Effect": "Allow",
      "Action": ["rds:*"],
      "Resource": "*",
      "Condition": {
        "StringEquals": {
          "aws:RequestedRegion": ["us-west-2", "us-east-1"],
          "aws:ResourceTag/ManagedBy": "ACK"
        }
      }
    }
  ]
}
```

この例では、S3 と RDS のオペレーションを特定のリージョンに制限し、RDS リソースに ManagedBy: ACK タグが必要です。

本番稼働: IAM ロールセレクター

本番稼働環境では、IAM ロールセレクターを使用して、最小特権のアクセスと名前空間レベルでの分離を実装します。

どのようなときに使うか:

- 本番稼働環境
- マルチチームクラスター
- マルチアカウントリソース管理
- 最小特権セキュリティ要件
- サービスごとに異なるアクセス許可が必要

利点:

- 名前空間ごとに必要なアクセス許可のみを取得できます。
- チーム分離 - チーム A は、チーム B のアクセス許可を使用できません。
- 監査とコンプライアンスが容易です。
- クロスアカウントリソース管理に必須です。

IAM ロールセレクターの設定の詳細については、「the section called “アクセス許可の設定”」を参照してください。

他の EKS 機能との統合

GitOps と Argo CD

EKS Capability for Argo CD を使用すると、Git リポジトリから ACK リソースをデプロイして、インフラストラクチャの管理に GitOps ワークフローを使用できます。

考慮事項:

- エンドツーエンドの GitOps のアプリケーションマニフェストと共に ACK リソースを保存する
- チーム構造に基づいて環境、サービス、またはリソースタイプ別に整理する
- Argo CD の自動同期を使用して継続的に調整を行う
- プルーニングを有効にして削除済みのリソースを自動的に除去する
- マルチクラスターのインフラストラクチャ管理にハブアンドスポークパターンを検討する

GitOps は、監査証跡、ロールバック機能、および宣言型のインフラストラクチャ管理を備えています。Argo CD の詳細については、「the section called “Argo CD の使用”」を参照してください。

kro によるリソース構成

EKS Capability for kro (Kube Resource Orchestrator) を使用すると、複数の ACK リソースを高次の抽象化とカスタム API に構成できます。

どのような場合に ACK で kro を使用するか:

- よく見られるインフラストラクチャスタック用に再利用可能なパターンを作成する場合 (データベース + バックアップ + モニタリング)
- アプリケーションチーム向けに API をシンプルにしてセルフサービスプラットフォームを構築する場合
- リソースの依存関係を管理し、リソース間で値を渡す (S3 バケット ARN を Lambda 関数に渡す) 場合
- チーム間でインフラストラクチャ設定を標準化する場合
- カスタムリソースの背後に実装の詳細を隠して複雑さを軽減する場合

パターンの例:

- アプリケーションスタック: S3 バケット + SQS キュー + 通知設定
- データベースセットアップ: RDS インスタンス + パラメータグループ + セキュリティグループ + シークレット
- ネットワーク: VPC + サブネット + ルートテーブル + セキュリティグループ

kro は、構成済みのリソースに対して依存関係の順序付け、ステータスの伝播、ライフサイクルの管理を行います。kro の詳細については、「the section called “kro の概念”」を参照してください。

リソースを整理する

Kubernetes 名前空間と AWS リソースタグを使用して ACK リソースを整理すると、管理、アクセスコントロール、コスト追跡を改善できます。

名前空間の整理

Kubernetes 名前空間を使用すると、環境 (本番稼働、ステージング、開発)、チーム (プラットフォーム、データ、ml)、またはアプリケーション別に ACK リソースを論理的に分離できます。

利点:

- アクセスコントロール用に RBAC を名前空間範囲で限定する
- 注釈を使用して名前空間ごとにデフォルトリージョンを設定する
- リソース管理とクリーンアップが容易になる
- 組織構造に沿って論理的に分離できる

リソースのタグ付け

EKS ACK 機能は、作成するすべての AWS リソースにデフォルトのタグを自動的に適用します。これらのタグはセルフマネージド ACK とは異なり、トレーサビリティが強化されています。

この機能によって適用されるデフォルトのタグ:

タグキー	説明
`eks:controller-version`	ACK コントローラーのバージョン
`eks:kubernetes-namespace`	ACK リソースの Kubernetes 名前空間
`eks:kubernetes-resource-name`	Kubernetes リソースの名前
`eks:kubernetes-api-group`	Kubernetes API グループ (例: `s3.services.k8s.aws`)
`eks:eks-capability-arn`	EKS ACK 機能の ARN

> **Note**
>
> セルフマネージド ACK は、`services.k8s.aws/controller-version` や `services.k8s.aws/namespace` といった異なるデフォルトタグを使用します。この機能のタグは、他の EKS 機能との一貫性を保つために `eks:` プレフィックスを使用します。

その他の推奨タグ:

コスト配分、所有権追跡、および整理目的でカスタムタグを追加します。

- 環境 (本番稼働、ステージング、開発)
- チームまたは部門の所有権
- 請求を配分するためのコストセンター
- アプリケーション名やサービス名

他の Infrastructure as Code ツールからの移行

多くの組織が、Kubernetes を土台にワークロードのオーケストレーションを超えた標準化を図ることに価値を見出しています。インフラストラクチャと AWS リソースの管理を ACK に移行することで、アプリケーションワークロードと共に Kubernetes API を使用してインフラストラクチャ管理を標準化できます。

Kubernetes を土台にインフラストラクチャを標準化するメリット:

- 信頼できる単一のソース: Kubernetes でアプリケーションとインフラストラクチャの両方を管理して、エンドツーエンドの GitOps プラクティスを実現する
- 統一されたツール: チームは Kubernetes のリソースとツールを使用すればよく、いくつものツールやフレームワークを学習する必要がない
- 一貫性のある調整: ACK は、Kubernetes がワークロードに対してそうであるように、AWS リソースを継続的に調整し、必須のツールと比較してドリフトを検出して修正する
- ネイティブコンポジション: kro と ACK を一緒に使用して、アプリケーションとリソースのマニフェストで AWS リソースを直接参照し、リソース間で接続文字列と ARN を渡す
- シンプルなオペレーション: 1 つのコントロールプレーンでシステム全体のデプロイ、ロールバック、オブザーバビリティを実現する

ACK は、AWS リソースを再作成せずに既存のものを採用するため、CloudFormation、Terraform、またはクラスター外のリソースからダウンタイムなしで移行できます。

既存のリソースを採用する:

```
apiVersion: s3.services.k8s.aws/v1alpha1
kind: Bucket
metadata:
  name: existing-bucket
  annotations:
```

```
    services.k8s.aws/adoption-policy: "adopt-or-create"
spec:
  name: my-existing-bucket-name
```

採用されたリソースは、ACK によって管理され、Kubernetes マニフェストを介して更新できます。段階的に移行できるため、必要に応じてリソースを採用しながら、他のリソースでは既存の IaC ツールを維持できます。

また、ACK は読み取り専用リソースもサポートしています。他のチームやツールで管理されるリソースを変更するのではなく参照するだけにする場合は、採用と `retain` 削除ポリシーとを組み合わせて、読み取り IAM アクセス許可のみを付与します。これにより、アプリケーションは変更のリスクを冒すことなく Kubernetes API を介して共有インフラストラクチャ (VPC、IAM ロール、KMS キー) を検出できます。

リソースの採用の詳細については、「the section called “ACK の概念”」を参照してください。

削除ポリシー

削除ポリシーは、対応する Kubernetes リソースを削除したときに AWS リソースがどうなるかを制御します。リソースのライフサイクルと実際の運用要件に基づいて、適切なポリシーを選択します。

削除する (デフォルト)

Kubernetes リソースを削除すると、AWS リソースも削除されます。これにより、クラスターと AWS 間の整合性が維持され、リソースの蓄積が防止されます。

どのような場合に削除を使用するか:

- 開発環境とテスト環境でクリーンアップが重要な場合
- エフェメラルリソースがアプリケーションのライフサイクルに結びついている場合 (テストデータベースや一時バケット)
- リソースがアプリケーションよりも長く存続してはいけない場合 (SQS キューや ElastiCache クラスター)
- コスト最適化 - 未使用のリソースを自動的にクリーンアップする場合
- GitOps で管理された環境で、Git からリソースが削除されたらインフラストラクチャも削除される場合

Kubernetes の宣言型モデルに合わせてデフォルトの削除ポリシーを調整します。クラスターの内容は AWS に存在するものと一致します。

保持する

Kubernetes リソースを削除しても、AWS リソースは保持されます。これにより、重要なデータが保護され、リソースは Kubernetes 表現よりも長く存続できます。

どのような場合に保持を使用するか:

- 本番稼働用データベースにクラスターの変更後も存続する必要がある重要なデータが保存されている場合
- 長期ストレージバケットにコンプライアンス要件や監査要件がある場合
- 複数のアプリケーションやチームが共有リソースを使用している場合
- さまざまな管理ツールにリソースを移行する場合
- ディザスタリカバリでインフラストラクチャを維持する場合
- 廃止に慎重を要する複雑な依存関係がリソースにある場合

```
apiVersion: rds.services.k8s.aws/v1alpha1
kind: DBInstance
metadata:
  name: production-db
  annotations:
    services.k8s.aws/deletion-policy: "retain"
spec:
  dbInstanceIdentifier: prod-db
  # ... configuration
```

> **Important**
>
> 保持したリソースは、引き続き AWS コストが発生するため、不要になったら AWS から手動で削除する必要があります。リソースのタグ付けを使用すると、保持したリソースの中にクリーンアップが必要なものがないかを追跡できます。

削除ポリシーの詳細については、「the section called “ACK の概念”」を参照してください。

アップストリームドキュメント

ACK の使用の詳細については、ACK ウェブサイトの以下のページを参照してください:

- リソースの作成および管理については、「ACK 使用ガイド」を参照してください。

- すべてのサービスの詳細な API ドキュメントについては、「ACK API リファレンス」を参照してください。
- 包括的なユーザードキュメントについては、「ACK ドキュメント」を参照してください。

次のステップ

- the section called “アクセス許可の設定” - IAM アクセス許可およびマルチアカウントパターンを設定する
- the section called “ACK の概念” - ACK の概念およびリソースライフサイクルを理解する
- the section called “トラブルシューティング” - ACK の問題をトラブルシューティングする
- the section called “Argo CD の使用” - GitOps で ACK リソースをデプロイする
- the section called “kro の概念” - ACK リソースを高次の抽象化に構成する

ACK 機能に関する問題をトラブルシューティングする

> **Note**
>
> EKS の機能は完全に管理され、クラスターの外部で実行されます。コントローラーの名前空間に直接アクセスすることはできません。コントローラーの動作の可視化のためにコントローラーのログ配信を設定できます。「the section called “機能コントローラーログ”」を参照してください。トラブルシューティングでは、機能のヘルス、リソースのステータス、IAM の設定に焦点を当てています。

機能が ACTIVE であるのにリソースが作成されない

ACK 機能のステータスが `ACTIVE` であるのに AWS にリソースが作成されない場合は、機能のヘルス、リソースのステータス、IAM アクセス許可を確認してください。

機能のヘルスを確認する:

機能のヘルスとステータスの問題は、EKS コンソールまたは AWS CLI を使用して表示できます。

コンソール:

1. https://console.aws.amazon.com/eks/home#/clusters で Amazon EKS コンソールを開きます。
2. クラスター名を選択します。

3. [オブザーバビリティ] タブを選択します。
4. [クラスターを監視する] を選択します。
5. [機能] タブを選択すると、すべての機能のヘルスとステータスが表示されます。

AWSCLI:

```
# View capability status and health
aws eks describe-capability \
  --region region-code \
  --cluster-name my-cluster \
  --capability-name my-ack

# Look for issues in the health section
```

一般的な原因:

- IAM アクセス許可がない: 機能ロールに AWS サービスのアクセス許可が付与されていません。
- 名前空間が間違っている: IAMRoleSelector が適切でない名前空間にリソースが作成されました。
- リソース仕様が無効である: リソースのステータス条件に検証エラーがないか確認してください。
- API スロットリング: AWS API レートリミットに達しています。
- アドミッションウェブフック: アドミッションウェブフックのために、コントローラーがリソースステータスにパッチを適用できません。

リソースのステータスを確認する:

```
# Describe the resource to see conditions and events
kubectl describe bucket my-bucket -n default

# Look for status conditions
kubectl get bucket my-bucket -n default -o jsonpath='{.status.conditions}'

# View resource events
kubectl get events --field-selector involvedObject.name=my-bucket -n default
```

IAM アクセス許可を確認する:

```
# View the Capability Role's policies
aws iam list-attached-role-policies --role-name my-ack-capability-role
```

```
aws iam list-role-policies --role-name my-ack-capability-role

# Get specific policy details
aws iam get-role-policy --role-name my-ack-capability-role --policy-name policy-name
```

AWS にリソースが作成されているのに Kubernetes に表示されない

ACK で追跡できるリソースは、Kubernetes マニフェストを介して作成されたものだけです。ACK で既存の AWS リソースを管理するには、採用機能を使用します。

```
apiVersion: s3.services.k8s.aws/v1alpha1
kind: Bucket
metadata:
  name: existing-bucket
  annotations:
    services.k8s.aws/adoption-policy: "adopt-or-create"
spec:
  name: my-existing-bucket-name
```

リソースの採用の詳細については、「the section called “ACK の概念”」を参照してください。

クロスアカウントリソースが作成されない

IAM ロールセレクタの使用時にターゲット AWS アカウントにリソースが作成されない場合は、信頼関係と IAMRoleSelector 設定を確認します。

信頼関係を確認する:

```
# Check the trust policy in the target account role
aws iam get-role --role-name cross-account-ack-role --query
 'Role.AssumeRolePolicyDocument'
```

信頼ポリシーは、ソースアカウントの機能ロールに引き受けられるようになっている必要があります。

IAMRoleSelector 設定を確認する:

```
# List IAMRoleSelectors (cluster-scoped)
kubectl get iamroleselector

# Describe specific selector
```

```
kubectl describe iamroleselector my-selector
```

名前空間の配置を確認する:

IAMRoleSelectors は、クラスター範囲のリソースですが、特定の名前空間をターゲットとします。ACK リソースが IAMRoleSelector の名前空間セレクターに一致する名前空間にあることを確認してください。

```
# Check resource namespace
kubectl get bucket my-cross-account-bucket -n production

# List all IAMRoleSelectors (cluster-scoped)
kubectl get iamroleselector

# Check which namespace the selector targets
kubectl get iamroleselector my-selector -o jsonpath='{.spec.namespaceSelector}'
```

IAMRoleSelected 条件を確認する:

ACK.IAMRoleSelected 条件を調べて、IAMRoleSelector がリソースに正常に一致したものであることを確認します。

```
# Check if IAMRoleSelector was matched
kubectl get bucket my-cross-account-bucket -n production -o
 jsonpath='{.status.conditions[?(@.type=="ACK.IAMRoleSelected")]}'
```

条件が False であるか存在しない場合は、IAMRoleSelector の名前空間セレクタがリソースの名前空間と一致しません。セレクターの namespaceSelector がリソースの名前空間ラベルと一致していることを確認します。

機能ロールのアクセス許可を確認する:

機能ロールには、ターゲットアカウントロールの sts:AssumeRole および sts:TagSession のアクセス許可が必要です。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": ["sts:AssumeRole", "sts:TagSession"],
```

```
      "Resource": "arn:aws:iam::444455556666:role/cross-account-ack-role"
    }
  ]
}
```

クロスアカウント設定の詳細については、「the section called “アクセス許可の設定”」を参照してください。

次のステップ

- the section called “考慮事項” - ACK の考慮事項とベストプラクティス
- the section called “アクセス許可の設定” - IAM アクセス許可およびマルチアカウントパターンを設定する
- the section called “ACK の概念” - ACK の概念およびリソースライフサイクルを理解する
- the section called “機能をトラブルシューティングする” - 一般的な機能をトラブルシューティングする際のガイダンス

EKS Capability for ACK とセルフマネージド ACK を比較する

EKS Capability for ACK が提供する機能はセルフマネージド ACK コントローラーと同じですが、その操作性に大きなメリットがあります。EKS 機能とセルフマネージドソリューションの全般的な比較については、「the section called “考慮事項”」を参照してください。このトピックでは、ACK に固有の違いに焦点を当てます。

アップストリーム ACK との違い

EKS Capability for ACK は、アップストリーム ACK コントローラーに基づいていますが、IAM との統合に違いがあります。

IAM 機能ロール: この機能は、信頼ポリシーが付与された専用の IAM ロールを使用して `capabilities.eks.amazonaws.com` サービスプリンシパルを許可します。IRSA (サービスアカウントの IAM ロール) は使用しません。IAM ポリシーを直接アタッチできるため、Kubernetes サービスアカウントを作成して注釈を付ける必要も、OIDC プロバイダーを設定する必要もありません。本番稼働のユースケースでは、`IAMRoleSelector` を使用してサービスアクセス許可を設定するのがベストプラクティスです。詳細については、「the section called “アクセス許可の設定”」を参照してください。

セッションタグ: マネージド機能は、すべての AWS API リクエストにセッションタグを自動的に設定し、きめ細かなアクセスコントロールと監査を可能にします。タグには、`eks:eks-`

capability-arn、eks:kubernetes-namespace、および eks:kubernetes-api-group が含まれます。これは、これらのタグをデフォルトで設定しないセルフマネージド ACK とは異なります。IAM ポリシーでのセッションタグの使用の詳細については、「the section called “アクセス許可の設定”」を参照してください。

リソースタグ: この機能は、セルフマネージド ACK とは異なるデフォルトタグを AWS リソースに適用します。この機能では、セルフマネージド ACK で使用される services.k8s.aws/ タグの代わりに eks: プレフィックス付きタグ (eks:kubernetes-namespace、eks:eks-capability-arn など) を使用します。デフォルトのリソースタグの完全なリストについては、「the section called “考慮事項”」を参照してください。

リソースの互換性: ACK カスタムリソースは、アップストリーム ACK と同じように動作するため、ACK リソース YAML ファイルを変更する必要はありません。この機能は同じ Kubernetes API と CRD を使用するため、kubectl のようなツールは同じように機能します。この機能は、アップストリーム ACK で一般利用可能な (GA) コントローラーとリソースのみをサポートします。この機能には、アップストリームでプレビューされているコントローラーは含まれません。コントローラーのステータスは、時間の経過とともにプレビューから GA アップストリームに変更され、この機能は自動的に管理を開始できます。この機能とともにセルフマネージドプレビューコントローラーを実行する場合は、移行する前に the section called “プレビューコントローラーと自動昇格” をレビューしてください。

ACK ドキュメントとサービス固有のガイドの詳細については、ACK ウェブサイトの「ACK ドキュメント」を参照してください。

移行パス

AWS リソースの中断を最小限に抑えながら、セルフマネージド ACK からマネージド機能に移行できます。移行は Kubernetes リーダー選出に依存します。セルフマネージドコントローラーと機能は同じリースに対して競合するため、そのうちの 1 つだけがいつでも特定のリソースを調整します。これを機能させるには、両方が同じ名前空間でリースを共有する必要があります。この機能では、実行中のセルフマネージドコントローラーからリースを強制的に取得しないため、セルフマネージドコントローラーをスケールダウンして、ハンドオーバーが発生するタイミングを制御できます。

> **Important**
>
> 開始する前に、セルフマネージドコントローラーが現在使用しているアクセス許可と同等なアクセス許可を IAM 機能ロールに付与します。この機能は、セルフマネージドコントローラーが IRSA や EKS Pod Identity などの現在使用しているメカニズムを通じてではなく、capabilities.eks.amazonaws.com サービスプリンシパルを通じて専用の機能ロー

ルで認証します (the section called “アクセス許可の設定” を参照)。機能ロールにアクセス許可がない場合、この機能はリソースを採用します。その後、調整に失敗し、AccessDenied エラーがログに記録されます。

移行するには、以下のステップを実行します。ステップでは、例として S3 コントローラー (ack-s3-controller) を使用します。機能に移行するセルフマネージド ACK コントローラーごとにそれらを繰り返し、対応するコントローラー名と Helm チャートを置き換えます。

> **Note**
>
> 機能とともにコントローラーをセルフマネージドで実行することは、長期的な設定ではなく、移行中の一時的な状態を想定しています。両方が実行されている間、どちらか一方の中断 (機能のデプロイやセルフマネージドコントローラーのアップグレードなど) によってリースが解放され、もう一方の側でリースが取得され、調整がそれらの間で予期せず切り替わる可能性があります。機能とともに無期限にセルフマネージドで実行するのではなく、各コントローラーの移行を完了してください。

1. セルフマネージド ACK コントローラーでリーダー選出を有効にし、リースを kube-system に移動します:

```
helm upgrade --install ack-s3-controller \
  oci://public.ecr.aws/aws-controllers-k8s/s3-chart \
  --namespace ack-system \
  --set leaderElection.enabled=true \
  --set leaderElection.namespace=kube-system
```

両方の値を設定する必要があります。ACK Helm チャートでは、--leader-election-namespace フラグは leaderElection.enabled が true の場合にのみ適用され、リーダー選出はデフォルトで無効になっています。leaderElection.namespace を設定するのみでは、一切影響を与えません。コントローラーはリースなしで実行され続け、両方のコントローラーは機能の作成後に同じリソースを同時に調整します。これは、S3 だけでなく、すべての ACK サービスコントローラーチャートに適用されます。

これにより、コントローラーのリースが kube-system に移動するため、マネージド機能がコントローラーを調整できるようになります。

2. クラスターに ACK 機能を作成します (the section called “ACK 機能を作成する” を参照)。この機能は起動してリースに対して競合しますが、セルフマネージドコントローラーはリースを保持し続けます。セルフマネージドコントローラーはリソースの調整を継続し、この機能はテイクオーバーを適用するのではなくリーダーシップを待機します。
3. 移行を開始する準備ができたら、セルフマネージドコントローラーをゼロレプリカにスケールダウンします。これによりリースが解放されるので、この機能がリーダーシップを取得して調整を引き継ぐことができます。

   ```
   kubectl scale deployment ack-s3-controller \
     --namespace ack-system --replicas=0
   ```

   セルフマネージドコントローラーをスケールダウンすると、この機能はリースを取得し、通常は短時間で調整を開始します。セルフマネージドコントローラーをスケールアップしても、この機能がリースを保持および更新し続けるため、リースは返されません。セルフマネージドコントローラーに調整を返すには、少なくとも 1 つのレプリカにスケールアップしてから ACK 機能を削除します。この機能を削除すると、セルフマネージドコントローラーはリースを再取得し、調整を再開します。

   採用時に、この機能はセルフマネージド ACK で使用される `services.k8s.aws/` タグの代わりに、独自のデフォルトリソースタグ (`eks:` プレフィックス付き) を適用します (the section called “考慮事項” を参照)。採用したリソースに対するタグ付け API コールの 1 回限りのセットを想定し、`services.k8s.aws/` タグプレフィックスをキーとするコスト配分またはポリシーツールを更新します。
4. この機能が正常であり、リソースの調整を引き継いでいることを確認します。リソースが `True` の `Synced` 条件をレポートし、この機能が `AccessDenied` エラーをログに記録していないことを確認します。
5. この機能が正しくリソースを管理していることを確認したら、セルフマネージドコントローラーを削除します。

   ```
   helm uninstall ack-s3-controller --namespace ack-system
   ```

このアプローチにより、移行中、両方のコントローラーを安全に共存させることができます。マネージド機能は、リースを解放した後に、これまでセルフマネージドコントローラーで管理されていたリソース採用するため、引き続き競合なく調整を行うことができます。

プレビューコントローラーと自動昇格

この機能は、アップストリーム ACK で GA であるコントローラーのみをサポートします。現在プレビュー中のコントローラーは、後で GA アップストリームに昇格できます。それが発生した場合、この機能はユーザー側のアクションなしで、そのコントローラーの管理を自動的に開始します。

これにより、他のコントローラーにもこの機能を使用しているクラスターでセルフマネージドプレビューコントローラーを実行すると、リスクが発生します。プレビューコントローラーは、調整する他のコントローラーがないため、リーダー選出が無効になっている単一のレプリカとして実行できます。そのコントローラーが GA に昇格すると、この機能はそのコントローラーの管理を開始します。この時点で、2 つのリコンシラーは同じリソースに対して動作し、それらを調整するための共有リースはありません。その結果、移行ステップが防止するように設計されているのと同じ二重調整の競合が発生します。どちらのリコンシラーも競合する AWS API コールを発行し、相反する更新をカスタムリソースステータスに書き込みます。

これを回避するには、この機能とともにセルフマネージドプレビューコントローラーを実行する前に、次の操作を行います:

- セルフマネージドプレビューコントローラーでリーダー選出を有効にし、移行ステップに示されているのと同じ `leaderElection.enabled=true` および `leaderElection.namespace=kube-system` 設定を使用して、リースを `kube-system` にポイントします。これにより、コントローラーが昇格され、この機能がそれを引き継ぐ場合、2 つは並行して調整するのではなく、共有リースを介して調整されます。
- 依存するプレビューコントローラーのアップストリーム GA ステータスを追跡し、昇格時に移行パスに従ってそれらを移行する計画を立てます。各コントローラーの現在のステータスは、ACK ウェブサイトの 「ACK サービスページ」で確認できます。

次のステップ

- the section called “ACK 機能を作成する” - ACK の機能リソースを作成する
- the section called “ACK の概念” - ACK の概念およびリソースライフサイクルを理解する
- the section called “アクセス許可の設定” - IAM およびアクセス許可を設定する

Argo CD による継続的デプロイ

Argo CD は、Kubernetes 向けの宣言型 GitOps 継続的デリバリーツールです。Argo CD を使用すると、複数のクラスターと環境にわたるアプリケーションのデプロイとライフサイクルを自動的に管理

できます。Argo CD は、Git リポジトリ、Helm レジストリ (HTTP および OCI)、OCI イメージなど複数のソースタイプをサポートしており、セキュリティとコンプライアンスの要件がさまざまに異なる組織に柔軟に対応できます。

AWS では EKS の機能を使用して Argo CD を完全に管理できるため、クラスターに Argo CD コントローラーとその依存関係をインストールしてメンテナンスおよびスケールする必要がありません。

Argo CD の仕組み

Argo CD は GitOps パターンに従い、アプリケーションソース (Git リポジトリ、Helm レジストリ、または OCI イメージ) を、目的のアプリケーション状態を定義するための信頼できるソースとします。Argo CD `Application` リソースを作成するときは、アプリケーションマニフェストが含まれているソースと、ターゲットとなる Kubernetes クラスターおよび名前空間を指定します。Argo CD は、クラスター内のソース状態とライブ状態の両方を継続的にモニタリングし、変更が加えられたら自動的に同期して、クラスターの状態が目的の状態と一致するようにします。

> **Note**
>
> EKS Capability for Argo CD では、Argo CD ソフトウェアはワーカーノードではなく AWS コントロールプレーンで動作します。つまり、ワーカーノードが Git リポジトリや Helm レジストリに直接アクセスする必要はありません。この機能は、AWS アカウントからのソースアクセスを処理します。

Argo CD には、主に 3 つのリソースタイプがあります。

- Application: Git リポジトリからターゲットクラスターへのデプロイを定義します。
- ApplicationSet: 複数のクラスターにデプロイできるように、テンプレートから複数の Application を生成します
- AppProject: Application を論理的にグループ化し、アクセスコントロールを付与できます。

例: Argo CD Application を作成する

次の例は、Argo CD `Application` リソースを作成する方法を示しています。

```
apiVersion: argoproj.io/v1alpha1
kind: Application
metadata:
```

```
  name: guestbook
  namespace: argocd
spec:
  project: default
  source:
    repoURL: https://github.com/argoproj/argocd-example-apps.git
    targetRevision: HEAD
    path: guestbook
  destination:
    name: in-cluster
    namespace: guestbook
  syncPolicy:
    automated:
      prune: true
      selfHeal: true
```

> **Note**
>
> クラスターの登録時に使用したクラスター名で destination.name を使用します (ローカルクラスターの in-cluster の場合など)。destination.server フィールドは EKS クラスター ARN でも機能しますが、読みやすくするためにクラスター名を使用することをお勧めします。

Argo CD のメリット

GitOps ワークフローを実装しているため、Git リポジトリにアプリケーション設定を定義できます。また、目的の状態に合わせてアプリケーションを自動的に同期する機能も備えています。この Git 中心のアプローチにより、すべての変更が漏れなく監査証跡に残され、ロールバックが容易になり、既存のコードレビューおよび承認プロセスと必然的に統合されます。Argo CD は、Git の目的の状態とクラスターの実際の状態との間のドリフトを自動的に検出して調整し、デプロイと宣言された設定との整合性が確保されるようにします。

Argo CD を使用すると、単一の Argo CD インスタンスから複数のクラスターにアプリケーションをデプロイして管理できるため、マルチクラスター環境とマルチリージョン環境でのオペレーションがシンプルになります。Argo CD UI に可視化とモニタリングの機能が用意されているため、アプリケーションのデプロイステータス、ヘルス、履歴を確認できます。この UI は AWS アイデンティティセンター (以前の AWS SSO) と統合されており、認証と認可をシームレスに進めて、既存の ID 管理インフラストラクチャを使用してアクセスを制御できます。

AWS では、Argo CD が EKS マネージド機能の一部として完全に管理されるため、Argo CD インフラストラクチャを導入、設定、メンテナンスする必要がありません。AWS がスケール、パッチ適用、運用管理を行うので、チームはツールのメンテナンスではなくアプリケーションの配信に集中できます。

AWS アイデンティティセンターとの統合

EKS マネージド機能により、Argo CD と AWS アイデンティティセンターを直接統合して、ユーザーの認証と認可をシームレスに進めることができます。Argo CD 機能を有効にすると、アイデンティティセンターのグループとユーザーを Argo CD RBAC ロールにマッピングするように AWS アイデンティティセンターの統合を設定して、Argo CD で誰がアプリケーションにアクセスおよび管理できるかを制御できます。

他の EKS マネージド機能との統合

Argo CD は、他の EKS マネージド機能と統合されています。

- Kubernetes 用 AWS コントローラー (ACK): Argo CD を使用すると、複数のクラスターにまたがる ACK リソースのデプロイを管理して、AWS インフラストラクチャの GitOps ワークフローを有効にできます。
- kro (Kube Resource Orchestrator): Argo CD を使用すると、複数のクラスターに kro コンポジションをデプロイして、Kubernetes エステート全体で一貫性のあるリソース構成を実現できます。

Argo CD の使用を開始する

EKS Capability for Argo CD の使用を開始するには:

1. IAM 機能ロールを作成し、Argo CD がソースにアクセスしてアプリケーションを管理できるように必要なアクセス許可を設定します。
2. AWS コンソール、AWS CLI、または任意の Infrastructure as Code ツールを使用して、EKS クラスターに Argo CD 機能リソースを作成します。
3. アプリケーションをデプロイするために、リポジトリアクセスを設定し、クラスターを登録します。
4. Application リソースを作成して、宣言型ソースからアプリケーションをデプロイします。

Argo CD 機能を作成する

このトピックでは、Amazon EKS クラスターに Argo CD 機能を作成する方法について説明します。

前提条件

Argo CD 機能を作成する前に、以下の点を確認してください。

- サポート対象の Kubernetes バージョン (標準サポートと延長サポートのすべてのバージョンがサポート対象) を実行している Amazon EKS クラスターが存在する
- AWS アイデンティティセンターが設定されている - Argo CD の認証に必要 (ローカルユーザーはサポート対象外)
- Argo CD に対するアクセス許可が付与された IAM 機能ロールがある
- EKS クラスターに機能リソースを作成するための十分な IAM アクセス許可がある
- クラスターと通信できるように `kubectl` が設定されていること
- (オプション) Argo CD CLI をインストールしてクラスターとリポジトリの管理を容易にしている
- (CLI/eksctl の場合) 適切な CLI ツールがインストールおよび設定されている

IAM 機能ロールを作成する手順については、「the section called “機能 IAM ロール”」を参照してください。アイデンティティセンターのセットアップについては、「AWS アイデンティティセンターの使用を開始する」を参照してください。

> **Important**
>
> Argo CD でどの AWS リソースにアクセスできるかは、どの IAM 機能ロールを使用するかによって決まります。例えば、Git リポジトリへのアクセスには CodeConnections を利用し、シークレットへのアクセスには Secrets Manager を使用します。最小特権のアクセス許可を付与した適切なロールを作成する際のガイダンスについては、「the section called “機能 IAM ロール”」および「the section called “EKS 機能に関する考慮事項”」を参照してください。

ツールを選択する

AWS マネジメントコンソール、AWS CLI、または eksctl を使用して、Argo CD 機能を作成できます。

- the section called “コンソール” - 指示に従って操作する場合はコンソールを使用します。

- the section called “ AWS CLI” - スクリプトを作成して自動化を図る場合は AWS CLI を使用します。
- the section called “eksctl” - Kubernetes ネイティブの操作をする場合は eksctl を使用します。

Argo CD 機能を作成するとどうなるか

Argo CD 機能を作成すると、次のようになります。

1. EKS が AWS コントロールプレーンに Argo CD 機能サービスを作成します。
2. カスタムリソース定義 (CRD) がクラスターにインストールされます。
3. Kubernetes のベースラインアクセス許可を付与する機能固有のアクセスエントリポリシーを使用して、IAM 機能ロールのアクセスエントリを自動的に作成します (「the section called “EKS 機能に関する考慮事項”」を参照)。
4. Argo CD がカスタムリソース (Application、ApplicationSet、AppProject) の監視を開始します。
5. 機能のステータスが `CREATING` から `ACTIVE` に変わります。
6. Argo CD UI が URL 経由でアクセス可能になります。

機能がアクティブになったら、宣言型ソースからデプロイするようにクラスターに Argo CD Application を作成できます。

> **Note**
>
> 自動的に作成されるアクセスエントリは、アプリケーションをクラスターにデプロイするアクセス許可を付与しません。アプリケーションをデプロイするには、ターゲットクラスターごとに追加の Kubernetes RBAC アクセス許可を設定する必要があります。クラスターの登録とアクセスの設定の詳細については、「the section called “クラスターを登録する”」を参照してください。

次のステップ

Argo CD 機能の作成後:

- the section called “Argo CD の概念” - GitOps の原則、同期ポリシー、マルチクラスターパターンについて学ぶ

- the section called “Argo CD の使用” - リポジトリアクセスを設定し、ターゲットクラスターを登録して、Application を作成する
- the section called “Argo CD に関する考慮事項” - マルチクラスターアーキテクチャパターンと高度な設定を探る

コンソールを使用して Argo CD 機能を作成する

AWS マネジメントコンソールを使用して、Amazon EKS クラスターで Argo CD 機能を作成します。

前提条件

- AWS アイデンティティセンターが設定されていること – Argo CD での認証には AWS アイデンティティセンターが必要です。ローカルユーザーはサポートされていません。AWS アイデンティティセンターをセットアップしていない場合は、「AWS アイデンティティセンターの使用を開始する」を参照してアイデンティティセンターインスタンスを作成し、「ユーザーを追加する」と「グループを追加する」を参照して Argo CD へのアクセスに必要なユーザーとグループを作成します。
- AWS アイデンティティセンターのユーザーまたはグループが少なくとも 1 つ – Argo CD の RBAC ロールマッピングを割り当て、Argo CD UI へのアクセスを提供するには、アイデンティティセンターインスタンスに少なくとも 1 人のユーザーまたは 1 つのグループを設定する必要があります。

Argo CD 機能を作成する

1. https://console.aws.amazon.com/eks/home#/clusters で Amazon EKS コンソールを開きます。
2. クラスター名を選択して、クラスターの詳細ページを開きます。
3. [機能] タブを選択します。
4. 左のナビゲーションで、[Argo CD] を選択します。
5. [Argo CD 機能を作成] を選択します。
6. IAM 機能ロールに対して以下の操作を行います。
   - IAM 機能ロールが既に存在する場合は、ドロップダウンから選択します。
   - ロールを作成する必要がある場合は、[Argo CD ロールを作成] を選択します。

     IAM コンソールが新しいタブに開かれます。信頼ポリシーには値があらかじめ入力され、Secrets Manager にはフル読み取りアクセスが付与されています。デフォルトでは他にア

クセス許可は追加されませんが、必要に応じて追加することもできます。CodeCommit リポジトリや他の AWS サービスを使用する場合は、ロールを作成する前に適切なアクセス許可を追加します。

ロールを作成して EKS コンソールに戻ると、そのロールが自動的に選択されます。

> **Note**
>
> AWS Secrets Manager または AWS CodeConnections とをオプションで統合する場合は、ロールにアクセス許可を追加する必要があります。IAM ポリシーの例と設定ガイダンスについては、「the section called “ AWS 秘密マネジャー”」および「the section called “ AWS CodeConnections”」を参照してください。

7. AWS アイデンティティセンターの統合を設定します。
   a. [AWS アイデンティティセンターの統合を有効にする] を選択します。
   b. ドロップダウンからアイデンティティセンターインスタンスを選択します。
   c. ユーザーまたはグループを Argo CD ロール (ADMIN、EDITOR、または VIEWER) に割り当てて、RBAC 用にロールマッピングを設定します。
8. (オプション) プライベートエンドポイントを設定する:

   デフォルトでは、Argo CD UI と API エンドポイントはインターネット経由でパブリックにアクセスできます。アクセスを制限する必要がある場合は、VPC エンドポイントを設定できます。ネットワークセキュリティ要件が厳格な環境では、この設定をお勧めします。

   a. 機能を作成する前に、VPC 内に `com.amazonaws.<region>.eks-capabilities` サービス用のインターフェイス VPC エンドポイントを作成します。VPC エンドポイントは次の要件を満たす必要があります。
      - 高可用性を実現するために、異なるアベイラビリティーゾーン内のサブネットに関連付けられている
      - Argo CD UI と API にアクセスする必要があるネットワークからのインバウンド HTTPS (ポート 443) トラフィックを許可するセキュリティグループがある
      - VPC エンドポイントの作成とカスタマイズの詳細については、AWS PrivateLink ガイドの「VPC エンドポイントを作成する」を参照してください。
   b. [Argo CD 機能の作成] ページの Argo CD エンドポイントアクセス – オプション セクションで、[プライベート] を選択します。
   c. ドロップダウンから、作成した VPC エンドポイントを選択します。

> **Note**
>
> プライベートエンドポイントを有効にすると、Argo CD UI と API は VPC エンドポイント経由でのみアクセスできます。Argo CD インターフェイスにアクセスするには、ユーザーは VPC (またはピアリングされたネットワーク) に接続されている必要があります。

9. [作成] を選択します。

機能作成プロセスが始まります。

機能がアクティブであることを確認する

1. [機能] タブで、Argo CD 機能のステータスを表示します。
2. ステータスが `CREATING` から `ACTIVE` に変わるまで待ちます。
3. アクティブになったら、この機能を使用する準備は完了です。

機能のステータスとトラブルシューティングの詳細については、「the section called “機能の使用”」を参照してください。

Argo CD UI にアクセスする

機能がアクティブになったら、以下の手順で Argo CD UI にアクセスできます。

1. Argo CD 機能のページで、[Argo CD UI を開く] を選択します。
2. Argo CD UI がブラウザの新しいタブに開かれます。
3. アプリケーションの作成とデプロイの管理を UI から行えるようになりました。

次のステップ

- the section called “Argo CD の使用” – リポジトリを設定し、クラスターを登録し、Application を作成する
- the section called “Argo CD に関する考慮事項” – マルチクラスターアーキテクチャと高度な設定
- the section called “機能の使用” – Argo CD 機能リソースを管理する

AWS CLI を使用して Argo CD 機能を作成する

AWS CLI を使用して Amazon EKS クラスターで Argo CD 機能を作成します。この手順では、IAM ロールの作成、AWS Identity Center 統合の設定、および機能がアクティブであることの確認について説明します。

前提条件

- AWS CLI バージョン `2.12.3` 以降。バージョンを確認するには、`aws --version` を実行します。詳細についてはAWS コマンドラインインターフェイスユーザーガイドの「インストール」を参照してください。
- **`kubectl`** - Kubernetes クラスターを操作するためのコマンドラインツール。詳細については、「the section called “kubectl と eksctl のセットアップ”」を参照してください。
- AWS アイデンティティセンターが設定されていること – Argo CD での認証には AWS アイデンティティセンターが必要です。ローカルユーザーはサポートされていません。AWS アイデンティティセンターをセットアップしていない場合は、「AWS アイデンティティセンターの使用を開始する」を参照してアイデンティティセンターインスタンスを作成し、「ユーザーを追加する」と「グループを追加する」を参照して Argo CD へのアクセスに必要なユーザーとグループを作成します。
- AWS アイデンティティセンターのユーザーまたはグループが少なくとも 1 つ – Argo CD の RBAC ロールマッピングを割り当て、Argo CD UI へのアクセスを提供するには、アイデンティティセンターインスタンスに少なくとも 1 人のユーザーまたは 1 つのグループを設定する必要があります。

ステップ 1: IAM 機能ロールを作成する

信頼ポリシーファイルを作成します。

```
cat > argocd-trust-policy.json << 'EOF'
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "capabilities.eks.amazonaws.com"
      },
      "Action": [
        "sts:AssumeRole",
```

```
        "sts:TagSession"
      ]
    }
  ]
}
EOF
```

IAM ロールを作成します。

```
aws iam create-role \
  --role-name ArgoCDCapabilityRole \
  --assume-role-policy-document file://argocd-trust-policy.json
```

> **Note**
>
> AWS Secrets Manager または AWS CodeConnections をオプションで統合する場合は、ロールにアクセス許可を追加します。IAM ポリシーの例と設定ガイダンスについては、「the section called “ AWS 秘密マネジャー”」および「the section called “ AWS CodeConnections”」を参照してください。

(オプション) プライベートエンドポイントを設定する

デフォルトでは、Argo CD UI と API エンドポイントはインターネット経由でパブリックにアクセスできます。アクセスを制限する必要がある場合は、VPC エンドポイントを設定できます。ネットワークセキュリティ要件が厳格な環境では、この設定をお勧めします。

EKS 機能用の VPC エンドポイントを作成する

VPC 内に EKS 機能サービスのインターフェイス VPC エンドポイントを作成します。*vpc-id*、*subnet-id-1 subnet-id-2*、*sg-id*、*region-code* を実際の値に置き換えます。

```
aws ec2 create-vpc-endpoint \
  --vpc-endpoint-type Interface \
  --service-name com.amazonaws.region-code.eks-capabilities \
  --vpc-id vpc-id \
  --subnet-ids subnet-id-1 subnet-id-2 \
  --security-group-ids sg-id \
  --region region-code
```

> **Note**
>
> - 高可用性を実現するには、サブネットが異なるアベイラビリティーゾーンにある必要があります。
> - セキュリティグループでは、Argo CD UI と API にアクセスする必要があるネットワークからのインバウンド HTTPS (ポート 443) トラフィックを許可する必要があります。
> - このコマンドで返される VPC エンドポイント ID をメモしてください。機能を作成するときに必要になります。

VPC エンドポイントが利用可能であることを確認する

```
aws ec2 describe-vpc-endpoints \
  --vpc-endpoint-ids vpce-xxxxxxxx \
  --query 'VpcEndpoints[0].State' \
  --output text \
  --region region-code
```

続行する前に、状態が `available` と表示されるまで待ちます。

ステップ 2: Argo CD 機能を作成する

クラスターに Argo CD 機能リソースを作成します。

まず、アイデンティティセンターを設定するための環境変数を設定します。

```
# Get your Identity Center instance ARN (replace region if your IDC instance is in a
 different region)
export IDC_INSTANCE_ARN=$(aws sso-admin list-instances --region region-code --query
 'Instances[0].InstanceArn' --output text)

# Get a user ID for RBAC mapping (replace with your username and region if needed)
export IDC_USER_ID=$(aws identitystore list-users \
  --region region-code \
  --identity-store-id $(aws sso-admin list-instances --region region-code --query
 'Instances[0].IdentityStoreId' --output text) \
  --query 'Users[?UserName==your-username].UserId' --output text)

echo "IDC_INSTANCE_ARN=$IDC_INSTANCE_ARN"
```

```
echo "IDC_USER_ID=$IDC_USER_ID"
```

アイデンティティセンター統合で機能を作成します。*region-code* をクラスターが配置されている AWS リージョンに、*my-cluster* をクラスター名に、*idc-region-code* を Identity Center インスタンスが配置されているリージョンコードに置き換えます。

```
aws eks create-capability \
  --region region-code \
  --cluster-name my-cluster \
  --capability-name my-argocd \
  --type ARGOCD \
  --role-arn arn:aws:iam::$(aws sts get-caller-identity --query Account --output
 text):role/ArgoCDCapabilityRole \
  --delete-propagation-policy RETAIN \
  --configuration '{
    "argoCd": {
      "awsIdc": {
        "idcInstanceArn": "'$IDC_INSTANCE_ARN'",
        "idcRegion": "'idc-region-code'"
      },
      "rbacRoleMappings": [{
        "role": "ADMIN",
        "identities": [{
          "id": "'$IDC_USER_ID'",
          "type": "SSO_USER"
        }]
      }]
    }
  }'
```

プライベートアクセス用に VPC エンドポイントを設定した場合は、network-configuration パラメータを含めて、プライベートエンドポイントを持つ機能を作成します。*vpce-xxxxxxxx* を実際の VPC エンドポイント ID に置き換えます。

```
aws eks create-capability \
  --region region-code \
  --cluster-name my-cluster \
  --capability-name my-argocd \
  --type ARGOCD \
  --role-arn arn:aws:iam::$(aws sts get-caller-identity --query Account --output
 text):role/ArgoCDCapabilityRole \
  --delete-propagation-policy RETAIN \
```

```
  --network-configuration '{
    "elasticNetworkInterfaces": {
      "vpcEndpointId": "'vpce-xxxxxxxx'"
    }
  }' \
  --configuration '{
    "argoCd": {
      "awsIdc": {
        "idcInstanceArn": "'$IDC_INSTANCE_ARN'",
        "idcRegion": "'idc-region-code'"
      },
      "rbacRoleMappings": [{
        "role": "ADMIN",
        "identities": [{
          "id": "'$IDC_USER_ID'",
          "type": "SSO_USER"
        }]
      }]
    }
  }'
```

このコマンドはすぐに戻りますが、EKS が必要な機能インフラストラクチャとコンポーネントを作成するため、機能がアクティブになるまでにはしばらく時間がかかります。EKS は、機能作成時に Kubernetes カスタムリソース定義 (CRD) をクラスターにインストールします。

> **Note**
>
> クラスターが存在しないというエラーやアクセス許可がないというエラーが表示された場合は、以下の点を確認します。
>
> - クラスター名が正しいこと
> - AWS CLI が正しいリージョンに設定されていること
> - 必要な IAM アクセス許可を追加したこと

ステップ 3: 機能がアクティブであることを確認する

機能がアクティブになるまで待機します。*region-code* はクラスターが配置された AWS リージョンに置き換え、*my-cluster* はクラスターの名前に置き換えます。

```
aws eks describe-capability \
```

```
  --region region-code \
  --cluster-name my-cluster \
  --capability-name my-argocd \
  --query 'capability.status' \
  --output text
```

ステータスが ACTIVE と表示されたら、機能の準備は完了です。ステータスが ACTIVE になるまで待ってから、次のステップに進みます。

機能の詳細全体を表示することもできます。

```
aws eks describe-capability \
  --region region-code \
  --cluster-name my-cluster \
  --capability-name my-argocd
```

ステップ 4: カスタムリソースが使用可能であることを確認する

機能がアクティブになったら、Argo CD カスタムリソースがクラスターで使用可能になっていることを確認します。

```
kubectl api-resources | grep argoproj.io
```

Application と ApplicationSet のリソースタイプがリストされます。

次のステップ

- the section called “Argo CD の使用” - リポジトリを設定し、クラスターを登録し、Application を作成する
- the section called “Argo CD に関する考慮事項” - マルチクラスターアーキテクチャと高度な設定
- the section called “機能の使用” - Argo CD 機能リソースを管理する

eksctl を使用して Argo CD 機能を作成する

eksctl を使用して Amazon EKS クラスターで Argo CD 機能を作成します。

> **Note**
>
> 以下のステップを実行するには、eksctl バージョン `0.215.0` 以降が必要です。バージョンを確認するには、`eksctl version` を実行します。

前提条件

- AWS アイデンティティセンターが設定されていること – Argo CD での認証には AWS アイデンティティセンターが必要です。ローカルユーザーはサポートされていません。AWS アイデンティティセンターをセットアップしていない場合は、「AWS アイデンティティセンターの使用を開始する」を参照してアイデンティティセンターインスタンスを作成し、「ユーザーを追加する」と「グループを追加する」を参照して Argo CD へのアクセスに必要なユーザーとグループを作成します。
- AWS アイデンティティセンターのユーザーまたはグループが少なくとも 1 つ – Argo CD の RBAC ロールマッピングを割り当て、Argo CD UI へのアクセスを提供するには、アイデンティティセンターインスタンスに少なくとも 1 人のユーザーまたは 1 つのグループを設定する必要があります。

ステップ 1: IAM 機能ロールを作成する

信頼ポリシーファイルを作成します。

```
cat > argocd-trust-policy.json << 'EOF'
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "capabilities.eks.amazonaws.com"
      },
      "Action": [
        "sts:AssumeRole",
        "sts:TagSession"
      ]
    }
  ]
}
```

```
EOF
```

IAM ロールを作成します。

```
aws iam create-role \
  --role-name ArgoCDCapabilityRole \
  --assume-role-policy-document file://argocd-trust-policy.json
```

> **Note**
>
> こうした基本的なセットアップの場合、追加の IAM ポリシーは必要ありません。リポジトリ認証情報や CodeConnections に Secrets Manager を使用する場合は、ロールにアクセス許可を追加する必要があります。IAM ポリシーの例と設定ガイダンスについては、「the section called “ AWS 秘密マネジャー”」および「the section called “ AWS CodeConnections”」を参照してください。

ステップ 2: AWS アイデンティティセンター設定を取得する

RBAC 設定用にアイデンティティセンターインスタンス ARN とユーザー ID を取得します。

```
# Get your Identity Center instance ARN
aws sso-admin list-instances --query 'Instances[0].InstanceArn' --output text

# Get a user ID for admin access (replace 'your-username' with your Identity Center
 username)
aws identitystore list-users \
  --identity-store-id $(aws sso-admin list-instances --query
 'Instances[0].IdentityStoreId' --output text) \
  --query 'Users[?UserName==`your-username`].UserId' --output text
```

これらの値をメモしてください。次のステップで必要になります。

(オプション) プライベートエンドポイントを設定する

デフォルトでは、Argo CD UI と API エンドポイントはインターネット経由でパブリックにアクセスできます。アクセスを制限する必要がある場合は、VPC エンドポイントを設定できます。ネットワークセキュリティ要件が厳格な環境では、この設定をお勧めします。

EKS 機能用の VPC エンドポイントを作成する

VPC 内に EKS 機能サービスのインターフェイス VPC エンドポイントを作成します。*vpc-id*、*subnet-id-1 subnet-id-2*、*sg-id*、*region-code* を実際の値に置き換えます。

```
aws ec2 create-vpc-endpoint \
  --vpc-endpoint-type Interface \
  --service-name com.amazonaws.region-code.eks-capabilities \
  --vpc-id vpc-id \
  --subnet-ids subnet-id-1 subnet-id-2 \
  --security-group-ids sg-id \
  --region region-code
```

Note

- 高可用性を実現するには、サブネットが異なるアベイラビリティーゾーンにある必要があります。
- セキュリティグループでは、Argo CD UI と API にアクセスする必要があるネットワークからのインバウンド HTTPS (ポート 443) トラフィックを許可する必要があります。
- このコマンドで返される VPC エンドポイント ID をメモしてください。機能を作成するときに必要になります。

VPC エンドポイントが利用可能であることを確認する

```
aws ec2 describe-vpc-endpoints \
  --vpc-endpoint-ids vpce-xxxxxxxx \
  --query 'VpcEndpoints[0].State' \
  --output text \
  --region region-code
```

続行する前に、状態が `available` と表示されるまで待ちます。

ステップ 3: eksctl 設定ファイルを作成する

`argocd-capability.yaml` という名前のファイルを作成し、次の内容を記述します。各プレースホルダーの値をそれぞれクラスター名、クラスターのリージョン、IAM ロール ARN、アイデンティ

ティセンターインスタンス ARN、アイデンティティセンターリージョン、およびユーザー ID に置き換えます。

```
apiVersion: eksctl.io/v1alpha5
kind: ClusterConfig

metadata:
  name: my-cluster
  region: cluster-region-code

capabilities:
  - name: my-argocd
    type: ARGOCD
    roleArn: arn:aws:iam::111122223333:role/ArgoCDCapabilityRole
    deletePropagationPolicy: RETAIN
    configuration:
      argocd:
        awsIdc:
          idcInstanceArn: arn:aws:sso:::instance/ssoins-123abc
          idcRegion: idc-region-code
        rbacRoleMappings:
          - role: ADMIN
            identities:
              - id: 38414300-1041-708a-01af-5422d6091e34
                type: SSO_USER
```

> **Note**
>
> RBAC マッピングに複数のユーザーまたはグループを追加できます。グループの場合、type: SSO_GROUP を使用してグループ ID を指定します。使用可能なロールは、ADMIN、EDITOR、および VIEWER です。

プライベートアクセス用に VPC エンドポイントを設定した場合は、機能定義に networkConfiguration セクションを追加します。*vpce-xxxxxxxx* を実際の VPC エンドポイント ID に置き換えます。

```
apiVersion: eksctl.io/v1alpha5
kind: ClusterConfig

metadata:
```

```
  name: my-cluster
  region: cluster-region-code

capabilities:
  - name: my-argocd
    type: ARGOCD
    roleArn: arn:aws:iam::111122223333:role/ArgoCDCapabilityRole
    deletePropagationPolicy: RETAIN
    networkConfiguration:
      elasticNetworkInterfaces:
        vpcEndpointId: vpce-xxxxxxxx
    configuration:
      argocd:
        awsIdc:
          idcInstanceArn: arn:aws:sso:::instance/ssoins-123abc
          idcRegion: idc-region-code
        rbacRoleMappings:
          - role: ADMIN
            identities:
              - id: 38414300-1041-708a-01af-5422d6091e34
                type: SSO_USER
```

> **Note**
>
> プライベートエンドポイントを有効にすると、Argo CD UI と API は VPC エンドポイント経由でのみアクセスできます。Argo CD インターフェイスにアクセスするには、ユーザーは VPC (またはピアリングされたネットワーク) に接続されている必要があります。

ステップ 4: Argo CD 機能を作成する

設定ファイルを適用します。

```
eksctl create capability -f argocd-capability.yaml
```

このコマンドはすぐに戻りますが、機能がアクティブになるまでにはしばらく時間がかかります。

ステップ 5: 機能がアクティブであることを確認する

機能のステータスを確認します。*region-code* はクラスターがある AWS リージョンに、*my-cluster* はクラスターの名前に置き換えます。

```
eksctl get capability \
  --region region-code \
  --cluster my-cluster \
  --name my-argocd
```

ステータスが ACTIVE と表示されたら、機能の準備は完了です。

ステップ 6: カスタムリソースが使用可能であることを確認する

機能がアクティブになったら、Argo CD カスタムリソースがクラスターで使用可能になっていることを確認します。

```
kubectl api-resources | grep argoproj.io
```

Application と ApplicationSet のリソースタイプがリストされます。

次のステップ

- the section called “Argo CD の使用” – Argo CD Application を作成および管理する方法を学ぶ
- the section called “Argo CD に関する考慮事項” – SSO とマルチクラスターアクセスを設定する
- the section called “機能の使用” – Argo CD 機能リソースを管理する

Argo CD の概念

Argo CD は、Git をアプリケーションデプロイの信頼できる単一のソースとして扱うことで GitOps を実装します。このトピックでは、実践的な例を取り上げ、EKS Capability for Argo CD を使用する際に知っておくべきコアとなる概念について説明します。

Argo CD の使用を開始する

Argo CD 機能を作成したら (「the section called “Argo CD 機能を作成する”」を参照)、アプリケーションのデプロイを開始できます。この例では、クラスターの登録と Application の作成を取り上げます。

ステップ 1: 設定

クラスターを登録する (必須)

アプリケーションのデプロイ先となるクラスターを登録します。この例では、Argo CD が実行されているのと同じクラスターを登録します (`in-cluster` という名前を使用すると、Argo CD のほとんどの例との互換性を維持できます)。

```
# Get your cluster ARN
CLUSTER_ARN=$(aws eks describe-cluster \
  --name my-cluster \
  --query 'cluster.arn' \
  --output text)

# Register the cluster using Argo CD CLI
argocd cluster add $CLUSTER_ARN \
  --aws-cluster-name $CLUSTER_ARN \
  --name in-cluster \
  --project default
```

> **Note**
>
> EKS で Argo CD 機能を使用するように Argo CD CLI を設定する方法については、「the section called “マネージド機能で Argo CD CLI を使用する”」を参照してください。

あるいは、Kubernetes シークレットを使用してクラスターを登録します (詳細については、「the section called “クラスターを登録する”」を参照)。

リポジトリアクセスを設定する (任意)

この例では、パブリック GitHub リポジトリを使用しているため、リポジトリの設定は不要です。プライベートリポジトリの場合は、AWS Secrets Manager、CodeConnections、または Kubernetes シークレットを使用してアクセスを設定します (詳細については、「the section called “リポジトリを設定する”」を参照)。

AWS サービス (ECR Helm チャート、CodeConnections、および CodeCommit) の場合、リポジトリを作成しなくても、Application リソースで直接参照できます。機能ロールには、IAM アクセス許可が必要です。詳細については、「the section called “リポジトリを設定する”」を参照してください。

ステップ 2: アプリケーションの作成

`my-app.yaml` でこの Application マニフェストを作成します。

```
apiVersion: argoproj.io/v1alpha1
```

```
kind: Application
metadata:
  name: guestbook
  namespace: argocd
spec:
  project: default
  source:
    repoURL: https://github.com/argoproj/argocd-example-apps.git
    targetRevision: HEAD
    path: guestbook
  destination:
    name: in-cluster
    namespace: guestbook
  syncPolicy:
    automated:
      prune: true
      selfHeal: true
    syncOptions:
    - CreateNamespace=true
```

Application を適用します。

```
kubectl apply -f my-app.yaml
```

この Application を適用したら、Argo CD で次の手順を実行します。1. Git からクラスターにアプリケーションを同期します (初期デプロイ)。2. Git リポジトリに対する変更をモニタリングします。3. 今後変更があればクラスターに自動的に同期します。4. 目的の状態からのドリフトを検出して修正します。5. UI でヘルスステータスと同期履歴を提供します。

アプリケーションのステータスを表示します。

```
kubectl get application guestbook -n argocd
```

Argo CD CLI または Argo CD UI (クラスターの [機能] タブの EKS コンソールからアクセス可能) を使用して、アプリケーションを表示することもできます。

> **Note**
>
> マネージド機能で Argo CD CLI を使用する場合は、名前空間プレフィックス `argocd app get argocd/guestbook` を使用してアプリケーションを指定します。

> **Note**
>
> `destination.name` でクラスター名 (クラスターの登録時に使用した名前) を使用します。マネージド機能は、ローカルのクラスター内デフォルト (`kubernetes.default.svc`) をサポートしていません。

重要な概念

GitOps の原則とソースタイプ

Argo CD は、GitOps を実装しており、アプリケーションソースがデプロイの信頼できる単一のソースとなります。

- 宣言型 - 目的の状態は、YAML マニフェスト、Helm チャート、または Kustomize オーバーレイを使用して宣言されます。
- バージョン管理 - すべての変更が完全な監査証跡で追跡されます。
- 自動 - Argo CD は、ソースを継続的にモニタリングし、変更を自動的に同期します。
- 自己修復 - 目的のクラスター状態と実際のクラスター状態とのドリフトを検出して修正します。

サポートされているソースタイプ:

- Git リポジトリ - GitHub、GitLab、Bitbucket、CodeCommit (HTTPS、SSH、または CodeConnections)
- Helm レジストリ - HTTP レジストリ (`https://aws.github.io/eks-charts` など) および OCI レジストリ (`public.ecr.aws` など)
- OCI イメージ - マニフェストまたは Helm チャートが含まれているコンテナイメージ (`oci://registry-1.docker.io/user/my-app` など)

この柔軟性により、組織のセキュリティ要件とコンプライアンス要件を満たすソースを選択できます。例えば、クラスターからの Git アクセスを制限する場合は、Helm チャートや OCI イメージに ECR を使用できます。

詳細は、Argo CD ドキュメントの「Application ソース」を参照してください。

同期と調整

Argo CD は、ソースとクラスターを継続的にモニタリングして、差分を検出して修正します。

1. 変更がないかソースをポーリングします (デフォルト: 6 分ごと)。
2. 目的の状態とクラスターの状態を比較します。
3. アプリケーションを Synced または OutOfSync としてマークします。
4. 変更を自動的に同期するか (設定されている場合)、手動による承認を待ちます。
5. 同期後にリソースのヘルスをモニタリングします。

同期ウェーブは、注釈を使用してリソースの作成順序を制御します。

```
metadata:
  annotations:
    argocd.argoproj.io/sync-wave: "0"  # Default if not specified
```

リソースはウェーブ順に適用されます (先頭は -1 のような負の数値など、小さい数値です)。指定しない場合は、ウェーブ 0 がデフォルトになります。これにより、サービス (ウェーブ 1) の前にデプロイ (ウェーブ 0)、その前に名前空間 (ウェーブ -1) などの依存関係を作成できます。

自己修復は、手動による変更を自動的に元に戻します。

```
spec:
  syncPolicy:
    automated:
      selfHeal: true
```

> **Note**
>
> マネージド機能は、注釈ベース (ラベルベースではない) のリソース追跡を使用して、Kubernetes の規則やその他のツールとの互換性を高めます。

同期フェーズ、フック、高度なパターンの詳細については、Argo CD 同期のドキュメントを参照してください。

アプリケーションのヘルス

Argo CD は、アプリケーション内のすべてのリソースのヘルスをモニタリングします。

ヘルスステータス: * 正常 - すべてのリソースが想定どおりに動作している * 進行中 - リソースが作成中または更新中 * 低下 - 一部のリソースが正常でない (ポッドのクラッシュやジョブの失敗) * 中断 -

アプリケーションが意図的に一時停止している * 欠落 - Git に定義されたリソースがクラスターに存在しない

よく使用される Kubernetes リソース (デプロイ、StatefulSet、ジョブなど) に対しては組み込みのヘルスチェックで対処し、CRD に対してはカスタムヘルスチェックをサポートしています。

アプリケーションのヘルスは、アプリケーションのすべてのリソースによって決まります。いずれかのリソースが `Degraded` であれば、アプリケーションも `Degraded` です。

詳細については、Argo CD ドキュメントの「リソースのヘルス」を参照してください。

マルチクラスターパターン

Argo CD は、主に 2 つのデプロイパターンをサポートしています。

ハブアンドスポーク - 複数のワークロードクラスターにデプロイする専用の管理クラスターで Argo CD を実行します。* 一元管理と可視性 * すべてのクラスターで一貫したポリシー * Argo CD インスタンスを 1 つ管理 * コントロールプレーンとワークロードとの明確な分離

クラスターあたり - 各クラスターで Argo CD を実行して、そのクラスターのアプリケーションのみを管理します。* クラスターの分離 (あるクラスターの障害が他のクラスターに影響を与えない) * シンプルなネットワーク (クラスター間の通信なし) * シンプルな初期セットアップ (クラスターの登録なし)

プラットフォームチームが多数のクラスターを管理する場合はハブアンドスポークを選択し、チームが独立している場合やクラスターを完全に分離する必要がある場合はクラスターあたりを選択します。

マルチクラスター設定の詳細については、「the section called “Argo CD に関する考慮事項”」を参照してください。

プロジェクト

Project を使用すると、Application を論理的にグループ化して、アクセスコントロールを付与できます。

- ソースの制限 - 使用できる Git リポジトリを制限します。
- 送信先の制限 - ターゲットにできるクラスターと名前空間を制限します。
- リソースの制限 - デプロイできる Kubernetes リソースタイプを制限します。

- RBAC の統合 - Project を AWS アイデンティティセンターのユーザー ID とグループ ID にマッピングします。

Application は単一の Project に属します。指定しない場合は、`default` プロジェクトが使用されます。これにはデフォルトの制限はありません。本番稼働用の場合は、`default` プロジェクトを編集してアクセスを制限し、適切な制限で新しいプロジェクトを作成します。

Project の設定と RBAC パターンについては、「the section called “アクセス許可の設定”」を参照してください。

同期オプション

よく使用されるオプションで同期動作をファインチューニングします。

- `CreateNamespace=true` - 送信先名前空間を自動的に作成します。
- `ServerSideApply=true` - サーバー側の適用機能を使用して的確に競合を解決します。
- `SkipDryRunOnMissingResource=true` - CRD がまだ存在しない場合には、ドライランをスキップします (kro インスタンスに有用です)。

```
spec:
  syncPolicy:
    syncOptions:
    - CreateNamespace=true
    - ServerSideApply=true
    - SkipDryRunOnMissingResource=true
```

同期オプションの完全なリストについては、Argo CD 同期オプションのドキュメントを参照してください。

次のステップ

- the section called “リポジトリを設定する” - Git リポジトリアクセスを設定する
- the section called “クラスターを登録する” - デプロイ用にターゲットクラスターを登録する
- the section called “Application を作成する” - 最初の Application を作成する
- the section called “Argo CD に関する考慮事項” - EKS 固有のパターン、アイデンティティセンター統合、マルチクラスター設定

- Argo CD ドキュメント - 同期フック、ヘルスチェック、高度なパターンなどについて説明した包括的な Argo CD ドキュメント

Argo CD アクセス許可を設定する

Argo CD マネージド機能は、AWS アイデンティティセンターとの統合により認証を実施し、組み込みの RBAC ロールを使用して認可を行います。このトピックでは、ユーザーとチームに対するアクセス許可を設定する方法について説明します。

Argo CD によるアクセス許可の仕組み

Argo CD 機能は、AWS アイデンティティセンターを使用して認証を実施し、組み込みの 3 つの RBAC ロールを使用して認可を行います。

ユーザーが Argo CD にアクセスすると、次のようになります。

1. AWS アイデンティティセンターを使用してユーザーが認証されます (社内の ID プロバイダーにフェデレーションできます)。
2. AWS アイデンティティセンターは、ユーザーとグループの情報を Argo CD に提供します。
3. Argo CD は、現在の設定に基づいてユーザーとグループを RBAC ロールにマッピングします。
4. ユーザーには、自分がアクセス許可を持つアプリケーションとリソースのみが表示されます。

組み込みの RBAC ロール

Argo CD 機能には、AWS アイデンティティセンターのユーザーとグループにマッピングする組み込みのロールが 3 つあります。これらは、プロジェクト、クラスター、リポジトリなどの Argo CD リソースへのアクセスを制御するグローバルスコープのロールです。

> **Important**
>
> グローバルロールは、Application などのプロジェクトスコープのリソースではなく、Argo CD 自体へのアクセスを制御します。EDITOR ユーザーおよび VIEWER ユーザーは、デフォルトでは Application を表示または管理できません。プロジェクトスコープのリソースにアクセスするには、プロジェクトロールが必要です。Application およびその他のプロジェクトスコープのリソースへのアクセス許可の詳細については、「the section called “プロジェクトロールとプロジェクトスコープのアクセス”」を参照してください。

ADMIN

すべての Argo CD リソースと設定へのフルアクセス:

- 任意のプロジェクト内の Application と ApplicationSet を作成、更新、削除する
- Argo CD 設定を管理する
- デプロイターゲットクラスターを登録して管理する
- リポジトリアクセスを設定する
- プロジェクトを作成および管理する
- すべてのアプリケーションのステータスと履歴を表示する
- すべてのクラスターとリポジトリを一覧表示してアクセスする

EDITOR

プロジェクトを更新してプロジェクトロールを設定できますが、グローバル Argo CD 設定を変更することはできません。

- 既存のプロジェクトを更新する (プロジェクトの新規作成および削除はできません)
- プロジェクトロールとアクセス許可を設定する
- GPG キーと証明書を表示する
- グローバル Argo CD 設定を変更できない
- クラスターとリポジトリを直接管理できない
- プロジェクトロールがないと Application を表示または管理できない

VIEWER

Argo CD リソースへの読み取り専用アクセス:

- プロジェクト設定を表示する
- すべてのプロジェクト (ユーザーが割り当てられていないプロジェクトを含む) を一覧表示する
- GPG キーと証明書を表示する
- クラスターまたはリポジトリを一覧表示できない
- 変更を加えることはできない
- プロジェクトロールがないと Application を表示または管理できない

> **Note**
>
> EDITOR ユーザーまたは VIEWER ユーザーに Application へのアクセス権を付与するには、ADMIN または EDITOR が、アイデンティティセンターグループをプロジェクト内の特定のアクセス許可にマッピングするプロジェクトロールを作成する必要があります。

プロジェクトロールとプロジェクトスコープのアクセス

グローバルロール (ADMIN、EDITOR、VIEWER) は、Argo CD 自体へのアクセスを制御します。プロジェクトロールは、以下を含む特定のプロジェクト内のリソースと機能へのアクセスを制御します。

- リソース: Application、ApplicationSet、リポジトリ認証情報、クラスター認証情報
- 機能: アプリケーションポッドへのログアクセスと実行アクセス

2 レベルのアクセス許可モデルについて:

- グローバルスコープ: 組み込みロールは、ユーザーがプロジェクト、クラスター、リポジトリ、および Argo CD 設定で何ができるかを決定します
- プロジェクトスコープ: プロジェクトロールは、ユーザーが特定のプロジェクト内のリソースと機能で何ができるかを決定します

つまり、次のようになります。

- ADMIN ユーザーは、追加の設定なしですべてのプロジェクトリソースと機能にアクセスできます
- EDITOR ユーザーおよび VIEWER ユーザーには、プロジェクトのリソースと機能にアクセスするためのプロジェクトロールを付与する必要があります
- EDITOR ユーザーは、更新できるプロジェクト内で、自分やそれ以外のユーザーにアクセス権を付与するプロジェクトロールを作成できます。

ワークフローの例:

1. ADMIN は、アイデンティティセンターグループを EDITOR ロールにグローバルにマッピングします。
2. ADMIN がチームのプロジェクトを作成する

3. EDITOR は、そのプロジェクト内のプロジェクトロールを設定して、チームメンバーにプロジェクトスコープのリソースへのアクセス権を付与します。
4. チームメンバー (VIEWER グローバルロールを持つ可能性がある) は、プロジェクトロールのアクセス許可に基づいて、そのプロジェクトの Application を表示および管理できるようになりました。

プロジェクトロールの設定の詳細については、「the section called “Project ベースのアクセスコントロール”」を参照してください。

ロールマッピングを設定する

機能の作成時や更新時に、AWS アイデンティティセンターのユーザーとグループを Argo CD ロールにマッピングします。

ロールマッピングの例:

```
{
  "rbacRoleMappings": {
    "ADMIN": ["AdminGroup", "alice@example.com"],
    "EDITOR": ["DeveloperGroup", "DevOpsTeam"],
    "VIEWER": ["ReadOnlyGroup", "bob@example.com"]
  }
}
```

> **Note**
>
> ロール名は、大文字と小文字が区別され、大文字 (ADMIN、EDITOR、VIEWER) にする必要があります。

> **Important**
>
> EKS 機能と AWS アイデンティティセンターが統合されている場合、Argo CD 機能あたり最大 1,000 個の ID がサポートされます。ユーザーやグループを ID にすることができます。

ロールマッピングを更新する:

```
aws eks update-capability \
```

```
  --region us-east-1 \
  --cluster-name cluster \
  --capability-name capname \
  --role-arn "arn:aws:iam::111122223333:role/EKSCapabilityRole" \
  --configuration '{
    "argoCd": {
      "rbacRoleMappings": {
        "addOrUpdateRoleMappings": [
          {
            "role": "ADMIN",
            "identities": [
              { "id": "686103e0-f051-7068-b225-e6392b959d9e", "type": "SSO_USER" }
            ]
          }
        ]
      }
    }
  }'
```

管理者アカウントの使用法

管理者アカウントは、クラスターの登録やリポジトリの設定など、初期セットアップと管理タスク向けに設計されています。

どのような場合に管理者アカウントが適切か:

- 機能の初期セットアップと設定
- 単独開発や迅速なデモンストレーション
- 管理タスク (クラスターの登録、リポジトリの設定、Project の作成)

管理者アカウントのベストプラクティス:

- アカウントトークンをバージョン管理にコミットしない
- トークンが漏洩したらすぐにローテーションさせる
- アカウントトークンの用途をセットアップと管理タスクに制限する
- 有効期限を短くする (最大 12 時間)
- 一度に作成できるアカウントトークンを 5 つまでとする

どのような場合に Project ベースのアクセスを使用するか:

- 複数のユーザーと開発環境を共有する場合
- いずれの環境も本番稼働と同様にする場合
- 誰がアクションを実行したかを監査証跡に残す場合
- リソース制限やアクセス境界を設ける場合

本番稼働環境とマルチユーザーシナリオでは、専用の RBAC ロールを AWS アイデンティティセンターグループにマッピングして、Project ベースのアクセスコントロールを使用します。

Project ベースのアクセスコントロール

Argo CD Project (AppProject) を使用すると、チームを対象にアクセスコントロールとリソース分離をきめ細かく設定できます。

> **Important**
>
> ユーザーまたはグループをプロジェクト固有のロールに割り当てる前に、まず機能設定でグローバル Argo CD ロール (ADMIN、EDITOR、または VIEWER) にマッピングする必要があります。ユーザーは、プロジェクトロールに割り当てられていても、グローバルロールマッピングなしで Argo CD にアクセスすることはできません。
> ユーザーを VIEWER ロールにグローバルにマッピングし、プロジェクト固有のロールを通じて追加のアクセス許可を付与することを検討してください。これにより、ベースラインアクセスが可能になり、プロジェクトレベルでのきめ細かな制御が可能になります。

Project でできることは以下のとおりです。

- ソースの制限: 使用できる Git リポジトリを制限します。
- 送信先の制限: ターゲットにできるクラスターと名前空間を制限します。
- リソースの制限: デプロイできる Kubernetes リソースタイプを制限します。
- RBAC の統合: Project を AWS アイデンティティセンターグループまたは Argo CD ロールにマッピングします。

チームを分離する Project の例:

```
apiVersion: argoproj.io/v1alpha1
kind: AppProject
```

```
metadata:
  name: team-a
  namespace: argocd
spec:
  description: Team A applications

  # Required: Specify which namespaces this project watches for Applications
  sourceNamespaces:
  - argocd

  # Source restrictions
  sourceRepos:
  - https://github.com/myorg/team-a-apps

  # Destination restrictions
  destinations:
  - namespace: team-a-*
    server: arn:aws:eks:us-west-2:111122223333:cluster/production

  # Resource restrictions
  clusterResourceWhitelist:
  - group: ''
    kind: Namespace
  namespaceResourceWhitelist:
  - group: 'apps'
    kind: Deployment
  - group: ''
    kind: Service
  - group: ''
    kind: ConfigMap
```

ソース名前空間

EKS Argo CD 機能を使用する場合は、AppProject 定義で `spec.sourceNamespaces` フィールドが必要です。このフィールドは、このプロジェクトを参照する Application または ApplicationSet を含めることができる名前空間を指定します。

> **Important**
>
> EKS Argo CD 機能は、Application と ApplicationSet 向けに一つの名前空間のみをサポートします。具体的には、機能の作成時に指定した名前空間になります (通常は `argocd`)。これは、複数の名前空間をサポートするオープンソースの Argo CD とは異なります。

AppProject の設定

すべての AppProject は、その機能の設定済み名前空間を sourceNamespaces に含める必要があります。

```
apiVersion: argoproj.io/v1alpha1
kind: AppProject
metadata:
  name: team-a-project
  namespace: argocd
spec:
  description: Applications for Team A

  # Required: Specify the capability's configured namespace
 (configuration.argoCd.namespace)
  sourceNamespaces:
    - argocd  # Must match your capability's namespace configuration

  # Source repositories this project can deploy from
  sourceRepos:
    - 'https://github.com/my-org/team-a-*'

  # Destination restrictions
  destinations:
    - namespace: 'team-a-*'
      server: arn:aws:eks:us-west-2:111122223333:cluster/my-cluster
```

> **Note**
>
> 機能の名前空間を sourceNamespaces に含めないと、その名前空間の Application または ApplicationSet はこのプロジェクトを参照できず、デプロイが失敗します。

ユーザーを Project に割り当てる:

プロジェクトロールは、EDITOR ユーザーおよび VIEWER ユーザーにプロジェクトリソース (Application、ApplicationSet、リポジトリ、クラスター認証情報) と機能 (ログ、実行) へのアクセスを許可します。プロジェクトロールがないと、これらのユーザーは、グローバルロールアクセスを持っていてもこれらのリソースにアクセスできません。

ADMIN ユーザーは、プロジェクトロールを必要とせずに、すべての Application にアクセスできます。

例: チームメンバーに Application へのアクセスを許可する

```
apiVersion: argoproj.io/v1alpha1
kind: AppProject
metadata:
  name: team-a
  namespace: argocd
spec:
  # ... project configuration ...

  sourceNamespaces:
  - argocd

  # Project roles grant Application-level access
  roles:
  - name: developer
    description: Team A developers - can manage Applications
    policies:
    - p, proj:team-a:developer, applications, *, team-a/*, allow
    - p, proj:team-a:developer, clusters, get, *, allow  # See cluster names in UI
    groups:
    - 686103e0-f051-7068-b225-e6392b959d9e  # Identity Center group ID

  - name: viewer
    description: Team A viewers - read-only Application access
    policies:
    - p, proj:team-a:viewer, applications, get, team-a/*, allow
    - p, proj:team-a:viewer, clusters, get, *, allow  # See cluster names in UI
    groups:
    - 786203e0-f051-7068-b225-e6392b959d9f  # Identity Center group ID
```

Note

プロジェクトロールに `clusters, get, *, allow` を含めることで、ユーザーが UI でクラスター名を表示できるようにします。このアクセス許可がない場合、送信先クラスターは「Unknown」として表示されます。

プロジェクトロールポリシーについて:

ポリシーの形式は次のとおりです: `p, proj:<project>:<role>, <resource>, <action>, <object>, <allow/deny>`

リソースポリシー:

- `applications, , team-a/, allow` – team-a プロジェクト内のすべての Application にフルアクセスできます
- `applications, get, team-a/*, allow` – Application に読み取り専用でアクセスできます
- `applications, sync, team-a/*, allow` – Application を同期できますが、作成/削除することはできません
- `applications, delete, team-a/*, allow` – Application を削除できます (注意して使用すること)
- `applicationsets, , team-a/, allow` – ApplicationSet にフルアクセスできます
- `repositories, *, *, allow` – リポジトリ認証情報にアクセスできます
- `clusters, *, *, allow` – クラスター認証情報にアクセスできます

機能ポリシー:

- `logs, , team-a/, allow` – アプリケーションログにアクセスできます
- `exec, , team-a/, allow` – アプリケーションポッドへの実行アクセス権を持ちます

> **Note**
>
> EDITOR ユーザーは、更新できるプロジェクト内で、自分やそれ以外のユーザーにアクセス許可を付与するプロジェクトロールを作成できます。これにより、チームリーダーは ADMIN の介入を必要とせずに、チームのプロジェクトスコープのリソースへのアクセスを制御できます。

> **Note**
>
> `groups` フィールドでアイデンティティセンターグループ ID (グループ名ではない) を使用します。個別のユーザーアクセスにアイデンティティセンターユーザー ID を使用することもできます。AWS アイデンティティセンターコンソール内、または AWS CLI を使用してこれらの ID を検索します。

アクセス許可のよくあるパターン

パターン 1: 管理チームにフルアクセスを付与する

```
{
  "rbacRoleMappings": {
    "ADMIN": ["PlatformTeam", "SRETeam"]
  }
}
```

ADMIN ユーザーは、追加の設定なしで、すべてのプロジェクトスコープのリソースを表示および管理できます。

パターン 2: チームリーダーがプロジェクトを管理し、開発者がプロジェクトロール経由でアクセスする

```
{
  "rbacRoleMappings": {
    "ADMIN": ["PlatformTeam"],
    "EDITOR": ["TeamLeads"],
    "VIEWER": ["AllDevelopers"]
  }
}
```

1. ADMIN はチームごとにプロジェクトを作成する
2. チームリーダー (EDITOR) は、開発者にプロジェクトリソース (Application、ApplicationSet、認証情報) と機能 (ログ、実行) へのアクセスを許可するようにプロジェクトロールを設定する
3. 開発者 (VIEWER) は、プロジェクトロールで許可されているリソースと機能にのみアクセスできる

パターン 3: プロジェクトロールを使用したチームベースのアクセス

1. ADMIN はプロジェクトを作成し、チームリーダーを EDITOR ロールにグローバルにマッピングする
2. チームリーダー (EDITOR) は、チームメンバーをプロジェクト内のプロジェクトロールに割り当てる
3. チームメンバーには VIEWER グローバルロールのみが必要 – プロジェクトロールはプロジェクトリソースと機能へのアクセスを提供する

```
{
  "rbacRoleMappings": {
    "ADMIN": ["PlatformTeam"],
    "EDITOR": ["TeamLeads"],
    "VIEWER": ["AllDevelopers"]
  }
}
```

ベストプラクティス

個々のユーザーではなくグループを使用する: AWS アイデンティティセンターグループを個々のユーザーではなく Argo CD ロールにマッピングして、管理を容易にします。

最小特権から開始する: VIEWER アクセスから開始し、必要に応じて EDITOR または ADMIN を付与します。

Project を使用してチームを分離する: チームや環境ごとに AppProject を作成して、境界を確立します。

アイデンティティセンターのフェデレーションを活用する: 社内の ID プロバイダーとフェデレーションするように AWS アイデンティティセンターを設定して、ユーザー管理を一元化します。

アクセスを定期的に見直す: ロールのマッピングと Project の割り当てを定期的に見直して、適切なアクセスレベルを確保します。

クラスターへのアクセスを制限する: Argo CD RBAC は Argo CD リソースとオペレーションへのアクセスを制御しますが、Kubernetes RBAC には対応していないことに注意してください。Argo CD にアクセスできるユーザーは、Argo CD がアクセスできるクラスターにアプリケーションをデプロイできます。Argo CD がアクセスできるクラスターを制限し、Project の宛先制限を使用して、アプリケーションをデプロイできる場所を制御します。

AWS サービスのアクセス許可

Repository リソースを作成することなく Application リソースで AWS サービスを直接使用するには、必要な IAM アクセス許可を機能ロールにアタッチします。

ECR Helm チャート:

```
{
  "Version": "2012-10-17",
  "Statement": [
```

```
    {
      "Effect": "Allow",
      "Action": [
        "ecr:GetAuthorizationToken",
        "ecr:BatchCheckLayerAvailability",
        "ecr:GetDownloadUrlForLayer",
        "ecr:BatchGetImage"
      ],
      "Resource": "*"
    }
  ]
}
```

CodeCommit リポジトリ:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "codecommit:GitPull"
      ],
      "Resource": "arn:aws:codecommit:region:account-id:repository-name"
    }
  ]
}
```

CodeConnections (GitHub、GitLab、Bitbucket):

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "codeconnections:UseConnection"
      ],
      "Resource": "arn:aws:codeconnections:region:account-id:connection/connection-id"
    }
  ]
}
```

こうした統合の使用の詳細については、「the section called “リポジトリを設定する”」を参照してください。

次のステップ

- the section called “Argo CD の使用” - アプリケーションを作成してデプロイを管理する方法を学ぶ
- the section called “Argo CD の概念” - Project など Argo CD の概念を理解する
- the section called “EKS 機能に関する考慮事項” - 機能のセキュリティのベストプラクティスを見直す

Argo CD の使用

Argo CD では、Git リポジトリでアプリケーションを定義し、Argo CD はそれらを Kubernetes クラスターに自動的に同期します。これにより、自動ドリフト検出による宣言型のバージョン管理されたアプリケーションのデプロイが可能になります。

前提条件

Argo CD を使用する前に、以下が必要です。

- EKS クラスターに Argo CD 機能が作成されていること (「the section called “Argo CD 機能を作成する”」を参照)
- Git リポジトリに Kubernetes マニフェストが含まれていること
- クラスターと通信できるように `kubectl` が設定されていること

一般的なタスク

以下のトピックでは、一般的な Argo CD タスクについて説明します。

the section called “リポジトリを設定する” – AWS Secrets Manager、AWS CodeConnections、または Kubernetes Secrets を使用して、Git リポジトリにアクセスするように Argo CD を設定します。

the section called “クラスターを登録する” – Argo CD がアプリケーションをデプロイするターゲットクラスターを登録します。

the section called “プロジェクト” – マルチテナント環境向けにプロジェクトを使用してアプリケーションを整理し、セキュリティ境界を適用します。

the section called “設定を構成する” – カスタムリソースのカスタムヘルスチェックなど、`argocd-cm` ConfigMap を使用して Argo CD の動作を設定します。

the section called “Application を作成する” – 自動または手動の同期ポリシーを使用して Git リポジトリからデプロイするアプリケーションを作成します。

the section called “ApplicationSet” – テンプレートとジェネレーターを利用し、複数の環境またはクラスターにアプリケーションをデプロイするために ApplicationSets を使用します。

Argo CD UI にアクセスする

EKS コンソールから Argo CD UI にアクセスします。

1. Amazon EKS コンソールを開きます。
2. クラスターを選択します。
3. [機能] タブを選択します。
4. [Argo CD] を選択します。
5. [Argo CD UI を開く] を選択します。

UI は、アプリケーションのトポロジを視覚的に表示し、同期のステータスと履歴、リソースのヘルスとイベント、手動同期コントロール、アプリケーション管理を提供します。

Argo CD エンドポイント URL

Argo CD 機能を作成すると、Argo CD UI と API の両方に対して一意のエンドポイントを受け取ります。エンドポイントのホスト名は決定論的です。機能名、クラスター名、AWS アカウント ID、AWS リージョンなど、既に知っている値を使用します。機能を削除し、同じクラスター名と機能名で再作成すると、同じエンドポイントが得られます。ホスト名は予測可能であるため、事前に構築できます。例えば、機能のプロビジョニングが完了する前に、DNS、オートメーション、または Git Webhook 統合を事前に構成できます。どの機能でも、`describe-capability` の `serverUrl` 値が正式なエンドポイントです。

> **Note**
>
> エンドポイントが再利用されるため、以前の機能を指していた統合 (Git Webhook、トークン、DNS レコード) は、インスタンスが変更されたことを示すことなく、新しい機能に自動的に接続します。機能を再作成した後は、これらの統合を再確認し、以前のインスタンスと共有していた Webhook シークレットまたはトークンをローテーションしてください。

エンドポイントを取得する

機能の正確なエンドポイントを取得するには、describe-capability コマンドを使用し、serverUrl 値を読み取ります。*region-code*、*my-cluster*、および *my-argocd* を実際の値に置き換えます。

```
aws eks describe-capability \
  --region region-code \
  --cluster-name my-cluster \
  --capability-name my-argocd \
  --query 'capability.configuration.argoCd.serverUrl' \
  --output text
```

> **Note**
>
> serverUrl フィールドとエンドポイントの DNS レコードは、機能のステータスが ACTIVE に変わると使用可能になります。

エンドポイントフォーマット

Argo CD エンドポイントは次の形式を使用します。

```
https://capability-name-hash-account-id.eks-capabilities.region-code.amazonaws.com
```

ホスト名は次のコンポーネントで構成されます。

- *capability-name* – Argo CD 機能の名前。小文字に正規化され、各アンダースコア (_) がハイフン (-) に置き換えられます。正規化された名前が 41 文字を超える場合は、41 文字に切り詰められます。
- *hash* – ホスト名を一意にする 8 文字の 16 進数値。これは、機能名とクラスター名の SHA-256 ハッシュの最初の 8 文字です。名前はフォワードスラッシュ (*capability-name/cluster-name*) で結合されます。機能の作成時に指定した正確な名前を使用してください。この値は一意性を提供するだけであり、セキュリティ制御ではありません。
- *account-id* – 12 桁の AWS アカウント ID。
- eks-capabilities.*region-code*.amazonaws.com – AWS リージョンの EKS 機能サービスドメイン。

例えば、us-west-2 リージョンのアカウント 111122223333 にあるクラスター my-cluster 上の my-argocd という名前の機能には、次のようなエンドポイントがあります。

```
https://my-argocd-dc855fdf-111122223333.eks-capabilities.us-west-2.amazonaws.com
```

ホスト名は正規化された機能名を使用しますが、ハッシュは指定した未加工の名前を使用します。例えば、クラスター my-cluster 上の機能 Payments_GitOps はホスト名では payments-gitops に正規化され、そのハッシュは未加工の文字列 Payments_GitOps/my-cluster から生成されます。

```
https://payments-gitops-4ce0b382-111122223333.eks-capabilities.us-west-2.amazonaws.com
```

> **Note**
>
> AWS リージョンの正確なエンドポイントを確認するには、describe-capability 出力を使用します。

hash コンポーネントは自分で計算できます。Linux では sha256sum を、macOS では shasum -a 256 を使用します。次のコマンドでは、クラスター my-cluster 上の機能 my-argocd の例を使用します。機能の作成時に指定した正確な名前に置き換えてください。

```
# Linux
printf '%s' "my-argocd/my-cluster" | sha256sum | cut -c1-8

# macOS
printf '%s' "my-argocd/my-cluster" | shasum -a 256 | cut -c1-8
```

どちらのコマンドも dc855fdf を返します。

エンドポイントのアクセスとセキュリティ

エンドポイントのホスト名はシークレットではなく、それを知っているからといって Argo CD インスタンスへのアクセスが許可されるわけではありません。ホスト名は、機能の Amazon リソースネーム (ARN) に既に含まれているのと同じ非シークレット情報を公開します。create-capability、describe-capability、および list-capabilities の操作はこの ARN を返します。AWSアイデンティティセンター認証と Argo CD のロールベースアクセスコントロール (RBAC) が、Argo CD UI と API へのアクセスを制御します。

プライベートエンドポイントアクセスを設定すると、ホスト名は VPC エンドポイントに関連付けられたプライベート IP アドレスにのみ解決されます。エンドポイントには、VPC または接続されたネットワークからのみアクセスできます。詳細については、「the section called “プライベートエンドポイントアクセス”」を参照してください。

アップストリームドキュメント

Argo CD 機能の詳細については、以下を参照してください。

- Argo CD ドキュメント – 完全なユーザーガイド
- アプリケーション仕様 – フルアプリケーション API リファレンス
- ApplicationSet ガイド – ApplicationSet のパターンと例
- Argo CD GitHub – ソースコードと例

リポジトリアクセスを設定する

アプリケーションをデプロイする前に、Git リポジトリおよび Helm チャートレジストリにアクセスするように Argo CD を設定します。Argo CD は、GitHub、GitLab、Bitbucket、AWS CodeCommit、AWS ECR に対応するために複数の認証方法をサポートしています。

> **Note**
>
> 直接 AWS サービス統合 (ECR Helm チャート、CodeCommit リポジトリ、CodeConnections) の場合、リポジトリ設定を作成しなくても、Application リソースで直接参照できます。機能ロールには、IAM アクセス許可が必要です。詳細については、「the section called “アクセス許可の設定”」を参照してください。

前提条件

- EKS クラスターに Argo CD 機能が作成されていること
- Git リポジトリに Kubernetes マニフェストが含まれていること
- クラスターと通信できるように `kubectl` が設定されていること

> **Note**
>
> AWS CodeConnections は、AWS クラウドまたはオンプレミスにある Git サーバーに接続できます。詳細については、「AWS CodeConnections」を参照してください。

認証方法

Method	ユースケース	必要な IAM アクセス許可
AWS サービスとの直接統合		
CodeCommit	AWS CodeCommit Git リポジトリとの直接統合。リポジトリ設定は不要です。	`codecommit:GitPull`
CodeConnections	マネージド認証で GitHub、GitLab、または Bitbucket に接続します。接続のセットアップが必要です。	`codeconnections:UseConnection`
ECR OCI アーティファクト	AWS ECR for OCI Helm チャートとマニフェスト画像の直接統合。リポジトリ設定は不要です。	`arn:aws:iam::aws:policy/AmazonEC2ContainerRegistryPullOnly`
認証情報によるリポジトリ設定		
AWS Secrets Manager (ユーザー名/トークン)	個人用アクセストークンまたはパスワードを保存します。Kubernetes アクセスなしで認証情報のローテーションを有効にします。	`arn:aws:iam::aws:policy/AWSSecretsManagerClientReadOnlyAccess`
AWS Secrets Manager (SSH キー)	SSH キー認証を使用します。Kubernetes アクセスなしで認証情報のローテーションを有効にします。	`arn:aws:iam::aws:policy/AWSSecretsManagerClientReadOnlyAccess`
AWS Secrets Manager (GitHub アプリ)	プライベートキーによる GitHub アプリの認証。Kubernetes アクセスなし	`arn:aws:iam::aws:policy/AWSSecretsManagerClientReadOnlyAccess`

Method	ユースケース	必要な IAM アクセス許可
	で認証情報のローテーションを有効にします。	
Kubernetes シークレット	クラスター内シークレットを使用した標準の Argo CD 方法	なし (Kubernetes RBAC を使用した EKS アクセスエントリによって処理されるアクセス許可)

AWS サービスへの直接アクセス

AWS サービスの場合、リポジトリ設定を作成しなくても、Application リソースで直接参照できます。機能ロールには、IAM アクセス許可が必要です。

CodeCommit リポジトリ

Application 内で直接 CodeCommit リポジトリを参照します。

```
apiVersion: argoproj.io/v1alpha1
kind: Application
metadata:
  name: my-app
  namespace: argocd
spec:
  source:
    repoURL: https://git-codecommit.region.amazonaws.com/v1/repos/repository-name
    targetRevision: main
    path: kubernetes/manifests
```

必要な機能ロールアクセス許可:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "codecommit:GitPull",
      "Resource": "arn:aws:codecommit:region:account-id:repository-name"
    }
  ]
}
```

CodeConnections

CodeConnections を介して GitHub、GitLab、Bitbucket のリポジトリを参照します。リポジトリ URL 形式は、CodeConnections 接続 ARN から取得されます。

リポジトリ URL 形式は、次のとおりです。

```
apiVersion: argoproj.io/v1alpha1
kind: Application
metadata:
  name: my-app
  namespace: argocd
spec:
  source:
    repoURL: https://codeconnections.region.amazonaws.com/git-http/account-
id/region/connection-id/owner/repository.git
    targetRevision: main
    path: kubernetes/manifests
```

必要な機能ロールアクセス許可:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "codeconnections:UseConnection",
      "Resource": "arn:aws:codeconnections:region:account-id:connection/connection-id"
    }
  ]
}
```

ECR Helm チャート

ECR は、Helm チャートを OCI アーティファクトとして保存します。Argo CD では、次の 2 つの方法でそれらを参照できます。

Helm 形式 (Helm チャートの場合に推奨):

```
apiVersion: argoproj.io/v1alpha1
kind: Application
metadata:
  name: my-app-helm
```

```
  namespace: argocd
spec:
  source:
    repoURL: account-id.dkr.ecr.region.amazonaws.com/repository-name
    targetRevision: chart-version
    chart: chart-name
    helm:
      valueFiles:
        - values.yaml
```

注意: Helm 形式を使用するときは、oci:// プレフィックスを含めないでください。chart フィールドにチャート名を指定します。

OCI 形式 (Kubernetes マニフェストを含む OCI アーティファクトの場合):

```
apiVersion: argoproj.io/v1alpha1
kind: Application
metadata:
  name: my-app-oci
  namespace: argocd
spec:
  source:
    repoURL: oci://account-id.dkr.ecr.region.amazonaws.com/repository-name
    targetRevision: artifact-version
    path: path-to-manifests
```

注意: OCI 形式を使用するときは、oci:// プレフィックスを含めてください。chart ではなく path フィールドを使用します。

必要な機能ロールのアクセス許可 – マネージドポリシーをアタッチします。

```
arn:aws:iam::aws:policy/AmazonEC2ContainerRegistryPullOnly
```

このポリシーには、必要な ECR アクセス許可である ecr:GetAuthorizationToken、ecr:BatchGetImage、および ecr:GetDownloadUrlForLayer が含まれます。

AWS Secrets Manager を使用する

Secrets Manager にリポジトリ認証情報を保存して、Argo CD リポジトリ設定で参照します。Secrets Manager を使用すると、Kubernetes RBAC アクセスを必要とせずに認証情報の自動

ローテーションが有効になります。認証情報は Secrets Manager への IAM アクセス許可を使用してローテーションでき、Argo CD は更新された値を自動的に読み取ります。

> **Note**
>
> 複数のリポジトリ (GitHub 組織内のすべてのリポジトリなど) で認証情報を再利用するには、`argocd.argoproj.io/secret-type: repo-creds` でリポジトリ認証情報テンプレートを使用します。これにより、個別のリポジトリシークレットを作成するよりも UX が向上します。詳細については、Argo CD ドキュメントの「Repository Credentials」を参照してください。

ユーザー名とトークンの認証

HTTPS リポジトリに個人用アクセストークンやパスワードが含まれている場合:

Secrets Manager にシークレットを作成する:

```
aws secretsmanager create-secret \
  --name argocd/my-repo \
  --description "GitHub credentials for Argo CD" \
  --secret-string '{"username":"your-username","token":"your-personal-access-token"}'
```

オプションの TLS クライアント証明書フィールド (プライベート Git サーバーの場合):

```
aws secretsmanager create-secret \
  --name argocd/my-private-repo \
  --secret-string '{
    "username":"your-username",
    "token":"your-token",

 "tlsClientCertData":"LS0tLS1CRUdJTiBDRVJUSUZJQ0FURS0tLS0tCi4uLgotLS0tLUVORCBDRVJUSUZJQ0FURS0tL

 "tlsClientCertKey":"LS0tLS1CRUdJTiBQUklWQVRFIEtFWS0tLS0tCi4uLgotLS0tLUVORCBQUklWQVRFIEtFWS0tLS
  }'
```

> **Note**
>
> `tlsClientCertData` と `tlsClientCertKey` の値は、Base64 でエンコードされている必要があります。

Secrets Manager を参照するリポジトリシークレットを作成する:

```
apiVersion: v1
kind: Secret
metadata:
  name: my-repo
  namespace: argocd
  labels:
    argocd.argoproj.io/secret-type: repository
stringData:
  type: git
  url: https://github.com/your-org/your-repo
  secretArn: arn:aws:secretsmanager:us-west-2:111122223333:secret:argocd/my-repo-AbCdEf
  project: default
```

SSH キー認証

SSH ベースの Git アクセスの場合、プライベートキーを JSON ではなくプレーンテキストとして保存します。

SSH プライベートキーでシークレットを作成する:

```
aws secretsmanager create-secret \
  --name argocd/my-repo-ssh \
  --description "SSH key for Argo CD" \
  --secret-string "-----BEGIN OPENSSH PRIVATE KEY-----
b3BlbnNzaC1rZXktdjEAAAAABG5vbmUAAAAEbm9uZQAAAAAAAAABAAABlwAAAAdzc2gtcn
...
-----END OPENSSH PRIVATE KEY-----"
```

SSH 用にリポジトリシークレットを作成する:

```
apiVersion: v1
kind: Secret
metadata:
  name: my-repo-ssh
  namespace: argocd
  labels:
    argocd.argoproj.io/secret-type: repository
stringData:
  type: git
  url: git@github.com:your-org/your-repo.git
```

```
  secretArn: arn:aws:secretsmanager:us-west-2:111122223333:secret:argocd/my-repo-ssh-
AbCdEf
  project: default
```

GitHub アプリ認証

プライベートキーによる GitHub アプリ認証の場合:

GitHub アプリ認証情報でシークレットを作成する:

```
aws secretsmanager create-secret \
  --name argocd/github-app \
  --description "GitHub App credentials for Argo CD" \
  --secret-string '{

 "githubAppPrivateKeySecret":"LS0tLS1CRUdJTiBSU0EgUFJJVkFURSBLRVktLS0tLQouLi4KLS0tLS1FTkQgUlNBI
    "githubAppID":"123456",
    "githubAppInstallationID":"12345678"
  }'
```

> **Note**
>
> `githubAppPrivateKeySecret` 値は、Base64 でエンコードされている必要があります。

GitHub Enterprise のオプションのフィールド:

```
aws secretsmanager create-secret \
  --name argocd/github-enterprise-app \
  --secret-string '{

 "githubAppPrivateKeySecret":"LS0tLS1CRUdJTiBSU0EgUFJJVkFURSBLRVktLS0tLQouLi4KLS0tLS1FTkQgUlNBI
    "githubAppID":"123456",
    "githubAppInstallationID":"12345678",
    "githubAppEnterpriseBaseUrl":"https://github.example.com/api/v3"
  }'
```

GitHub アプリ用にリポジトリシークレットを作成する:

```
apiVersion: v1
```

```
kind: Secret
metadata:
  name: my-repo-github-app
  namespace: argocd
  labels:
    argocd.argoproj.io/secret-type: repository
stringData:
  type: git
  url: https://github.com/your-org/your-repo
  secretArn: arn:aws:secretsmanager:us-west-2:111122223333:secret:argocd/github-app-
AbCdEf
  project: default
```

リポジトリ認証情報テンプレート

複数のリポジトリ (GitHub 組織またはユーザーのすべてのリポジトリなど) で認証情報を再利用するには、argocd.argoproj.io/secret-type: repo-creds でリポジトリ認証情報テンプレートを使用します。これにより、リポジトリごとに個別のリポジトリシークレットを作成するよりも UX が向上します。

リポジトリ認証情報テンプレートを作成します。

```
apiVersion: v1
kind: Secret
metadata:
  name: github-org-creds
  namespace: argocd
  labels:
    argocd.argoproj.io/secret-type: repo-creds
stringData:
  type: git
  url: https://github.com/your-org
  secretArn: arn:aws:secretsmanager:us-west-2:111122223333:secret:argocd/github-org-
AbCdEf
```

この認証情報テンプレートは、URL プレフィックス https://github.com/your-org に一致するすべてのリポジトリに適用されます。その後、追加のシークレットを作成せずに、この組織内の任意のリポジトリを Application で参照できます。

詳細については、Argo CD ドキュメントの「Repository Credentials」を参照してください。

> **Important**
>
> IAM 機能ロールにマネージドポリシー `arn:aws:iam::aws:policy/AWSSecretsManagerClientReadOnlyAccess` がアタッチされているか、`secretsmanager:GetSecretValue` および KMS 復号アクセス許可を含む同等のアクセス許可がアタッチされていることを確認します。IAM ポリシーの設定については、「the section called “Argo CD に関する考慮事項”」を参照してください。

AWS CodeConnections を使用する

CodeConnections 統合については、「the section called “ AWS CodeConnections”」を参照してください。

CodeConnections のマネージド認証により、認証情報を保存することなく GitHub、GitLab、Bitbucket の認証を行うことができます。

Kubernetes シークレットを使用する

Argo CD の標準の方法を使用して、認証情報を Kubernetes に直接保存します。

HTTPS で個人用アクセストークンを使用する場合:

```
apiVersion: v1
kind: Secret
metadata:
  name: my-repo
  namespace: argocd
  labels:
    argocd.argoproj.io/secret-type: repository
stringData:
  type: git
  url: https://github.com/your-org/your-repo
  username: your-username
  password: your-personal-access-token
```

SSH の場合:

```
apiVersion: v1
kind: Secret
metadata:
  name: my-repo-ssh
```

```
  namespace: argocd
  labels:
    argocd.argoproj.io/secret-type: repository
stringData:
  type: git
  url: git@github.com:your-org/your-repo.git
  sshPrivateKey: |
    -----BEGIN OPENSSH PRIVATE KEY-----
    ... your private key ...
    -----END OPENSSH PRIVATE KEY-----
```

CodeCommit リポジトリ

AWS CodeCommit の場合、IAM 機能ロールに CodeCommit アクセス許可を付与します (`codecommit:GitPull`)。

リポジトリを設定します。

```
apiVersion: v1
kind: Secret
metadata:
  name: codecommit-repo
  namespace: argocd
  labels:
    argocd.argoproj.io/secret-type: repository
stringData:
  type: git
  url: https://git-codecommit.us-west-2.amazonaws.com/v1/repos/my-repo
  project: default
```

IAM ポリシー設定の詳細については、「the section called “Argo CD に関する考慮事項”」を参照してください。

リポジトリ接続を検証する

Argo CD UI の [設定] → [リポジトリ] から接続ステータスを確認します。UI には、接続ステータスと、認証エラーがあればそれも表示されます。

リポジトリシークレットにステータス情報は含まれません。

その他のリソース

- the section called “クラスターを登録する” - デプロイ用にターゲットクラスターを登録する

- the section called “Application を作成する” - 最初の Application を作成する
- the section called “Argo CD に関する考慮事項” - IAM アクセス許可とセキュリティ設定
- プライベートリポジトリ - アップストリームリポジトリ設定リファレンス

ターゲットクラスターを登録する

クラスターを登録すると、Argo CD がアプリケーションをそのクラスターにデプロイできるようになります。Argo CD が動作しているのと同じクラスター (ローカルクラスター) を登録することも、別のアカウントやリージョンのリモートクラスターを登録することもできます。クラスターが新規登録されると、そのクラスター内にアプリケーションが作成されるまで、クラスターの接続ステータスは「不明」のままになります。クラスターの登録後に Argo CD アプリケーションを作成するには、「the section called “Application を作成する”」を参照してください。

前提条件

- EKS クラスターに Argo CD 機能が作成されていること
- クラスターと通信できるように `kubectl` が設定されていること
- リモートクラスターの場合: 適切な IAM アクセス許可およびアクセスエントリ

ローカルクラスターを登録する

Argo CD が動作しているのと同じクラスターにアプリケーションをデプロイするには、そのクラスターをデプロイターゲットとして登録します。

> **Important**
>
> Argo CD 機能は、ローカルクラスターを自動的には登録しません。アプリケーションを同じクラスターにデプロイするには、そのクラスターを明示的に登録する必要があります。クラスター名 `in-cluster` を使用して、ほとんどのオンラインの Argo CD の例と互換性を持たせることができます。

> **Note**
>
> EKS アクセスエントリは、Argo CD 機能ロールを持つローカルクラスターに対して自動的に作成されますが、デフォルトでは Kubernetes RBAC アクセス許可は付与されません。これは最小特権の原則に従います。ユースケースに基づいて Argo CD に必要なアクセス許可を明

示的に設定する必要があります。例えば、このクラスターを Argo CD ハブとしてのみ使用してリモートクラスターを管理する場合、ローカルデプロイのアクセス許可は必要ないはずです。設定オプションについては、以下のアクセスエントリ RBAC 要件セクションを参照してください。

Argo CD CLI を使用する:

```
argocd cluster add <cluster-context-name> \
  --aws-cluster-name arn:aws:eks:us-west-2:111122223333:cluster/my-cluster \
  --name local-cluster
```

Kubernetes シークレットを使用する:

```
apiVersion: v1
kind: Secret
metadata:
  name: local-cluster
  namespace: argocd
  labels:
    argocd.argoproj.io/secret-type: cluster
stringData:
  name: local-cluster
  server: arn:aws:eks:us-west-2:111122223333:cluster/my-cluster
  project: default
```

以下のとおり設定を適用します。

```
kubectl apply -f local-cluster.yaml
```

Note

Kubernetes API サーバー URL ではなく、`server` フィールドで EKS クラスター ARN を使用します。マネージド機能では、ARN でクラスターを識別する必要があります。デフォルトの `kubernetes.default.svc` はサポートされていません。

リモートクラスターを登録する

リモートクラスターにデプロイするには:

ステップ 1: リモートクラスターにアクセスエントリを作成する

region-code はリモートクラスターがある AWS リージョンに置き換え、*remote-cluster* はリモートクラスターの名前に置き換え、ARN は Argo CD 機能ロール ARN に置き換えます。

```
aws eks create-access-entry \
  --region region-code \
  --cluster-name remote-cluster \
  --principal-arn arn:aws:iam::111122223333:role/ArgoCDCapabilityRole \
  --type STANDARD
```

ステップ 2: アクセスポリシーを Kubernetes RBAC アクセス許可に関連付ける

アクセスエントリには、Argo CD がアプリケーションをデプロイするための Kubernetes RBAC アクセス許可が必要です。すぐに開始するには、AmazonEKSClusterAdminPolicy を使用できます。

```
aws eks associate-access-policy \
  --region region-code \
  --cluster-name remote-cluster \
  --principal-arn arn:aws:iam::111122223333:role/ArgoCDCapabilityRole \
  --policy-arn arn:aws:eks::aws:cluster-access-policy/AmazonEKSClusterAdminPolicy \
  --access-scope type=cluster
```

Important

AmazonEKSClusterAdminPolicy は、完全なクラスター管理者アクセスを提供します (system:masters に相当)。これは開始するのに便利ですが、本番環境では使用しないでください。本番環境では、アクセスエントリをカスタム Kubernetes グループに関連付け、適切なロールバインディングまたはクラスターロールバインディングを作成することで、より制限の厳しいアクセス許可を使用してください。最小特権の設定については、以下の本番環境設定セクションを参照してください。

ステップ 3: Argo CD にクラスターを登録する

Argo CD CLI を使用する:

```
argocd cluster add <cluster-context-name> \
  --aws-cluster-name arn:aws:eks:us-west-2:111122223333:cluster/remote-cluster \
  --name remote-cluster
```

Kubernetes シークレットを使用する:

```
apiVersion: v1
kind: Secret
metadata:
  name: remote-cluster
  namespace: argocd
  labels:
    argocd.argoproj.io/secret-type: cluster
stringData:
  name: remote-cluster
  server: arn:aws:eks:us-west-2:111122223333:cluster/remote-cluster
  project: default
```

以下のとおり設定を適用します。

```
kubectl apply -f remote-cluster.yaml
```

クロスアカウントクラスター

さまざまな AWS アカウント内のクラスターにデプロイするには:

1. ターゲットアカウントで、ソースアカウントの Argo CD IAM 機能ロール ARN をプリンシパルとして使用して、ターゲット EKS クラスター上にアクセスエントリを作成します。
2. アクセスポリシーを適切な Kubernetes RBAC アクセス許可に関連付けします。
3. EKS クラスター ARN を使用して Argo CD にクラスターを登録します。

追加で IAM ロールを作成したり、信頼ポリシーを設定したりする必要はありません。EKS アクセスエントリがクロスアカウントアクセスを処理します。

クラスター ARN 形式にはリージョンが含まれているため、クロスリージョンデプロイでは同じリージョンデプロイと同じプロセスが使用されます。

クラスター登録を検証する

登録済みのクラスターを表示します。

```
kubectl get secrets -n argocd -l argocd.argoproj.io/secret-type=cluster
```

あるいは、Argo CD UI の [設定] → [クラスター] でクラスターステータスを確認します。

プライベートクラスター

Argo CD 機能では、VPC ピアリングや特殊なネットワーク設定を施すことなく、完全にプライベートな EKS クラスターに透過的にアクセスできます。

AWS は、Argo CD 機能とプライベートリモートクラスターとの接続を自動的に管理します。

単に ARN を使用してプライベートクラスターを登録するだけで、別途ネットワークをセットアップする必要はありません。

アクセスエントリの RBAC 要件

Argo CD 機能を作成すると、機能ロールに対して EKS アクセスエントリが自動的に作成されますが、デフォルトでは Kubernetes RBAC アクセス許可は付与されません。この設計は、最小特権の原則に従った意図的なものです。異なるユースケースには、異なるアクセス許可が必要となります。

例: *クラスターを Argo CD ハブとしてのみ使用してリモートクラスターを管理する場合、ローカルデプロイのアクセス許可は必要ありません。*アプリケーションをローカルにデプロイする場合は、クラスター全体の読み取りアクセスと特定の名前空間への書き込みアクセスが必要です。*CRD を作成する必要がある場合は、追加のクラスター管理者アクセス許可が必要です。

要件に基づいて Argo CD に必要なアクセス許可を明示的に設定する必要があります。

Argo CD の最小アクセス許可

Argo CD には、エラーなしで機能するために 2 種類のアクセス許可が必要です。

読み取りアクセス許可 (クラスター全体): Argo CD は以下のために、クラスター全体のすべてのリソースタイプとカスタムリソース定義 (CRD) を読み取ることができる必要があります。

- リソース検出とヘルスチェック
- 目的の状態と実際の状態の間のドリフトの検出
- デプロイ前のリソースの検証

書き込みアクセス許可 (名前空間固有): Argo CD には、Application で定義されたリソースの作成、更新、削除のアクセス許可が必要です。

- アプリケーションワークロード (デプロイ、サービス、ConfigMaps など) をデプロイする
- カスタムリソースの適用 (アプリケーション固有の CRD)
- アプリケーションのライフサイクルを管理する

Quick Setup

迅速な開始、テスト、または開発環境については、AmazonEKSClusterAdminPolicy を使用します。

```
aws eks associate-access-policy \
  --region region-code \
  --cluster-name my-cluster \
  --principal-arn arn:aws:iam::111122223333:role/ArgoCDCapabilityRole \
  --policy-arn arn:aws:eks::aws:cluster-access-policy/AmazonEKSClusterAdminPolicy \
  --access-scope type=cluster
```

> **Important**
>
> AmazonEKSClusterAdminPolicy は、完全なクラスター管理者アクセス (system:masters に相当) を提供します。これには、CRD の作成、クラスター全体のリソースの変更、任意の名前空間へのデプロイが含まれます。これは開発や POC には便利ですが、本番環境では使用しないでください。本番環境では、以下の最小特権設定を使用します。

最小特権原則に基づく本番環境設定

本番環境では、以下を許可するカスタム Kubernetes RBAC を作成します。

- すべてのリソースへのクラスター全体の読み取りアクセス (検出とヘルスチェック用)
- 名前空間固有の書き込みアクセス (デプロイ用)

ステップ 1: アクセスエントリをカスタム Kubernetes グループに関連付ける

```
aws eks associate-access-policy \
  --region region-code \
  --cluster-name my-cluster \
  --principal-arn arn:aws:iam::111122223333:role/ArgoCDCapabilityRole \
  --policy-arn arn:aws:eks::aws:cluster-access-policy/AmazonEKSEditPolicy \
  --access-scope type=namespace,namespaces=app-namespace
```

ステップ 2: 読み取りアクセス用のクラスターロールを作成する

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: argocd-read-all
rules:
# Read access to all resources for discovery and health checks
- apiGroups: ["*"]
  resources: ["*"]
  verbs: ["get", "list", "watch"]
```

ステップ 3: アプリケーション名前空間への書き込みアクセス用ロールを作成する

```
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  name: argocd-deploy
  namespace: app-namespace
rules:
# Full access to deploy application resources
- apiGroups: ["*"]
  resources: ["*"]
  verbs: ["*"]
```

ステップ 4: ロールを Kubernetes グループにバインドする

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: argocd-read-all
subjects:
- kind: Group
  name: eks-access-entry:arn:aws:iam::111122223333:role/ArgoCDCapabilityRole
  apiGroup: rbac.authorization.k8s.io
roleRef:
  kind: ClusterRole
  name: argocd-read-all
  apiGroup: rbac.authorization.k8s.io
---
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: argocd-deploy
```

```
  namespace: app-namespace
subjects:
- kind: Group
  name: eks-access-entry:arn:aws:iam::111122223333:role/ArgoCDCapabilityRole
  apiGroup: rbac.authorization.k8s.io
roleRef:
  kind: Role
  name: argocd-deploy
  apiGroup: rbac.authorization.k8s.io
```

> **Note**
>
> アクセスエントリのグループ名形式は、`eks-access-entry:` の後にプリンシパル ARN が続きます。Argo CD がアプリケーションをデプロイする名前空間ごとに、このロールバインド手順を繰り返します。

> **Important**
>
> Argo CD は、特定の名前空間にのみデプロイする場合でも、ヘルスチェックと検出のためにクラスター全体のすべてのリソースタイプを読み取ることができる必要があります。クラスター全体の読み取りアクセスがないと、アプリケーションの状態をチェックするときに Argo CD にエラーが表示されます。

Project でクラスターアクセスを制限する

プロジェクトを使用して、`spec.destinations` で許可されるターゲットクラスターと名前空間を設定することで、Application がデプロイできるクラスターと名前空間を制御します。

```
apiVersion: argoproj.io/v1alpha1
kind: AppProject
metadata:
  name: production
  namespace: argocd
spec:
  destinations:
  - server: arn:aws:eks:us-west-2:111122223333:cluster/prod-cluster
    namespace: '*'
  - server: arn:aws:eks:eu-west-1:111122223333:cluster/prod-eu-cluster
```

```
    namespace: '*'
  sourceRepos:
  - 'https://github.com/example/production-apps'
```

詳細については、「the section called “プロジェクト”」を参照してください。

その他のリソース

- the section called “プロジェクト” - Application を整理し、セキュリティ境界を確立する
- the section called “Application を作成する” - 最初のアプリケーションをデプロイする
- the section called “ApplicationSet” - ApplicationSet で複数のクラスターにデプロイする
- the section called “Argo CD に関する考慮事項” - マルチクラスターパターンとクロスアカウントセットアップ
- 宣言型クラスターセットアップ - アップストリームクラスター設定リファレンス

Argo CD プロジェクトを操作する

Argo CD プロジェクト (AppProject) では、アプリケーションを論理的にグループ化して、アクセスコントロールを付与できます。プロジェクトは、アプリケーションが使用できる Git リポジトリ、ターゲットクラスター、名前空間を定義して、共有 Argo CD インスタンスにマルチテナンシーとセキュリティの境界を確立します。

どのような場合にプロジェクトを使用するか

プロジェクトを使用すると、以下のことが行えます。

- チーム、環境、またはビジネスユニットごとにアプリケーションを分離します。
- チームがどのリポジトリからデプロイできるかを制限します。
- チームがどのクラスターと名前空間にデプロイできるかを制限します。
- リソースクォータと許可されたリソースタイプを適用します。
- ガードレールに沿ってセルフサービスアプリケーションをデプロイします。

デフォルトプロジェクト

すべての Argo CD 機能に `default` プロジェクトがあり、すべてのリポジトリ、クラスター、名前空間へのアクセスを許可する設定になっています。初期テストには有用ですが、本番稼働に使用する場合は制限を明記した専用のプロジェクトを作成します。

デフォルトプロジェクトの設定とその制限方法の詳細については、Argo CD ドキュメントの「デフォルトプロジェクト」を参照してください。

プロジェクトの作成

プロジェクトを作成するには、クラスターに AppProject リソースを適用します。

例: チーム固有のプロジェクト

```
apiVersion: argoproj.io/v1alpha1
kind: AppProject
metadata:
  name: team-a
  namespace: argocd
spec:
  description: Applications for Team A

  # Source repositories this project can deploy from
  sourceRepos:
    - 'https://github.com/my-org/team-a-*'
    - 'https://github.com/my-org/shared-libs'


  # Source namespaces (required for EKS capability)
  sourceNamespaces:
    - argocd
    - team-a-dev
    - team-a-prod

  # Destination clusters and namespaces
  destinations:
    - name: dev-cluster
      namespace: team-a-dev
    - name: prod-cluster
      namespace: team-a-prod

  # Allowed resource types
  clusterResourceWhitelist:
    - group: ''
      kind: Namespace

  namespaceResourceWhitelist:
    - group: 'apps'
      kind: Deployment
```

```
    - group: ''
      kind: Service
    - group: ''
      kind: ConfigMap
```

プロジェクトを適用します。

```
kubectl apply -f team-a-project.yaml
```

プロジェクトの設定

ソースリポジトリ

このプロジェクトのアプリケーションが使用できる Git リポジトリを制御します。

```
spec:
  sourceRepos:
    - 'https://github.com/my-org/app-*'  # Wildcard pattern
    - 'https://github.com/my-org/infra'  # Specific repo
```

ワイルドカードと否定パターン (! プレフィックス) を使用すると、特定のリポジトリを許可または拒否できます。詳細については、Argo CD ドキュメントの プロジェクトの管理 を参照してください。

ソース名前空間

EKS Argo CD 機能を使用する場合は、カスタム AppProject 定義に `spec.sourceNamespaces` フィールドが 必要 です。このフィールドは、このプロジェクトを参照するアプリケーションまたは ApplicationSet を含めることができる名前空間を指定します。

> **Important**
>
> これは EKS Argo CD 機能の必須フィールドです。このフィールドがオプションである OSS Argo CD とは異なります。

デフォルトの AppProject の動作

`default` AppProject には、`sourceNamespaces` の `argocd` 名前空間が自動的に含まれます。追加の名前空間でアプリケーションまたは ApplicationSets を作成する必要がある場合は、`sourceNamespaces` フィールドを変更してそれらの名前空間を追加します。

```
apiVersion: argoproj.io/v1alpha1
kind: AppProject
metadata:
  name: default
  namespace: argocd
spec:
  sourceNamespaces:
    - argocd           # Already included by default
    - team-a-apps      # Add additional namespaces as needed
    - team-b-apps
```

カスタム AppProject の設定

カスタム AppProject を作成するときは、argocd システム名前空間と、アプリケーションまたは ApplicationSets を作成する予定のその他の名前空間を手動で含める必要があります。

```
apiVersion: argoproj.io/v1alpha1
kind: AppProject
metadata:
  name: team-a-project
  namespace: argocd
spec:
  description: Applications for Team A

  # Required: Manually specify all namespaces
  sourceNamespaces:
    - argocd           # ArgoCD system namespace (required)
    - team-a-dev       # Custom namespace for dev Applications
    - team-a-prod      # Custom namespace for prod Applications

  # Source repositories this project can deploy from
  sourceRepos:
    - 'https://github.com/my-org/team-a-*'

  # Destination restrictions
  destinations:
    - namespace: 'team-a-*'
      server: arn:aws:eks:us-west-2:111122223333:cluster/my-cluster  # Use cluster ARN
 from: aws eks describe-cluster
```

> **Note**
>
> sourceNamespaces から名前空間を省略した場合、その名前空間で作成されたアプリケーションまたは ApplicationSets はこのプロジェクトを参照できなくなり、デプロイが失敗します。

送信先の制限

アプリケーションがデプロイできる場所を制限します。

```
spec:
  destinations:
    - name: prod-cluster  # Specific cluster by name
      namespace: production
    - name: '*'  # Any cluster
      namespace: team-a-*  # Namespace pattern
```

> **Important**
>
> 本番稼働のプロジェクトでは、ワイルドカードではなく、特定のクラスター名と名前空間パターンを使用します。これにより、不正なクラスターや名前空間が偶発的にデプロイされるのを防ぐことができます。

ワイルドカードと否定パターンを使用して、送信先を制御できます。詳細については、Argo CD ドキュメントの「プロジェクトの管理」を参照してください。

リソースの制限

どの Kubernetes リソースタイプをデプロイできるかを制御します。

クラスター範囲のリソース:

```
spec:
  clusterResourceWhitelist:
    - group: ''
      kind: Namespace
    - group: 'rbac.authorization.k8s.io'
```

```
      kind: Role
```

名前空間範囲のリソース:

```
spec:
  namespaceResourceWhitelist:
    - group: 'apps'
      kind: Deployment
    - group: ''
      kind: Service
    - group: ''
      kind: ConfigMap
    - group: 's3.services.k8s.aws'
      kind: Bucket
```

ブラックリストを使用して、特定のリソースを拒否します。

```
spec:
  namespaceResourceBlacklist:
    - group: ''
      kind: Secret  # Prevent direct Secret creation
```

プロジェクトにアプリケーションを割り当てる

アプリケーションを作成するときは、spec.project フィールドにプロジェクトを指定します。

```
apiVersion: argoproj.io/v1alpha1
kind: Application
metadata:
  name: my-app
  namespace: argocd
spec:
  project: team-a  # Assign to team-a project
  source:
    repoURL: https://github.com/my-org/my-app
    path: manifests
  destination:
    name: prod-cluster
    namespace: team-a-prod
```

アプリケーションにプロジェクトを指定しないと、default プロジェクトが使用されます。

プロジェクトのロールと RBAC

プロジェクトでは、カスタムロールを定義して、アクセスコントロールをきめ細かく設定できます。機能設定でプロジェクトのロールを AWS アイデンティティセンターのユーザーとグループにマッピングすると、アプリケーションを同期、更新、または削除できるユーザーを制御できます。

例: プロジェクトに開発者ロールと管理者ロールを付与する

```
apiVersion: argoproj.io/v1alpha1
kind: AppProject
metadata:
  name: team-a
  namespace: argocd
spec:
  sourceRepos:
    - '*'
  destinations:
    - name: '*'
      namespace: 'team-a-*'

  roles:
    - name: developer
      description: Developers can sync applications
      policies:
        - p, proj:team-a:developer, applications, sync, team-a/*, allow
        - p, proj:team-a:developer, applications, get, team-a/*, allow
      groups:
        - team-a-developers

    - name: admin
      description: Admins have full access
      policies:
        - p, proj:team-a:admin, applications, *, team-a/*, allow
      groups:
        - team-a-admins
```

プロジェクトのロール、CI/CD パイプラインの JWT トークン、および RBAC 設定の詳細については、Argo CD ドキュメントの「プロジェクトのロール」を参照してください。

一般的なパターン

環境ベースのプロジェクト

環境ごとにプロジェクトを作成します。

```
apiVersion: argoproj.io/v1alpha1
kind: AppProject
metadata:
  name: production
  namespace: argocd
spec:
  sourceRepos:
    - 'https://github.com/my-org/*'
  destinations:
    - name: prod-cluster
      namespace: '*'
  # Strict resource controls for production
  clusterResourceWhitelist: []
  namespaceResourceWhitelist:
    - group: 'apps'
      kind: Deployment
    - group: ''
      kind: Service
```

チームベースのプロジェクト

専用のプロジェクトでチームを分離します。

```
apiVersion: argoproj.io/v1alpha1
kind: AppProject
metadata:
  name: platform-team
  namespace: argocd
spec:
  sourceRepos:
    - 'https://github.com/my-org/platform-*'
  destinations:
    - name: '*'
      namespace: 'platform-*'
  # Platform team can manage cluster resources
  clusterResourceWhitelist:
    - group: '*'
```

```
      kind: '*'
```

マルチクラスターのプロジェクト

ポリシーに一貫性のある複数のクラスターにデプロイします。

```
apiVersion: argoproj.io/v1alpha1
kind: AppProject
metadata:
  name: global-app
  namespace: argocd
spec:
  sourceRepos:
    - 'https://github.com/my-org/global-app'
  destinations:
    - name: us-west-cluster
      namespace: app
    - name: eu-west-cluster
      namespace: app
    - name: ap-south-cluster
      namespace: app
```

ベストプラクティス

制限のあるプロジェクトから始める: 最初から幅広いアクセスを付与するのではなく、絞り込んだアクセス許可から始め、必要に応じて拡張します。

名前空間パターンを使用する: 名前空間の制限 (`team-a-*` など) にワイルドカードを活用して、境界を維持しながら柔軟性を確保します。

本番稼働用のプロジェクトを分離する: 本番稼働専用のプロジェクトを使用する場合は、制御を厳格にし、手動同期ポリシーを実施します。

プロジェクトの目的を文書化する: `description` フィールドを使用して、各プロジェクトの目的と誰が使用すべきかを説明します。

プロジェクトのアクセス許可を定期的に見直す: プロジェクトを定期的に監査して、制限がチームのニーズとセキュリティ要件に沿っていることを確認します。

その他のリソース

- the section called “アクセス許可の設定” - RBAC とアイデンティティセンターとの統合を設定する
- the section called “Application を作成する” - プロジェクト内にアプリケーションを作成する

- the section called “ApplicationSet” - プロジェクトで ApplicationSet を使用してマルチクラスターデプロイを実施する
- Argo CD プロジェクトドキュメント - 詳細なアップストリームリファレンス

Argo CD 設定を構成する

Argo CD の EKS 機能は、フルマネージド型の Argo CD エクスペリエンスをもたらします。アップストリームの Argo CD には多くのオプション設定と機能があり、この機能はその一部をサポートしています。サポートされている設定は、クラスター内の `argocd-cm` ConfigMap を使用して、アップストリームの Argo CD と同じ方法で設定します。この機能は、この ConfigMap からサポートされているフィールドを読み取り、マネージド Argo CD インスタンスに適用します。

以下のセクションでは、サポートされている設定で `argocd-cm` ConfigMap を設定する方法について説明します。

前提条件

Argo CD 設定を構成する前に、以下が必要です:

- Argo CD 機能が作成されている EKS クラスター (the section called “Argo CD 機能を作成する” を参照)
- この機能の Argo CD 用に設定された名前空間 (デフォルトでは `argocd` 名前空間)
- クラスターと通信できるように設定された `kubectl` CLI

argocd-cm ConfigMap を設定する

サポートされている Argo CD 設定を構成するには、クラスターに `argocd-cm` という名前の ConfigMap を作成します。マネージド機能は、この ConfigMap からサポートされている設定を読み取り、マネージド Argo CD インスタンスに適用します。この機能がサポートする設定とその適用方法については、「the section called “サポートされている設定”」を参照してください。

以下の要件で ConfigMap を作成します。

- ConfigMap に `argocd-cm` という名前を付けます。
- これをこの機能の Argo CD 用に設定された名前空間 (この機能の作成時に Argo CD 設定で設定した名前空間) に作成します。デフォルトでは、これは `argocd` 名前空間です。
- ラベル `app.kubernetes.io/part-of: argocd` を適用します。このラベルは必須であり、アップストリームの Argo CD の動作と一致します。

- アップストリーム Argo CD と同じフィールド形式とキーを使用します。

次の例は、Argo CD UI 全体にバナーを表示する設定を含む ConfigMap の構造を示しています。その他のサポートされている設定も、同じ方法で `data` の下に追加します。

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: argocd-cm
  namespace: argocd
  labels:
    app.kubernetes.io/part-of: argocd
data:
  ui.bannercontent: "Production cluster"
```

Important

ConfigMap は安全なストアではありません。`argocd-cm` ConfigMap にシークレット、認証情報、またはその他の機密情報を入れないでください。

この機能が設定を適用する方法

Argo CD を設定するには、独自のクラスターで `argocd-cm` ConfigMap を作成します。この機能は、ConfigMap のサポートされている設定をマネージド Argo CD インスタンスに適用します。適用されるのはサポートされている設定のみであり、設定したその他のフィールドや機能は無視されます。the section called “サポートされている設定” に記載されていない設定はサポートされておらず、効果はありません。

この機能は、設定した値を検証します。値が無効または不正な形式の場合、この機能はその値を無視し、その設定のデフォルト設定で実行を継続します。ConfigMap の間違いによって、マネージド Argo CD インスタンスが壊れることはありません。

Note

この機能は、クラスター内の `argocd-cm` ConfigMap から設定を適用します。この ConfigMap への書き込みアクセス権を持つプリンシパルは、マネージド Argo CD インスタンスの設定を変更できます。ConfigMap へのアクセスを制御するのは、クラスターの Kubernetes ロールベースアクセス制御 (RBAC) であり、機能リソースを制御する IAM アク

セス許可ではありません。セキュリティのベストプラクティスとして、Argo CD 名前空間内のオブジェクトを変更するアクセス許可は、信頼できるユーザーとサービスアカウントにのみ付与してください。Kubernetes RBAC はリソースタイプごとにアクセス許可をスコープ設定するため、他のユーザーが必要とするアクセス権は引き続き付与できます。例えば、デベロッパーに Application の作成と管理を許可しつつ、ConfigMap の変更は許可しないようにできます。これにより、デベロッパーが `argocd-cm` 設定を変更することを防げます。
責任共有モデル、Kubernetes RBAC、および Argo CD 機能の名前空間分離の詳細については、「the section called “EKS 機能に関する考慮事項”」を参照してください。Argo CD 内のアクセスを制御する方法については、「the section called “アクセス許可の設定”」を参照してください。

サポートされている設定

次のセクションでは、マネージド機能がサポートする `argocd-cm` 設定をカテゴリ別に示します。各設定は、アップストリームの Argo CD と同じキーと形式を使用します。各テーブルの値の適用方法列は、値が機能のデフォルト設定に追加されるのか、上書きされるのかを示します。各設定の詳細については、Argo CD ドキュメントのウェブサイトにある argocd-cm ConfigMap リファレンスを参照してください。

ユーザーインターフェイス

これらの設定は Argo CD UI をカスタマイズします。

設定	説明	値の適用方法
`ui.bannercontent`	UI 全体に表示されるバナーのテキスト (環境識別子やメンテナンス通知など)。	上書き
`ui.bannerurl`	バナーがリンクする URL (ランブックや Wiki ページなど)。	上書き
`ui.bannerpermanent`	ユーザーがバナーを閉じることを防止するには、`true` に設定します。	上書き
`ui.bannerposition`	バナーの表示位置: `top`、`bottom`、または `both`。	上書き

設定	説明	値の適用方法
ui.cssurl	ブランディングやスタイル設定用のカスタム CSS ファイルの URL。CSS はブラウザで実行されます。	上書き

リソース設定

これらの設定は、Argo CD が管理するリソースをこの機能がどのように監視、比較、表示するかを制御します。

設定	説明	値の適用方法
resource.customizations.ignoreDifferences.<group>_<kind>	Argo CD がリソースタイプについて Git とクラスターを比較する際に無視するフィールド (Horizontal Pod Autoscaler が管理するレプリカ数など)。	追加
resource.customizations.ignoreDifferences.all	Git とクラスターを比較する際に無視するフィールド。すべてのリソースタイプに適用されます。	追加
resource.customizations.ignoreResourceUpdates.<group>_<kind>	更新イベントが調整をトリガーするかどうかを判断する際に Argo CD が無視するフィールド。これにより、負荷が軽減されます。イベント自体は発生し、Argo CD はこれらのフィールドへの変更のみを無視します。	追加
resource.customizations.ignoreResourceUpdates.all	更新イベントを処理する際に Argo CD が無視するフィールド。これは、すべてのリソースタイプに適用されます。	追加

設定	説明	値の適用方法
`resource.customizations.knownTypeFields.<group>_<kind>`	カスタムリソースのフィールドタイプ (リスト、マップ、またはプリミティブ)。Argo CD がフィールド全体の置換を表示する代わりに、正確な差分を計算できるようにします。	追加
`resource.customizations.health.<group>_<kind>`	Lua スクリプトとして定義された、リソースタイプのカスタムヘルスチェック。この機能には、ACK リソースと kro リソースの組み込みヘルスチェックが含まれています。「the section called “カスタムヘルスチェック”」を参照してください。	上書き
`resource.exclusions`	Argo CD が監視しないリソースタイプ。これにより、変更の多いタイプのパフォーマンスが向上します。	追加
`resource.inclusions`	Argo CD が監視するリソースタイプ。設定すると、Argo CD はリストされたタイプのみを監視します。	追加
`resource.compareoptions`	Argo CD の差分計算方法を制御するオプション (`ignoreAggregatedRoles` など)。	上書き
`resource.respectRBAC`	コントローラーが、読み取りの RBAC アクセス許可を持つリソースのみを監視するかどうか。`normal` または `strict` を受け入れます。	上書き
`resource.customLabels`	UI のリソースビューに表示する追加のリソースラベル。	上書き

設定	説明	値の適用方法
resource.includeEventLabelKeys	Argo CD が生成する Kubernetes イベントにコピーする、Application と Project のラベル。	上書き
resource.excludeEventLabelKeys	Argo CD が生成する Kubernetes イベントから除外するラベル。	上書き
resource.sensitive.mask.annotations	UI または CLI がシークレットを表示するときにマスクするアノテーション。	上書き

リポジトリとツールの設定

これらの設定は、Argo CD がマニフェストをレンダリングするために使用するマニフェストツールを制御します。

設定	説明	値の適用方法
kustomize.enable	Kustomize をマニフェストソースタイプとして有効にするかどうか。	上書き
helm.enable	Helm をマニフェストソースタイプとして有効にするかどうか。	上書き
jsonnet.enable	Jsonnet をマニフェストソースタイプとして有効にするかどうか。	上書き
kustomize.buildOptions	すべての kustomize build に渡されるグローバルコマンドラインフラグ。この機能はフラグの一部をサポートしています。「the section called “サポートされている kustomize.buildOptions フラグ”」を参照してください。	上書き

サポートされている kustomize.buildOptions フラグ

kustomize.buildOptions の場合、この機能は値を安全でサポートされているフラグのセットにフィルタリングします。ビルドが任意のファイルを読み取ったり、任意のコードを実行したりできるフラグはサポートしていません。サポートされていないフラグや無効な値は個別に破棄し、残りのサポートされているフラグを適用します。フラグは --flag value 形式または --flag=value 形式のいずれでも記述できます。

フラグ	サポートされる値	注意事項
--reorder	legacy, none	レンダリングされた YAML の順序のみを変更します。
--enable-helm	ブール値	パスからマネージド Helm バイナリを実行します。
--enable-managedby-label	ブール値	ラベルのみを追加します。

この機能は、--load-restrictor、--enable-exec、--enable-alpha-plugins を含むその他のフラグを破棄します。

カスタムヘルスチェック

Argo CD は、デプロイするリソースの状態を評価します。Deployments や Services などの標準 Kubernetes リソースの場合、Argo CD にはヘルスロジックが組み込まれています。Argo CD が認識しないカスタムリソースでは、組み込みのヘルスロジックはなく、ヘルスステータスもレポートされません。

カスタムリソースにヘルスチェックがない場合、Argo CD は状態をレポートせず、Application の全体的な状態から除外します。その結果、リソースがまだプロビジョニング中または失敗した場合でも、Application は Healthy をレポートできます。つまり、同期の順序はレポートされたヘルスに依存するため、同期ウェーブはリソースの準備が完了する前に進行する可能性があります。

カスタムヘルスチェックを使用すると、カスタムリソースのヘルスロジックを定義できるため、Argo CD は正確なヘルスをレポートし、デプロイを正しく順序付けできます。同じ設定キーを使用して、アップストリーム Argo CD で行うのと同じ方法でカスタムヘルスチェックを定義します。既存のアップストリームスクリプトとコミュニティの例は、変更なしで Argo CD の EKS 機能と連携します。

ACK と kro の組み込みヘルスチェック

EKS Capability for Argo CD には、AWS Controllers for Kubernetes (ACK) および kro (Kube Resource Orchestrator) リソースの組み込みヘルスチェックが含まれています。これらのリソースは、追加の設定なしで正確な状態をレポートします。

機能が ACK または kro リソースのヘルスを評価する方法を変更するために、そのリソースタイプのカスタムヘルスチェックを定義できます。リソースタイプ用に定義したカスタムヘルスチェックは、そのタイプの組み込みヘルスチェックを上書きします。

カスタムヘルスチェックの記述

次の形式のキーを使用して、Lua スクリプトを `argocd-cm` ConfigMap に追加することで、カスタムヘルスチェックを定義します:

```
resource.customizations.health.<group>_<kind>
```

`<group>` をカスタムリソースの API グループに置き換え、*`<kind>`* をその種類に置き換えます。例えば、API グループ `example.com` と種類 `Database` を持つカスタムリソースのキーは `resource.customizations.health.example.com_Database` です。

Lua スクリプトは、グローバル `obj` 変数を介してリソースオブジェクトにアクセスできます。スクリプトは、`Healthy`、`Progressing`、`Degraded`、または `Suspended` のいずれかに設定された `status` フィールドを持つテーブルを返す必要があります。スクリプトは、オプションの `message` フィールドを設定して、説明的なステータスメッセージを提供することもできます。

次の ConfigMap の例では、`Database` カスタムリソースのヘルスチェックを定義します。スクリプトは、ステータスフェーズが `Ready` の場合にリソースを `Healthy` としてレポートし、それ以外の場合は `Progressing` としてレポートします:

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: argocd-cm
  namespace: argocd
  labels:
    app.kubernetes.io/part-of: argocd
data:
  resource.customizations.health.example.com_Database: |
    hs = {}
    hs.status = "Progressing"
```

```
    hs.message = "Waiting for the resource to become ready"
    if obj.status ~= nil then
      if obj.status.phase == "Ready" then
        hs.status = "Healthy"
        hs.message = "Database is ready"
      end
    end
    return hs
```

ヘルスチェックスクリプトの形式、組み込みヘルスチェックのリスト、および適応できるコミュニティの例の詳細については、Argo CD ドキュメントウェブサイトの「リソースヘルス」を参照してください。

安全と制限事項

マネージド機能を使用すると、カスタムヘルスチェックスクリプトは分離されたフルマネージドコンピューティングで実行されます。実行環境は機能ごとに分離され、クラスターのデータや AWS API にはアクセスできません。実行環境の一部をプロビジョニング、パッチ適用、または操作することはありません。

EKS 機能で使用するカスタムヘルスチェックを記述するときは、次の点に注意してください:

- 標準 Lua ライブラリは使用できません。`useOpenLibs` オプションは常に無効になっています。これはアップストリーム Argo CD でのデフォルトです。スクリプトはオペレーティングシステムまたはファイルシステムにアクセスできません。標準 Lua ライブラリに依存するセルフマネージド Argo CD からスクリプトを移行すると、この機能の同じ方法で実行されない場合があります。本番環境で使用する前に、開発環境でヘルスチェックスクリプトをテストすることをお勧めします。

ヘルス評価が一時的に利用できない場合、この機能はヘルスステータスを削除するのではなく、影響を受けたカスタムリソースを `Progressing` としてレポートします。これにより、評価が回復するまで、影響を受けたリソースが Application の状態に表示されます。

カスタムヘルスチェックを検証する

`argocd-cm` ConfigMap を適用または更新したら、ヘルスチェックがアクティブであることを確認します。

1. Argo CD UI で、ヘルスチェックを定義した種類のカスタムリソースを含む Application を選択します。スクリプトが返すヘルスステータスをリソースがレポートしていることを確認します。ま

たは、`argocd app get <application-name>` を実行して、リソースのヘルスステータスを確認します。

2. リソースが予想されるヘルスをレポートしない場合は、以下を確認します:
   - ConfigMap は `argocd-cm` という名前であり、この機能の Argo CD 用に設定された名前空間にあります。
   - ConfigMap には必要な `app.kubernetes.io/part-of: argocd` ラベルがあります。
   - ヘルスチェックキーは、リソースタイプで正しい `<group>_<kind>` を使用します。

Application を作成する

Application は、ターゲットクラスター内のデプロイを表します。Application ごとに、ソース (Git リポジトリ) と宛先 (クラスターと名前空間) を定義します。適用すると、Argo CD は Git リポジトリのマニフェストで指定されたリソースをクラスター内の名前空間に作成します。Application では、ワークロードのデプロイを指定することが多くなるものの、送信先クラスターで使用可能な Kubernetes リソースを管理することもできます。

前提条件

- EKS クラスターに Argo CD 機能が作成されていること
- リポジトリアクセスが設定されていること (「the section called “リポジトリを設定する”」を参照)
- ターゲットクラスターが登録されていること (「the section called “クラスターを登録する”」を参照)
- クラスターと通信できるように `kubectl` が設定されていること

基本的な Application を作成する

Git リポジトリからデプロイする Application を定義します。

```
apiVersion: argoproj.io/v1alpha1
kind: Application
metadata:
  name: guestbook
  namespace: argocd
spec:
  project: default
  source:
    repoURL: https://github.com/argoproj/argocd-example-apps
```

```
    targetRevision: HEAD
    path: guestbook
  destination:
    name: in-cluster
    namespace: default
```

> **Note**
>
> クラスターの登録時に使用したクラスター名で destination.name を使用します (ローカルクラスターの in-cluster の場合など)。destination.server フィールドは EKS クラスター ARN でも機能しますが、読みやすくするためにクラスター名を使用することをお勧めします。

Application を適用します。

```
kubectl apply -f application.yaml
```

アプリケーションのステータスを表示します。

```
kubectl get application guestbook -n argocd
```

ソース設定

Git リポジトリ:

```
spec:
  source:
    repoURL: https://github.com/example/my-app
    targetRevision: main
    path: kubernetes/manifests
```

特定の Git タグまたはコミット:

```
spec:
  source:
    targetRevision: v1.2.0  # or commit SHA
```

Helm チャート:

```
spec:
  source:
    repoURL: https://github.com/example/helm-charts
    targetRevision: main
    path: charts/my-app
    helm:
      valueFiles:
      - values.yaml
      parameters:
      - name: image.tag
        value: v1.2.0
```

外部 Git リポジトリの値を含む Helm チャート (マルチソースパターン):

```
spec:
  sources:
  - repoURL: https://github.com/example/helm-charts
    targetRevision: main
    path: charts/my-app
    helm:
      valueFiles:
      - $values/environments/production/values.yaml
  - repoURL: https://github.com/example/config-repo
    targetRevision: main
    ref: values
```

詳細については、Argo CD ドキュメントの「Helm Value Files from External Git Repository」を参照してください。

ECR Helm チャート:

```
spec:
  source:
    repoURL: oci://account-id.dkr.ecr.region.amazonaws.com/repository-name
    targetRevision: chart-version
    chart: chart-name
```

機能ロールに必要な ECR アクセス許可がある場合、リポジトリが直接使用されるため、リポジトリ設定は必要ありません。詳細については、「the section called “リポジトリを設定する”」を参照してください。

CodeCommit の Git リポジトリ:

```
spec:
  source:
    repoURL: https://git-codecommit.region.amazonaws.com/v1/repos/repository-name
    targetRevision: main
    path: kubernetes/manifests
```

機能ロールに必要な CodeCommit アクセス許可がある場合、リポジトリが直接使用されるため、リポジトリ設定は必要ありません。詳細については、「the section called “リポジトリを設定する”」を参照してください。

CodeConnections の Git リポジトリ:

```
spec:
  source:
    repoURL: https://codeconnections.region.amazonaws.com/git-http/account-
id/region/connection-id/owner/repository.git
    targetRevision: main
    path: kubernetes/manifests
```

リポジトリ URL 形式は、CodeConnections 接続 ARN から取得されます。機能ロールに必要な CodeConnections アクセス許可があり、接続が設定されている場合、リポジトリが直接使用されるため、リポジトリ設定は必要ありません。詳細については、「the section called “リポジトリを設定する”」を参照してください。

Kustomize:

```
spec:
  source:
    repoURL: https://github.com/example/kustomize-app
    targetRevision: main
    path: overlays/production
    kustomize:
      namePrefix: prod-
```

同期ポリシー

Argo CD によるアプリケーション同期の方法を制御します。

手動同期 (デフォルト):

アプリケーションを同期するには、手動による承認が必要です。

```
spec:
  syncPolicy: {}  # No automated sync
```

手動で同期をトリガーします。

```
kubectl patch application guestbook -n argocd \
  --type merge \
  --patch '{"operation": {"initiatedBy": {"username": "admin"}, "sync": {}}}'
```

自動同期:

アプリケーションは、Git の変更が検出されたら自動的に同期します。

```
spec:
  syncPolicy:
    automated: {}
```

自己修復:

クラスターに手動で加えられた変更を自動的に元に戻します。

```
spec:
  syncPolicy:
    automated:
      selfHeal: true
```

有効にすると、Argo CD はクラスターに手動で直接加えられた変更を元に戻して、Git を信頼できるソースとして維持します。

プルーニング:

Git から除去されたリソースを自動的に削除します。

```
spec:
  syncPolicy:
    automated:
      prune: true
```

> ⚠ Warning
>
> プルーニングにより、クラスターからリソースが削除されます。本番稼働環境での使用には注意が必要です。

自動同期を組み合わせる:

```
spec:
  syncPolicy:
    automated:
      prune: true
      selfHeal: true
    syncOptions:
    - CreateNamespace=true
```

再試行設定:

失敗した同期の再試行動作を設定します。

```
spec:
  syncPolicy:
    retry:
      limit: 5  # Number of failed sync attempts; unlimited if less than 0
      backoff:
        duration: 5s  # Amount to back off (default unit: seconds, also supports "2m",
 "1h")
        factor: 2  # Factor to multiply the base duration after each failed retry
        maxDuration: 3m  # Maximum amount of time allowed for the backoff strategy
```

これは、最初に作成される CRD に依存するリソースや、CRD がすぐに利用できない可能性がある kro インスタンスを操作する場合に特に便利です。

同期オプション

追加の同期設定:

名前空間が存在しない場合は作成する:

```
spec:
  syncPolicy:
```

```
    syncOptions:
    - CreateNamespace=true
```

リソースが見つからない場合は、ドライランをスキップする:

まだ存在しない CRD (kro インスタンスなど) に依存するリソースを適用する場合に便利です。

```
spec:
  syncPolicy:
    syncOptions:
    - SkipDryRunOnMissingResource=true
```

これは、リソース自体のラベルを使用して特定のリソースに適用することもできます。

適用する前にリソースを検証する:

```
spec:
  syncPolicy:
    syncOptions:
    - Validate=true
```

同期が取れていない場合にのみ適用する:

```
spec:
  syncPolicy:
    syncOptions:
    - ApplyOutOfSyncOnly=true
```

高度な同期機能

Argo CD では、複雑なデプロイ向けに高度な同期機能をサポートしています。

- 同期ウェーブ - `argocd.argoproj.io/sync-wave` 注釈を使用してリソースの作成順序を制御します。
- 同期フック - `argocd.argoproj.io/hook` 注釈を使用して同期の前後にジョブを実行します (PreSync、PostSync、SyncFail)
- リソースのヘルス評価 - アプリケーションに固有のリソースのカスタムヘルスチェックです。

詳細については、Argo CD ドキュメントの「同期ウェーブ」と「リソースフック」を参照してください。

差分を無視する

Argo CD が他のコントローラー (HPA 管理レプリカなど) で管理されている特定のフィールドを同期しないようにします。

```
spec:
  ignoreDifferences:
  - group: apps
    kind: Deployment
    jsonPointers:
    - /spec/replicas
```

無視パターンとフィールド除外の詳細については、Argo CD ドキュメントの「差分比較のカスタマイズ」を参照してください。

マルチ環境デプロイ

複数の環境に同じアプリケーションをデプロイします。

開発:

```
apiVersion: argoproj.io/v1alpha1
kind: Application
metadata:
  name: my-app-dev
  namespace: argocd
spec:
  project: default
  source:
    repoURL: https://github.com/example/my-app
    targetRevision: develop
    path: overlays/development
  destination:
    name: dev-cluster
    namespace: my-app
```

本番稼働:

```
apiVersion: argoproj.io/v1alpha1
kind: Application
metadata:
  name: my-app-prod
```

```
  namespace: argocd
spec:
  project: default
  source:
    repoURL: https://github.com/example/my-app
    targetRevision: main
    path: overlays/production
  destination:
    name: prod-cluster
    namespace: my-app
  syncPolicy:
    automated:
      prune: true
      selfHeal: true
```

Application をモニタリングおよび管理する

アプリケーションのステータスを表示する:

```
kubectl get application my-app -n argocd
```

Argo CD UI にアクセスする:

EKS コンソールから Argo CD UI を開いて、アプリケーションのトポロジ、同期ステータス、リソースヘルス、デプロイ履歴を表示します。UI アクセスの手順については、「the section called “Argo CD の使用”」を参照してください。

Application をロールバックする:

以前のリビジョンにロールバックするには、Argo CD UI、Argo CD CLI を使用するか、Application 仕様の `targetRevision` を以前の Git コミットまたはタグに更新します。

Argo CD CLI を使用する:

```
argocd app rollback argocd/my-app <revision-id>
```

> **Note**
>
> マネージド機能で Argo CD CLI を使用する場合は、名前空間プレフィックス `namespace/appname` を使用してアプリケーションを指定します。

詳細は、Argo CD ドキュメントの「argocd app rollback」を参照してください。

その他のリソース

- the section called “プロジェクト” - Project でマルチテナント環境向けの Application を整理する
- the section called “ApplicationSet” - テンプレートで複数のクラスターにデプロイする
- Application 仕様 - 詳細なアプリケーション API リファレンス
- 同期オプション - 高度な同期設定

ApplicationSet を使用する

ApplicationSet によりテンプレートから複数の Application が生成されるため、単一のリソース定義で複数のクラスター、環境、または名前空間に同じアプリケーションをデプロイできます。

前提条件

- EKS クラスターに Argo CD 機能が作成されていること
- リポジトリアクセスが設定されていること (「the section called “リポジトリを設定する”」を参照)
- クラスターと通信できるように `kubectl` が設定されていること

> **Note**
>
> ApplicationSet には複数のターゲットクラスターは必要ありません。クラスタージェネレーター以外のジェネレーター (リスト、git、マトリックスジェネレーターなど) を使用して、リモートクラスターなしでアプリケーションをデプロイできます。

ApplicationSet の仕組み

ApplicationSet は、ジェネレーターを使用してパラメータを生成し、生成したパラメータを Application テンプレートに適用します。生成したパラメータのセットごとに Application が 1 つ作成されます。

EKS のデプロイによく使用されるジェネレーター:

- リストジェネレーター - 環境ごとにクラスターとパラメータを明示的に定義します。
- クラスタージェネレーター - 登録されたすべてのクラスターに自動的にデプロイします。

- Git ジェネレーター - リポジトリ構造から Application を生成します。
- マトリックスジェネレーター - 多次元デプロイ用にいくつかジェネレーターを組み合わせます。
- マージジェネレーター – 複数のジェネレーターからのパラメータをマージします。

ジェネレーターの詳細なリファレンスについては、「ApplicationSet ドキュメント」を参照してください。

リストジェネレーター

複数のクラスターにデプロイするように明示的に設定します。

```
apiVersion: argoproj.io/v1alpha1
kind: ApplicationSet
metadata:
  name: guestbook-all-clusters
  namespace: argocd
spec:
  generators:
  - list:
      elements:
      - environment: dev
        replicas: "2"
      - environment: staging
        replicas: "3"
      - environment: prod
        replicas: "5"
  template:
    metadata:
      name: 'guestbook-{{environment}}'
    spec:
      project: default
      source:
        repoURL: https://github.com/example/guestbook
        targetRevision: HEAD
        path: 'overlays/{{environment}}'
      destination:
        name: '{{environment}}-cluster'
        namespace: guestbook
      syncPolicy:
        automated:
          prune: true
```

```
        selfHeal: true
```

> **Note**
>
> 読みやすくするために、destination.name をクラスター名と合わせて使用します。destination.server フィールドは、必要に応じて EKS クラスター ARN とも連携します。

これにより、guestbook-dev、guestbook-staging、guestbook-prod の 3 つの Application が作成されます。

クラスタージェネレーター

登録済みのすべてのクラスターに自動的にデプロイします。

```
apiVersion: argoproj.io/v1alpha1
kind: ApplicationSet
metadata:
  name: cluster-addons
  namespace: argocd
spec:
  generators:
  - clusters: {}
  template:
    metadata:
      name: '{{name}}-addons'
    spec:
      project: default
      source:
        repoURL: https://github.com/example/cluster-addons
        targetRevision: HEAD
        path: addons
      destination:
        server: '{{server}}'
        namespace: kube-system
      syncPolicy:
        automated:
          prune: true
          selfHeal: true
```

これにより、登録済みのクラスターごとに Application が自動的に作成されます。

クラスターをフィルタリングする:

matchLabels を使用して特定のクラスターを含めるか、matchExpressions を使用してクラスターを除外します。

```
spec:
  generators:
  - clusters:
      selector:
        matchLabels:
          environment: production
        matchExpressions:
        - key: skip-appset
          operator: DoesNotExist
```

Git ジェネレーター

Git ジェネレーターは、リポジトリ構造に基づいて Application を作成します。

- ディレクトリジェネレーター - 各ディレクトリを個別の Application としてデプロイします (マイクロサービスに有用です)。
- ファイルジェネレーター - パラメータファイルから Application を生成します (マルチテナントデプロイに有用です)。

例: マイクロサービスのデプロイ

```
apiVersion: argoproj.io/v1alpha1
kind: ApplicationSet
metadata:
  name: microservices
  namespace: argocd
spec:
  generators:
  - git:
      repoURL: https://github.com/example/microservices
      revision: HEAD
      directories:
      - path: services/*
  template:
    metadata:
      name: '{{path.basename}}'
```

```
    spec:
      project: default
      source:
        repoURL: https://github.com/example/microservices
        targetRevision: HEAD
        path: '{{path}}'
      destination:
        name: my-cluster
        namespace: '{{path.basename}}'
      syncPolicy:
        automated:
          prune: true
          selfHeal: true
        syncOptions:
        - CreateNamespace=true
```

Git ジェネレーターとファイルベースの設定の詳細については、Argo CD ドキュメントの「Git ジェネレーター」を参照してください。

マトリックスジェネレーター

複数のジェネレーターを組み合わせて、複数のディメンション (環境 × クラスター) にデプロイします。

```
apiVersion: argoproj.io/v1alpha1
kind: ApplicationSet
metadata:
  name: multi-env-multi-cluster
  namespace: argocd
spec:
  generators:
  - matrix:
      generators:
      - list:
          elements:
          - environment: dev
          - environment: staging
          - environment: prod
      - clusters:
          selector:
            matchLabels:
              region: us-west-2
  template:
```

```
    metadata:
      name: 'app-{{environment}}-{{name}}'
    spec:
      project: default
      source:
        repoURL: https://github.com/example/app
        targetRevision: HEAD
        path: 'overlays/{{environment}}'
      destination:
        name: '{{name}}'
        namespace: 'app-{{environment}}'
```

ジェネレーターを組み合わせる方法の詳細については、Argo CD ドキュメントの「マトリックスジェネレーター」を参照してください。

マルチリージョンデプロイ

複数のリージョンでクラスターにデプロイします。

```
apiVersion: argoproj.io/v1alpha1
kind: ApplicationSet
metadata:
  name: global-app
  namespace: argocd
spec:
  generators:
  - list:
      elements:
      - clusterName: prod-us-west
        region: us-west-2
      - clusterName: prod-us-east
        region: us-east-1
      - clusterName: prod-eu-west
        region: eu-west-1
  template:
    metadata:
      name: 'app-{{region}}'
    spec:
      project: default
      source:
        repoURL: https://github.com/example/app
        targetRevision: HEAD
        path: kubernetes
```

```
        helm:
          parameters:
          - name: region
            value: '{{region}}'
      destination:
        name: '{{clusterName}}'
        namespace: app
      syncPolicy:
        automated:
          prune: true
          selfHeal: true
```

ApplicationSet を管理する

ApplicationSet と生成された Application を表示する:

```
kubectl get applicationsets -n argocd
kubectl get applications -n argocd -l argocd.argoproj.io/application-set-
name=<applicationset-name>
```

ApplicationSet を更新する:

ApplicationSet の仕様を変更して再適用します。Argo CD は、生成したすべての Application を自動的に更新します。

```
kubectl apply -f applicationset.yaml
```

ApplicationSet を削除する:

```
kubectl delete applicationset <name> -n argocd
```

> **Warning**
>
> ApplicationSet を削除すると、生成されたすべての Application も削除されます。削除対象の Application に `prune: true` がある場合、そのリソースもターゲットクラスターから削除されます。
> ApplicationSet を削除するときにデプロイされたリソースを保持するには、ApplicationSet 仕様で `.syncPolicy.preserveResourcesOnDeletion` を `true` に設定します。詳細は、Argo CD ドキュメントの「Application Pruning & Resource Deletion」を参照してください。

> **Important**
>
> Argo CD の ApplicationSet 機能には、ApplicationSet を使用する前に注意すべきセキュリティ上の考慮事項があります。詳細は、Argo CD ドキュメントの「ApplicationSet Security」を参照してください。

その他のリソース

- the section called “プロジェクト” - Project で ApplicationSet を整理する
- the section called “Application を作成する” - Application 設定を理解する
- ApplicationSet ドキュメント - ジェネレーターの詳細なリファレンスとパターン
- ジェネレーターのリファレンス - ジェネレーターの詳細な仕様

Argo CD のプライベートエンドポイントアクセスを設定する

デフォルトでは、パブリックインターネット経由で Argo CD エンドポイントにアクセスできます。VPC と Argo CD エンドポイント間のトラフィックがパブリックインターネットを経由せずに AWS ネットワーク内に留まるようにプライベートエンドポイントアクセスを設定できます。

プライベートアクセスを有効にすると、AWS はドメインネームシステム (DNS) を更新して、Argo CD エンドポイントのホスト名が VPC エンドポイントに関連付けられたプライベート IP アドレスに解決されるようにします。Argo CD エンドポイントにアクセスするには、これらのプライベート IP へのネットワーク接続 (VPC、仮想プライベートネットワーク (VPN) 接続、または AWS Direct Connect 経由) がある必要があります。

プライベート IP VPN の仕組み

Argo CD 機能のプライベートアクセスを設定する場合:

1. VPC 内の `com.amazonaws.`*`region-code`*`.eks-capabilities` サービス用の VPC エンドポイントを作成します。
2. Argo CD 機能を作成または更新するときに、VPC エンドポイント ID を指定します。
3. AWS は、Argo CD エンドポイントのホスト名が VPC エンドポイントに関連付けられたプライベート IP アドレスに解決されるように DNS レコードを設定します。
4. Argo CD エンドポイントへのすべてのトラフィックは、VPC エンドポイントを介してプライベートにフローします。

前提条件

- アクティブな Argo CD 機能を持つ Amazon Elastic Kubernetes Service (Amazon EKS) クラスター (the section called “Argo CD 機能を作成する” を参照)
- サブネット、セキュリティグループ、インターフェイス VPC エンドポイントなど、Amazon VPC の概念に精通していること
- プライベートアクセスが必要なアベイラビリティーゾーンにサブネットを持つ VPC
- AWS コマンドラインインターフェイス (AWS CLI) バージョン `2.12.3` 以降

ステップ 1: eks-capabilities 用の VPC エンドポイントを作成する

`com.amazonaws.`*`region-code`*`.eks-capabilities` サービス用のインターフェイス VPC エンドポイントを作成します。

Console

1. Amazon VPC コンソール を開きます。
2. 左のナビゲーションペインで [エンドポイント] を選択してください。
3. エンドポイントの作成 を選択します。
4. [サービスカテゴリ] で、[AWS のサービス] を選択します。
5. 検索フィールドに `eks-capabilities` と入力して、`com.amazonaws.`*`region-code`*`.eks-capabilities` を選択します。
6. [VPC] では、Argo CD エンドポイントへのアクセス元になる VPC を選択します。
7. [サブネット] では、プライベートアクセスが必要な各アベイラビリティーゾーンのサブネットを選択します。
8. [セキュリティグループ] では、クライアントネットワークからのインバウンド HTTPS トラフィック (ポート 443) を許可するセキュリティグループを選択します。
9. エンドポイントの作成 を選択します。
10. VPC エンドポイント ID (例: `vpce-0123456789abcdef0`) を書き留めます。

CLI

`region-code` を自分の AWS リージョンに、*`vpc-id`* を VPC ID に、*`subnet-id-1 subnet-id-2`* を異なるアベイラビリティーゾーンのサブネット ID に、および *`sg-id`* を HTTPS インバウンドトラフィックを許可するセキュリティグループに置き換えます。

```
aws ec2 create-vpc-endpoint \
  --region region-code \
  --vpc-endpoint-type Interface \
  --service-name com.amazonaws.region-code.eks-capabilities \
  --vpc-id vpc-id \
  --subnet-ids subnet-id-1 subnet-id-2 \
  --security-group-ids sg-id
```

レスポンス内の VpcEndpointId を書き留めます。

> **Important**
>
> VPC エンドポイントにアタッチされたセキュリティグループがインバウンド HTTPS (ポート 443) トラフィックを許可していることを確認します。トラフィックソースには、ユーザーまたはアプリケーションが Argo CD エンドポイントにアクセスするすべてのネットワークが含まれている必要があります。

ステップ 2: Argo CD 機能でプライベートアクセスを設定する

VPC エンドポイントが利用可能になったら、それを使用するように Argo CD 機能を設定します。新しい機能を作成するとき、または既存の機能を更新することで、プライベートアクセスを設定できます。

プライベートアクセスで新しい機能を作成する

create-capability コマンドに networkAccess を含めます。*region-code*、*my-cluster*、およびその他のプレースホルダーを自分の値に置き換えます。

```
aws eks create-capability \
  --region region-code \
  --cluster-name my-cluster \
  --capability-name my-argocd \
  --type ARGOCD \
  --role-arn arn:aws:iam::111122223333:role/ArgoCDCapabilityRole \
  --configuration '{
    "argoCd": {
      "awsIdc": {
        "idcInstanceArn": "arn:aws:sso:::instance/ssoins-1234567890abcdef",
```

```
        "idcRegion": "idc-region-code"
      },
      "rbacRoleMappings": [{
        "role": "ADMIN",
        "identities": [{
          "id": "user-id",
          "type": "SSO_USER"
        }]
      }],
      "networkAccess": {
        "vpceIds": ["vpce-0123456789abcdef0"]
      }
    }
  }'
```

既存の機能を更新してプライベートアクセスを有効にする

region-code、*my-cluster*、*my-argocd*、および VPC エンドポイント ID を自分の値に置き換えます。

```
aws eks update-capability \
  --region region-code \
  --cluster-name my-cluster \
  --capability-name my-argocd \
  --configuration '{
    "argoCd": {
      "networkAccess": {
        "vpceIds": ["vpce-0123456789abcdef0"]
      }
    }
  }'
```

機能の更新が完了するまで待機します。

```
aws eks describe-capability \
  --region region-code \
  --cluster-name my-cluster \
  --capability-name my-argocd \
  --query 'capability.status' \
  --output text
```

更新が完了すると、ステータスが ACTIVE に戻ります。

ステップ 3: DNS の解決と接続を確認する

プライベートアクセスが設定された状態で機能がアクティブになったら、Argo CD エンドポイントのホスト名がプライベート IP アドレスに解決されていることを確認します。

任意のネットワークから、次の nslookup コマンドを実行します:

```
nslookup your-argocd-endpoint.eks-capabilities.region-code.amazonaws.com
```

レスポンスには、プライベート IP アドレス (例えば、10.x.x.x や 172.16.x.x などの RFC 1918 範囲のアドレス) が表示されます。

エンドポイントがプライベート IP に解決されない場合は、以下を確認します:

- 機能ステータスが ACTIVE である (AWS は、この機能がアクティブになった後に DNS レコードを作成します)
- VPC エンドポイントが available 状態である
- 機能設定の VPC エンドポイント ID が、作成したエンドポイントと一致する

> **Note**
>
> プライベートアクセスを設定すると、Argo CD エンドポイントのホスト名はプライベート IP アドレスに解決されます。接続するには、VPC、VPN 接続、または AWS Direct Connect を介して、これらのプライベート IP へのネットワーク接続がある必要があります。

プライベートアクセスを無効にする

プライベートアクセスを無効にし、パブリックエンドポイントアクセスを復元するには、VPC エンドポイント ID を削除するように機能を更新します。

```
aws eks update-capability \
  --region region-code \
  --cluster-name my-cluster \
  --capability-name my-argocd \
  --configuration '{
    "argoCd": {
      "networkAccess": {
```

```
        "vpceIds": []
      }
    }
  }'
```

考慮事項

- プライベートアクセスを有効にすると、Argo CD エンドポイントには VPC 内またはそれに接続されているネットワーク内からのみ到達できます (VPN や AWS Direct Connect など)。
- プライベートアクセスが機能するには、VPC エンドポイントが `available` 状態のままである必要があります。VPC エンドポイントを削除すると、新しいエンドポイントを作成して機能を更新するまで、Argo CD エンドポイントにアクセスできなくなります。
- AWS は現在、`eks-capabilities` サービスエンドポイントの VPC エンドポイントポリシーをサポートしていません。

Argo CD に関する考慮事項

このトピックでは、EKS Capability for Argo CD を使用する際の重要な考慮事項を示します。具体的には、計画、アクセス許可、認証、マルチクラスターデプロイパターンなどです。

計画

Argo CD をデプロイする前に、以下の点を考慮してください。

リポジトリ戦略: アプリケーションマニフェストの保存場所を決定します (CodeCommit、GitHub、GitLab、Bitbucket)。異なる環境ごとにリポジトリ構造と分岐戦略を計画します。

RBAC 戦略: どのチームやユーザーが管理者、編集者、またはビューワーのアクセス権を持つべきかを計画します。これらを AWS アイデンティティセンターグループまたは Argo CD ロールにマッピングします。

マルチクラスターアーキテクチャ: 単一の Argo CD インスタンスから複数のクラスターを管理するかどうかを決定します。Argo CD に専用の管理クラスターを使用することを検討してください。

Application の編成: Application と ApplicationSet をどのように構造化するかを計画します。Project を使用してチームや環境ごとに Application を編成することを検討してください。

同期ポリシー: アプリケーションを自動的に同期するか、手動で承認するかを決定します。開発環境では自動同期が一般的ですが、本番稼働環境では手動で同期します。

アクセス許可

IAM 機能ロール、信頼ポリシー、セキュリティベストプラクティスの詳細については、「the section called “機能 IAM ロール”」および「the section called “EKS 機能に関する考慮事項”」を参照してください。

IAM 機能ロールの概要

Argo CD 機能リソースを作成するときに、IAM 機能ロールを指定できます。ACK とは異なり、Argo CD は主に Kubernetes リソースを管理し、AWS リソースを直接には管理しません。一方、次の場合には IAM 機能ロールが必要です。

- CodeCommit でプライベート Git リポジトリにアクセスする場合
- 認証のために AWS アイデンティティセンターと統合する場合
- AWS Secrets Manager でシークレットにアクセスする場合 (設定されている場合)
- 他の EKS クラスターにクロスクラスターでデプロイする場合

CodeCommit 統合

CodeCommit リポジトリを使用している場合は、読み取りアクセス許可を付与したポリシーをアタッチします。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "codecommit:GitPull"
      ],
      "Resource": "*"
    }
  ]
}
```

> **Important**
>
> 本番稼働で使用する場合は、"*" を使用する代わりに、Resource フィールドを特定のリポジトリ ARN に制限します。

例:

```
"Resource": "arn:aws:codecommit:us-west-2:111122223333:my-app-repo"
```

これにより、Argo CD 機能のアクセスが要管理のリポジトリにのみ制限されます。

Secrets Manager の統合

Secrets Manager にリポジトリ認証情報を保存する場合は、読み取りアクセスのマネージドポリシーをアタッチします。

```
arn:aws:iam::aws:policy/AWSSecretsManagerClientReadOnlyAccess
```

このポリシーには、`secretsmanager:GetSecretValue`、`secretsmanager:DescribeSecret`、および KMS 復号アクセス許可という必要なアクセス許可が含まれています。

基本セットアップ

基本的な Argo CD 機能にパブリック Git リポジトリがある場合、信頼ポリシー以外に追加の IAM ポリシーは必要ありません。

認証

AWS アイデンティティセンターの統合

Argo CD マネージド機能を AWS アイデンティティセンター (以前の AWS SSO) と直接統合すると、認証に既存の ID プロバイダーを使用できます。

AWS アイデンティティセンター統合を設定した場合:

1. ユーザーは、EKS コンソールから Argo CD UI にアクセスします。
2. AWS アイデンティティセンターを使用してユーザーが認証されます (社内の ID プロバイダーにフェデレーションできます)。
3. AWS アイデンティティセンターは、ユーザーとグループの情報を Argo CD に提供します。
4. Argo CD は、現在の設定に基づいてユーザーとグループを RBAC ロールにマッピングします。
5. ユーザーには、自分がアクセス許可を持つアプリケーションとリソースのみが表示されます。

アイデンティティセンターアクセス許可セットでアクセスをシンプルにする

AWS アイデンティティセンターでは、Argo CD を使用する際に以下の 2 種類の認証パスを利用できます。

Argo CD API 認証: アイデンティティセンターにより、Argo CD UI と API から SSO 認証を行うことができます。これを設定するには、Argo CD 機能の RBAC ロールマッピングを使用します。

EKS クラスターアクセス: Argo CD 機能は、お客様提供の IAM ロールを使用して、アクセスエントリを基に EKS クラスターで認証します。こうしたアクセスエントリは、アクセス許可を追加または削除するように手動で設定できます。

アイデンティティセンターアクセス許可セットを使用すると、単一の ID で Argo CD クラスターと EKS クラスターの両方にアクセスできるため、ID 管理がシンプルになります。Argo CD アクセス用とクラスターアクセス用の認証情報を別々に維持するのではなく、両方のシステムで 1 つの ID だけを管理すればよいので、オーバーヘッドが軽減されます。

RBAC ロールマッピング

Argo CD には、AWS アイデンティティセンターのユーザーとグループにマッピングできるロールが組み込まれています。

ADMIN: すべてのアプリケーションと設定にフルアクセスできます。アプリケーションの作成、デプロイ、更新、削除が可能です。Argo CD の設定を管理できます。

EDITOR: アプリケーションを作成および変更できます。Argo CD の設定を変更したり、アプリケーションを削除したりすることはできません。

VIEWER: アプリケーションに読み取り専用でアクセスできます。アプリケーションのステータスおよび履歴を表示できます。変更を加えることはできません。

> **Note**
>
> ロール名は、大文字と小文字が区別され、大文字 (ADMIN、EDITOR、VIEWER) にする必要があります。

> **Important**
>
> EKS 機能と AWS アイデンティティセンターが統合されている場合、Argo CD 機能あたり最大 1,000 個の ID がサポートされます。ユーザーやグループを ID にすることができます。

マルチクラスターデプロイ

Argo CD マネージド機能は、マルチクラスターデプロイをサポートしているため、単一の Argo CD インスタンスから開発、ステージング、本番稼働のクラスター全体にわたってアプリケーションを管理できます。

マルチクラスターの仕組み

Argo CD に追加でクラスターを登録する場合:

1. ARN でターゲット EKS クラスターを参照するクラスターシークレットを作成します。
2. ターゲットが異なるクラスターごとに Application または ApplicationSet を作成します。
3. Argo CD は、各クラスターに接続してデプロイし、リソースを監視する
4. 単一の Argo CD UI からすべてのクラスターを表示して管理します。

マルチクラスターの前提条件

追加でクラスターを登録する前に:

- ターゲットクラスターで Argo CD 機能ロールのアクセスエントリを作成すること
- Argo CD 機能とターゲットクラスターとの間にネットワーク接続を確保すること
- ターゲットクラスターにアクセスするための IAM アクセス許可を検証すること

クラスターを登録する

Kubernetes シークレットを使用して `argocd` 名前空間にクラスターを登録します。

ターゲットクラスター ARN を取得します。*`region-code`* はターゲットクラスターがある AWS リージョンに置き換え、*`target-cluster`* はターゲットクラスターの名前に置き換えます。

```
aws eks describe-cluster \
  --region region-code \
  --name target-cluster \
  --query 'cluster.arn' \
  --output text
```

クラスター ARN を使用して、クラスターシークレットを作成します。

```
apiVersion: v1
```

```
kind: Secret
metadata:
  name: target-cluster
  namespace: argocd
  labels:
    argocd.argoproj.io/secret-type: cluster
type: Opaque
stringData:
  name: target-cluster
  server: arn:aws:eks:us-west-2:111122223333:cluster/target-cluster
  project: default
```

Important

Kubernetes API サーバー URL ではなく、`server` フィールドで EKS クラスター ARN を使用します。マネージド機能では、ARN でターゲットクラスターを識別する必要があります。

シークレットを適用します。

```
kubectl apply -f cluster-secret.yaml
```

ターゲットクラスターでアクセスエントリを設定する

アプリケーションのデプロイに必要なアクセス許可を Argo CD 機能ロールに付与するには、ターゲットクラスターでアクセスエントリを設定する必要があります。*region-code* はターゲットクラスターがある AWS リージョンに置き換え、*target-cluster* はターゲットクラスターの名前に置き換え、ARN は Argo CD 機能ロール ARN に置き換えます。

```
aws eks create-access-entry \
  --region region-code \
  --cluster-name target-cluster \
  --principal-arn arn:aws:iam::111122223333:role/ArgoCDCapabilityRole \
  --type STANDARD \
  --kubernetes-groups system:masters
```

Note

本番稼働で使用する場合は、`system:masters` ではなくより制限の厳しい Kubernetes グループを使用することを検討してください。

プライベートクラスターアクセス

Argo CD マネージド機能では、VPC ピアリングや特殊なネットワーク設定を施すことなく、完全にプライベートな EKS クラスターにデプロイできます。AWS が Argo CD 機能とプライベートリモートクラスターとの接続を自動的に管理します。リポジトリのアクセスコントロールと Argo CD RBAC ポリシーが正しく設定されていることを確認します。

クロスアカウントデプロイ

クロスアカウントデプロイの場合は、ソースアカウントの Argo CD IAM 機能ロールをターゲットクラスターの EKS アクセスエントリに追加します。

1. ターゲットアカウントでは、ターゲット EKS クラスターにアクセスエントリを作成します。
2. ソースアカウントの Argo CD IAM 機能ロール ARN をプリンシパルとして使用します。
3. アクセスエントリに適切な Kubernetes RBAC アクセス許可を設定します。
4. EKS クラスター ARN を使用して Argo CD にターゲットクラスターを登録します。

追加で IAM ロールを作成したり、信頼ポリシーを設定したりする必要はありません。EKS アクセスエントリがクロスアカウントアクセスを処理します。

ベストプラクティス

宣言型ソースを信頼できるソースとして使用する: すべてのアプリケーションマニフェストを宣言型ソース (Git リポジトリ、Helm レジストリ、または OCI イメージ) に保存して、バージョン管理、監査証跡、およびコラボレーションを有効にできます。

適切な RBAC を実装する: AWS アイデンティティセンター統合を使用すると、Argo CD でアプリケーションにアクセスして管理できるユーザーを制御できます。Argo CD では、Application 内のリソース (デプロイ、ポッド、ConfigMap、シークレット) へのアクセスをきめ細かく制御できます。

マルチ環境デプロイに ApplicationSet を使用する: ApplicationSet を使用すると、設定が異なる複数のクラスターまたは名前空間にアプリケーションをデプロイできます。

ライフサイクル管理

アプリケーション同期ポリシー

Argo CD によるアプリケーション同期の方法を制御します。

手動同期: アプリケーションで変更を同期するには、手動承認が必要です。本番稼働環境に推奨されます。

自動同期: Git での変更が検出されると、アプリケーションが自動的に同期します。開発環境とステージング環境でよく使用されます。

自己修復: クラスターに手動で加えられた変更を自動的に元に戻します。クラスターの状態が Git と一致するようになります。

プルーニング: Git から除去されたリソースを自動的に削除します。リソースを削除できるため、使用には注意が必要です。

アプリケーションのヘルス

Argo CD は、アプリケーションのヘルスを継続的にモニタリングします。

- 正常: すべてのリソースが想定どおりに動作しています。
- 進行中: リソースは作成中または更新中です。
- 低下: 一部のリソースが正常ではありません。
- 中断: アプリケーションが一時停止しています。
- 欠落: クラスターにリソースがありません。

同期期間

同期期間を設定して、どのような場合にアプリケーションを同期できるかを制御します。

- メンテナンス期間中にのみ同期を許可します。
- 営業時間中の同期をブロックします。
- 特定の時間に自動同期をスケジュールします。
- 変更を行い、同期を停止する必要がある状況で同期ウィンドウを使用する (ブレークグラスシナリオ)

ウェブフックを設定して同期を高速化する

デフォルトでは、Argo CD は 6 分ごとに Git リポジトリをポーリングして変更がないか検出します。デプロイの応答性を高めるには、変更がプッシュされたらすぐに同期をトリガーするように Git ウェブフックを設定します。

ウェブフックには、次のような利点があります。

- コードがプッシュされたらすぐに同期応答が返されます (秒か分かの違い)

- ポーリングのオーバーヘッドが軽減され、システムのパフォーマンスが向上します。
- API レートリミットをより効率的に使用できます。
- フィードバックがすぐに返されるので、ユーザーエクスペリエンスが向上します。

ウェブフックエンドポイント

ウェブフック URL はパターン `${serverUrl}/api/webhook` に従います。ここでは、Argo CD サーバー URL は `serverUrl` です。サーバーの URL を確認するには、「the section called “Argo CD エンドポイント URL”」を参照してください。

例えば、Argo CD サーバー URL が `https://my-argocd-dc855fdf-111122223333.eks-capabilities.us-west-2.amazonaws.com` の場合、ウェブフック URL は次のようになります。

```
https://my-argocd-dc855fdf-111122223333.eks-capabilities.us-west-2.amazonaws.com/api/webhook
```

Git プロバイダーでウェブフックを設定する

GitHub: リポジトリ設定に、Argo CD ウェブフック URL と共にウェブフックを追加します。コンテンツタイプを `application/json` に設定し、「Just the push event」を選択します。

GitLab: Project 設定に、Argo CD ウェブフック URL と共にウェブフックを追加します。「Push events」とオプションで「Tag push events」を有効にします。

Bitbucket: リポジトリ設定に、Argo CD ウェブフック URL と共にウェブフックを追加します。トリガーとして「Repository push」を選択します。

CodeCommit: Amazon EventBridge に、CodeCommit リポジトリの状態が変化したらトリガーして、Argo CD ウェブフックエンドポイントに通知を送信するというルールを作成します。

ウェブフック設定の詳細な手順については、「Argo CD ウェブフック設定」を参照してください。

> **Note**
>
> ウェブフックは、ポーリングを補完するものであり、ポーリングに置き換わるものではありません。Argo CD は、ウェブフック通知が見逃された場合に備えて、フォールバックメカニズムとしてリポジトリを継続的にポーリングします。

次のステップ

- the section called “Argo CD の使用” - Argo CD Application を作成および管理する方法を学ぶ
- the section called “トラブルシューティング” - Argo CD の問題をトラブルシューティングする
- the section called “機能の使用” - Argo CD 機能リソースを管理する

Argo CD 機能に関する問題をトラブルシューティングする

> **Note**
>
> EKS の機能は完全に管理され、クラスターの外部で実行されます。コントローラーの名前空間に直接アクセスすることはできません。コントローラーの動作の可視化のためにコントローラーのログ配信を設定できます。「the section called “機能コントローラーログ”」を参照してください。トラブルシューティングでは、機能のヘルス、アプリケーションのステータス、および設定に焦点を当てています。

機能が ACTIVE であるのにアプリケーションが同期しない

Argo CD 機能のステータスが `ACTIVE` であるのにアプリケーションが同期しない場合は、機能のヘルスとアプリケーションのステータスを確認してください。

機能のヘルスを確認する:

機能のヘルスとステータスの問題は、EKS コンソールまたは AWS CLI を使用して表示できます。

コンソール:

1. https://console.aws.amazon.com/eks/home#/clusters で Amazon EKS コンソールを開きます。
2. クラスター名を選択します。
3. [オブザーバビリティ] タブを選択します。
4. [クラスターを監視する] を選択します。
5. [機能] タブを選択すると、すべての機能のヘルスとステータスが表示されます。

AWSCLI:

```
# View capability status and health
```

```
aws eks describe-capability \
  --region region-code \
  --cluster-name my-cluster \
  --capability-name my-argocd

# Look for issues in the health section
```

一般的な原因:

- リポジトリが設定されていない: Git リポジトリが Argo CD に追加されていません。
- 認証に失敗した: SSH キー、トークン、または CodeCommit 認証情報が無効です。
- Application が作成されていない: クラスターに Application リソースが存在しません。
- 同期ポリシー: 手動同期が必要です (自動同期は有効になっていません)。
- IAM アクセス許可: CodeCommit または Secrets Manager に対するアクセス許可がありません。

アプリケーションのステータスを確認する:

```
# List applications
kubectl get application -n argocd

# View sync status
kubectl get application my-app -n argocd -o jsonpath='{.status.sync.status}'

# View application health
kubectl get application my-app -n argocd -o jsonpath='{.status.health}'
```

アプリケーションの条件を確認する:

```
# Describe application to see detailed status
kubectl describe application my-app -n argocd

# View application health
kubectl get application my-app -n argocd -o jsonpath='{.status.health}'
```

アプリケーションが「進行中」のまま変化しない

アプリケーションが `Progressing` であるのに `Healthy` に達しない場合は、アプリケーションのリソースステータスとイベントを確認してください。

リソースのヘルスを確認する:

```
# View application resources
kubectl get application my-app -n argocd -o jsonpath='{.status.resources}'

# Check for unhealthy resources
kubectl describe application my-app -n argocd | grep -A 10 "Health Status"
```

一般的な原因:

- デプロイの準備が完了していない: ポッドの起動に失敗したか、準備状況プローブが失敗しています。
- リソースの依存関係: リソースが他のリソースの準備を待っています。
- イメージプルエラー: コンテナイメージにアクセスできません。
- リソースが不十分: クラスターでポッドの CPU やメモリが不足しています。

ターゲットクラスターの設定を検証する (マルチクラスター設定の場合):

```
# List registered clusters
kubectl get secret -n argocd -l argocd.argoproj.io/secret-type=cluster

# View cluster secret details
kubectl get secret cluster-secret-name -n argocd -o yaml
```

リポジトリの認証に失敗する

Argo CD が Git リポジトリにアクセスできない場合は、認証設定を確認してください。

CodeCommit リポジトリの場合:

IAM 機能ロールに CodeCommit へのアクセス許可があることを確認します。

```
# View IAM policies
aws iam list-attached-role-policies --role-name my-argocd-capability-role
aws iam list-role-policies --role-name my-argocd-capability-role

# Get specific policy details
aws iam get-role-policy --role-name my-argocd-capability-role --policy-name policy-name
```

ロールにはリポジトリに対する codecommit:GitPull アクセス許可が必要です。

プライベート Git リポジトリの場合:

リポジトリの認証情報が正しく設定されていることを確認します。

```
# Check repository secret exists
kubectl get secret -n argocd repo-secret-name -o yaml
```

シークレットに適切な認証情報 (SSH キー、トークン、またはユーザー名/パスワード) が含まれていることを確認します。

Secrets Manager を使用するリポジトリの場合:

```
# Verify IAM Capability Role has Secrets Manager permissions
aws iam list-attached-role-policies --role-name my-argocd-capability-role

# Test secret retrieval
aws secretsmanager get-secret-value --secret-id arn:aws:secretsmanager:region-
code:111122223333:secret:my-secret
```

マルチクラスターデプロイに関する問題

アプリケーションがリモートクラスターにデプロイされていない場合は、クラスターの登録とアクセスの設定を確認してください。

クラスターの登録を確認する:

```
# List registered clusters
kubectl get secret -n argocd -l argocd.argoproj.io/secret-type=cluster

# Verify cluster secret format
kubectl get secret CLUSTER_SECRET_NAME -n argocd -o yaml
```

server フィールドに、Kubernetes API URL ではなく EKS クラスター ARN が含まれていることを確認します。

ターゲットクラスターのアクセスエントリを確認する:

ターゲットクラスターで、Argo CD 機能ロールにアクセスエントリがあることを確認します。

```
# List access entries (run on target cluster or use AWS CLI)
aws eks list-access-entries --cluster-name target-cluster

# Describe specific access entry
aws eks describe-access-entry \
  --cluster-name target-cluster \
  --principal-arn arn:aws:iam::111122223333:role/my-argocd-capability-role
```

クロスアカウントの IAM アクセス許可を確認する:

クロスアカウントデプロイの場合、ターゲットクラスターで Argo CD 機能ロールにアクセスエントリがあることを確認します。マネージド機能は、IAM ロールの引き受けではなく、クロスアカウントアクセスの EKS アクセスエントリを使用します。

マルチクラスター設定の詳細については、「the section called “クラスターを登録する”」を参照してください。

アプリケーションの同期時間の増加

アプリケーションが同期しているが、予想以上に時間がかかる場合は、次の診断ステップを使用して原因を特定します。

最後の同期時間を確認する

アプリケーションが最後に同期されたタイミングをレビューして、遅延を確認します:

```
# View last sync time for all applications
kubectl get application -n argocd -o jsonpath='{range .items[*]}{.metadata.name}{"\t"}
{.status.operationState.finishedAt}{"\n"}{end}'

# View last sync time for a specific application
kubectl get application my-app -n argocd -o
 jsonpath='{.status.operationState.finishedAt}'
```

アプリケーションの条件を確認する

調整キューの遅延に関するアプリケーション条件をレビューします:

```
# Check conditions on an application
```

```
kubectl get application my-app -n argocd -o jsonpath='{.status.conditions}'
```

targetRevision 設定を確認する

`targetRevision: HEAD` を使用するアプリケーションは、リポジトリへのコミットごとにマニフェストキャッシュを無効化するため、同期時間が遅くなります。

```
# List applications using HEAD as targetRevision
kubectl get application -n argocd -o jsonpath='{range .items[?
(@.spec.source.targetRevision=="HEAD")]}{.metadata.name}{"\n"}{end}'
```

一般的な原因

- ウェブフック設定なし: ウェブフックがない場合、Argo CD はデフォルトの 6 分間隔でリポジトリをポーリングします。これにより、新しいコミットの検出が遅れます。
- HEAD に設定された targetRevision: リポジトリへのコミットごとにマニフェストキャッシュが無効化されます。次に、Argo CD は各調整でマニフェストを再生成します。
- 大規模または複雑な Git リポジトリ: モノレポまたは複雑な Helm チャートでは、処理するファイルやテンプレートの量が多いため、マニフェストの生成が遅くなります。
- 単一のアプリケーションでの多数の Kubernetes リソース: 多くのリソースを管理するアプリケーションでは、Argo CD が各リソースの状態を追跡する必要があるため、クラスターキャッシュの同期が遅くなります。

緩和策

- Git ウェブフックを設定する: ウェブフックは、変更がプッシュされるとすぐに Argo CD に通知し、デフォルトのポーリング間隔をバイパスします。設定手順については、「the section called “Argo CD に関する考慮事項”」を参照してください。
- 特定のブランチ名またはコミット SHA を使用する: 同期の間にマニフェストキャッシュを保持するには、`HEAD` の代わりに `targetRevision` を ブランチ名またはコミット SHA に設定します。
- 大きなモノレポジトリを分割する: 大きなリポジトリをより小さく、焦点を絞ったリポジトリに分割して、マニフェストの生成時間を短縮します。
- アプリケーションあたりのリソースを削減する: Kubernetes リソースが多いアプリケーションを複数の小さなアプリケーションに分割して、クラスターキャッシュの同期時間を短縮します。
- コントローラーログ配信を有効にする: コントローラーログは、調整動作とキュー処理を可視化します。設定手順については、「the section called “機能コントローラーログ”」を参照してください。

アプリケーションが繰り返し同期するか、同期がスタックする

アプリケーションが同期されてすぐに OutOfSync になる場合、または同期ループでスタックしたままの場合、通常、Git が定義するものとクラスター内に存在するものの間のドリフトが原因です。ベースライン診断から始めます。

診断情報を収集する

```
# View current sync and health status
argocd app get my-app

# Show exact fields that differ between Git and live state
argocd app diff my-app

# Check whether the app has ever reached a stable state
argocd app history my-app
```

argocd app diff コマンドは最も便利な開始点です。アプリケーションが同期しない原因となるフィールドが正確に表示されます。

セルフマネージド証明書によってドリフトが発生する

cert-manager、OPA Gatekeeper、KEDA などのコントローラーは、ランタイム時に証明書を生成します。これらのランタイム値は Git にないため、Argo CD はすべての調整でドリフトを検出します。

症状は次のとおりです:

- アプリケーションの同期後、すぐに OutOfSync が表示される
- ドリフトが、ウェブフック caBundle フィールドまたは TLS シークレット dataフ ィールドでの変更を示す

これを解決するには、影響を受けるフィールドで ignoreDifferences を追加し、同期オプションで RespectIgnoreDifferences を有効にします。

```
apiVersion: argoproj.io/v1alpha1
kind: Application
metadata:
  name: my-app
spec:
```

```
  ignoreDifferences:
    - group: admissionregistration.k8s.io
      kind: ValidatingWebhookConfiguration
      jsonPointers:
        - /webhooks/0/clientConfig/caBundle
    - group: ""
      kind: Secret
      jsonPointers:
        - /data/tls.crt
        - /data/tls.key
  syncPolicy:
    syncOptions:
      - RespectIgnoreDifferences=true
```

自己修復によって起動の遅いワークロードが中断される

selfHeal を有効にすると、Argo CD はドリフトを検出したときにアプリケーションを再同期します。ワークロードの起動に 30～60 秒かかると、ワークロードが Healthy になる前に自己修復がトリガーされます。prune を有効にすると、部分的に起動されたリソースが破損する可能性があります。

これを解決するには、まず基盤となるドリフトを修正します (証明書シナリオを参照)。ドリフトが原因でない場合は、Git を通じてのみ管理するワークロードの自己修復を無効にすることを検討してください。

```
apiVersion: argoproj.io/v1alpha1
kind: Application
metadata:
  name: my-app
spec:
  syncPolicy:
    automated:
      selfHeal: false
      prune: false
```

> **Note**
>
> 自己修復バックオフタイミングは、インスタンスレベルのコントローラー設定です。無効にするのではなく自己修復タイミングを調整する必要がある場合は、AWS サポートケースを開きます。

ApplicationSet またはリソース所有権が衝突する

2 つの Applications または ApplicationSets が同じ Kubernetes リソースを管理している場合、Argo CD には SharedResourceWarning が表示されます。リソースが安定状態に到達することはありません。これは通常、共有リソース名が環境またはクラスターごとにスコープされていない場合に発生します。

これを解決するには:

- 競合しているリソースを所有者ごとに一意にします。リソース名に環境またはクラスターサフィックスを追加します。
- ApplicationSet の名前を変更するときは、既存のリソースが破壊的に破損されないように、最初に preserveResourcesOnDeletion: true を設定します。

```
apiVersion: argoproj.io/v1alpha1
kind: ApplicationSet
metadata:
  name: my-appset
spec:
  syncPolicy:
    preserveResourcesOnDeletion: true
```

リソースファイナライザーによる削除のスタック

アプリケーションが Terminating 状態でスタックする場合、または「削除のために N 個のオブジェクトが残っている」と表示される場合、resources-finalizer.argocd.argoproj.io ファイナライザーはすべてのマネージドリソースが削除されるまで削除をブロックします。独自の未処理のファイナライザーを持つマネージドリソースは、削除を無期限にブロックします。

確認するには、削除タイムスタンプがあるが、削除されていないリソースを一覧表示します。

```
kubectl get all -n my-namespace -o json | \
  jq '.items[] | select(.metadata.deletionTimestamp != null) | {name: .metadata.name,
 kind: .kind, finalizers: .metadata.finalizers}'
```

これを解決するには:

- ブロッキングファイナライザーを所有するコントローラーが正常で実行中であることを確認します。

- 所有コントローラーが正常であるが、ファイナライザーが処理されていない場合は、スタックしたリソースからブロッキングファイナライザーを削除します。

前のコマンドが出力したファイナライザーのリストを使用して、リストを保持するファイナライザーに設定し、ブロックしているファイナライザーだけを除外します。finalizer-a と finalizer-b をそれらの名前に置き換えます。

```
kubectl patch resource-kind
            resource-name -n my-namespace \
  --type merge -p '{"metadata":{"finalizers":["finalizer-a","finalizer-b"]}}'
```

ブロックしているファイナライザーがリソース上の唯一のものである場合は、空のリスト --type merge -p '{"metadata":{"finalizers":[]}}' を渡します。

> **Warning**
>
> 位置を指定してエントリを削除するのではなく、ファイナライザーのリストを明示的に設定します。リソースは複数のコントローラーからのファイナライザーを保持していることがあります。ファイナライザーの順序は保証されていません。最初のエントリを削除すると誤ったファイナライザーが削除される可能性があり、その結果、ブロックしているファイナライザーが残ったままになり、リソースはスタックしたままになります。
> ファイナライザーを削除すると、所有コントローラーが実行したはずのクリーンアップもスキップされるため、クラスターに記録が残らないまま AWS リソースが取り残される可能性があります。ファイナライザーを削除するのは、所有コントローラーがファイナライザーを処理できないことを確認した後だけにしてください。

同期が失敗しても同じリビジョンに自動的に再試行されない

特定のリビジョンへの同期が失敗すると、Argo CD は同じリビジョンを自動再試行しません。これは通常、重複する環境変数キーからの ComparisonError などのマニフェストの欠陥が原因で発生します。

アプリケーションのステータスをチェックして確認します:

```
argocd app get my-app
# Look for: Operation: Sync  Phase: Failed  Revision: <sha>
```

これを解決するには、Git リポジトリのマニフェストの不具合を修正し、新しいコミットをプッシュします。または、手動同期をトリガーします:

```
argocd app sync my-app
```

モノレポのコミットチャーンが広範な再生成をトリガーする

多くのアプリケーションが同じリポジトリで HEAD を追跡している場合、そのリポジトリへのコミットはすべてのアプリケーションの HEAD を変更します。これにより、それらのファイルが変更されなかった場合でも、すべてのアプリケーションに対してマニフェストの再生成がトリガーされます。targetRevision とキャッシュの詳細については、このページの「アプリケーション同期時間の増加」セクションを参照してください。

各アプリケーションが使用するファイルのみに再生成をスコープするには、manifest-generate-paths 注釈を追加します。

```
apiVersion: argoproj.io/v1alpha1
kind: Application
metadata:
  name: my-app
  annotations:
    argocd.argoproj.io/manifest-generate-paths: /apps/my-app
spec:
  source:
    repoURL: https://github.com/my-org/my-monorepo.git
    targetRevision: HEAD
    path: apps/my-app
```

この注釈を使用すると、Argo CD は指定されたパスの下にあるファイルが変更された場合にのみマニフェストを再生成します。アプリケーション間で使用されている共有ライブラリでは、複数のパスをセミコロン (;) で区切って指定できます。

可能な場合は、HEAD の代わりにブランチ名またはタグに targetRevision をピン留めします。

Kubernetes のデフォルトおよび変異ウェブフックによりファントム差分が発生する

同期の直後にアプリケーションが OutOfSync を表示する場合は、設定していないフィールド (terminationGracePeriodSeconds、dnsPolicy、/spec/replicas など) の差分をチェックします。Kubernetes API サーバーまたは変異ウェブフックが、適用時にこれらのフィールドを追加しました。

別のコントローラーによって管理されているフィールド (HPA がスケーリングを管理する場合の `/spec/replicas` など) でこれを解決するには、`ignoreDifferences` を追加します。

```
apiVersion: argoproj.io/v1alpha1
kind: Application
metadata:
  name: my-app
spec:
  ignoreDifferences:
    - group: apps
      kind: Deployment
      jsonPointers:
        - /spec/replicas
  syncPolicy:
    syncOptions:
      - RespectIgnoreDifferences=true
```

Kubernetes のデフォルトまたは変異ウェブフックによって追加されたフィールドでは、アプリケーションでサーバー側の差分を有効にできます。

```
apiVersion: argoproj.io/v1alpha1
kind: Application
metadata:
  name: my-app
  annotations:
    argocd.argoproj.io/compare-options: ServerSideDiff=true,IncludeMutationWebhook=true
```

サーバー側の差分は、リソースごとにドライラン適用を実行するため、Kubernetes API サーバーの負荷が増加します。これを広範に有効にする前に、少数のアプリケーションでテストしてください。

高チャーンのコントローラー所有リソース

一部のコントローラーは、有効期間の短い、または頻繁に更新される多数のリソースを生成します。例としては、Karpenter ノードオブジェクト、Cilium ID とエンドポイントオブジェクト、Kyverno ポリシーレポートなどがあります。これらのリソースが大量のウォッチイベントを生成し、同期チャーンを引き起こす場合、それらのリソースタイプを除外するか、ウォッチイベントをフィルタリングすることで、負荷を軽減できます。これらの変更には、インスタンスレベルのコントローラー設定が必要です。

マネージド機能で、AWS サポートケースを開いて、これらのリソースタイプのリソース除外またはウォッチイベントフィルタリングをリクエストします。

ベストプラクティス

- 最初にアプリケーション差分を使用する: 同期が繰り返される問題の最初の診断ステップとして `argocd app diff` を実行します。ドリフトの正確な原因が示されます。
- ignoreDifferences を絞り込む: 特定のリソースタイプで特定のフィールドをターゲットにします。実際の設定ドリフトをマスクできる広範な無視ルールは避けてください。
- ignoreDifferences と RespectIgnoreDifferences を組み合わせる: 常に `RespectIgnoreDifferences=true` 同期オプションを追加します。これがない場合、同期は無視されたフィールドを上書きします。
- リソース名を一意に保つ: Applications または ApplicationSets の間の所有権の衝突を避けるため、環境とクラスターごとにリソース名をスコープします。
- prune と selfHeal に注意する: 起動に時間がかかるワークロードでは、両方を有効にしないでください。自己修復は、リソースが正常になる前にリソースを破損する可能性があります。
- targetRevision をピン留めしてマニフェストパスをスコープする: 大規模な共有リポジトリ内のアプリケーションでは、`HEAD` の代わりにブランチまたはタグを使用して、`manifest-generate-paths` 注釈を追加します。

AWS サポートに連絡すべきケース

以下の状況では AWS サポートケースを開いてください:

- インスタンスレベルのコントローラー調整が必要と思われる (プロセッサ数、自己修復タイミング、またはリソースの除外)。
- リポジトリサーバーまたはコントローラーの容量がアプリケーション数には不十分であると思われる。
- ワークロード設定、ドリフト、所有権、またはファイナライザーでは動作を説明できない。

サポートケースには、影響を受けるアプリケーションの `argocd app get` および `argocd app diff` の出力を含めてください。

次のステップ

- the section called “Argo CD に関する考慮事項” - Argo CD の考慮事項とベストプラクティス
- the section called “Argo CD の使用” - Argo CD Application を作成して管理する
- the section called “クラスターを登録する” - マルチクラスターデプロイを設定する

- the section called “機能をトラブルシューティングする” - 一般的な機能をトラブルシューティングする際のガイダンス

EKS Capability for Argo CD とセルフマネージド Argo CD を比較する

EKS Capability for Argo CD により、完全マネージド型の Argo CD を EKS で動作させることができます。EKS 機能とセルフマネージドソリューションの全般的な比較については、「the section called “考慮事項”」を参照してください。このトピックでは、認証、マルチクラスター管理、アップストリーム機能のサポートなど、Argo CD 固有の違いに焦点を当てます。

アップストリーム Argo CD との違い

EKS Capability for Argo CD は、アップストリームの Argo CD に基づいていますが、アクセス方法、設定方法、AWS サービスとの統合方法が異なります。

RBAC と認証: 機能が備える RBAC ロールには管理者、編集者、ビューワーの 3 つがあり、Argo CD に組み込みの認証ではなく AWS アイデンティティセンターを使用して認証を行います。Argo CD の `argocd-rbac-cm` ConfigMap ではなく、機能の `rbacRoleMapping` パラメータを使用して、アイデンティティセンターのグループを Argo CD ロールにマッピングするように、ロールマッピングを設定します。Argo CD UI は独自のダイレクト URL (クラスターの [機能] タブの EKS コンソールで確認) でホストされ、API アクセスは IAM を介した AWS 認証と認可を使用します。

クラスター設定: ローカルクラスターやハブアンドスポークトポロジは自動的には設定されません。自分でデプロイターゲットクラスターと EKS アクセスエントリを設定します。Amazon EKS クラスターがデプロイターゲットとしてサポートされるのは、Kubernetes API サーバー URL ではなく EKS クラスター ARN を使用している場合のみです。ローカルクラスター (`kubernetes.default.svc`) がデプロイターゲットとして自動的に追加されることはありません。機能の作成先と同じクラスターにデプロイするには、ARN を使用してそのクラスターを明示的に登録します。

シンプルなリモートクラスターアクセス: EKS アクセスエントリを使用して Argo CD にリモートクラスターへのアクセスを許可することで、マルチクラスターのデプロイをシンプルにします。そのため、サービスアカウントの IAM ロール (IRSA) を設定したり、クロスアカウント IAM ロールの前提条件をセットアップしたりする必要がありません。また、VPC ピアリングや特殊なネットワーク設定を施すことなく、完全にプライベートな EKS クラスターに透過的にアクセスできます。AWS が Argo CD 機能とプライベートリモートクラスターとの接続を自動的に管理します。

AWS サービスの直接統合: 機能ロールの IAM アクセス許可により、AWS サービスを直接統合します。リポジトリ設定を作成しなくても、CodeCommit リポジトリ、ECR Helm チャート、および

CodeConnections を Application リソースで直接参照できます。これにより、認証がシンプルになり、AWS サービスごとに認証情報を管理する必要がありません。詳細については、「the section called “リポジトリを設定する”」を参照してください。

名前空間のサポート: この機能では、Argo CD Application、ApplicationSet、および AppProject カスタムリソースを作成する必要がある単一の名前空間を指定する必要があります。

> **Note**
>
> この名前空間の制限は、Argo CD 独自のカスタムリソース (Application、ApplicationSet、AppProject) にのみ適用されます。アプリケーションワークロードは、任意のターゲットクラスター内の任意の名前空間にデプロイできます。例えば、名前空間 `argocd` でこの機能を作成する場合、すべての Application CR を `argocd` 名前空間に作成する必要がありますが、それらの Application はワークロードを `default`、`production`、`staging`、またはその他の名前空間にデプロイできます。

> **Note**
>
> マネージド機能には、CLI の使用と AppProject 設定に関する特定の要件があります。
>
> - Argo CD CLI を使用する場合は、名前空間プレフィックス `argocd app sync namespace/appname` を使用してアプリケーションを指定します。
> - AppProject リソースは `.spec.sourceNamespaces` を指定して、プロジェクトが Application を監視できる名前空間を定義する必要があります (通常は、機能の作成時に指定した名前空間を設定します)。
> - リソース追跡注釈は形式 `namespace_appname:group/kind:namespace/name` を使用します。

サポートされていない機能: 次の機能は、マネージド機能では使用できません。

- Config Management Plugin (CMP) によるカスタムマニフェストの生成
- Notifications コントローラー
- カスタム SSO プロバイダー (AWS アイデンティティセンターを利用したサードパーティーのフェデレーション ID を含め、AWS アイデンティティセンターのみがサポートされています)
- UI の拡張機能とカスタムのバナー

- `argocd-params` およびほとんどの設定 ConfigMap への直接アクセス (`argocd-cm` フィールドのサブセットがサポートされています。「the section called “設定を構成する”」を参照してください)
- 同期タイムアウトの変更 (120 秒に固定)

互換性: Application と ApplicationSet は、アップストリーム Argo CD と同じように動作し、マニフェストの変更は不要です。この機能は同じ Kubernetes API と CRD を使用するため、`kubectl` のようなツールは同じように機能します。完全にサポートされるのは、Application と ApplicationSet、自動同期を備えた GitOps ワークフロー、マルチクラスターデプロイ、同期ポリシー (自動、プルーニング、自己修復)、同期ウェーブと同期フック、標準 Kubernetes リソースのヘルス評価、ロールバック機能、Git リポジトリソース (HTTPS と SSH)、Helm、Kustomize、プレーン YAML マニフェスト、GitHub アプリ認証情報、マルチテナンシーのプロジェクト、リソースの除外と包含です。

マネージド機能で Argo CD CLI を使用する

Argo CD CLI は、ほとんどのオペレーションでアップストリーム Argo CD と同じように動作しますが、認証とクラスター登録が異なります。

前提条件

アップストリームインストール手順に従って Argo CD CLI をインストールしていること。

設定

環境変数を使用して CLI を設定する:

1. EKS コンソール (クラスターの [機能] タブの下) または AWS CLI を使用して、Argo CD サーバー URL を取得します。`https://` プレフィックスを削除する必要があります。

```
export ARGOCD_SERVER=$(aws eks describe-capability \
  --cluster-name my-cluster \
  --capability-name my-argocd \
  --query 'capability.configuration.argoCd.serverUrl' \
  --output text \
  --region region-code | sed 's|^https://||')
```

2. Argo CD UI ([設定] → [アカウント] → [管理者] → [新しいトークンを生成]) からアカウントトークンを生成して、環境変数として設定します。

```
export ARGOCD_AUTH_TOKEN="your-token-here"
```

> **Important**
>
> このように生成した管理者アカウントトークンは、初期セットアップと開発ワークフローに使用します。本番稼働のユースケースでは、プロジェクト範囲のロールとトークンを使用して、最小特権の原則に従います。Project のロールと RBAC の詳細については、「the section called “アクセス許可の設定”」を参照してください。

1. 必要な gRPC オプションを設定します。

```
export ARGOCD_OPTS="--grpc-web"
```

これらの環境変数を設定すると、`argocd login` コマンドを実行しなくても Argo CD CLI を使用できます。

主な違い

マネージド機能では、CLI に次の制限があります。

- `argocd admin` コマンドはサポートされていません (直接ポッドアクセスが必要です)。
- `argocd login` はサポートされていません (代わりにアカウントトークンまたはプロジェクトトークンを使用します)。
- `argocd cluster add` には、EKS クラスター ARN を指定した `--aws-cluster-name` フラグが必要です。

例: クラスターを登録する

アプリケーションをデプロイするための EKS クラスターを登録します。

```
# Get the cluster ARN
CLUSTER_ARN=$(aws eks describe-cluster \
  --name my-cluster \
  --query 'cluster.arn' \
  --output text)

# Register the cluster
argocd cluster add $CLUSTER_ARN \
  --aws-cluster-name $CLUSTER_ARN \
```

```
  --name in-cluster \
  --project default
```

Argo CD CLI の詳細なドキュメントについては、「Argo CD CLI リファレンス」を参照してください。

移行パス

セルフマネージド Argo CD からマネージド機能に移行できます。

1. サポートされていない機能 (Notifications コントローラー、CMP、カスタムヘルスチェック、UI 拡張機能) の現在の Argo CD 設定を確認します。
2. セルフマネージド Argo CD コントローラーをゼロレプリカにスケールして、競合を防ぐ
3. クラスターに Argo CD 機能リソースを作成します。
4. 既存の Application、ApplicationSet、AppProject をエクスポートします。
5. リポジトリ認証情報、クラスターシークレット、リポジトリ認証情報テンプレート (repocreds) を移行します。
6. GPG キー、TLS 証明書、または SSH 既知のホストを使用する場合は、こうした設定も移行します。
7. クラスター名または EKS クラスター ARN を使用するように `destination.server` フィールドを更新します。
8. それらをマネージド Argo CD インスタンスに適用します。
9. アプリケーションが正しく同期されていることを確認します。
10. セルフマネージド Argo CD インストールを廃止します。

マネージド機能は同じ Argo CD API とリソース定義を使用するため、既存のマニフェストは最小限の変更を加えるだけで動作します。

次のステップ

- the section called “Argo CD 機能を作成する” - Argo CD 機能を作成する
- the section called “Argo CD の使用” - 最初のアプリケーションをデプロイする
- the section called “Argo CD に関する考慮事項” - AWS アイデンティティセンターの統合を設定する

kro (Kube Resource Orchestrator) を使用したリソース構成

kro (Kube Resource Orchestrator) はオープンソースの Kubernetes ネイティブプロジェクトで、シンプルで簡単な設定を使用してカスタム Kubernetes API を定義できます。kro を使用すると、Kubernetes オブジェクトのグループとその間の論理オペレーションを作成する新しいカスタム API を簡単に設定できます。

AWS では EKS の機能を使用して kro を完全に管理できるため、クラスターに kro コントローラーを導入して保守およびスケールする必要がありません。

kro の仕組み

kro では、カスタム Kubernetes API を簡潔かつ効率的に作成できる `ResourceGraphDefinition` (RGD) というカスタムリソース定義 (CRD) が導入されています。`ResourceGraphDefinition` を作成すると、kro はネイティブの Kubernetes 拡張機能を使用して、クラスターに新しい API を作成および管理します。この単一のリソース仕様から、kro は仕様に基づいて新しい CRD を作成して登録し、新しく定義されたカスタムリソースを管理するように適応します。

RGD には複数のリソースを含めることができ、kro が相互依存関係とリソースの順序を決定するため、ユーザーがそれを行う必要はありません。シンプルな構文を使用して、あるリソースから別のリソースに設定を挿入できるため、構成が大幅に簡素化され、クラスター内の「グルー」演算子が不要になります。kro を使用すると、カスタムリソースには、ネイティブの Kubernetes リソースだけでなく、クラスターにインストールされている任意のカスタムリソース定義 (CRD) も含めることができます。

kro は、1 つの主要なリソースタイプをサポートしています。

- ResourceGraphDefinition (RGD): Kubernetes カスタムリソースを定義し、1 つ以上の基盤となるネイティブまたはカスタム Kubernetes リソースをカプセル化します。

このリソースに加えて、kro は、RGD を使用して作成されたカスタムリソースと、それらを構成するすべてのリソースのライフサイクルを作成および管理します。

kro は AWS Controllers for Kubernetes (ACK) とシームレスに統合されており、ワークロードリソースと AWS リソースを組み合わせて、より高いレベルの抽象化を作成できます。これにより、独自のクラウド構成要素を作成し、リソース管理を簡素化し、組織標準に基づくデフォルトおよびイミュータブルな構成設定で再利用可能なパターンを有効にすることができます。

kro のメリット

kro を使用すると、プラットフォームチームは、複数のリソースを高レベルの抽象化に構成するカスタム Kubernetes API を作成できます。これにより、開発者は、シンプルで標準化され、バージョン管理されたカスタムリソースを使用して複雑なアプリケーションをデプロイできるため、リソース管理が簡素化されます。一般的なリソースの組み合わせに再利用可能なパターンを定義することで、組織全体で一貫したリソースの作成が可能になります。

kro は、Kubernetes の Common Expression Language (CEL) を使用してリソース間で値を渡し、条件付きロジックを組み込むことで、リソース構成に柔軟性を提供します。Kubernetes リソースと、ACK によって管理される AWS リソースの両方を統合したカスタム API に構成できるため、アプリケーションとインフラストラクチャの完全な定義が可能になります。

kro は、Kubernetes マニフェストによる宣言型設定をサポートしており、既存の開発プロセスとシームレスに統合される GitOps ワークフローや Infrastructure as Code のプラクティスを実現します。AWS では、EKS マネージド機能の一部として、kro を完全に管理できるため、クラスターに kro コントローラーを導入して設定および保守する必要がありません。

例: ResourceGraphDefinition の作成

次の例は、デプロイとサービスを使用してウェブアプリケーションを作成するシンプルな ResourceGraphDefinition を示しています。

```
apiVersion: kro.run/v1alpha1
kind: ResourceGraphDefinition
metadata:
  name: web-application
spec:
  schema:
    apiVersion: v1alpha1
    kind: WebApplication
    spec:
      name: string
      replicas: integer | default=3
  resources:
    - id: deployment
      template:
        apiVersion: apps/v1
        kind: Deployment
        metadata:
          name: ${schema.spec.name}
        spec:
```

```
          replicas: ${schema.spec.replicas}
    - id: service
      template:
        apiVersion: v1
        kind: Service
        metadata:
          name: ${schema.spec.name}
```

ユーザーが WebApplication カスタムリソースのインスタンスを作成すると、kro は対応するデプロイおよびサービスリソースを自動的に作成し、カスタムリソースと共にライフサイクルを管理します。

他の EKS マネージド機能との統合

kro は、他の EKS マネージド機能と統合されています。

- Kubernetes 用 AWS コントローラー (ACK): kro を使用して ACK リソースを高レベルの抽象化に構成し、AWS リソース管理を簡素化します。
- Argo CD: Argo CD を使用して、複数のクラスターにまたがる kro カスタムリソースのデプロイを管理し、プラットフォームの構成要素とアプリケーションスタックの GitOps ワークフローを有効にします。

kro の開始方法

EKS Capability for kro の使用を開始するには:

1. AWS コンソール、AWS CLI、または任意の Infrastructure as Code ツールを使用して、EKS クラスターに kro 機能リソースを作成します。
2. カスタム API とリソース構成を定義する ResourceGraphDefinitions (RGDs) を作成します。
3. カスタムリソースのインスタンスを適用して、基盤となる Kubernetes リソースと AWS リソースをプロビジョニングおよび管理します。

kro 機能の作成

このトピックでは、Amazon EKS クラスター上で kro 機能を作成する方法について説明します。

前提条件

kro 機能を作成する前に、以下があることを確認してください。

- サポート対象の Kubernetes バージョン (標準サポートと延長サポートのすべてのバージョンがサポート対象) を実行している Amazon EKS クラスターが存在する
- EKS クラスターに機能リソースを作成するための十分な IAM アクセス許可がある
- (CLI/eksctl の場合) 適切な CLI ツールがインストールおよび設定されている

> **Note**
>
> ACK や Argo CD とは異なり、kro は信頼ポリシー以外の追加の IAM アクセス許可を必要としません。kro はクラスター内で完全に動作し、AWS API コールを行いません。ただし、IAM 機能ロールに適切な信頼ポリシーを指定する必要があります。kro の Kubernetes RBAC アクセス許可の設定については、「the section called “アクセス許可の設定”」を参照してください。

ツールを選択する

AWS マネジメントコンソール、AWS CLI、または eksctl を使用して、kro 機能を作成できます。

- the section called “コンソール” - 指示に従って操作する場合はコンソールを使用します。
- the section called “ AWS CLI” - スクリプトを作成して自動化を図る場合は AWS CLI を使用します。
- the section called “eksctl” - Kubernetes ネイティブの操作をする場合は eksctl を使用します。

kro 機能を作成した場合の動作

kro 機能を作成すると、次の処理が行われます。

1. EKS は kro 機能サービスを作成し、クラスター内のリソースを監視および管理するように設定する
2. カスタムリソース定義 (CRD) がクラスターにインストールされます。
3. ResourceGraphDefinitions とそのインスタンスを管理するアクセス許可を付与する `AmazonEKSKROPolicy` を使用して、IAM 機能ロールのアクセスエントリが自動的に作成されます (「the section called “EKS 機能に関する考慮事項”」を参照)。
4. kro 機能が、指定した IAM 機能ロールを引き受ける (信頼関係にのみ使用される)
5. kro が `ResourceGraphDefinition` リソースとそのインスタンスの監視を開始する

6. 機能のステータスが CREATING から ACTIVE に変わります。

アクティブになったら、ResourceGraphDefinitions を作成してカスタム API を定義し、それらの API のインスタンスを作成できます。

> **Note**
>
> 自動的に作成されたアクセスエントリには、ResourceGraphDefinitions とそのインスタンスを管理するための kro アクセス許可を付与する AmazonEKSKROPolicy が含まれます。ResourceGraphDefinitions で定義されている基盤となる Kubernetes リソース (デプロイ、サービス、ACK リソースなど) を kro で作成するには、追加のアクセスエントリポリシーを設定する必要があります。アクセスエントリの詳細と追加のアクセス許可の設定方法については、「the section called “アクセス許可の設定”」および「the section called “EKS 機能に関する考慮事項”」を参照してください。

次のステップ

kro 機能を作成した後:

- the section called “kro の概念” - kro の概念とリソース構成を理解する

コンソールを使用して kro 機能を作成する

AWS マネジメントコンソールを使用して、Amazon EKS クラスターで kro 機能を作成します。

kro 機能を作成する

1. https://console.aws.amazon.com/eks/home#/clusters で Amazon EKS コンソールを開きます。
2. クラスター名を選択して、クラスターの詳細ページを開きます。
3. [機能] タブを選択します。
4. 左側のナビゲーションで、[kro (Kube Resource Orchestrator)] を選択します。
5. [kro 機能の作成] を選択します。
6. IAM 機能ロールに対して以下の操作を行います。
   - IAM 機能ロールが既に存在する場合は、ドロップダウンから選択します。
   - 新しいロールを作成する必要がある場合は、[kro ロールを作成] を選択します。

これにより、信頼ポリシーが事前に入力された新しいタブで IAM コンソールが開きます。kro はクラスター内で完全に動作するため、このロールには追加の IAM アクセス許可は必要ありません。

ロールを作成して EKS コンソールに戻ると、そのロールが自動的に選択されます。

> **Note**
>
> ACK や Argo CD とは異なり、kro は信頼ポリシー以外の追加の IAM アクセス許可を必要としません。kro はクラスター内で完全に動作し、AWS API コールを行いません。

7. [作成] を選択します。

機能作成プロセスが始まります。

機能がアクティブであることを確認する

1. [機能] タブで、kro 機能のステータスを表示します。
2. ステータスが `CREATING` から `ACTIVE` に変わるまで待ちます。
3. アクティブになったら、この機能を使用する準備は完了です。

機能のステータスとトラブルシューティングの詳細については、「the section called “機能の使用”」を参照してください。

Kubernetes リソースを管理するためのアクセス許可を付与する

kro 機能を作成すると、`AmazonEKSKROPolicy` を使用して EKS アクセスエントリが自動的に作成されます。これにより、kro で ResourceGraphDefinitions とそのインスタンスを管理できます。ただし、ResourceGraphDefinitions で定義されている基盤となる Kubernetes リソース (デプロイ、サービス、ConfigMaps など) を作成するアクセス許可はデフォルトでは付与されません。

この設計は、最小特権の原則に従った意図的なものです。異なる ResourceGraphDefinitions にはそれぞれ異なるアクセス許可が必要となります。それぞれの ResourceGraphDefinitions が管理するリソースに基づいて、kro に必要なアクセス許可を明示的に設定する必要があります。

迅速な開始、テスト、または開発環境については、`AmazonEKSClusterAdminPolicy` を使用します。

1. EKS コンソールで、クラスターの [アクセス] タブに移動します。

2. [アクセスエントリ] で、kro 機能ロールのエントリを見つけます (以前に作成したロール ARN が表示されます)。
3. アクセスエントリを選択して詳細を開きます。
4. [アクセスポリシー] セクションで、[アクセスポリシーを関連付ける] を選択します。
5. ポリシーリストから `AmazonEKSClusterAdminPolicy` を選択します。
6. [アクセスの範囲] で、[クラスター] を選択します。
7. 関連付ける を選択してください。

> **Important**
>
> `AmazonEKSClusterAdminPolicy` は、すべての名前空間について任意のリソースタイプを作成する機能など、すべての Kubernetes リソースを作成および管理するための広範なアクセス許可を付与します。これは開発や POC には便利ですが、本番環境では使用しないでください。本番環境では、それぞれの ResourceGraphDefinitions が管理する特定のリソースに必要なアクセス許可のみを付与する、カスタム RBAC ポリシーを作成してください。最小特権のアクセス許可の設定に関するガイダンスについては、「the section called “アクセス許可の設定”」および「the section called “EKS 機能に関する考慮事項”」を参照してください。

カスタムリソースが使用可能であることを確認する

機能がアクティブになったら、kro カスタムリソースがクラスターで使用可能になっていることを確認します。

コンソールの使用

1. Amazon EKS コントロールでクラスターに移動します。
2. [リソース] タブを選択します。
3. [拡張機能] を選択します。
4. [CustomResourceDefinitions] を選択します。

`ResourceGraphDefinition` リソースタイプがリストされます。

kubectl を使用する

```
kubectl api-resources | grep kro.run
```

ResourceGraphDefinition リソースタイプがリストされます。

次のステップ

- the section called “kro の概念” - kro の概念とリソース構成を理解する
- the section called “機能の使用” - kro 機能リソースを管理する

AWS CLI を使用して kro 機能を作成する

AWS CLI を使用して Amazon EKS クラスターで kro 機能を作成します。この手順では、IAM ロールの作成、および機能がアクティブであることの確認について説明します。

前提条件

- AWS CLI バージョン 2.12.3 以降。バージョンを確認するには、aws --version を実行します。詳細についてはAWS コマンドラインインターフェイスユーザーガイドの「インストール」を参照してください。
- **kubectl** - Kubernetes クラスターを操作するためのコマンドラインツール。詳細については、「the section called “kubectl と eksctl のセットアップ”」を参照してください。

ステップ 1: IAM 機能ロールを作成する

信頼ポリシーファイルを作成します。

```
cat > kro-trust-policy.json << 'EOF'
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "capabilities.eks.amazonaws.com"
      },
      "Action": [
        "sts:AssumeRole",
        "sts:TagSession"
      ]
    }
  ]
}
```

```
EOF
```

IAM ロールを作成します。

```
aws iam create-role \
  --role-name KROCapabilityRole \
  --assume-role-policy-document file://kro-trust-policy.json
```

> **Note**
>
> ACK や Argo CD とは異なり、kro には追加の IAM アクセス許可は必要ありません。kro はクラスター内で完全に動作し、AWS API コールを行いません。このロールは、EKS 機能サービスとの信頼関係を確立するためにのみ必要です。

ステップ 2: kro 機能を作成する

クラスターに kro 機能リソースを作成します。*region-code* をクラスターが配置されている AWS リージョン (us-west-2 など) に、*my-cluster* をクラスター名に置き換えます。

```
aws eks create-capability \
  --region region-code \
  --cluster-name my-cluster \
  --capability-name my-kro \
  --type KRO \
  --role-arn arn:aws:iam::$(aws sts get-caller-identity --query Account --output
 text):role/KROCapabilityRole \
  --delete-propagation-policy RETAIN
```

このコマンドはすぐに戻りますが、EKS が必要な機能インフラストラクチャとコンポーネントを作成するため、機能がアクティブになるまでにはしばらく時間がかかります。EKS は、この機能に関連する Kubernetes カスタムリソース定義をその作成時にクラスターにインストールします。

> **Note**
>
> クラスターが存在しないというエラーやアクセス許可がないというエラーが表示された場合は、以下の点を確認します。
>
> - クラスター名が正しいこと
> - AWS CLI が正しいリージョンに設定されていること

- 必要な IAM アクセス許可を追加したこと

ステップ 3: 機能がアクティブであることを確認する

機能がアクティブになるまで待機します。*region-code* はクラスターがある AWS リージョンに、*my-cluster* はクラスターの名前に置き換えます。

```
aws eks describe-capability \
  --region region-code \
  --cluster-name my-cluster \
  --capability-name my-kro \
  --query 'capability.status' \
  --output text
```

ステータスが ACTIVE と表示されたら、機能の準備は完了です。

機能の詳細全体を表示することもできます。

```
aws eks describe-capability \
  --region region-code \
  --cluster-name my-cluster \
  --capability-name my-kro
```

ステップ 4: Kubernetes リソースを管理するためのアクセス許可を付与する

kro 機能を作成すると、AmazonEKSKROPolicy を使用して EKS アクセスエントリが自動的に作成されます。これにより、kro で ResourceGraphDefinitions とそのインスタンスを管理できます。ただし、ResourceGraphDefinitions で定義されている基盤となる Kubernetes リソース (デプロイ、サービス、ConfigMaps など) を作成するアクセス許可はデフォルトでは付与されません。

この設計は、最小特権の原則に従った意図的なものです。異なる ResourceGraphDefinitions にはそれぞれ異なるアクセス許可が必要となります。例: *ConfigMaps と Secrets のみを作成する ResourceGraphDefinition には、デプロイとサービスを作成する ResourceGraphDefinition とは異なるアクセス許可が必要です。*ACK リソースを作成する ResourceGraphDefinition には、それらの特定のカスタムリソースに対するアクセス許可が必要です。*一部の ResourceGraphDefinitions は、新しいリソースを作成せずに既存のリソースのみを読み取る場合があります。

それぞれの ResourceGraphDefinitions が管理するリソースに基づいて、kro に必要なアクセス許可を明示的に設定する必要があります。

Quick Setup

迅速な開始、テスト、または開発環境については、AmazonEKSClusterAdminPolicy を使用します。

機能ロール ARN を取得します。

```
CAPABILITY_ROLE_ARN=$(aws eks describe-capability \
  --region region-code \
  --cluster-name my-cluster \
  --capability-name my-kro \
  --query 'capability.roleArn' \
  --output text)
```

クラスター管理ポリシーを関連付けます。

```
aws eks associate-access-policy \
  --region region-code \
  --cluster-name my-cluster \
  --principal-arn $CAPABILITY_ROLE_ARN \
  --policy-arn arn:aws:eks::aws:cluster-access-policy/AmazonEKSClusterAdminPolicy \
  --access-scope type=cluster
```

> **Important**
>
> AmazonEKSClusterAdminPolicy は、すべての名前空間について任意のリソースタイプを作成する機能など、すべての Kubernetes リソースを作成および管理するための広範なアクセス許可を付与します。これは開発や POC には便利ですが、本番環境では使用しないでください。本番環境では、それぞれの ResourceGraphDefinitions が管理する特定のリソースに必要なアクセス許可のみを付与する、カスタム RBAC ポリシーを作成してください。最小特権のアクセス許可の設定に関するガイダンスについては、「the section called “アクセス許可の設定”」および「the section called “EKS 機能に関する考慮事項”」を参照してください。

ステップ 5: カスタムリソースが使用可能であることを確認する

機能がアクティブになったら、kro カスタムリソースがクラスターで使用可能になっていることを確認します。

```
kubectl api-resources | grep kro.run
```

`ResourceGraphDefinition` リソースタイプがリストされます。

次のステップ

- the section called “kro の概念” - kro の概念とリソース構成を理解する
- the section called “機能の使用” - kro 機能リソースを管理する

eksctl を使用して kro 機能を作成する

eksctl を使用して Amazon EKS クラスターで kro 機能を作成します。

> **Note**
>
> 以下のステップを実行するには、eksctl バージョン `0.215.0` 以降が必要です。バージョンを確認するには、`eksctl version` を実行します。

ステップ 1: IAM 機能ロールを作成する

信頼ポリシーファイルを作成します。

```
cat > kro-trust-policy.json << 'EOF'
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "capabilities.eks.amazonaws.com"
      },
      "Action": [
        "sts:AssumeRole",
        "sts:TagSession"
      ]
    }
  ]
}
EOF
```

IAM ロールを作成します。

```
aws iam create-role \
  --role-name KROCapabilityRole \
  --assume-role-policy-document file://kro-trust-policy.json
```

> **Note**
>
> ACK や Argo CD とは異なり、kro は信頼ポリシー以外の追加の IAM アクセス許可を必要としません。kro はクラスター内で完全に動作し、AWS API コールを行いません。

ステップ 2: kro 機能を作成する

eksctl を使用して kro 機能を作成します。*region-code* はクラスターがある AWS リージョンに、*my-cluster* はクラスターの名前に置き換えます。

```
eksctl create capability \
  --region region-code \
  --cluster my-cluster \
  --name my-kro \
  --type KRO \
  --role-arn arn:aws:iam::111122223333:role/KROCapabilityRole
```

このコマンドはすぐに戻りますが、機能がアクティブになるまでにはしばらく時間がかかります。

ステップ 3: 機能がアクティブであることを確認する

機能のステータスを確認します。*region-code* はクラスターがある AWS リージョンに、*my-cluster* はクラスターの名前に置き換えます。

```
eksctl get capability \
  --region region-code \
  --cluster my-cluster \
  --name my-kro
```

ステータスが `ACTIVE` と表示されたら、機能の準備は完了です。

ステップ 4: Kubernetes リソースを管理するためのアクセス許可を付与する

デフォルトでは、kro は ResourceGraphDefinitions とそのインスタンスのみを作成および管理できます。kro が ResourceGraphDefinitions で定義されている基盤となる Kubernetes リソースを作成およ

び管理できるようにするには、AmazonEKSClusterAdminPolicy アクセスポリシーを機能のアクセスエントリに関連付けます。

機能ロール ARN を取得します。

```
CAPABILITY_ROLE_ARN=$(aws eks describe-capability \
  --region region-code \
  --cluster-name my-cluster \
  --capability-name my-kro \
  --query 'capability.roleArn' \
  --output text)
```

クラスター管理ポリシーを関連付けます。

```
aws eks associate-access-policy \
  --region region-code \
  --cluster-name my-cluster \
  --principal-arn $CAPABILITY_ROLE_ARN \
  --policy-arn arn:aws:eks::aws:cluster-access-policy/AmazonEKSClusterAdminPolicy \
  --access-scope type=cluster
```

> **Important**
>
> AmazonEKSClusterAdminPolicy は、すべての Kubernetes リソースを作成および管理するための広範なアクセス許可を付与し、使用開始を効率化することを目的としています。本番稼働用には、ResourceGraphDefinitions が管理する特定のリソースに必要なアクセス許可のみを付与する、より制限の厳しい RBAC ポリシーを作成します。最小特権のアクセス許可の設定に関するガイダンスについては、「the section called “アクセス許可の設定”」および「the section called “EKS 機能に関する考慮事項”」を参照してください。

ステップ 5: カスタムリソースが使用可能であることを確認する

機能がアクティブになったら、kro カスタムリソースがクラスターで使用可能になっていることを確認します。

```
kubectl api-resources | grep kro.run
```

ResourceGraphDefinition リソースタイプがリストされます。

次のステップ

- the section called “kro の概念” - kro の概念とリソース構成を理解する
- the section called “機能の使用” - kro 機能リソースを管理する

kro の概念

kro を使用すると、プラットフォームチームは、複数のリソースを高レベルの抽象化に構成するカスタム Kubernetes API を作成できます。このトピックでは、実用的な例を取り上げ、EKS Capability for kro を使用する際に理解しておく必要がある重要な概念について説明します。

kro の開始方法

kro 機能を作成した後 (「the section called “kro 機能の作成”」を参照)、クラスターで ResourceGraphDefinitions を使用して、カスタム API の作成を開始できます。

シンプルなウェブアプリケーションの抽象化を作成する完全な例を次に示します。

```
apiVersion: kro.run/v1alpha1
kind: ResourceGraphDefinition
metadata:
  name: webapplication
spec:
  schema:
    apiVersion: v1alpha1
    kind: WebApplication
    group: kro.run
    spec:
      name: string | required=true
      image: string | default="nginx:latest"
      replicas: integer | default=3
  resources:
  - id: deployment
    template:
      apiVersion: apps/v1
      kind: Deployment
      metadata:
        name: ${schema.spec.name}
      spec:
        replicas: ${schema.spec.replicas}
        selector:
```

```
          matchLabels:
            app: ${schema.spec.name}
        template:
          metadata:
            labels:
              app: ${schema.spec.name}
          spec:
            containers:
            - name: app
              image: ${schema.spec.image}
              ports:
              - containerPort: 80
  - id: service
    template:
      apiVersion: v1
      kind: Service
      metadata:
        name: ${schema.spec.name}
      spec:
        selector:
          app: ${schema.spec.name}
        ports:
        - protocol: TCP
          port: 80
          targetPort: 80
```

この ResourceGraphDefinition を適用すると、アプリケーションチームは、簡素化された API を使用してウェブアプリケーションを作成できます。

```
apiVersion: kro.run/v1alpha1
kind: WebApplication
metadata:
  name: my-app
spec:
  name: my-app
  replicas: 5
```

kro は、適切な設定でデプロイとサービスを自動的に作成します。image が指定されていないため、スキーマのデフォルト値である nginx:latest が使用されます。

重要な概念

> **Important**
>
> kro は、ランタイムではなく作成時に ResourceGraphDefinitions を検証します。RGD を作成すると、kro は CEL 構文を検証し、式を実際の Kubernetes スキーマと照合して型チェックし、フィールドの存在を検証し、循環依存関係を検出します。つまり、RGD を作成すると、インスタンスがデプロイされる前にエラーがすぐに検出されます。

ResourceGraphDefinition

ResourceGraphDefinition (RGD) は、以下を指定することでカスタム Kubernetes API を定義します。

- スキーマ - SimpleSchema 形式 (フィールド名、型、デフォルト、検証) を使用した API 構造
- リソース - 作成する基盤となる Kubernetes または AWS リソースのテンプレート
- 依存関係 - リソースの相互関係 (フィールド参照から自動的に検出)

RGD を適用すると、kro はクラスターに新しいカスタムリソース定義 (CRD) を登録します。その後、アプリケーションチームはカスタム API のインスタンスを作成でき、kro は基盤となるすべてのリソースの作成と管理を処理します。

詳細については、kro ドキュメントの「ResourceGraphDefinition Overview」を参照してください。

SimpleSchema 形式

SimpleSchema は、OpenAPI の知識を必要とせずに API スキーマを定義する簡単な方法を提供します。

```
schema:
  apiVersion: v1alpha1
  kind: Database
  spec:
    name: string | required=true description="Database name"
    size: string | default="small" enum=small,medium,large
    replicas: integer | default=1 minimum=1 maximum=5
```

SimpleSchema は、`required`、`default`、`minimum/maximum`、`enum`、`pattern` などの制約がある、`string`、`integer`、`boolean`、`number` 型をサポートしています。

詳細については、kro ドキュメントの「SimpleSchema」を参照してください。

CEL 式

kro は、値を動的に参照し、条件付きロジックを追加するために、Common Expression Language (CEL) を使用します。CEL 式は ${ と } で囲まれ、次の 2 つの方法で使用できます。

スタンドアロン式 - フィールド値全体が 1 つの式です。

```
spec:
  replicas: ${schema.spec.replicaCount}  # Expression returns integer
  labels: ${schema.spec.labelMap}        # Expression returns object
```

式の結果はフィールド値全体を置き換え、フィールドで想定されている型と一致する必要があります。

文字列テンプレート - 文字列に埋め込まれた 1 つ以上の式:

```
metadata:
  name: "${schema.spec.prefix}-${schema.spec.name}"  # Multiple expressions
  annotation: "Created by ${schema.spec.owner}"      # Single expression in string
```

文字列テンプレートのすべての式は文字列を返す必要があります。他の型を変換するには、`string()` を使用します (例: `"replicas-${string(schema.spec.count)}"`)。

フィールドリファレンス - `schema.spec` を使用してインスタンスの仕様値にアクセスします。

```
template:
  metadata:
    name: ${schema.spec.name}-deployment
    namespace: ${schema.metadata.namespace}  # Can also reference metadata
  spec:
    replicas: ${schema.spec.replicas}
```

オプションのフィールドアクセス - 存在しない可能性があるフィールドには ? を使用します。

```
# For ConfigMaps or Secrets with unknown structure
value: ${configmap.data.?DATABASE_URL}

# For optional status fields
```

```
ready: ${deployment.status.?readyReplicas > 0}
```

フィールドが存在しない場合、式は失敗せず null を返します。

条件付きリソース - 条件が満たされた場合にのみリソースを含めます。

```
resources:
- id: ingress
  includeWhen:
    - ${schema.spec.enableIngress == true}
  template:
    # ... ingress configuration
```

includeWhen フィールドは、ブール式のリストを受け入れます。リソースを作成するには、すべての条件が true である必要があります。現在、includeWhen は schema.spec フィールドのみを参照できます。

変換 - 三項演算子と関数を使用して値を変換します。

```
template:
  spec:
    resources:
      requests:
        memory: ${schema.spec.size == "small" ? "512Mi" : "2Gi"}

    # String concatenation
    image: ${schema.spec.registry + "/" + schema.spec.imageName}

    # Type conversion
    port: ${string(schema.spec.portNumber)}
```

リソース間の参照 - 他のリソースからの参照値。

```
resources:
- id: bucket
  template:
    apiVersion: s3.services.k8s.aws/v1alpha1
    kind: Bucket
    spec:
      name: ${schema.spec.name}-data
```

```
- id: configmap
  template:
    apiVersion: v1
    kind: ConfigMap
    data:
      BUCKET_NAME: ${bucket.spec.name}
      BUCKET_ARN: ${bucket.status.ackResourceMetadata.arn}
```

CEL 式で別のリソースを参照すると、依存関係が自動的に作成されます。kro は、参照されたリソースが先に作成されることを保証します。

詳細については、kro ドキュメントの「CEL Expressions」を参照してください。

リソースの依存関係

kro は CEL 式から依存関係を自動的に推論します。順序を指定するのではなく、関係を記述します。あるリソースが CEL 式を使用して別のリソースを参照すると、kro は依存関係を作成し、正しい作成順序を決定します。

```
resources:
- id: bucket
  template:
    apiVersion: s3.services.k8s.aws/v1alpha1
    kind: Bucket
    spec:
      name: ${schema.spec.name}-data

- id: notification
  template:
    apiVersion: s3.services.k8s.aws/v1alpha1
    kind: BucketNotification
    spec:
      bucket: ${bucket.spec.name}  # Creates dependency: notification depends on bucket
```

式 `${bucket.spec.name}` は依存関係を作成します。kro は、すべてのリソースとその依存関係から有向非巡回グラフ (DAG) を構築し、作成のためのトポロジ順を計算します。

作成順序: リソースはトポロジ順 (依存関係が先) で作成されます。

並列作成: 依存関係のないリソースは同時に作成されます。

削除順序: リソースは逆トポロジ順 (依存が先) で削除されます。

循環依存関係: 許可されていません。kro は検証時に循環依存関係を含む ResourceGraphDefinitions を拒否します。

計算された作成順序を表示できます。

```
kubectl get resourcegraphdefinition my-rgd -o jsonpath='{.status.topologicalOrder}'
```

詳細については、kro ドキュメントの「Graph inference」を参照してください。

ACK を使用した構成

kro は、EKS Capability for ACK とシームレスに連携し、AWS リソースを Kubernetes と組み合わせて構成します。

```
resources:
# Create {aws} S3 bucket with ACK
- id: bucket
  template:
    apiVersion: s3.services.k8s.aws/v1alpha1
    kind: Bucket
    spec:
      name: ${schema.spec.name}-files

# Inject bucket details into Kubernetes ConfigMap
- id: config
  template:
    apiVersion: v1
    kind: ConfigMap
    data:
      BUCKET_NAME: ${bucket.spec.name}
      BUCKET_ARN: ${bucket.status.ackResourceMetadata.arn}

# Use ConfigMap in application deployment
- id: deployment
  template:
    apiVersion: apps/v1
    kind: Deployment
    spec:
      template:
        spec:
          containers:
          - name: app
```

```
        envFrom:
        - configMapRef:
            name: ${config.metadata.name}
```

このパターンでは、AWS リソースを作成し、その詳細 (ARN、URL、エンドポイント) を抽出してアプリケーション設定に挿入できます。これらはすべて、1 つのユニットとして管理されます。

その他の構成パターンと詳細な例については、「the section called “考慮事項”」を参照してください。

次のステップ

- the section called “考慮事項” - EKS 固有のパターン、RBAC、および ACK と Argo CD との統合について説明します。
- kro ドキュメント - 高度な CEL 式、検証パターン、トラブルシューティングを含む、kro の包括的なドキュメントです。

kro アクセス許可の設定

ACK や Argo CD とは異なり、kro には IAM アクセス許可は必要ありません。kro は Kubernetes クラスター内で完全に動作し、AWS API コールを行いません。標準の Kubernetes RBAC を使用して kro リソースへのアクセスを制御します。

kro でのアクセス許可の仕組み

kro は、異なるスコープを持つ 2 種類の Kubernetes リソースを使用します。

ResourceGraphDefinitions: カスタム API を定義するクラススコープのリソース。通常、組織の標準を設計および維持するプラットフォームチームによって管理されます。

インスタンス: ResourceGraphDefinitions から作成された名前空間スコープのカスタムリソース。適切な RBAC アクセス許可を持つアプリケーションチームが作成できます。

デフォルトでは、kro 機能には、`AmazonEKSKROPolicy` アクセスエントリポリシーを通じて、ResourceGraphDefinitions とそのインスタンスを管理するアクセス許可があります。ただし、kro には、ResourceGraphDefinitions で定義されている基盤となる Kubernetes リソース (デプロイ、サービス、ACK リソースなど) を作成および管理するための追加のアクセス許可が必要です。これらのアクセス許可は、アクセスエントリポリシーまたは Kubernetes RBAC を通じて付与する必要があります。これらのアクセス許可の付与に関する詳細については、「the section called

“eksctl”」、「the section called “ AWS CLI”」、「the section called “コンソール”」を参照してください。

プラットフォームチームのアクセス許可

プラットフォームチームには、ResourceGraphDefinitions を作成および管理するためのアクセス許可が必要です。

プラットフォームチームの ClusterRole の例:

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: kro-platform-admin
rules:
- apiGroups: ["kro.run"]
  resources: ["resourcegraphdefinitions"]
  verbs: ["*"]
```

プラットフォームチームメンバーにバインドする:

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: platform-team-kro-admin
subjects:
- kind: Group
  name: platform-team
  apiGroup: rbac.authorization.k8s.io
roleRef:
  kind: ClusterRole
  name: kro-platform-admin
  apiGroup: rbac.authorization.k8s.io
```

アプリケーションチームのアクセス許可

アプリケーションチームには、名前空間にカスタムリソースのインスタンスを作成するためのアクセス許可が必要です。

アプリケーションチームのロールの例:

```
apiVersion: rbac.authorization.k8s.io/v1
```

```
kind: Role
metadata:
  name: kro-app-developer
  namespace: my-app
rules:
- apiGroups: ["kro.run"]
  resources: ["webapps", "databases"]
  verbs: ["create", "get", "list", "update", "delete", "patch"]
```

アプリケーションチームメンバーにバインドする:

```
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: app-team-kro-developer
  namespace: my-app
subjects:
- kind: Group
  name: app-team
  apiGroup: rbac.authorization.k8s.io
roleRef:
  kind: Role
  name: kro-app-developer
  apiGroup: rbac.authorization.k8s.io
```

> **Note**
>
> ロールの API グループ (この例では `kro.run`) は、ResourceGraphDefinition のスキーマで定義されている `apiVersion` と一致する必要があります。

読み取り専用アクセス

変更許可なしで ResourceGraphDefinitions とインスタンスを表示する読み取り専用アクセスを付与します。

読み取り専用 ClusterRole:

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: kro-viewer
```

```
rules:
- apiGroups: ["kro.run"]
  resources: ["resourcegraphdefinitions"]
  verbs: ["get", "list", "watch"]
```

インスタンスの読み取り専用ロール:

```
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  name: kro-instance-viewer
  namespace: my-app
rules:
- apiGroups: ["kro.run"]
  resources: ["webapps", "databases"]
  verbs: ["get", "list", "watch"]
```

マルチ名前空間アクセス

RoleBindings と ClusterRoles を使用して、アプリケーションチームに複数の名前空間へのアクセスを付与します。

マルチ名前空間アクセス用の ClusterRole:

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: kro-multi-namespace-developer
rules:
- apiGroups: ["kro.run"]
  resources: ["webapps"]
  verbs: ["create", "get", "list", "update", "delete"]
```

特定の名前空間にバインドする:

```
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: app-team-dev-access
  namespace: development
subjects:
- kind: Group
```

```
  name: app-team
  apiGroup: rbac.authorization.k8s.io
roleRef:
  kind: ClusterRole
  name: kro-multi-namespace-developer
  apiGroup: rbac.authorization.k8s.io
---
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: app-team-staging-access
  namespace: staging
subjects:
- kind: Group
  name: app-team
  apiGroup: rbac.authorization.k8s.io
roleRef:
  kind: ClusterRole
  name: kro-multi-namespace-developer
  apiGroup: rbac.authorization.k8s.io
```

ベストプラクティス

最小特権の原則: 各チームの責任に必要な最小限のアクセス許可のみを付与します。

個々のユーザーの代わりにグループを使用する: 管理を容易にするために、ロールは個々のユーザーではなくグループにバインドします。

プラットフォームとアプリケーションの責務を分離する: プラットフォームチームが ResourceGraphDefinitions を管理し、アプリケーションチームがインスタンスを管理します。

名前空間の分離: 名前空間を使用してさまざまなチームや環境を分離し、RBAC が各名前空間へのアクセスを制御します。

監査のための読み取り専用アクセス: 監査目的で、セキュリティおよびコンプライアンスチームに読み取り専用アクセスを提供します。

次のステップ

- the section called “kro の概念” - kro の概念とリソース構成を理解する
- the section called “EKS 機能に関する考慮事項” - 機能のセキュリティのベストプラクティスを見直す

EKS を利用する場合の kro の考慮事項

リソース構成、RBAC パターン、他の EKS 機能との統合を使用するタイミングなど、kro 向けの EKS マネージド機能を使用する際の重要な考慮事項を確認します。

kro を使用する場合

kro は、複雑なリソース管理を簡素化する、再利用可能なインフラストラクチャパターンとカスタム API を作成するように設計されています。

次の場合に kro を使用します。

- アプリケーションチーム向けに、簡素化された API を備えたセルフサービスプラットフォームを作成する場合
- チーム間でインフラストラクチャパターン (データベース + バックアップ + モニタリング) を標準化する場合
- リソースの依存関係を管理し、リソース間で値を渡す場合
- 実装の複雑さを隠すカスタム抽象化を構築する場合
- 複数の ACK リソースを高レベルの構成要素に構成する場合
- 複雑なインフラストラクチャスタックの GitOps ワークフローを有効にする場合

次の場合は kro を使用しないでください。

- シンプルでスタンドアロンなリソースを管理する場合 (ACK または Kubernetes リソースを直接使用します)
- 動的なランタイムロジックが必要な場合 (kro は命令型ではなく、宣言型です)
- リソースに依存関係や共有設定がない場合

kro は、チームが複雑なインフラストラクチャを正しくデプロイしやすくする、意見が組み込まれた再利用可能なパターン (「paved paths」) の作成に優れています。

RBAC パターン

kro を使用すると、ResourceGraphDefinitions を作成するプラットフォームチームと、インスタンスを作成するアプリケーションチームとの間で責務を分離できます。

プラットフォームチームの責任

プラットフォームチームは、カスタム API を定義する ResourceGraphDefinitions (RGDs) を作成および維持します。

必要となる権限:

- ResourceGraphDefinitions の作成、更新、削除
- 基盤となるリソースタイプ (デプロイ、サービス、ACK リソース) の管理
- RGDs が使用されるすべての名前空間へのアクセス

プラットフォームチーム向け ClusterRole の例:

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: platform-kro-admin
rules:
- apiGroups: ["kro.run"]
  resources: ["resourcegraphdefinitions"]
  verbs: ["*"]
- apiGroups: ["*"]
  resources: ["*"]
  verbs: ["get", "list", "watch"]
```

RBAC 設定の詳細については、「the section called “アクセス許可の設定”」を参照してください。

アプリケーションチームの責任

アプリケーションチームは、基盤となる複雑さを理解することなく、RGDs で定義されたカスタムリソースのインスタンスを作成します。

必要となる権限:

- カスタムリソースのインスタンスの作成、更新、削除
- 自身の名前空間への読み取りアクセス
- 基盤となるリソースまたは RGDs へのアクセス権は不要

利点:

- チームはシンプルで高レベルの API を使用
- プラットフォームチームが実装の詳細を制御
- 設定ミスのリスクを低減
- 新しいチームメンバーのオンボーディングを高速化

他の EKS 機能との統合

ACK リソースの構成

kro は、ACK と組み合わせることで、インフラストラクチャパターンの作成において特に高い効果を発揮します。

一般的なパターン:

- ストレージを含むアプリケーション: S3 バケット + SQS キュー + 通知設定
- データベーススタック: RDS インスタンス + パラメータグループ + セキュリティグループ + Secrets Manager シークレット
- ネットワーク: VPC + サブネット + ルートテーブル + セキュリティグループ + NAT ゲートウェイ
- ストレージを含むコンピューティング: EC2 インスタンス + EBS ボリューム + IAM インスタンスプロファイル

kro は、依存関係の順序付けを処理し、リソース間で (ARN や接続文字列などの) 値を渡し、ライフサイクル全体を 1 つのユニットとして管理します。

ACK リソースを作成する例については、「the section called “ACK の概念”」を参照してください。

GitOps と Argo CD

EKS Capability for Argo CD を使用して、Git リポジトリから RGDs とインスタンスの両方をデプロイします。

リポジトリの整理:

- プラットフォームリポジトリ: プラットフォームチームが管理する ResourceGraphDefinitions を格納
- アプリケーションリポジトリ: アプリケーションチームが管理するカスタムリソースのインスタンスを格納
- 共有リポジトリ: 小規模な組織向けに、RGDs とインスタンスの両方を格納

考慮事項:

- インスタンスより先に RGDs をデプロイする (Argo CD の同期ウェーブが役立つ)
- プラットフォームチームとアプリケーションチームに別々の Argo CD プロジェクトを使用する
- プラットフォームチームが RGD リポジトリへのアクセスを制御する
- アプリケーションチームには RGD 定義への読み取り専用アクセスを付与する

Argo CD の詳細については、「the section called “Argo CD の使用”」を参照してください。

ResourceGraphDefinitions の整理

目的、複雑さ、所有権別に RGDs を整理します。

目的別:

- インフラストラクチャ: データベーススタック、ネットワーク、ストレージ
- アプリケーション: ウェブアプリケーション、API、バッチジョブ
- プラットフォーム: 共有サービス、モニタリング、ログ記録

複雑さ別:

- シンプル: 依存関係が最小限の 2～3 個のリソース
- 中程度: 一部に依存関係がある 5～10 個のリソース
- 複雑: 複雑な依存関係を持つ 10 個以上のリソース

命名規則:

- わかりやすい名前を使用する: `webapp-with-database`、`s3-notification-queue`
- 重大な変更には、名前にバージョンを含める: `webapp-v2`
- 関連する RGDs には、一貫したプレフィックスを使用する: `platform-` 、`app-`

名前空間戦略:

- RGDs はクラスタースコープである (名前空間なし)
- インスタンスは名前空間に属する

- RGDs で名前空間セレクタを使用して、インスタンスを作成できる場所を制御する

バージョニングと更新

RGD の進化とインスタンスの移行を計画します。

RGD の更新:

- 重要な変更なし: RGD をそのまま更新する (オプションフィールドを追加、includeWhen を使用した新しいリソースを追加)
- 重大な変更: 異なる名前の新しい RGD を作成する (webapp-v2)
- 非推奨化: 古い RGDs に注釈を付け、移行タイムラインを周知する

インスタンスの移行:

- 更新された RGD を使用して新しいインスタンスを作成する
- 新しいインスタンスが正しく動作することを検証する
- 古いインスタンスを削除する
- kro は、基盤となるリソースの更新を自動的に処理します

ベストプラクティス:

- 最初に本番稼働用環境以外で RGD の変更をテストする
- 主な変更には、RGD 名にセマンティックバージョニングを使用する
- 重大な変更と移行パスを文書化する
- アプリケーションチームに移行例を提供する

検証とテスト

本番環境にデプロイする前に RGDs を検証します。

検証戦略:

- スキーマ検証: kro は RGD の構造を自動的に検証する
- ドライランインスタンス: 開発用名前空間でテストインスタンスを作成する
- 統合テスト: 構成されたリソースが正しく連携することを確認する

- ポリシーの適用: アドミッションコントローラーを使用して組織標準を適用する

テストする一般的な問題:

- リソースの依存関係と順序付け
- リソース間で渡す値 (CEL 式)
- 条件付きリソースの包含 (includeWhen)
- 基盤となるリソースからのステータス伝播
- インスタンス作成の RBAC アクセス許可

アップストリームドキュメント

kro の使用の詳細については、以下を参照してください。

- kro の開始方法 - ResourceGraphDefinitions の作成
- CEL 式 - CEL 式の記述
- kro ガイド - 高度な構成パターン
- トラブルシューティング - トラブルシューティングとデバッグ

次のステップ

- the section called “アクセス許可の設定” - プラットフォームチームとアプリケーションチームに RBAC を設定する
- the section called “kro の概念” - kro の概念とリソースライフサイクルを理解する
- the section called “トラブルシューティング” - kro の問題をトラブルシューティングする
- the section called “ACK の概念” - 構成に使用する ACK リソースについて説明する
- the section called “Argo CD の使用” - GitOps を使用して RGDs とインスタンスをデプロイする

kro 機能に関する問題をトラブルシューティングする

> **Note**
>
> EKS の機能は完全に管理され、クラスターの外部で実行されます。コントローラーの名前空間に直接アクセスすることはできません。コントローラーの動作の可視化のためにコント

ローラーのログ配信を設定できます。「the section called “機能コントローラーログ”」を参照してください。トラブルシューティングでは、機能のヘルス、RBAC の設定、リソースのステータスに焦点を当てています。

機能は ACTIVE だが、ResourceGraphDefinitions が機能していない

kro 機能のステータスが `ACTIVE` なのに、ResourceGraphDefinitions が基盤となるリソースを作成していない場合は、機能ヘルス、RBAC アクセス許可、リソースステータスを確認してください。

機能のヘルスを確認する:

機能のヘルスとステータスの問題は、EKS コンソールまたは AWS CLI を使用して表示できます。

コンソール:

1. https://console.aws.amazon.com/eks/home#/clusters で Amazon EKS コンソールを開きます。
2. クラスター名を選択します。
3. [オブザーバビリティ] タブを選択します。
4. [クラスターを監視する] を選択します。
5. [機能] タブを選択すると、すべての機能のヘルスとステータスが表示されます。

AWSCLI:

```
# View capability status and health
aws eks describe-capability \
  --region region-code \
  --cluster-name my-cluster \
  --capability-name my-kro

# Look for issues in the health section
```

一般的な原因:

- RBAC アクセス許可がない: kro には、基盤となる Kubernetes リソースを作成するアクセス許可がありません
- 無効な CEL 式: ResourceGraphDefinition に構文エラーがあります。
- リソースの依存関係: 依存するリソースの準備ができていません。

- スキーマ検証: インスタンスが RGD スキーマの要件と一致していません。

RBAC アクセス許可の確認:

```
# Check if capability has cluster admin policy
kubectl get accessentry -A | grep kro
```

機能に必要なアクセス許可がない場合は、AmazonEKSClusterAdminPolicy を kro 機能のアクセスエントリに関連付けるか、本番稼働用のより制限の厳しい RBAC ポリシーを作成します。詳細については、「the section called “アクセス許可の設定”」を参照してください。

ResourceGraphDefinition のステータスの確認:

```
# List all RGDs
kubectl get resourcegraphdefinition

# Describe specific RGD
kubectl describe resourcegraphdefinition my-rgd

# Check for validation errors
kubectl get resourcegraphdefinition my-rgd -o jsonpath='{.status.conditions}'
```

ResourceGraphDefinitions には 3 つの主要なステータス条件があります。

- ResourceGraphAccepted - RGD が検証 (CEL 構文、型チェック、フィールドの存在) に合格したかどうか
- KindReady - カスタム API の CRD が生成され、登録されたかどうか
- ControllerReady - kro がカスタム API のインスタンスをアクティブに監視しているかどうか

ResourceGraphAccepted が False の場合は、不明なフィールド、型の不一致、循環依存関係などの検証エラーがないか、条件メッセージを確認します。

インスタンスは作成されているが、基盤となるリソースが表示されない

カスタムリソースインスタンスが存在するにもかかわらず、基盤となる Kubernetes リソース (デプロイ、サービス、ConfigMaps) が作成されていない場合は、kro にアクセス許可があることを確認し、構成エラーがないかを確認します。

インスタンスのステータスの確認:

```
# Describe the instance (replace with your custom resource kind and name)
kubectl describe custom-kind my-instance

# View instance events
kubectl get events --field-selector involvedObject.name=my-instance

# Check instance status conditions
kubectl get custom-kind my-instance -o jsonpath='{.status.conditions}'

# Check instance state
kubectl get custom-kind my-instance -o jsonpath='{.status.state}'
```

インスタンスには、高レベルのステータスを示す次の state フィールドがあります。

- ACTIVE - インスタンスは正常に実行されています。
- IN_PROGRESS - インスタンスは処理中または調整中です。
- FAILED - インスタンスの調整に失敗しました。
- DELETING - インスタンスを削除中です。
- ERROR - 処理中にエラーが発生しました。

インスタンスには、次の 4 つのステータス条件もあります。

- InstanceManaged - ファイナライザーとラベルが適切に設定されています。
- GraphResolved - ランタイムグラフが作成され、リソースが解決されています。
- ResourcesReady - すべてのリソースが作成され、準備完了です。
- Ready - インスタンス全体のヘルス (すべてのサブ条件が Trueの場合にのみ True になります)。

インスタンスのヘルスを判断するには、Ready 条件に注目します。Ready が False の場合は、サブ条件を確認して、どのフェーズで失敗したかを特定します。

RBAC アクセス許可の確認:

kro 機能には、ResourceGraphDefinitions で定義されている基盤となる Kubernetes リソースを作成するためのアクセス許可が必要です。

```
# Check if the capability has the AmazonEKSClusterAdminPolicy
kubectl get accessentry -A | grep kro
```

アクセス許可がない場合は、AmazonEKSClusterAdminPolicy を kro 機能のアクセスエントリに関連付けるか、本番稼働用のより制限の厳しい RBAC ポリシーを作成します。詳細については、「the section called “アクセス許可の設定”」を参照してください。

CEL 式のエラー

CEL 式のエラーは、インスタンスの作成時ではなく、ResourceGraphDefinition の作成時に検出されます。kro は、RGD の作成時に、すべての CEL 構文を検証し、Kubernetes スキーマに対して式の型チェックを行い、フィールドの存在を検証します。

一般的な CEL 検証エラー:

- 未定義のフィールド参照: スキーマまたはリソースに存在しないフィールドを参照しています。
- 型の不一致: 式が期待される型と異なる型 (例: 整数が期待される箇所で文字列) を返します。
- 無効な構文: CEL 式に括弧、引用符、または演算子がありません。
- 不明なリソースタイプ: クラスターに存在しない CRD を参照しています。

RGD の検証ステータスの確認:

```
# Check if RGD was accepted
kubectl get resourcegraphdefinition my-rgd -o jsonpath='{.status.conditions[?
(@.type=="ResourceGraphAccepted")]}'

# View detailed validation errors
kubectl describe resourcegraphdefinition my-rgd
```

ResourceGraphAccepted が False の場合、条件メッセージには検証エラーが含まれます。

有効な CEL 式の例:

```
# Reference schema field
${schema.spec.appName}

# Conditional expression
${schema.spec.replicas > 1}

# String template (expressions must return strings)
name: "${schema.spec.appName}-service"
```

```
# Standalone expression (can be any type)
replicas: ${schema.spec.replicaCount}

# Resource reference
${deployment.status.availableReplicas}

# Optional field access (returns null if field doesn't exist)
${configmap.data.?DATABASE_URL}
```

リソースの依存関係が解決されない

kro は CEL 式から依存関係を自動的に推測し、正しい順序でリソースを作成します。リソースが想定どおりに作成されていない場合は、依存関係の順序とリソースの準備状況を確認してください。

計算された作成順序の表示:

```
# See the order kro will create resources
kubectl get resourcegraphdefinition my-rgd -o jsonpath='{.status.topologicalOrder}'
```

これは、リソース間の CEL 式の参照に基づいて計算された順序を示しています。

リソースの準備状況の確認:

```
# View instance status to see which resources are ready
kubectl get custom-kind my-instance -o jsonpath='{.status}'

# Check specific resource status
kubectl get deployment my-deployment -o jsonpath='{.status.conditions}'
```

readyWhen 条件の確認 (使用している場合):

`readyWhen` フィールドはオプションです。指定しない場合、リソースは作成直後に準備完了とみなされます。`readyWhen` 条件を定義している場合は、リソースの準備状況が正しくチェックされていることを確認します。

```
resources:
  - id: deployment
    readyWhen:
      - ${deployment.status.availableReplicas == deployment.spec.replicas}
```

リソースイベントの確認:

```
# View events for the underlying resources
kubectl get events -n namespace --sort-by='.lastTimestamp'
```

スキーマ検証の失敗

スキーマ検証エラーが原因でインスタンスの作成に失敗した場合は、インスタンスが RGD スキーマの要件と一致していることを確認します。

検証エラーの確認:

```
# Attempt to create instance and view error
kubectl apply -f instance.yaml

# View existing instance validation status
kubectl describe custom-kind my-instance | grep -A 5 "Validation"
```

一般的な検証の問題:

- 必須フィールドがない: インスタンスがすべての必須スキーマフィールドを提供していません。
- 型の不一致: 整数が期待される箇所で文字列が指定されています。
- 無効な列挙値: 許可されているリストに含まれていない値が使用されています。
- パターンの不一致: 文字列が正規表現パターンと一致していません。

RGD スキーマの確認:

```
# View the schema definition
kubectl get resourcegraphdefinition my-rgd -o jsonpath='{.spec.schema}'
```

インスタンスで、正しい型の必須フィールドがすべて指定されていることを確認します。

次のステップ

- the section called “考慮事項” - kro の考慮事項とベストプラクティス
- the section called “アクセス許可の設定” - プラットフォームチームとアプリケーションチームに RBAC を設定する

- the section called “kro の概念” - kro の概念とリソースライフサイクルを理解する
- the section called “機能をトラブルシューティングする” - 一般的な機能をトラブルシューティングする際のガイダンス

EKS Capability for kro とセルフマネージド kro の比較

EKS Capability for kro は、セルフマネージド kro と同じ機能を提供しますが、運用上の大きな利点があります。EKS 機能とセルフマネージドソリューションの全般的な比較については、「the section called “考慮事項”」を参照してください。

EKS Capability for kro は、同じアップストリームの kro コントローラーを使用し、アップストリームの kro と完全に互換性があります。ResourceGraphDefinitions、CEL 式、およびリソース構成は同じように機能します。完全な kro のドキュメントと例については、kro ドキュメントを参照してください。

移行パス

セルフマネージド kro からマネージド機能にダウンタイムなしで移行できます。

> **Important**
>
> 移行する前に、セルフマネージド kro コントローラーが EKS Capability for kro と同じバージョンで実行されていることを確認します。EKS コンソールまたは `aws eks describe-capability` を使用して機能バージョンを確認し、そのバージョンに一致するようにセルフマネージドのインストールをアップグレードします。これにより、移行中の互換性の問題を防ぐことができます。

1. リーダー選出リースに `kube-system` を使用するようにセルフマネージド kro コントローラーを更新します。

   ```
   helm upgrade --install kro \
     oci://ghcr.io/awslabs/kro/kro-chart \
     --namespace kro \
     --set leaderElection.namespace=kube-system
   ```

   これにより、コントローラーのリースが `kube-system` に移動するため、マネージド機能がコントローラーを調整できるようになります。

2. クラスターに kro 機能を作成します (「the section called “kro 機能の作成”」を参照)。
3. マネージド機能が既存の ResourceGraphDefinitions とインスタンスを認識して、調整を引き継ぎます。
4. セルフマネージド kro のデプロイを段階的にスケールダウンするか、削除します。

```
helm uninstall kro --namespace kro
```

このアプローチにより、移行中、両方のコントローラーを安全に共存させることができます。マネージド機能がこれまでセルフマネージド kro で管理されていた ResourceGraphDefinitions とインスタンスを自動的に採用するので、引き続き競合なく調整を図ることができます。

次のステップ

- the section called “kro 機能の作成” - kro の機能リソースを作成する
- the section called “kro の概念” - kro の概念とリソース構成を理解する

EKS 機能をトラブルシューティングする

> **Note**
>
> EKS の機能は完全に管理され、クラスターの外部で実行されます。コントローラーの名前空間に直接アクセスすることはできません。トラブルシューティングでは、機能のヘルス、リソースのステータス、設定、コントローラーログに焦点を当てています。コントローラーの動作の可視化のためにコントローラーのログ配信を設定できます。「the section called “機能コントローラーログ”」を参照してください。

一般的なトラブルシューティングのアプローチ

EKS 機能をトラブルシューティングするときは、次の一般的なアプローチに従ってください。

1. 機能のヘルスを確認する: `aws eks describe-capability` を使用して、機能のステータスとヘルスに関する問題を表示します。
2. リソースのステータスを確認する: 作成した Kubernetes リソース (CRD) のステータス条件とイベントを確認します。

3. コントローラーログを確認する: ログ配信が設定されている場合、コントローラーログをクエリしてエラーと調整の詳細を確認します。
4. IAM アクセス許可を確認する: 機能ロールに必要なアクセス許可があることを確認します。
5. 設定を確認する: 機能固有の設定が適切であることを確認します。

トラブルシューティングにコントローラーログを使用する

コントローラーログ配信を設定している場合 (「the section called “機能コントローラーログ”」を参照)、ログをクエリして、調整エラー、リソースの競合、設定の問題を特定できます。

すべてのコントローラーのエラーをクエリする

```
fields @timestamp, controller, message, error
| filter level = "error"
| sort @timestamp desc
| limit 50
```

特定の ACK サービスコントローラーのログをフィルタリングする

`controllerGroup` フィールドを使用して、特定の ACK サービスコントローラーからログを分離します。

```
fields @timestamp, message, error
| filter controllerGroup = "s3.services.k8s.aws"
| filter level = "error"
| sort @timestamp desc
```

リソースの種類 (EC2 コントローラーからの `SecurityGroup` ログのみなど) でさらにフィルタリングするには、次のようにします。

```
fields @timestamp, message, error
| filter controllerGroup = "ec2.services.k8s.aws"
| filter controllerKind = "SecurityGroup"
| sort @timestamp desc
| limit 100
```

特定の Argo CD アプリケーションのログをフィルタリングする

`application` フィールドを使用して、特定の Argo CD アプリケーションのログを分離します。

```
fields @timestamp, message, error
| filter application = "my-application"
| sort @timestamp desc
| limit 100
```

特定のリソースの調整を追跡する

reconcileID フィールドを使用して、1 つの調整サイクルを追跡します。

```
fields @timestamp, level, message, error
| filter reconcileID = "your-reconcile-id"
| sort @timestamp asc
```

問題を示す一般的なログパターン

- 繰り返される調整エラー — コントローラーがリソースの目的の状態に到達できません。IAM アクセス許可の失敗や無効なリソース設定などの詳細について、error フィールドを確認します。
- AWS API エラーでの「調整エラー」 — 機能ロールに特定の AWS サービスオペレーションのアクセス許可がない可能性があります。エラーメッセージを確認し、それに応じて IAM ポリシーを更新します。
- リソースのログエントリがない — コントローラーによる調整が期待されるリソースのログが表示されない場合は、機能が ACTIVE であり、機能がアクセスできる名前空間にリソースが存在することを確認します。

機能のヘルスを確認する

すべての EKS 機能が、EKS コンソールと describe-capability API を介してヘルス情報を提供します。

コンソール:

1. https://console.aws.amazon.com/eks/home#/clusters で Amazon EKS コンソールを開きます。
2. クラスター名を選択します。
3. [オブザーバビリティ] タブを選択します。
4. [クラスターを監視する] を選択します。
5. [機能] タブを選択すると、すべての機能のヘルスとステータスが表示されます。

[機能] タブには以下が表示されます。

- 機能の名前とタイプ
- 現在のステータス
- ヘルスの問題とその説明

AWSCLI:

```
aws eks describe-capability \
  --region region-code \
  --cluster-name my-cluster \
  --capability-name my-capability-name
```

レスポンスは以下のとおりです。

- status: 機能の現在の状態
  (CREATING、ACTIVE、UPDATING、DELETING、CREATE_FAILED、UPDATE_FAILED)
- health: 機能が検出した問題などのヘルス情報

すべてのクラスターにわたって機能のヘルスを確認するには、「the section called “Amazon EKS ダッシュボード”」を参照してください。

機能でよく見られるステータス

CREATING: 機能のセットアップ中です。

ACTIVE: 機能の実行中ですぐに使用できるようになります。リソースが想定どおりに動作しない場合は、リソースのステータスと IAM アクセス許可を確認してください。

UPDATING: 設定変更の適用中です。ステータスが ACTIVE に戻るまで待ちます。

CREATE_FAILED または UPDATE_FAILED: セットアップまたは更新でエラーが発生しました。詳細については、ヘルスに関するセクションを確認してください。一般的な原因:

- IAM ロールの信頼ポリシーが正しくないか、見つかりません。
- IAM ロールが存在しないか、アクセスできません
- クラスターアクセスに問題があります。

- 設定パラメータが無効です。

Kubernetes リソースのステータスを確認する

EKS 機能は、クラスターで Kubernetes カスタムリソース定義 (CRD) を作成および管理します。トラブルシューティングするときは、作成したリソースのステータスを確認してください。

```
# List resources of a specific type
kubectl get resource-kind -A

# Describe a specific resource to see conditions and events
kubectl describe resource-kind resource-name -n namespace

# View resource status conditions
kubectl get resource-kind resource-name -n namespace -o jsonpath='{.status.conditions}'

# View events related to the resource
kubectl get events --field-selector involvedObject.name=resource-name -n namespace
```

リソースのステータス条件では、以下の情報を確認できます。

- リソースの準備ができているかどうか
- 何かエラーが発生しているか
- 現在の調整状態

IAM アクセス許可とクラスターアクセスを確認する

機能で発生する問題の多くは、IAM アクセス許可に問題があるか、クラスターアクセス設定が欠落していることに起因しています。機能ロールのアクセス許可とクラスターアクセスのエントリの両方を検証します。

IAM ロール許可をチェックする

必要なアクセス許可が機能ロールに付与されていることを確認します。

```
# List attached managed policies
aws iam list-attached-role-policies --role-name my-capability-role
```

```
# List inline policies
aws iam list-role-policies --role-name my-capability-role

# Get specific policy details
aws iam get-role-policy --role-name my-capability-role --policy-name policy-name

# View the role's trust policy
aws iam get-role --role-name my-capability-role --query 'Role.AssumeRolePolicyDocument'
```

この信頼ポリシーは、capabilities.eks.amazonaws.com サービスプリンシパルを許可する必要があります。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "capabilities.eks.amazonaws.com"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
```

EKS アクセスエントリとアクセスポリシーを確認する

どの機能でも、動作しているクラスターで適切な EKS アクセスエントリとアクセスポリシーが必要です。

アクセスエントリが存在することを確認する:

```
aws eks list-access-entries \
  --cluster-name my-cluster \
  --region region-code
```

リスト内で機能ロール ARN を探します。見つからない場合、機能はクラスターにアクセスできません。

エントリにアタッチされているアクセスポリシーを確認する:

```
aws eks list-associated-access-policies \
  --cluster-name my-cluster \
  --principal-arn arn:aws:iam::111122223333:role/my-capability-role \
  --region region-code
```

どの機能にも、適切なアクセスポリシーが必要です。

- ACK: Kubernetes リソースを作成および管理するためのアクセス許可が必要です。
- kro: Kubernetes リソースを作成および管理するためのアクセス許可が必要です。
- Argo CD: Application を作成および管理するためのアクセス許可が必要です。また、マルチクラスターのデプロイには、リモートターゲットクラスターにアクセスエントリが必要です。

Argo CD によるマルチクラスターのデプロイの場合:

リモートクラスターにデプロイする場合は、ターゲットクラスターごとに機能ロールのアクセスエントリがあることを確認します。

```
# Check Access Entry on target cluster
aws eks describe-access-entry \
  --cluster-name target-cluster \
  --principal-arn arn:aws:iam::111122223333:role/argocd-capability-role \
  --region region-code
```

ターゲットクラスターにアクセスエントリがない場合、Argo CD はそのクラスターにアプリケーションをデプロイできません。設定の詳細については、「the section called “クラスターを登録する”」を参照してください。

機能に固有のトラブルシューティング

各機能タイプに固有の詳しいトラブルシューティングガイダンスについては、以下を参照してください。

- the section called “トラブルシューティング” - ACK リソースの作成、IAM アクセス許可、クロスアカウントアクセスをトラブルシューティングします。
- the section called “トラブルシューティング” - アプリケーション同期、リポジトリ認証、マルチクラスターデプロイをトラブルシューティングします。

- the section called “トラブルシューティング” - ResourceGraphDefinitions、CEL 式、RBAC アクセス許可をトラブルシューティングします。

すべての機能に共通する問題

CREATING 状態で機能が停止する

機能が想定よりも長く CREATING 状態のままである場合:

1. コンソール ([オブザーバビリティ] > [クラスターをモニタリング] > [機能] タブ) または AWS CLI を使用して、機能のヘルスをチェックして特定の問題がないか確認します。

```
aws eks describe-capability \
  --region region-code \
  --cluster-name my-cluster \
  --capability-name my-capability-name \
  --query 'capability.health'
```

2. IAM ロールが存在し、信頼ポリシーが適切であることを確認します。
3. クラスターがアクセス可能で正常であることを確認します。
4. 機能のセットアップを阻害するようなクラスターレベルの問題がないか確認します。

リソースが作成または更新されない

機能が ACTIVE であるのにリソースが作成または更新されない場合:

1. リソースのステータスをチェックしてエラー条件を確認します。
2. 特定の AWS サービス (ACK) またはリポジトリ (Argo CD) に対する IAM アクセス許可を検証します。
3. 基礎となるリソース (kro) を作成するための RBAC アクセス許可を確認します。
4. リソース仕様をチェックして検証エラーがないか確認します。

機能のヘルスに問題がある

`describe-capability` がヘルスの問題を示している場合:

1. 問題の説明を注意深く読みます。具体的な問題を示していることがよくあります。

2. 根本原因 (IAM アクセス許可や設定エラーなど) に対処します。
3. 問題が解決すると、機能は自動的に復旧します。

次のステップ

- the section called “機能の使用” - 機能リソースを管理する
- the section called “トラブルシューティング” - ACK に固有のトラブルシューティングを行う
- the section called “トラブルシューティング” - Argo CD に固有のトラブルシューティングを行う
- the section called “トラブルシューティング” - kro に固有のトラブルシューティングを行う
- the section called “EKS 機能に関する考慮事項” - 機能のセキュリティ面のベストプラクティス

クラスターリソースを整理およびモニタリングする

この章では、クラスターの管理に役立つ以下のトピックについて説明します。また、AWS マネジメントコンソール で Kubernetes リソースに関する情報を表示することもできます。

- Kubernetes ダッシュボードは、Kubernetes クラスター用の汎用でウェブベースの UI です。これにより、ユーザーはクラスター自体を管理するだけでなく、クラスターで実行されているアプリケーションの管理およびトラブルシューティングを行うことができます。詳細については、「Kubernetes ダッシュボード」GitHub リポジトリを参照してください。
- the section called “メトリクスサーバー” – Kubernetes メトリクスサーバーは、クラスター内のリソース使用状況データを集約します。デフォルトでは、クラスターにはデプロイされませんが、Kubernetes ダッシュボードや the section called “Horizontal Pod Autoscaler” などの Kubernetes のアドオンで使用されます。このトピックでは、メトリクスサーバーのインストール方法を説明します。
- the section called “Helm を使用してアプリケーションをデプロイする” – Kubernetes 用の Helm パッケージマネージャーを使用すると、Kubernetes クラスターにアプリケーションをインストールし、管理できます。このトピックは、Helm バイナリをインストールして実行する際に役立ちます。これにより、ローカルコンピューターで Helm CLI を使用してチャートをインストールおよび管理することができます。
- the section called “リソースのタグ付け” – Amazon EKS リソースを管理しやすくするために、タグ形式で各リソースに独自のメタデータを割り当てることができます。ここでは、タグとその作成方法について説明します。
- the section called “Service Quotas” – AWS アカウントには、AWS のサービスごとにデフォルトのクォータ (以前は制限と呼ばれていました) があります。Amazon EKS のクォータと拡大方法について説明します。

Amazon EKS クラスターのコストをモニタリングして最適化する

コストのモニタリングは、Amazon EKS で Kubernetes クラスターを管理する上で不可欠な要素です。クラスターのコストを可視化することで、リソースの使用率を最適化し、予算を設定し、デプロイに関するデータ主導の意思決定を行うことができます。Amazon EKS には 2 つのコストモニタリングソリューションがあり、それぞれに独自の利点があり、コストを効果的に追跡して割り当てるのに役立ちます。

AWS 請求を活用した Amazon EKS のコスト配分データの分割 — このネイティブ機能は、AWS 請求コンソールにシームレスに統合されるため、他の AWS のサービスと同様の使い慣れたインターフェイスとワークフローを使って、コストの分析および割り当てを行うことができます。コスト配分の分割を使用すると、Kubernetes コストに関するインサイトを他の AWS 支出と並行して直接得ることができるため、AWS 環境全体でコストを簡単に、総合的に最適化できるようになります。コストカテゴリーやコスト異常検出など、既存の AWS 請求の機能を活用して、コスト管理機能をさらに強化することもできます。詳細については、「AWS ユーザーガイド」の「Understanding split cost allocation data」を参照してください。

Kubecost — Amazon EKS は、Kubernetes のコストモニタリングツールである Kubecost をサポートしています。Kubecost には、リソースごとの詳細なコストの内訳、コスト最適化のために推奨される方法、すぐに使用できるダッシュボードやレポートなど、Kubernetes ネイティブのコストモニタリングに特化した機能が多数用意されています。また、Kubecost を使用して AWS コストと使用状況レポートと統合すると、正確な価格データを取得でき、Amazon EKSのコストを正確に把握することができます。Kubecost をインストールする方法について説明します。無料の Kubecost サブスクリプションの取得については、Kubecost AWS Marketplace ページを参照してください。

分割コスト割り当てを使用した AWS 請求で Pod ごとにコストを表示する

Amazon EKS の AWS コスト配分データの分割を使用したコストモニタリング

Amazon EKS の AWS コスト配分データの分割を使用すると、Amazon EKS クラスターのコストをきめ細かく把握することができます。これにより、Kubernetes アプリケーションのコストと使用状況の分析、最適化、チャージバックが可能になります。アプリケーションのコストは、Kubernetes アプリケーションが使用した Amazon EC2 CPU とメモリリソースに基づいて個々の部門およびチームに割り当てられます。Amazon EKS のコスト配分データの分割を使用することで、Pod あたりのコストを可視化し、名前空間、クラスター、その他の Kubernetes プリミティブを使用している Pod ごとのコストデータを集計することができます。以下は、Amazon EKS コスト配分データの分析に使用できる Kubernetes プリミティブの一例です。

- クラスター名
- デプロイ
- 名前空間
- ノード
- ワークロード名
- ワークロードタイプ

ユーザー定義のコスト配分タグもサポートされています。コスト配分データの分割の詳細については、「AWS Billing ユーザーガイド」の「Understanding split cost allocation data」を参照してください。

コストと使用状況レポートの設定

コスト管理のコンソール、AWS コマンドラインインターフェイス、AWS SDK のいずれかで、[EKS のコスト配分データの分割] をオンにします。

コスト配分データの分割には以下を使用します。

1. コスト配分データの分割にオプトインします。詳細については、AWS コストと使用状況レポートユーザーガイドの「分割コスト配分データの有効化」を参照してください。
2. 新しいレポートまたは既存のレポートにデータを含めます。
3. レポートを表示します。請求とコスト管理コンソールを使用するか、Amazon Simple Storage Service でレポートファイルを表示できます。

Kubecost のインストール

Amazon EKS がサポートする Kubecost を使用して、Pod、ノード、名前空間、ラベルなどの Kubernetes リソースごとにコストを分類して監視することができます。このトピックでは、Kubecost のインストールと Kubecost ダッシュボードへのアクセスについて説明します。

Amazon EKS は、クラスターコストの可視化のため、AWS に最適化された Kubecost のバンドルを提供します。既存の AWS サポート契約を使用してサポートを受けることができます。使用できる Kubecost のバージョンの詳細については、「the section called “Kubecost の詳細”」を参照してください。

> **Note**
>
> Kubecost v3 では、パフォーマンスの劇的な向上や自動化機能の強化など、アーキテクチャの大幅な改善が導入されています。Kubecost v3 の詳細はこちらです。
> Kubecost v2 では、いくつかの主要な新機能が導入されています。Kubecost v2 の詳細はこちらです。

Kubecost の詳細については、Kubecost ドキュメントと「よくある質問」を参照してください。

Amazon EKS 最適化 Kubecost バンドルをインストールする

Amazon EKS 最適化 Kubecost バンドルをインストールするには、次のいずれかの手順を使用します。

- 開始する前に、「Kubecost – アーキテクチャの概要」を確認して、Kubecost が Amazon EKS でどのように機能するかを理解することをお勧めします。
- Amazon EKS を初めて使用する場合は、Amazon EKS 最適化 Kubecost バンドルのインストールが簡素化されるため、Amazon EKS アドオンを使用してインストールすることをお勧めします。詳細については、「Amazon EKS アドオンを使用して、Amazon EKS クラスターに Kubecost をデプロイする」を参照してください。
- インストールをカスタマイズするには、Amazon EKS 最適化 Kubecost バンドルを Helm で設定できます。詳細については、Kubecost ドキュメントの「Deploying Kubecost on an Amazon EKS cluster using Helm」を参照してください。

> **Important**
>
> Kubecost v3 では、Helm チャートの場所が `public.ecr.aws/kubecost/kubecost` に変更されました。v2 からアップグレードする場合は、それに応じて Helm リポジトリのリファレンスを更新します。

> **Note**
>
> Kubecost v3 を使用したマルチクラスターデプロイでは、メトリクスストレージに S3 互換オブジェクトストレージ (EKS のお客様の場合は AWS S3) が必要です。これにより、v2 で使用される Prometheus 互換ストレージが置き換えられます。詳細については、Kubecost ドキュメントの「マルチクラスターのインストール」を参照してください。

Kubecost ダッシュボードにアクセスする

Amazon EKS 最適化 Kubecost バンドルのセットアップが完了したら、Kubecost ダッシュボードにアクセスできます。詳細については、「the section called “Kubecost ダッシュボードにアクセスする”」を参照してください。

Kubecost ダッシュボードにアクセスする

前提条件

1. Kubecost 関連のポッドの状態が「実行中」であることを確認します。

```
kubectl get pods --namespace kubecost
```

Kubecost ダッシュボードにアクセスする

1. デバイス上でポート転送を有効にして、Kubecost ダッシュボードを公開します。
   - Helm を使用して Kubecost v3 をインストールしている場合:

     ```
     kubectl port-forward deployment/kubecost-frontend 9090 --namespace kubecost
     ```

   - Helm を使用して Kubecost v1 または v2 をインストールしている場合:

     ```
     kubectl port-forward deployment/kubecost-cost-analyzer 9090 --namespace kubecost
     ```

   - Amazon EKS アドオンを使用して Kubecost をインストールしている場合:

     ```
     kubectl port-forward deployment/cost-analyzer 9090 --namespace kubecost
     ```

     あるいは、AWS Load Balancer Controller を使用して、認証、認可、およびユーザー管理のために Kubecost を公開して Amazon Cognito を使用することもできます。詳細については、「Application Load Balancer および Amazon Cognito を使用して Kubernetes ウェブアプリのユーザーを認証する方法」を参照してください。
2. 前の手順を実行したのと同じデバイスで、Web ブラウザを開き、次のアドレスを入力します。

   ```
   http://localhost:9090
   ```

   ブラウザに Kubecost の概要ページが表示されます。クラスターのサイズによっては、Kubecost がメトリクスを収集するまでに 5 ~ 10 分 (またはそれ以上) かかる場合があります。クラスターにかかる累積コスト、関連する Kubernetes 資産コスト、および毎月の総支出を含む Amazon EKS の支出額を確認できます。

3. クラスターレベルでコストを追跡するには、請求対象の Amazon EKS リソースにタグを付けます。詳細については、「the section called “請求用のリソースにタグを付ける”」を参照してください。
   - コスト配分 – Amazon EKS の月間費用と、各名前空間およびその他のディメンションの過去 7 日間の累積費用を表示します。これは、アプリケーションのどの部分が Amazon EKS の支出に寄与しているかを理解するのに役立ちます。
   - アセット – Amazon EKS リソースに関連する AWS インフラストラクチャアセットの費用を表示します。

Kubecost の詳細

Amazon EKS は、クラスターコストの可視化のため、AWS に最適化された Kubecost のバンドルを提供します。Amazon EKS がサポートする Kubecost を使用して、Pod、ノード、名前空間、ラベルなどの Kubernetes リソースごとにコストを分類して監視することができます。

このトピックでは、Kubecost の使用可能なバージョンと、使用可能な利用枠の違いについて説明します。EKS は Kubecost のバージョン 1、バージョン 2、バージョン 3 をサポートしています。各バージョンは異なる利用枠で利用できます。Amazon EKS 最適化 Kubecost バンドルは、Amazon EKS クラスターにおいて追加コストなしで使用できます。Amazon Managed Service for Prometheus などの関連する AWS サービスの使用に対して課金される場合があります。また、既存の AWS サポート契約を使用してサポートを受けることもできます。

Kubernetes のプラットフォーム管理者および財務リーダーとして、Kubecost を使用して Amazon EKS の請求の内訳を可視化できます。また、コストを配分し、アプリケーションチームなどの組織単位にチャージバックすることも可能です。社内チームや事業部門に、実際の AWS の請求に基づく透明で正確なコストデータを提供できます。さらに、インフラストラクチャ環境とクラスター内の使用パターンに基づいて、コストを最適化するためのカスタマイズされた推奨事項を取得することもできます。Kubecost の詳細については、Kubecost ドキュメントを参照してください。

Kubecost のカスタムバンドルと Kubecost の無料バージョン (OpenCost としても知られている) の違いは何ですか?

AWS と Kubecost が協力して Kubecost のカスタマイズバージョンを提供しました。このバージョンには、追加料金なしで商用機能のサブセットが含まれています。Kubecost のカスタムバンドルに含まれる機能については、以下の表を参照してください。

Kubecost v3

Kubecost の v2 と v3 の違いは何ですか?

Kubecost 3.0 は、大幅なパフォーマンスの向上、強化されたスケーラビリティ、プロアクティブな最適化機能を提供する大規模なアーキテクチャアップグレードです。最も重要な変更は、ClickHouse データベースへの移行によりバージョン 2.8 からの DuckDB を置き換えたことです。これにより、クエリが大幅に高速化され、大規模なパフォーマンスの信頼性向上が実現されます。Kubecost 3.0 では、Kubecost と Cloudability の機能を組み合わせた統合エージェントも導入されており、OpenCost の互換性を維持しながら Prometheus の依存関係を排除し、メモリフットプリントを縮小しています。

> **Important**
>
> v3 へのアップグレードの前に Kubecost ドキュメントを確認してください。v2 からの移行には慎重なプランニングが必要であり、また移行中のレポートの可用性に影響する可能性があります。Helm チャートの場所が `public.ecr.aws/kubecost/kubecost` に変更されました。

v3 での主なアーキテクチャ改善点:

- ClickHouse データベース: DuckDB に代わって ClickHouse データベースを導入したことで、クエリの劇的な高速化とスケーラビリティの向上を実現
- 統合エージェント: Kubecost 機能と Cloudability 機能の組み合わせにより Prometheus の依存関係を排除
- マルチクラスター用の S3 互換ストレージ: マルチクラスターデプロイの場合、v3 は Amazon Managed Service for Prometheus などの Prometheus 互換ストレージの代わりに S3 互換オブジェクトストレージ (EKS のお客様の場合は AWS S3) を使用します。FinOps エージェントは Kubernetes API からメトリクスをプルして S3 互換ストレージにプッシュし、アグリゲータはそのデータをプルして取得ステップを実行し、フロントエンドに結果を表示します。詳細については、Kubecost ドキュメントの「Multi-Cluster Installation」と「Secondary Clusters Guide」を参照してください。
- メモリフットプリントの縮小: 機能を維持しながらリソース要件を大幅に軽減
- アーキテクチャの簡素化: 独立したスケーリングと耐障害性の向上のための単一コンテナポッドトポロジを採用
- 自動化の強化: マルチクラスター対応とカスタムプロファイルによるコンテナリクエストの自動サイズ設定

v3 での Amazon EKS 最適化バンドルの利点:

Amazon EKS 最適化 Kubecost バンドルは追加料金なしで引き続き利用でき、Kubecost v3 無料利用枠で導入された新しい 100,000 USD の支出制限から除外されます。EKS ユーザーは、支出レベルに関係なく、すべての Kubernetes 支出機能へのフルアクセスを保持します。

主要機能の比較:

機能	Kubecost 無料利用枠 3.0	Amazon EKS 最適化 Kubecost バンドル 3.0	Kubecost Enterprise 3.0
[クラスターコストの可視化]	30 日間で 100,000 USD の支出をゲートとする無制限のクラスター	支出制限のない統合マルチクラスター	無制限の数の環境 (マルチクラウドなど) で統合クラスターの数が無制限
データベースバックエンド	ClickHouse (ローカル)	マルチクラスターメトリクス用の S3 互換ストレージを備えた ClickHouse	カスタムデータベースオプションを使用した ClickHouse
パフォーマンス	v2 よりも大幅に高速なクエリ	v2 よりも大幅に高速なクエリ	v2 よりも大幅に高速なクエリ
メモリフットプリント	v2 と比較して縮小 (Prometheus の依存関係なし)	v2 と比較して縮小 (Prometheus の依存関係なし)	v2 と比較して縮小 (Prometheus の依存関係なし)
自動コンテナリクエストのサイズ設定	使用可能 (250 コアに制限)	コア制限なしで使用可能	コア制限なしで使用可能
支出制限	30 日間で 100,000 USD	支出制限なし	支出制限なし
マルチクラスターオートメーション	制限あり	安全なメッセージングによる完全マルチクラスター対応	安全なメッセージングによる完全マルチクラスター対応

Kubecost v2

Kubecost の v1 と v2 の違いは何ですか?

Kubecost 2.0 は以前のバージョンからのメジャーアップグレードであり、まったく新しい API バックエンドなどの主要な新機能が含まれています。なお、Allocation API と Assets API は完全に下位互換性があります。Kubecost ドキュメントを確認して、スムーズに移行してください。機能強化の完全なリストについては、Kubecost v2.0 の発表とすべてのリリースノートを参照してください。

> **Important**
> アップグレードの前に Kubecost ドキュメントを確認してください。アップグレードすると、レポートの可用性に影響する場合があります。

主要機能の比較:

機能	Kubecost 無料利用枠 2.0	Amazon EKS 最適化 Kubecost バンドル 2.0	Kubecost Enterprise 2.0
[クラスターコストの可視化]	クラスター数に制限なし (コア数は最大 250)	Amazon Managed Service for Prometheus と統合した場合、コア制限のない統合マルチクラスター	無制限の数の環境 (マルチクラウドなど) で統合クラスターの数が無制限
デプロイ	ユーザーホスト	ユーザーホスト	ユーザーホスティング、Kubecost ホスティング (専用テナント)、SaaS
[サポートされるデータベース]	ローカル Prometheus	Amazon Managed Service for Prometheus またはローカル Prometheus	任意の Prometheus 風およびカスタムのデータベース

機能	Kubecost 無料利用枠 2.0	Amazon EKS 最適化 Kubecost バンドル 2.0	Kubecost Enterprise 2.0
データベース保持のサポート (未加工メトリクス)	15 日間	無制限の履歴データ	無制限の履歴データ
Kubecost API および UI の保持 (ETL)	15 日間	15 日間	無制限
[ハイブリッドクラウドの可視性]	-	Amazon EKS と Amazon EKS Anywhere クラスター	マルチクラウドおよびハイブリッドクラウド
[アラートと定期レポート]	プライマリクラスターでのみサポート、250 コアに制限	すべてのクラスターで効率性アラート、予算アラート、支出変更アラートなどをサポート	すべてのクラスターで効率性アラート、予算アラート、支出変更アラートなどをサポート
[保存されたレポート]	-	15 日間のメトリクスを使用したレポート	無制限の履歴データおよびメトリクスを使用したレポート
[クラウド請求の統合]	プライマリクラスターでのみサポート、250 コアに制限	AWS のカスタム価格サポート (複数のクラスターと複数のアカウントを含む)	任意のクラウドに対するカスタム料金のサポート
[割引に関する推奨事項]	プライマリクラスターでのみサポート、250 コアに制限	プライマリクラスターのインサイト、ただし 250 コアの制限なし	マルチクラスターの洞察
[ガバナンス: 監査]	-	-	過去のコストイベントの監査

機能	Kubecost 無料利用枠 2.0	Amazon EKS 最適化 Kubecost バンドル 2.0	Kubecost Enterprise 2.0
[シングルサインオン (SSO) のサポート]	-	Amazon Cognito のサポート	Okta、Auth 0、PingID、KeyCloak、その他のカスタムのもの
SAML 2.0 使用したロールベースのアクセスコントロール (RBAC)	-	-	Okta、Auth 0、PingID、KeyCloak、その他のカスタムのもの
[エンタープライズトレーニングとオンボーディング]	-	-	フルサービストレーニングおよび FinOps オンボーディング
チーム	-	-	はい

新機能:

以下の機能にはメトリクス制限があります。

- Kubecost アグリゲータ
- ネットワークモニタリング
- Kubecost アクション
- コレクション
- 異常検出
- コンテナリクエストの適切なサイズ設定
- Kubecost 予測
- フィルタリングと集計のオートコンプリート

メトリクスの制限:

メトリクス	Kubecost 無料利用枠 2.0	Amazon EKS 最適化 Kubecost バンドル 2.0	Kubecost Enterprise 2.0
クラスターサイズ	クラスター数に制限なし (コア数は最大 250)	無制限	無制限
メトリクスの保持	15 日間	15 日間	無制限
マルチクラスターのサポート	利用不可	利用可能	使用可能
コアの制限	クラスターあたり 250 コア	コア制限なし	コア制限なし

Kubecost v1

機能	Kubecost 無料利用枠	Amazon EKS 最適化 Kubecost バンドル	Kubecost Enterprise
デプロイメント	ユーザーホスト	ユーザーホスト	ユーザーホストまたは Kubecost ホスト (SaaS)
[サポートされるクラスター数]	無制限	無制限	無制限
[サポートされるデータベース]	ローカル Prometheus	ローカル Prometheus または Amazon Managed Service for Prometheus	Prometheus、Amazon Managed Service for Prometheus、Cortex、または Thanos
[データベース保持サポート]	15 日間	無制限の履歴データ	無制限の履歴データ

機能	Kubecost 無料利用枠	Amazon EKS 最適化 Kubecost バンドル	Kubecost Enterprise
Kubecost API 保持 (ETL)	15 日間	15 日間	無制限の履歴データ
[クラスターコストの可視化]	単一クラスター	統合マルチクラスター	統合マルチクラスター
[ハイブリッドクラウドの可視性]	-	Amazon EKS と Amazon EKS Anywhere クラスター	マルチクラウドとハイブリッドクラウドのサポート
[アラートと定期レポート]	-	効率性アラート、予算アラート、支出変更アラートなどをサポート	効率性アラート、予算アラート、支出変更アラートなどをサポート
[保存されたレポート]	-	15 日間のデータを使用したレポート	無制限の履歴データを使用したレポート
[クラウド請求の統合]	個々のクラスターに必須	AWS のカスタム価格サポート (複数のクラスターと複数のアカウントを含む)	AWS のカスタム価格サポート (複数のクラスターと複数のアカウントを含む)
[割引に関する推奨事項]	単一クラスターの洞察	単一クラスターの洞察	マルチクラスターの洞察
[ガバナンス: 監査]	-	-	過去のコストイベントの監査
[シングルサインオン (SSO) のサポート]	-	Amazon Cognito のサポート	Okta、Auth 0、PingID、KeyCloak
[SAML **2.0** を使用したロールベースのアクセスコントロール (RBAC)]	-	-	Okta、Auth 0、PingID、Keycloak

機能	Kubecost 無料利用枠	Amazon EKS 最適化 Kubecost バンドル	Kubecost Enterprise
[エンタープライズトレーニングとオンボーディング]	-	-	フルサービストレーニングおよび FinOps オンボーディング

よくある質問

Amazon EKS での Kubecost の使用に関する次の一般的な質問と回答を参照してください。

Kubecost API 保持 (ETL) 機能とは何ですか?

Kubecost ETL 機能では、メトリクスを集約して整理し、さまざまな詳細度レベル (`namespace-level`、`pod-level`、`deployment-level`など) で表面的なコストを可視化します。Amazon EKS 最適化 Kubecost バンドルでは、お客様は過去 15 日間のメトリクスからデータとインサイトを取得します。

アラート機能と定期レポート機能とは何ですか? どのようなアラートとレポートが含まれていますか?

Kubecost アラートにより、チームはリアルタイムの Kubecost 支出とクラウド支出に関する最新情報を受け取ることができます。定期レポートでは、チームは Kubernetes 支出とクラウド支出の履歴のカスタマイズされたビューを受け取ることができます。どちらも、Kubecost UI または Helm 値で設定できます。E メール、Slack、Microsoft Teams がサポートされています。

保存されたレポートには何が含まれていますか?

Kubecost の保存されたレポートは、コストと効率のメトリクスに関する事前定義されたビューです。クラスター、名前空間、ラベル、その他を基準とするコストが含まれます。

クラウド請求インテグレーションとは何ですか?

AWS 請求 API との統合により、Kubecost でクラスター以外のコスト (Amazon S3 など) を表示できます。さらに、Kubecost で、スポット使用量、Savings Plans、法人割引を考慮して、Kubecost のクラスター内予測を実際の請求データと一致させられるようになります。

割引に関する推奨事項にはどのようなものがありますか?

Kubecost では、ユーザーが Kubernetes インフラストラクチャおよび支出を最適化するのに役立つインサイトと自動化が提供されます。

この機能には料金がかかりますか?

いいえ。Amazon EKS 最適化 Kubecost バンドルは、追加料金なしで使用できます。これに含まれていない Kubecost 機能が追加で必要な場合は、AWS Marketplace を通じて、または直接 Kubecost から、Kubecost のエンタープライズライセンスを購入できます。

Amazon EKS 最適化 Kubecost バンドルでは、サポートを利用できますか?

はい。Amazon EKS 最適化 Kubecost バンドルを使用している場合にのみ利用可能です。

どうすれば、Amazon EKS 最適化 Kubecost バンドルのサポートを受けることができますか?

AWS へのお問い合わせから AWS サポートチームに対してサポートケースを開くことができます。

Amazon EKS 統合によって提供される Kubecost 機能を使用するのにライセンスが必要ですか?

いいえ。

より正確なレポートを作成するために、Kubecost と AWS コストと使用状況レポートを統合できますか?

はい。AWS コストと使用状況レポートからデータを取り込むように Kubecost を構成することで、割引、スポット料金、予約インスタンス料金などを含む正確なコストを可視化することができます。詳細については、Kubecost ドキュメントの「AWS Cloud Billing Integration」を参照してください。

このバージョンは、Amazon EC2 でのセルフマネージド Kubernetes クラスターのコスト管理をサポートしていますか?

いいえ。Amazon EKS 最適化 Kubecost バンドルは、Amazon EKS クラスターとのみ互換性があります。

Kubecost は AWS Fargate での Amazon EKS のコストを追跡できますか?

Kubecost では、Fargate での Amazon EKS のクラスターコストが可能な限り可視化されますが、Amazon EC2 での Amazon EKS に比べて精度が低くなります。これは主に、使用量に対する課金方法の違いによるものです。Fargate での Amazon EKS では、消費したリソースに対して課金されます。Amazon EC2 ノードでの Amazon EKS では、プロビジョニングしたリソースに対して課金されます。Kubecost は、CPU、RAM、エフェメラルストレージなどのノード仕様に基づいて Amazon EC2 ノードのコストを計算します。Fargate では、コストは Fargate ポッドのリクエストされたリソースに基づいて計算されます。

Kubecost の更新と新バージョンはどうすれば入手できますか?

Kubecost バージョンは、標準の Helm アップグレード手順でアップグレードできます。Kubecost v3 の場合、最新バージョンは新しい Helm チャートの場所 `public.ecr.aws/kubecost/kubecost` で入手できます。以前のバージョン (v1 および v2) は、Amazon ECR Public Gallery で引き続き使用できます。

> **Important**
>
> Kubecost v3 にアップグレードする場合、Helm チャートの場所が `public.ecr.aws/kubecost/cost-analyzer` から `public.ecr.aws/kubecost/kubecost` に変更されていることに注意してください。それに応じて Helm リポジトリのリファレンスを更新してください。

`kubectl-cost` CLI はサポートされていますか? どのようにインストールすればよいですか?

はい。`kubectl-cost` は、Kubecost (Apache 2.0 ライセンス) によるオープンソースツールで、Kubernetes コスト配分のメトリクスに CLI でアクセスできます。`kubectl-cost` をインストールするには、「GitHub」の「インストール」を参照してください。

Kubecost ユーザーインタフェイスはサポートされていますか? どうすれば使用できますか?

Kubecost では、`kubectl` ポート転送、イングレス、またはロードバランサーを介してアクセスできる Web ダッシュボードが提供されます。または、AWS Load Balancer Controller を使用して Kubecost を公開し、Amazon Cognito を使用して認証、認可、およびユーザー管理を行うこともできます。詳細については、AWS ブログの「How to use Application Load Balancer and Amazon Cognito to authenticate users for your Kubernetes web apps」を参照してください。

Kubecost v3 の新しい 100,000 USD の支出制限は Amazon EKS ユーザーに影響しますか?

いいえ。Kubecost v3 無料利用枠に導入される 30 日間の 100,000 USD 支出制限は、Amazon EKS 最適化 Kubecost バンドルユーザーには適用されません。EKS ユーザーは、支出レベルに関係なく、すべての Kubernetes 支出機能へのフルアクセスを保持します。

Kubecost v3 での主なパフォーマンス改善点はどんなものがありますか?

Kubecost v3 では、ClickHouse データベースバックエンドの導入によりパフォーマンスが大幅に向上しています。これにより、v2.8 で使用している DuckDB と比較してクエリが大幅に高速化されます。さらに、統合エージェントアーキテクチャは Prometheus の依存関係を排除し、フル機能と OpenCost の互換性を維持しながらメモリフットプリントを縮小します。

Kubecost v3 はマルチクラスターデプロイにどのようなストレージバックエンドを使用しますか?

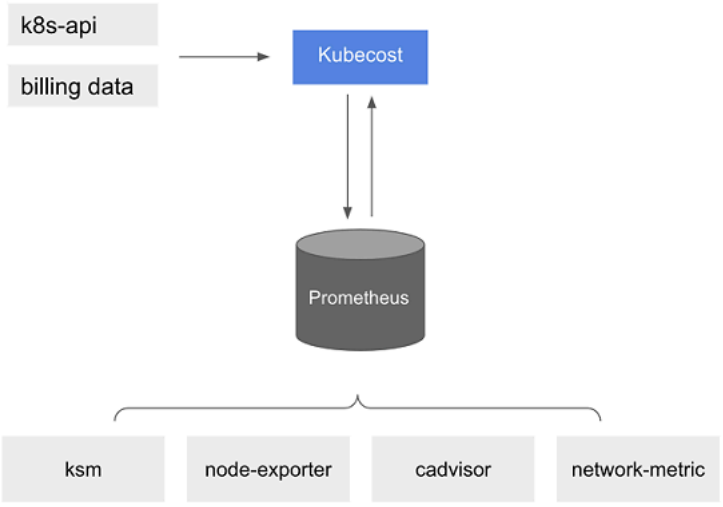
Kubecost v3 では、v2 で使用されていた Prometheus 互換ストレージに代わり、マルチクラスターメトリクスストレージとして S3 互換オブジェクトストレージ (EKS のお客様の場合は AWS S3) を使用します。FinOps エージェントは、Kubernetes API からメトリクスを収集し、S3 互換ストレージにプッシュします。次に、アグリゲータはこのデータを取得し、コスト計算を実行し、フロントエンドに結果を表示します。マルチクラスターの設定手順の詳細については、Kubecost ドキュメントの「Multi-Cluster Installation」と「Secondary Clusters Guide」を参照してください。

Kubecost v1 から v3 に直接アップグレードできますか?

いいえ。v1 から v3 への直接アップグレードはサポートされていません。まず v2 にアップグレードしてから v3 に移行する必要があります。このプロセスには慎重なプランニングが必要であり、移行中のレポートの可用性に影響する可能性もあるため、詳細な移行ガイダンスについては Kubecost ドキュメントを参照してください。

Kubecost のその他の機能

- Kubecost の v1、v2、v3 で次の機能を使用できます。
  - コストメトリクスのエクスポート – Amazon EKS 最適化コストモニタリングは Kubecost でデプロイされます。v1 および v2 では、Kubecost は Prometheus と統合してメトリクスの保存と処理を行います。v3 では、Kubecost は ClickHouse データベースを使用して、OpenCost の互換性を維持しながらパフォーマンスを大幅に向上させます。v3 のマルチクラスターデプロイの場合、メトリクスは Prometheus 互換ストレージではなく S3 互換オブジェクトストレージ (EKS のお客様の場合は AWS S3) に保存されます。Kubecost は、メトリクスを読み取り、コスト配分計算を実行し、API とユーザーインターフェイスを通じてデータを提供します。アーキテクチャはバージョンによって異なりますが、一貫した機能性は維持されます。



クエリを記述して Kubecost データをインジェストし、お使いのビジネスインテリジェンスシステムに取り込んでさらに分析できます。また、現在の Grafana ダッシュボードのデータソースとして使用して、社内チームが精通している Amazon EKS クラスターの費用を表示することもできます。クエリの記述方法の詳細については「OpenCost Configuration」ドキュメントを参照するか、「Kubecost GitHub リポジトリ」の Grafana JSON モデル例を参考にしてください。

- AWS コストと使用状況レポートの統合 — Amazon EKS クラスターのコスト配分計算を実行するために、Kubecost は AWS Price List API から AWS サービスおよび AWS リソースの公開価格情報を取得します。Kubecost と AWS コストと使用状況レポートを統合して、AWS アカウントに固有の価格情報の正確性を高めることもできます。この情報には、エンタープライズディスカウントプログラム、リザーブドインスタンスの使用量、Savings Plans、スポット使用量が含まれます。AWS コストと使用状況レポートの統合の動作方法の詳細については、Kubecost ドキュメントの「AWS Cloud Billing Integration」を参照してください。

Kubernetes Metrics Server でのリソース使用状況の表示

Kubernetes Metrics Server は、クラスター内のリソース使用状況データを集約する働きをし、デフォルトでは Amazon EKS クラスターにデプロイされません。詳細については、GitHub の「Kubernetes Metrics Server (Kubernetes メトリクスサーバー)」を参照してください。Kubernetes Metrics Server は、Scale pod deployments with Horizontal Pod Autoscaler や Kubernetes Dashboard など他の Kubernetes アドオンによってよく使用されます。詳細については、Kubernetes ドキュメントの「Resource metrics pipeline」を参照してください。このトピックでは、Kubernetes メトリクスサーバーを Amazon EKS クラスターにデプロイする方法について説明します。

> **Important**
>
> メトリクスはポイントインタイム分析を目的としており、履歴分析の正確なソースではありません。監視ソリューションとして、またはその他の自動スケーリング以外の目的には使用できません。監視ツールの詳細については「クラスターをモニタリングする」を参照してください。

考慮事項

- マニフェストを使用して Kubernetes メトリクスサーバーを Fargate ノードに手動でデプロイする場合は、デフォルトの `10250` 以外のポートを使用するように `metrics-server` デプロイを設定

します。このポートは Fargate で予約されています。メトリクスサーバーの Amazon EKS アドオンバージョンは、10251 ポートを使用するように事前設定されています。

- セキュリティグループとネットワーク ACL で、metrics-server ポッドと他のすべてのノードおよびポッド間の 10250 ポートが許可されていることを確認します。Kubernetes メトリクスサーバーでは、引き続き 10250 ポートを使用してクラスター内の他のエンドポイントからメトリクスを収集します。Fargate ノードにデプロイする場合は、設定された代替メトリクスサーバーポートと 10250 ポートの両方を許可します。

Amazon EKS アドオンを使用してコミュニティアドオンとしてデプロイする

新規: AWSコンソールまたは Amazon EKS APIs Server をコミュニティアドオンとしてデプロイできるようになりました。

AWS コンソールでデプロイ

1. AWS コンソールで EKS クラスターを開きます
2. 「アドオン」タブから Get More Add-ons.を選択します。
3. 「コミュニティアドオン」セクションで、メトリクスサーバーを選択し、次へを選択してください。
4. EKS はクラスターに適したバージョンのアドオンを決定します。バージョンドロップダウン メニューを使用してバージョンを変更できます。
5. 次へ を選択し、作成 を選択してアドオンをインストールします。

その他のリソース

the section called “コミュニティアドオン” の詳細を確認してください。

コミュニティアドオンは他の Amazon EKS アドオンと同じ方法でインストールまたは更新します。

- the section called “アドオンを作成する”
- the section called “アドオンを更新する”
- the section called “アドオンを削除する”

マニフェストを使用してデプロイする

新規: AWSコンソールまたは Amazon EKS APIs Server をコミュニティアドオンとしてデプロイできるようになりました。これらのマニフェストのインストール手順はアーカイブされます。

1. 次のコマンドを使用してメトリクスサーバーをデプロイします。

```
kubectl apply -f https://github.com/kubernetes-sigs/metrics-server/releases/latest/download/components.yaml
```

   Fargate を使用している場合はこのファイルを変更する必要があります。デフォルト設定ではメトリクスサーバーはポート 10250 を使用します。このポートは Fargate で予約されています。components.yaml のポート 10250 への参照を 10251 などの別のポートに置き換えます。

2. 次のコマンドを使用して、`metrics-server` デプロイで必要な数の Pod が実行されていることを確認します。

```
kubectl get deployment metrics-server -n kube-system
```

   出力例は次のとおりです。

```
NAME             READY   UP-TO-DATE   AVAILABLE   AGE
metrics-server   1/1     1            1           6m
```

3. ノードのリソース (CPU/メモリ) 使用状況を表示して、メトリクスサーバーが動作しているかテストします。

```
kubectl top nodes
```

4. エラーメッセージ `Error from server (Forbidden)` が表示された場合はKubernetes RBAC 設定を更新する必要があります。Kubernetes RBAC ID にはクラスターメトリクスを読み取るための十分なアクセス許可が必要です。メトリクスを読み取るために必要な最低限の Kubernetes API アクセス許可を GitHub で確認します。役割などの AWS IAM ID に Kubernetes API へのアクセス権を付与する方法について説明します。

Helm を使用して Amazon EKS にアプリケーションをデプロイする

Kubernetes 用の Helm パッケージマネージャーを使用すると、Kubernetes クラスターにアプリケーションをインストールし、管理できます。詳細についてはHelm のドキュメントを参照してください。このトピックはHelm バイナリをインストールして実行する際に役立ちます。これにより、ローカルシステムで Helm CLI を使用してチャートをインストールおよび管理することができます。

> **Important**
>
> Amazon EKS クラスターに Helm チャートをインストールするにはあらかじめ Amazon EKS で動作するように `kubectl` を設定しておく必要があります。この設定をまだ行っていない場合は続行する前に「the section called “kubectl を使用してクラスターにアクセスする”」を参照してください。次のコマンドがクラスターに対して成功した場合は正しく設定されています。
>
> ```
> kubectl get svc
> ```

1. クライアントオペレーティングシステムに適したコマンドを実行してください。
   - macOS で Homebrew を使用している場合は次のコマンドを使用してバイナリをインストールします。

     ```
     brew install helm
     ```
   - その他のインストールオプションについては、Helm ドキュメントの「Helm のインストール」を参照してください。

     > **Note**
     >
     > 最初に `openssl` をインストールする必要があるというメッセージを受け取った場合、次のコマンドを使用してインストールできます。

```
sudo yum install openssl
```

1. PATH で新しいバイナリを取得するため、現在のターミナルウィンドウを閉じて新しいターミナルウィンドウを開きます。
2. インストールした Helm のバージョンを確認してください。

```
helm version --template='{{ .Version }}{{ "\n" }}'
```

出力例は次のとおりです。

```
v3.17.2
```

3. インストールされているバージョンがクラスターのバージョンと互換性があることを確認してください。詳細については、「Supported Version Skew」を確認してください。例えば、`3.17.x` で稼働している場合、サポート対象の Kubernetes バージョンは `1.29.x ~ 1.32.x` の範囲内でなければなりません。
4. この時点で、任意の Helm コマンド (`helm install` *`chart-name`* など 実行して、クラスター内の Helm チャートをインストール、変更、削除、またはクエリすることができます。Helm を初めて使用する場合で、インストールするチャートがないときは次の操作を実行できます：
   - サンプルチャートをインストールして試します。Helm の「クイックスタートガイド」の「サンプルチャートをインストールする」を参照してください。
   - サンプルチャートを作成し、Amazon ECR にプッシュします。詳細については、「Amazon Elastic Container Registry ユーザーガイド」の「Pushing a Helm chart (Helm チャートをプッシュする)」を参照してください。
   - [eks-charts] GitHub リポジトリか [ArtifactHub] から Amazon EKS チャートをインストールします。

タグを使用して Amazon EKS リソースを整理する

タグ を使用すると、Amazon EKS リソースの管理に役立ちます。このトピックでは、タグ機能の概要とタグの作成方法について説明します。

トピック

- タグの基本
- リソースのタグ付け
- タグの制限
- 請求用のリソースにタグを付ける

- コンソールでのタグの処理
- CLI、API または eksctl でのタグの操作

> **Note**
>
> タグは、Kubernetes のラベルやアノテーションとは別のメタデータのタイプです。これらのその他のメタデータタイプの詳細については、Kubernetes ドキュメントの次のセクションを参照してください。
>
> - ラベル (Labels) とセレクター (Selectors)
> - 注釈

タグの基本

タグとはAWS リソースに割り当てるラベルです。各タグはキーとオプションの値で構成されます。

タグを使用して AWS リソースを分類できます。例えば、目的、所有者、環境などに基づいてリソースを分類できます。同じ型のリソースが多い場合に、特定のリソースに割り当てたタグを使用して、そのリソースをすばやく識別できます。例えば Amazon EKS クラスターに一連のタグを定義して、各クラスターの所有者とスタックレベルを追跡できます。リソースタイプごとに一貫した一連のタグキーを考案することをお勧めします。追加したタグに基づいてリソースを検索およびフィルタリングできます。

タグを追加したら、いつでもタグキーと値は編集でき、タグはリソースからいつでも削除できます。リソースを削除すると、リソースのタグも削除されます。

タグには Amazon EKS に関する意味論的な意味はなく、完全に文字列として解釈されます。タグの値は空の文字列に設定できます。ただし、タグの値を null に設定することはできません。そのリソースの既存のタグと同じキーを持つタグを追加した場合、古い値は新しい値によって上書きされます。

AWS Identity and Access Management (IAM) を使用すると、AWS アカウント内のどのユーザーがタグを管理するアクセス許可を持つかをコントロールできます。

リソースのタグ付け

Amazon EKS の以下のリソースがタグをサポートしています。

- クラスター
- マネージド型ノードグループ
- Fargate プロファイル

以下を使用して、これらのリソースにタグ付けできます。

- Amazon EKS コンソールを使用している場合、新規または既存のリソースにいつでもタグを適用できます。これを行うには、関連リソースページの [Tags] (タグ) タブを使用します。詳細については、「the section called “コンソールでのタグの処理”」を参照してください。
- `eksctl` を使用している場合は、`--tags` オプションを使ってリソースを作成する際に、リソースにタグを適用できます。
- AWS CLI、Amazon EKS API、または AWS SDK を使用している場合は、関連する API アクションの `tags` パラメータを使用して、新しいリソースにタグを適用できます。`TagResource` API アクションを使用して既存のリソースにタグを適用することもできます。詳細については、「TagResource」を参照してください。

リソース作成アクションによっては、リソースの作成時にリソースのタグを指定することもできます。リソースの作成中にタグを適用できない場合、リソースの作成は失敗します。これにより、タグ付けするリソースが、指定したタグで作成されているか、まったく作成されていないかが確認できます。リソースの作成時にタグ付けを行う場合、リソースの作成後にカスタムタグ付けスクリプトを実行する必要はありません。

タグは、作成したリソースに関連付けられている他のリソースには伝達されません。例えば、Fargate プロファイルのタグは、それによってスケジュールされた Pod などの、Fargate プロファイルに関連付けられている他のリソースには適用されません。

タグの制限

タグには以下の制限があります。

- 1 つのリソースに対して最大 50 個のタグを関連付けることができます。
- 1 つのリソースに対してタグキーを繰り返すことはできません。各タグキーは一意である必要があり、それぞれに使用できる値は 1 つのみです。
- キーの値には最大 128 UTF-8 文字を使用できます。
- 値の最大長は 256 UTF-8 文字です。

- 複数の AWS サービスおよびリソースがタグ付けスキーマを使用する場合、使用する文字の種類を制限します。一部のサービスでは、使用できる文字に制限がある場合があります。通常、使用できる文字は、英字、数字、スペース、および特殊文字 +、-、=、.、_、:、/、@ です。
- タグのキーと値では、大文字と小文字が区別されます。
- キーまたは値のプレフィックスとして、aws:、 AWS:、またはその大文字と小文字の組み合わせを変えたものは使用しないでください。これらは AWS でのみ使用するように予約されています。このプレフィックスを持つタグのキーや値を編集または削除することはできません。このプレフィックスを持つタグは、リソースあたりのタグ数の制限にはカウントされません。

請求用のリソースにタグを付ける

Amazon EKS クラスターにタグを適用すると、そのタグを [Cost & Usage Reports] (コストと使用状況レポート) 内のコスト配分に使用できます。[Cost & Usage Reports] (コストと使用状況レポート) の計測データには、すべての Amazon EKS タスクの使用状況が示されます。詳細については、AWS 請求ユーザーガイドの「AWS コストと使用状況レポート」を参照してください。

AWS 生成されたコスト配分タグ、特に aws:eks:cluster-name では、[Cost Explorer] (コストエクスプローラー) で Amazon EC2 インスタンスのコストを個々の Amazon EKS クラスターごとに分類できます。ただし、このタグにはコントロールプレーンの費用を捕捉するものではありません。このタグは、Amazon EKS クラスターに参加している Amazon EC2 インスタンスに自動的に追加されます。この動作は、インスタンスが Amazon EKS マネージド型ノードグループまたは Karpenter を使用して、または直接 Amazon EC2 を使用してプロビジョニングされたかどうかに関係なく、発生します。この特定のタグは 50 個のタグの上限にはカウントされません。タグを使用するには、アカウント所有者が AWS 請求コンソールまたは API を使用してタグをアクティブ化する必要があります。AWS Organizations 管理アカウントの所有者がこのタグをアクティブ化すると、すべての組織のメンバーアカウントにも有効になります。

同じタグキー値を持つリソースに基づいて、請求情報を整理することもできます。例えば、複数のリソースに特定のアプリケーション名のタグを付け、請求情報を整理することができます。これにより、複数のサービスを利用しているアプリケーションの合計コストを確認することができます。タグを使用したコスト配分レポートの設定の詳細については、 AWS 請求ユーザーガイドの「コスト配分月次レポート」を参照してください。

> **Note**
>
> レポートを有効にすると、約 24 時間後に、今月のデータを表示できるようになります。

Cost Explorer は、AWS 無料利用枠の一部として利用できるレポートツールです。Cost Explorer を使用すると、過去 13 か月間の Amazon EKS リソースのグラフを表示できます。また、次の 3 か月間のリソース使用量も予測できます。時間の経過と共に AWS リソースに費やす金額のパターンを確認できます。例えば、Cost Explorer を使用して、さらに調べる必要がある分野を特定し、コストを把握するために使用できる傾向を確認できます。データの時間範囲を指定したり、時間データを日または月ごとに表示することもできます。

コンソールでのタグの処理

Amazon EKS コンソールを使用して、新規または既存のクラスターおよびマネージド型ノードグループに関連付けられたタグを管理できます。

Amazon EKS コンソールでリソース固有のページを選択すると、そのページにリソースのリストが表示されます。例えば、左のナビゲーションペインで [クラスター] を選択すると、Amazon EKS クラスターのリストがコンソールに表示されます。これらのリストの1つ(例えば、特定のクラスター)からタグをサポートするリソースを選択すると、Tagsタブでタグを表示および管理できます。

また、AWS マネジメントコンソール の [タグエディタ] を使用することもできます。これにより、タグを一貫性のある方法で管理できます。詳細については、AWS タグエディタユーザーガイドの「タグエディタを使用して AWS リソースにタグを付ける」を参照してください。

リソース作成時のタグの追加

Amazon EKS クラスターおよびマネージド型ノードグループ、および Fargate プロファイルの作成時に、タグを追加できます。詳細については、「the section called “クラスターを作成する”」を参照してください。

リソースでのタグの追加と削除

クラスターに関連付けられたタグをリソースのページから直接追加または削除できます。

1. Amazon EKS コンソールを開きます。
2. ナビゲーションバーから、使用する AWS リージョンを選択します。
3. 左のナビゲーションペインで [クラスター] を選択します。
4. 指定するクラスターを選択します。
5. [タグ] タブを選択し、[タグ管理] を選択します。
6. [Manage tags] (タグの管理) ページで、必要に応じてタグを追加または削除します。

- タグを追加するには、[タグの追加] を選択します。その後、タグごとにキーと値を指定します。
- タグを削除するには、[タグの削除] を選択します。

7. 追加または削除するタグごとにこのプロセスを繰り返します。
8. [更新] を選択して終了します。

CLI、API または `eksctl` でのタグの操作

リソースのタグの追加、更新、リスト表示、および削除には、次の AWS CLI コマンドまたは Amazon EKS API オペレーションを使用します。1 つのコマンドで新しいリソースを同時に作成しながらタグを追加するのに使用できるのは `eksctl` だけです。

タスク	AWS CLI	AWS Tools for Windows PowerShell	API アクション
1 つ以上のタグを追加、または上書きします。	タグリソース	Add-EKSResourceTag	TagResource
1 つ以上のタグを削除します。	タグなしリソース	Remove-EKSResourceTag	UntagResource

以下の例では、AWS CLI を使用して、リソースに対してタグ付けまたはタグ削除する方法を示しています。

例 1: 既存のクラスターへのタグ付け

次のコマンドは既存のクラスターにタグ付けします。

```
aws eks tag-resource --resource-arn resource_ARN --tags team=devs
```

例 2: 既存のクラスターでのタグ削除

次のコマンドは既存のクラスターからタグを削除します。

```
aws eks untag-resource --resource-arn resource_ARN --tag-keys tag_key
```

例 3: リソースのタグのリスト取得

次のコマンドは、既存のリソースに関連付けられているタグのリストを取得します。

```
aws eks list-tags-for-resource --resource-arn resource_ARN
```

一部のリソース作成アクションでは、リソースの作成と同時にタグを指定できます。次のアクションでは、リソース作成時のタグの指定がサポートされます。

タスク	AWS CLI	AWS Tools for Windows PowerShell	API アクション	eksctl
クラスターを作成する	クラスター作成	New-EKSCluster	CreateCluste:	`create cluster`
マネージド型ノードグループの作成*	create-nodegroup	New-EKSNodegroup	CreateNodegroup	`create nodegroup`
Fargate プロファイルの作成	create-fargate-profile	New-EKSFargateProfile	CreateFargateProfile	`create fargateprofile`

- マネージド型ノードグループの作成時に Amazon EC2 インスタンスにもタグ付けを行う場合は、起動テンプレートを使用してマネージド型ノードグループを作成します。詳細については、「the section called “Amazon EC2 インスタンスへのタグ付け”」を参照してください。インスタンスがすでに存在する場合は、インスタンスに手動でタグ付けを行うことができます。詳細については、Amazon EC2 ユーザーガイドの「リソースのタグ付け」を参照してください。

Amazon EKS と Fargate Service Quotas を表示して管理する

Amazon EKS は、Service Quotas と統合されています。Service Quotas は、クォータを一元的な場所から表示および管理するために使用できる AWS サービスです。詳細は、Service Quotas ユーザーガイドの「Service Quotas とは」を参照してください。Service Quotas との統合により、AWS マネジメントコンソール および AWS CLI を使用して、Amazon EKS および AWS Fargate Service Quotas の値を簡単に検索できます。

AWS マネジメントコンソール で EKS Service Quotas を表示する

1. Service Quotas コンソールを開きます。
2. 左のナビゲーションペインで [AWS サービス] を選択します。
3. [AWS サービス] リストから、[Amazon Elastic Kubernetes Service (Amazon EKS)] または [AWS Fargate] を選択します。

   [Service Quotas] の一覧には、Service Quotas 名、適用された値 (使用可能な場合)、AWS のデフォルトのクォータ、クォータ値が調整可能かどうかが表示されます。
4. 説明など、Service Quotas に関する追加情報を表示するには、クォータ名を選択します。
5. (オプション) クォータの引き上げをリクエストするには、[Request quota increase (クォータ引き上げリクエスト)] を選択、または必要な情報を入力または選択して、[Request (リクエスト)] を選択します。

AWS マネジメントコンソール を使用してさらに Service Quotas の操作を行うには、「Service Quotas ユーザーガイド」を参照してください。クォータの引き上げをリクエストするには、Service Quotas ユーザーガイド の「クォータ引き上げリクエスト」を参照してください。

AWS CLI で EKS サービスクォータを表示する

次のコマンドを実行して、Amazon EKS クォータを表示します。

```
aws service-quotas list-aws-default-service-quotas \
    --query 'Quotas[*].
{Adjustable:Adjustable,Name:QuotaName,Value:Value,Code:QuotaCode}' \
    --service-code eks \
    --output table
```

次のコマンドを実行して、Fargate クォータを表示します。

```
aws service-quotas list-aws-default-service-quotas \
    --query 'Quotas[*].
{Adjustable:Adjustable,Name:QuotaName,Value:Value,Code:QuotaCode}' \
    --service-code fargate \
    --output table
```

> **Note**
>
> 返されるクォータは、現在の AWS リージョンのこのアカウントの Fargate で同時に実行可能な Amazon ECS タスクまたは Amazon EKS Pod の数です。

AWS CLI を使用して、さらに Service Quotas の操作を実行するには、AWS CLI コマンドリファレンスの「Service Quotas」を参照してください。クォータの引き上げをリクエストするには、AWS CLI コマンドリファレンスの request-service-quota-increase コマンドを参照してください。

Amazon EKS Service Quotas

AWS では、AWS マネジメントコンソール を使用して現在のクォータを表示することをお勧めします。詳細については、「the section called “AWS マネジメントコンソール で EKS Service Quotas を表示する”」を参照してください。

デフォルトの EKS サービスクォータを表示するには、「AWS 全般のリファレンス」の「Amazon Elastic Kubernetes Service endpoints and quotas」を参照してください。

これらの Service Quotas は、Service Quotas コンソールの [Amazon Elastic Kubernetes Service (Amazon EKS)] に表示されます。調整可能として表示される値のクォータの引き上げをリクエストすることもできます。「Service Quotas ユーザーガイド」の「クォータ引き上げのリクエスト」を参照してください。

> **Note**
>
> Service Quotas では、次のコンポーネントの調整はサポートされていません。* クラスターごとの Pod Identity の関連付け。制限については、「the section called “Pod Identity”」を参照してください。* リモートノードネットワーク用の CIDR またはハイブリッドノード用のリモートポッドネットワーク。制限については、「the section called “ハイブリッドノード”」を参照してください。

AWS Fargate サービスクォータ

Service Quotas コンソールの AWS Fargate サービスは、いくつかのサービスクォータを一覧表示します。使用量がサービスクォータに近づいたときに警告するアラームを設定することもできます。詳細については、「the section called “Fargate のリソース使用状況メトリクスをモニタリングするための CloudWatch アラームの作成”」を参照してください。

新規 AWS アカウントの初期の低いクォータは、経時的に引き上げられる可能性があります。Fargate は、各 AWS リージョン内のアカウント使用率を常に監視し、使用率に基づいてクォータを自動的に引き上げます。調整可能として表示される値のクォータの引き上げをリクエストすることもできます。詳細については、「Service Quotas ユーザーガイド」の「クォータの引き上げのリクエスト」を参照してください。

AWS では、AWS マネジメントコンソール を使用して現在のクォータを表示することをお勧めします。詳細については、「the section called “AWS マネジメントコンソール で EKS Service Quotas を表示する”」を参照してください。

AWS Fargate on EKS のデフォルトのサービスクォータを表示するには、「AWS 全般のリファレンス」の「Fargate サービスクォータ」を参照してください。

> **Note**
>
> Fargate は、Amazon ECS タスクと Amazon EKS Pod の起動レートクォータを追加で適用します。詳細については、「Amazon ECS ガイド」の「AWS Fargate スロットリングのクォータ」を参照してください。

Amazon EKS のセキュリティ

AWS では、クラウドのセキュリティが最優先事項です。セキュリティを最も重視する組織の要件を満たすために構築された「AWS」のデータセンターとネットワークアーキテクチャは、お客様に大きく貢献します。

セキュリティは、「AWS」と顧客の間の責任共有です。共有責任モデルでは、これをクラウドのセキュリティおよびクラウド内のセキュリティとして説明しています。

- クラウドのセキュリティ – AWS は、AWS クラウドで AWS のサービスを実行するインフラストラクチャを保護する責任を担います。Amazon EKS において、AWS は Kubernetes コントロールプレーンの責任を分担しており、この範囲にはコントロールプレーンノードと `etcd` データベースが含まれています。AWS コンプライアンスプログラムの一環として、サードパーティーの監査が定期的にセキュリティの有効性をテストおよび検証しています。Amazon EKS に適用されるコンプライアンスプログラムについては、「コンプライアンスプログラムによる対象範囲内の AWS サービス」を参照してください。
- クラウド内のセキュリティ – お客様の責任事項には、次の領域が含まれます。
  - データプレーンのセキュリティ設定。これには、Amazon EKS コントロールプレーンからお客様の VPC 内へのトラフィックの通過を許可する、セキュリティグループの設定を含みます。
  - ノードおよびコンテナに対する設定。
  - ノードのオペレーティングシステム (更新やセキュリティパッチを含みます)
  - その他の関連アプリケーションソフトウェア :
    - ファイアウォールのルールなど、ネットワークコントロールの設定および管理。
    - IAM を使用する、またはそれに追加して行う、Identity and Access Managementのプラットフォームレベルの管理。
  - お客様のデータの機密性、企業の要件、該当する法律および規制

Amazon EKS は、規制対象および機密性の高いアプリケーションの複数のコンプライアンスプログラムで認定されています。Amazon EKS は、SOC、PCI、ISO、FedRAMP-Moderate、IRAP、C5、K-ISMS、ENS High、OSPAR、HITRUST CSF に準拠しており、HIPAA 対象サービスです。詳細については、「アクセスを管理する」を参照してください。

このドキュメントは、Amazon EKS を使用する際の責任共有モデルの適用について理解するのに役立ちます。以下のトピックで、セキュリティおよびコンプライアンスの目的を満たすよう

に、Amazon EKS を設定する方法について説明します。Amazon EKS リソースのモニタリングやセキュリティ確保に役立つ他の AWS サービスの使用方法についても説明します。

> **Note**
>
> Linux コンテナはコントロールグループ (cgroup) と名前空間で構成されています。これらにより、コンテナのアクセスが可能な対象を制限しながら、すべてのコンテナに、ホストの Amazon EC2 インスタンスと同じ Linux カーネルを共有させることができます。コンテナをルートユーザー (UID 0) として実行したり、ホストリソースや (ホストネットワークやホスト PID の) 名前空間へのアクセスをコンテナに許可したりすることは一切お勧めしません。これは、コンテナが提供する分離の有効性を低下させます。

トピック

- ベストプラクティスによる Amazon EKS クラスターの保護
- Amazon EKS の脆弱性を分析する
- Amazon EKS クラスターのコンプライアンス検証
- アマゾン エラスティックKubernetesサービス のセキュリティに関する考慮事項
- Kubernetes のセキュリティに関する考慮事項
- Amazon EKS Auto Mode におけるセキュリティに関する考慮事項
- EKS 機能のセキュリティに関する考慮事項
- Amazon EKS の Identity and Access Management

ベストプラクティスによる Amazon EKS クラスターの保護

Amazon EKS でのセキュリティのベストプラクティスは、「Amazon EKS のベストプラクティスガイド」の「セキュリティのベストプラクティス」に記載されています。

Amazon EKS の脆弱性を分析する

セキュリティは、Kubernetes クラスターとアプリケーションの設定や管理を行う際の重要な考慮事項です。以下に、EKS クラスターのセキュリティ設定を分析するためのリソース、脆弱性をチェックするためのリソース、およびその分析を実行できる AWS サービスとの統合を示します。

Amazon EKS 用 Center for Internet Security (CIS) ベンチマーク

Center for Internet Security (CIS) Kubernetes Benchmark では、Amazon EKS のセキュリティ設定に関するガイダンスを提供しています。ベンチマーク:

- ユーザーが Kubernetes コンポーネントのセキュリティ設定を担当する (マネージド型とセルフマネージド型の両方の) Amazon EC2 ノードに適用されます。
- Amazon EKS の使用時に Kubernetes クラスターとノードを安全に設定できるように、コミュニティが承認した標準的な方法を提供します。
- これは、コントロールプレーンでのログ記録の設定、ノードのセキュリティ設定、ポリシー、マネージド型サービスの 4 つのセクションで構成されます。
- Amazon EKS で現在入手可能なすべての Kubernetes バージョンをサポートし、kube-bench (Kubernetes クラスターで CIS ベンチマークを使用して構成をチェックするための標準的なオープンソースのツール) により実行されます。

詳細については、「Introducing The CIS Amazon EKS Benchmark (CIS Amazon EKSベンチマークの紹介)」を参照してください。

CIS ベンチマークされた AMI でノードグループを更新する自動 `aws-sample` パイプラインについては、「EKS-Optimized AMI Hardening Pipeline」を参照してください。

Amazon EKS のプラットフォームバージョン

Amazon EKS プラットフォームのバージョンは、クラスターコントロールプレーンの機能を表しています。これには、Kubernetes API サーバーでどのフラグが有効であるかや、現在の Kubernetes パッチバージョンなどの情報が含まれます。新しいクラスターは、最新のプラットフォームバージョンでデプロイされます。詳細については、「EKS platform-versions」を参照してください。

Amazon EKS クラスターを更新し、新しい Kubernetes バージョンにすることができます。新しい Kubernetes バージョンが Amazon EKS で利用可能になったら、利用可能な最新のバージョンが使用できるように、クラスターをタイムリーに更新することをお勧めします。EKS での Kubernetes バージョンの詳細については、「Amazon EKS supported versions」を参照してください。

オペレーティングシステムの脆弱性リスト

AL2023 脆弱性リスト

Amazon Linux セキュリティセンターで Amazon Linux 2023 のセキュリティイベントまたはプライバシーイベントを追跡するか、関連する RSS フィードをサブスクライブします。セキュリティおよびプライバシーイベントには、影響を受ける問題の概要、パッケージ、および問題を修正するためにインスタンスを更新する手順などがあります。

Amazon Linux 2 脆弱性リスト

Amazon Linux セキュリティセンターで Amazon Linux 2 のセキュリティイベントまたはプライバシーイベントを追跡するか、関連する RSS フィードをサブスクライブします。セキュリティおよびプライバシーイベントには、影響を受ける問題の概要、パッケージ、および問題を修正するためにインスタンスを更新する手順などがあります。

Amazon Inspector によるノード検出

Amazon Inspector を使用すると、ノードからネットワークに意図しない接続が可能か、また、それらの Amazon EC2 インスタンスに脆弱性があるかを確認できます。

Amazon GuardDuty によるクラスターとノードの検出

Amazon GuardDuty は、AWS 環境内のアカウント、コンテナ、ワークロード、データを保護する脅威検知サービスです。GuardDuty には、EKS クラスターに対する潜在的な脅威を検出する EKS Protection とランタイムモニタリングの 2 つの機能があります。

詳細については、「the section called “Amazon GuardDuty”」を参照してください。

Amazon EKS クラスターのコンプライアンス検証

AWS のサービスが特定のコンプライアンスプログラムの対象であるかどうかを確認するには、「コンプライアンスプログラムによる対象範囲内の AWS サービス」をご覧いただき、関心のあるコンプライアンスプログラムを選択してください。一般的な情報については、「AWS コンプライアンスプログラム」を参照してください。

サードパーティーの監査報告書は、AWS アーティファクトを使用してダウンロードすることができます。詳細については、「AWS アーティファクトのレポートのダウンロード」を参照してください。

AWS のサービスの使用時におけるユーザーのコンプライアンス責任は、データの機密性、企業のコンプライアンス目的、適用法と規制に応じて異なります。AWS は、コンプライアンスに役立つ以下のリソースを提供しています。

- セキュリティのコンプライアンスとガバナンス – これらのソリューション実装ガイドでは、アーキテクチャ上の考慮事項について説明し、セキュリティとコンプライアンスの機能をデプロイする手順を示します。
- HIPAA 対応サービスのリファレンス – HIPAA 対応サービスの一覧が提供されています。すべての AWS のサービスが HIPAA 対象であるわけではありません。
- AWS コンプライアンスのリソース – このワークブックとガイドのコレクションは、お客様の業界や所在地に適用される場合があります。
- AWS カスタマーコンプライアンスガイド – コンプライアンスの観点から見た責任共有モデルを理解できます。このガイドは、AWS のサービスを保護するためのベストプラクティスを要約したものであり、複数のフレームワーク (米国標準技術研究所 (NIST)、ペイメントカード業界セキュリティ標準評議会 (PCI)、国際標準化機構 (ISO) など) にわたるセキュリティ統制へのガイダンスがまとめられています。
- AWS Config デベロッパーガイドの「ルールでのリソースの評価」 – AWS Config サービスは、リソースの設定が内部規定、業界のガイドライン、規制にどの程度適合しているかを評価します。
- AWS セキュリティハブ – この AWS サービスは、AWS 内のセキュリティ状態に関する包括的なビューを提供します。Security Hub では、セキュリティコントロールを使用して AWS リソースを評価し、セキュリティ業界標準とベストプラクティスに対するコンプライアンスをチェックします。サポートされているサービスとコントロールの一覧については、Security Hub のコントロールリファレンスを参照してください。
- Amazon GuardDuty – この AWS サービスは、環境をモニタリングして、疑わしいアクティビティや悪意のあるアクティビティがないか調べることで、AWS アカウント、ワークロード、コンテナ、データに対する潜在的な脅威を検出します。GuardDuty を使用すると、特定のコンプライアンスフレームワークで義務付けられている侵入検出要件を満たすことで、PCI DSS などのさまざまなコンプライアンス要件に対応できます。
- AWS Audit Manager – この AWS サービスでは、AWS の使用状況を継続的に監査し、リスクの管理方法と、規制や業界標準へのコンプライアンスの管理方法を簡素化できます。

アマゾン エラスティックKubernetesサービス のセキュリティに関する考慮事項

以下はアマゾン EKS に影響するため、クラウドのセキュリティに関する考慮事項です。

トピック

- Amazon EKS でのインフラストラクチャセキュリティ
- Amazon EKS クラスターの耐障害性の説明
- Amazon EKS におけるサービス間の混乱した代理の防止

Amazon EKS でのインフラストラクチャセキュリティ

マネージドサービスの Amazon Elastic Kubernetes Service は、AWS グローバルネットワークセキュリティによって保護されています。AWS セキュリティサービスと AWS がインフラストラクチャを保護する方法については「AWS クラウドセキュリティ」を参照してください。インフラストラクチャセキュリティのベストプラクティスを使用して AWS 環境を設計するには「セキュリティの柱 - AWS 適切なアーキテクチャを備えたフレームワーク」の「インフラストラクチャの保護」を参照してください。

AWS が公開している API コールを使用し、ネットワーク経由で Amazon EKS にアクセスします。クライアントは以下をサポートする必要があります。

- Transport Layer Security (TLS)。TLS 1.2 が必須で、TLS 1.3 をお勧めします。
- DHE (楕円ディフィー・ヘルマン鍵共有) や ECDHE (楕円曲線ディフィー・ヘルマン鍵共有) などの完全前方秘匿性 (PFS) による暗号スイート。これらのモードはJava 7 以降など、ほとんどの最新システムでサポートされています。

また、リクエストにはアクセスキー ID と、IAM プリンシパルに関連付けられているシークレットアクセスキーを使用して署名する必要があります。または、AWS Security Token Service (AWS STS) で一時的なセキュリティ認証情報を生成して、リクエストに署名することもできます。

Amazon EKS のクラスターの作成時には、使用するクラスターの VPC サブネットを指定します。Amazon EKS には、2 つ以上のアベイラビリティーゾーンにサブネットが必要です。パブリックサブネットとプライベートサブネットを持つ VPC をお勧めします。これにより、Kubernetes がパブリックサブネットにパブリックロードバランサーを作成し、プライベートサブネットにあるノードで実行されている Pod へのトラフィックを負荷分散します。

VPC に関する考慮事項の詳細については、「」を参照してくださいthe section called “VPC およびサブネットの要件”

チュートリアル「Amazon EKS の使用を開始する」で提供されている AWS CloudFormation テンプレートを使用して VPC およびノードグループを作成した場合、コントロールプレーンとノードのセキュリティグループは推奨設定で構成されます。

セキュリティグループの考慮事項の詳細については、「」を参照してくださいthe section called “セキュリティグループの要件”

新しいクラスターを作成すると、Amazon EKS によってマネージド型の Kubernetes API サーバーのエンドポイントが作成されます。ユーザーはこのエンドポイントを、(`kubectl` などの Kubernetes 管理ツールにより) クラスターとの通信に使用します。デフォルトでは、この API サーバーエンドポイントはインターネットに公開されます。API サーバーへのアクセスの保護には、AWS Identity and Access Management (IAM) と、ネイティブの Kubernetes ロールベースアクセスコントロール (RBAC) が組み合わせて使用されます。

Kubernetes API サーバーへのプライベートアクセスを有効にすると、ノードと API サーバー間のすべての通信が VPC 内で行われるようになります。インターネットから API サーバーにアクセスできる IP アドレスを制限したり、API サーバーへのインターネットアクセスを完全に無効にしたりできます。

クラスターエンドポイントアクセスの変更の詳細については、「」を参照してくださいthe section called “クラスターエンドポイントのアクセスの変更”

Amazon VPC CNI を使用するか、Project Calico などのサードパーティーツールを使用して、Kubernetes ネットワークポリシーを実装できます。Amazon VPC CNI をネットワークポリシーに使用する方法の詳細については、「the section called “Kubernetes ポリシー”」を参照してください。Project Calico はサードパーティのオープンソースプロジェクトです。詳細については、Project Calico のドキュメントを参照してください。

AWS プライベートリンクを使用して Amazon EKS にアクセスする

AWS プライベートリンク を使用して、VPC と Amazon Elastic Kubernetes Service 間にプライベート接続を作成できます。インターネットゲートウェイ、NAT デバイス、VPN 接続、AWS Direct Connect 接続のいずれかを使用せずに、VPC 内にあるかのように Amazon EKS にアクセスできます。VPC のインスタンスはパブリック IP アドレスがなくても Amazon EKS にアクセスできます。

AWS プライベートリンク が電源を供給するインターフェイスエンドポイントを作成することにより、このプライベート接続を確立します。インターフェイスエンドポイントに対して有効にする各サブネットにエンドポイントネットワークインターフェイスを作成します。これらはAmazon EKS 宛てのトラフィックのエントリポイントとして機能するリクエスタ管理型ネットワークインターフェイスです。

詳細については、AWS PrivateLink ガイドの「AWS PrivateLink を介した AWS サービスへのアクセス」を参照してください。

開始する前に

始める前に、次のタスクを実行していることを確認してください。

- AWS PrivateLink ガイドの「Access an AWS service using an interface VPC endpoint」を確認する

考慮事項

- サポートと制限: Amazon EKS のインターフェイスエンドポイントは、VPC からのすべての Amazon EKS API アクションへの安全なアクセスを可能にしますが、特定の制限があります。Kubernetes API へのアクセスはサポートされません。Kubernetes API は個別のプライベートエンドポイントがあるため、インターフェイスエンドポイントを介してのみアクセスできるように Amazon EKS を設定することはできません。サービスアカウント (IRSA) の IAM ロールで使用されるクラスター OIDC 検出と JWKS エンドポイントは、独自のサービス名とアクセスコントロール動作を持つ別のインターフェイスエンドポイントからも提供されます。the section called “AWS PrivateLink を使用してクラスター OIDC エンドポイントにアクセスする” を参照してください。
- 料金: Amazon EKS のインターフェイスエンドポイントを使用すると、標準の AWS PrivateLink 料金 (各アベイラビリティーゾーンでプロビジョニングされた各エンドポイントの時間単位の料金と、エンドポイントを経由するトラフィックのデータ処理料金) が発生します。詳細については、「AWS PrivateLink の料金」を参照してください。
- セキュリティとアクセスコントロール: セキュリティを強化し、アクセスを制御するために、以下の追加設定を推奨します。VPC エンドポイントポリシーを使用し、インターフェイスエンドポイントを介して Amazon EKS へのアクセスを制御し、エンドポイントネットワークインターフェイスにセキュリティグループを関連付けてトラフィックを管理し、VPC フローログを使用してインターフェイスエンドポイントを行き来する IP トラフィックをキャプチャおよびモニタリングし、ログを Amazon CloudWatch または Amazon S3 に発行できるようにします。詳細については、「Control access to VPC endpoints using endpoint policies」および「Logging IP traffic using VPC Flow Logs」を参照してください。
- 接続オプション: インターフェイスエンドポイントは、オンプレミスアクセス (AWS Direct Connect または AWS Site-to-Site VPN を使用してインターフェイスエンドポイントを持つ VPC にオンプレミスデータセンターを接続する) の使用または VPC 間接続 (AWS Transit Gateway または VPC ピアリングを使用して、インターフェイスエンドポイントを持つ VPC に他の VPC を接続し、トラフィックを AWS ネットワーク内に留める) を介して柔軟な接続オプションを実現します。

- IP バージョンのサポート: 2024 年 8 月より前に作成されたエンドポイントは、eks.region.amazonaws.com を使用した IPv4 のみをサポートします。2024 年 8 月以降に作成された新規エンドポイントは、デュアルスタック IPv4 および IPv6 (eks.region.amazonaws.com や eks.region.api.aws など) をサポートしています。
- リージョン別可用性: EKS API の AWS PrivateLink は、アジアパシフィック (マレーシア) (ap-southeast-5)、アジアパシフィック (タイ) (ap-southeast-7)、メキシコ (中部) (mx-central-1)、アジアパシフィック (台北) (ap-east-2) のリージョンでは使用できません。 AWSEKS-auth (EKS Pod Identity) の PrivateLink サポートは、アジアパシフィック (マレーシア) (ap-southeast-5) リージョンで利用できます。AWSクラスター OIDC エンドポイント (`com.amazonaws.region-code.oidc-eks`) の PrivateLink は、ほとんどの AWS リージョンで利用できます。

Amazon EKS 用のインターフェイスエンドポイントを作成します

Amazon VPC コンソールまたは AWS コマンドラインインターフェイス (AWS CLI) を使用して、Amazon EKS のインターフェイスエンドポイントを作成できます。詳細については、AWS PrivateLink ガイドの「VPC エンドポイントを作成する」を参照してください。

次のサービス名を使用して Amazon EKS のインターフェイスエンドポイントを作成します:

EKS API

- com.amazonaws.region-code.eks
- com.amazonaws.region-code.eks-fips (FIPS 準拠のエンドポイントの場合)

EKS Auth API (EKS Pod Identity)

- com.amazonaws.region-code.eks-auth

EKS クラスター OIDC エンドポイント (IRSA 検出/JWKS)

- com.amazonaws.region-code.oidc-eks

このエンドポイントには、EKS API エンドポイントとは異なるアクセスコントロール動作があります。詳細については、「the section called “AWS PrivateLink を使用してクラスター OIDC エンドポイントにアクセスする”」を参照してください。

EKS コンソールのリソースビュー

- com.amazonaws.region-code.eks-proxy

このエンドポイントサービスは、AWS コンソールのクラスターリソースビューを支えます。これは AWS が管理するコンソールおよびサービスによって使用されるものであり、お使いのアプリケーションから直接呼び出されることはありません (パブリック SDK や CLI はありません)。VPC にインターネットへの出力がない場合、またはプライベートホストゾーンで `eks.region-code.amazonaws.com` ゾーンを上書きする場合は、このエンドポイントを作成します。そのような場合は、コンソールのリソースビューが動作するためにこれが必要です。

このエンドポイントは、次の 2 つのプライベート DNS 名を登録します:

- `eks-proxy.region-code.api.aws` (デュアルスタック)
- `eks-proxy.eks.region-code.amazonaws.com` (IPv4。IPv4 のみのネットワークではこれを使用します)

Amazon EKS インターフェイスエンドポイントのプライベート DNS 機能

Amazon EKS およびその他の AWS サービスのインターフェイスエンドポイントに対してデフォルトで有効になっているプライベート DNS 機能は、デフォルトのリージョン DNS 名を使用して安全でプライベートな API リクエストを容易にします。この機能により、API コールがプライベート AWS ネットワーク経由でインターフェイスエンドポイントを通じてルーティングされ、セキュリティおよびパフォーマンスが向上します。

Amazon EKS または他の AWS サービスのインターフェイスエンドポイントを作成すると、プライベート DNS 機能は自動的にアクティブ化されます。有効にするには、特定の属性を設定して VPC を正しく設定する必要があります。

- enableDnsHostnames: VPC 内のインスタンスに DNS ホスト名を付けることができます。
- enableDnsSupport: VPC 全体で DNS 解決を有効にします。

これらの設定を確認または変更する手順については、「VPC の DNS 属性の表示と更新」を参照してください。

DNS 名と IP アドレスタイプ

プライベート DNS 機能を有効にすると、特定の DNS 名を使用して Amazon EKS に接続でき、これらのオプションは時間の経過とともに進化します。

- eks.region.amazonaws.com: 2024 年 8 月より前に IPv4 アドレスにのみ解決される従来の DNS 名。デュアルスタックに更新された既存のエンドポイントの場合、この名前は IPv4 アドレスおよび IPv6 アドレスの両方に解決されます。
- eks.region.api.aws: 2024 年 8 月以降に作成された新規エンドポイントに利用できるこのデュアルスタック DNS 名は、IPv4 アドレスおよび IPv6 アドレスの両方に解決されます。

2024 年 8 月以降、新規のインターフェイスエンドポイントには 2 つの DNS 名があり、デュアルスタック IP アドレスタイプを選択できます。既存のエンドポイントの場合、デュアルスタックに更新すると、IPv4 および IPv6 の両方をサポートするように eks.region.amazonaws.com が変更されます。

プライベート DNS 機能の使用

設定が完了したら、プライベート DNS 機能をワークフローに統合して次の機能が実現されます。

- API リクエスト: エンドポイントの設定に基づいて `eks.region.amazonaws.com` または `eks.region.api.aws` のいずれかのデフォルトリージョン DNS 名を使用し、Amazon EKS に API リクエストを行います。
- アプリケーションの互換性: EKS API を呼び出す既存のアプリケーションは、この機能を活用するために変更する必要はありません。
- AWS CLI とデュアルスタックの併用: デュアルスタックエンドポイントを AWS CLI と使用するには、「 AWS SDK とツールリファレンスガイド」の「デュアルスタックと FIPS エンドポイント」の設定を参照してください。
- 自動ルーティング: Amazon EKS のデフォルトサービスエンドポイントへの呼び出しはインターフェイスエンドポイントを介して自動的に転送されるため、プライベートで安全な接続が保証されます。

AWS PrivateLink を使用してクラスター OIDC エンドポイントにアクセスする

各 Amazon EKS クラスターは、OIDC 発行者 URL でパブリック OpenID Connect (OIDC) 検出ドキュメントと JSON ウェブキーセット (JWKS) を発行します。サービスアカウント (IRSA) の IAM ロールは、このエンドポイントを使用してクラスターのパブリック署名キーを発行します。IRSA を

セットアップしたり、トークンを検証したりするツールは、このエンドポイントからキーを直接取得します。

AWS PrivateLink を使用すると、インターネットへの出力なしで、VPC からこの OIDC エンドポイントにプライベートにアクセスできます。これは、クラスターの IAM OIDC ID プロバイダー (`eksctl`、Terraform、または AWS CLI) の作成や VPC 内での独自のトークン検証の実行など、VPC 内ツールが OIDC エンドポイントに到達する必要があるプライベート VPC または出力なし VPC に役立ちます。以前は、このエンドポイントに到達するにはインターネットアクセスまたは DNS 回避策が必要でした (GitHub の containers-roadmap#2038 を参照)。

以下のサービス名を使用して、クラスター OIDC エンドポイント用のインターフェイスエンドポイントを作成します:

- `com.amazonaws.region-code.oidc-eks`

考慮事項

- このエンドポイントは、VPC エンドポイントポリシーをサポートしていません。OIDC 検出ドキュメントと JWKS ドキュメントは匿名で提供されます。これは OpenID Connect 仕様で必須であり、IRSA の仕組みです。バリデータは認証情報を提示せずにパブリックキーを取得します。リクエストには IAM プリンシパルまたはアクションがないため、VPC エンドポイントポリシーには評価対象がありません。このエンドポイントは、デフォルトのフルアクセスポリシーのみを受け入れます。エンドポイントネットワークインターフェイスのセキュリティグループとサブネットルーティングを使用して、エンドポイントに到達できるユーザーを制御します。
- プライベート接続は認可境界ではありません。AWS PrivateLink 経由で OIDC エンドポイントに到達すると、このトラフィックは AWS ネットワーク上に保持されます。それ自体は、IAM ロールを引き受けることができるユーザーを制御するものではありません。提供されるデータはパブリックキーマテリアルです。引き受けたロールの信頼ポリシーは、OIDC エンドポイントに到達するために使用されるネットワークパスではなく、IRSA 認可を適用します。この適用は、ワークロードが `sts:AssumeRoleWithWebIdentity` を呼び出すときに発生します。ロールの信頼ポリシーの `aud` および `sub` 条件でロールを引き受けることができるワークロードを制御します。「Kubernetes サービスアカウントに IAM ロールを割り当てる」を参照してください。
- AWS STS は独自のパスでトークンを検証します。このエンドポイントは、AWS STS が IRSA トークンを検証する方法には影響しません。ワークロードが `AssumeRoleWithWebIdentity` を呼び出すと、STS は VPC やこのエンドポイントを通じてではなく、AWS 内からクラスターの JWKS を取得します。このエンドポイントを作成するか、VPC で OIDC DNS を上書きしても、トークンの検証は変更されません。また、ワークロードの `AssumeRoleWithWebIdentity` 呼び

出しをプライベートのままにするには、AWS STS インターフェイスエンドポイントを作成し、リージョンの STS エンドポイントを個別に設定します。「サービスアカウントの AWS セキュリティトークンサービスエンドポイントを設定する」を参照してください。

- 料金: 標準の AWS PrivateLink 料金が適用されます。「AWS PrivateLink の料金」を参照してください。

プライベート DNS 名とデュアルスタック名

各クラスターには、クラスターの IP ファミリーによって決定される OIDC 発行者のホスト名が 1 つあります。デフォルトである IPv4 クラスターは、`oidc.eks.region-code.amazonaws.com` を使用します。IPv6 クラスターは、IPv4 アドレスと IPv6 アドレスの両方に解決されるデュアルスタックホスト名 `oidc-eks.region-code.api.aws` を使用します。AWS 中国リージョンでは、デュアルスタックホスト名は `oidc-eks.region-code.api.amazonwebservices.com.cn` です。

インターフェイスエンドポイントは両方のホスト名を登録します。プライベート DNS が有効になっている場合 (デフォルト)、各ホスト名はインターフェイスエンドポイントに解決されます。したがって、クラスターの発行者 URL は、クラスターが `oidc.eks.region-code.amazonaws.com` ホスト名を使用するか、デュアルスタックホスト名を使用するかにかかわらず、エンドポイントに解決されます。トークンの発行者 URL に従って検出ドキュメントと JWKS を取得するクライアントは、いずれの場合もエンドポイントにプライベートに到達します。

Amazon EKS クラスターの耐障害性の説明

AWS のグローバルインフラストラクチャは、AWS リージョンとアベイラビリティーゾーンを中心として構築されています。AWSリージョンには、低レイテンシー、高いスループット、そして高度の冗長ネットワークで接続されている複数の物理的に独立し隔離されたアベイラビリティーゾーンがあります。アベイラビリティーゾーンでは、アベイラビリティーゾーン間で中断せずに、自動的にフェイルオーバーするアプリケーションとデータベースを設計および運用することができます。アベイラビリティーゾーンは、従来の単一または複数のデータセンターインフラストラクチャよりも可用性、耐障害性、および拡張性が優れています。

Amazon EKS は、複数の AWS アベイラビリティーゾーンで Kubernetes コントロールプレーンインスタンスを実行およびスケーリングして、高可用性を実現します。Amazon EKS は、負荷に基づいてコントロールプレーンインスタンスを自動的にスケーリングします。また、異常なコントロールプレーンインスタンスを自動的に検出して置換し、そのコントロールプレーンに自動的にパッチを適用します。バージョン更新を開始すると、Amazon EKS によってコントロールプレーンが自動的に更新され、更新中のコントロールプレーンの高可用性が維持されます。

このコントロールプレーンは、少なくとも 2 つの API サーバーインスタンスと、(1 つの AWS リージョンで 3 つのアベイラビリティーゾーンにまたがっている) 3 つの etcd インスタンスを含みます。Amazon EKS:

- コントロールプレーンインスタンスの負荷をアクティブに監視し、高パフォーマンスを確保するために、それらのインスタンスを自動的にスケーリングします。
- 異常なコントロールプレーンインスタンスを自動的に検出して置換します。それらのコントロールプレーンインスタンスは、必要に応じて AWS リージョン内のアベイラビリティーゾーンで再起動されます。
- 高可用性を維持するために、AWS リージョンのアーキテクチャを活用します。そのため、Amazon EKS では API サーバーエンドポイント向けの可用性の SLA をご利用いただけます。

AWS リージョンとアベイラビリティーゾーンの詳細については、「AWSグローバルインフラストラクチャ」を参照してください。

Amazon EKS におけるサービス間の混乱した代理の防止

混乱した代理問題とは、あるアクションを実行する許可を持たないエンティティが、より多くの特権を持つエンティティにアクションの実行を強制できることで生じるセキュリティ上の問題です。AWS では、サービス間でのなりすましによって、混乱した代理問題が発生する場合があります。サービス間でのなりすましは、1 つのサービス (呼び出し元サービス) が、別のサービス (呼び出し対象サービス) を呼び出すときに発生する可能性があります。呼び出し元サービスが操作され、それ自身のアクセス許可が使用されて、本来アクセス許可が付与されるべきではない方法により、別の顧客のリソースに影響を与える可能性があります。これを防ぐために、AWS には、アカウント内のリソースへのアクセス権が付与されたサービスプリンシパルですべてのサービスのデータを保護するために役立つツールが用意されています。

リソースポリシー内では aws:SourceArn および aws:SourceAccount のグローバル条件コンテキストキーを使用して、Amazon Elastic Kubernetes Service (Amazon EKS) が別のサービスに付与する、リソースへのアクセス許可を制限することをお勧めします。

aws:SourceArn

1 つのリソースだけをクロスサービスのアクセスに関連付ける場合は、aws:SourceArn を使用します。

aws:SourceAccount

アカウント内の任意のリソースをクロスサービスの使用に関連付ける場合は、aws:SourceAccount を使用します。

混乱した代理問題から保護するための最も効果的な方法は、リソースの完全な ARN を指定して aws:SourceArn グローバル条件コンテキストキーを使用することです。リソースの完全な ARN が不明な場合や、複数のリソースを指定する場合には、グローバルコンテキスト条件キー aws:SourceArn で、ARN の未知部分を示すためにワイルドカード文字 (*) を使用します。例えば、 arn:aws:<servicename>:*:<123456789012>:*。

aws:SourceArn の値に Amazon S3 バケット ARN などのアカウント ID が含まれていない場合は、両方の aws:SourceAccount と aws:SourceArn を使用して、アクセス許可を制限する必要があります。

Amazon EKS クラスターロールのクロスサービスにおける混乱した代理を防止する

Amazon EKS クラスター IAM ロールはクラスターごとに必要です。Amazon EKS によって管理される Kubernetes クラスターはこのロールを使用してノードを管理し、レガシークラウドプロバイダーはこのロールを使用して Elastic Load Balancing によるロードバランサーをサービス用に作成します。このいずれのクラスターアクションも同じアカウントにのみ影響するため、各クラスターロールをそのクラスターとアカウントに制限することをお勧めします。これは、最小特権の原則に従って AWS の推奨事項をアカウントに具体的に適用したものです。

ソースの ARN 形式

aws:SourceArn の値は、 arn:aws:eks:*region*:*account*:cluster/*cluster-name* という形式の EKS クラスターの ARN にする必要があります。例えば、 arn:aws:eks:us-west-2:123456789012:cluster/my-cluster です。

EKS クラスターロールの信頼ポリシーの形式

次の例では、Amazon EKS で aws:SourceArn と aws:SourceAccount のグローバル条件コンテキストキーを使用して、「混乱した代理」問題を回避する方法を示します。

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
```

```
      "Principal": {
        "Service": "eks.amazonaws.com"
      },
      "Action": "sts:AssumeRole",
      "Condition": {
        "ArnLike": {
          "aws:SourceArn": "arn:aws:eks:us-west-2:123456789012:cluster/my-cluster"
          },
        "StringEquals": {
            "aws:SourceAccount": "123456789012"
        }
      }
    }
  ]
}
```

Kubernetes のセキュリティに関する考慮事項

以下は、Amazon EKS クラスターの Kubernetes に影響する、クラウドのセキュリティに関する考慮事項です。Kubernetes のセキュリティコントロールとプラクティスの詳細については、Kubernetes ドキュメントの「Cloud Native Security and Kubernetes」を参照してください。

トピック

- Kubernetes 証明書でワークロードを保護する
- EKS クラスターの認証局 (CA) をローテーションする
- Amazon EKS 作成の RBAC ロールおよびユーザーの説明
- 既存のクラスターで KMS を使用して Kubernetes シークレットを暗号化する
- Amazon EKS ポッドで AWS 秘密マネジャー シークレットを使用する
- すべての Kubernetes API データを対象とするデフォルトのエンベロープ暗号化
- Amazon EKS における Kubernetes RBAC の強化

Kubernetes 証明書でワークロードを保護する

Kubernetes Certificates API により、X.509 認証情報のプロビジョニングを自動化します。この API には、Kubernetes API クライアントが X.509 証明書を認証機関 (CA) に要求し、取得するためのコマンドラインインターフェイスが備わっています。`CertificateSigningRequest` (CSR) リソースを使用して、示された署名者に証明書への署名を要求できます。要求は、署名前に承認または拒

否されます。Kubernetes では、組み込み型の署名者とカスタムの署名者の両方がサポートされており、動作が詳細に定義されています。この構成により、クライアントは自分の CSR に何が起こるかを予測できます。証明書の署名の詳細については、「signing requests」(リクエストへの署名) を参照してください。

組み込みの署名者の中には、kubernetes.io/legacy-unknown も含まれます。CSR リソースの v1beta1 API では、この未知のレガシー署名者を拒否していません。ただし、CSR の安定版 v1 APIでは、signerName を kubernetes.io/legacy-unknown に設定することはできません。

クラスターでの証明書の生成に Amazon EKS CA を使用する場合は、カスタム署名者を使用する必要があります。CSR v1 API バージョンを使用して新しい証明書を生成するには、既存のマニフェストと API クライアントを移行する必要があります。古い v1beta1 API で作成された既存の証明書の有効性は保たれ、その有効期限が切れるまで機能します。これには以下が含まれます。

- 信頼のディストリビューション: なし Kubernetes クラスターには、この署名者のための標準的な信頼またはディストリビューションはありません。
- 許可された対象: 任意
- 許可された x509 拡張機能: subjectAltName とキーの使用の拡張機能を受け入れ、他の拡張機能を破棄します
- 許可された鍵の使用法: ["key encipherment", "digital signature", "server auth"] (鍵暗号化、デジタル署名、サーバー認証) 以外の使用法を含めないでください。

> **Note**
>
> クライアント証明書署名はサポートされていません。

- 有効期限/証明書のライフタイム: 45 日 (デフォルトかつ最大の値)
- CA ビットの許可/不許可: 不許可

signerName を使用した CSR 生成の例

次の手順では、signerName: beta.eks.amazonaws.com/app-serving を使用した DNS 名 myserver.default.svc の配信用証明書の生成方法を示します。これは、実際の環境を構築するためのガイドとして使用できます。

1. openssl genrsa -out myserver.key 2048 コマンドを実行して、RSA プライベートキーを生成します。

```
openssl genrsa -out myserver.key 2048
```

2. 次のコマンドを実行して、証明書リクエストを生成します。

```
openssl req -new -key myserver.key -out myserver.csr -subj "/CN=myserver.default.svc"
```

3. CSR リクエストの base64 値を生成し、後の手順で使用するために変数に格納します。

```
base_64=$(cat myserver.csr | base64 -w 0 | tr -d "
")
```

4. 次のコマンドを実行して、mycsr.yaml という名前のファイルを作成します。次の例では、beta.eks.amazonaws.com/app-serving が signerName に使用されています。

```
cat >mycsr.yaml <<EOF
apiVersion: certificates.k8s.io/v1
kind: CertificateSigningRequest
metadata:
  name: myserver
spec:
  request: $base_64
  signerName: beta.eks.amazonaws.com/app-serving
  usages:
    - digital signature
    - key encipherment
    - server auth
EOF
```

5. CSR を送信します。

```
kubectl apply -f mycsr.yaml
```

6. 配信用証明書を承認します。

```
kubectl certificate approve myserver
```

7. 証明書が発行されたことを確認します。

```
kubectl get csr myserver
```

出力例は次のとおりです。

```
NAME       AGE     SIGNERNAME                                    REQUESTOR          CONDITION
myserver   3m20s   beta.eks.amazonaws.com/app-serving   kubernetes-admin
 Approved,Issued
```

8. 発行された証明書をエクスポートします。

```
kubectl get csr myserver -o jsonpath='{.status.certificate}'| base64 -d >
 myserver.crt
```

EKS クラスターの認証局 (CA) をローテーションする

公開鍵基盤 (PKI) では、認証局 (CA) はデジタル証明書を発行および署名する信頼されたエンティティです。これらの証明書はアイデンティティを確立し、TLS (Transport Layer Security) を使用してシステム間の暗号化通信を可能にします。クライアントがサーバーに接続すると、サーバーは CA によって署名された証明書を提示します。クライアントは、接続を続行する前に、信頼する CA に対してサーバーの証明書を検証します。

Amazon Elastic Kubernetes Service (Amazon EKS) では、クラスター作成時に各 EKS クラスター用の CA が作成されます。これはアップストリーム Kubernetes と同じモデルに従います。各 EKS クラスターには、API サーバーの証明書に署名する独自の CA があります。これにより、コントロールプレーンコンポーネント、ワーカーノード、およびクライアントが API サーバーに対して認証し、EKS クラスターへの暗号化接続を確立できるようになります。

このような認証局 (CA) には、定義された有効期間があります。CA ローテーションは、EKS クラスターの認証局が有効期限切れになる前に置き換え、クラスターが運用可能かつアクセス可能な状態を維持するようにするプロセスです。このプロセスを自動的に管理する組み込みのセーフガードがあります。CA ローテーションを自身で開始しない場合、既存の CA が有効期限切れになる前に後継 CA を自動的に追加して有効化し、クラスターが引き続き利用できるようにします。

CA ローテーション中に、後継 CA が EKS クラスターに追加されます。EKS は、後継 CA をすべての AWS マネージドコンポーネント (コントロールプレーン、EKS Auto Mode インスタンス、および AWS Fargate ノード) に自動的に配布します。お客様は、お客様が管理するワーカーノード (EKS Auto Mode 以外、Fargate 以外) および外部クライアント (kubeconfig ファイルや CI/CD パイプラインなど) を更新して、後継 CA が有効化される前に後継 CA を信頼させる責任があります。後継 CA が有効化されると、EKS クラスターは後継 CA で証明書に署名するように移行します。その後、既存の CA は廃止されます。

CA には有限の有効期間があるため、CA ローテーションはすべての EKS クラスターで必要です。有効期間は、クラスターの CA が作成された時期によって異なります。詳細については、よくある質問のセクションを参照してください。EKS のセーフガードにより、ローテーションのライフサイクル全体を通じてクラスター自体が利用可能な状態に保たれますが、ローテーションを成功させるには、後継 CA の有効化後も接続を維持できるように、お客様が管理するワーカーノードと外部クライアントを更新することも必要です。

以降のセクションで説明する API、コンソールエクスペリエンス、通知、およびステップバイステップのガイダンスは、このプロセスを通じてお客様を支援するように設計されています。責任の全範囲については、責任共有モデルのセクションで説明します。

CA ローテーションの仕組み

Amazon EKS の CA ローテーションは多段階プロセスです。お客様が操作するかどうかに関係なく、ローテーションのライフサイクル全体を通じて EKS クラスターの可用性を維持するための自動セーフガードを用意しています。すべてのコンポーネントが接続を維持できるローテーションを実現するには、以降のセクションで説明する手順を踏む必要があります。

ステージ 1: 後継 CA を追加する

後継 CA が EKS クラスターに追加されます。この時点から、EKS クラスターは既存の CA と後継 CA の両方を同時に信頼します。証明書は引き続き既存の CA によって発行されます。中断は発生しません。

クラスターがアクティブ状態であれば、AWS CLI、EKS API、コンソール、または AWS CloudFormation などの Infrastructure as Code (IaC) を使用して、いつでも自身で後継 CA を追加できます。追加しない場合、当社がお客様に代わって自動的に追加します。

AWS が EKS 内の AWS マネージドコンポーネントへの配布を完了するまで (ステージ 2)、後継 CA を有効化することはできません。この時点が、更新が必要なワーカーノードと外部クライアントの特定を開始するのに適した時期です。EKS クラスターの API サーバーに接続するすべてのシステムを特定するには時間がかかる場合があります。特に、複数のチーム、CI/CD パイプライン、監視ツールがある環境ではその傾向があります。このプロセスを早めに開始すると、独自のスケジュールで更新を調整する柔軟性が最も高まります。

ステージ 2: 後継 CA を配布する

後継 CA が追加されると、AWS は EKS クラスター内のマネージドコンポーネント (コントロールプレーン、EKS Auto Mode インスタンス、および AWS Fargate ノード) を更新して、両方の CA を認識および信頼するようにします。この進捗状況は、CA の配布ステータスで確認できます。配布が完了するまで、後継 CA を有効化することはできません。

当社がマネージドコンポーネントへの CA 配布を完了した後は、お客様には、後継 CA を信頼させるために、2 つのグループ (お客様が管理するワーカーノード (EKS Auto Mode 以外、Fargate 以外) と外部クライアント) を更新する責任があります。これにより、後継 CA の有効化後もそれらが API サーバーに確実に接続し続けられます。後継 CA の有効化前に一部のコンポーネントが更新されていない場合、残りの更新を完了する間、接続を復元するために CA ロールバックを利用できます。

ステージ 3: 後継 CA を有効化する

EKS 内のすべての AWS マネージドコンポーネントへの後継 CA の配布が完了した後、後継 CA を有効化できます。独自のスケジュールで後継 CA を有効化することをお勧めします。お客様が管理するワーカーノードと外部クライアントを検出および更新し、後継 CA を信頼させるのに十分な時間を確保してください。後継 CA を有効化した後、EKS クラスターは後継 CA によって署名された証明書を発行します。既存の CA は引き続き信頼されますが、署名には使用されなくなります。有効化後、限られた期間だけロールバックウィンドウが利用でき、必要に応じて既存の CA に戻すことができます。ロールバックについては、後のセクションで詳しく説明します。

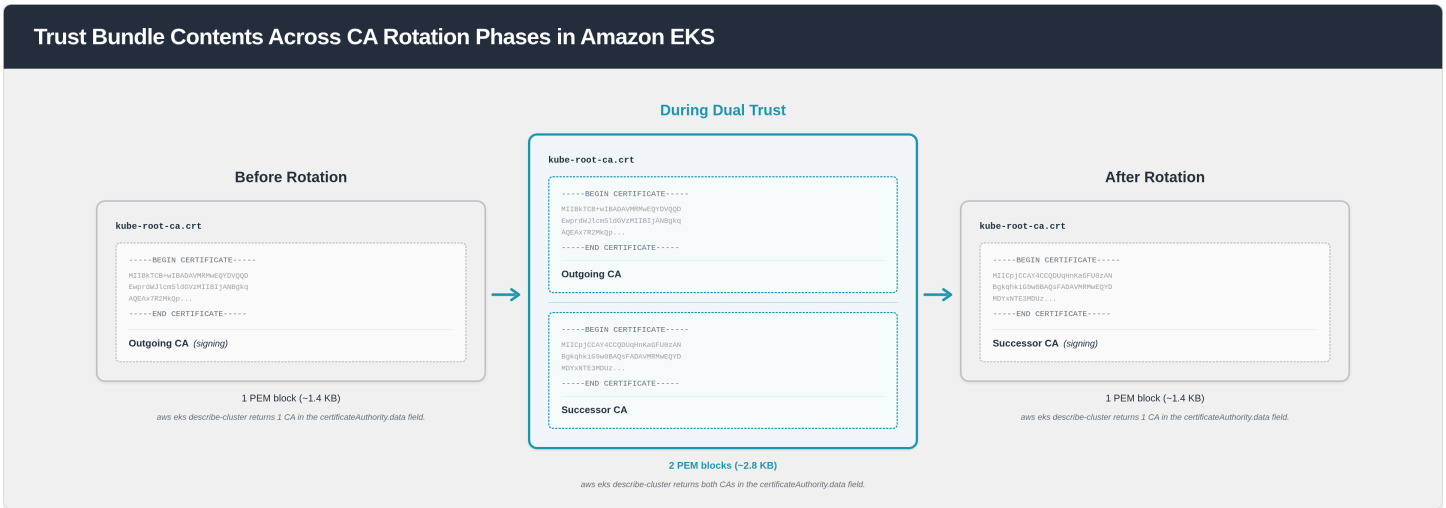
お客様が管理するワーカーノードとクライアントが更新されたことを確認できたら、自身で後継 CA を有効化できます。有効化しない場合、有効期限が近づくと当社が後継 CA を自動的に有効化します。

デュアルトラスト期間

後継 CA を追加してから (ステージ 1)、既存の CA を廃止するまでの期間は、デュアルトラスト期間と呼ばれます。この期間中、EKS クラスターは両方の CA を同時に信頼します。これにより、ローテーションが中断なしで行えるようになります。EKS クラスターはどちらかの CA によって署名された証明書をも受け入れるため、コンポーネントを段階的に更新できます。

デュアルトラスト期間により、すべての変更を同時に調整する必要なく、お客様が管理するすべてのワーカーノードと外部クライアントを特定および更新する時間を確保できます。

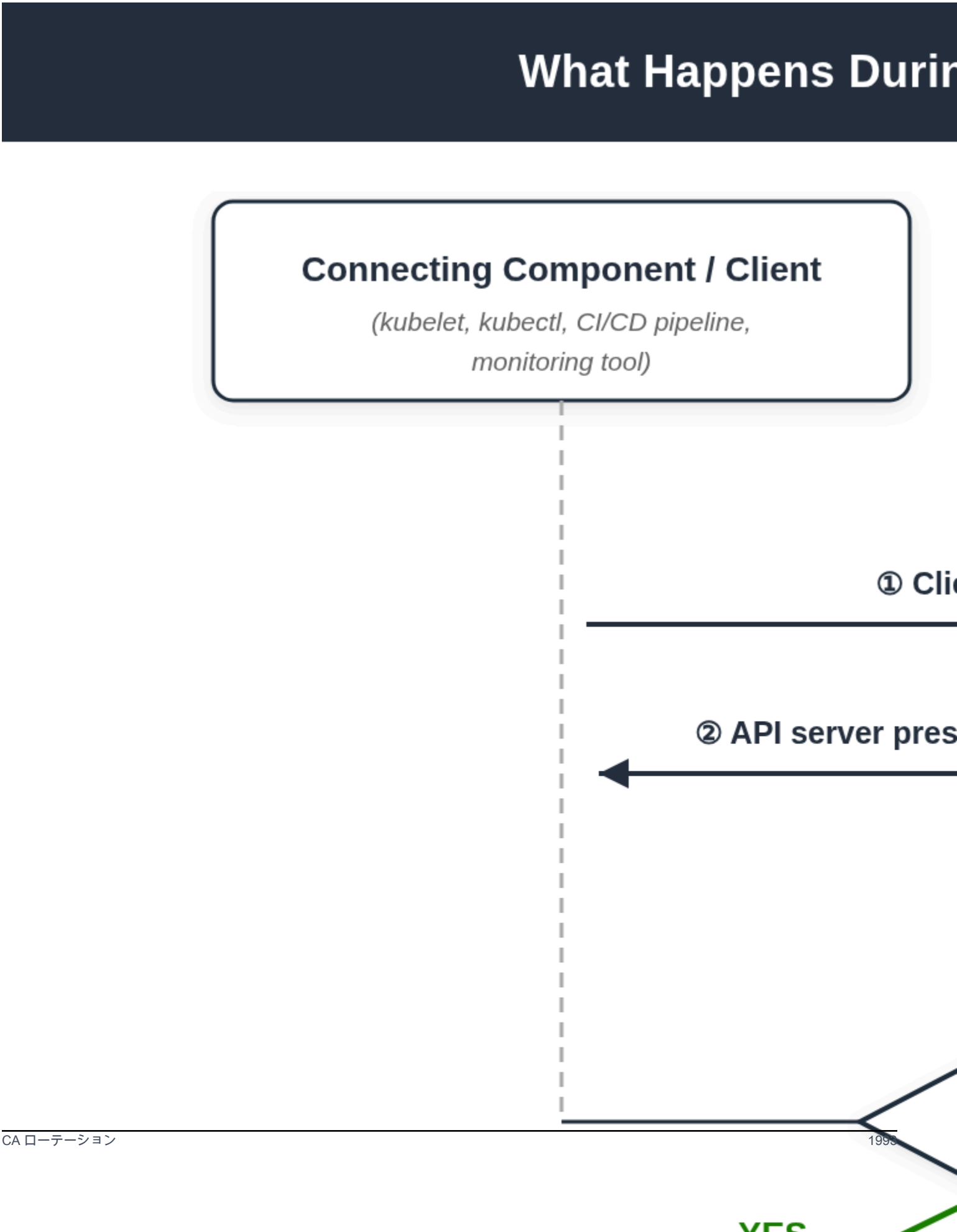
> **Note**
>
> デュアルトラスト期間中、クラスターの信頼バンドルには 2 つの CA が含まれます。これは .pem でエンコードされた信頼バンドルの標準的な動作です。ローテーションを開始する前に、厳密な単一 CA 検証または CA ピン留めを実行するアプリケーションを更新して、複数の CA を受け入れるようにしてください。



後継 CA の有効化が接続にどのように影響するかを理解することが重要です。クライアントが API サーバーに接続すると、サーバーの証明書が信頼する CA によって署名されていることを確認して、サーバーのアイデンティティを検証します。

後継 CA が有効化されると、API サーバーは後継 CA によって署名された証明書を提示します。信頼バンドルを更新して後継 CA を含めたクライアントは、検証に成功し、通常どおり接続できます。信頼バンドルを更新していないクライアントは、サーバーの証明書を認識できず、接続を確立できません。

次の図は、後継 CA の有効化後の TLS 接続フローを示しています。


What Happens Durin
Connecting Component / Client
(kubelet, kubectl, CI/CD pipeline,
monitoring tool)
① Cli
② API server pres

後継 CA の有効化前にワーカーノードと外部クライアントを更新することが重要なのは、API サーバーのアイデンティティを検証して接続するために、信頼バンドルに後継 CA が必要だからです。デュアルトラスト期間と CA ロールバックは、これを完了するための時間とセーフティネットを提供します。

Amazon EKS CA Rotation

Stage 1 — Append CA

WHAT HAPPENS

Successor CA added to the cluster. Cluster now tru
CAs. Certificates still issued by outgoing CA.

WHO ACTS

You or AWS (auto-append at T−2 years before CA

責任共有モデル

Amazon EKS の CA ローテーションは、AWS 全体に広く適用される同じ責任共有モデルに従います。AWS はクラウドインフラストラクチャのセキュリティと可用性に責任を負い、お客様はその中のワークロードのセキュリティと設定に責任を負います。責任共有が Amazon EKS にどのように適用されるかの詳細については、「EKS のセキュリティのベストプラクティス」を参照してください。

CA ローテーションのコンテキストでは、これは次のことを意味します。

AWS の責任:

- EKS クラスターのコントロールプレーンを更新して、後継 CA を信頼し、後継 CA から証明書を発行すること
- EKS Auto Mode ノードを更新して後継 CA を信頼させること
- AWS Fargate ノードを更新して後継 CA を信頼させること
- EKS 内の AWS マネージドコンポーネントへの配布が完了するまで後継 CA を有効化できないようにすること
- ローテーションのライフサイクル全体を通じて EKS クラスターの可用性を維持すること
- ローテーションプロセスの各段階でお客様に通知すること
- CA の有効期限が近づく前に操作を行わなかった場合にローテーションを自動的に開始すること

お客様の責任:

- 外部クライアント (デベロッパーワークステーション、CI/CD パイプライン、監視ツール、オートメーション) を更新して後継 CA を信頼させること
- ワーカーノード (マネージド型ノードグループ、Karpenter 制御ノード、セルフマネージド型ノード、ハイブリッドノード) を更新して後継 CA を信頼させること
- コンポーネントが更新されたことを確認できたら後継 CA を有効化すること

当社はこれらのアクションをお客様に代わって実行することはできません。外部クライアントは、AWS の運用境界の外に存在します。EKS Auto Mode または Fargate によって管理されていないワーカーノードは、起動時またはお客様のみが制御するブートストラッププロセスを通じて CA 信頼設定が設定されます。これは TLS 信頼の仕組みと一致しています。クライアントは独自のトラストストアを保持し、クライアントの管理者のみがそれを更新できます。

以降のセクションでは、それぞれの側面について詳しく説明します。AWS がお客様のために行うこと、およびお客様が行う必要があることと、その実行方法に関するステップバイステップのガイダンスを扱います。


Amazon EKS CA Rotation
AWS manages
EKS cluster control plane
EKS Auto Mode instances
AWS Fargate nodes
CA distribution to managed con
Blocking activation until distrib
Preserving cluster availability th

AWS がお客様のために行うこと

AWS は、EKS クラスターの CA ローテーションのライフサイクル全体を通じて以下を管理します。

CA の自動作成

独自のスケジュールで後継 CA を追加しない場合、EKS クラスターの既存の CA の有効期限が近づくと、当社が自動的に追加します。これにより、有効期限が切れる前にクライアントを検出して更新するための十分な時間を確保して、ローテーションプロセスを開始できます。

コントロールプレーンの更新

当社は EKS クラスターのコントロールプレーンを自動的に更新して、後継 CA を信頼するようにします。後継 CA の有効化後、コントロールプレーンは後継 CA によって署名された証明書を発行します。コントロールプレーンに対してお客様の操作は必要ありません。

EKS Auto Mode と Fargate の更新

当社は EKS Auto Mode ノードと Fargate ポッドを自動的に更新して、後継 CA を信頼するようにします。これらのコンポーネントは当社によって完全に管理されており、CA ローテーション中にお客様の操作は必要ありません。

EKS 機能の更新

当社は EKS 機能 (AWS Controllers for Kubernetes (ACK)、Argo CD、および kro (Kube Resource Orchestrator)) を自動的に更新して、後継 CA を信頼するようにします。これらのマネージドリソースはクラスターの API サーバーと通信し、CA 配布プロセスの一環として更新されます。EKS 機能のマネージドリソースに対してお客様の操作は必要ありません。詳細については、「EKS 機能」を参照してください。

配布ステータスの追跡

当社が EKS クラスター内のマネージドコンポーネントを更新する際、CA の配布ステータスを通じて進捗状況を監視できます。これにより、当社側のローテーションが完了したかどうかを確認できます。配布が完了するまで、後継 CA を有効化することはできません。

組み込みのセーフガード

ローテーション中に EKS クラスターを保護するための組み込みのセーフガードがあります。

- EKS 内のすべての AWS マネージドコンポーネントへの配布が完了するまで後継 CA を有効化できません

- AWS が追加した後継 CA は、クラスター上の唯一の後継 CA である間は削除できません。このセーフガードにより、クラスターに常に有効な CA パスが存在し、有効期限切れを確実に防ぐことができます。後継 CA が有効化された後は、既存の CA を削除できます。
- お客様が追加した後継 CA は、CA の有効期限の 2 年前に達すると削除できなくなります。この時点以降、有効期限が近づくにつれてクラスターに常に後継 CA が存在するようにすることで、CA は削除から保護されます。
- 有効期限が近づき、お客様が自身で後継 CA を有効化していない場合、当社が後継 CA を自動的に有効化します

このようなセーフガードにより、お客様が操作するかどうかに関係なく、ローテーションのライフサイクル全体を通じて EKS クラスターの可用性が維持されます。

通知

CA ローテーションのライフサイクルの各段階でお客様に通知します。通知は AWS Health、Cluster Insights、および E メールを通じて配信されます。各通知では、何が発生したか、お客様に必要なアクション (ある場合)、および EKS クラスターがローテーションタイムラインのどこにあるかが示されます。

通知	メトリクス	意味
CA 有効期限リマインダー	CA の有効期限の 2.5 年前	EKS クラスターの CA には、定義された有効期限があります。ローテーションを計画してください。
後継 CA が追加されました	お客様または AWS が後継 CA を追加したとき (有効期限の 2 年前に自動追加)	ローテーションプロセスが開始されました。AWS が後継 CA をマネージドコンポーネントに配布しています。
配布完了	追加直後 (クラスターによって異なります)	AWS 側が完了しました。これで、お客様が管理するワーカーノードと外部クライアントを更新できます。
有効化の警告	自動有効化の 60 日前	まもなく後継 CA を有効化します。まだ更新していない場

通知	メトリクス	意味
		合は、コンポーネントを更新してください。
後継 CA が有効化されました	お客様または AWS が有効化したとき (有効期限の 6 か月前に自動有効化)	EKS クラスターは、後継 CA から証明書を発行するようになりました。
最終自動有効化 (ロールバックした場合)	有効期限切れの 45 日前	AWS が後継 CA を有効化します。CA のロールバックは利用できません。

> **Note**
>
> 2018 ~ 2019 年に作成されたクラスターには、異なる通知タイムラインがあります。これらのクラスターには、調整されたスケジュールで自動通知が送信されます。

Amazon EventBridge を使用して独自の通知を設定し、CA ローテーションイベントを既存のモニタリングおよびアラートワークフローに統合することもできます。

お客様が行う必要があること (およびその理由)

CA ローテーションを成功させるには、AWS がお客様に代わってアクセスできないコンポーネントを更新する必要があります。これらは 2 つのカテゴリに分かれます。

外部クライアント

クラスターの外部から EKS クラスターの API サーバーに接続するあらゆるシステム。これには、デベロッパーワークステーション、CI/CD パイプライン (Jenkins、GitHub Actions、GitLab、Argo CD)、モニタリングおよびオブザーバビリティツール、オートメーションスクリプト、および kubeconfig を使用して API サーバーと通信するあらゆるアプリケーションが含まれます。

これらのシステムはそれぞれ独自の信頼設定を維持します。後継 CA が有効化されると、API サーバーは後継 CA によって署名された証明書を提示します。後継 CA の有効化前に、これらのクライアントが後継 CA を信頼するように更新することで、接続性が確実に維持されます。クライアントの更新が漏れた場合、CA のロールバックにより、更新が完了するまでアクセスを復元できます。

ワーカーノード (EKS Auto Mode 以外、Fargate 以外)

EKS Auto Mode または Fargate によって管理されていないワーカーノードは、起動時または kubelet ブートストラッププロセスを通じて CA 信頼設定が行われます。これらのノードは、後継 CA を信頼するようにリフレッシュする必要があります。必要なアクションは、ワーカーノードのタイプによって異なります。

マネージドノードグループ

ノードグループのバージョン更新を実行すると、ノードのローリング置き換えがトリガーされます。新しいノードは、更新された CA 信頼設定で自動的にブートストラップされます。

Karpenter 制御ノード

ドリフト検出が有効になっている場合、Karpenter は設定されたドリフトウィンドウ内でノードをサイクルし、新しいノードは手動操作なしで後継 CA を取得します。ドリフト検出が無効になっているか、長いウィンドウに設定されている場合は、これらのノードをセルフマネージド型ノードと同じように扱います。

セルフマネージド型ノード

更新された CA の信頼設定でブートストラップされるように、ノードを置き換えます。通常、これには、更新された CA データで起動テンプレートを更新し、Auto Scaling グループを通じてローリング置き換えをトリガーすることが含まれます。

ハイブリッドノード

各ハイブリッドノードの信頼設定を更新して、後継 CA を含めます。具体的なプロセスは、ハイブリッドノードがどのようにブートストラップされたか、およびその信頼設定がどのように管理されているかによって異なります。

クラスターインサイトを使用して、EKS クラスターで実行されているワーカーノードのタイプを特定できます。各タイプを更新するためのステップバイステップの手順は、後のセクションで説明します。

ワーカーノードの更新が漏れた場合のセーフティネットとして、CA ロールバックを提供しています。ただし、後継 CA の有効化前にすべてのワーカーノードを更新すると、接続の中断を完全に回避できます。後継 CA が有効化される前に更新されていないノードは、置き換えられるか CA ロールバックが実行されるまで、EKS クラスターの API サーバーへの接続を失います。

お客様だけがこれを実行できる理由

外部クライアントは、AWS の運用境界の外に存在します。企業ネットワークで実行されている CI/CD パイプライン、デベロッパーのラップトップ、オンプレミスでホストされているモニタリングツール: AWS にはこれらのシステムにアクセスして信頼設定を更新するメカニズムがありません。

EKS Auto Mode 以外のワーカーノードは、お客様が所有する起動テンプレート、ユーザーデータスクリプト、またはブートストラッププロセスを通じて信頼設定を制御できます。それらを更新するには、ノードを置き換えるか、設定を変更する必要があり、どちらもお客様のインフラストラクチャ内でのアクションです。

これは、現在 Kubernetes で使用されている TLS 信頼モデルの制約です。クライアントが信頼バンドルを更新したかどうかを API サーバーが照会するためのプロトコルレベルのメカニズムはありません。サーバーは、クライアントが接続したときにのみ証明書を提示できます。クライアントが、証明書に署名した CA を信頼している場合、接続は成功します。信頼していない場合、失敗します。AWS がお客様に代わってコンポーネントの準備状況を確認できる、有効化前の検証パスはありません。

いつ開始するか

できるだけ早く外部クライアントの特定を開始してください。これは CA ローテーションの中で最も時間のかかる部分であり、特に複数のチームが EKS クラスターに独立してワークロードをデプロイする環境ではそうです。クライアントの検出を早く開始するほど、プレッシャーなしでチーム間の更新を調整する時間が増えます。

各タイプのクライアントとワーカーノードを更新する方法に関する詳細なガイダンスは、以降のセクションで説明します。

前提条件

CA ローテーションを開始する前に、以下を確認してください。

- AWS CLI: バージョン 2.x 以降。CA ローテーション API は、最新の AWS CLI で利用できます。確認するには、`aws --version` を実行します。
- コンソールアクセス: CA ローテーションは、サポートされているリージョンの Amazon EKS コンソールで利用できます。
- リージョンの可用性: CA ローテーションは、Amazon EKS がサポートされているすべての AWS 商用リージョンで利用できます。

EKS クラスターの管理に必要な IAM アクセス許可以外に、追加の IAM アクセス許可は必要ありません。現在 EKS クラスターの EKS API を呼び出すことができる場合は、CA ローテーションを実行できます。

開始方法

AWS CLI または Amazon EKS コンソールを使用して CA ローテーションを実行できます。AWS CLI バージョン 2.x 以降を実行していることを確認してください (`aws --version` で確認)。

AWS CLI の使用

以下のウォークスルーでは、AWS CLI と EKS API を使用したエンドツーエンドの CA ローテーションプロセスについて説明します。

ステップ 1: アクティブな CA を確認する

EKS クラスターのアクティブな認証局を表示します。

```
aws eks list-certificate-authorities --cluster-name my-cluster --region us-west-2
```

正常な出力:

```
{
    "certificateAuthorities": [
        {
            "id": "a1b2c3d4-5678-90ab-cdef-example11111",
            "createdAt": "2024-01-15T10:30:00-07:00",
            "createdBy": "EKS",
            "activatedAt": "2024-01-15T10:30:00-07:00",
            "activatedBy": "EKS",
            "signingStatus": "IN_USE",
            "distributionStatus": "COMPLETE"
        }
    ]
}
```

これは、EKS クラスターの作成時に作成された CA を示します。現在使用中 (証明書に署名) であり、配布が完了しています (EKS 内のすべての AWS 管理コンポーネントがこれを信頼しています)。

ステップ 2: CA の詳細と有効期限を表示する

```
aws eks describe-certificate-authority --cluster-name my-cluster --certificate-
authority-id a1b2c3d4-5678-90ab-cdef-example11111 --region us-west-2
```

正常な出力:

```
{
    "certificateAuthority": {
        "id": "a1b2c3d4-5678-90ab-cdef-example11111",
        "createdAt": "2024-01-15T10:30:00-07:00",
        "createdBy": "EKS",
        "activatedAt": "2024-01-15T10:30:00-07:00",
        "activatedBy": "EKS",
        "signingStatus": "IN_USE",
        "distributionStatus": "COMPLETE",
        "validity": {
            "notBefore": "2024-01-15T10:30:00-07:00",
            "notAfter": "2029-01-14T10:30:00-07:00"
        },
        "rollbackAvailable": false
    }
}
```

validity ブロックには、CA が作成された日時 (notBefore) と有効期限 (notAfter) が表示されます。rollbackAvailable は、後継 CA の有効化後に以前の CA に戻すことができるかどうかを示します。EKS クラスターで作成された初期 CA の場合、戻す先の以前の CA がないため、これは false になります。

> **Note**
>
> scheduledEvents ブロック (firstAutoActivation と finalAutoActivation を含む) は、旧 CA ではなく後継 CA に表示されます。これらのフィールドは、お客様自身が有効化しなかった場合に、後継 CA を自動的に有効化する時期を示します。後継 CA を追加した後に後継 CA に対して describe を実行すると、これらのフィールドが表示されます。

ステップ 3: 後継 CA を追加する

```
aws eks create-certificate-authority --cluster-name my-cluster --region us-west-2
```

これにより、EKS クラスターに後継 CA が追加されます。応答には、進行状況を追跡するために使用できる updateId が含まれます。

ステップ 4: 更新を追跡する

```
aws eks describe-update --name my-cluster --update-id a1b2c3d4-update-id --region us-west-2
```

更新ステータスが Successful になるまで待機します。

> **Note**
>
> 更新ステータスが UPDATE_FAILED を示し、後継 CA の distributionStatus が FAILED を示す場合、CA の作成は成功していません。aws eks delete-certificate-authority を使用して失敗した CA を削除し、新しい CA を作成します。AWS が開始した自動ローテーションの場合、AWS は新しい後継を追加する前に失敗した CA を自動的に検出してクリーンアップするため、そのシナリオではお客様のアクションは必要ありません。

ステップ 5: 配布ステータスを確認する

```
aws eks list-certificate-authorities --cluster-name my-cluster --region us-west-2
```

これで 2 つの CA が表示されるはずです。後継 CA には signingStatus: NOT_USED があり、AWS が EKS クラスター内のすべての管理コンポーネントを更新した後、distributionStatus は IN_PROGRESS から COMPLETE に進行します。

後継 CA の配布ステータスが COMPLETE になるまで先に進まないでください。

ステップ 6: kubeconfig を更新する

```
aws eks update-kubeconfig --name my-cluster --region us-west-2
```

これにより、ローカルの kubeconfig が両方の CA を信頼するように更新されます。この後、後継 CA が有効化された後も、kubectl コマンドは引き続き機能します。

ステップ 7: お客様が管理するワーカーノードと外部クライアントを更新する

これについては、次のセクションで詳しく説明します。お客様が管理するすべてのワーカーノードと外部クライアントが後継 CA を信頼するように更新されたら、有効化に進みます。

ステップ 8: 後継 CA を有効化する

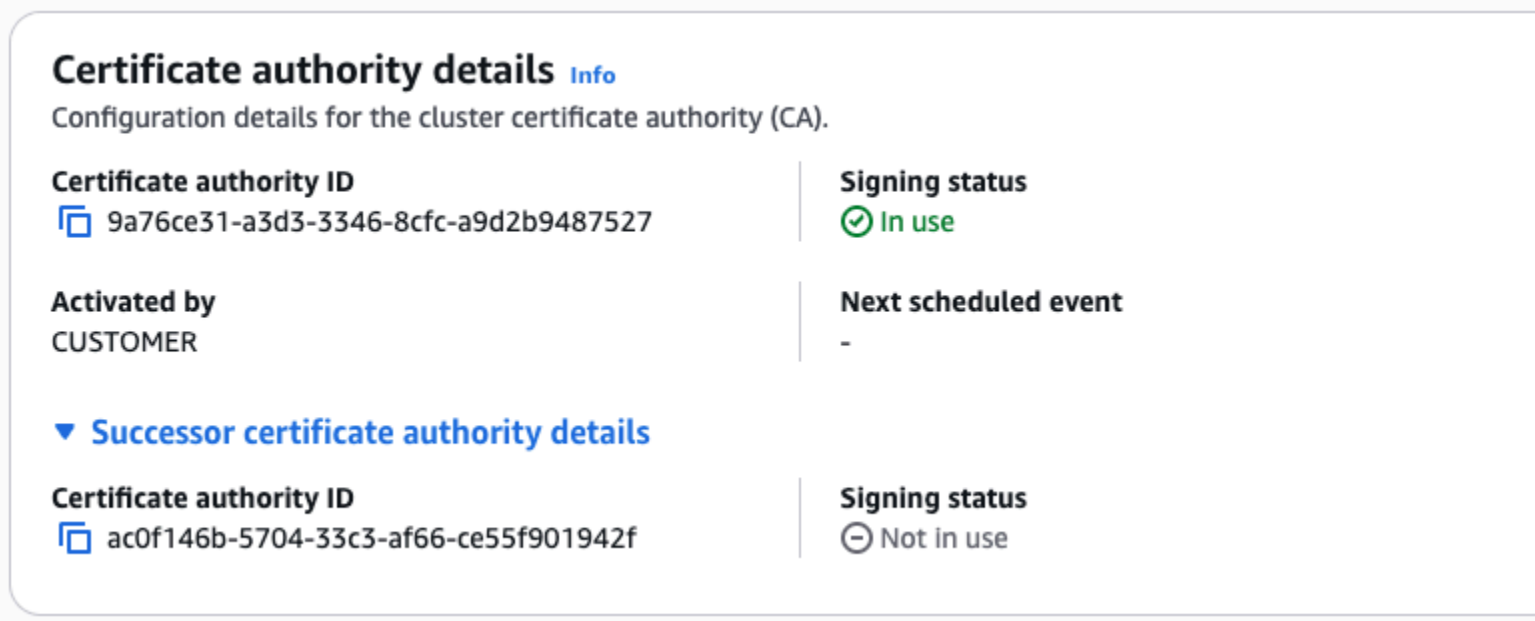
```
aws eks activate-certificate-authority --cluster-name my-cluster --certificate-
authority-id <successor-ca-id> --region us-west-2
```

後継 CA を有効化した後、EKS クラスターは後継 CA によって署名された証明書を発行します。接続を確認して、すべてのコンポーネントが期待どおりに動作していることを確認します。

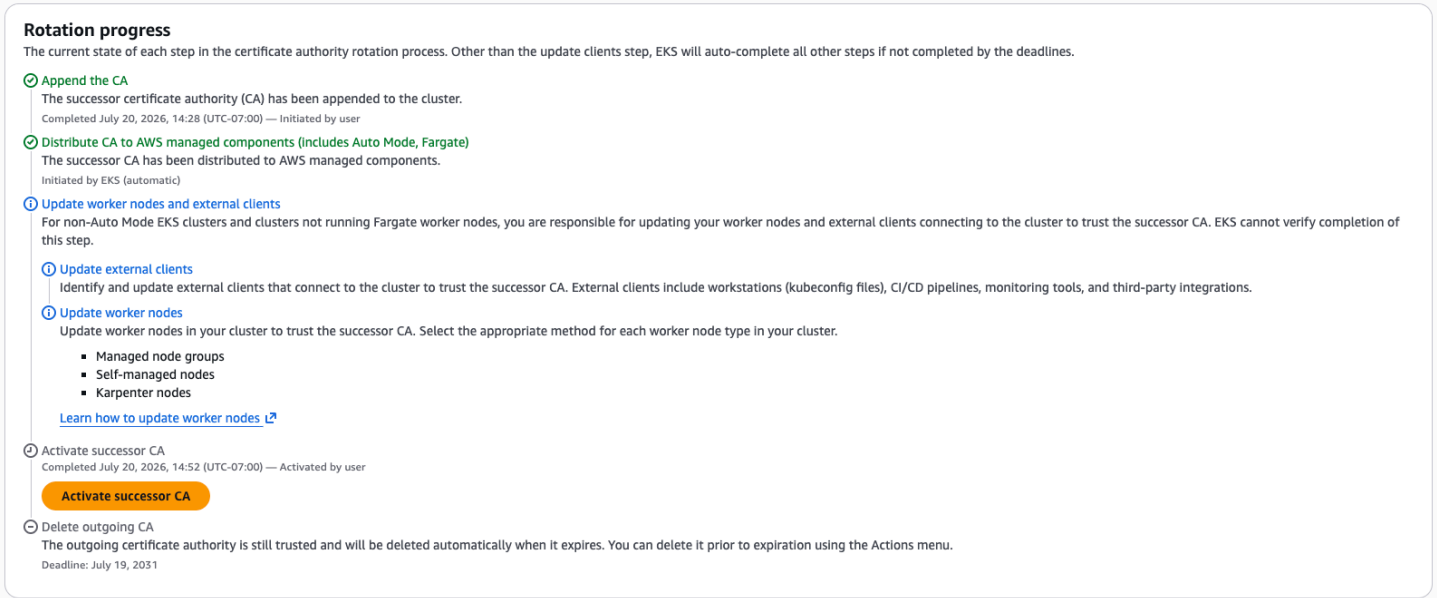
Amazon EKS コンソールの使用

Amazon EKS コンソールは、CA ローテーションのガイド付きエクスペリエンスを提供します。コンソールから直接、CA ステータスの表示、後継 CA の追加、配布の進行状況の監視、後継 CA の有効化を行うことができます。

次の図は、アクティブなローテーション中の Amazon EKS コンソールの認証局の詳細ビューを示しています。アクティブな CA と後継 CA の両方が、署名ステータス、有効期限、および有効期限までの日数とともに表示されます。



次の図は、Amazon EKS コンソールのローテーション進行状況ビューを示しています。ローテーションプロセスの各ステップは、現在の状態とともに表示され、これには追加、配布、ワーカーノードと外部クライアントの更新、有効化、および旧 CA の削除が含まれます。



Kubernetes クライアントを更新する

> **Important**
>
> デュアルトラスト期間中、クラスターの信頼バンドルには 2 つの CA 証明書が含まれます。2 つの base64 エンコードされた CA の合計サイズは約 2.8 KB (gzip 圧縮では約 1.9 KB) です。カスタムユーザーデータが EC2 起動テンプレートで提供されるワーカーノードの場合、合計ユーザーデータサイズが EC2 ユーザーデータの制限である 16KB を超えていないことを確認します。既存のユーザーデータがこの制限に近い場合は、gzip を使用してユーザーデータコンテンツを圧縮し、サイズを削減することを検討してください。

後継 CA が追加され、AWS が EKS クラスター内の管理コンポーネントへの配布を完了した後 (`distributionStatus: COMPLETE`)、後継 CA を信頼するように独自のコンポーネントを更新する必要があります。このコンテキストでの「クライアント」とは、EKS クラスターの API サーバーに接続するあらゆるシステムです。これには、ローカルの kubectl 設定、CI/CD パイプライン、モニタリングツール、オートメーションスクリプト、およびワーカーノードが含まれます。

EKS クラスターの更新された CA データ (現在の CA と後継 CA の両方が含まれるようになりました) は、次を使用して取得できます:

```
aws eks describe-cluster --name my-cluster --region us-west-2 --query
 'cluster.certificateAuthority.data' --output text
```

この値を使用して、以下のサブセクションで各クライアントタイプの信頼設定を更新します。

Kubeconfig (デベロッパーワークステーション、CI/CD パイプライン、オートメーション)

次を実行して、ローカルの kubeconfig を更新します:

```
aws eks update-kubeconfig --name my-cluster --region us-west-2
```

これにより、最新の CA データが自動的に取得され、kubeconfig が更新されます。この kubeconfig を使用するすべてのシステムは、両方の CA を信頼します。

独自の kubeconfig を生成する CI/CD パイプラインおよびオートメーションの場合 (例えば、EKS API を直接使用するか、kubeconfig をシークレットとして保存する場合)、`describe-cluster` から取得した値で `certificate-authority-data` フィールドを更新します。

マネージドノードグループ

ノードグループのバージョン更新を実行して、ノードのローリング置き換えをトリガーします:

```
aws eks update-nodegroup-version --cluster-name my-cluster --nodegroup-name my-
nodegroup --region us-west-2
```

新しいノードは、更新された CA データで自動的にブートストラップされます。ローリング置き換えにより、実行中のワークロードを中断することなく、ノードが一度に 1 つずつ置き換えられます。

Terraform または CloudFormation を通じてノードグループを管理している場合、CA ローテーションは IaC 状態にドリフトを発生させません。CA ライフサイクルは、クラスターリソース設定とは別の専用の EKS API を通じて管理されます。CA ローテーションが Infrastructure as Code とどのように相互作用するかの詳細については、「Infrastructure as Code」のセクションを参照してください。

カスタム AMI を使用したカスタム起動テンプレート

ノードグループがカスタム AMI でデプロイされている場合、AWS はユーザーデータをマージしません。更新された CA 信頼バンドルを含む、正しいブートストラップ設定を行う責任はお客様にあります。CA ローテーションはお客様のユーザーデータを自動的に更新せず、後継 CA がないノードはクラスターに参加できません。

1. 更新された CA データ (旧 CA と後継 CA の両方を含む結合信頼バンドル) を取得します:

```
aws eks describe-cluster --name my-cluster --region us-west-2 --query
 'cluster.certificateAuthority.data' --output text
```

2. 起動テンプレートのユーザーデータ内の CA データを更新します。CA データを指定する方法は、お使いのオペレーティングシステムとブートストラップメカニズムによって異なり、最初に指定した方法と一致します。マネージドノードのカスタマイズの詳細については、「起動テンプレートを使用してマネージドノードをカスタマイズする」を参照してください。
3. 更新されたユーザーデータで起動テンプレートの新しいバージョンを作成し、ノードグループをその起動テンプレートバージョンに更新します。これによりノードがリサイクルされ、次のように後継 CA でブートストラップされます:

```
aws eks update-nodegroup-version --cluster-name my-cluster --nodegroup-name my-
nodegroup --launch-template id=<lt-id>,version=<new-version> --region us-west-2
```

ノードグループを新しい起動テンプレートバージョンに更新する方法の詳細については、「クラスターのマネージドノードグループを更新する」を参照してください。

ノードが更新後の起動テンプレートで実行されていることを確認する

有効化に進む前に、マネージドノードグループ内のすべてのノードが最新の起動テンプレートバージョンで実行されていることを確認してください。このバージョンには、更新された CA 信頼バンドルが含まれている必要があります。

1. ノードグループの起動テンプレートを取得します。`describe-nodegroup` が `launchTemplate` フィールドを返す場合は、それを直接使用します:

```
aws eks describe-nodegroup --cluster-name my-cluster --nodegroup-name my-nodegroup --
region us-west-2 --query 'nodegroup.launchTemplate'
```

`launchTemplate` フィールドが返されない場合、AWS は起動テンプレートを内部で管理します。代わりに Auto Scaling グループを通じて見つけます:

```
ASG=$(aws eks describe-nodegroup --cluster-name my-cluster --nodegroup-name my-
nodegroup --region us-west-2 --query 'nodegroup.resources.autoScalingGroups[0].name'
 --output text)
aws autoscaling describe-auto-scaling-groups --auto-scaling-group-
names "$ASG" --region us-west-2 --query 'AutoScalingGroups[0].
{LaunchTemplate: LaunchTemplate, MixedInstancesPolicy:
 MixedInstancesPolicy.LaunchTemplate.LaunchTemplateSpecification}'
```

> **Note**
>
> 起動テンプレートは、Auto Scaling グループの設定に応じて `LaunchTemplate` または `MixedInstancesPolicy` の下にある可能性があります。

2. 起動テンプレートのユーザーデータをデコードします。前のステップの起動テンプレート ID とバージョンを使用します。ユーザーデータ内の CA データが、`describe-cluster` によって返された結合信頼バンドルと一致することを確認します。CA データを保持するフィールドは、お使いのオペレーティングシステムとブートストラップメカニズムによって異なります:

```
aws ec2 describe-launch-template-versions --launch-template-
id <lt-id> --versions <version> --region us-west-2 --query
 'LaunchTemplateVersions[0].LaunchTemplateData.UserData' --output text | base64 --
decode
```

3. アップグレード後に、すべてのワーカーノードが最新の起動テンプレートバージョンで実行されていることを確認します。ノードグループの Auto Scaling グループに対して describe を実行します。各インスタンスの起動テンプレートバージョンと、グループの現在の起動テンプレートバージョンを比較します。すべての `InService` インスタンスが現在のバージョンである必要があります。ローリング置換は、`Terminating` 状態のインスタンスをドレインします。次のインスタンスは無視できます:

```
ASG=$(aws eks describe-nodegroup --cluster-name my-cluster --nodegroup-name my-
nodegroup --region us-west-2 --query 'nodegroup.resources.autoScalingGroups[0].name'
 --output text)
aws autoscaling describe-auto-scaling-groups --auto-scaling-group-names "$ASG" --
region us-west-2 --query 'AutoScalingGroups[0].Instances[].{InstanceId: InstanceId,
 LifecycleState: LifecycleState, LaunchTemplateVersion: LaunchTemplate.Version}' --
output table
```

4. 置換ノードが正常であることを確認します。ノードグループ内のすべてのノードが `Ready` である必要があります。これは、kubelet が更新された CA 信頼バンドルを使用して API サーバーへの接続を確立したことを確認するものです:

```
kubectl get nodes -l eks.amazonaws.com/nodegroup=my-nodegroup
```

Karpenter 制御ノード

Karpenter NodePool でドリフト検出が有効になっている場合、Karpenter はノードが古い CA データで実行されていることを自動的に検出し、設定された中断ウィンドウ内でそれらを循環させます。新しいノードは、手動操作なしで後継 CA を取得します。

NodePool の設定でドリフト検出が有効になっていることを確認します:

```
apiVersion: karpenter.sh/v1
kind: NodePool
metadata:
  name: default
spec:
  disruption:
    consolidationPolicy: WhenEmptyOrUnderutilized
    budgets:
      - nodes: "10%"
  template:
    spec:
      nodeClassRef:
        group: karpenter.k8s.aws
        kind: EC2NodeClass
        name: default
```

disruption が統合ポリシーで設定されている場合、ドリフト検出はデフォルトで有効です。Karpenter は、望ましい状態からドリフトしたノードを置き換えます。これには CA データの変更が含まれます。

ドリフト検出が有効になっている場合は、中断予算によって、後継 CA の有効化日より前にすべての Karpenter 制御ノードが置き換えられることを確認します。予算が過度に制限されている場合 (例えば、置換率の低い狭いメンテナンスウィンドウ)、期限内に置き換えられないノードがある可能性があります。

ドリフト検出が無効になっている場合、または中断予算がローテーションのタイムラインを超えるウィンドウに置換を制限している場合は、ノードを遮断およびドレインすることで手動でノードの置換をトリガーできます:

```
kubectl cordon <node-name>
kubectl drain <node-name> --ignore-daemonsets --delete-emptydir-data
```

Karpenter は、更新された CA データでブートストラップする置換ノードをプロビジョニングします。

セルフマネージド型ノード

セルフマネージド型ノードの場合は、ノードがブートストラップ時に使用する起動テンプレートまたはユーザーデータスクリプト内の CA データを更新する必要があります:

1. 更新された CA データを取得します:

```
aws eks describe-cluster --name my-cluster --region us-west-2 --query
 'cluster.certificateAuthority.data' --output text
```

2. 起動テンプレート (またはユーザーデータ) を更新された CA データ値で更新します。
3. Auto Scaling グループを通じてノードのローリング置換をトリガーします (例えば、インスタンスの更新)。

新しいノードは、更新された CA データでブートストラップし、現在の CA と後継 CA の両方を信頼します。

AWS Fargate ポッド (EKS Fargate 起動タイプ)

EKS クラスター内の AWS Fargate ポッドは、他の EKS データプレーンの起動モード (マネージド型ノードグループ、セルフマネージド型ノード、Karpenter 制御ノード) とは異なる動作をします。

ポッドが API サーバーに接続すると、2 つのことが起こります。ポッドはサービスアカウントトークンを使用して自身を認証し (API サーバーに対して自身のアイデンティティを証明)、ポッドはサーバーの証明書が信頼する CA によって署名されていることを確認することで API サーバーのアイデンティティを検証します。ポッドの環境に保存されている CA データが、この検証を可能にします。API サーバーが、ポッドが信頼していない後継 CA によって署名された証明書を提示し始めると、ポッドは接続を拒否します。

他の EKS データプレーンの起動モード (マネージド型ノードグループ、セルフマネージド型ノード、Karpenter 制御ノード) では、CA データはノード上に存在します。ノードを置き換えると、新しいノードは更新された CA データでブートストラップします。その新しいノードにスケジュールされたポッドは、更新された CA データを受け取り、どの CA が証明書に署名したかに関係なく API サーバーを検証できるようになります。

Fargate では、各ポッドは独自の kubelet プロセスで専用のコンピューティング環境で実行されます。この kubelet は、ポッドの作成時の CA データでブートストラップします。その下に共有ノードはなく、基盤となるコンピューティングに直接アクセスすることはできません。

CA ローテーション中、EKS の AWS Fargate ノードに対してお客様の操作は必要ありません。AWS は、パッチ適用プロセスを通じて Fargate ノードのポッドを自然にリサイクルします。後継 CA が追加されると、既存の Fargate ポッドはこのプロセスの一部としてリサイクルされます。お客様が何も操作しなくても、それらは後継 CA を信頼するようになります。AWS は、EKS で AWS が開始する CA 有効化のタイムラインよりもはるかに早く後継 CA を追加するため、AWS が後継 CA を有効化するまでに、Fargate ポッドはリサイクルされ、後継 CA を信頼するようになります。Fargate ポッドがリサイクルを完了するまで後継 CA の有効化を防ぐ組み込みのセーフガードがあります。

つまり、EKS クラスターの両方のマネージドデータプレーンオプション (EKS Auto Mode と Fargate) は、CA ローテーションに関して同じユーザーエクスペリエンスを持ちます。ワーカーノードに対してお客様の操作は必要ありません。API サーバーに接続する外部クライアントの更新については、引き続きお客様の責任となります。

これはエッジケースです。ローテーションのタイムライン (有効期限の数年前に後継 CA が追加される) を考慮すると、大多数のシナリオで、自然なパッチ適用サイクルは後継 CA の有効化よりもはるかに早く完了します。このセーフガードは、お客様が早期に有効化を試みるという可能性の低いケースに対する予防措置として存在します。

外部クライアント (監視ツール、サードパーティ統合)

kubeconfig または証明書信頼設定を使用して EKS クラスターの API サーバーに接続するアプリケーションまたはツールは、更新された CA データで更新する必要があります。これには、以下が含まれます。

- 監視およびオブザーバビリティツール (Datadog、Prometheus、Grafana エージェント)
- クラスター外で実行される GitOps コントローラー (ArgoCD、Flux)
- Kubernetes API を呼び出すカスタムオートメーションまたはスクリプト
- `certificate-authority-data` を静的な値として保存する任意のシステム

これらのそれぞれについて、保存されている CA データを `describe-cluster` からの更新された値に置き換えます。

クライアントが更新されたことを確認する方法

クライアントを更新した後、それが API サーバーとまだ通信できることを確認します:

```
kubectl get nodes
```

コマンドが成功した場合、kubeconfig はアクティブな CA データを信頼します。後継 CA の有効化後に、同じコマンドを実行して接続が継続していることを確認します。

Infrastructure as Code

ドリフトを作成したり、IaC 設定の変更を必要としたりすることなく、既存の Infrastructure as Code (IaC) と併せて CA ローテーションを実行できます。

CA ローテーションが IaC の状態に影響しない理由

CA のライフサイクルは、EKS クラスターリソースの設定とは完全に別個の専用 EKS API (`create-certificate-authority`、`activate-certificate-authority`、`delete-certificate-authority`) を通じて管理されます。CA ローテーションがお客様によって開始されるか、AWS によって自動的に開始されるかに関係なく、EKS クラスターリソース上で IaC ツールが追跡するプロパティは変更されません。

つまり、次のようになります。

- CA の追加または有効化後に IaC スタックを適用または更新しても、ドリフトを検出したり、CA の状態を調整しようとしたりすることはありません
- CLI またはコンソールを介して開始された CA ローテーション操作は、IaC で管理されるクラスターリソースと競合しません

`describe-cluster` によって返される `certificateAuthority.data` フィールドは読み取り専用の出力です。これは現在の結合された信頼バンドル (デュアルトラスト期間中は両方の CA) を反映しますが、設定可能なプロパティではありません。IaC ツールは、それを調整すべきものとして追跡しません。

各 CA レコードの属性フィールド (`createdBy`、`activatedBy`) を使用すると、お客様が開始した操作と AWS が自動的に開始した操作を区別でき、監査および変更管理のワークフローをサポートします。

CA ローテーションで CloudFormation を使用する

CA ローテーションは、`AWS::EKS::Cluster` リソースの WriteOnly プロパティを使用して CloudFormation を通じてトリガーできます。このプロパティは有効化をトリガーしますが、スタックの状態には保存されないため、これを含まない後続のスタック更新で元に戻したり無効化したりしようとすることはありません。

```
# Phase 1: Add to existing stack that manages your cluster
Resources:
    NewCA:
        Type: AWS::EKS::CertificateAuthority
        Properties:
            ClusterName: my-cluster

    MyCluster:
        Type: AWS::EKS::Cluster
        Properties:
            Name: my-cluster
```

この最初のスタック更新は、クラスターに後継 CA を追加します。続行する前に、後継 CA の配布ステータスが `COMPLETE` に達するまで待ちます。

```
# Phase 2: After distribution completes, update your existing Cluster resource to
 activate
Resources:
    NewCA:
        Type: AWS::EKS::CertificateAuthority
        Properties:
            ClusterName: my-cluster

    # Update your existing Cluster resource to activate
    MyCluster:
        Type: AWS::EKS::Cluster
        Properties:
            Name: my-cluster
            ActiveCertificateAuthorityId: !GetAtt NewCA.Id
```

この 2 回目のスタック更新は、後継 CA の有効化をトリガーします。`ActiveCertificateAuthorityId` は WriteOnly プロパティであるため、読み取り時に返されず、CloudFormation 外でアクティブ CA が変更された場合 (例えば、AWS による自動有効化)、CloudFormation はドリフトを検出しません。

重要: CA ローテーションの CloudFormation 統合は、一般的な CloudFormation リソースとは異なるパターンに従います。EKS の認証局は、独自の ARN を持つ独立したリソースではありません。これはクラスターの証明書ライフサイクルの一部として存在し、クラスター自体を通じて承認されます。これは、IAM ロールポリシーが親ロール (`AWS::IAM::RolePolicy`) を通じて承認される方法や、EIP 関連付けがインスタンス (`AWS::EC2::EIPAssociation`) を通じて承認される方法に似ています。CloudFormation を通じて CA ローテーションを管理するお客様は、この違いを認識しておく必要があります。

マネージドノードグループと IaC

更新された CA 信頼設定でノードを更新するためにノードグループのバージョン更新を実行することは、運用アクションです。新しいノードは、クラスターからのアクティブな CA 信頼データで自動的にブートストラップします。IaC テンプレートが起動テンプレートまたはユーザーデータに CA データをハードコードしていない場合、テンプレートの変更は必要ありません。

CA ロールバック

後継 CA を有効化した後、管理するワーカーノード (EKS Auto Mode 以外、Fargate 以外) または外部クライアントで接続の問題が発生した場合、以前の CA にロールバックできます。ロールバックすると、EKS クラスターの署名機関として以前の CA が再び有効化されます。

CA ロールバックが利用可能な場合

CA ロールバックは、次の条件が満たされる限り、CA の有効化後に利用できます:

- CA の有効化がお客様によって開始されたか、AWS による最初の自動有効化 (旧 CA の有効期限の約 6 か月前) のいずれかである
- ロールバックウィンドウが期限切れになっていない

`describe-certificate-authority` によって返される `rollbackAvailable` フィールドを使用して、CA ロールバックがいつでも利用可能かどうかを確認できます:

```
aws eks describe-certificate-authority --cluster-name my-cluster --certificate-
authority-id <ca-id> --region us-west-2
```

```
{
    "certificateAuthority": {
        "id": "a1b2c3d4-5678-90ab-cdef-example22222",
```

```
        "signingStatus": "IN_USE",
        "distributionStatus": "COMPLETE",
        "rollbackAvailable": true
    }
}
```

CA ロールバックが利用できない場合

最終自動有効化後は、CA ロールバックは利用できません。AWS が後継 CA を最後に有効化する場合 (CA の有効期限の 45 日前)、ローテーションは前進させなければなりません。最終自動有効化は、最初の自動有効化が以前にロールバックされた場合にのみ発生します。これは、有効な CA が設定されていないまま EKS クラスターが CA の有効期限に達しないようにするための最後の手段のセーフガードとして存在します。

ロールバックする方法

ロールバックするには、以前の CA を再び有効化します:

```
aws eks activate-certificate-authority --cluster-name my-cluster --certificate-
authority-id <previous-ca-id> --region us-west-2
```

CA ロールバック中に何が起こるか

- 以前の CA が EKS クラスターの証明書の署名を再開します
- 後継 CA は信頼バンドルに残ります (両方の CA が引き続き信頼されます)
- 後継 CA を信頼するようにすでに更新されたワーカーノードとクライアントは引き続き動作します (両方の CA を信頼しています)
- まだ更新されていないワーカーノードとクライアントは通常の動作を再開します (API サーバーは、それらがすでに信頼している CA によって署名された証明書を提示しています)
- ワーカーノード上の kubelet プロセスは、組み込みの再試行ループを介して自動的に再接続します

CA ロールバックを検討すべき場合

CA ロールバックは、後継 CA の有効化によって事前に把握できなかった接続の問題が明らかになった状況に対する安全メカニズムです:

- 更新フェーズで特定されなかった外部クライアントが、後継 CA の有効化後に接続を失います
- 監視またはオブザーバビリティツールが新しい証明書の検証に失敗します

- CI/CD パイプラインが、ハードコードされた証明書信頼設定を使用しているために壊れる

ロールバック後は、後継 CA を再び有効化する前に問題を特定して修正するための完全なデュアルトラスト期間が保持されます。

CA ロールバックなしでの復旧

マネージドノードグループを更新する前に後継 CA を有効化し、CA ロールバックウィンドウが利用できなくなった場合 (例えば、最終自動有効化の期限後)、影響を受けるノードグループでローリング更新を実行することで復旧できます。ただし、切断されたノードは API サーバーからポッドのエビクションコマンドを受信できないため、最初のローリング更新の試行は失敗します。

復旧手順

1. NotReady 状態のノードを特定します:

```
kubectl get nodes
```

2. 各 NotReady ノード上のポッドを一覧表示します:

```
kubectl get pods --all-namespaces --field-selector spec.nodeName=<node-name>
```

3. 各 NotReady ノード上のすべてのポッドを強制削除します:

```
kubectl delete pod --force --grace-period=0 -n <namespace> <pod-name>
```

4. ローリング更新を再試行します:

```
aws eks update-nodegroup-version --cluster-name <cluster> --nodegroup-name
 <nodegroup> --region <region>
```

5. ノードが復旧することを確認します:

```
kubectl get nodes
```

> **Important**
>
> ポッドを強制削除すると、ワークロードが正常に終了されません。ステートフルなワークロードではデータ損失が発生する可能性があります。コンテナは、Auto Scaling グループに

よって終了されるまで、切断されたインスタンス上で実行され続ける可能性があります。これは、CA ロールバックウィンドウが利用できなくなった場合の最後の手段です。

考慮事項と制限事項

リージョン別の可用性

CA ローテーションは、Amazon EKS がサポートされているすべての AWS 商用リージョンで利用できます。

最大 2 つの CA

EKS クラスターは、いつでも最大 2 つの CA (アクティブ CA と 1 つの後継 CA) を持つことができます。以前のローテーションが完了するまで、2 つ目の後継 CA を追加することはできません。

証明書の失効なし

CA ローテーションは、個々の証明書の失効をサポートしていません。これは、証明書の失効 (CRL または OCSP) を実装していないアップストリームの Kubernetes と一致しています。ローテーションは CA 全体を置き換え、これにより、旧 CA が信頼バンドルから削除された後、旧 CA によって署名されたすべての証明書が自然に無効になります。

AWS が追加した CA は削除できません

AWS が後継 CA を自動的に追加した場合、それを削除することはできません。このセーフガードにより、誤った削除によってローテーションプロセスが中断されないようにします。お客様が追加した CA は、アクティブな署名 CA でない限り削除できます。

CA の有効期間

クラスターとともに作成された元の CA の有効期間は 10 年間です。ローテーションプロセスを通じて作成された後継 CA の有効期間は 5 年間です。クラスターの今後のすべての CA の有効期間は 5 年間です。`describe-certificate-authority` を使用して CA の有効期限を確認します。

デュアルトラスト期間中の EC2 ユーザーデータサイズ

デュアルトラスト期間中は、2 つの CA 証明書が含まれるため、クラスターの信頼バンドルのサイズが増加します (合計で約 2.8 KB、gzip 圧縮では約 1.9 KB)。カスタムユーザーデータが EC2 起動テンプレートで提供されるワーカーノードの場合、合計ユーザーデータサイズが EC2 ユーザーデータ

の制限である 16KB を超えていないことを確認します。既存のユーザーデータがこの上限に近い場合、2 つ目の CA の追加により起動テンプレートの作成が失敗し、新しいノードがプロビジョニングされなくなる可能性があります。サイズを削減するために、gzip を使用してユーザーデータの内容を圧縮することを検討してください。

CA ローテーション中のバージョンアップグレード

EKS クラスターのバージョンアップグレードと CA ローテーションは独立した操作です。ただし、両方を同時に実行することはできません。CA ローテーション操作が進行中の場合、CA 操作が完了するまでバージョンアップグレードは拒否され、その逆も同様です。

BYOCA (自分の CA を使用する)

独自の AWS プライベート CA を使用して EKS クラスターの証明書を支えることは、現在サポートされていません。

よくある質問

CA ローテーションを開始するにはどうすればよいですか?

開始するには、`aws eks list-certificate-authorities --cluster-name my-cluster` を実行してアクティブな CA とその有効期限を表示します。ローテーションを開始する準備ができたら、`aws eks create-certificate-authority --cluster-name my-cluster` を実行して後継 CA を追加します。段階的なプロセス全体は「開始方法」セクションで説明されています。Amazon EKS コンソールを通じて CA ローテーションを実行することもできます。

CA ローテーションに関連するコストは発生しますか?

いいえ。CA ローテーションは、すべての EKS クラスターで追加コストなしに利用できます。

CA の有効期限が切れる前に CA をローテーションしないとどうなりますか?

有効な CA が存在しない状態で EKS クラスターが CA の有効期限切れに達することを防ぐ自動保護機能があります。お客様自身が CA ローテーションを開始しない場合、有効期限が切れる前に後継 CA を自動的に追加して有効化し、クラスターが引き続き利用できるようにします。

ただし、ローテーションを成功させるには、お客様が管理するワーカーノード (EKS Auto Mode 以外、Fargate 以外) と外部クライアントを、後継 CA を信頼するように更新する必要もあります。これらのコンポーネントが後継 CA の有効化前に更新されていない場合、API サーバーへの接続性を失います。

Kubernetes では、CA の有効期限が切れる前に CA をローテーションしないと、その CA によって署名されたすべての証明書が無効になります。どのクライアントからも API サーバーに到達できなくなり、クラスターは利用できなくなります。

CA ローテーションを完了するためにどのくらいの時間がありますか?

CA ローテーションを完了するまでの総時間は、AWS の自動保護機能とお客様自身の更新プロセスの両方に左右されます。

AWS は、自らが管理する対象について確実なタイムラインを示します。後継 CA は、既存の CA の有効期限が切れる約 2 年前に追加されます。お客様自身が後継 CA を有効化しない場合、有効期限の約 6 か月前に自動的に有効化します。自動有効化後にロールバックした場合、有効期限の 45 日前に最終自動有効化を実行します。これらの保護機能により、お客様が操作するかどうかに関係なく、EKS クラスターは利用可能なままです。

CA ローテーションを成功させるには、後継 CA の有効化前に、お客様が管理するワーカーノード (EKS Auto Mode 以外、Fargate 以外) と外部クライアントを後継 CA を信頼するように更新することも必要です。これにかかる時間は、データプレーンのワーカーノード設定、外部クライアントの規模、およびこれらのコンポーネントの検出と更新にかかる時間に左右されます。

CA ローテーションによってクラスターにダウンタイムが発生しますか?

いいえ。EKS クラスターは、CA ローテーションライフサイクル全体を通じて引き続き利用できます。コントロールプレーンは、すべての段階でリクエストの処理を続けます。デュアルトラスト期間中は、既存の CA と後継 CA の両方が同時に信頼されるため、クラスター運用を中断することなくコンポーネントを段階的に更新できます。クライアント側の問題を検出したために以前の CA にロールバックした場合、AWS はセーフガードとして、既存の CA の有効期限が切れる約 45 日前に最終ロールフォワードを実行します。

EKS クラスター (コントロールプレーン) とデータプレーンコンポーネントを区別することが重要です。コントロールプレーンは AWS によって完全に管理されており、ローテーション中も利用可能なままです。

EKS Auto Mode と Fargate の場合、AWS がワーカーノードを自動的に更新します。これらのデータプレーン起動モードでは、ワーカーノードの接続性が失われるリスクはありません。API サーバーに接続する外部クライアントの更新については、引き続きお客様の責任となります。

マネージド型ノードグループ、セルフマネージド型ノード、Karpenter 制御インスタンス (ドリフト検出が有効になっていない場合)、およびハイブリッドノードでは、後継 CA の有効化前にそれらのノードを置き換えるか更新する責任はお客様にあります。置き換えられていない場合、コントロー

ルプレーン自体は完全に動作し続けますが、後継 CA の有効化後にそれらのノードはコントロールプレーンへの接続を失います。

CA ローテーション中にワークロードは中断されますか?

実行中のワークロード (ポッド) は、CA ローテーション自体によって中断されません。ポッドはクラスターネットワークを通じて相互に通信し、このネットワークは CA の変更による影響を受けません。CA は、コンポーネントと API サーバー間の通信に使用され、ポッド間トラフィックには使用されません。

後継 CA を信頼するようにワーカーノードを更新する一環としてノードを置き換える必要がある場合 (例えば、マネージド型ノードグループがローリング更新を実行する場合や、Karpenter がドリフトしたノードを置き換える場合)、それらのノード上のポッドは通常のノード置き換えプロセスの一部として再スケジュールされます。これはノード置き換え時の標準的な Kubernetes の動作であり、CA ローテーションの副作用ではありません。ノード置き換え時にポッドがどのように エビクションされるかを制御するため、重要なワークロードにはポッド中断予算 (PDB) を設定するようにしてください。

クラスターの CA の有効期限はいつ切れますか?

クラスターの CA の有効期限がいつ切れるかは、AWS CLI、EKS API、または Amazon EKS コンソールを使用して確認できます。例えば、次のコマンドを実行できます。

```
aws eks describe-certificate-authority --cluster-name my-cluster
 --certificate-authority-id <ca-id> --region us-west-2 --query
 'certificateAuthority.validity.notAfter'
```

`aws eks list-certificate-authorities --cluster-name my-cluster` を実行して、CA の ID を確認できます。

クラスターの CA の有効期間はどのくらいですか?

クラスターとともに作成された元の CA の有効期間は 10 年間です。ローテーションプロセスを通じて作成された後継 CA の有効期間は 5 年間です。クラスターの今後のすべての CA の有効期間は 5 年間です。`describe-certificate-authority` を使用して、CA の有効期限を確認できます。

AWS が自動的に実行する前に、自分で CA ローテーションを開始できますか?

はい。`aws eks create-certificate-authority --cluster-name my-cluster` を使用して、いつでも後継 CA を追加できます。AWS がローテーションを開始するのを待つ必要はありませ

ん。早期に開始すると、自分のスケジュールでワーカーノードと外部クライアントを特定して更新するための時間をより多く確保できます。

後継 CA を有効化した後にロールバックできますか?

はい。後継 CA の有効化後でも、ロールバックウィンドウが期限切れになっていなければ、CA ロールバックを利用できます。CA ロールバックは、以前の CA を署名機関として再度有効化します。最終自動有効化後 (CA の有効期限切れの 45 日前) は利用できません。`describe-certificate-authority` を使用して、CA の `rollbackAvailable` フィールドを確認できます。詳細については、CA ロールバックのセクションを参照してください。

後継 CA を有効化しても安全な時期は、どうすればわかりますか?

お客様が管理するすべてのワーカーノード (EKS Auto Mode 以外、Fargate 以外) と外部クライアントが後継 CA を信頼するように更新されたら、後継 CA を有効化しても安全です。これを確認するには、各クライアントが更新された信頼設定を使用して API サーバーと正常に通信できることを確認します。AWS は外部システムを参照できないため、すべてのクライアントの準備が完了したことを示す単一の指標は提供しません。すべてのクライアントを特定する時間を確保するために、できるだけ早く検出プロセスを開始してください。

複数のクラスターにまたがる CA ローテーションの進行状況を監視するにはどうすればよいですか?

AWS CLI または EKS API を使用して、プログラムでこれを実行できます。各クラスターに対して `aws eks list-certificate-authorities` を使用します。次のフィールドは、フリートレベルの監視における状況把握に役立ちます。

- `signingStatus`: CA がアクティブに証明書に署名しているかどうかを示します (`NOT_USED`、`ACTIVATING`、`IN_USE`)
- `distributionStatus`: AWS が管理対象コンポーネントへの CA の配布を完了したかどうかを示します (`IN_PROGRESS`、`COMPLETE`、`FAILED`、`DELETING`)
- `rollbackAvailable`: 後継 CA の有効化後に CA ロールバックを利用できるかどうかを示します
- `createdBy` / `activatedBy`: お客様が開始した操作と AWS が開始した操作を区別します (`CUSTOMER`、`EKS`)
- `scheduledEvents.firstAutoActivation` / `scheduledEvents.finalAutoActivation`: 今後の AWS の自動有効化日を示します

次のサンプルスクリプトは、クラスターのリスト全体で CA ローテーションのステータスを確認します。

```
#!/bin/bash
CLUSTERS=("cluster-1" "cluster-2" "cluster-3")
REGION="us-west-2"

for CLUSTER in "${CLUSTERS[@]}"; do
  echo "--- $CLUSTER ---"
  aws eks list-certificate-authorities \
    --cluster-name "$CLUSTER" \
    --region "$REGION" \
    --query 'certificateAuthorities[].
{Id:id,Signing:signingStatus,Distribution:distributionStatus,Expiry:validity.notAfter}'
 \
    --output table
done
```

これを拡張して複数のリージョンとアカウントを対象にし、後継 CA が追加されているクラスター、お客様の操作を待っているクラスター、または自動有効化の期限が近づいているクラスターをフィルタリングできます。

通知は、ローテーションライフサイクルの各段階で、AWS Health とメールを通じてクラスターごとにも配信されます。

すべてのクライアントを更新する前に後継 CA を有効化するとどうなりますか?

後継 CA を信頼するように更新されていないクライアントは、EKS クラスターへの接続性を失います。後継 CA の有効化後、API サーバーは後継 CA によって署名された証明書を提示します。それを信頼しないクライアントは TLS 検証に失敗し、接続できなくなります。これが発生した場合は、以前の CA にロールバックして (ロールバックウィンドウがまだ利用可能な場合)、残りのクライアントを修正する間、接続を復元できます。

CA の有効期限に間に合わなかった場合はどうなりますか?

AWS がこれを防ぎます。自動保護機能により、有効な CA が存在しない状態で EKS クラスターが CA の有効期限切れに達することはありません。お客様が追加していない場合は当社が後継 CA を追加し、有効期限の約 6 か月前に自動的に有効化します。最初の自動有効化後にロールバックした場合、AWS は有効期限の 45 日前に最終ロールフォワードを実行します。クラスターは引き続き利用可能です。

ただし、自動有効化が発生するまでに、お客様が管理するワーカーノード (EKS Auto Mode 以外、Fargate 以外) と外部クライアントが後継 CA を信頼するように更新されていない場合、それらのコンポーネントは API サーバーへの接続を失います。

CA ローテーション中にクラスターへのアクセスを失うことはありますか?

EKS クラスター (コントロールプレーン) は、CA ローテーションライフサイクル全体を通じて利用可能なままです。AWS のセーフガードにより、クラスター自体が利用できなくなることはありません。

ただし、お客様が管理する個々のクライアントは、後継 CA の有効化前に後継 CA を信頼するように更新されていない場合、アクセスを失う可能性があります。例えば、kubeconfig、CI/CD パイプライン、または監視ツールが依然として既存の CA のみを参照している場合、後継 CA の有効化後にそれらのクライアントは接続できなくなります。これが発生した場合、影響を受けるクライアントを更新する間、CA ロールバックによってアクセスを復元できます。

ポッドを再起動する必要はありますか?

実行中のワークロードポッドは、CA ローテーションによる直接的な影響を受けません。各ノードの kubelet が API サーバーとの通信を処理するため、ノードが更新されている限り、ワークロードポッドは中断なく実行され続けます。ただし、client-go を使用して API サーバーと通信するクラスター内のコントローラーとオペレーターは、後継 CA の有効化後に再起動が必要になる場合があります。client-go は CA 信頼バンドルを動的に再読み取りしないためです。特に AWS Fargate ノードの場合、AWS が通常のポッドリサイクルプロセスを通じて更新を自動的に処理します。

ローテーション中に信頼バンドルは変更されますか?

はい。CA ローテーション中、クラスターの信頼バンドルには、既存の CA と後継 CA の 2 つの認証局が同時に含まれます。これはデュアルトラスト期間中の想定された動作であり、ローテーションプロセスがすべてのコンポーネントの接続性を維持する方法です。

アプリケーションとクライアントは、単一の CA 証明書にピン留めするのではなく、CA バンドルを信頼するように設定する必要があります。CA のピン留め (単一の CA に対する厳密な検証) は、信頼バンドルが更新されたときに障害を引き起こすため、推奨されません。これは、EKS クラスターの API サーバーに接続するすべてのクライアント側 TLS 設定に適用されます。

通知のタイムラインがこのドキュメントの説明と異なるのはなぜですか?

クラスターが 2018 年から 2019 年に作成された場合、クラスターは調整されたタイムラインで自動通知を受け取ります。最初の通知には、クラスターに固有の関連日付と次のステップが含まれます。標準の通知マイルストーンは、クラスターの CA の有効期限を基準に計算されます。この範囲のクラスターでは、計算された日付がこの機能の提供開始より前になるため、調整されたスケジュールが適用されます。

Amazon EKS 作成の RBAC ロールおよびユーザーの説明

Kubernetes クラスターを作成すると、Kubernetes が正常に機能するようにそのクラスターに複数のデフォルト Kubernetes ID が作成されます。Amazon EKS はデフォルトコンポーネントごとに Kubernetes ID を作成します。ID により、クラスターコンポーネントの Kubernetes ロールベースのアクセスコントロール (RBAC) を実現できます。詳細についてはKubernetes ドキュメントの「RBAC 認可を使用する」を参照してください。

オプションのアドオンをクラスターにインストールすると、クラスターに追加の Kubernetes ID が追加されることがあります。このトピックで扱われていない ID の詳細については「アドオンのドキュメント」を参照してください。

AWS マネジメントコンソール または `kubectl` コマンドラインツールを使用して、クラスターで Amazon EKS が作成した Kubernetes ID のリストを表示できます。すべてのユーザー ID はお客様に対して、Amazon CloudWatch を通じて利用可能な `kube` 監査ログに表示されます。

AWS マネジメントコンソール

前提条件

使用する IAM プリンシパルには「必要なアクセス許可」に記載されている許可が必要です。

AWS マネジメントコンソール を使用して Amazon EKS で作成された ID を表示するには

1. Amazon EKS コンソールを開きます。
2. [Clusters] (クラスター) のリストで、表示したい ID を含んでいるクラスターを選択してください。
3. [リソース] タブを選択してください。
4. [Resource types] (リソースタイプ) で [Authorization] (承認)を 選択してください。
5. [クラスター役割]、[クラスター役割ビンディング]、[役割]、または [バインディング役割] を選択してください。[eks] というプレフィックスが付いたリソースはすべて、Amazon EKS によって作成されます。Amazon EKS で作成される追加の ID リソースは次のとおりです：
   - [aws-node] という名前の [クラスター役割] と [クラスター役割バインディング]。[aws-node] リソースは Amazon EKS がすべてのクラスターにインストールする Amazon VPC CNI plugin for Kubernetes をサポートします。
   - [vpc-resource-controller-role] という名前の [クラスター役割] および [vpc-resource-controller-rolebinding] という名前の [クラスター役割バインディング]。これらのリソースはAmazon EKS

がすべてのクラスターにインストールする「Amazon VPC resource controller」(Amazon VPC リソースコントローラー) サポートします。

コンソールに表示されるリソースに加えて、次の特別なユーザー ID がクラスターに存在しますが、クラスターの設定には表示されません：

- **`eks:cluster-bootstrap`** - クラスターブートストラップ中の `kubectl` 操作に使用されます。
- **`eks:support-engineer`** - クラスター管理操作に使用されます。

6. 特定のリソースを選択すると、そのリソースの詳細が表示されます。デフォルトでは情報は [Structured view] (構造化ビュー) で表示されます。詳細ページの右上隅で、[Raw View] (生のビュー) 選択すると、リソースの情報をすべて表示できます。

Kubectl

前提条件

クラスター上の Kubernetes リソースを一覧表示するために使用するエンティティ (AWS Identity and Access Management (IAM) または OpenID Connect (OIDC)) は IAM または OIDC ID プロバイダーによって認証される必要があります。エンティティにはそのエンティティに使用させたいクラスター上の、`Role`、`ClusterRole`、`RoleBinding`、および `ClusterRoleBinding` リソースに対してKubernetes `get` および `list` 動詞を使用するアクセス許可が付与されている必要があります。IAM エンティティにクラスターへのアクセスを付与する方法の詳細については「the section called “Kubernetes API アクセス”」を参照してください。独自の OIDC プロバイダーによって認証されたエンティティにクラスターへのアクセスを付与する方法の詳細については「the section called “OIDC プロバイダーをリンクする”」を参照してください。

`kubectl` を使用して Amazon EKS で作成された ID を表示するには

表示するリソースのタイプのコマンドを実行します。[eks] というプレフィックスが付いたすべての返されるリソースはAmazon EKS によって作成されます。コマンドからの出力で返されるリソースに加えて、次の特別なユーザー ID がクラスターに存在しますが、クラスターの設定には表示されません：

- **`eks:cluster-bootstrap`** - クラスターブートストラップ中の `kubectl` 操作に使用されます。
- **`eks:support-engineer`** - クラスター管理操作に使用されます。

ClusterRoles - `ClusterRoles` はクラスターにスコープが設定されているため、ロールに付与されたすべてのアクセス許可はクラスター上の任意の Kubernetes 名前空間内のリソースに適用されます。

次のコマンドは、クラスター上で Amazon EKSによって作成されたすべての Kubernetes `ClusterRoles` を返します。

```
kubectl get clusterroles | grep eks
```

出力で返される `eks:` というプレフィックスが付けられた `ClusterRoles` の他に、次の `ClusterRoles` が存在します。

- **`aws-node`** – この `ClusterRole` はAmazon EKS がすべてのクラスターにインストールする Amazon VPC CNI plugin for Kubernetes, をサポートします。
- **`vpc-resource-controller-role`** - この `ClusterRole` はAmazon EKS がすべてのクラスターにインストールする「Amazon VPC resource controller」(Amazon VPC リソースコントローラー) サポートします。

`ClusterRole` の仕様を確認するには次のコマンドの *`eks:k8s-metrics`* を前のコマンドの出力で返された `ClusterRole` に置き換えます。次の例では*`eks:k8s-metrics`* `ClusterRole` の仕様を返します。

```
kubectl describe clusterrole eks:k8s-metrics
```

出力例は次のとおりです。

```
Name:         eks:k8s-metrics
Labels:       <none>
Annotations:  <none>
PolicyRule:
  Resources         Non-Resource URLs  Resource Names  Verbs
  ---------         -----------------  --------------  -----
                    [/metrics]         []              [get]
  endpoints         []                 []              [list]
  nodes             []                 []              [list]
  pods              []                 []              [list]
  deployments.apps  []                 []              [list]
```

クラスター役割Bindings - `ClusterRoleBindings` はクラスターを対象としています。

次のコマンドは、クラスター上で Amazon EKSによって作成されたすべての Kubernetes ClusterRoleBindings を返します。

```
kubectl get clusterrolebindings | grep eks
```

出力で返される ClusterRoleBindings 以外に、次の ClusterRoleBindings が存在します。

- **aws-node** – この ClusterRoleBinding はAmazon EKS がすべてのクラスターにインストールする Amazon VPC CNI plugin for Kubernetes, をサポートします。
- **vpc-resource-controller-rolebinding** - この ClusterRoleBinding はAmazon EKS がすべてのクラスターにインストールする「Amazon VPC resource controller」(Amazon VPC リソースコントローラー) サポートします。

ClusterRoleBinding の仕様を確認するには次のコマンドの *eks:k8s-metrics* を前のコマンドの出力で返された ClusterRoleBinding に置き換えます。次の例では*eks:k8s-metrics* ClusterRoleBinding の仕様を返します。

```
kubectl describe clusterrolebinding eks:k8s-metrics
```

出力例は次のとおりです。

```
Name:         eks:k8s-metrics
Labels:       <none>
Annotations:  <none>
Role:
  Kind:  ClusterRole
  Name:  eks:k8s-metrics
Subjects:
  Kind  Name             Namespace
  ----  ----             ---------
  User  eks:k8s-metrics
```

Roles - Roles は Kubernetes 名前空間にスコープされます。Roles で作成されたすべての Amazon EKS はkube-system 名前空間にスコープされます。

次のコマンドは、クラスター上で Amazon EKSによって作成されたすべての Kubernetes Roles を返します。

```
kubectl get roles -n kube-system | grep eks
```

Role の仕様を確認するには、次のコマンドの *eks:k8s-metrics* を、前のコマンドの出力で返された Role の名前に置き換えます。次の例では*eks:k8s-metrics* Role の仕様を返します。

```
kubectl describe role eks:k8s-metrics -n kube-system
```

出力例は次のとおりです。

```
Name:         eks:k8s-metrics
Labels:       <none>
Annotations:  <none>
PolicyRule:
  Resources         Non-Resource URLs  Resource Names              Verbs
  ---------         -----------------  --------------              -----
  daemonsets.apps   []                 [aws-node]                  [get]
  deployments.apps  []                 [vpc-resource-controller]  [get]
```

RoleBindings - RoleBindings は Kubernetes 名前空間にスコープされます。RoleBindings で作成されたすべての Amazon EKS はkube-system 名前空間にスコープされます。

次のコマンドは、クラスター上で Amazon EKSによって作成されたすべての Kubernetes RoleBindings を返します。

```
kubectl get rolebindings -n kube-system | grep eks
```

RoleBinding の仕様を確認するには次のコマンドの *eks:k8s-metrics* を前のコマンドの出力で返された RoleBinding に置き換えます。次の例では*eks:k8s-metrics* RoleBinding の仕様を返します。

```
kubectl describe rolebinding eks:k8s-metrics -n kube-system
```

出力例は次のとおりです。

```
Name:         eks:k8s-metrics
Labels:       <none>
Annotations:  <none>
Role:
  Kind:  Role
  Name:  eks:k8s-metrics
Subjects:
  Kind  Name             Namespace
  ----  ----             ---------
```

```
User  eks:k8s-metrics
```

既存のクラスターで KMS を使用して Kubernetes シークレットを暗号化する

> **Important**
>
> 非推奨の通知: この手順は非推奨となっており、適用対象となるのは Kubernetes バージョン 1.27 以前を実行している EKS クラスターのみです。Kubernetes バージョン 1.28 以降を実行している場合、Amazon EKS はデフォルトでエンベロープ暗号化を使用して Kubernetes シークレットを保護します。詳細については、「the section called “すべての Kubernetes API データを対象とするデフォルトのエンベロープ暗号化”」を参照してください。

シークレット暗号化を有効にすると、選択した AWS KMS キーを使用して Kubernetes シークレットが暗号化されます。KMS キーは次の条件を満たす必要があります：

- 対称
- データの暗号化と復号が可能
- クラスターと同じ AWS リージョンに作成
- KMS キーが別のアカウントで作成された場合、IAM プリンシパルにはその KMS キーへのアクセス権が必要となります。

詳細については、「AWS Key Management Service デベロッパーガイド」の「他のアカウントの IAM プリンシパルが KMS キーを使用することを許可する」を参照してください。

> **Warning**
>
> 一度有効化したシークレット暗号化は無効化できません。このアクションを元に戻すことはできません。

eksctl

この手順の対象となるのは、Kubernetes バージョン 1.27 以前を実行している EKS クラスターのみです。詳細については、「the section called “すべての Kubernetes API データを対象とするデフォルトのエンベロープ暗号化”」を参照してください。

暗号化は次の 2 つの方法のいずれかで有効にできます：

- 1 つのコマンドでクラスターに暗号化を追加します。

  シークレットを自動的に再暗号化するには次のコマンドを実行してください。

  ```
  eksctl utils enable-secrets-encryption \
      --cluster my-cluster \
      --key-arn arn:aws:kms:region-code:account:key/key
  ```

  シークレットの自動的な再暗号化をオプトアウトするには次のコマンドを実行してください。

  ```
  eksctl utils enable-secrets-encryption \
      --cluster my-cluster \
      --key-arn arn:aws:kms:region-code:account:key/key \
      --encrypt-existing-secrets=false
  ```

- kms-cluster.yaml ファイルを使用してクラスターに暗号化を追加します。

  ```
  apiVersion: eksctl.io/v1alpha5
  kind: ClusterConfig

  metadata:
    name: my-cluster
    region: region-code

  secretsEncryption:
    keyARN: arn:aws:kms:region-code:account:key/key
  ```

  シークレットに自動的な再暗号化をさせるには次のコマンドを実行してください。

  ```
  eksctl utils enable-secrets-encryption -f kms-cluster.yaml
  ```

  シークレットの自動的な再暗号化をオプトアウトするには次のコマンドを実行してください。

  ```
  eksctl utils enable-secrets-encryption -f kms-cluster.yaml --encrypt-existing-
  secrets=false
  ```

AWS マネジメントコンソール

a. この手順の対象となるのは、Kubernetes バージョン 1.27 以前を実行している EKS クラスターのみです。詳細については、「the section called “すべての Kubernetes API データを対象とするデフォルトのエンベロープ暗号化”」を参照してください。

b. Amazon EKS コンソールを開きます。

c. KMS 暗号化を追加するクラスターを選択してください。

d. [Overview] (概要 タブを選択してください (これはデフォルトで選択されています)。

e. [秘密の暗号化] (シークレットの暗号化 セクションまでスクロールダウンし、[Enable] (有効化 ボタンを選択してください。

f. ドロップダウンリストからキーを選択し、[Enable] (有効化 ボタンを選択してください。キーが一覧表示されていない場合は最初にキーを作成する必要があります。詳細については「キーの作成」を参照してください。

g. [Confirm] (確認) ボタンを選択して、指定したキーを使用します。

AWS CLI

a. この手順の対象となるのは、Kubernetes バージョン 1.27 以前を実行している EKS クラスターのみです。詳細については、「the section called “すべての Kubernetes API データを対象とするデフォルトのエンベロープ暗号化”」を参照してください。

b. 次の AWS CLI コマンドを使用して、シークレット暗号化設定をクラスターに関連付けます。値の例 は実際に使用する値に置き換えます。

```
aws eks associate-encryption-config \
    --cluster-name my-cluster \
    --encryption-config '[{"resources":["secrets"],"provider":
{"keyArn":"arn:aws:kms:region-code:account:key/key"}}]'
```

出力例は次のとおりです。

```
{
  "update": {
    "id": "3141b835-8103-423a-8e68-12c2521ffa4d",
    "status": "InProgress",
    "type": "AssociateEncryptionConfig",
    "params": [
      {
        "type": "EncryptionConfig",
```

```
        "value": "[{\"resources\":[\"secrets\"],\"provider\":{\"keyArn\":
\"arn:aws:kms:region-code:account:key/key\"}}]"
      }
    ],
    "createdAt": 1613754188.734,
    "errors": []
  }
}
```

c. 次のコマンドを使用すると、暗号化更新のステータスをモニタリングできます。前の出力で返された特定の cluster name と update ID を使用します。Successful ステータスが表示されると、更新は完了です。

```
aws eks describe-update \
    --region region-code \
    --name my-cluster \
    --update-id 3141b835-8103-423a-8e68-12c2521ffa4d
```

出力例は次のとおりです。

```
{
  "update": {
    "id": "3141b835-8103-423a-8e68-12c2521ffa4d",
    "status": "Successful",
    "type": "AssociateEncryptionConfig",
    "params": [
      {
        "type": "EncryptionConfig",
        "value": "[{\"resources\":[\"secrets\"],\"provider\":{\"keyArn\":
\"arn:aws:kms:region-code:account:key/key\"}}]"
      }
    ],
    "createdAt": 1613754188.734,
    "errors": []
  }
}
```

d. クラスターで暗号化が有効になっていることを確認するにはdescribe-cluster コマンドを実行してください。レスポンスの内容は文字列 EncryptionConfig です。

```
aws eks describe-cluster --region region-code --name my-cluster
```

クラスターで暗号化を有効にしたら、既存のすべてのシークレットを新しいキーで暗号化する必要があります：

> **Note**
>
> eksctl を使用する場合、シークレットを自動的に再暗号化することをオプトアウトした場合にのみ、次のコマンドを実行する必要があります。

```
kubectl get secrets --all-namespaces -o json | kubectl annotate --overwrite -f - kms-encryption-timestamp="time value"
```

> **Warning**
>
> 既存のクラスターでシークレット暗号化を有効にし、使用する KMS キーが削除されている場合はこのクラスターを復旧する手段はありません。KMS キーを削除すると、クラスターは永続的にパフォーマンスが低下した状態になります。詳細については「AWSKMS キーを削除する」を参照してください。

> **Note**
>
> create-key コマンドではデフォルトで対称暗号化 KMS キーが作成されます。この際にはアカウントのルート管理者に AWS KMS アクションとリソースへのアクセスを許可する、キーポリシーが使用されます。アクセス許可の範囲を絞り込む場合はcreate-cluster API を呼び出すプリンシパルのポリシーで、kms:DescribeKey および kms:CreateGrant アクションが許可されていることを確認します。
> KMS エンベロープ暗号化を使用するクラスターの場合、kms:CreateGrant アクセス許可が必要です。条件 kms:GrantIsForAWSResource はクラスター作成 アクションでサポートされていないため、クラスター作成 を実行するユーザーの kms:CreateGrant アクセス許可を制御する KMS ポリシーで使用しないでください。

Amazon EKS ポッドで AWS 秘密マネジャー シークレットを使用する

Secrets Manager のシークレットとパラメータストアのパラメータを、Amazon EKS Pod にマウントされたファイルとして表示するには、Kubernetes の Secrets Store CSI ドライバー向けの AWS Secrets and Configuration Provider (ASCP) を使用します。

ASCP を使用すると、Secrets Manager でシークレットを保存および管理し、Amazon EKS で実行されているワークロードを介して、そのシークレットを取得することが可能です。IAM ロールとポリシーを使用して、シークレットへのアクセスをクラスターにある特定の Kubernetes Pod に制限することができます。ASCP は、Pod Identity を取得して IAM ロールと交換します。ASCP が Pod の IAM ロールを引き受けると、Secrets Manager からそのロールに対して認可されているシークレットを取得できます。

シークレットに 秘密マネジャー の自動ローテーションを使用する場合は シークレットストア CSI ドライバーのローテーション調整機能を使用して、秘密マネジャー から最新のシークレットを取得することもできます。

> **Note**
>
> AWS Fargate (Fargate) ノードグループはサポートされていません。

詳細についてはAWS 秘密マネジャー ユーザーガイドの「Using 秘密マネジャー secrets in Amazon EKS」を参照してください。

すべての Kubernetes API データを対象とするデフォルトのエンベロープ暗号化

Amazon Elastic Kubernetes Service (Amazon EKS) はデフォルトでエンベロープ暗号化を備えており、Kubernetes バージョン 1.28 以降を実行している EKS クラスター内のすべての Kubernetes API データが暗号化の対象となります。

エンベロープ暗号化は、Kubernetes API サーバーで保存するデータを保護します。例えば、エンベロープ暗号化は `ConfigMaps` といった Kubernetes クラスターの設定に適用されます。一方、ノードや EBS ボリューム上のデータにはエンベロープ暗号化は適用されません。EKS は以前には Kubernetes シークレットの暗号化をサポートしていましたが、現在ではこのエンベロープ暗号化の対象がすべての Kubernetes API データへと広がっています。

Kubernetes アプリケーションの多層防御を実装するマネージドサービスがデフォルトで提供されるため、お客様側で対策を講じる必要はありません。

Amazon EKS は、AWS Key Management Service (KMS) を Kubernetes KMS プロバイダー v2 と組み合わせた仕組みを採用しており、Amazon Web Services 所有のキーによるセキュリティレイヤーの追加と、AWS KMS で作成した独自のカスタマーマネージドキー (CMK) を使用するオプションを提供しています。

エンベロープ暗号化について

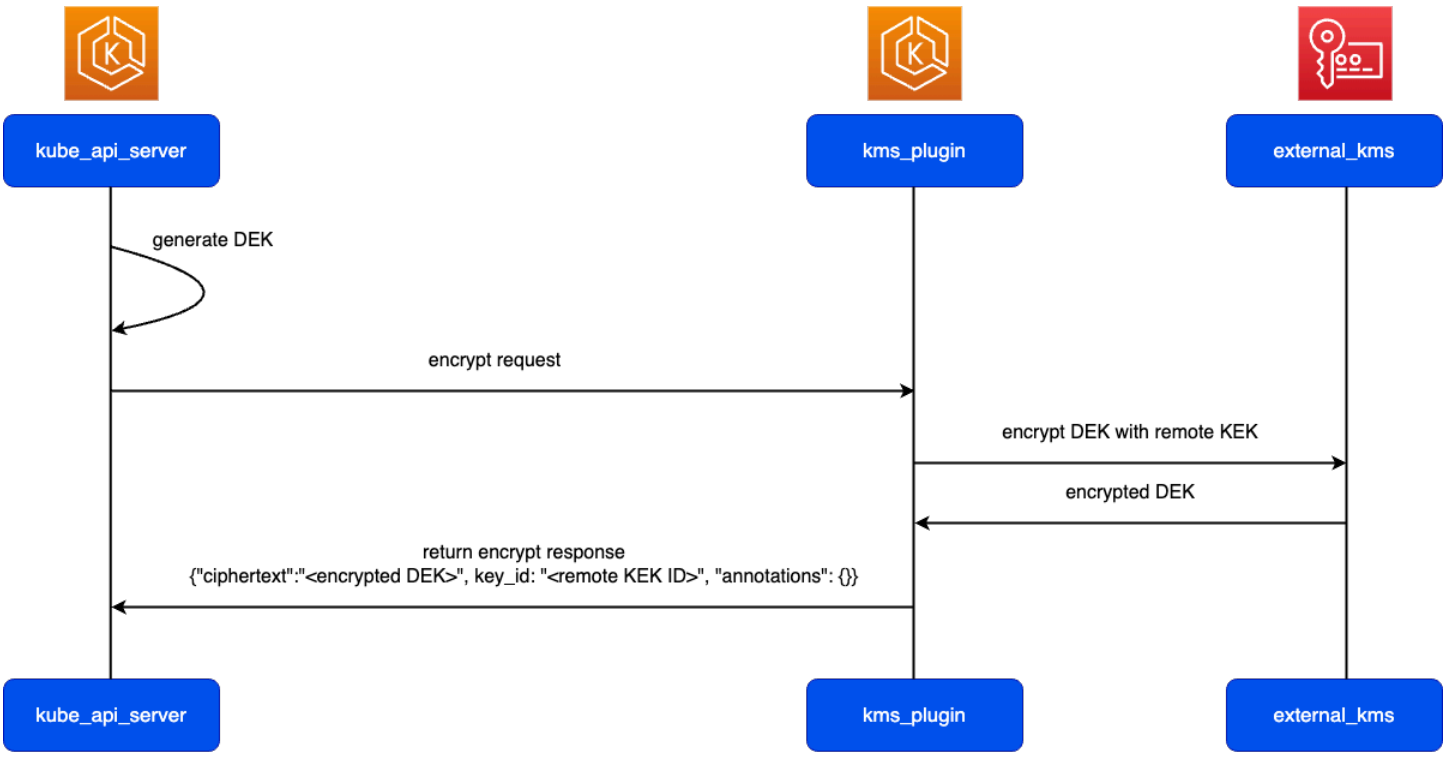
エンベロープ暗号化とは、データストア (etcd) への送信に先立ってプレーンテキストデータをデータ暗号化キー (DEK) で暗号化し、次にリモートで一元管理される KMS システム (AWS KMS) に保存されているルート KMS キーで DEK を暗号化するプロセスです。これは多層防御といわれる戦略で、暗号化キー (DEK) でデータを保護し、さらにその DEK をキー暗号化キー (KEK) という別途安全に保存された暗号化キーで保護してセキュリティ層をもう 1 つ追加することからそう呼ばれています。

Amazon EKS が AWS KMS v2 と KMS でデフォルトのエンベロープ暗号化を実現する仕組み

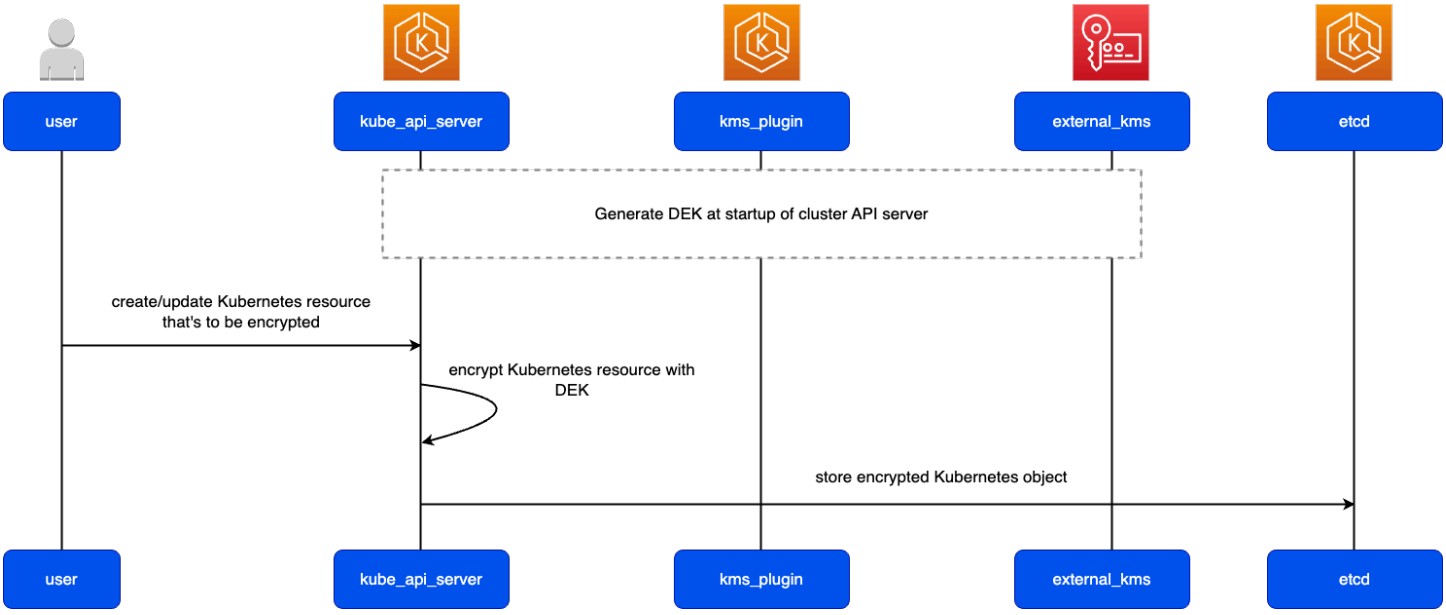
Amazon EKS は、KMS v2 を使用して、マネージド Kubernetes コントロールプレーン内のすべての API データをデフォルトでエンベロープ暗号化してから etcd データベースに保存します。起動時に、クラスター API サーバーはランダムに生成されたデータと組み合わせて、シークレットシードからデータ暗号化キー (DEK) を生成します。同じく起動時に、API サーバーは KMS プラグインへの呼び出しを行い、AWS KMS からリモートのキー暗号化キー (KEK) を使用して DEK シードを暗号化します。これは、API サーバーの起動時と KEK のローテーション時に実行される 1 回限りの呼び出しです。暗号化された DEK シードは API サーバーにキャッシュされます。API サーバーは、そのキャッシュした DEK シードを使用して、キー導出関数 (KDF) に基づいて 1 回限り使用される DEK を別途生成します。こうして生成した各 DEK を Kubernetes リソースごとに 1 回のみ使用して暗号化し、その後 etcd に保存します。KMS v2 で暗号化してキャッシュされた DEK シードを使用すると、API サーバーで Kubernetes リソースを暗号化するプロセスのパフォーマンスとコスト効率が向上します。

この KEK はデフォルトでは AWS によって所有されているものですが、必要に応じて AWS KMS から独自のものを取り込むこともできます。

以下の図は、API サーバーの起動時に行われる DEK の生成と暗号化を示しています。



以下の概要図は、etcd に保存される前に行われる Kubernetes リソースの暗号化を示しています。



よくある質問

デフォルトのエンベロープ暗号化によって、EKS クラスターのセキュリティ体制はどのように改善されますか?

メタデータとカスタマーコンテンツが暗号化されない対象領域と期間を減らします。デフォルトのエンベロープ暗号化により、メタデータとカスタマーコンテンツは etcd に保存される前に kube-apiserver のメモリ内で一時的に暗号化されていない状態になるだけです。kube-apiserver のメモリは、Nitro システムによって安全に保護されます。Amazon EKS で Nitro ベースの EC2 インスタンスが使用されるのは、マネージド Kubernetes コントロールプレーンに限られます。これらのインスタンスでは、システムや個人がメモリにアクセスできないよう、しっかりとセキュリティ管理が設計されています。

この機能を利用するためには、Kubernetes のどのバージョンを実行する必要がありますか?

デフォルトのエンベロープ暗号化を有効にするには、Kubernetes バージョン 1.28 以降で Amazon EKS クラスターを実行している必要があります。

この機能をサポートしていない Kubernetes クラスターバージョンを実行していても、データは引き続き安全に保護されますか?

はい。AWS では、セキュリティを最優先事項としています。Amazon のデジタルトランスフォーメーションとイノベーションはすべてセキュリティが最も高い運用プラクティスに基づいており、Amazon はそのレベルアップに真摯に取り組んでいます。

実行中の Kubernetes バージョンに関係なく、etcd に保存されているすべてのデータが EKS クラスターごとにディスクレベルで暗号化されます。EKS では、ルートキーを使用してボリューム暗号化キーを生成し、それを EKS サービスで管理しています。また、あらゆる Amazon EKS クラスターが、クラスター固有の仮想マシンを使用して、分離された VPC で実行されます。このアーキテクチャに加え、運用セキュリティにまつわる Amazon のプラクティスにより、Amazon EKS は SOC 1、2、3、PCI-DSS、ISO、HIPAA 適格性といったコンプライアンス評価と基準をいくつも実現しています。こうしたコンプライアンス評価と基準は、デフォルトのエンベロープ暗号化の有無にかかわらず、すべての EKS クラスターに対して維持されています。

Amazon EKS では、エンベロープ暗号化はどのように機能しますか?

起動時に、クラスター API サーバーはランダムに生成されたデータと組み合わせて、シークレットシードからデータ暗号化キー (DEK) を生成します。同じく起動時に、API サーバーは KMS プラグインへの呼び出しを行い、AWS KMS からリモートのキー暗号化キー (KEK) を使用して DEK を暗号化します。これは、API サーバーの起動時と KEK のローテーション時に実行される 1 回限りの呼び

出しです。暗号化された DEK シードは API サーバーにキャッシュされます。API サーバーは、そのキャッシュした DEK シードを使用して、キー導出関数 (KDF) に基づいて 1 回限り使用される DEK を別途生成します。こうして生成した各 DEK を Kubernetes リソースごとに 1 回のみ使用して暗号化し、その後 etcd に保存します。

AWS KMS 統合のヘルスと通常の機能性を検証するために、API サーバーから追加で呼び出しが行われることに留意してください。こうした追加のヘルスチェックは、AWS CloudTrail に表示されます。

この機能を EKS クラスターで動作させるために何かを実行したり、アクセス許可を変更したりする必要はありますか?

いいえ。特に何もする必要はありません。Amazon EKS のエンベロープ暗号化は今ではデフォルトの設定となり、Kubernetes バージョン 1.28 以降を実行しているすべてのクラスターで有効になっています。AWS KMS 統合は、AWS 側で管理する Kubernetes API サーバーによって確立されます。つまり、クラスターで KMS 暗号化を使い始めるためにアクセス許可を設定する必要はありません。

クラスターでデフォルトのエンベロープ暗号化が有効になっているかどうかを確認するには、どうすればよいですか?

独自の CMK の使用に切り替えると、クラスターに関連付けられている KMS キーの ARN が表示されます。また、クラスターの CMK の使用に関連する AWS CloudTrail イベントログを確認することもできます。

クラスターが AWS 所有のキーを使用している場合には、その情報が EKS コンソールに詳しく表示されます (キーの ARN を除く)。

Amazon EKS でデフォルトのエンベロープ暗号化に使用される AWS 所有のキーに AWS からアクセスできますか?

いいえ。AWS では Amazon EKS のセキュリティ管理は厳しく、etcd データベース内のデータの保護に使用されるプレーンテキストの暗号化キーには誰もアクセスできません。こうしたセキュリティ対策は、AWS 所有の KMS キーにも適用されます。

デフォルトのエンベロープ暗号化は既存の EKS クラスターで有効になっていますか?

Kubernetes バージョン 1.28 以降で Amazon EKS クラスターを実行している場合は、すべての Kubernetes API データに対してエンベロープ暗号化が有効になっています。既存のクラスターの場合、Amazon EKS は `eks:kms-storage-migrator` RBAC ClusterRole を使用して、以前に etcd でエンベロープ暗号化されていなかったデータをこの新しい暗号化状態に移行します。

これは、EKS クラスターに対してシークレットのエンベロープ暗号化を既に有効にしていた場合にはどうなりますか?

Kubernetes シークレットをエンベロープ暗号化するために使用されたカスタマーマネージドキー (CMK) が KMS に既にある場合は、その同じキーがクラスター内のすべての Kubernetes API データタイプをエンベロープ暗号化するための KEK として使用されます。

`EncryptionConfig` の `resources` フィールドを設定する必要がありますか?

いいえ。`EncryptionConfig` の `resources` フィールドは非推奨となっています。Kubernetes バージョン 1.28 以降を実行しているクラスターでは、Amazon EKS はデフォルトですべての Kubernetes API データをエンベロープ暗号化で暗号化します。このため、どのリソースが暗号化されるかという点において `resources` フィールドが影響を及ぼすことはなくなりました。Amazon EKS は、引き続き `resources` フィールドを受け入れ、下位互換性を確保します。ただし、暗号化の対象範囲には影響しません。

`CreateCluster` または `AssociateEncryptionConfig` リクエストで、`resources` を省略したり、null または空のリストを渡したりできるようになりました。以前は、これを行うと `InvalidParameterException` が返されていました。現在 Amazon EKS はリクエストを受け入れ、当該フィールドをデフォルト値である `["secrets"]` として処理します。そのため、API レスポンスでは引き続き `["secrets"]` が返され、以前の API 契約との整合性が保たれます。

`resources` を設定する場合でも、値はやはり `["secrets"]` である必要があります。また、`EncryptionConfig` を指定する際には、`provider` (KMS キー) を指定する必要があります。

デフォルトのエンベロープ暗号化を有効にして EKS クラスターを実行すると、追加でコストが発生しますか?

デフォルトのエンベロープ暗号化に Amazon Web Services 所有のキーを使用している場合、マネージド Kubernetes コントロールプレーンに関連して追加でコストが発生することはありません。デフォルトでは、Kubernetes バージョン 1.28 以降を実行しているすべての EKS クラスター Amazon Web Services 所有のキー を使用します。ただし、独自の AWS KMS キーを使用した場合は、通常の KMS 料金 が適用されます。

独自の AWS KMS キーを使用してクラスター内の Kubernetes API データを暗号化するには、どのくらいのコストがかかりますか?

作成または KMS にインポートするカスタムキーを保しておくには、1 か月あたり 1 ドルの支払いが発生します。KMS では、暗号化と復号をリクエストすると課金されます。アカウントごとに 1 か月

あたり 20,000 件のリクエストという無料利用枠があり、この無料利用枠を超えると 1 か月あたり 10,000 件のリクエストごとに 0.03 ドルの支払いが発生します。これはアカウントで使用したすべての KMS に適用されるため、クラスターで独自の AWS KMS キーを使用した場合のコストは、このキーをアカウント内の他のクラスターや AWS リソースで使用した場合の影響を受けます。

カスタマーマネージドキー (CMK) を使用してシークレットだけでなくすべての Kubernetes API データをエンベロープ暗号化すると、KMS の料金は高くなりますか?

いいえ。KMS v2 で実装したことにより、AWS KMS に対する呼び出しの数が大幅に減ります。そのため、EKS クラスターで Kubernetes データが追加で暗号化または復号されたとしても、CMK に関連するコストは削減されます。

既に詳述したように、Kubernetes リソースの暗号化のために生成された DEK シードは、リモート KEK で暗号化されたうえで Kubernetes API サーバーのキャッシュにローカルに保存されます。暗号化された DEK シードが API サーバーのキャッシュ内にない場合、API サーバーは AWS KMS を呼び出して DEK シードを暗号化します。その後、API サーバーは暗号化された DEK シードをキャッシュして今後 KMS を呼び出さなくてもクラスターで使用できるようにします。同じく復号リクエストでも、API サーバーは最初の復号リクエストに対して AWS KMS を呼び出し、その後復号された DEK シードをキャッシュして今後の復号操作に使用します。

詳細については、GitHub で Kubernetes 拡張機能の「KEP-3299: KMS v2 Improvements」を参照してください。

複数の Amazon EKS クラスターで同じ CMK キーを使用できますか?

はい。キーを再度使用するには、キーの作成時に ARN をクラスターに関連付けることで、キーを同じリージョン内のクラスターにリンクします。ただし、複数の EKS クラスターで同じ CMK を使用している場合は、CMK が任意に無効にされないように必要な対策を講じる必要があります。そうしないと、無効になった CMK が複数の EKS クラスターに関連付けられて、キーによってはクラスターに幅広く影響が及びます。

デフォルトのエンベロープ暗号化が有効になった後で CMK が使用できなくなった場合、EKS クラスターはどうなりますか?

KMS キーを無効にすると、そのキーは暗号化オペレーションで使用できなくなります。既存の CMK にアクセスできないと、API サーバーは新規に作成された Kubernetes オブジェクトを暗号化して保持できないほか、以前に暗号化されて etcd に保存された Kubernetes オブジェクトを復号できなくなります。CMK が無効になると、クラスターはすぐに異常/パフォーマンス低下状態になり、関連付けられた CMK を再び有効にするまでサービスコミットメントを満たすことができなくなります。

CMK が無効になると、EKS クラスターがパフォーマンス低下状態になり、Kubernetes コントロールプレーンリソースを正常に復元するためには CMK が無効になってから 30 日以内に再び有効にする必要があるとの通知が届きます。

CMK が無効化/削除された場合の影響から EKS クラスターを保護するには、どうすればよいですか?

このような事態の発生から EKS クラスターを保護するには、キー管理者が最小特権の原則に基づく IAM ポリシーを使用して KMS キーオペレーションへのアクセスを管理し、EKS クラスターに関連付けられているキーが任意に無効化または削除されるリスクを軽減する必要があります。また、CMK の状態を通知する CloudWatch アラームを設定できます。

CMK を再び有効にすると、EKS クラスターが復元されますか?

EKS クラスターが正常に復元されるように、CMK が無効になってから 30 日以内に再び有効にすることを強くお勧めします。ただし、EKS クラスターが正常に復元されるかどうかは、クラスターが異常/パフォーマンス低下状態のときに自動 Kubernetes アップグレードが発生したために API が大きく変更されたかどうかによっても異なります。

CMK を無効にした後で、EKS クラスターが異常/パフォーマンス低下状態になるのはなぜですか?

EKS コントロールプレーンの API サーバーは、暗号化されて API サーバーのメモリにキャッシュされた DEK キーを使用して、作成/更新操作時にすべてのオブジェクトを暗号化してから etcd に保存します。etcd から既存のオブジェクトを取得するときに、API サーバーはキャッシュされたその同じ DEK キーを使用して、Kubernetes リソースオブジェクトを復号します。CMK を無効にしても、DEK キーが API サーバーのメモリにキャッシュされているため、API サーバーはすぐには影響を受けません。ただし、API サーバーインスタンスを再起動すると、キャッシュされた DEK が保持されないため、暗号化と復号の操作を行うには AWS KMS を呼び出す必要があります。CMK がない場合、このプロセスは KMS_KEY_DISABLED エラーコードで失敗し、API サーバーが正常に起動しなくなります。

CMK を削除すると、EKS クラスターはどうなりますか?

EKS クラスターに関連付けられている CMK キーを削除すると、ヘルスが低下して回復の見込みがなくなります。クラスターの CMK がないと、API サーバーは新規の Kubernetes オブジェクトを暗号化して保持できないほか、以前に暗号化されて etcd データベースに保存された Kubernetes オブジェクトを復号できなくなります。EKS クラスターの CMK キーを削除するのは、今後 EKS クラスターを使用する必要はないと確信できるときだけにしてください。

CMK が見つからない (KMS_KEY_NOT_FOUND) 場合やクラスターに関連付けられた CMK の許可が取り消されている (KMS_GRANT_REVOKED) 場合は、クラスターを回復できなくなるので注意して

ください。クラスターのヘルスとエラーコードの詳細については、「クラスターの正常性に関するよくある質問およびエラーコードと解決パス」を参照してください。

CMK を無効または削除したために EKS クラスターでパフォーマンス低下/異常が発生した場合でも課金されますか?

はい。CMK が無効になると EKS コントロールプレーンは使用できなくなりますが、EKS クラスターに割り当てられた専用のインフラストラクチャリソースはお客様が削除するまで AWS で引き続き稼働します。また、このような状況では Amazon のサービスコミットメントは適用されません。お客様が自発的にアクションを起こしたこと、あるいは何もアクションを起こさなかったことで、EKS クラスターの通常のヘルスとオペレーションが阻害されたためです。

CMK が無効であるために異常/パフォーマンス低下状態になったときでも、EKS クラスターを自動的にアップグレードできますか?

はい。ただし、クラスターで CMK が無効である場合、30 日以内であれば再び有効にできます。この 30 日の間、Kubernetes クラスターは自動的にアップグレードされません。ただし、この期間が過ぎても CMK を再び有効にしなかった場合、クラスターは次のバージョン (n+1) に自動的にアップグレードされます。これは標準のサポートであり、EKS の Kubernetes バージョンライフサイクルに従った動作です。

クラスターが影響を受けていることに気づいたら、無効になった CMK をすぐに再び有効にすることを強くお勧めします。なお、EKS はこうした影響を受けたクラスターを自動的にアップグレードしますが、正常に回復する保証はありません。特に、クラスターが複数回の自動アップグレードを経験している場合にはそうです。この場合、Kubernetes API に変更が加えられ、API サーバーのブートストラッププロセスが予期しない動作をする可能性があるからです。

KMS キーのエイリアスを使用できますか?

はい。Amazon EKS は、KMS キーのエイリアスの使用をサポートしています。エイリアスは、Amazon Web Services KMS キーをわかりやすくした名前です。例えば、エイリアスを使用すると、KMS キーを **1234abcd-12ab-34cd-56ef-1234567890ab** ではなく my-key として参照できます。

独自の Kubernetes バックアップソリューションを使用して、クラスターリソースをバックアップおよび復元できますか?

はい。Kubernetes クラスターのディザスタリカバリ、データ移行、データ保護に Kubernetes バックアップソリューション (Velero など) を使用できます。API サーバーを介してクラスターリソースにアクセスする Kubernetes バックアップソリューションを実行している場合、アプリケーション

が取得するデータは復号されてからクライアントに送られます。そのため、別の Kubernetes クラスターにクラスターリソースを復元できます。

Amazon EKS における Kubernetes RBAC の強化

Kubernetes ロールベースのアクセスコントロール (RBAC) は、クラスター内で各 ID が実行できるアクションを制御します。CSI ドライバーや DaemonSets としてインストールされるその他のアドオンなど、多くのクラスターコンポーネントは、機能するために広範な権限を必要とします。これらのアクセス許可を見直し、範囲を明確にすることで、意図しないアクセスが発生する可能性を低減できます。

このトピックでは、一般的なクラスターコンポーネントのアクセス許可に関する考慮事項と、推奨される制御策について説明します。

DaemonSet サービスアカウントのアクセス許可

DaemonSet Pod はクラスター内のすべてのノードで実行されるため、サービスアカウントトークンと、それらのトークンによって付与される RBAC アクセス許可はすべてのノードに存在します。

ノード上の不正なプロセスが、DaemonSet Pod を含む、同じノードで実行されている他の Pod のサービスアカウントトークンにアクセスできる可能性があります。aemonSet サービスアカウントに付与される RBAC アクセス許可は、クラスター内のすべてのノードで同じです。

DaemonSets として一般的にデプロイされるコンポーネントは、次のとおりです。

- CSI ノードドライバー (`ebs-csi-node`、`efs-csi-node`、`mountpoint-s3-csi-node`)
- Amazon VPC CNI プラグイン (`aws-node`)
- `kube-proxy`

DaemonSet Pod が EKS Pod Identity または IAM Roles for Service Accounts (IRSA) を介して AWS IAM 認証情報を持っている場合、同じノード上のコンテナ外からアクセスするプロセスも、これらの認証情報にアクセスできる可能性があります。これにより、影響の範囲が Kubernetes RBAC を超えて、DaemonSet の IAM ロールに付与された AWS API アクセス許可にまで拡張されます。

> **Important**
>
> アクセス許可を確認するときは、Kubernetes RBAC アクセス許可とすべての DaemonSet サービスアカウントの IAM アクセス許可を、クラスター内のすべてのノードからアクセス可能として扱います。

CSI ドライバー RBAC スコープ

CSI ドライバーは、ノード、永続ボリューム、ストレージ API とやり取りするため、一般的に広範な RBAC アクセス許可を保持しています。

ノードオブジェクトのアクセス許可

CSI ドライバーでは、テイントの削除やその他のノード管理タスクなどの機能をサポートするために、ノードオブジェクトを変更する際に RBAC アクセス許可が必要になる場合があります。Kubernetes RBAC の制限により、これらのアクセス許可は、ドライバーが実行されているローカルノードだけでなく、クラスター内の すべて のノードオブジェクトに適用されます。

EBS CSI ドライバーの場合、Helm チャートには、`ebs-csi-node` サービスアカウントからノード変更許可を削除するパラメータ (`node.serviceAccount.disableMutation`) が表示されます。これを有効にすると、テイント削除機能は無効になります。

サービスアカウントトークンの漏洩

CSI ドライバー Pod は、認証に投影されたサービスアカウントトークンを使用する場合があります。不正なプロセスがコンテナ外部にアクセスしたノードでは、これらのトークンはコンテナファイルシステムまたは kubelet API からアクセスできる場合があります。サービスアカウントが EKS Pod Identity または IRSA を介して IAM ロールにも関連付けられている場合、公開されたトークンを使用して AWS IAM 認証情報を取得できます。

推奨コントロール

RBAC を最小特権にスコープ

- CSI ドライバーと DaemonSet サービスアカウントにバインドされている ClusterRoles を確認します。ワークロードに不要なアクセス許可を削除します。
- EBS CSI ドライバーでテイント削除機能を使用しない場合は、`node.serviceAccount.disableMutation` を `true` に設定します。
- `kubectl auth can-i --list --as=system:serviceaccount:NAMESPACE:SERVICE_ACCOUNT` を使用して、有効なアクセス許可を監査します。

Pod セキュリティ標準を適用

組み込みの Pod Security Admission Controller またはポリシーエンジンを使用して、Kubernetes Pod Security Standards を適用します。少なくとも、`baseline` プロファイルをクラスター全体に適用

し、restricted プロファイルをワークロード名前空間に適用します。これにより、システム名前空間外で特権コンテナを作成する機能が制限されます。

ネットワークポリシーの使用

ネットワークポリシーを適用して、CSI ドライバーと DaemonSet Pod からの出力を、必要なエンドポイント (Kubernetes API サーバーと AWS サービスエンドポイントなど) のみに制限します。これにより、可能なアクションの範囲が縮小されます。

RBAC アクティビティの監視

Kubernetes 監査ログを有効にし、DaemonSet サービスアカウントからの予期しない API コールを監視します。以下を探します。

- CSI ドライバーサービスアカウントからのノードの変更
- システム名前空間でのポッドの作成
- シークレットに対する異常な get または list 呼び出し

詳細については、the section called “コントロールプレーンのログ” を参照してください。

Amazon EKS Auto Mode におけるセキュリティに関する考慮事項

このトピックでは、Amazon EKS Auto Mode のセキュリティアーキテクチャ、コントロール、ベストプラクティスについて説明します。組織がコンテナ化されたアプリケーションを大規模にデプロイするにつれて、強力なセキュリティ体制を維持することがますます複雑になります。EKS Auto Mode は、自動化されたセキュリティコントロールを実装し、AWS セキュリティサービスと統合して、クラスターのインフラストラクチャ、ワークロード、データを保護します。EKS Auto Mode は、強制ノードライフサイクル管理や自動パッチデプロイなどの組み込みセキュリティ機能を通じて、運用オーバーヘッドを削減しながらセキュリティのベストプラクティスを維持するのに役立ちます。

このトピックを進める前に、基本的な EKS Auto Mode の概念を理解していること、およびクラスターで EKS Auto Mode を有効にするための前提条件を確認しているようにしてください。Amazon EKS セキュリティに関する一般的な情報については、「セキュリティ」を参照してください。

Amazon EKS Auto Mode は、Amazon EKS の既存のセキュリティ基盤に基づいて構築されると同時に、EC2 マネージドインスタンスに対する追加の自動セキュリティコントロールを導入します。

API セキュリティと認証

Amazon EKS Auto Mode は、AWS プラットフォームのセキュリティメカニズムを使用して、Amazon EKS API への呼び出しを保護および認証します。

- Kubernetes API へのアクセスは、AWS IAM ID と統合された EKS アクセスエントリを介して保護されます。
  - 詳細については、「the section called “アクセスエントリ”」を参照してください。
- お客様は、EKS アクセスエントリの設定を通じて、Kubernetes API エンドポイントへのきめ細かなアクセスコントロールを実装できます。

ネットワークセキュリティ

Amazon EKS Auto Mode は、複数のネットワークセキュリティレイヤーをサポートしています。

- VPC 統合
  - Amazon Virtual Private Cloud (VPC) 内で動作
  - カスタムの VPC 設定とサブネットレイアウトをサポート
  - クラスターコンポーネント間のプライベートネットワーキングを有効化
  - 詳細については、「Amazon Virtual Private Cloud のセキュリティの責任の管理」を参照してください
- ネットワークポリシー
  - Kubernetes ネットワークポリシーのネイティブサポート
  - 詳細なネットワークトラフィックルールを定義する機能
  - 詳細については、the section called “Kubernetes ポリシー”を参照してください。

EC2 マネージドインスタンスのセキュリティ

Amazon EKS Auto Mode は、次のセキュリティコントロールを使用して EC2 マネージドインスタンスを操作します。

EC2 セキュリティ

- EC2 マネージドインスタンスは、Amazon EC2 のセキュリティ機能を維持します。

- EC2 マネージドインスタンスの詳細については、「Security in Amazon EC2」を参照してください。

インスタンスのライフサイクル管理

EKS オートモードによって運用される EC2 マネージドインスタンスの最大存続期間は 21 日です。Amazon EKS Auto Mode は、この存続期間を超えたインスタンスを自動的に終了します。この存続期間制限は、設定のドリフトを防ぎ、セキュリティ体制を維持するのに役立ちます。

データ保護

- Amazon EC2 インスタンスストレージは暗号化されており、インスタンスに直接アタッチされたストレージになっています。詳細については、「Amazon EC2 でのデータ保護」を参照してください。
- EKS Auto Mode は、ルートボリュームやデータボリュームなど、作成時に EC2 インスタンスにアタッチされたボリュームを管理します。EKS Auto Mode は、Kubernetes 永続ストレージ機能を使用して作成された EBS ボリュームを完全には管理しません。

パッチ管理

- Amazon EKS Auto Mode は、マネージドインスタンスにパッチを自動的に適用します。
- パッチには以下が含まれます。
  - オペレーティングシステムの更新
  - セキュリティパッチ
  - Amazon EKS Auto Mode コンポーネント

> **Note**
>
> これらのインスタンスで実行されているワークロードを保護および更新するのはお客様です。

アクセス制御

- インスタンスへの直接アクセスは制限されています。
  - SSH アクセスは使用できません。

  - AWS Systems Manager Session Manager (SSM) アクセスは使用できません。
- 管理オペレーションは、Amazon EKS API と Kubernetes API を介して実行されます。

自動リソース管理

Amazon EKS Auto Mode は、Kubernetes 永続ストレージ機能を使用して作成された Amazon Elastic Block Store (Amazon EBS) ボリュームを完全には管理しません。EKS Auto Mode は Elastic Load Balancer (ELB) も管理しません。Amazon EKS Auto Mode は、これらのリソースのルーチンタスクを自動化します。

ストレージセキュリティ

- AWS では、Kubernetes 永続ストレージ機能によってプロビジョニングされた EBS ボリュームの暗号化を有効にすることをお勧めします。詳細については、「the section called “StorageClass を作成する”」を参照してください。
- AWS KMS を使用した保管時の暗号化
- 作成した新しい EBS ボリュームとスナップショットコピーの暗号化を強制するように AWS アカウントを設定できます。詳細については、「Amazon EBS ユーザーガイド」の「Enable Amazon EBS encryption by default」を参照してください。
- 詳細については、「Security in Amazon EBS」を参照してください。

ロードバランサーのセキュリティ

- Elastic Load Balancer の自動設定
- AWS Certificate Manager 統合による SSL/TLS 証明書管理
- ロードバランサーアクセスコントロールのセキュリティグループ自動化
- 詳細については、「Security in Elastic Load Balancing」を参照してください。

セキュリティのベストプラクティス

次のセクションでは、Amazon EKS Auto Mode におけるセキュリティのベストプラクティスについて説明します。

- AWS IAM ポリシーと EKS アクセスエントリを定期的に確認します。

- NodeClass でサブネットとセキュリティグループのタグセレクターを使用する場合、EKS Auto Mode は、一致するタグを持つ任意のサブネットまたはセキュリティグループ (後からタグ付けされたリソースを含む) を自動的に選択します。Auto Mode が使用できるサブネットとセキュリティグループを厳密に制御する必要がある場合は、タグに基づく選択機能だけに依存しないでください。代わりに、Amazon EKS Auto Mode クラスターの IAM ロールのスコープを設定して、ec2:CreateNetworkInterface が特定のサブネットとセキュリティグループに限定されるようにします。 AmazonEKSNetworkingPolicy マネージドポリシーをそのままアタッチするのではなく、ec2:CreateNetworkInterface を目的のサブネットとセキュリティグループのリソースに制限するカスタマー管理ポリシーを使用するか、他のすべてのリソースに対する当該アクションを拒否する追加のポリシーをアタッチします。詳細については、「IAM ユーザーガイド」の「IAM JSON ポリシー要素 NotResource」を参照してください。
- ワークロードに最小特権のアクセスパターンを実装します。
- AWS CloudTrail と Amazon CloudWatch を使用してクラスターのアクティビティをモニタリングします。詳細については、「the section called “ AWS CloudTrail”」および「the section called “Amazon CloudWatch”」を参照してください。
- セキュリティ体制の評価には AWS Security Hub を使用します。
- ワークロードに適したポッドセキュリティ標準を実装します。

EKS 機能のセキュリティに関する考慮事項

このトピックでは、EKS 機能のセキュリティを考慮するうえで重要な事項について説明します。具体的には、IAM ロール設定、Kubernetes アクセス許可、マルチクラスターデプロイとクロスアカウント AWS リソース管理のアーキテクチャパターンなどです。

EKS 機能では、IAM ロール、EKS アクセスエントリ、Kubernetes RBAC を組み合わせることにより、AWS サービスとクラスター内 Kubernetes リソースに安全にアクセスし、AWS CodeConnections、AWS Secrets Manager、およびその他の AWS サービスと統合できます。

機能 IAM ロール

機能を作成するときは、自分の代わりに EKS がアクションを実行するように IAM 機能ロールを指定できます。このロールの要件は次のとおりです。

- クラスターおよび機能リソースと同じ AWS アカウントに存在していること
- 信頼ポリシーにより、capabilities.eks.amazonaws.com サービスプリンシパルがロールを引き受けることを許可すること

- 要件に応じて、機能タイプとユースケースに適した IAM アクセス許可を付与すること 必要な IAM アクセス許可の詳細については、「the section called “ AWS CodeConnections”」、「the section called “ AWS 秘密マネジャー”」、「the section called “アクセス許可の設定”」を参照してください。

ここでのベストプラクティスは、特定のユースケースに必要な権限の範囲を考慮したうえで、要件を満たすために必要なアクセス許可のみを付与することです。例えば、Kube Resource Orchestrator に EKS 機能を使用する場合には IAM アクセス許可は必要ありませんが、Kubernetes 用 AWS コントローラーに EKS 機能を使用する場合には 1 つ以上の AWS サービスへのフルアクセスを許可できます。

> **Important**
>
> ユースケースによっては広範な管理者権限の使用を許可できますが、特定のユースケースに必要な最小限の IAM アクセス許可のみを付与して最小特権の原則に従うと、ワイルドカードアクセス許可を使用するのではなく、ARN と条件キーを使用して、特定のリソースへのアクセスを制限できます。

機能 IAM ロールの作成と設定の詳細については、「the section called “機能 IAM ロール”」を参照してください。

EKS アクセスエントリ

IAM ロールと共に機能を作成すると、Amazon EKS がそのロールのアクセスエントリをクラスターに自動的に作成します。このアクセスエントリにより、動作に必要なベースラインとなる Kubernetes アクセス許可が機能に付与されます。

> **Note**
>
> アクセスエントリが作成されるクラスターは、機能が作成されているクラスターです。Argo CD でリモートクラスターにデプロイする場合は、Argo CD 機能がアプリケーションをデプロイおよび管理できるように、そのクラスターにアクセスエントリを作成して適切なアクセス許可を付与する必要があります。

アクセスエントリの内容は以下のとおりです。

- IAM ロール ARN をプリンシパルとします。
- 機能固有のアクセスエントリポリシーでベースライン Kubernetes アクセス許可を付与します。
- 機能タイプに基づいて適切な範囲 (クラスター全体または名前空間範囲内) を指定します。

> **Note**
>
> Argo CD の場合、名前空間範囲のアクセス許可は機能設定に指定された名前空間に付与されます (デフォルトは `argocd`)。

機能別のデフォルトアクセスエントリポリシー

機能タイプごとに必要なアクセス許可が機能ロールに付与され、次のように異なるデフォルトアクセスエントリポリシーが設定されます。

kro

- `arn:aws:eks::aws:cluster-access-policy/AmazonEKSKROPolicy` (クラスター範囲)

  ResourceGraphDefinitions を監視および管理するアクセス許可を付与し、RGD に定義されたカスタムリソースのインスタンスを作成します。

ACK

- `arn:aws:eks::aws:cluster-access-policy/AmazonEKSACKPolicy` (クラスター範囲)

  すべての名前空間で ACK カスタムリソースを作成、読み取り、更新、削除できるアクセス許可を付与します。

Argo CD

- `arn:aws:eks::aws:cluster-access-policy/AmazonEKSArgoCDClusterPolicy` (クラスター範囲)

  Argo CD がリソースを検出し、クラスター範囲のオブジェクトを管理するために必要なクラスターレベルのアクセス許可を付与します。
- `arn:aws:eks::aws:cluster-access-policy/AmazonEKSArgoCDPolicy` (名前空間範囲)

  Argo CD がアプリケーションをデプロイおよび管理するために必要な名前空間レベルのアクセス許可を付与します。機能設定に指定された名前空間に限定されます (デフォルトは `argocd`)。

詳細については、「the section called “アクセスポリシーを確認する”」を参照してください。

その他の Kubernetes アクセス許可

機能によっては、デフォルトのアクセスエントリポリシーの範囲を超える Kubernetes アクセス許可が追加で必要になる場合があります。こうしたアクセス許可を付与するには、次のいずれかを使用します。

- アクセスエントリポリシー: 追加の管理ポリシーをアクセスエントリに関連付けます。
- Kubernetes RBAC: 機能の Kubernetes ユーザーに対して `Role` または `ClusterRole` バインディングを作成します。

ACK シークレット読み取りアクセス許可

ACK コントローラーによっては、データベースパスワードなどの機密データを取得するために Kubernetes シークレットを読み取る必要があります。次の ACK コントローラーでは、シークレット読み取りアクセスが必要です。

- `acm`, `acmpca`, `documentdb`, `memorydb`, `mq`, `rds`, `secretsmanager`

シークレット読み取りアクセス許可を付与するには:

1. `arn:aws:eks::aws:cluster-access-policy/AmazonEKSSecretReaderPolicy` アクセスエントリポリシーを機能のアクセスエントリに関連付けます。
2. ACK リソースがシークレットを参照するか、クラスター全体のアクセスを許可する、特定の名前空間にポリシーの範囲を限定します。

> **Important**
>
> シークレット読み取りアクセス許可の範囲は、アクセスエントリポリシーを関連付けるときに指定する名前空間に限定されます。これにより、機能がどのシークレットにアクセスできるかを制限できます。

kro による任意のリソースアクセス許可

デフォルトでは、kro は ResourceGraphDefinitions (RGDs) とそのインスタンスを監視および管理できます。Deployments、Services、ConfigMaps などの構成済みリソースを管理するには、追加の Kubernetes アクセス許可を設定します。

kro にリソースを作成するためのアクセス許可を付与するには:

オプション 1: アクセスエントリポリシー

`AmazonEKSAdminPolicy` や `AmazonEKSEditPolicy` などの事前定義済みのアクセスエントリポリシーを機能のアクセスエントリに関連付けます。

オプション 2: Kubernetes RBAC

`ClusterRoleBinding` を作成して、機能の Kubernetes ユーザーに必要なアクセス許可を付与します。

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: kro-cluster-admin
subjects:
- kind: User
  name: arn:aws:sts::111122223333:assumed-role/my-kro-role/KRO
  apiGroup: rbac.authorization.k8s.io
roleRef:
  kind: ClusterRole
  name: cluster-admin
  apiGroup: rbac.authorization.k8s.io
```

> **Note**
>
> kro の Kubernetes ユーザー名は、`arn:aws:sts::ACCOUNT_ID:assumed-role/ROLE_NAME/KRO` というパターンに従います。
> セッション名 `/KRO` (大文字) は、EKS kro 機能によって自動的に設定されます。

機能で必要な IAM アクセス許可

kro (Kube Resource Orchestrator)

必須の IAM アクセス許可はありません。ポリシーをアタッチせずに機能ロールを作成できます。kro に必要なのは Kubernetes RBAC アクセス許可のみです。

ACK (Kubernetes 用 AWS コントローラー)

ACK が作成および管理する AWS リソースを管理するためのアクセス許可が必要です。要件に基づいて、特定のサービス、アクション、リソースに対するアクセス許可の範囲を限定する必要があります。IAM ロールセレクターによる本番稼働のベストプラクティスなど、ACK アクセス許可の設定の詳細については、「the section called “アクセス許可の設定”」を参照してください。

Argo CD

デフォルトでは、必須の IAM アクセス許可はありません。以下のものには、オプションでアクセス許可が必要になる場合があります。

- AWS Secrets Manager: Git リポジトリ認証情報を Secrets Manager に保存する場合
- AWS CodeConnections: Git リポジトリの認証に CodeConnections を使用する場合
- Amazon ECR: Amazon ECR に OCI 形式で保存された Helm チャートを使用する場合

セキュリティのベストプラクティス

IAM 最小特権

ユースケースに必要なアクセス許可のみを機能リソースに付与します。これは、機能には必要に応じて広範な管理アクセス許可を付与できないという意味ではありません。このような場合は、そうしたリソースに対するアクセスを適切に管理する必要があります。

機能ロール:

- ACK: 可能であれば、ユースケースと要件に基づいて、チームに必要な特定の AWS サービスとリソースに IAM アクセス許可を制限します。
- Argo CD: 特定の Git リポジトリと Kubernetes 名前空間に対するアクセスを制限します。
- kro: 信頼ポリシーに機能ロールは必要ですが、IAM アクセス許可は必要ありません (クラスター RBAC のみを使用します)。

例: "Resource": "*" ではなく、特定のリソースやリソースグループに対するパターンを指定します。

```
"Resource": [
  "arn:aws:s3:::my-app-*",
  "arn:aws:rds:us-west-2:111122223333:db:prod-*"
]
```

IAM 条件キーを使用して、アクセスをさらに制限します。

```
"Condition": {
  "StringEquals": {
    "aws:ResourceTag/Environment": "production"
  }
}
```

IAM 設定の詳細については、各機能の考慮事項に関するセクションを参照してください。

Argo CD シークレットの名前空間の分離

マネージド Argo CD 機能は、自身に設定された名前空間内のすべての Kubernetes シークレットにアクセスできます (デフォルト: argocd)。最適なセキュリティ体制を維持するには、名前空間の分離に関する以下のプラクティスに従ってください。

- Argo CD 関連のシークレットのみを Argo CD 名前空間内に保持します。
- 無関係なアプリケーションシークレットを Argo CD と同じ名前空間に保存しないようにします。
- Argo CD オペレーションに必要ないアプリケーションシークレットには別の名前空間を使用します。

この分離により、Argo CD のシークレットアクセスは、Git リポジトリの認証やその他の Argo CD に固有のオペレーションに必要な認証情報のみに制限されます。

Kubernetes RBAC

機能リソースを作成および管理できるユーザーおよびサービスアカウントを制御します。ここでのベストプラクティスは、適切な RBAC ポリシーに沿って専用の名前空間に機能リソースをデプロイすることです。

例: ACK で RBAC ロールを使用して、app-team 名前空間で S3 バケットリソースを管理できるようにします。

```
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  name: ack-s3-manager
  namespace: app-team
rules:
- apiGroups: ["s3.services.k8s.aws"]
  resources: ["buckets"]
  verbs: ["get", "list", "create", "update", "delete"]
```

監査ログ

CloudTrail: すべての EKS Capability API オペレーション (作成、更新、削除) が AWS CloudTrail に記録されます。

CloudTrail ログ記録では以下を追跡できます。

- 誰が機能を作成または変更したか
- いつ機能設定が変更されたか
- どのような機能ロールが使用されているか

ネットワークアクセスおよび VPC エンドポイント

プライベート Argo CD API アクセス

Argo CD API サーバーへのアクセスを制限するには、ホストされた Argo CD エンドポイントに 1 つ以上の VPC エンドポイントを関連付けます。これにより、パブリックインターネットを経由することなく、VPC 内からのプライベート接続が可能になります。VPC エンドポイントは、Argo CD ウェブ UI と Argo CD API (CLI アクセスを含む) の両方へのアクセスを提供します。

> **Note**
>
> ホストされた Argo CD API エンドポイントに接続されている VPC エンドポイント (eks-capabilities.*region*.amazonaws.com を使用) は、VPC エンドポイントポリシーをサポートしていません。

プライベートクラスターにデプロイする

Argo CD 機能ではアプリケーションを完全にプライベートな EKS クラスターにデプロイできるため、VPC ピアリングや複雑なネットワーク設定が不要になり、運用面で大きなメリットが得られます。ただし、このアーキテクチャを設計するときは、Argo CD が Git リポジトリ (パブリックの場合あり) から設定をプルして、プライベートクラスターに適用するようにすることを検討してください。

以下の点を確認します。

- 機密のワークロードにプライベートな Git リポジトリを使用している
- Git リポジトリに適切なアクセスコントロールと認証を実装している
- マージする前にプルリクエストを通じて変更を確認のうえ承認している
- Argo CD の同期期間を使用してデプロイのタイミングを制御することを検討している
- Argo CD 監査ログをモニタリングして不正な設定変更がないか確認している

コンプライアンス

EKS 機能は、完全マネージド型で、Amazon EKS のコンプライアンス認定を受けています。

現在のコンプライアンス情報については、「コンプライアンスプログラムによる対象範囲内の AWS サービス」を参照してください。

次のステップ

- the section called “アクセス許可の設定” - ACK 用に IAM アクセス許可を設定する
- the section called “アクセス許可の設定” - kro 用に Kubernetes RBAC を設定する
- the section called “アクセス許可の設定” - Argo CD 用にアイデンティティセンター統合を設定する
- the section called “機能をトラブルシューティングする” - セキュリティとアクセス許可の問題をトラブルシューティングする

Amazon EKS の Identity and Access Management

AWS Identity and Access Management (IAM) は、管理者が AWS リソースへのアクセスを安全に制御するのに役立つ AWS のサービスです。IAM 管理者は、誰の (サインイン) を認証し、誰による

Amazon EKS リソースの使用を承認する (アクセス許可を付与する) かを制御します。IAM は、AWS のサービスで追加料金は発生しません。

オーディエンス

AWS Identity and Access Management (IAM) の使用方法は、Amazon EKS で行う作業によって異なります。

サービスユーザー – ジョブを実行するために Amazon EKS サービスを使用する場合は、管理者から必要なアクセス許可と認証情報が与えられます。さらに多くの Amazon EKS 機能を使用して作業を行う場合は、追加のアクセス許可が必要になることがあります。アクセスの管理方法を理解すると、管理者から適切なアクセス許可をリクエストするのに役に立ちます。Amazon EKS の機能にアクセスできない場合は、「the section called “トラブルシューティング”」を参照してください。

サービス管理者 – 社内の Amazon EKS リソースを管理しているユーザーには、通常、Amazon EKS への完全なアクセス権限があります。サービスのユーザーがどの Amazon EKS 機能やリソースにアクセスするかを決めるのは管理者の仕事です。その後、IAM 管理者にリクエストを送信して、サービスユーザーの権限を変更する必要があります。このページの情報を確認して、IAM の基本概念を理解してください。企業が Amazon EKS で IAM を利用する方法については、「the section called “IAM および Amazon EKS”」を参照してください。

IAM 管理者 – IAM 管理者には、Amazon EKS へのアクセスを管理するポリシーの作成方法を、詳細に把握する必要が生じる場合があります。IAM で使用可能な、Amazon EKS アイデンティティベースのポリシーの例を確認するには、「the section called “アイデンティティベースのポリシー”」を参照してください。

アイデンティティを使用した認証

認証とは、アイデンティティ認証情報を使用して AWS にサインインする方法です。ユーザーは、AWS アカウントのルートユーザーもしくは IAM ユーザーとして、または IAM ロールを引き受けることによって、認証を受ける (AWS にサインインする) 必要があります。

ID ソースから提供された認証情報を使用して、フェデレーティッドアイデンティティとして AWS にサインインできます。AWSフェデレーティッドアイデンティティの例としては、IAM アイデンティティセンター (IAM アイデンティティセンター) ユーザー、貴社のシングルサインオン認証、Google または Facebook の認証情報などがあります。フェデレーティッド ID としてサインインする場合、IAM ロールを使用して、前もって管理者により ID フェデレーションが設定されています。フェデレーションを使用して AWSにアクセスする場合、間接的にロールを引き受けることになります。

ユーザーのタイプに応じて、AWS マネジメントコンソール または AWS アクセスポータルにサインインできます。AWS へのサインインの詳細については、AWS サインインユーザーガイドの「AWS アカウントにサインインする方法」を参照してください。

プログラムを使用して AWS にアクセスする場合、AWSは Software Development Kit (SDK) とコマンドラインインターフェイス (CLI) を提供し、認証情報を使用してリクエストに暗号で署名します。AWS ツールを使用しない場合は、リクエストに自分で署名する必要があります。リクエストに署名する推奨方法の使用については、「IAM ユーザーガイド」の「AWS API リクエストの署名」を参照してください。

使用する認証方法を問わず、追加セキュリティ情報の提供をリクエストされる場合もあります。例えば、AWS は、アカウントのセキュリティを強化するために多要素認証 (MFA) を使用することをお勧めします。詳細については、「AWS IAM Identity Center ユーザーガイド」の「多要素認証 (MFA)」および「IAM ユーザーガイド」の「AWS での多要素認証 (MFA) の使用」を参照してください。

AWS アカウントのルートユーザー

AWS アカウントを作成する場合は、このアカウントのすべての AWS サービスとリソースに対して完全なアクセス権を持つ 1 つのサインインアイデンティティから始めます。このアイデンティティは AWS アカウントルートユーザーと呼ばれ、アカウントの作成に使用したメールアドレスとパスワードでのサインインによりアクセスされます。日常的なタスクには、ルートユーザーを使用しないことを強くお勧めします。ルートユーザー資格情報は保護し、ルートユーザーでしか実行できないタスクを実行するときに使用します。ルートユーザーとしてサインインする必要があるタスクの完全なリストについては、「IAM ユーザーガイド」の「ルートユーザー認証情報が必要なタスク」を参照してください。

IAM ユーザーとグループ

https://docs.aws.amazon.com/IAM/latest/UserGuide/id_users.htmlIAM ユーザーは、単一のユーザーまたはアプリケーションに対する特定の許可を持つ AWS アカウント内のアイデンティティです。可能であれば、パスワードやアクセスキーなどの長期的な認証情報を保有する IAM ユーザーを作成する代わりに、一時的な認証情報を使用することをお勧めします。ただし、IAM ユーザーでの長期的な認証情報が必要な特定のユースケースがある場合は、アクセスキーをローテーションすることをお勧めします。詳細については、「IAM ユーザーガイド」の「長期的な認証情報を必要とするユースケースのためにアクセスキーを定期的にローテーションする」を参照してください。

IAM グループは、IAM ユーザーのコレクションを指定するアイデンティティです。グループとしてサインインすることはできません。グループを使用して、複数のユーザーに対して一度に権限を指定できます。多数のユーザーグループがある場合、グループを使用することで権限の管理が簡単になり

ます。例えば、IAMAdmins という名前のグループを設定して、そのグループに IAM リソースを管理する許可を与えることができます。

ユーザーは、ロールとは異なります。ユーザーは 1 人の人または 1 つのアプリケーションに一意に関連付けられますが、ロールはそれを必要とする任意の人が引き受けるようになっています。ユーザーには永続的な長期の認証情報がありますが、ロールでは一時的な認証情報が提供されます。詳細については、「IAM ユーザーガイド」の「(ロールではなく) IAM ユーザーの作成が適している場合」を参照してください。

IAM ロール

https://docs.aws.amazon.com/IAM/latest/UserGuide/id_roles.htmlIAM ロールは、特定のアクセス許可を持つ、AWS アカウント内のアイデンティティです。これは IAM ユーザーに似ていますが、特定のユーザーに関連付けられていません。ロールを切り替えることによって、AWS マネジメントコンソールで IAM ロールを一時的に引き受けることができます。ロールを引き受けるには、AWS CLI または AWS API オペレーションを呼び出すか、カスタム URL を使用します。ロールを使用する方法の詳細については、「IAM ユーザーガイド」の「IAM ロールの使用」を参照してください。

IAM ロールと一時的な認証情報は、次の状況で役立ちます:

- フェデレーションユーザーアクセス – フェデレーティッド ID に許可を割り当てるには、ロールを作成してそのロールの許可を定義します。フェデレーテッド ID が認証されると、その ID はロールに関連付けられ、ロールで定義されている許可が付与されます。フェデレーションの詳細については、「IAM ユーザーガイド」の「サードパーティーアイデンティティプロバイダー向けロールの作成」を参照してください。IAM Identity Center を使用する場合は、許可セットを設定します。アイデンティティが認証後にアクセスできるものを制御するため、IAM Identity Center は、権限セットを IAM のロールに関連付けます。アクセス許可セットの詳細については、「AWS IAM Identity Center ユーザーガイド」の「アクセス許可セット」を参照してください。
- 一時的な IAM ユーザー権限 - IAM ユーザーまたはロールは、特定のタスクに対して複数の異なる権限を一時的に IAM ロールで引き受けることができます。
- クロスアカウントアクセス - IAM ロールを使用して、自分のアカウントのリソースにアクセスすることを、別のアカウントの人物 (信頼済みプリンシパル) に許可できます。ロールは、クロスアカウントアクセスを許可する主な方法です。ただし、一部の AWS のサービスでは、 (ロールをプロキシとして使用する代わりに) リソースにポリシーを直接アタッチできます。クロスアカウントアクセスにおけるロールとリソースベースのポリシーの違いについては、「IAM ユーザーガイド」の「IAM でのクロスアカウントのリソースへのアクセス」を参照してください。
- クロスサービスアクセス – 一部の AWS のサービスは、AWS の他のサービスの機能を使用します。例えば、サービスで呼び出しを行うと、通常そのサービスによって Amazon EC2 でアプリ

ケーションが実行されたり、Amazon S3 にオブジェクトが保存されたりします。サービスは、呼び出し元のプリンシパルの許可、サービスロール、サービスリンクロールを使用してこれを行う場合があります。

- 転送アクセスセッション (FAS) – IAM ユーザーまたはロールを使用して AWS でアクションを実行するユーザーは、プリンシパルと見なされます。一部のサービスを使用する際に、アクションを実行することで、別のサービスの別のアクションがトリガーされることがあります。FAS は、AWS サービスを呼び出すプリンシパルの権限を、AWS サービスのリクエストと合わせて使用し、ダウンストリームのサービスに対してリクエストを行います。FAS リクエストは、サービスが、完了するために他の AWS サービスまたはリソースとのやりとりを必要とするリクエストを受け取ったときにのみ行われます。この場合、両方のアクションを実行するためのアクセス許可が必要です。FAS リクエストを行う際のポリシーの詳細については、「転送アクセスセッション」を参照してください。
- サービスロール - サービスがユーザーに代わってアクションを実行するために引き受ける IAM ロールです。IAM 管理者は、IAM 内からサービスロールを作成、変更、削除することができます。詳細については、『IAM ユーザーガイド』の「AWS のサービスにアクセス許可を委任するロールの作成」を参照してください。
- サービスリンクロール – サービスリンクロールは、AWS のサービスにリンクされているサービスロールの一種です。サービスがロールを引き受け、ユーザーに代わってアクションを実行できるようになります。サービスにリンクされたロールは、AWS アカウント内に表示され、サービスによって所有されます。IAM 管理者は、サービスリンクロールのアクセス許可を表示できますが、編集することはできません。

- Amazon EC2 で実行されているアプリケーション – EC2 インスタンスで実行され、AWS CLI または AWS API リクエストを作成しているアプリケーションの一時的な認証情報を管理するには、IAM ロールを使用します。これは、EC2 インスタンス内でのアクセスキーの保存に推奨されます。AWS ロールを EC2 インスタンスに割り当て、そのすべてのアプリケーションで使用できるようにするには、インスタンスに添付されたインスタンスプロファイルを作成します。インスタンスプロファイルにはロールが含まれ、EC2 インスタンスで実行されるプログラムは一時的な認証情報を取得できます。詳細については、IAM ユーザーガイドのAmazon EC2 インスタンスで実行されるアプリケーションに IAM ロールを使用して許可を付与するを参照してください。

IAM ロールと IAM ユーザーのどちらを使用するかについては、IAM ユーザーガイドの(IAM ユーザーではなく) IAM ロールをいつ作成したら良いのか?を参照してください。

ポリシーを使用したアクセス権の管理

AWS でアクセスを制御するには、ポリシーを作成して AWS ID またはリソースにアタッチします。ポリシーは AWS のオブジェクトであり、アイデンティティやリソースに関連付けて、これらのアクセス許可を定義します。AWS は、プリンシパル (ユーザー、ルートユーザー、またはロールセッション) がリクエストを行うと、これらのポリシーを評価します。ポリシーでの権限により、リクエストが許可されるか拒否されるかが決まります。大半のポリシーは JSON ドキュメントとして AWS に保存されます。JSON ポリシードキュメントの構造と内容の詳細については、「IAM ユーザーガイド」の「JSON ポリシー概要」を参照してください。

管理者は AWS JSON ポリシーを使用して、だれが何にアクセスできるかを指定できます。つまり、どのプリンシパルがどのリソースに対してどのような条件下でアクションを実行できるかということです。

デフォルトでは、ユーザーやロールに権限はありません。IAM 管理者は、リソースで必要なアクションを実行するための権限をユーザーに付与する IAM ポリシーを作成できます。その後、管理者はロールに IAM ポリシーを追加し、ユーザーはロール引き継ぐことができます。

IAM ポリシーは、オペレーションの実行方法を問わず、アクションの許可を定義します。例えば、`iam:GetRole` アクションを許可するポリシーがあるとします。このポリシーがあるユーザーは、AWS マネジメントコンソール、AWS CLI、または AWS API からロール情報を取得できます。

アイデンティティベースポリシー

アイデンティティベースのポリシーは、IAM ユーザーグループ、ユーザーのグループ、ロールなど、アイデンティティにアタッチできる JSON 許可ポリシードキュメントです。これらのポリシーは、ユーザーとロールが実行できるアクション、リソース、および条件をコントロールします。アイデンティティベースのポリシーを作成する方法については、「IAM ユーザーガイド」の「IAM ポリシーの作成」を参照してください。

アイデンティティベースのポリシーは、さらにインラインポリシーまたはマネージドポリシーに分類できます。インラインポリシーは、単一のユーザー、グループ、またはロールに直接埋め込まれています。管理ポリシーは、AWS アカウント内の複数のユーザー、グループ、およびロールにアタッチできるスタンドアロンポリシーです。管理ポリシーには、AWS 管理ポリシーとカスタマー管理ポリシーが含まれます。マネージドポリシーまたはインラインポリシーのいずれかを選択する方法については、IAM ユーザーガイドのマネージドポリシーとインラインポリシーの比較を参照してください。

リソースベースのポリシー

リソースベースのポリシーは、リソースに添付する JSON ポリシードキュメントです。リソースベースのポリシーには例として、IAM ロールの信頼ポリシーや Amazon S3 バケットポリシーがあげられます。リソースベースのポリシーをサポートするサービスでは、サービス管理者はポリシーを使用して特定のリソースへのアクセスをコントロールできます。ポリシーがアタッチされているリソースの場合、指定されたプリンシパルがそのリソースに対して実行できるアクションと条件は、ポリシーによって定義されます。リソースベースのポリシーで、[specify a principal] (プリンシパルを指定する) 必要があります。プリンシパルには、アカウント、ユーザー、ロール、フェデレーティッドユーザー、または AWS のサービスを含めることができます。

リソースベースのポリシーは、そのサービス内にあるインラインポリシーです。リソースベースのポリシーでは、IAM からの AWS マネージド型ポリシーを使用することはできません。

アクセスコントロールリスト (ACL)

アクセスコントロールリスト (ACL) は、どのプリンシパル (アカウントメンバー、ユーザー、またはロール) がリソースにアクセスするためのアクセス許可を持つかを制御します。ACL はリソースベースのポリシーに似ていますが、JSON ポリシードキュメント形式は使用しません。

Amazon S3、AWS WAF、および Amazon VPC は、ACL をサポートするサービスの例です。ACL の詳細については、Amazon Simple Storage Service デベロッパーガイド の アクセスコントロールリスト (ACL) の概要 を参照してください。

その他のポリシータイプ

AWS では、他の一般的ではないポリシータイプをサポートしています。これらのポリシータイプでは、より一般的なポリシータイプで付与された最大の権限を設定できます。

- アクセス許可の境界 - アクセス許可の境界は、アイデンティティベースポリシーによって IAM エンティティ (IAM ユーザーまたはロール) に付与できる権限の上限を設定する高度な機能です。エンティティにアクセス許可の境界を設定できます。結果として得られるアクセス許可は、エンティティのアイデンティティベースポリシーとそのアクセス許可の境界の共通部分になります。`Principal` フィールドでユーザーまたはロールを指定するリソースベースのポリシーでは、アクセス許可の境界は制限されません。これらのポリシーのいずれかを明示的に拒否した場合、権限は無効になります。アクセス許可の境界の詳細については、「IAM ユーザーガイド」の「IAM エンティティのアクセス許可の境界」を参照してください。
- サービスコントロールポリシー (SCP) – SCP とは、AWS Organizations 内の組織または組織単位 (OU) に対し、アクセス許可の上限を指定するための JSON ポリシーです。AWSOrganizations

は、ビジネスが所有する複数の AWS アカウントを、グループ化および一元管理するためのサービスです。組織内のすべての機能を有効にすると、サービスコントロールポリシー (SCP) を一部またはすべてのアカウントに適用できます。SCP はメンバーアカウントのエンティティに対するアクセス許可を制限します (各 AWS アカウントルートユーザーなど)。Organizations と SCP の詳細については、「AWS Organizations ユーザーガイド」の「サービスコントロールポリシー」を参照してください。

- セッションポリシー - セッションポリシーは、ロールまたはフェデレーションユーザーの一時的なセッションをプログラムで作成する際にパラメータとして渡す高度なポリシーです。結果として得られるセッションの許可は、ユーザーまたはロールのアイデンティティベースポリシーとセッションポリシーの共通部分です。また、リソースベースのポリシーから権限が派生する場合もあります。これらのポリシーのいずれかを明示的に拒否した場合、権限は無効になります。詳細については、「IAM ユーザーガイド」の「セッションポリシー」を参照してください。

複数のポリシータイプ

1 つのリクエストに複数のタイプのポリシーが適用されると、結果として作成されるアクセス許可を理解するのがさらに難しくなります。複数のポリシータイプが関連するとき、リクエストを許可するかどうかを AWS が決定する方法の詳細については、IAM ユーザーガイドのポリシーの評価ロジックを参照してください。

アマゾン EKS で IAM を使用する方法

IAM を使用して アマゾン EKS へのアクセスを管理する前に、アマゾン EKS で使用できる IAM 機能について理解しておく必要があります。アマゾン EKS およびその他の AWS のサービスが IAM と連携する方法の概要を把握するにはIAM ユーザーガイド の「IAM と連携する AWS のサービス」を参照してください。

トピック

- アマゾン EKS のアイデンティティベースのポリシー
- アマゾン EKS でのリソースベースのポリシー
- アマゾン EKS タグに基づいた認可
- アマゾン EKS でのIAM 役割

アマゾン EKS のアイデンティティベースのポリシー

IAM アイデンティティベースのポリシーでは許可または拒否するアクションとリソース、またアクションを許可または拒否する条件を指定できます。アマゾン EKS は特定のアクション、リソース、および条件キーをサポートしています。JSON ポリシーで使用するすべての要素については「IAM ユーザーガイド」の「IAM JSON ポリシーエレメントのリファレンス」を参照してください。

アクション

管理者は AWS JSON ポリシーを使用して、誰が何にアクセスできるかを指定できます。つまり、どのプリンシパルがどのリソースに対してどのような条件下でアクションを実行できるかということです。

JSON ポリシーの `Action` 要素にはポリシー内のアクセスを許可または拒否するために使用できるアクションが記述されます。ポリシーアクションの名前は通常、関連する AWS API オペレーションと同じです。一致する API オペレーションのないアクセス許可のみのアクションなど、いくつかの例外があります。また、ポリシーに複数のアクションが必要なオペレーションもあります。これらの追加アクションは依存アクションと呼ばれます。

このアクションは関連付けられたオペレーションを実行するためのアクセス許可を付与するポリシーで使用されます。

アマゾン EKS のポリシーアクションはアクションの前にプレフィックス `eks:` を付加します。例えば、アマゾン EKS クラスターについて説明する情報を入手するアクセス許可を他のユーザーに付与するにはポリシーに `DescribeCluster` アクションを含めます。ポリシーステートメントには`Action` または `NotAction` 要素を含める必要があります。

単一のステートメントに複数のアクションを指定するには次のようにコンマで区切ります。

```
"Action": ["eks:action1", "eks:action2"]
```

ワイルドカード (*) を使用して複数アクションを指定できます。例えば、`Describe` という単語で始まるすべてのアクションを指定するには次のアクションを含めます。

```
"Action": "eks:Describe*"
```

アマゾン EKS アクションの一覧については「サービス認可リファレンス」の「アマゾン エラスティックKubernetesサービス で定義されるアクション」を参照してください。

リソース

管理者は AWS JSON ポリシーを使用して、誰が何にアクセスできるかを指定できます。つまり、どのプリンシパルがどのリソースに対してどのような条件下でアクションを実行できるかということです。

Resource JSON ポリシー要素はアクションが適用されるオブジェクトを指定します。ステートメントにはResource または NotResource 要素を含める必要があります。ベストプラクティスとして、Amazon リソースネーム (ARN) を使用してリソースを指定します。これはリソースレベルの許可と呼ばれる特定のリソースタイプをサポートするアクションに対して実行できます。

オペレーションのリスト化など、リソースレベルの許可をサポートしないアクションの場合はステートメントがすべてのリソースに適用されることを表示するワイルドカード (*) を使用します。

```
"Resource": "*"
```

アマゾン EKS クラスターリソースの ARN は次のようになります。

```
arn:aws:eks:region-code:account-id:cluster/cluster-name
```

ARN の形式の詳細については「アマゾン リソースネーム (ARN) と AWS のサービスの名前空間」を参照してください。

例えば、ステートメントで ####### という名前のクラスターを指定するには次の ARN を使用します。

```
"Resource": "arn:aws:eks:region-code:111122223333:cluster/my-cluster"
```

特定のアカウントと AWS リージョンに属するすべてのクラスターを指定するにはワイルドカード (*) を使用します。

```
"Resource": "arn:aws:eks:region-code:111122223333:cluster/*"
```

リソースの作成を含む、一部の アマゾン EKS アクションは特定のリソースで実行できません。このような場合はワイルドカード *を使用する必要があります。

```
"Resource": "*"
```

アマゾン EKS リソースタイプとその ARN の一覧については「サービス認可リファレンス」の「アマゾン エラスティックKubernetesサービス で定義されるリソース」を参照してください。各リソースの ARN を、どのアクションで指定できるかについては「アマゾン エラスティックKubernetesサービス で定義されるアクション」を参照してください。

条件キー

アマゾン EKS では独自の条件キーが定義されており、また一部のグローバル条件キーの使用がサポートされています。すべての AWS グローバル条件キーを確認するには、IAM ユーザーガイドの「AWS グローバル条件コンテキストキー」を参照してください。

OpenID Connect プロバイダーをクラスターに関連付ける場合に条件キーを設定できます。詳細については、「the section called “IAM ポリシーの例”」を参照してください。

すべての Amazon EC2 アクションは `aws:RequestedRegion` および `ec2:Region` 条件キーをサポートします。詳細については「例: 特定の AWS リージョンへのアクセスの制限」を参照してください。

Amazon EKS 条件キーのリストについては「サービス認可リファレンス」の「Amazon エラスティックKubernetesサービス によって定義された条件」を参照してください。条件キーを使用できるアクションとリソースについては「Amazon エラスティックKubernetesサービス で定義されるアクション」を参照してください。

例

アマゾン EKS でのアイデンティティベースのポリシーの例は「the section called “アイデンティティベースのポリシー”」でご確認ください。

Amazon EKS クラスターを作成すると、このクラスターを作成する IAM プリンシパルには Amazon EKS コントロールプレーンのクラスター役割ベースアクセスコントロール (RBAC) 設定で、`system:masters` 許可が自動的に付与されます。このプリンシパルは表示可能な設定の中には表示されません。したがって、どのプリンシパルが最初にクラスターを作成したのかを追跡する必要があります。追加の IAM プリンシパルがクラスターとインタラクションできるようにするには Kubernetes 内の `aws-auth ConfigMap` を編集し、`aws-auth ConfigMap` で指定した `group` の名前を使用して、Kubernetes `rolebinding` または `clusterrolebinding` を作成します。

ConfigMap の使用に関する詳細については「the section called “Kubernetes API アクセス”」を参照してください。

アマゾン EKS でのリソースベースのポリシー

アマゾン EKS では リソースベースのポリシーはサポートされていません。

アマゾン EKS タグに基づいた認可

タグはアマゾン EKS リソースにアタッチすることも、アマゾン EKS へのリクエストの際に渡すこともできます。タグに基づいてアクセスを制御するには`aws:ResourceTag/key-name`、`aws:RequestTag/key-name` 、または `aws:TagKeys` の条件キーを使用して、ポリシーの条件要素でタグ情報を提供します。アマゾン EKS リソースのタグ付けの詳細については「the section called “リソースのタグ付け”」を参照してください。条件キーでタグを使用できるアクションの詳細については「Service Authorization Reference」(サービス認可のリファレンス) の「アマゾン EKS で定義されたアクション」を参照してください。

アマゾン EKS でのIAM 役割

IAM 役割 は特定のアクセス許可を持つ、AWS アカウント内のエンティティです。

アマゾン EKS での一時的な認証情報の使用

一時的な認証情報を使用して、フェデレーションでサインインする、IAM 役割を引き受ける、またはクロスアカウント役割を引き受けることができます。一時的なセキュリティ認証情報を取得するには AssumeRole または GetFederationToken などの AWS STS API オペレーションを呼び出します。

アマゾン EKS では一時認証情報が使用できます。

サービスにリンクされたロール

サービスリンクロールは、AWS サービスが他のサービスのリソースにアクセスしてお客様の代わりにアクションを完了することを許可します。サービスリンクロールは IAM アカウント内に表示され、サービスによって所有されます。管理者はサービスにリンクされたロールのアクセス許可を確認できますが、編集することはできません。

アマゾン EKS はサービスにリンクされた役割をサポートしています。アマゾン EKS でのサービスにリンクされた役割の作成または管理の詳細については「the section called “サービスにリンクされた役割”」を参照してください。

サービス役割

この機能により、ユーザーに代わってサービスがサービスロールを引き受けることが許可されます。この役割により、サービスがお客様に代わって他のサービスのリソースにアクセスし、アクションを

完了することが許可されます。サービスロールはIAM アカウントに表示され、アカウントによって所有されます。つまり、IAM 管理者はこの役割の権限を変更できます。ただし、これを行うことにより、サービスの機能が損なわれる場合があります。

アマゾン EKS ではサービス役割がサポートされています。詳細についてはthe section called “クラスター IAM ロール”およびthe section called “ノード の IAM ロール”を参照してください。

アマゾン EKS での IAM 役割の選択

アマゾン EKS でクラスターリソースを作成する場合は他のいくつかの AWS リソースに アマゾン EKS が自動的にアクセスすることを許可するための役割を、ユーザーが選択する必要があります。以前に作成したサービス役割がある場合、アマゾン EKS により、選択できる役割のリストが提示されます。アマゾン EKS 管理のポリシーがアタッチされた役割を選択することが重要です。詳細についてはthe section called “既存のクラスターロールの確認”およびthe section called “既存のノードロールの確認”を参照してください。

Amazon EKS でのアイデンティティベースのポリシーの例

デフォルトでは、IAM ユーザーおよびロールには Amazon EKS リソースを作成または変更するアクセス許可はありません。また、AWS マネジメントコンソール、AWS CLI、または AWS API を使用してタスクを実行することもできません。IAM 管理者は、ユーザーとロールに必要な、指定されたリソースで特定の API オペレーションを実行する権限をユーザーとロールに付与する IAM ポリシーを作成する必要があります。続いて、管理者はそれらの権限が必要な IAM ユーザーまたはグループにそのポリシーをアタッチする必要があります。

これらの JSON ポリシードキュメント例を使用して IAM のアイデンティティベースのポリシーを作成する方法については、IAM ユーザーガイドの「JSON タブでのポリシーの作成」を参照してください。

Amazon EKS クラスターを作成すると、このクラスターを作成する IAM プリンシパルには、Amazon EKS コントロールプレーンのクラスターロールベースアクセスコントロール (RBAC) 設定で、`system:masters` 許可が自動的に付与されます。このプリンシパルは表示可能な設定の中には表示されません。したがって、どのプリンシパルが最初にクラスターを作成したのかを追跡する必要があります。追加の IAM プリンシパルがクラスターとインタラクションできるようにするには Kubernetes 内の `aws-auth ConfigMap` を編集し、`aws-auth ConfigMap` で指定した `group` の名前を使用して、Kubernetes `rolebinding` または `clusterrolebinding` を作成します。

ConfigMap の使用に関する詳細については、「the section called “Kubernetes API アクセス”」を参照してください。

トピック

- ポリシーに関するベストプラクティス
- Amazon EKS コンソールの使用
- 自分のアクセス許可の表示を IAM ユーザーに許可する
- AWS Cloud に Kubernetes クラスターを作成する
- Outpost にローカル Kubernetes クラスターを作成する
- Kubernetes クラスターの更新
- すべてのクラスターの一覧表示または説明
- クラスターの更新をキャンセルする (ロールバック)

ポリシーに関するベストプラクティス

ID ベースのポリシーは、ユーザーのアカウント内で誰かが Amazon EKS リソースを作成、アクセス、または削除できるどうかを決定します。これらのアクションを実行すると、AWS アカウントに追加料金が発生する可能性があります。アイデンティティベースポリシーを作成したり編集したりする際には、以下のガイドラインと推奨事項に従ってください:

- AWS マネージドポリシーの使用を開始し、最小特権のアクセス許可に移行する – ユーザーとワークロードへのアクセス許可の付与を開始するには、多くの一般的なユースケースのためにアクセス許可を付与する AWS マネージドポリシーを使用します。これらは AWS アカウントで使用できます。ユースケースに固有の AWS カスタマー管理ポリシーを定義して、アクセス許可を絞り込むことをお勧めします。詳細については、IAM ユーザーガイド の AWS マネージドポリシー または ジョブ機能の AWS マネージドポリシー を参照してください。
- 最小特権を適用する – IAM ポリシーでアクセス許可を設定する場合は、タスクの実行に必要な許可のみを付与します。これを行うには、特定の条件下で特定のリソースに対して実行できるアクションを定義します。これは、最小特権アクセス許可とも呼ばれています。IAM を使用して許可を適用する方法の詳細については、IAM ユーザーガイド の IAM でのポリシーとアクセス許可 を参照してください。
- IAM ポリシーで条件を使用してアクセスをさらに制限する – ポリシーに条件を追加して、アクションやリソースへのアクセスを制限できます。例えば、ポリシー条件を記述して、すべてのリクエストを SSL を使用して送信するように指定できます。また、AWS CloudFormation などの特定の AWS サービスを介して使用する場合、条件を使用してサービスアクションへのアクセスを許可することもできます。詳細については、IAM ユーザーガイド」の「IAM JSON ポリシー要素: 条件」を参照してください。

- IAM アクセスアナライザー を使用して IAM ポリシーを検証し、安全で機能的な権限を確保する – IAM アクセスアナライザー は、新規および既存のポリシーを検証して、ポリシーが IAM ポリシー言語 (JSON) および IAM のベストプラクティスに準拠するようにします。IAM アクセスアナライザーは 100 を超えるポリシーチェックと実用的な推奨事項を提供し、安全で機能的なポリシーの作成をサポートします。詳細については、IAM ユーザーガイド の IAM Access Analyzer ポリシーの検証 を参照してください。
- 多要素認証 (MFA) を要求する – AWS アカウントで IAM ユーザーまたはルートユーザーを要求するシナリオがある場合は、セキュリティを強化するために MFA をオンにします。API オペレーションが呼び出されるときに MFA を必須にするには、ポリシーに MFA 条件を追加します。詳細については、IAM ユーザーガイド の MFA 保護 API アクセスの設定 を参照してください。

IAM でのベストプラクティスの詳細については、IAM ユーザーガイド の IAM でのセキュリティのベストプラクティス を参照してください。

Amazon EKS コンソールの使用

Amazon EKS コンソールにアクセスするには、IAM プリンシパルに最小限の許可のセットが必要です。これらの許可により、プリンシパルは AWS アカウントの Amazon EKS リソースの詳細をリストおよび表示できます。最小限必要な許可よりも制限されたアイデンティティベースのポリシーを作成すると、そのポリシーをアタッチしたプリンシパルに対してはコンソールが意図したとおりに機能しません。

これらの IAM プリンシパルがまだ Amazon EKS コンソールを使用できるようにするには、`AmazonEKSAdminPolicy` など、独自の名前を付けてポリシーを作成します。ポリシーをプリンシパルにアタッチします。詳細については、「IAM User Guide」(IAM ユーザーガイド) の「Adding and removing IAM identity permissions」(IAM ID アクセス許可の追加と削除) を参照してください。

> **Important**
>
> 次のポリシー例では、プリンシパルはコンソールの [設定] タブで情報を表示できます。AWS マネジメントコンソール で [概要] タブと [リソース] タブの情報を表示するには、プリンシパルに Kubernetes の許可も必要です。詳細については、「the section called “必要なアクセス許可”」を参照してください。

```
{
```

```
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "eks:*"
            ],
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": "iam:PassRole",
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "iam:PassedToService": "eks.amazonaws.com"
                }
            }
        }
    ]
}
```

AWS CLI または AWS API のみを呼び出すプリンシパルには、最小限のコンソール許可を付与する必要はありません。代わりに、実行する API オペレーションと一致するアクションのみへのアクセスを許可します。

自分のアクセス許可の表示を IAM ユーザーに許可する

この例では、ユーザーアイデンティティにアタッチされたインラインおよびマネージドポリシーの表示を IAM ユーザーに許可するポリシーの作成方法を示します。このポリシーには、コンソールで、または AWS CLI もしくは AWS API を使用してプログラム的に、このアクションを完了するための許可が含まれています。

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "ViewOwnUserInfo",
            "Effect": "Allow",
            "Action": [
                "iam:GetUserPolicy",
                "iam:ListGroupsForUser",
```

```
                "iam:ListAttachedUserPolicies",
                "iam:ListUserPolicies",
                "iam:GetUser"
            ],
            "Resource": ["arn:aws:iam::*:user/${aws:username}"]
        },
        {
            "Sid": "NavigateInConsole",
            "Effect": "Allow",
            "Action": [
                "iam:GetGroupPolicy",
                "iam:GetPolicyVersion",
                "iam:GetPolicy",
                "iam:ListAttachedGroupPolicies",
                "iam:ListGroupPolicies",
                "iam:ListPolicyVersions",
                "iam:ListPolicies",
                "iam:ListUsers"
            ],
            "Resource": "*"
        }
    ]
}
```

AWS Cloud に Kubernetes クラスターを作成する

このポリシー例には、*us-west-2* AWS リージョンに *my-cluster* という名前の Amazon EKS クラスターを作成するために必要な最小限の許可が含まれています。AWS リージョンを、クラスターを作成する AWS リージョンに置き換えることができます。AWS マネジメントコンソール で [ポリシーのアクションはリソースレベルの許可をサポートしていないため、**All resources** を選択する必要があります] という警告が表示された場合、無視しても問題ありません。アカウントに *AWSServiceRoleForAmazonEKS* ロールが既にある場合は、ポリシーから `iam:CreateServiceLinkedRole` アクションを削除できます。アカウントで Amazon EKS クラスターを作成したことがある場合、削除していない限り、このロールは既に存在します。

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "eks:CreateCluster",
            "Resource": "arn:aws:eks:us-west-2:111122223333:cluster/my-cluster"
```

```
        },
        {
            "Effect": "Allow",
            "Action": "iam:CreateServiceLinkedRole",
            "Resource": "arn:aws:iam::111122223333:role/aws-service-role/
eks.amazonaws.com/AWSServiceRoleForAmazonEKS",
            "Condition": {
                "ForAnyValue:StringEquals": {
                    "iam:AWSServiceName": "eks"
                }
            }
        },
        {
            "Effect": "Allow",
            "Action": "iam:PassRole",
            "Resource": "arn:aws:iam::111122223333:role/cluster-role-name"
        }
    ]
}
```

Outpost にローカル Kubernetes クラスターを作成する

このポリシー例には、*us-west-2* AWS リージョンの Outpost 上に *my-cluster* という名前の Amazon EKS ローカルクラスターを作成するために必要な最小限の許可が含まれています。AWS リージョンを、クラスターを作成する AWS リージョンに置き換えることができます。AWS マネジメントコンソール で [ポリシーのアクションはリソースレベルの許可をサポートしていないため、**All resources** を選択する必要があります] という警告が表示された場合、無視しても問題ありません。アカウントに AWSServiceRoleForAmazonEKSLocalOutpost ロールがすでにある場合、ポリシーから iam:CreateServiceLinkedRole アクションを削除できます。アカウントで Outpost 上に Amazon EKS ローカルクラスターを作成したことがある場合、削除していない限り、このロールはすでに存在します。

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "eks:CreateCluster",
            "Resource": "arn:aws:eks:us-west-2:111122223333:cluster/my-cluster"
        },
        {
            "Action": [
```

```
                "ec2:DescribeSubnets",
                "ec2:DescribeVpcs",
                "iam:GetRole"
            ],
            "Resource": "*",
            "Effect": "Allow"
        },
        {
            "Effect": "Allow",
            "Action": "iam:CreateServiceLinkedRole",
            "Resource": "arn:aws:iam::111122223333:role/aws-service-role/outposts.eks-
local.amazonaws.com/AWSServiceRoleForAmazonEKSLocalOutpost"
        },
        {
            "Effect": "Allow",
            "Action": [
                "iam:PassRole",
                "iam:ListAttachedRolePolicies"
            ],
            "Resource": "arn:aws:iam::111122223333:role/cluster-role-name"
        },
        {
            "Action": [
                "iam:CreateInstanceProfile",
                "iam:TagInstanceProfile",
                "iam:AddRoleToInstanceProfile",
                "iam:GetInstanceProfile",
                "iam:DeleteInstanceProfile",
                "iam:RemoveRoleFromInstanceProfile"
            ],
            "Resource": "arn:aws:iam::*:instance-profile/eks-local-*",
            "Effect": "Allow"
        }
    ]
}
```

Kubernetes クラスターの更新

このポリシー例には、us-west-2 AWS リージョンに *my-cluster* という名前のクラスターを更新するために必要な最小限の許可が含まれています。

```
{
    "Version":"2012-10-17",
```

```
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "eks:UpdateClusterVersion",
            "Resource": "arn:aws:eks:us-west-2:111122223333:cluster/my-cluster"
        }
    ]
}
```

すべてのクラスターの一覧表示または説明

このポリシーの例には、アカウント内のすべてのクラスターを一覧表示して記述するために必要な最小限の許可が含まれています。update-kubeconfig AWS CLI コマンドを使用するには、IAM プリンシパルがクラスターを一覧表示および記述できる必要があります。

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "eks:DescribeCluster",
                "eks:ListClusters"
            ],
            "Resource": "*"
        }
    ]
}
```

クラスターの更新をキャンセルする (ロールバック)

ユーザーが進行中のロールバックオペレーションをキャンセルできるようにするには、eks:CancelUpdate アクションを付与します。これは、リソース ARN またはタグベースの条件を使用してスコープできます。

リソース ARN ベース:

```
{
  "Version": "2012-10-17",
  "Statement": [{
    "Effect": "Allow",
```

```
    "Action": "eks:CancelUpdate",
    "Resource": "arn:aws:eks:us-west-2:111122223333:cluster/my-cluster"
  }]
}
```

タグベース:

```
{
  "Version": "2012-10-17",
  "Statement": [{
    "Effect": "Allow",
    "Action": "eks:CancelUpdate",
    "Resource": "*",
    "Condition": {
      "StringEquals": {
        "aws:ResourceTag/environment": "development"
      }
    }
  }]
}
```

クラスターの完全なロールバックプロセスの詳細については、the section called “Kubernetes バージョンのロールバック” を参照してください。

Amazon EKS でのサービスにリンクされたロールの使用

Amazon エラスティック Kus サービス はAWS アイデンティティとアクセス管理 (IAM) のサービberneteスにリンクされた役割を使用します。サービスにリンクされた役割はAmazon EKS に直接リンクされた一意のタイプの IAM 役割です。サービスにリンクされた役割はAmazon EKS で事前定義されています。この役割にはサービスがユーザーに代わって他の AWS のサービスを呼び出すために必要な、すべてのアクセス許可が付与されています。

トピック

- Amazon EKS クラスターのロールの使用
- Amazon EKS ノードグループでのロールの使用
- Amazon EKS Fargate プロファイルのロールの使用
- ロールを使用して Kubernetes クラスターを Amazon EKS に接続する
- アウトポスト で Amazon EKS ローカルクラスターの役割の使用
- Amazon EKS ダッシュボードのロールの使用

Amazon EKS クラスターのロールの使用

Amazon エラスティック Kus サービス はAWS アイデンティティとアクセス管理 (IAM) のサービberneteスにリンクされた役割を使用します。サービスにリンクされた役割はAmazon EKS に直接リンクされた一意のタイプの IAM 役割です。サービスにリンクされた役割はAmazon EKS で事前定義されています。この役割にはサービスがユーザーに代わって他の AWS のサービスを呼び出すために必要な、すべてのアクセス許可が付与されています。

サービスにリンクされた役割を使用することで、必要なアクセス許可を手動で追加する必要がなくなるため、Amazon EKS の設定が簡単になります。サービスにリンクされた役割のアクセス許可は Amazon EKS により定義されます。特に指定されている場合を除き、Amazon EKS のみがその役割を引き受けることができます。定義される許可は信頼ポリシーと許可ポリシーに含まれており、その許可ポリシーを他の IAM エンティティにアタッチすることはできません。

サービスリンクロールを削除するには、まずその関連リソースを削除します。これにより、リソースへのアクセス許可を不用意に削除することが防止され、Amazon EKS リソースが保護されます。

サービスにリンクされた役割をサポートする他のサービスについては「IAM と連動する AWS サービス」を参照し、[サービス-linked roles] (サービスにリンクされた役割) の列内で [Yes] (はい) と表記されたサービスを確認してください。サービスにリンクされた役割に関するドキュメントをサービスで表示するには[はい] リンクを選択してください。

Amazon EKS でのサービスにリンクされた役割のアクセス許可

Amazon EKS は、AWSServiceRoleForAmazonEKS という名前のサービスにリンクされたロールを使用します。このロールにより、Amazon EKS はアカウント内のクラスターを管理できます。アタッチされたポリシーにより、ロールはネットワークインターフェイス、セキュリティグループ、ログ、VPC のリソースを管理できるようになります。

> **Note**
>
> AWSServiceRoleForAmazonEKS サービスにリンクされたロールは、クラスターの作成に必要なロールとは異なります。詳細については「the section called “クラスター IAM ロール”」を参照してください。

AWSServiceRoleForAmazonEKS サービスリンク役割は役割の引き受けについて以下のサービスを信頼します。

- eks.amazonaws.com

役割のアクセス許可ポリシーは指定したリソースに対して以下のアクションを実行することを Amazon EKS に許可します。

- AmazonEKSServiceRolePolicy

サービスにリンクされた役割の作成、編集、削除を IAM エンティティ (ユーザー、グループ、役割など) に許可するにはアクセス許可を設定する必要があります。詳細についてはIAM ユーザーガイド の「サービスにリンクされた役割のアクセス許可」を参照してください。

Amazon EKS でのサービスにリンクされた役割の作成

サービスリンク役割を手動で作成する必要はありません。AWS マネジメントコンソール、AWS CLI、または AWS API でクラスターを作成すると、Amazon EKS によってサービスリンク役割が作成されます。

このサービスリンク役割を削除した後で再度作成する必要が生じた場合は同じ方法でアカウントに役割を再作成できます。サービスにリンクされた役割はクラスターの作成時に Amazon EKS で自動的に再作成されます。

Amazon EKS でのサービスにリンクされた役割の編集

Amazon EKS では`AWSServiceRoleForAmazonEKS` のサービスにリンクされた役割を編集することはできません。サービスにリンクされた役割を作成すると、多くのエンティティによって役割が参照される可能性があるため、役割名を変更することはできません。ただし、IAM を使用してロールの説明を編集することはできます。詳細については『IAM ユーザーガイド』の「サービスにリンクされた役割の編集」を参照してください。

Amazon EKS でのサービスにリンクされた役割の削除

サービスリンク役割が必要な機能またはサービスが不要になった場合にはその役割を削除することをお勧めします。そうすることで、モニタリングや保守が積極的に行われていない未使用のエンティティを排除できます。ただし、手動で削除する前に、サービスリンク役割をクリーンアップする必要があります。

サービスリンク役割のクリーンアップ

IAM を使用してサービスにリンクされた役割を削除するには最初に、その役割で使用されているリソースをすべて削除する必要があります。

> **Note**
>
> リソースの削除を試みた際に、対応する役割が Amazon EKS サービスで使用されている場合、削除が失敗することがあります。失敗した場合は数分待ってから操作を再試行してください。

1. Amazon EKS コンソールを開きます。
2. 左のナビゲーションペインで、 [クラスター] を選択してください。
3. クラスターにノードグループまたは Fargate プロファイルがある場合は、クラスターを削除する前にそれらを削除する必要があります。詳細については「the section called “削除”」および「the section called “プロファイルを削除する”」を参照してください。
4. [Clusters (クラスター)] ページで、削除するクラスターを選択し、[Delete (削除)] を選択してください。
5. 削除確認ウィンドウにクラスター名を入力し、[Delete (削除)] を選択してください。
6. この手順をアカウント内の他のすべてのクラスターに対して繰り返します。すべての削除操作が完了するまで待ちます。

サービスリンク役割の手動による削除

サービスにリンクされた役割 `AWSServiceRoleForAmazonEKS` を削除するにはIAM コンソール、AWS CLI、または AWS API を使用します。詳細についてはIAM ユーザーガイド の「サービスにリンクされた役割の削除」を参照してください。

Amazon EKS のサービスにリンクされた役割がサポートされるリージョン

Amazon EKS ではこのサービスを利用できるすべてのリージョンで、サービスにリンクされた役割の使用がサポートされます。詳細については「Amazon EKS endpoints and quotas」(Amazon EKS エンドポイントとクォータ) を参照してください。

Amazon EKS ノードグループでのロールの使用

Amazon EKS はAWS アイデンティティとアクセス管理 (IAM) のサービスにリンクされた役割を使用します。サービスにリンクされた役割はAmazon EKS に直接リンクされた一意のタイプの IAM 役割です。サービスにリンクされた役割はAmazon EKS で事前定義されています。この役割にはサービスがユーザーに代わって他の AWS のサービスを呼び出すために必要な、すべてのアクセス許可が付与されています。

サービスにリンクされた役割を使用することで、必要なアクセス許可を手動で追加する必要がなくなるため、Amazon EKS の設定が簡単になります。サービスにリンクされた役割のアクセス許可は Amazon EKS により定義されます。特に指定されている場合を除き、Amazon EKS のみがその役割を引き受けることができます。定義される許可は信頼ポリシーと許可ポリシーに含まれており、その許可ポリシーを他の IAM エンティティにアタッチすることはできません。

サービスリンクロールを削除するには、まずその関連リソースを削除します。これにより、リソースへのアクセス許可を不用意に削除することが防止され、Amazon EKS リソースが保護されます。

サービスにリンクされたロールをサポートする他のサービスについては、「IAM と連動する AWS サービス」を参照し、[Service-linked roles] (サービスにリンクされたロール) の列内で [Yes] (はい) と表記されたサービスを確認してください。サービスにリンクされた役割に関するドキュメントをサービスで表示するには[はい] リンクを選択してください。

Amazon EKS でのサービスにリンクされた役割のアクセス許可

Amazon EKS はAWSServiceRoleForAmazonEKSNodegroup という名前のサービスにリンクされたロールを使用します。このロールにより、Amazon EKS はアカウント内のノードグループを管理できます。アタッチされた AWSServiceRoleForAmazonEKSNodegroup ポリシーにより、ロールは 自動スケーリング グループ、セキュリティグループ、起動テンプレート、および IAM インスタンスプロファイルのリソースを管理できるようになります。詳細については「the section called “AWS マネージドポリシー: AWSServiceRoleForAmazonEKSNodegroup”」を参照してください。

AWSServiceRoleForAmazonEKSNodegroup サービスリンクロールはロールの引き受けについて以下のサービスを信頼します。

- eks-nodegroup.amazonaws.com

ロールのアクセス許可ポリシーは指定したリソースに対して以下のアクションを実行することを Amazon EKS に許可します。

- AWSServiceRoleForAmazonEKSNodegroup

サービスにリンクされたロールの作成、編集、削除を IAM エンティティ (ユーザー、グループ、ロールなど) に許可するにはアクセス許可を設定する必要があります。詳細についてはIAM ユーザーガイド の「サービスにリンクされた役割のアクセス許可」を参照してください。

Amazon EKS でのサービスにリンクされた役割の作成

サービスリンクロールを手動で作成する必要はありません。AWS マネジメントコンソール、AWS CLI、または AWS API でマネージド型ノードグループを作成すると、Amazon EKS によってサービスリンクロールが作成されます。

> **Important**
>
> このサービスリンクロールはこのロールでサポートされている機能を使用する別のサービスでアクションが完了した場合にアカウントに表示されます。2017 年 1 月 1 日より前に Amazon EKS サービスを使用している場合、サービスにリンクされたロールのサポートが開始された時点で、AWSServiceRoleForAmazonEKSNodegroup ロールは Amazon EKS によりアカウントに作成されています。詳細については[IAM アカウントに新しいロールが表示される]を参照してください。

Amazon EKS でのサービスにリンクされた役割の作成 (AWS API)

サービスリンクロールを手動で作成する必要はありません。AWS マネジメントコンソール、AWS CLI、または AWS API でマネージド型ノードグループを作成すると、Amazon EKS によってサービスリンクロールが作成されます。

このサービスリンクロールを削除した後で再度作成する必要が生じた場合は同じ方法でアカウントにロールを再作成できます。別のマネージド型ノードグループを作成すると、Amazon EKS によってサービスリンク役割が再度作成されます。

Amazon EKS でのサービスにリンクされた役割の編集

Amazon EKS では`AWSServiceRoleForAmazonEKSNodegroup` のサービスにリンクされた役割を編集することはできません。サービスにリンクされた役割を作成すると、多くのエンティティによって役割が参照される可能性があるため、役割名を変更することはできません。ただし、IAM を使用してロールの説明を編集することはできます。詳細については『IAM ユーザーガイド』の「[サービスにリンクされた役割の編集]」を参照してください。

Amazon EKS でのサービスにリンクされた役割の削除

サービスリンク役割が必要な機能またはサービスが不要になった場合にはその役割を削除することをお勧めします。そうすることで、モニタリングや保守が積極的に行われていない未使用のエンティティを排除できます。ただし、手動で削除する前に、サービスリンク役割をクリーンアップする必要があります。

サービスリンク役割のクリーンアップ

IAM を使用してサービスにリンクされた役割を削除するには最初に、その役割で使用されているリソースをすべて削除する必要があります。

> **Note**
>
> リソースの削除を試みた際に、対応する役割が Amazon EKS サービスで使用されている場合、削除が失敗することがあります。失敗した場合は数分待ってから操作を再試行してください。

1. Amazon EKS コンソールを開きます。
2. 左のナビゲーションペインで [クラスター] を選択してください。
3. [Compute] (コンピューティング) タブを選択してください。
4. [Node groups] (ノードグループ) セクションで、削除するノードグループを選択してください。
5. 削除確認ウィンドウにノードグループの名前を入力し、[Delete (削除)] を選択してください。
6. この手順をクラスター内の他のすべてのノードグループに対して繰り返します。すべての削除操作が完了するまで待ちます。

サービスリンク役割の手動による削除

サービスにリンクされた役割 `AWSServiceRoleForAmazonEKSNodegroup` を削除するにはIAM コンソール、AWS CLI、または AWS API を使用します。詳細についてはIAM ユーザーガイド の「サービスにリンクされた役割の削除」を参照してください。

Amazon EKS のサービスにリンクされた役割がサポートされるリージョン

Amazon EKS ではこのサービスを利用できるすべてのリージョンで、サービスにリンクされた役割の使用がサポートされます。詳細については、「Amazon EKS endpoints and quotas」(Amazon EKS エンドポイントとクォータ) を参照してください。

Amazon EKS Fargate プロファイルのロールの使用

Amazon EKS はAWS アイデンティティとアクセス管理 (IAM) のサービスにリンクされたロールを使用します。サービスにリンクされたロールは Amazon EKS に直接リンクされた一意のタイプの IAM ロールです。サービスにリンクされたロールはAmazon EKS で事前定義されています。このロール

にはサービスがユーザーに代わって他の AWS のサービスを呼び出すために必要な、すべてのアクセス許可が付与されています。

サービスにリンクされたロールを使用することで、必要なアクセス許可を手動で追加する必要がなくなるため、Amazon EKS の設定が簡単になります。サービスにリンクされたロールのアクセス許可はAmazon EKS により定義されます。特に指定されている場合を除き、Amazon EKS のみがそのロールを引き受けることができます。定義される許可は信頼ポリシーと許可ポリシーに含まれており、その許可ポリシーを他の IAM エンティティにアタッチすることはできません。

サービスリンクロールを削除するには、まずその関連リソースを削除します。これにより、リソースへのアクセス許可を不用意に削除することが防止され、Amazon EKS リソースが保護されます。

サービスにリンクされたロールをサポートする他のサービスについては、「IAM と連動する AWS サービス」を参照し、[Service-linked roles] (サービスにリンクされたロール) の列内で [Yes] (はい) と表記されたサービスを確認してください。サービスにリンクされたロールに関するドキュメントをサービスで表示するには[はい] リンクを選択してください。

Amazon EKS でのサービスにリンクされたロールのアクセス許可

Amazon EKS は`AWSServiceRoleForAmazonEKSForFargate` という名前のサービスにリンクされたロールを使用します。このロールにより、Amazon EKS Fargate は Fargate Pod に必要な VPC ネットワーキングを設定できます。アタッチされたポリシーにより、ロールは Elastic Network Interface の作成と削除、および Elastic Network Interface とリソースの記述が可能になります。

`AWSServiceRoleForAmazonEKSForFargate` サービスリンクロールはロールの引き受けについて以下のサービスを信頼します。

- `eks-fargate.amazonaws.com`

ロールのアクセス許可ポリシーは指定したリソースに対して以下のアクションを実行することを Amazon EKS に許可します。

- アAmazonEKSForFargateServiceRolePolicy

サービスリンクロールの作成、編集、削除を IAM エンティティ (ユーザー、グループ、ロールなど) に許可するにはアクセス許可を設定する必要があります。詳細についてはIAM ユーザーガイド の「サービスにリンクされたロールのアクセス許可」を参照してください。

Amazon EKS でのサービスにリンクされたロールの作成

サービスリンクロールを手動で作成する必要はありません。AWS マネジメントコンソール、AWS CLI、または AWS API で Fargate プロファイルを作成すると、Amazon EKS がサービスリンクロールを作成します。

> **Important**
>
> このサービスリンクロールはこのロールでサポートされている機能を使用する別のサービスでアクションが完了した場合にアカウントに表示されます。2019 年 12 月 13 日より前に Amazon EKS サービスを使用している場合、サービスにリンクされたロールのサポートが開始された時点で、AWSServiceRoleForAmazonEKSForFargate は Amazon EKS によりアカウントに作成されています。詳細については IAM アカウントに新しいロールが表示される を参照してください。

Amazon EKS でのサービスにリンクされたロールの作成 (AWS API)

サービスリンクロールを手動で作成する必要はありません。AWS マネジメントコンソール、AWS CLI、または AWS API で Fargate プロファイルを作成すると、Amazon EKS がサービスリンクロールを作成します。

このサービスリンクロールを削除した後で再度作成する必要が生じた場合は同じ方法でアカウントにロールを再作成できます。別の Fargate プロファイルを作成すると、サービスリンクロールが Amazon EKS で自動的に再作成されます。

Amazon EKS でのサービスにリンクされたロールの編集

Amazon EKS では`AWSServiceRoleForAmazonEKSForFargate` のサービスにリンクされたロールを編集することはできません。サービスにリンクされたロールを作成すると、多くのエンティティによってロールが参照される可能性があるため、ロール名を変更することはできません。ただし、IAM を使用してロールの説明を編集することはできます。詳細については、IAM ユーザーガイドの「サービスにリンクされたロールの編集」を参照してください。

Amazon EKS でのサービスにリンクされたロールの削除

サービスリンクロールが必要な機能またはサービスが不要になった場合にはそのロールを削除することをお勧めします。そうすることで、モニタリングや保守が積極的に行われていない未使用のエンティティを排除できます。ただし、手動で削除する前に、サービスリンクロールをクリーンアップする必要があります。

サービスリンクロールのクリーンアップ

IAM を使用してサービスにリンクされたロールを削除するには最初に、そのロールで使用されているリソースをすべて削除する必要があります。

> **Note**
>
> リソースの削除を試みた際に、対応するロールが Amazon EKS サービスで使用されている場合、削除が失敗することがあります。失敗した場合は数分待ってから操作を再試行してください。

1. Amazon EKS コンソールを開きます。
2. 左のナビゲーションペインで [クラスター] を選択します。
3. [Clusters] (クラスター) ページで、クラスターを選択してください。
4. [Compute] (コンピューティング) タブを選択してください。
5. [Fargate profiles] (Fargateプロファイル) セクションに Fargate プロファイルがある場合、それぞれを個別に選択し、[Delete] (削除)を選択してください。
6. 削除確認ウィンドウにプロファイルの名前を入力し、[Delete (削除)] を選択してください。
7. クラスター内のその他の Fargate プロファイルと、アカウント内のその他のクラスターについて、この手順を繰り返します。

サービスリンクロールの手動による削除

IAM コンソール、AWS CLI、または AWS API を使用して、AWSServiceRoleForAmazonEKSForFargate のサービスにリンクされたロールを削除します。詳細についてはIAM ユーザーガイド の「サービスにリンクされたロールの削除」を参照してください。

Amazon EKS のサービスにリンクされたロールがサポートされるリージョン

Amazon EKS ではこのサービスを利用できるすべてのリージョンで、サービスにリンクされたロールの使用がサポートされます。詳細については、「Amazon EKS エンドポイントとクォータ」を参照してください。

ロールを使用して Kubernetes クラスターを Amazon EKS に接続する

Amazon EKS はAWS アイデンティティとアクセス管理 (IAM) のサービスにリンクされた役割を使用します。サービスにリンクされた役割はAmazon EKS に直接リンクされた一意のタイプの IAM 役割です。サービスにリンクされた役割はAmazon EKS で事前定義されています。この役割にはサービスがユーザーに代わって他の AWS のサービスを呼び出すために必要な、すべてのアクセス許可が付与されています。

サービスにリンクされた役割を使用することで、必要なアクセス許可を手動で追加する必要がなくなるため、Amazon EKS の設定が簡単になります。サービスにリンクされた役割のアクセス許可は Amazon EKS により定義されます。特に指定されている場合を除き、Amazon EKS のみがその役割を引き受けることができます。定義される許可は信頼ポリシーと許可ポリシーに含まれており、その許可ポリシーを他の IAM エンティティにアタッチすることはできません。

サービスリンクロールを削除するには、まずその関連リソースを削除します。これにより、リソースへのアクセス許可を不用意に削除することが防止され、Amazon EKS リソースが保護されます。

サービスにリンクされた役割をサポートする他のサービスについては「IAM と連動する AWS サービス」を参照し、[サービス-linked roles] (サービスにリンクされた役割) の列内で [Yes] (はい) と表記されたサービスを確認してください。サービスにリンクされた役割に関するドキュメントをサービスで表示するには[はい] リンクを選択してください。

Amazon EKS でのサービスにリンクされた役割のアクセス許可

Amazon EKS はAWSServiceRoleForAmazonEKSConnector という名前のサービスにリンクされた役割を使用します。このロールにより、Amazon EKS は Kubernetes クラスターに接続できます。アタッチされたポリシーにより、ロールは、登録された Kubernetes クラスターに接続するために必要なリソースを管理できます。

AWSServiceRoleForAmazonEKSConnector サービスリンク役割は役割の引き受けについて以下のサービスを信頼します。

- eks-connector.amazonaws.com

役割のアクセス許可ポリシーは指定したリソースに対して以下のアクションを実行することを Amazon EKS に許可します。

- AmazonEKSConnectorServiceRolePolicy

サービスリンク役割の作成、編集、削除を IAM エンティティ (ユーザー、グループ、役割など) に許可するにはアクセス許可を設定する必要があります。詳細についてはIAM ユーザーガイド の「サービスにリンクされた役割のアクセス許可」を参照してください。

このロールは、SSM (Systems Manager) アクセス許可を使用して安全な接続を確立し、接続された Kubernetes クラスターを管理します。

Amazon EKS でのサービスにリンクされた役割の作成

サービスにリンクされた役割を手動で作成してクラスターを接続する必要はありません。AWS マネジメントコンソール、AWS CLI、`eksctl` または AWS API でクラスターを接続すると、Amazon EKS によってサービスにリンクされた役割が作成されます。

このサービスリンク役割を削除した後で再度作成する必要が生じた場合は同じ方法でアカウントに役割を再作成できます。サービスにリンクされた役割はクラスターの接続時に Amazon EKS で自動的に再作成されます。

Amazon EKS でのサービスにリンクされた役割の編集

Amazon EKS では`AWSServiceRoleForAmazonEKSConnector` のサービスにリンクされた役割を編集することはできません。サービスにリンクされた役割を作成すると、多くのエンティティによって役割が参照される可能性があるため、役割名を変更することはできません。ただし、IAM を使用してロールの説明を編集することはできます。詳細については『IAM ユーザーガイド』の「サービスにリンクされた役割の編集」を参照してください。

Amazon EKS でのサービスにリンクされた役割の削除

サービスリンク役割が必要な機能またはサービスが不要になった場合にはその役割を削除することをお勧めします。そうすることで、モニタリングや保守が積極的に行われていない未使用のエンティティを排除できます。ただし、手動で削除する前に、サービスリンク役割をクリーンアップする必要があります。

サービスリンク役割のクリーンアップ

IAM を使用してサービスにリンクされた役割を削除するには最初に、その役割で使用されているリソースをすべて削除する必要があります。

> **Note**
>
> リソースの削除を試みた際に、対応する役割が Amazon EKS サービスで使用されている場合、削除が失敗することがあります。失敗した場合は数分待ってから操作を再試行してください。

1. Amazon EKS コンソールを開きます。
2. 左のナビゲーションペインで [Clusters] (クラスター) を選択してください。
3. [Clusters] (クラスター) ページで、クラスターを選択してください。
4. [登録解除]、[OK] の順に選択します。

サービスリンク役割の手動による削除

IAM コンソール、AWS CLI、または AWS API を使用して、AWSServiceRoleForAmazonEKSConnector のサービスにリンクされた役割を削除します。詳細については IAM ユーザーガイド の「サービスにリンクされた役割の削除」を参照してください。

アウトポスト で Amazon EKS ローカルクラスターの役割の使用

Amazon エラスティック Kus サービス はAWS アイデンティティとアクセス管理 (IAM) のサービberneteスにリンクされた役割を使用します。サービスにリンクされた役割はAmazon EKS に直接リンクされた一意のタイプの IAM 役割です。サービスにリンクされた役割はAmazon EKS で事前定義されています。この役割にはサービスがユーザーに代わって他の AWS のサービスを呼び出すために必要な、すべてのアクセス許可が付与されています。

サービスにリンクされた役割を使用することで、必要なアクセス許可を手動で追加する必要がなくなるため、Amazon EKS の設定が簡単になります。サービスにリンクされた役割のアクセス許可は Amazon EKS により定義されます。特に指定されている場合を除き、Amazon EKS のみがその役割を引き受けることができます。定義される許可は信頼ポリシーと許可ポリシーに含まれており、その許可ポリシーを他の IAM エンティティにアタッチすることはできません。

サービスリンクロールを削除するには、まずその関連リソースを削除します。これにより、リソースへのアクセス許可を不用意に削除することが防止され、Amazon EKS リソースが保護されます。

サービスにリンクされた役割をサポートする他のサービスについては「IAM と連動する AWS サービス」を参照し、[サービス-linked roles] (サービスにリンクされた役割) の列内で [Yes] (はい) と表記さ

れたサービスを確認してください。サービスにリンクされた役割に関するドキュメントをサービスで表示するには[はい] リンクを選択してください。

Amazon EKS でのサービスにリンクされた役割のアクセス許可

Amazon EKS は`AWSServiceRoleForAmazonEKSLocalOutpost` という名前のサービスにリンクされた役割を使用します。この役割により、Amazon EKS はアカウント内のローカルクラスターを管理できます。アタッチされたポリシーにより、役割はネットワークインターフェイス、セキュリティグループ、ログ、Amazon EC2 インスタンスのリソースを管理できるようになります。

> **Note**
>
> `AWSServiceRoleForAmazonEKSLocalOutpost` サービスにリンクされた役割はクラスターの作成に必要な役割とは異なります。詳細については「the section called “クラスター IAM ロール”」を参照してください。

`AWSServiceRoleForAmazonEKSLocalOutpost` サービスリンク役割は役割の引き受けについて以下のサービスを信頼します。

- `outposts.eks-local.amazonaws.com`

役割のアクセス許可ポリシーは指定したリソースに対して以下のアクションを実行することを Amazon EKS に許可します。

- AmazonEKSLocalOutpostServiceRolePolicy

サービスリンクロールの作成、編集、削除を IAM エンティティ (ユーザー、グループ、ロールなど) に許可するにはアクセス許可を設定する必要があります。詳細についてはIAM ユーザーガイド の「サービスにリンクされた役割のアクセス許可」を参照してください。

Amazon EKS でのサービスにリンクされた役割の作成

サービスリンク役割を手動で作成する必要はありません。AWS マネジメントコンソール、AWS CLI、または AWS API でクラスターを作成すると、Amazon EKS によってサービスリンク役割が作成されます。

このサービスリンク役割を削除した後で再度作成する必要が生じた場合は同じ方法でアカウントに役割を再作成できます。サービスにリンクされた役割はクラスターの作成時に Amazon EKS で自動的に再作成されます。

Amazon EKS でのサービスにリンクされた役割の編集

Amazon EKS では`AWSServiceRoleForAmazonEKSLocalOutpost` のサービスにリンクされた役割を編集することはできません。サービスリンク役割を作成すると、多くのエンティティによって役割が参照される可能性があるため、役割名を変更することはできません ただし、IAM を使用した役割記述の編集はできます。詳細については『IAM ユーザーガイド』の「サービスにリンクされた役割の編集」を参照してください。

Amazon EKS でのサービスにリンクされた役割の削除

サービスリンク役割が必要な機能またはサービスが不要になった場合にはその役割を削除することをお勧めします。そうすることで、モニタリングや保守が積極的に行われていない未使用のエンティティを排除できます。ただし、手動で削除する前に、サービスリンク役割をクリーンアップする必要があります。

サービスリンク役割のクリーンアップ

IAM を使用してサービスにリンクされた役割を削除するには最初に、その役割で使用されているリソースをすべて削除する必要があります。

> **Note**
>
> リソースの削除を試みた際に、対応する役割が Amazon EKS サービスで使用されている場合、削除が失敗することがあります。失敗した場合は数分待ってから操作を再試行してください。

1. Amazon EKS コンソールを開きます。
2. 左のナビゲーションペインで Amazon EKS [Clusters] (クラスター) を選択してください。
3. クラスターにノードグループまたは Fargate プロファイルがある場合はクラスターを削除する前にそれらを削除する必要があります。詳細については「the section called “削除”」および「the section called “プロファイルを削除する”」を参照してください。
4. [Clusters (クラスター)] ページで、削除するクラスターを選択し、[Delete (削除)] を選択してください。
5. 削除確認ウィンドウにクラスター名を入力し、[Delete (削除)] を選択してください。

6. この手順をアカウント内の他のすべてのクラスターに対して繰り返します。すべての削除操作が完了するまで待ちます。

サービスリンク役割の手動による削除

サービスにリンクされた役割 `AWSServiceRoleForAmazonEKSLocalOutpost` を削除するには IAM コンソール、AWS CLI、または AWS API を使用します。詳細についてはIAM ユーザーガイドの「サービスにリンクされた役割の削除」を参照してください。

Amazon EKS のサービスにリンクされた役割がサポートされるリージョン

Amazon EKS ではこのサービスを利用できるすべてのリージョンで、サービスにリンクされた役割の使用がサポートされます。詳細については「Amazon EKS endpoints and quotas」(Amazon EKS エンドポイントとクォータ) を参照してください。

Amazon EKS ダッシュボードのロールの使用

Amazon EKS ダッシュボードでは、このサービスにリンクされたロールを使用して、複数のリージョンとアカウントからクラスターリソースに関する情報を集約します。ダッシュボードは AWS Organizations と統合され、組織内の複数のアカウントに関する情報を安全に読み取ります。

Amazon EKS ダッシュボードの詳細については、「the section called “Amazon EKS ダッシュボード”」を参照してください。

背景

Amazon EKS はAWS アイデンティティとアクセス管理 (IAM) のサービスにリンクされた役割を使用します。サービスにリンクされた役割はAmazon EKS に直接リンクされた一意のタイプの IAM 役割です。サービスにリンクされた役割はAmazon EKS で事前定義されています。この役割にはサービスがユーザーに代わって他の AWS のサービスを呼び出すために必要な、すべてのアクセス許可が付与されています。

サービスにリンクされた役割を使用することで、必要なアクセス許可を手動で追加する必要がなくなるため、Amazon EKS の設定が簡単になります。サービスにリンクされた役割のアクセス許可は Amazon EKS により定義されます。特に指定されている場合を除き、Amazon EKS のみがその役割を引き受けることができます。定義される許可は信頼ポリシーと許可ポリシーに含まれており、その許可ポリシーを他の IAM エンティティにアタッチすることはできません。

サービスリンクロールを削除するには、まずその関連リソースを削除します。これにより、リソースへのアクセス許可を不用意に削除することが防止され、Amazon EKS リソースが保護されます。

サービスにリンクされた役割をサポートする他のサービスについては「IAM と連動する AWS サービス」を参照し、[サービス-linked roles] (サービスにリンクされた役割) の列内で [Yes] (はい) と表記されたサービスを確認してください。サービスにリンクされた役割に関するドキュメントをサービスで表示するには[はい] リンクを選択してください。

Amazon EKS でのサービスにリンクされた役割のアクセス許可

Amazon EKS はAWSServiceRoleForAmazonEKSDashboard という名前のサービスにリンクされた役割を使用します。このロールにより、Amazon EKS はクラスターリソースに関して集約された情報を含む EKS ダッシュボードを生成して表示できます。アタッチされた AmazonEKSDashboardServiceRolePolicy ポリシーにより、ロールは 自動スケーリング グループ、セキュリティグループ、起動テンプレート、および IAM インスタンスプロファイルのリソースを管理できるようになります。詳細については、「the section called “AWS マネージドポリシー: AmazonEKSDashboardServiceRolePolicy”」を参照してください。

AWSServiceRoleForAmazonEKSDashboard サービスリンク役割は役割の引き受けについて以下のサービスを信頼します。

- dashboard.eks.amazonaws.com

JSON ポリシードキュメントの最新バージョンを確認するには『AWS管理ポリシーリファレンスガイド』の「AmazonEKSDashboardServiceRolePolicy」を参照してください。

サービスリンクロールの作成、編集、削除を IAM エンティティ (ユーザー、グループ、ロールなど) に許可するにはアクセス許可を設定する必要があります。詳細についてはIAM ユーザーガイド の「サービスにリンクされた役割のアクセス許可」を参照してください。

Amazon EKS でのサービスにリンクされた役割の作成

サービスリンク役割を手動で作成する必要はありません。AWS コンソールでダッシュボードを有効にすると、Amazon EKS によってサービスにリンクされたロールが作成されます。

EKS ダッシュボードの有効化についての詳細は、「the section called “組織を設定する”」を参照してください。

> **Important**
>
> このサービスリンクロールはこのロールでサポートされている機能を使用する別のサービスでアクションが完了した場合にアカウントに表示されます。

Amazon EKS でのサービスにリンクされた役割の編集

Amazon EKS では`AWSServiceRoleForAmazonEKSDashboard` のサービスにリンクされた役割を編集することはできません。サービスにリンクされた役割を作成すると、多くのエンティティによって役割が参照される可能性があるため、役割名を変更することはできません。ただし、IAM を使用してロールの説明を編集することはできます。詳細については『IAM ユーザーガイド』の「サービスにリンクされた役割の編集」を参照してください。

Amazon EKS でのサービスにリンクされた役割の削除

サービスリンク役割が必要な機能またはサービスが不要になった場合にはその役割を削除することをお勧めします。そうすることで、モニタリングや保守が積極的に行われていない未使用のエンティティを排除できます。ただし、手動で削除する前に、サービスリンク役割をクリーンアップする必要があります。

サービスリンク役割のクリーンアップ

IAM を使用してサービスにリンクされた役割を削除するには最初に、その役割で使用されているリソースをすべて削除する必要があります。

> **Note**
>
> リソースの削除を試みた際に、対応する役割が Amazon EKS サービスで使用されている場合、削除が失敗することがあります。失敗した場合は数分待ってから操作を再試行してください。

EKS ダッシュボードの無効化についての詳細は、「the section called “組織を設定する”」を参照してください。

サービスリンク役割の手動による削除

サービスにリンクされた役割 `AWSServiceRoleForAmazonEKSDashboard` を削除するにはIAM コンソール、AWS CLI、または AWS API を使用します。詳細についてはIAM ユーザーガイド の「サービスにリンクされた役割の削除」を参照してください。

Amazon EKS のサービスにリンクされた役割がサポートされるリージョン

Amazon EKS ではこのサービスを利用できるすべてのリージョンで、サービスにリンクされた役割の使用がサポートされます。詳細については「Amazon EKS endpoints and quotas」(Amazon EKS エンドポイントとクォータ) を参照してください。

Amazon EKS Pod 実行 IAM ロール

Pod を AWS Fargate インフラストラクチャで実行するには、Amazon EKS Pod 実行ロールが必要になります。

クラスターが AWS Fargate インフラストラクチャ上で Pod を作成する場合は、Fargate インフラストラクチャ上で実行されているコンポーネントがユーザーに代わって AWS API を呼び出す必要があります。これは、Amazon ECR からコンテナイメージをプルしたり、ログを他の AWS サービスにルーティングしたりするなどのアクションを実行できるようにするためです。Amazon EKS の Pod 実行ロールにより、これらを行うための IAM アクセス許可が付与されます。

Fargate プロファイルを作成するときは、プロファイルを使用して Fargate インフラストラクチャで Amazon EKS コンポーネントを実行できるように、Pod 実行ロールを指定する必要があります。このロールは、承認のためにクラスターの Kubernetes ロールベースのアクセス制御 (RBAC) に追加されます。これにより、Fargate インフラストラクチャで実行されている `kubelet` が Amazon EKS クラスターに登録され、クラスター内でノードとして表示されるようになります。

> **Note**
>
> Fargate プロファイルには、Amazon EC2 ノードグループとは異なる IAM ロールが必要です。

> **Important**
>
> Fargate Pod で実行されているコンテナは、Pod 実行ロールに関連付けられた IAM アクセス許可を引き受けることはできません。Fargate Pod 内のコンテナに他の AWS サービスへのアクセス許可を付与するには、IAM ロールをサービスアカウントとして使用する必要があります。

Fargate プロファイルを作成する前に、AmazonEKSFargatePodExecutionRolePolicy で IAM ロールを作成する必要があります。

正しく設定された既存の Pod 実行ロールをチェックする

次の手順を使用して、正しく設定された Amazon EKS の Pod 実行ロールがアカウントに既に存在しているかどうかをチェックします。「混乱した代理」のセキュリティ上の問題を回避するには、ロー

ルが SourceArn に基づいてアクセスを制限することが重要です。必要に応じて実行ロールを変更し、他のクラスター上で Fargate プロファイルのサポートを含めることができます。

1. IAM コンソール (https://console.aws.amazon.com/iam/) を開きます。
2. 左のナビゲーションペインで、[ロール] を選択してください。
3. [ロール] ページで、ロールの一覧から AmazonEKSFargatePodExecutionRole を検索します。ロールが存在しない場合は、「the section called “Amazon EKS での Pod 実行ロールの作成”」を参照してロールを作成します。ロールが存在する場合は、そのロールを選択します。
4. [AmazonEKSFargatePodExecutionRole] ページで、次の操作を実行します。
   a. [許可] を選択してください。
   b. AmazonEKSFargatePodExecutionRolePolicy Amazon マネージドポリシーがロールにアタッチされていることを確認します。
   c. [信頼関係] を選択します。
   d. [信頼ポリシーを編集] を選択します。
5. [信頼ポリシーを編集] ページで、信頼関係に次のポリシーが含まれており、クラスター上で Fargate プロファイルの行が存在していることを確認します。そうである場合は、[キャンセル] を選択します。

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Condition": {
         "ArnLike": {
            "aws:SourceArn": "arn:aws:eks:us-east-1:111122223333:fargateprofile/my-
cluster/*"
         }
      },
      "Principal": {
        "Service": "eks-fargate-pods.amazonaws.com"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
```

ポリシーは一致するが、クラスター上で Fargate プロファイルを指定する行が存在しない場合は、ArnLike オブジェクトの上部に次の行を追加します。*region-code* はクラスターがある AWS リージョンに、*111122223333* はアカウント ID に、*my-cluster* はクラスターの名前に置き換えます。

```
"aws:SourceArn": "arn:aws:eks:region-code:111122223333:fargateprofile/my-cluster/*",
```

ポリシーが一致しない場合は、前のポリシー全体をフォームにコピーして、[ポリシーの更新] を選択します。*region-code* を、クラスターのある AWS リージョンに置き換えます。アカウントで、すべての AWS リージョンで同じロールを使用する場合は、*region-code* を * に置き換えます。*111122223333* をアカウント ID に、*my-cluster* をクラスター名に置き換えます。アカウント内のすべてのクラスターに同じロールを使用する場合は、*my-cluster* を * に置き換えます。

Amazon EKS での Pod 実行ロールの作成

クラスター用の Amazon EKS Pod 実行ロールをまだ作成していない場合は、AWS マネジメントコンソール または AWS CLI を使用して作成できます。

AWS マネジメントコンソール

a. IAM コンソール (https://console.aws.amazon.com/iam/) を開きます。

b. 左のナビゲーションペインで、[ロール] を選択してください。

c. [ロール] ページで、[ロールの作成] を選択してください。

d. [信頼されたエンティティを選択] ページで、以下の操作を実行してください：

   i. [信頼するエンティティのタイプ] で [AWS サービス] を選択します。

   ii. [他の AWS サービスのユースケース] ドロップダウンリストから、[EKS] を選択します。

   iii. [EKS - Fargate Pod] を選択します。

   iv. [次へ] を選択します。

e. [アクセス許可を追加] ページで [次へ] を選択します。

f. [名前を付けて、レビューし、作成する] ページで、以下の操作を実行します。

   i. [ロール名] に、AmazonEKSFargatePodExecutionRole などのロールの一意の名前を入力します。

ii. [タグの追加 (オプション)] で、タグをキーバリューのペアとして添付して、メタデータをロールに追加します。IAM でのタグの使用に関する詳細については『IAM ユーザーガイド』の「IAM リソースにタグを付ける」を参照してください。

iii. [ロールの作成] を選択してください。

g. [ロール] ページで、ロールの一覧から AmazonEKSFargatePodExecutionRole を検索します。ロールを選択します。

h. [AmazonEKSFargatePodExecutionRole] ページで、次の操作を実行します。

i. [信頼関係] を選択します。

ii. [信頼ポリシーを編集] を選択します。

i. [信頼ポリシーを編集] ページで、次の操作を実行します。

i. 次の内容をコピーして、[信頼ポリシーを編集] フォームに貼り付けます。*region-code* を、クラスターのある AWS リージョンに置き換えます。アカウントで、すべての AWS リージョンで同じロールを使用する場合は、*region-code* を * に置き換えます。*111122223333* をアカウント ID に、*my-cluster* をクラスター名に置き換えます。アカウント内のすべてのクラスターに同じロールを使用する場合は、*my-cluster* を * に置き換えます。

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Condition": {
         "ArnLike": {
            "aws:SourceArn": "arn:aws:eks:us-
east-1:111122223333:fargateprofile/my-cluster/*"
         }
      },
      "Principal": {
        "Service": "eks-fargate-pods.amazonaws.com"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
```

ii. [ポリシーの更新] を選択してください。

AWS CLI

a. 次の内容をコピーして、pod-execution-role-trust-policy.json という名前のファイルに貼り付けます。*region-code* を、クラスターのある AWS リージョンに置き換えます。アカウントで、すべての AWS リージョンで同じロールを使用する場合は、*region-code* を * に置き換えます。*111122223333* をアカウント ID に、*my-cluster* をクラスター名に置き換えます。アカウント内のすべてのクラスターに同じロールを使用する場合は、*my-cluster* を * に置き換えます。

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Condition": {
         "ArnLike": {
            "aws:SourceArn": "arn:aws:eks:us-east-1:111122223333:fargateprofile/
my-cluster/*"
         }
      },
      "Principal": {
        "Service": "eks-fargate-pods.amazonaws.com"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
```

b. Pod 実行 IAM ロールを作成します。

```
aws iam create-role \
  --role-name AmazonEKSFargatePodExecutionRole \
  --assume-role-policy-document file://"pod-execution-role-trust-policy.json"
```

c. このロールに、必要な Amazon EKS 管理の IAM ポリシーをアタッチします。

```
aws iam attach-role-policy \
  --policy-arn arn:aws:iam::aws:policy/AmazonEKSFargatePodExecutionRolePolicy \
  --role-name AmazonEKSFargatePodExecutionRole
```

Amazon EKS コネクタの IAM ロール

Kubernetes クラスターを接続して AWS マネジメントコンソール に表示することができます。Kubernetes クラスターに接続するには、IAM ロールを作成します。

既存の EKS Connector ロールの確認

以下の手順を使用して、アカウントに既に Amazon EKS コネクタロールがあるかどうかを確認できます。

1. IAM コンソール (https://console.aws.amazon.com/iam/) を開きます。
2. 左のナビゲーションペインで、[ロール] を選択してください。
3. ロールのリストで `AmazonEKSConnectorAgentRole` を検索します。`AmazonEKSConnectorAgentRole` が含まれているロールが存在しない場合は、the section called “Amazon EKS コネクタエージェントロールの作成” を参照してロールを作成します。`AmazonEKSConnectorAgentRole` が含まれているロールが存在する場合はこのロールを選択してアタッチされているポリシーを表示します。
4. [許可] を選択してください。
5. AmazonEKSConnectorAgentPolicy 管理ポリシーがロールにアタッチされていることを確認します。ポリシーがアタッチされている場合、Amazon EKS Connector ロールは適切に設定されています。
6. [信頼関係] を選択し、[信頼ポリシーの編集] を選択してください。
7. 信頼関係に以下のポリシーが含まれていることを確認します。信頼関係が以下のポリシーと一致する場合、[キャンセル] を選択してください。信頼関係が一致しない場合、ポリシーを [信頼ポリシーの編集] ウィンドウにコピーし、[ポリシーの更新] を選択します。

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "Service": [
                    "ssm.amazonaws.com"
                ]
            },
            "Action": "sts:AssumeRole"
        }
    ]
```

```
}
```

Amazon EKS コネクタエージェントロールの作成

コネクタエージェントロールを作成するには、AWS マネジメントコンソール または AWS CloudFormation を使用できます。

AWS CLI

a. IAM ロールに使用する次の JSON が含まれる `eks-connector-agent-trust-policy.json` という名前のファイルを作成します。

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "Service": [
                    "ssm.amazonaws.com"
                ]
            },
            "Action": "sts:AssumeRole"
        }
    ]
}
```

b. IAM ロールに使用する次の JSON が含まれる `eks-connector-agent-policy.json` という名前のファイルを作成します。

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "SsmControlChannel",
            "Effect": "Allow",
            "Action": [
                "ssmmessages:CreateControlChannel"
            ],
            "Resource": "arn:aws:eks:*:*:cluster/*"
        },
        {
```

```
            "Sid": "ssmDataplaneOperations",
            "Effect": "Allow",
            "Action": [
                "ssmmessages:CreateDataChannel",
                "ssmmessages:OpenDataChannel",
                "ssmmessages:OpenControlChannel"
            ],
            "Resource": "*"
        }
    ]
}
```

c. 前のリストアイテムで作成した信頼ポリシーとポリシーを使用して、Amazon EKS Connector エージェントロールを作成します。

```
aws iam create-role \
     --role-name AmazonEKSConnectorAgentRole \
     --assume-role-policy-document file://eks-connector-agent-trust-policy.json
```

d. Amazon EKS Connector エージェントのロールにポリシーを添付します。

```
aws iam put-role-policy \
     --role-name AmazonEKSConnectorAgentRole \
     --policy-name AmazonEKSConnectorAgentPolicy \
     --policy-document file://eks-connector-agent-policy.json
```

AWS CloudFormation

a. 以下の AWS CloudFormation テンプレートを、テキストファイルとしてローカルシステムに保存します。

> **Note**
>
> このテンプレートでは、ここで作成されなければ `registerCluster` API が呼び出されたときに作成される、サービスにリンクされたロールも作成されます。詳細については、「the section called “クラスターコネクタの役割”」を参照してください。

```
---
AWSTemplateFormatVersion: '2010-09-09'
Description: 'Provisions necessary resources needed to register clusters in EKS'
Parameters: {}
```

```
Resources:
  EKSConnectorSLR:
    Type: AWS::IAM::ServiceLinkedRole
    Properties:
      AWSServiceName: eks-connector.amazonaws.com

  EKSConnectorAgentRole:
    Type: AWS::IAM::Role
    Properties:
      AssumeRolePolicyDocument:
        Version: '2012-10-17'
        Statement:
          - Effect: Allow
            Action: [ 'sts:AssumeRole' ]
            Principal:
              Service: 'ssm.amazonaws.com'

  EKSConnectorAgentPolicy:
    Type: AWS::IAM::Policy
    Properties:
      PolicyName: EKSConnectorAgentPolicy
      Roles:
        - {Ref: 'EKSConnectorAgentRole'}
      PolicyDocument:
        Version: '2012-10-17'
        Statement:
          - Effect: 'Allow'
            Action: [ 'ssmmessages:CreateControlChannel' ]
            Resource:
            - Fn::Sub: 'arn:${AWS::Partition}:eks:*:*:cluster/*'
          - Effect: 'Allow'
            Action: [ 'ssmmessages:CreateDataChannel',
 'ssmmessages:OpenDataChannel', 'ssmmessages:OpenControlChannel' ]
            Resource: "*"
Outputs:
  EKSConnectorAgentRoleArn:
    Description: The agent role that EKS connector uses to communicate with AWS
 services.
    Value: !GetAtt EKSConnectorAgentRole.Arn
```

b. AWS CloudFormation コンソールを開きます。

c. 新しいリソースを使用して [スタックを作成] を選択します (標準)。

d. [テンプレートの指定] で、[テンプレートファイルのアップロード] を選択し、[ファイルの選択] を選択します。

e. 作成したファイルを選択し、[Next (次へ)] を選択します。

f. [スタックの名前] に `eksConnectorAgentRole` などのロール名を入力し、[次へ] を選択します。

g. [スタックオプションの設定] ページで、[Next (次へ)] を選択します。

h. [Review (レビュー)] ページの情報から、スタックにより IAM リソースが作成されることを確認し、[Create stack (スタックの作成)] を選択します。

Amazon Elastic Kubernetes Service の AWS マネージドポリシー

AWS マネージドポリシーは AWS が作成および管理するスタンドアロンポリシーです。AWS マネージドポリシーは多くの一般的なユースケースに対してアクセス許可を提供するように設計されているため、ユーザー、グループ、ロールへのアクセス権の割り当てを開始できます。

AWS マネージドポリシーは特定のユースケースに対して最小特権の許可を付与しない場合があることに注意してください。これはすべての AWS 顧客が使用できる状態になっているためです。ユースケースに固有のカスタマー管理ポリシーを定義して、アクセス許可を絞り込むことをお勧めします。

AWS マネージドポリシーで定義したアクセス権限は変更できません。AWS が AWS マネージドポリシーに定義されているアクセス許可を更新すると、更新はポリシーがアタッチされているすべてのプリンシパルアイデンティティ (ユーザー、グループ、ロール) に影響します。新しい AWS サービスを起動するか、既存のサービスで新しい API オペレーションが使用可能になると、AWS が AWS マネージドポリシーを更新する可能性が最も高くなります。

詳細については、IAM ユーザーガイドの「AWS マネージドポリシー」を参照してください。

AWS マネージドポリシー: AmazonEKS_CNI_Policy

`AmazonEKS_CNI_Policy` ポリシーを IAM エンティティにアタッチできます。Amazon EC2 ノードグループを作成する前に、このポリシーをノードの IAM ロール、または Amazon VPC CNI Plugin for Kubernetes 専用で使用する IAM ロールにアタッチする必要があります。これはユーザーに代わってアクションを実行できるようにするためです。プラグインでのみ使用されるロールにポリシーをアタッチすることをお勧めします。詳細については、「the section called “Amazon VPC CNI”」および「the section called “IRSA 用の設定”」を参照してください。

アクセス許可の詳細

このポリシーには Amazon EKS に以下のタスクを完了させるための以下の権限が含まれています。

- **`ec2:*NetworkInterface`** および **`ec2:*PrivateIpAddresses`** – Amazon VPC CNI プラグインが Pod の Elastic Network Interface や IP アドレスのプロビジョニングなどのアクションを実行し、Amazon EKS で実行されるアプリケーションにネットワークを提供できるようにします。
- **`ec2`** 読み取りアクション – Amazon VPC CNI プラグインがインスタンスやサブネットを記述して、Amazon VPC サブネット内の空き IP アドレスの量を確認するなどのアクションを実行できるようにします。VPC CNI は各サブネットの空き IP アドレスを使用して、エラスティック・ネットワーク・インターフェイス の作成時に使用する空き IP アドレスが最も多いサブネットを選択できます。

JSON ポリシードキュメントの最新バージョンを確認するには、AWS マネージドポリシーのリファレンスガイドの「AmazonEKS_CNI_Policy」を参照してください。

AWS マネージドポリシー: AmazonEKSClusterPolicy

IAM エンティティに `AmazonEKSClusterPolicy` をアタッチできます。クラスターを作成する前に、このポリシーをアタッチしたクラスター IAM ロールが必要となります。Amazon EKS によって管理される Kubernetes クラスターは、ユーザーに代わって他の AWS のサービスを呼び出します。これにより、サービスで使用するリソースを管理します。

このポリシーにはAmazon EKS に以下のタスクを完了させるための以下の権限が含まれています。

- **`autoscaling`** – Auto Scaling グループの設定を読み取り、更新します。これらの権限は Amazon EKS では使用されませんが、下位互換性のためにポリシーに残ります。
- **`ec2`** – Amazon EC2 ノードに関連付けられているボリュームとネットワークリソースを操作します。これは、Kubernetes コントロールプレーンがインスタンスをクラスターに参加させ、Kubernetes 永続ボリュームによってリクエストされる Amazon EBS ボリュームを動的にプロビジョニングおよび管理できるようにするために必要です。
- **`ec2`** – VPC CNI によって作成された Elastic Network Interface を削除します。こうした権限付与が必要なのは、VPC CNI が予期せず終了した場合に、残った Elastic Network Interface を EKS によってクリーンアップするためです。
- **`elasticloadbalancing`** – Elastic Load Balancers を操作し、ノードをターゲットとして追加します。これは、Kubernetes コントロールプレーンが Kubernetes サービスによってリクエストされる Elastic Load Balancer を動的にプロビジョニングできるようにするために必要です。
- **`iam`** – サービスにリンクされたロールを作成します。これは、Kubernetes コントロールプレーンが Kubernetes サービスによってリクエストされる Elastic Load Balancer を動的にプロビジョニングできるようにするために必要です。

- **kms** – AWS KMS からキーを読み取ります。これは、Kubernetes コントロールプレーンが etcd に保存される Kubernetes シークレットのシークレット暗号化を管理するために必要です。

JSON ポリシードキュメントの最新バージョンを確認するには、AWS マネージドポリシーのファレンスガイドの「AmazonEKSClusterPolicy」を参照してください。

AWS マネージドポリシー: AmazonEKSDashboardConsoleReadOnly

IAM エンティティに AmazonEKSDashboardConsoleReadOnly をアタッチできます。

このポリシーにはAmazon EKS に以下のタスクを完了させるための以下の権限が含まれています。

- **eks** – EKS ダッシュボードのデータ、リソース、クラスターバージョン情報への読み取り専用アクセス。これにより、EKS 関連のメトリクスおよびクラスター設定の詳細を表示できます。
- **organizations** – 以下のような AWS Organizations 情報への読み取り専用アクセス。
  - 組織の詳細とサービスアクセスの表示
  - 組織のルート、アカウント、組織単位の一覧表示
  - 組織構造の表示

JSON ポリシードキュメントの最新バージョンを確認するには、AWS マネージドポリシーのリファレンスガイドの「AmazonEKSDashboardConsoleReadOnly」を参照してください。

AWS マネージドポリシー: AmazonEKSFargatePodExecutionRolePolicy

IAM エンティティに AmazonEKSFargatePodExecutionRolePolicy をアタッチできます。Fargate プロファイルを作成する前に、Fargate Pod 実行ロールを作成し、このポリシーをアタッチする必要があります。詳細については、the section called “ステップ 2: Fargate Pod 実行ロールを作成する” および the section called “プロファイルを定義する” を参照してください。

このポリシーはロールに対して、Fargate で Amazon EKS Pod を実行するために必要な他の AWS サービスリソースにアクセスするための権限を付与します。

アクセス許可の詳細

このポリシーにはAmazon EKS に以下のタスクを完了させるための以下の権限が含まれています。

- **ecr** – Fargate で実行中のポッドが、Amazon ECR に格納されているコンテナイメージを取得できるようにします。

JSON ポリシードキュメントの最新バージョンを確認するには、AWS マネージドポリシーのリファレンスガイドの「AmazonEKSFargatePodExecutionRolePolicy」を参照してください。

AWS マネージドポリシー: AmazonEKSConnectorServiceRolePolicy

IAM エンティティに AmazonEKSConnectorServiceRolePolicy をアタッチすることはできません。このポリシーはユーザーに代わって Amazon EKS がアクションを実行することを許可する、サービスにリンクされたロールにアタッチされます。詳細については、「the section called “クラスターコネクタの役割”」を参照してください。

このロールにより、Amazon EKS は Kubernetes クラスターに接続できます。アタッチされたポリシーにより、ロールは、登録された Kubernetes クラスターに接続するために必要なリソースを管理できます。

アクセス許可の詳細

このポリシーには Amazon EKS が以下のタスクを完了できるようにする以下のアクセス許可が含まれています。

- **SSM Management** – SSM アクティベーションを作成、説明、削除し、マネージドインスタンスの登録を解除します。これにより、基本的な Systems Manager オペレーションが可能になります。
- **Session Management** – EKS クラスター専用の SSM セッションを開始し、AmazonEKS ドキュメントを使用して非インタラクティブコマンドを実行します。
- **IAM Role Passing** – SSM サービスに対して特定の IAM ロールを渡します。この操作は、渡されるロールを ssm.amazonaws.com に制限する条件によって制御されます。
- **EventBridge Rules** – EventBridge ルールとターゲットを作成します。ただし、eks-connector.amazonaws.com によって管理される場合にのみ許可されます。ルールは、イベントソースとして AWS SSM のみに制限されます。

JSON ポリシードキュメントの最新バージョンを確認するには、AWS マネージドポリシーのリファレンスガイドの「AmazonEKSConnectorServiceRolePolicy」を参照してください。

AWS マネージドポリシー: AmazonEKSForFargateServiceRolePolicy

IAM エンティティに AmazonEKSForFargateServiceRolePolicy をアタッチすることはできません。このポリシーはユーザーに代わって Amazon EKS がアクションを実行することを許可する、サービスにリンクされたロールにアタッチされます。詳細については、「AWSServiceRoleForAmazonEKSForFargate」を参照してください。

このポリシーは Fargate タスクを実行するために必要なアクセス権限を Amazon EKS に付与します。このポリシーはFargate ノードがある場合にのみ使用されます。

アクセス許可の詳細

このポリシーにはAmazon EKS が以下のタスクを完了できるようにする以下の権限が含まれています。

- **ec2** – Elastic Network Interfaces を作成および削除し、Elastic Network Interfaces とリソースについて説明します。これは Amazon EKS Fargate サービスが Fargate ポッドに必要な VPC ネットワーキングを設定できるようにするために必要です。

JSON ポリシードキュメントの最新バージョンを確認するには、AWS マネージドポリシーのリファレンスガイドの「AmazonEKSForFargateServiceRolePolicy」を参照してください。

AWS マネージドポリシー: AmazonEKSComputePolicy

IAM エンティティに `AmazonEKSComputePolicy` をアタッチできます。このポリシーをクラスター IAM ロールにアタッチして、EKS がアカウントで管理できるリソースを拡張できます。

このポリシーによって Amazon EKS が EKS クラスター用の EC2 インスタンスを作成および管理するために必要なアクセス許可と、EC2 を設定するために必要な IAM アクセス許可が付与されます。さらに、このポリシーによって、Amazon EKS がユーザーに代わって EC2 スポットサービスにリンクされたロールを作成するためのアクセス許可が付与されます。

アクセス許可の詳細

このポリシーには Amazon EKS に以下のタスクを完了させるための以下の権限が含まれています。

- **ec2** 許可:
  - `ec2:CreateFleet` および `ec2:RunInstances` – EC2 インスタンスの作成と、EKS クラスターノードのための特定の EC2 リソース (イメージ、セキュリティグループ、サブネット、オンデマンドキャパシティ予約) の使用を許可します。
  - `ec2:CreateLaunchTemplate` – EKS クラスターノードのための EC2 起動テンプレートの作成を許可します。
  - また、このポリシーにはこれらの EC2 許可の使用を、EKS クラスター名および他の関連タグでタグ付けされたリソースに制限する条件も含まれています。
  - `ec2:CreateTags` – `CreateFleet`、`RunInstances`、および `CreateLaunchTemplate` アクションによって作成された EC2 リソースへのタグの追加を許可します。

- `ec2:DescribeCapacityReservations` – 中断可能なキャパシティ予約からの EC2 インスタンスの作成を許可します
- **`iam`** 許可:
  - `iam:AddRoleToInstanceProfile` – EKS コンピューティングインスタンスプロファイルへの IAM ロールの追加を許可します。
  - `iam:PassRole` – 必要な IAM ロールを EC2 サービスに渡すことを許可します。

JSON ポリシードキュメントの最新バージョンを確認するには、AWS マネージドポリシーリファレンスガイドの「AmazonEKSComputePolicy」を参照してください。

AWS マネージドポリシー: AmazonEKSNetworkingPolicy

IAM エンティティに `AmazonEKSNetworkingPolicy` をアタッチできます。このポリシーをクラスター IAM ロールにアタッチして、EKS がアカウントで管理できるリソースを拡張できます。

このポリシーは Amazon EKS が EKS クラスターのネットワークインターフェイスを作成および管理するために必要なアクセス許可を付与し、コントロールプレーンとワーカーノードが適切に通信および機能するように設計されています。

アクセス許可の詳細

このポリシーは Amazon EKS がクラスターのネットワークインターフェイスを管理できるようにする以下のアクセス許可を付与します。

- **`ec2`**ネットワークインターフェイスアクセス許可:
  - `ec2:CreateNetworkInterface` – EC2 ネットワークインターフェイスの作成を許可します。
  - このポリシーにはこのアクセス許可の使用を EKS クラスター名と Kubernetes CNI ノード名でタグ付けされたネットワークインターフェイスに制限するための条件が含まれています。
  - `ec2:CreateTags` – `CreateNetworkInterface` アクションによって作成されたネットワークインターフェイスへのタグの追加を許可します。
- **`ec2`** ネットワークインターフェイス管理のアクセス許可:
  - `ec2:AttachNetworkInterface`、`ec2:ModifyNetworkInterfaceAttribute`、`ec2:DetachNet` – EC2 インスタンスへのネットワークインターフェイス属性のアタッチ、変更、およびネットワークインターフェイスのデタッチを許可します。
  - `ec2:UnassignPrivateIpAddresses`、`ec2:UnassignIpv6Addresses`、`ec2:AssignPrivateIp` – ネットワークインターフェイスの IP アドレス割り当ての管理を許可します。

- これらのアクセス許可は EKS クラスター名でタグ付けされたネットワークインターフェイスに制限されます。

JSON ポリシードキュメントの最新バージョンを確認するには、AWS マネージドポリシーのリファレンスガイドの「AmazonEKSNetworkingPolicy」を参照してください。

AWS マネージドポリシー: AmazonEKSBlockStoragePolicyV2

IAM エンティティに AmazonEKSBlockStoragePolicyV2 をアタッチできます。このポリシーをクラスター IAM ロールにアタッチして、EKS がアカウントで管理できるリソースを拡張できます。

このポリシーは Amazon EKS が EKS クラスターのために EC2 ボリュームとスナップショットを作成、管理、メンテナンスするために必要な許可を付与し、Kubernetes ワークロードが必要とする永続的ストレージをコントロールプレーンとワーカーノードがプロビジョニングおよび使用できるようにします。

アクセス許可の詳細

この IAM ポリシーは Amazon EKS が EC2 ボリュームとスナップショットを管理できるように、次の許可を付与します:

- **ec2**ボリューム管理の許可:
  - ec2:AttachVolume、ec2:DetachVolume、ec2:ModifyVolume、ec2:EnableFastSnapshotRe – EC2 ボリュームの高速スナップショット復元のアタッチ、デタッチ、変更、有効化を許可します。
  - これらの許可は EKS クラスター名でタグ付けされたボリュームに制限されます。
  - ec2:CreateTags – CreateVolume および CreateSnapshot アクションによって作成された EC2 ボリュームとスナップショットへのタグの追加を許可します。
- **ec2**ボリューム作成の許可:
  - ec2:CreateVolume – 新しい EC2 ボリュームの作成を許可します。
  - このポリシーにはこの許可の使用を、EKS クラスター名および他の関連タグでタグ付けされたボリュームに制限する条件が含まれています。
  - ec2:CreateSnapshot – 新しい EC2 ボリュームスナップショットの作成を許可します。
  - このポリシーにはこの許可の使用を、EKS クラスター名および他の関連タグでタグ付けされたスナップショットに制限する条件が含まれています。

JSON ポリシードキュメントの最新バージョンを確認するには、AWS マネージドポリシーのリファレンスガイドの「AmazonEKSBlockStoragePolicyV2」を参照してください。

AWS マネージドポリシー: AmazonEKSLoadBalancingPolicy

IAM エンティティに AmazonEKSLoadBalancingPolicy をアタッチできます。このポリシーをクラスター IAM ロールにアタッチして、EKS がアカウントで管理できるリソースを拡張できます。

この IAM ポリシーは Amazon EKS が Elastic ロードバランサー (ELB) および関連リソースを管理するためにさまざまな AWS サービスと連携するために必要な許可を付与します。

アクセス許可の詳細

このポリシーによって付与される主な許可は次のとおりです:

- **elasticloadbalancing** : Elastic Load Balancers とターゲットグループの作成、変更、管理を許可します。これにはロードバランサー、ターゲットグループ、リスナー、ルールを作成、更新、削除するための許可が含まれます。
- **ec2** : Kubernetes コントロールプレーンがインスタンスをクラスターに参加させ、Amazon EBS ボリュームを管理するために必要なセキュリティグループの作成と管理を許可します。また、インスタンス、VPC、サブネット、セキュリティ グループ、その他のネットワーク リソースなどの EC2 リソースを記述および一覧表示することもできます。
- **iam** : Kubernetes コントロールプレーンが ELB を動的にプロビジョニングするために必要な、Elastic Load Balancing 用のサービスにリンクされたロールの作成を許可します。
- **kms**: etcd に保存されている Kubernetes シークレットの暗号化をサポートするために Kubernetes コントロールプレーンが必要とする AWS KMS からのキーの読み取りを許可します。
- **wafv2** および **shield**: ウェブ ACL の一覧表示、関連付け、関連付け解除、および Elastic ロードバランサー のための AWS Shield 保護の作成/削除を許可します。
- **cognito-idp**、**acm**、および **elasticloadbalancing**: ユーザープールクライアントの記述、証明書の一覧表示および記述、ならびにターゲットグループの記述を実行するための許可を付与します。これはKubernetes コントロールプレーンが Elastic ロードバランサー を管理するために必要です。

また、このポリシーにはeks:eks-cluster-name タグを使用して、許可の範囲が管理対象の特定の EKS クラスターに設定されているようにするための条件チェックもいくつか含まれています。

JSON ポリシードキュメントの最新バージョンを確認するには、AWS マネージドポリシーのリファレンスガイドの「AmazonEKSLoadBalancingPolicy」を参照してください。

AWS マネージドポリシー: AmazonEKSMCPReadOnlyAccess

IAM エンティティに `AmazonEKSMCPReadOnlyAccess` をアタッチできます。このポリシーは、Amazon EKS リソースおよび関連する AWS サービスへの読み取り専用アクセスを提供し、Amazon EKS モデルコンテキストプロトコル (MCP) サーバーがインフラストラクチャを変更することなくオブザーバビリティおよびトラブルシューティングのオペレーションを実行できるようにします。

アクセス許可の詳細

このポリシーには、プリンシパルに以下のタスクを完了させるための以下の権限が含まれています。

- **`eks`** – プリンシパルが EKS クラスター、ノードグループ、アドオン、アクセスエントリ、インサイトに対して describe および一覧表示を行い、読み取り専用オペレーションのために Kubernetes API にアクセスできるようにします。
- **`iam`** – プリンシパルが IAM ロール、ポリシー、およびそれらのアタッチメントに関する情報を取得して、EKS リソースに関連付けられたアクセス許可を理解できるようにします。
- **`ec2`** – プリンシパルが VPC、サブネット、ルートテーブルを describe して、EKS クラスターのネットワーク設定を理解できるようにします。
- **`sts`** – プリンシパルが認証および認可の目的で発信者 ID 情報を取得できるようにします。
- **`logs`** – プリンシパルが、トラブルシューティングとモニタリングのために、CloudWatch Logs でクエリを開始し、クエリ結果を取得できるようにします。
- **`cloudwatch`** – プリンシパルがクラスターとワークロードのパフォーマンスをモニタリングするためのメトリクスデータを取得できるようにします。
- **`eks-mcp`** – プリンシパルが MCP オペレーションを呼び出し、Amazon EKS MCP サーバー内で読み取り専用ツールを呼び出せるようにします。

JSON ポリシードキュメントの最新バージョンを確認するには、AWS マネージドポリシーのリファレンスガイドの「AmazonEKSMCPReadOnlyAccess」を参照してください。

AWS マネージドポリシー: AmazonEKSServicePolicy

IAM エンティティに `AmazonEKSServicePolicy` をアタッチできます。2020 年 4 月 16 日より前に作成されたクラスターでは IAM ロールを作成し、このポリシーをアタッチする必要がありました。2020 年 4 月 16 日以降に作成されたクラスターではロールを作成する必要はなく、このポリシーの割り当ても必要ありません。`iam:CreateServiceLinkedRole` アクセス許可を持つ IAM プリンシパルを使用してクラスターを作成する場合、AWSServiceRoleForAmazonEKS のサービスに

リンクされたロールが自動的に作成されます。このサービスにリンクされたロールには マネージドポリシー: AmazonEKSServiceRolePolicy がアタッチされています。

このポリシーにより、Amazon EKS は Amazon EKS クラスターを操作するために必要なリソースを作成および管理できるようになります。

アクセス許可の詳細

このポリシーには、Amazon EKS が以下のタスクを完了できるようにする以下の権限が含まれています。

- **eks** – 更新を開始した後、クラスターの Kubernetes バージョンを更新します。この権限は Amazon EKS では使用されませんが、下位互換性のためにポリシーに残ります。
- **ec2** – Elastic Network Interfaces およびその他のネットワークリソースとタグを操作します。これは、ノードと Kubernetes コントロールプレーン間の通信を容易にするネットワーキングを設定するために、Amazon EKS で必要です。セキュリティグループに関する情報を確認します。セキュリティグループのタグを更新します。
- **route53** – VPC をホストゾーンに関連付けます。これは、Kubernetes クラスター API サーバーのプライベートエンドポイントネットワーキングを有効にするために、Amazon EKS で必要です。
- **logs** – ログイベント これは、Amazon EKS が Kubernetes コントロールプレーンログを CloudWatch に送信できるようにするために必要です。
- **iam** – サービスにリンクされたロールを作成します。これは Amazon EKS がユーザーに代わって the section called “Amazon EKS でのサービスにリンクされた役割のアクセス許可” のサービスにリンクされたロールを作成するために必要です。

JSON ポリシードキュメントの最新バージョンを確認するには、AWSマネージドポリシーのリファレンスガイドの「AmazonEKSServicePolicy」を参照してください。

AWS マネージドポリシー: AmazonEKSServiceRolePolicy

IAM エンティティに AmazonEKSServiceRolePolicy をアタッチすることはできません。このポリシーはユーザーに代わって Amazon EKS がアクションを実行することを許可する、サービスにリンクされたロールにアタッチされます。詳細については、the section called “Amazon EKS でのサービスにリンクされた役割のアクセス許可” を参照してください。iam:CreateServiceLinkedRole アクセス許可を持つ IAM プリンシパルを使用してクラスターを作成すると、AWSServiceRoleForAmazonEKS サービスにリンクされたロールが自動的に作成され、このポリシーがそれにアタッチされます。

このポリシーにより、サービスにリンクされたロールはユーザーに代わって AWS サービスを呼び出します。

アクセス許可の詳細

このポリシーにはAmazon EKS が以下のタスクを完了できるようにする以下の権限が含まれています。

- **ec2** – クラスターの作成に必要な Elastic Network Interface、Amazon EC2 インスタンスとそのステータス、クラスターセキュリティグループ、および VPC を作成、記述します。詳細については、「the section called “セキュリティグループの要件”」を参照してください。セキュリティグループに関する情報を確認します。セキュリティグループのタグを更新します。「オンデマンドキャパシティ予約」に関する情報を参照してください。プレイスメントグループに関する情報を読み取ります。クラスターインサイトの一部として設定の問題を検出するために、ルートテーブルやネットワーク ACL を含む VPC 設定を読み取ります。
- **ec2** Auto Mode – EKS Auto Mode によって作成された EC2 インスタンスを終了します。詳細については、「EKS Auto Mode」を参照してください。
- **iam** – IAM ロールにアタッチされているすべてのマネージドポリシーを一覧表示します。これは、Amazon EKS がクラスターの作成に必要なすべてのマネージドポリシーと権限を一覧表示および検証できるようにするために必要です。
- VPC をホストゾーンに関連付ける – これは、Kubernetes クラスターの API サーバーのプライベートエンドポイントネットワーキングを有効にするために、Amazon EKS で必要です。
- ログイベント – これは、Amazon EKS が Kubernetes コントロールプレーンログを CloudWatch に送信できるようにするために必要です。
- Put メトリクス – これは Amazon EKS が Kubernetes コントロールプレーンログを CloudWatch に送信できるようにするために必要です。
- **eks** – クラスターアクセスエントリとポリシーを管理し、EKS リソースにアクセスできるユーザーと、それらのユーザーが実行できるアクションをきめ細かく制御できるようにします。これにはコンピューティング、ネットワーキング、ロードバランシング、ストレージオペレーションのために標準アクセスポリシーを関連付けることが含まれます。
- **elasticloadbalancing** – EKS クラスターに関連付けられているロードバランサーとそのコンポーネント (リスナー、ターゲットグループ、証明書) を作成、管理、削除します。ロードバランサーの属性とヘルスステータスを表示します。
- **events** – EKS クラスターに関連する EC2 イベントと AWS ヘルスイベントをモニタリングするための EventBridge ルールを作成および管理し、インフラストラクチャの変更とヘルスアラートへの自動応答を可能にします。

- **iam** – 「eks」というプレフィックスが付いた EC2 インスタンスプロファイルを管理します。EKS ノードの管理に必要なリスト化、作成、削除、ロールの関連付けなどが含まれます。ユーザーがそれぞれのワーカーノードのカスタムインスタンスプロファイルを定義できるように、インスタンスプロファイルの記述を許可します。
- **pricing** および **shield** – AWS の料金情報と Shield 保護ステータスにアクセスし、EKS リソースのコスト管理と高度なセキュリティ機能を可能にします。
- リソースのクリーンアップ – クラスターのクリーンアップオペレーション中に、ボリューム、スナップショット、起動テンプレート、ネットワークインターフェイスなどの EKS タグ付きリソースを安全に削除します。

JSON ポリシードキュメントの最新バージョンを確認するには、AWSマネージドポリシーのリファレンスガイドの「AmazonEKSServiceRolePolicy」を参照してください。

AWS マネージドポリシー: AmazonEKSVPCResourceController

AmazonEKSVPCResourceController ポリシーを IAM IDにアタッチできます。ポッドのセキュリティグループ を使用している場合はユーザーに代わってアクションを実行させるために、このポリシーを Amazon EKS クラスター IAM ロール にアタッチする必要があります。

このポリシーはノードの エラスティック・ネットワーク・インターフェイス と IP アドレスを管理するアクセス権限をクラスターロールに付与します。

アクセス許可の詳細

このポリシーには、Amazon EKS に以下のタスクを完了させるための以下の権限が含まれています。

- **ec2** – Elastic Network Interface と IP アドレスを管理し、Pod のセキュリティグループと Windows ノードをサポートします。

JSON ポリシードキュメントの最新バージョンを確認するには、AWS マネージドポリシーのリファレンスガイドの「AmazonEKSVPCResourceController」を参照してください。

AWS マネージドポリシー: AmazonEKSWorkerNodePolicy

AmazonEKSWorkerNodePolicy ポリシーを IAM エンティティにアタッチできます。このポリシーを、Amazon EKS がユーザーに代わってアクションを実行することを許可する Amazon EC2 ノードを作成するときに指定する ノード IAM ロール にアタッチしなければなりません。eksctl を使用し

てノードグループを作成すると、ノード IAM ロールが作成され、このポリシーをロールに自動的にアタッチします。

このポリシーはAmazon EKS Amazon EC2 ノードに Amazon EKS クラスターに接続するためのアクセス権限を付与します。

アクセス許可の詳細

このポリシーには Amazon EKS に以下のタスクを完了させるための以下の権限が含まれています。

- **ec2** – インスタンスボリュームとネットワーク情報を読み取ります。これは、Kubernetes ノードが Amazon EKS クラスターに参加するのに必要な Amazon EC2 リソースに関する情報を記述できるようにするために必要です。
- **eks** – オプションで、ノードのブートストラップの一部としてクラスターを記述します。
- **eks-auth:AssumeRoleForPodIdentity** – ノード上の EKS ワークロードの認証情報を取得できるようにします。これは EKS Pod Identity が正しく機能するために必要です。

JSON ポリシードキュメントの最新バージョンを確認するには、AWS マネージドポリシーのリファレンスガイドの「AmazonEKSWorkerNodePolicy Policy」を参照してください。

AWS マネージドポリシー: AmazonEKSWorkerNodeMinimalPolicy

IAM エンティティに AmazonEKSWorkerNodeMinimalPolicy をアタッチできます。このポリシーを、Amazon EKS がユーザーに代わってアクションを実行することを許可する Amazon EC2 ノードを作成するときに指定するノード IAM ロールにアタッチすることができます。

このポリシーはAmazon EKS Amazon EC2 ノードに Amazon EKS クラスターに接続するためのアクセス権限を付与します。このポリシーのアクセス許可は AmazonEKSWorkerNodePolicy に比べて少なくなります。

アクセス許可の詳細

このポリシーには Amazon EKS に以下のタスクを完了させるための以下の権限が含まれています。

- `eks-auth:AssumeRoleForPodIdentity` – ノード上の EKS ワークロードの認証情報を取得できるようにします。これは EKS Pod Identity が正しく機能するために必要です。

JSON ポリシードキュメントの最新バージョンを確認するには、AWS マネージドポリシーのリファレンスガイドの「AmazonEKSWorkerNodeMinimalPolicy」を参照してください。

AWS マネージドポリシー: AWSServiceRoleForAmazonEKSNodegroup

IAM エンティティに AWSServiceRoleForAmazonEKSNodegroup をアタッチすることはできません。このポリシーはユーザーに代わって Amazon EKS がアクションを実行することを許可する、サービスにリンクされたロールにアタッチされます。詳細については、the section called “Amazon EKS でのサービスにリンクされた役割のアクセス許可” を参照してください。

このポリシーは AWSServiceRoleForAmazonEKSNodegroup ロール権限を付与し、アカウント内の Amazon EC2 ノードグループを作成および管理できるようにします。

アクセス許可の詳細

このポリシーには Amazon EKS に以下のタスクを完了させるための以下の権限が含まれています。

- **ec2** – セキュリティグループ、タグ、キャパシティ予約、および起動テンプレートを操作します。これは Amazon EKS マネージド型ノードグループがリモートアクセス設定を有効にし、マネージド型ノードグループで使用できるキャパシティ予約を記述するために必要です。さらに、Amazon EKS マネージド型ノードグループはユーザーに代わって起動テンプレートを作成します。これは各マネージド型ノードグループをバックアップする Amazon EC2 Auto Scaling グループを設定するためです。
- **iam** – サービスにリンクされたロールを作成し、ロールを渡します。これは Amazon EKS マネージド型ノードグループが、マネージドノードグループの作成時に渡されるロールのインスタンスプロファイルを管理するために必要です。このインスタンスプロファイルはマネージド型ノードグループの一部として起動される Amazon EC2 インスタンスによって使用されます。Amazon EKS は Amazon EC2 Auto Scaling グループなどの他のサービスに対してサービスリンクされたロールを作成する必要があります。これらの権限はマネージド型ノードグループの作成に使用されます。
- **autoscaling** –セキュリティのAuto Scaling グループを使用します。これは Amazon EKS マネージド型ノードグループが、各マネージドノードグループをバックアップする Amazon EC2 Auto Scaling グループを管理するために必要です。また、ノードグループの更新中にノードが終了またはリサイクルされた場合のポッドのエビクションや、マネージド型ノードグループで構成されたウォームプールの管理などの機能のサポートにも使用されます。

JSON ポリシードキュメントの最新バージョンを確認するには、AWS マネージドポリシーのリファレンスガイドの「AWSServiceRoleForAmazonEKSNodegroup」を参照してください。

AWS マネージドポリシー: AmazonEKSDashboardServiceRolePolicy

IAM エンティティに AmazonEKSDashboardServiceRolePolicy をアタッチすることはできません。このポリシーはユーザーに代わって Amazon EKS がアクションを実行することを許可する、サービスにリンクされたロールにアタッチされます。詳細については、「the section called “Amazon EKS でのサービスにリンクされた役割のアクセス許可”」を参照してください。

このポリシーは、AWS Organizations の構造とアカウントに関する情報を表示するためのアクセス許可を AWSServiceRoleForAmazonEKSDashboard ロールに付与します。

アクセス許可の詳細

このポリシーには、以下のタスクを完了させるためにアクセスできる次の権限が含まれています。

- **organizations** – AWS Organizations の構造とアカウントに関する情報を表示します。これには、組織内のアカウントを一覧表示する、組織単位とルートを表示する、委任された管理者を一覧表示する、組織にアクセスできるサービスを表示する、および組織とアカウントに関する詳細情報を取得する権限が含まれます。

JSON ポリシードキュメントの最新バージョンを確認するには、AWSマネージドポリシーのリファレンスガイドの「AmazonEKSDashboardServiceRolePolicy」を参照してください。

AWS マネージドポリシー: AmazonEBSCSIDriverPolicy

> **Important**
>
> より厳格なアクセス許可スコープを提供する、2 つの新しい AWS マネージド IAM ポリシーが利用可能になりました。タグベースのスコープ設定には AmazonEBSCSIDriverPolicyV2、クラスタースコープの分離にはAmazonEBSCSIDriverEKSClusterScopedPolicy が使用できます。移行にご興味がある場合は、「EBS CSI ドライバーポリシーの移行」を参照してください。

AmazonEBSCSIDriverPolicy ポリシーは Amazon EBS コンテナストレージインターフェース (CSI) ドライバーが、ユーザーに代わってボリュームを作成、変更、コピー、アタッチ、デタッチ、削除できるようにします。これには既存のボリュームのタグの変更や、EBS ボリュームでの高速スナップショット復元 (FSR) の有効化が含まれます。また、EBS CSI ドライバーに、スナップショットを作成、ロック、復元、削除したり、インスタンス、ボリューム、スナップショットを一覧表示する権限も付与します。

JSON ポリシードキュメントの最新バージョンを確認するには、AWS マネージドポリシーのリファレンスガイドの「AmazonEBSCSIDriverServiceRolePolicy」を参照してください。

AWS マネージドポリシー: AmazonEBSCSIDriverPolicyV2

この AmazonEBSCSIDriverPolicyV2 ポリシーは、AmazonEBSCSIDriverPolicy のより制限の厳しい代替手段です。Amazon EBS CSI ドライバーは、true に設定されたキー ebs.csi.aws.com/cluster でタグ付けされた EBS ボリュームとスナップショットのみを管理するように制限されます。ツリー内 Kubernetes ボリュームプラグイン (CSI 移行ボリューム) によってプロビジョニングされたボリュームは、 kubernetes.io/created-for/pvc/name リソースタグでもサポートされます。

AmazonEBSCSIDriverPolicy から移行する場合は、EBS CSI ドライバーポリシーの移行 を参照してください。

JSON ポリシードキュメントの最新バージョンを確認するには、AWS マネージドポリシーのリファレンスガイドの「AmazonEBSCSIDriverPolicyV2」を参照してください。

AWS マネージドポリシー: AmazonEBSCSIDriverEKSClusterScopedPolicy

この AmazonEBSCSIDriverEKSClusterScopedPolicy ポリシーは、特定の EKS クラスターに属する EBS ボリュームとスナップショットのみを管理するように Amazon EBS CSI ドライバーを制限します。リソースタグは IAM プリンシパルの eks-cluster-name タグ ebs.csi.aws.com/cluster-name が一致する必要があり、複数のクラスターが同じ AWS アカウントを共有する場合のクラスター間のアクセスを防止します。インスタンスのアタッチおよびデタッチ操作は、eks:cluster-name タグ (マネージドノードグループの EKS によって自動的に設定)、eks:eks-cluster-name タグ (オープンソースの Karpenter や Amazon EKS Auto Mode を含む EKS コントローラーのタグ付け標準に従うコントローラーによって設定)、または ebs.csi.aws.com/cluster-name タグ (手動でタグ付けされたインスタンスの場合) でタグ付けされたインスタンスに制限されます。

AmazonEBSCSIDriverPolicy から移行する場合は、「EBS CSI ドライバーポリシーの移行」を参照してください。

JSON ポリシードキュメントの最新バージョンを確認するには、AWS マネージドポリシーのリファレンスガイドの「AmazonEBSCSIDriverEKSClusterScopedPolicy」を参照してください。

AWS マネージドポリシー: AmazonEFSCSIDriverPolicy

AmazonEFSCSIDriverPolicy ポリシーでは Amazon EFS コンテナストレージインターフェイス (CSI) がユーザーに代わってアクセスポイントを作成および削除できるようにすることができます。

また、Amazon EFS CSI ドライバーに、アクセスポイント、ファイルシステム、マウントターゲット、Amazon EC2 アベイラビリティゾーンを一覧表示するアクセス許可を付与することもできます。

JSON ポリシードキュメントの最新バージョンを確認するには、AWS マネージドポリシーのリファレンスガイドの「AmazonEFSCSIDriverPolicy」を参照してください。

AWS マネージドポリシー: AmazonS3FilesCSIDriverPolicy

`AmazonS3FilesCSIDriverPolicy` ポリシーでは Amazon EFS コンテナストレージインターフェイス (CSI) がユーザーに代わって Amazon S3 Files アクセスポイントを作成および削除できるようにすることができます。また、Amazon EFS CSI ドライバーに、Amazon S3 Files のアクセスポイントとファイルシステムを一覧表示するアクセス許可を付与します。

JSON ポリシードキュメントの最新バージョンを確認するには、AWS マネージドポリシーのリファレンスガイドの「AmazonS3FilesCSIDriverPolicy」を参照してください。

AWS マネージドポリシー: AmazonEKSLocalOutpostClusterPolicy

このポリシーを IAM エンティティにアタッチできます。ローカルクラスターを作成する前に、このポリシーを クラスターロール にアタッチする必要があります。Amazon EKS によって管理される Kubernetes クラスターは、ユーザーに代わって他の AWS のサービスを呼び出します。これにより、サービスで使用するリソースを管理します。

`AmazonEKSLocalOutpostClusterPolicy` には以下の権限が含まれています。

- **`ec2`** 読み取りアクション – コントロールプレーンインスタンスがアベイラビリティーゾーン、ルートテーブル、インスタンス、ネットワークインターフェイスのプロパティを記述できるようにします。Amazon EC2 インスタンスがコントロールプレーンインスタンスとして正常にクラスターに参加するために必要なアクセス許可です。
- **`ssm`** – Amazon EC2 Systems Manager がコントロールプレーンインスタンスに接続できるようにします。これはアカウントのローカルクラスターと通信して管理するために、Amazon EKS によって使用されます。
- **`logs`** – インスタンスが Amazon CloudWatch にログをプッシュできるようにします。
- **`secretsmanager`** – インスタンスがコントロールプレーンインスタンスのブートストラップデータを AWS 秘密マネジャー から安全に取得および削除できるようにします。
- **`ecr`** – コントロールプレーンインスタンス上で動作している Pod とコンテナが、Amazon Elastic Container Registry に格納されているコンテナイメージをプルできるようにします。

JSON ポリシードキュメントの最新バージョンを確認するには、AWS マネージドポリシーのリファレンスガイドの「AmazonEKSLocalOutpostClusterPolicy」を参照してください。

AWS マネージドポリシー: AmazonEKSLocalOutpostServiceRolePolicy

このポリシーを IAM エンティティにアタッチすることはできません。`iam:CreateServiceLinkedRole` アクセス許可のある IAM プリンシパルを使用してクラスターを作成する場合、Amazon EKS は AWSServiceRoleForAmazonEKSLocalOutpost のサービスにリンクされたロールを自動的に作成し、このポリシーをアタッチします。このポリシーはサービスにリンクされたロールはローカルクラスターの代わりに AWS サービスを呼び出すことを許可します。

`AmazonEKSLocalOutpostServiceRolePolicy` には以下の権限が含まれています。

- **`ec2`** – Amazon EKS がセキュリティ、ネットワーク、その他のリソースと連携して、アカウント内のコントロールプレーンインスタンスを正常に起動および管理できるようにします。
- **`ssm`、`ssmmessages`** – Amazon EC2 Systems Manager がコントロールプレーンインスタンスに接続できるようにします。アカウントのローカルクラスターと通信および管理するため、Amazon EKS によって使用されます。
- **`iam`** – Amazon EKS がコントロールプレーンインスタンスに関連付けられたインスタンスプロファイルを管理できるようにします。
- **`secretsmanager`** – Amazon EKS がコントロールプレーンインスタンスのブートストラップデータを AWS 秘密マネジャー に格納できるようにします。これにより、インスタンスのブートストラップ中に安全に参照できるようになります。
- **`outposts`** – Amazon EKS がお客様のアカウントから Outpost 情報を取得して、Outpost でローカルクラスターを正常に起動できるようにします。

JSON ポリシードキュメントの最新バージョンを確認するには、AWS マネージドポリシーのリファレンスガイドの「AmazonEKSLocalOutpostServiceRolePolicy」を参照してください。

Amazon EKS による AWS マネージドポリシーの更新

Amazon EKS の AWS マネージドポリシーの更新に関する詳細を、このサービスがこれらの変更の追跡を開始した以降の分について表示します。

この特定のドキュメントページへのすべてのソースファイルの変更通知を受け取るには、RSS リーダーを使用して次の URL をサブスクライブできます。

```
https://github.com/awsdocs/amazon-eks-user-guide/commits/mainline/latest/ug/security/iam-reference/security-iam-awsmanpol.adoc.atom
```

変更	説明	日付
the section called “AWS マネージドポリシー: AmazonEKSLoadBalancingPolicy” へのアクセス許可を更新	AmazonEKSLoadBalancingPolicy のリソースタグ付け許可リストに gateway.eks.amazonaws.com.alb/stack 、gateway.eks.amazonaws.com.alb/resource 、gateway.eks.amazonaws.com.nlb/stack 、gateway.eks.amazonaws.com.nlb/resource のタグキーを追加しました。これにより、Amazon EKS Auto Mode コントローラーが Kubernetes Gateway API (ALB および NLB) ワークロードの AWS リソースを管理できるようになります。	2026 年 9 月 2 日
the section called “AWS マネージドポリシー: AmazonEKSLoadBalancingPolicy” にアクセス許可が追加されました。	AmazonEKSLoadBalancingPolicy に wafv2:ListWebACLs アクセス許可を追加しました。これにより、Amazon EKS Auto Mode コントローラーが Application Load Balancer の関連付けを調整するときに AWS WAF Web ACL を一覧表示できるようになります。	2026 年 9 月 2 日

変更	説明	日付
AmazonEKSServiceRolePolicy アクセス許可が追加されました	サービスにリンクされたロールがクラスターアクセスエントリを関連付けおよび関連付け解除できるアクセスポリシーのリストに AmazonEKSPodIdentityPolicy が追加されました。この追加により、Amazon EKS は EKS クラスターで実行されているワークロードの Pod Identity 認証情報を管理できます。	2026 年 6 月 26 日
the section called “AWS マネージドポリシー: AmazonEKSLoadBalancingPolicy” へのアクセス許可を更新	AmazonEKSLoadBalancingPolicy に elasticloadbalancing:SetRulePriorities 、ec2:DescribeIpamPools アクセス許可が追加されました。これにより、Amazon EKS Auto Mode コントローラーが ALB の指定されたリスナールールの優先順位を設定し、それぞれ IPAM プールを describe できます。	2026 年 6 月 15 日

変更	説明	日付
AmazonEBSCSIDriverEKSClusterScopedPolicy にアクセス許可を追加しました。	eks:eks-cluster-name タグでタグ付けされたインスタンスで ec2:AttachVolume および ec2:DetachVolume が承認され、オープンソースの Karpenter や Amazon EKS Auto Mode など、EKS コントローラーのタグ付け標準に従うコントローラーとの互換性が追加されました。	2026 年 5 月 28 日
the section called “AWS マネージドポリシー: AmazonEKSComputePolicy” にアクセス許可が追加されました。	ec2:DescribeCapacityReservations アクションを含めるためのリソースアクセス許可が追加されました。これにより、Amazon EKS Auto Mode は、アカウントの中断可能なキャパシティ予約からインスタンスを起動できるようになります。	2026 年 5 月 18 日
the section called “AWS マネージドポリシー: AmazonEKSBlockStoragePolicyV2” について説明しました。	AWS は AmazonEKSBlockStoragePolicyV2 を導入しました。	2026 年 5 月 12 日
AmazonEBSCSIDriverPolicy にアクセス許可を追加しました。	EBS CSI ドライバーがボリューム制限/カード情報を実行時に動的に取得できるように、新しい API アクション ec2:DescribeInstanceTypes が追加されました。	2026 年 5 月 12 日

変更	説明	日付
に アクセス許可を追加しましたthe section called “AWS マネージドポリシー: AmazonEKSLoadBalancingPolicy”	Amazon EKS Auto Mode コントローラーがロードバランサーを記述できるように、AmazonEKSLoadBalancingPolicy に elasticloadbalancing:DescribeLoadBalancers アクセス許可が追加されました。	2026 年 4 月 27 日
に アクセス許可を追加しましたthe section called “AWS マネージドポリシー: AmazonEKSLoadBalancingPolicy”	Amazon EKS Auto Mode コントローラーで LoadBalancer キャパシティユニットを記述できるように、AmazonEKSLoadBalancingPolicy に elasticloadbalancing:DescribeCapacityReservation アクセス許可が追加されました	2026 年 4 月 20 日
に アクセス許可を追加しましたthe section called “AWS マネージドポリシー: AmazonEKSServiceRolePolicy”	AmazonEKSServiceRolePolicy に ec2:DescribeInstanceStatus および iam:ListInstanceProfiles アクセス許可が追加されました。これらのアクセス許可により、Amazon EKS Auto Mode でマネージドインスタンスの EC2 ヘルスイベントを検出し、マネージドインスタンスプロファイルをクリーンアップできます	2026 年 4 月 20 日

変更	説明	日付
the section called “AWS マネージドポリシー: AmazonEKSComputePolicy” にアクセス許可が追加されました。	`ec2:RunInstances` および `ec2:CreateFleet` アクションのリソースのアクセス許可が更新され、プレイスメントグループ `arn:aws:ec2:*:*:placement-group/*` が含まれるようになりました。これにより、Amazon EKS Auto Mode でアカウントの EC2 プレイスメントグループにインスタンスを起動できます。	2026 年 4 月 17 日

変更	説明	日付
AmazonEBSCSIDriverPolicyV2 および AmazonEBSCSIDriverEKSClusterScopedPolicy を導入しました。	AmazonEBSCSIDriverPolicyV2 および AmazonEBSCSIDriverEKSClusterScopedPolicy のより制限の厳しい代替手段として AmazonEBSCSIDriverPolicy が導入されました。AmazonEBSCSIDriverPolicyV2 は、EBS CSI ドライバーを true に設定された ebs.csi.aws.com/cluster でタグ付けされたボリュームとスナップショットにスコープします。AmazonEBSCSIDriverEKSClusterScopedPolicy は、ebs.csi.aws.com/cluster-name タグを使用して、特定の EKS クラスターに属するボリュームとスナップショットにドライバーをスコープします。	2026 年 4 月 16 日
AmazonEKSServiceRolePolicy にアクセス許可が追加されました	Amazon EKS Auto Mode でアカウントの EC2 プレイスメントグループにインスタンスを起動できるように、ec2:DescribePlacementGroups アクセス許可が追加されました。	2026 年 4 月 15 日

変更	説明	日付
the section called “AWS マネージドポリシー: AmazonEKSLoadBalancingPolicy” へのアクセス許可を更新	AmazonEKSLoadBalancingPolicy における shield:CreateProtection 、shield:DeleteProtection のアクセス許可を更新しました。これにより、Amazon EKS Auto Mode コントローラーはシールド保護を追加できます。	2026 年 4 月 14 日
the section called “AWS マネージドポリシー: AmazonEKSLoadBalancingPolicy” にアクセス許可を追加しました。	AmazonEKSLoadBalancingPolicy の elasticloadbalancing:RegisterTargets アクセス許可を更新して、マルチクラスターおよびカスタムターゲットグループ割り当てのサポートを追加	2026 年 3 月 20 日
the section called “AWS マネージドポリシー: AmazonEKSLoadBalancingPolicy” にアクセス許可を追加しました。	ALB の IPAM プールの設定のサポートを追加するための elasticloadbalancing:ModifyIpPools アクセス許可が AmazonEKSLoadBalancingPolicy に追加されました	2026 年 3 月 20 日

変更	説明	日付
the section called “AWS マネージドポリシー: AmazonEKSNetworkingPolicy” にアクセス許可が追加されました。	AmazonEKSNetworkingPolicy に ec2:ModifyNetworkInterfaceAttribute アクセス許可を追加しました。これにより、Amazon EKS Auto Mode コントローラーは EC2 インスタンスに関連するネットワークインターフェイス属性を変更できます	2026 年 2 月 3 日
the section called “AWS マネージドポリシー: AWSServiceRoleForAmazonEKSNodegroup” にアクセス許可を追加しました。	AWSServiceRoleForAmazonEKSNodegroup に autoscaling:PutWarmPool 、autoscaling:DeleteWarmPool および autoscaling:DescribeWarmPool のアクセス許可を追加しました。これにより、Amazon EKS Managed Nodegroup はノードグループのライフサイクル全体で基盤となる ASG ウォームプールリソースを管理できます。	2026 年 2 月 17 日

変更	説明	日付
the section called “AWS マネージドポリシー: AmazonEKSServiceRolePolicy” にアクセス許可が追加されました。	AmazonEKSServiceRolePolicy 内の、iam:GetInstanceProfile アクセス許可のターゲットインスタンスプロファイル名に「eks」プレフィックスを付ける要件を削除しました。これにより、Amazon EKS Auto Mode は「eks」という命名プレフィックスを必要とせずに NodeClasses のカスタムインスタンスプロファイルを検証して使用できます。	2026 年 2 月 2 日
AmazonEBSCSIDriverPolicy にアクセス許可を追加しました。	EBS CSI ドライバーが EBS スナップショットを直接ロックできるように、ec2:LockSnapshot アクセス許可を追加しました。	2026 年 1 月 15 日
the section called “AWS マネージドポリシー: AmazonEKSMCPReadOnlyAccess” について説明しました。	Amazon EKS は、オブザーバビリティとトラブルシューティングのために、Amazon EKS MCP サーバーで読み取り専用ツールを有効にする新しいマネージドポリシー AmazonEKSMCPReadOnlyAccess を導入しました。	2025 年 11 月 21 日
AmazonEBSCSIDriverPolicy にアクセス許可を追加しました。	EBS CSI ドライバーが EBS ボリュームを直接コピーできるようにする ec2:CopyVolumes アクセス許可を追加しました。	2025 年 11 月 17 日

変更	説明	日付
the section called “AWS マネージドポリシー: AmazonEKSServiceRolePolicy” にアクセス許可が追加されました。	AmazonEKSServiceRolePolicy に ec2:DescribeRouteTables および ec2:DescribeNetworkAcls アクセス許可を追加しました。これにより、Amazon EKS はクラスターインサイトの一部として、ハイブリッドノードの VPC ルートテーブルとネットワーク ACL の設定問題を検出できます。	2025 年 10 月 22 日
AWSServiceRoleForAmazonEKSConnector にアクセス許可が追加されました	AmazonEKSConnectorServiceRolePolicy に ssmmessages:OpenDataChannel アクセス許可を追加しました。	2025 年 10 月 15 日
に アクセス許可を追加しましたthe section called “AWS マネージドポリシー: AmazonEKSServiceRolePolicy”	このロールは、新しいアクセスポリシー AmazonEKSEventPolicy をアタッチできます。ec2:DeleteLaunchTemplate と ec2:TerminateInstances の制限されたアクセス許可。	2025 年 8 月 26 日
the section called “AWS マネージドポリシー: AmazonEKSLocalOutpostServiceRolePolicy” にアクセス許可が追加されました	AmazonEKSLocalOutpostServiceRolePolicy に ssmmessages:OpenDataChannel アクセス許可が追加されました。	2025 年 6 月 26 日

変更	説明	日付
the section called “AWS マネージドポリシー: AmazonEKSComputePolicy” にアクセス許可が追加されました。	キャパシティ予約 `arn:aws:ec2:*:*:capacity-reservation/*` を含めるため、`ec2:RunInstances` および `ec2:CreateFleet` アクションのリソースアクセス許可が更新されました。これにより、Amazon EKS Auto Mode は、アカウントの EC2 オンデマンドキャパシティ予約を使用してインスタンスを起動できるようになります。Amazon EKS Auto Mode がユーザーに代わって EC2 スポットサービスにリンクされたロール `AWSServiceRoleForEC2Spot` を作成できるように、`iam:CreateServiceLinkedRole` が追加されました。	2025 年 6 月 20 日
AmazonEKSServiceRolePolicy にアクセス許可が追加されました	アカウントの EC2 オンデマンドキャパシティ予約を使用して Amazon EKS Auto Mode がインスタンスを起動できるように、`ec2:DescribeCapacityReservations` アクセス許可が追加されました。	2025 年 6 月 20 日

変更	説明	日付
the section called “AWS マネージドポリシー: AmazonEKSDashboardConsoleReadOnly” について説明しました。	新しい AmazonEKSDashboardConsoleReadOnly ポリシーを導入しました。	2025 年 6 月 19 日
the section called “AWS マネージドポリシー: AmazonEKSDashboardServiceRolePolicy” について説明しました。	新しい AmazonEKSDashboardServiceRolePolicy ポリシーを導入しました。	2025 年 5 月 21 日
AmazonEKSClusterPolicy アクセス許可を追加しました。	VPC CNI が予期せず終了した場合に残された Elastic Network Interface を Amazon EKS が削除できるように、ec2:DeleteNetworkInterfaces アクセス許可が追加されました。	2025 年 4 月 16 日
AmazonEKSServiceRolePolicy アクセス許可が追加されました	EKS 1.33 バージョンリリースの一環として、EKS AI/ML の顧客が EFA と互換性のあるセキュリティグループ Egress (アウトバウンド) ルールをデフォルトの EKS クラスター SG に追加できるように、ec2:RevokeSecurityGroupEgress および ec2:AuthorizeSecurityGroupEgress アクセス許可が追加されました。	2025 年 4 月 14 日

変更	説明	日付
AmazonEKSServiceRolePolicy アクセス許可を追加しました。	EKS Auto Mode によって作成された EC2 インスタンスを終了できるようにアクセス許可を追加しました。	2025 年 2 月 28 日
AmazonEBSCSIDriverPolicy にアクセス許可を追加しました。	EBS CSI ドライバーがすべてのスナップショットを復元することを許可する新しいステートメントを追加しました。これは既存のポリシーでは以前は許可されていましたが、`CreateVolume` の IAM の処理が変更されたため、新しい明示的なステートメントが必要になりました。 EBS CSI ドライバーが既存のボリュームのタグを変更する機能を追加しました。EBS CSI ドライバーは、Kubernetes VolumeAttributesClasses のパラメータを介して既存のボリュームのタグを変更できます。 EBS ボリューム上で高速スナップショット復元 (FSR) を有効にするために EBS CSI ドライバーの機能を追加しました。EBS CSI ドライバーは、Kubernetes ストレージクラスのパラメータを介して、新しいボリュームで FSR を有効にできます。	2025 年 1 月 13 日

変更	説明	日付
the section called “AWS マネージドポリシー: AmazonEKSLoadBalancingPolicy” にアクセス許可を追加しました。	AmazonEKSLoadBalancingPolicy を更新して、ネットワークおよび IP アドレス リソースの一覧表示と説明ができるようになりました。	2024 年 12 月 26 日
the section called “AWS マネージドポリシー: AWSServiceRoleForAmazonEKSNodegroup” にアクセス許可を追加しました。	AWSServiceRoleForAmazonEKSNodegroup が中国リージョンとの互換性のために更新されました。	2024 年 11 月 22 日
the section called “AWS マネージドポリシー: AmazonEKSLocalOutpostClusterPolicy” にアクセス許可を追加しました。	各ノードが存在するアベイラビリティーゾーンを、クラスターコントロールプレーンの AWS Cloud Controller Manager が識別できるように、AmazonEKSLocalOutpostClusterPolicy に ec2:DescribeAvailabilityZones 許可を追加しました。	2024 年 11 月 21 日
the section called “AWS マネージドポリシー: AWSServiceRoleForAmazonEKSNodegroup” にアクセス許可を追加しました。	Amazon EKS マネージドノードグループによって作成されたインスタンスのために ec2:RebootInstances を許可するように AWSServiceRoleForAmazonEKSNodegroup ポリシーを更新しました。Amazon EC2 リソースについての ec2:CreateTags 許可を制限しました。	2024 年 11 月 20 日

変更	説明	日付
the section called “AWS マネージドポリシー: AmazonEKSServiceRolePolicy” にアクセス許可を追加しました。	EKS が AWS マネージドポリシー AmazonEKSServiceRolePolicy を更新しました。EKS アクセスポリシー、ロードバランサーの管理、クラスターリソースの自動クリーンアップの許可を追加しました。	2024 年 11 月 16 日
the section called “AWS マネージドポリシー: AmazonEKSComputePolicy” について説明しました。	EKS が AWS マネージドポリシー AmazonEKSComputePolicy を更新しました。iam:AddRoleToInstanceProfile アクションのリソース許可を更新しました。	2024 年 11 月 7 日
the section called “AWS マネージドポリシー: AmazonEKSComputePolicy” について説明しました。	AWS は AmazonEKSComputePolicy を導入しました。	2024 年 11 月 1 日
AmazonEKSClusterPolicy にアクセス許可を追加しました。	Amazon EKS がトポロジ情報をラベルとしてノードにアタッチできるようにする ec2:DescribeInstanceTopology 許可を追加しました。	2024 年 11 月 1 日
AmazonEKSBlockStoragePolicy を導入しました。	AWS は AmazonEKSBlockStoragePolicy を導入しました。	2024 年 10 月 30 日

変更	説明	日付
the section called “AWS マネージドポリシー: AmazonEKSLoadBalancingPolicy” について説明しました。	AWS は AmazonEKSLoadBalancingPolicy を導入しました。	2024 年 10 月 30 日
AmazonEKSServiceRolePolicy に許可を追加しました。	Amazon EKS が Amazon CloudWatch にメトリクスを発行できるようにする cloudwatch:PutMetricData 許可を追加しました。	2024 年 10 月 29 日
the section called “AWS マネージドポリシー: AmazonEKSNetworkingPolicy” について説明しました。	AWS は AmazonEKSNetworkingPolicy を導入しました。	2024 年 10 月 28 日
AmazonEKSServicePolicy および AmazonEKSServiceRolePolicy にアクセス許可を追加しました。	EKS がセキュリティグループ情報を読み取り、関連するタグを更新できるように、ec2:GetSecurityGroupsForVpc および関連するタグの許可を追加しました。	2024 年 10 月 10 日
AmazonEKSWorkerNodeMinimalPolicy を導入しました。	AWS は AmazonEKSWorkerNodeMinimalPolicy を導入しました。	2024 年 10 月 3 日

変更	説明	日付
AWSServiceRoleForAmazonEKSNodegroup にアクセス許可を追加しました。	Amazon EKS が管理するAuto Scaling グループで Amazon EKS が AZRebalance を一時停止および再開できるようにする autoscaling:ResumeProcesses および autoscaling:SuspendProcesses アクセス許可が追加されました。	2024 年 8 月 21 日
AWSServiceRoleForAmazonEKSNodegroup にアクセス許可を追加しました。	Amazon EKS がユーザーのアカウントでキャパシティ予約を記述できるようにする ec2:DescribeCapacityReservations アクセス許可を追加しました。CAPACITY_BLOCK ノードグループでスケジュールされたスケーリングの設定を有効にする autoscaling:PutScheduledUpdateGroupAction アクセス許可を追加しました。	2024 年 6 月 27 日

変更	説明	日付
AmazonEKS_CNI_Policy – 既存のポリシーへの更新	Amazon EKS は、Amazon VPC CNI Plugin for Kubernetes で Amazon VPC サブネット内の空き IP アドレスの量を確認できるようにする新しい `ec2:DescribeSubnets` 許可を追加しました。VPC CNI は各サブネットの空き IP アドレスを使用して、エラスティック・ネットワーク・インターフェイス の作成時に使用する空き IP アドレスが最も多いサブネットを選択できます。	2024 年 3 月 4 日
AmazonEKSWorkerNodePolicy – 既存のポリシーへの更新	Amazon EKS は EKS Pod Identity を許可する新しいアクセス権限を追加しました。Amazon EKS Pod Identity エージェントはノードロールを使用します。	2023 年 11 月 26 日
AmazonEFSCSIDriverPolicy を導入しました。	AWS は `AmazonEFSCSIDriverPolicy` を導入しました。	2023 年 7 月 26 日
AmazonEKSClusterPolicy にアクセス許可を追加しました。	ロードバランサーを作成しながら、サブネットの自動検出中に Amazon EKS が AZ の詳細を取得できるようにする `ec2:DescribeAvailabilityZones` 許可が追加されました。	2023 年 2 月 7 日

変更	説明	日付
AmazonEBSCSIDriverPolicy のポリシー条件が更新されました。	StringLike キーフィールドにワイルドカード文字を含む無効なポリシー条件を削除しました。また ec2:DeleteVolume に新しい条件 ec2:ResourceTag/kubernetes.io/created-for/pvc/name: "*" も追加され、ツリー内プラグインで作成されたボリュームを EBS CSI ドライバーが削除できるようになりました。	2022 年 11 月 17 日
AmazonEKSLocalOutpostServiceRolePolicy にアクセス許可を追加しました。	前提条件の検証とマネージドライフサイクルの管理が向上するように、ec2:DescribeVPCAttribute 、ec2:GetConsoleOutput 、ec2:DescribeSecret を追加しました。また、Outposts のコントロールプレーン Amazon EC2 インスタンスの配置制御をサポートするため、ec2:DescribePlacementGroups 、"arn:aws:ec2:*:*:placement-group/*" 、ec2:RunInstances が追加しました。	2022 年 10 月 24 日

変更	説明	日付
AmazonEKSLocalOutpostClusterPolicy の Amazon Elastic Container Registry のアクセス権限を更新します。	ecr:GetDownloadUrlForLayer アクションを、全リソースセクションからスコープされたセクションに移動しました。 arn:aws:ecr:*:*:repository/eks/ リソースを追加しました。 arn:aws:ecr: リソースを削除しました。このリソースは追加された arn:aws:ecr:*:*:repository/eks/* リソースでカバーします。	2022 年 10 月 20 日
AmazonEKSLocalOutpostClusterPolicy にアクセス許可を追加しました。	クラスターコントロールプレーンインスタンスがいくつかの kubelet 引数を更新できるように、 arn:aws:ecr:*:*:repository/kubelet-config-updater Amazon Elastic Container Registry リポジトリを追加しました。	2022 年 8 月 31 日
AmazonEKSLocalOutpostClusterPolicy を導入しました。	AWS は AmazonEKSLocalOutpostClusterPolicy を導入しました。	2022 年 8 月 24 日
AmazonEKSLocalOutpostServiceRolePolicy を導入しました。	AWS は AmazonEKSLocalOutpostServiceRolePolicy を導入しました。	2022 年 8 月 23 日

変更	説明	日付
AmazonEBSCSIDriverPolicy を導入しました。	AWS は AmazonEBSCSIDriverPolicy を導入しました。	2022 年 4 月 4 日
AmazonEKSWorkerNodePolicy にアクセス許可を追加しました。	インスタンスレベルのプロパティを自動検出できる Amazon EKS に最適化された AMI を有効化する ec2:DescribeInstanceTypes を追加しました。	2022 年 3 月 21 日
AWSServiceRoleForAmazonEKSNodegroup にアクセス許可を追加しました。	Amazon EKS がメトリクスの収集を有効にすることを許可する autoscaling:EnableMetricsCollection アクセス許可を追加しました。	2021 年 12 月 13 日
AmazonEKSClusterPolicy にアクセス許可を追加しました。	ec2:DescribeAccountAttributes 、ec2:DescribeAddresses 、ec2:DescribeInternetGateways の権限を追加し、Amazon EKS が Network Load Balancer のサービスにリンクされたロールを作成できるようになりました。	2021 年 6 月 17 日
Amazon EKS が変更の追跡を開始しました。	Amazon EKS が AWS マネージドポリシーの変更の追跡を開始しました。	2021 年 6 月 17 日

IAM のトラブルシューティング

このトピックでは、IAM での Amazon EKS の使用中に表示される一般的なエラーとその回避方法について説明します。

AccessDeniedException

AWS API オペレーションの呼び出し時に `AccessDeniedException` を受け取った場合、使用している IAM プリンシパルの認証情報には、その呼び出しに必要な許可がありません。

```
An error occurred (AccessDeniedException) when calling the DescribeCluster operation:
User: arn:aws:iam::111122223333:user/user_name is not authorized to perform:
eks:DescribeCluster on resource: arn:aws:eks:region:111122223333:cluster/my-cluster
```

前記のメッセージの例では、ユーザーには Amazon EKS `DescribeCluster` API オペレーションを呼び出す許可がありません。IAM プリンシパルに Amazon EKS 管理者の許可を付与するには、「the section called “アイデンティティベースのポリシー”」を参照してください。

IAM の一般的な情報については、IAM ユーザーガイドの「ポリシーを使用したアクセスの制御」を参照してください。

[Compute] (コンピューティング) タブに [Nodes] (ノード) が表示されず、また [Resources] (リソース) タブにも何も表示されず、AWS マネジメントコンソール でエラーが表示される

`Your current user or role does not have access to Kubernetes objects on this EKS cluster`というコンソールのエラーメッセージが表示されることがあります。AWS マネジメントコンソール を使用している IAM プリンシパルユーザーに必要な許可があることを確認してください。詳細については、「the section called “必要なアクセス許可”」を参照してください。

aws-auth **`ConfigMap`** がクラスターへのアクセスを許可しない

AWS IAM Authenticator は、`ConfigMap` で使用されるロール ARN でパスを許可しません。したがって、`rolearn` を指定する前に、パスを削除してください。例えば、 `arn:aws:iam::`*`111122223333`*`:role/`*`team`*`/`*`developers`*`/`*`eks-admin`* を `arn:aws:iam::`*`111122223333`*`:role/`*`eks-admin`* に変更します。

iam:PassRole を実行する権限がありません

`iam:PassRole` アクションを実行する権限がないというエラーが表示された場合は、ポリシーを更新して Amazon EKS にロールを渡せるようにする必要があります。

一部の AWS サービスでは、新しいサービスロールまたはサービスリンクロールを作成せずに、既存のロールをサービスに渡すことができます。そのためには、サービスにロールを渡すアクセス許可が必要です。

次の例のエラーは、`marymajor` という IAM ユーザーがコンソールを使用して Amazon EKS でアクションを実行しようとする場合に発生します。ただし、このアクションをサービスが実行するには、サービスロールから付与されたアクセス許可が必要です。Mary には、ロールをサービスに渡すアクセス許可がありません。

```
User: {arn-aws}iam::123456789012:user/marymajor is not authorized to perform:
 iam:PassRole
```

この場合、メアリーのポリシーを更新してメアリーに `iam:PassRole` アクションの実行を許可する必要があります。

サポートが必要な場合は、AWS 管理者に問い合わせてください。サインイン認証情報を提供した担当者が管理者です。

AWS アカウント以外のユーザーに Amazon EKS リソースへのアクセスを許可したい

他のアカウントのユーザーや組織外のユーザーが、リソースへのアクセスに使用できるロールを作成できます。ロールの引き受けを委託するユーザーを指定できます。リソースベースのポリシーまたはアクセスコントロールリスト (ACL) をサポートするサービスの場合、それらのポリシーを使用して、リソースへのアクセスを付与できます。

詳細については、以下を参照してください。

- Amazon EKS がこれらの機能をサポートしているかどうかについては、「the section called “IAM および Amazon EKS”」を参照してください。
- 所有している AWS アカウント間でリソースへのアクセスを付与する方法については、IAM ユーザーガイド の「所有している別の AWS アカウントへのアクセスを IAM ユーザーに許可」を参照してください。
- リソースへのアクセスをサードパーティーの AWS アカウントに提供する方法については、IAM ユーザーガイドの「サードパーティーが所有する AWS アカウントへのアクセスの提供」を参照してください。

- ID フェデレーションを介してアクセスを提供する方法については、IAM ユーザーガイド の 外部で認証されたユーザー (ID フェデレーション) へのアクセスの許可 を参照してください。
- クロスアカウントアクセスにおけるロールとリソースベースのポリシーの使用方法の違いについては、IAM ユーザーガイド の IAM でのクロスアカウントのリソースへのアクセス を参照してください。

ポッドコンテナは次のエラーを受け取ります: **An error occurred (SignatureDoesNotMatch) when calling the GetCallerIdentity operation: Credential should be scoped to a valid region**

アプリケーションが AWS STS グローバルエンドポイント (`https://sts.amazonaws.com`) に明示的にリクエストを行っていて、Kubernetes サービスアカウントがリージョンエンドポイントを使用するように設定されている場合、コンテナはこのエラーを受け取ります。次のいずれかのオプションを使用して問題を解決できます。

- アプリケーションコードを更新して、AWS STS グローバルエンドポイントへの明示的な呼び出しを削除します。
- アプリケーションコードを更新して、`https://sts.us-west-2.amazonaws.com` などのリージョンエンドポイントへの明示的な呼び出しを行います。AWS リージョン内のサービスに障害が発生した場合に別の AWS リージョンを選択するよう、アプリケーションに冗長性が組み込まれている必要があります。詳細については、IAM ユーザーガイドの「AWS リージョンでの AWS STS の管理」を参照してください。
- グローバルエンドポイントを使用するようにサービスアカウントを設定します。クラスターでは、デフォルトでリージョンエンドポイントが使用されます。詳細については、「the section called “STS エンドポイント”」を参照してください。

Amazon EKS クラスター の IAM ロール

Amazon EKS クラスター IAM ロールはクラスターごとに必要です。Amazon EKS によって管理される Kubernetes クラスターはこのロールを使用してノードを管理し、レガシークラウドプロバイダーはこのロールを使用して Elastic Load Balancing によるロードバランサーをサービス用に作成します。

Amazon EKS クラスターを使用するには、事前に次の IAM ポリシーのいずれかを持つ IAM ロールを作成する必要があります。

- AmazonEKSClusterPolicy
- カスタム IAM ポリシー。以下の最小限の許可では、Kubernetes クラスターがノードを管理することはできますが、レガシークラウドプロバイダーが Elastic Load Balancing によるロードバランサーを作成することはできません。カスタム IAM ポリシーには、少なくとも以下の許可が必要です。

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "ec2:CreateTags"
      ],
      "Resource": "arn:aws:ec2:*:*:instance/*",
      "Condition": {
        "ForAnyValue:StringLike": {
          "aws:TagKeys": "kubernetes.io/cluster/*"
        }
      }
    },
    {
      "Effect": "Allow",
      "Action": [
        "ec2:DescribeInstances",
        "ec2:DescribeNetworkInterfaces",
        "ec2:DescribeVpcs",
        "ec2:DescribeDhcpOptions",
        "ec2:DescribeAvailabilityZones",
        "ec2:DescribeInstanceTopology",
        "kms:DescribeKey"
      ],
      "Resource": "*"
    }
  ]
}
```

> **Note**
>
> 2023 年 10 月 3 日より前は、各クラスターの IAM ロールには AmazonEKSClusterPolicy が必要でした。

2020 年 4 月 16 日までは、AmazonEKSServicePolicy および AmazonEKSClusterPolicy が必須であり、ロールに推奨される名前は eksServiceRole でした。AWSServiceRoleForAmazonEKS のサービスにリンクされたロールにより、2020 年 4 月 16 日以降に作成されたクラスターに対しては、AmazonEKSServicePolicy ポリシーは必要なくなりました。

既存のクラスターロールの確認

次の手順を使用して、アカウントに Amazon EKS クラスターロールが既に存在しているかどうかが確認できます。

1. IAM コンソール (https://console.aws.amazon.com/iam/) を開きます。
2. 左のナビゲーションペインで、[ロール] を選択してください。
3. ロールのリストで eksClusterRole を検索します。eksClusterRole が含まれているロールが存在しない場合は、the section called “Amazon EKS でのクラスターロールの作成” を参照してロールを作成します。eksClusterRole が含まれているロールが存在する場合はこのロールを選択してアタッチされているポリシーを表示します。
4. [許可] を選択してください。
5. AmazonEKSClusterPolicy 管理ポリシーがロールにアタッチされていることを確認します。ポリシーがアタッチされている場合、Amazon EKS クラスターロールは適切に設定されています。
6. [Trust relationships] (信頼関係) を 選択し、[Edit trust policy] (信頼ポリシーの編集) を選択してください。
7. 信頼関係に以下のポリシーが含まれていることを確認します。信頼関係が以下のポリシーと一致する場合、[キャンセル] を選択してください。信頼関係が一致しない場合、ポリシーを [信頼ポリシーの編集] ウィンドウにコピーし、[ポリシーの更新] を選択します。

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "eks.amazonaws.com"
      },
      "Action": "sts:AssumeRole"
    }
  ]
```

```
}
```

Amazon EKS でのクラスターロールの作成

クラスターロールを作成するにはAWS マネジメントコンソール または AWS CLI を使用できます。

AWS マネジメントコンソール

a. https://console.aws.amazon.com/iam/ で IAM コンソールを開きます。
b. [ロール] を選択してから [ロールの作成] を選びます。
c. [信頼できるエンティティタイプ] の下で、[AWS サービス] を選択してください。
d. [他の AWS サービスのユースケース] ドロップダウンリストから、[EKS] を選択してください。
e. ユースケースに [EKS - Cluster] (EKS - クラスター) を 選択し、[次へ] (次へ) を 選択してください。
f. [Add permissions] (アクセス許可を追加する) タブで、[Next] (次へ) を選択します。
g. [ロール名] に、`eksClusterRole` などのロールの一意の名前を入力します。
h. [Description] (説明) には`Amazon EKS - Cluster role` のような説明文を入力してください。
i. [ロールの作成] を選択してください。

AWS CLI

a. 次の内容を *`cluster-trust-policy.jsonl`* という名前のファイルにコピーします。

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "eks.amazonaws.com"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
```

b. ロールを作成します。*`eksClusterRole`* は、任意の名前で置き換えることができます。

```
aws iam create-role \
  --role-name eksClusterRole \
  --assume-role-policy-document file://"cluster-trust-policy.json"
```

c. 必要な IAM ポリシーをロールにアタッチします。

```
aws iam attach-role-policy \
  --policy-arn arn:aws:iam::aws:policy/AmazonEKSClusterPolicy \
  --role-name eksClusterRole
```

Amazon EKS ノードの IAM ロール

Amazon EKS ノード kubelet デーモンが、ユーザーに代わって AWS API への呼び出しを実行します。ノードは、IAM インスタンスプロファイルおよび関連ポリシーを通じて、これらの API コールのアクセス許可を受け取ります。ノードを起動してクラスターに登録する前に、起動するときに使用するノード用の IAM ロールを作成する必要があります。この要件は、Amazon が提供する Amazon EKS 最適化 AMI で起動されたノード、または使用する予定の他のノード AMI で起動されたノードに適用されます。さらに、この要件はマネージド型ノードグループとセルフマネージド型ノードの両方に適用されます。

> **Note**
>
> クラスターの作成に使用したロールは使用できません。

ノードを使用するには、次の アクセス許可を持つ IAM ロールを作成する必要があります。

- kubelet が VPC 内の Amazon EC2 リソースを記述するためのアクセス許可 (AmazonEKSWorkerNodePolicy ポリシーで規定されているなど)。このポリシーは、Amazon EKS Pod Identity エージェントのアクセス許可も提供します。
- kubelet が Amazon Elastic Container Registry (Amazon ECR) のコンテナイメージを使用するためのアクセス許可 (AmazonEC2ContainerRegistryPullOnly ポリシーで規定されているなど)。ネットワーキング用の組み込みアドオンは Amazon ECR のコンテナイメージを使用するポッドを実行するため、Amazon Elastic Container Registry (Amazon ECR) のコンテナイメージを使用する許可が必要です。
- (オプション) Amazon EKS Pod Identity エージェントが eks-auth:AssumeRoleForPodIdentity アクションを使用してポッドの認証情報を取得するための

アクセス許可。AmazonEKSWorkerNodePolicy を使用しない場合、EKS Pod Identity を使用するには EC2 アクセス許可に加えてこのアクセス許可も提供する必要があります。

- (任意) IRSA または EKS Pod Identity を使用して VPC CNI ポッドにアクセス許可を付与しない場合は、インスタンスロールで VPC CNI のアクセス許可を付与する必要があります。AmazonEKS_CNI_Policy マネージドポリシー (IPv4 ファミリーを使用してクラスターを作成した場合) または作成した IPv6 ポリシー (IPv6 ファミリーを使用してクラスターを作成した場合) のいずれかを使用できます。ただし、このロールにポリシーをアタッチするのではなく、Amazon VPC CNI アドオン専用の別のロールにポリシーをアタッチすることをお勧めします。Amazon VPC CNI アドオン用に別のロールを作成する方法の詳細については、the section called “IRSA 用の設定” を参照してください。
- (オプション) kubelet が AmazonElasticContainerRegistryPublicReadOnly ポリシーなどで提供される、Amazon ECR Public (public.ecr.aws) のコンテナイメージを使用するためのアクセス許可。ECR Public からコンテナイメージを取得する場合は、これらのアクセス許可を設定することをお勧めします。アクセス許可がなくてもプルは成功しますが、レート制限がかかる可能性があります。

> **Note**
>
> Amazon EC2 ノードグループには、Fargate プロファイルとは異なる IAM ロールが必要です。詳細については、the section called “ポッド実行IAMロール” を参照してください。

既存のノードロールの確認

以下の手順を使用して、アカウントに既に Amazon EKS ノードロールがあるかどうかが確認できます。

1. IAM コンソール (https://console.aws.amazon.com/iam/) を開きます。
2. 左のナビゲーションペインで、[ロール] を選択してください。
3. ロールのリストで eksNodeRole、AmazonEKSNodeRole、または NodeInstanceRole を検索します。これらの名前のいずれかを持つロールが存在しない場合、「the section called “Amazon EKS ノードでの IAM ロールの作成”」を参照してロールを作成してください。eksNodeRole、AmazonEKSNodeRole、あるいは NodeInstanceRole を含むロールが存在する場合、そのロールを選択して、アタッチされているポリシーを表示します。
4. [許可] を選択してください。

5. AmazonEKSWorkerNodePolicy および AmazonEC2ContainerRegistryPullOnly のマネージドポリシーがロールにアタッチされていること、またはカスタムポリシーが最小限の許可でアタッチされていることを確認します。

> **Note**
>
> そのロールに [AmazonEKS_CNI_Policy] ポリシーがアタッチされている場合は、そのポリシーを一度削除して、aws-node Kubernetes サービスアカウントにマッピングされた IAM ロールにアタッチし直すことをお勧めします。詳細については、the section called "IRSA 用の設定" を参照してください。

6. [信頼関係] を選択し、[信頼ポリシーの編集] を選択します。
7. 信頼関係に以下のポリシーが含まれていることを確認します。信頼関係が以下のポリシーと一致する場合、[キャンセル] を選択してください。信頼関係が一致しない場合、ポリシーを [信頼ポリシーの編集] ウィンドウにコピーし、[ポリシーの更新] を選択してください。

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "sts:AssumeRole"
            ],
            "Principal": {
                "Service": [
                    "ec2.amazonaws.com"
                ]
            }
        }
    ]
}
```

Amazon EKS ノードでの IAM ロールの作成

AWS マネジメントコンソール または AWS CLI を使用して、ノードの IAM ロールを作成できます。

AWS マネジメントコンソール

a. IAM コンソール (https://console.aws.amazon.com/iam/) を開きます。

b. 左のナビゲーションペインで、[ロール] を選択してください。

c. [ロール] ページで、[ロールの作成] を選択してください。

d. [信頼されたエンティティを選択] ページで、以下の操作を実行します。

i. [信頼するエンティティのタイプ] で [AWS サービス] を選択してください。

ii. [ユースケース] で、[EC2] を選択してください。

iii. [次へ] を選択します。

e. [アクセス許可を追加] ページで、カスタムポリシーをアタッチするか、以下の操作を行います。

i. [フィルタポリシー] ボックスに `AmazonEKSWorkerNodePolicy` と入力します。

ii. 検索結果の [AmazonEKSWorkerNodePolicy] の左にあるチェックボックスを選択します。

iii. [フィルターのクリア] を選択します。

iv. [フィルタポリシー] ボックスに `AmazonEC2ContainerRegistryPullOnly` と入力します。

v. 検索結果の [AmazonEC2ContainerRegistryPullOnly] の左にあるチェックボックスを選択します。

このロールまたは `aws-node` Kubernetes サービスアカウントにマッピングされた別のロールに、AmazonEKS_CNI_Policy マネージドポリシーまたは作成した IPv6 ポリシーをアタッチする必要もあります。このロールに対してではなく、Kubernetes サービスアカウントに関連付けられたロールに、前出のポリシーを割り当てることをお勧めします。詳細については、the section called “IRSA 用の設定” を参照してください。

vi. [次へ] を選択します。

f. [名前を付けて、レビューし、作成する] ページで、以下の操作を実行します。

i. [ロール名] に、`AmazonEKSNodeRole` などのロールの一意の名前を入力します。

ii. [説明] では現在のテキストを「`Amazon EKS - Node role`」などの説明文に置き換えます。

iii. [タグの追加 (オプション)] で、タグをキーバリューのペアとして添付して、メタデータをロールに追加します。IAM でのタグの使用に関する詳細については『IAM ユーザーガイド』の「IAM リソースにタグを付ける」を参照してください。

iv. [ロールの作成] を選択してください。

AWS CLI

a. 次のコマンドを実行して、`node-role-trust-relationship.json` ファイルを作成します。

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "sts:AssumeRole"
            ],
            "Principal": {
                "Service": [
                    "ec2.amazonaws.com"
                ]
            }
        }
    ]
}
```

b. IAM ロールを作成します。

```
aws iam create-role \
  --role-name AmazonEKSNodeRole \
  --assume-role-policy-document file://"node-role-trust-relationship.json"
```

c. IAM ロールに、2 つの必須なマネージド IAM ポリシーをアタッチします。

```
aws iam attach-role-policy \
  --policy-arn arn:aws:iam::aws:policy/AmazonEKSWorkerNodePolicy \
  --role-name AmazonEKSNodeRole
aws iam attach-role-policy \
  --policy-arn arn:aws:iam::aws:policy/AmazonEC2ContainerRegistryPullOnly \
  --role-name AmazonEKSNodeRole
```

d. クラスターを作成した IP ファミリーに応じて、次の IAM ポリシーの 1 つを IAM ロールにアタッチします。このポリシーは、このロール、または、Kubernetes の aws-node サービスアカウントに関連付けられたロールにアタッチされる必要があります。これらのロールは、Amazon VPC CNI plugin for Kubernetes で使用されます。このポリシーは、Kubernetes サービスアカウントに関連付けられたロールにを割り当てることをお勧めします。Kubernetes サービスアカウントに関連付けられたロールへのポリシーの割り当てについては、「the section called “IRSA 用の設定”」を参照してください。

- IPv4

```
aws iam attach-role-policy \
  --policy-arn arn:aws:iam::aws:policy/AmazonEKS_CNI_Policy \
  --role-name AmazonEKSNodeRole
```

- IPv6

  A. 次のテキストをコピーし、vpc-cni-ipv6-policy.json という名前のファイルに保存します。

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "ec2:AssignIpv6Addresses",
                "ec2:DescribeInstances",
                "ec2:DescribeTags",
                "ec2:DescribeNetworkInterfaces",
                "ec2:DescribeInstanceTypes"
            ],
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "ec2:CreateTags"
            ],
            "Resource": [
                "arn:aws:ec2:*:*:network-interface/*"
            ]
        }
    ]
}
```

  B. IAM ポリシーを作成します。

```
aws iam create-policy --policy-name AmazonEKS_CNI_IPv6_Policy --policy-
document file://vpc-cni-ipv6-policy.json
```

  C. IAM ロールに IAM ポリシーをアタッチします。*111122223333* は、ご自分のアカウント ID に置き換えます。

```
aws iam attach-role-policy \
  --policy-arn arn:aws:iam::111122223333:policy/AmazonEKS_CNI_IPv6_Policy \
  --role-name AmazonEKSNodeRole
```

Amazon EKS 自動モードI クラスター IAM ロール

Amazon EKS クラスター IAM ロールはクラスターごとに必要です。Amazon EKS によって管理される Kubernetes クラスターはこのロールを使用して、ストレージ、ネットワーキング、コンピューティングの自動スケーリングのルーチンタスクを自動化します。

Amazon EKS クラスターを使用するにはEKS 自動モードI に必要なポリシーを持つ IAM ロールを作成する必要があります。推奨の AWS IAM マネージドポリシーをアタッチするか、同等のアクセス許可を持つカスタムポリシーを作成することができます。

- Amazon EKSコンピュートポリシー
- AmazonEKSBlockStoragePolicyV2
- Amazon EKSロードバランシングポリシー
- Amazon EKSネットワーキングポリシー
- AmazonEKSクラスターポリシー

EKS Auto リソースのカスタム AWS タグ

デフォルトでは、EKS Auto Mode に関連するマネージドポリシーでは、ユーザー定義タグを Auto Mode でプロビジョニングされた AWS リソースに適用することはできません。ユーザー定義タグを AWS リソースに適用する場合は、AWS リソースのタグを作成および変更するのに十分なアクセス許可を持つクラスター IAM ロールに追加のアクセス許可をアタッチする必要があります。以下は、無制限のタグ付けアクセスを許可するポリシーの例です。

カスタムタグポリシーの例を表示する

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "Compute",
            "Effect": "Allow",
            "Action": [
```

```
                "ec2:CreateFleet",
                "ec2:RunInstances",
                "ec2:CreateLaunchTemplate"
            ],
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "aws:RequestTag/eks:eks-cluster-name": "${aws:PrincipalTag/eks:eks-
cluster-name}"
                },
                "StringLike": {
                    "aws:RequestTag/eks:kubernetes-node-class-name": "*",
                    "aws:RequestTag/eks:kubernetes-node-pool-name": "*"
                }
            }
        },
        {
            "Sid": "Storage",
            "Effect": "Allow",
            "Action": [
                "ec2:CreateVolume",
                "ec2:CreateSnapshot"
            ],
            "Resource": [
                "arn:aws:ec2:*:*:volume/*",
                "arn:aws:ec2:*:*:snapshot/*"
            ],
            "Condition": {
                "StringEquals": {
                    "aws:RequestTag/eks:eks-cluster-name": "${aws:PrincipalTag/eks:eks-
cluster-name}"
                }
            }
        },
        {
            "Sid": "Networking",
            "Effect": "Allow",
            "Action": "ec2:CreateNetworkInterface",
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "aws:RequestTag/eks:eks-cluster-name": "${aws:PrincipalTag/eks:eks-
cluster-name}"
                },
```

```
                "StringLike": {
                    "aws:RequestTag/eks:kubernetes-cni-node-name": "*"
                }
            }
        },
        {
            "Sid": "LoadBalancer",
            "Effect": "Allow",
            "Action": [
                "elasticloadbalancing:CreateLoadBalancer",
                "elasticloadbalancing:CreateTargetGroup",
                "elasticloadbalancing:CreateListener",
                "elasticloadbalancing:CreateRule",
                "ec2:CreateSecurityGroup"
            ],
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "aws:RequestTag/eks:eks-cluster-name": "${aws:PrincipalTag/eks:eks-
cluster-name}"
                }
            }
        },
        {
            "Sid": "ShieldProtection",
            "Effect": "Allow",
            "Action": [
                "shield:CreateProtection"
            ],
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "aws:RequestTag/eks:eks-cluster-name": "${aws:PrincipalTag/eks:eks-
cluster-name}"
                }
            }
        },
        {
            "Sid": "ShieldTagResource",
            "Effect": "Allow",
            "Action": [
                "shield:TagResource"
            ],
            "Resource": "arn:aws:shield::*:protection/*",
```

```
            "Condition": {
                "StringEquals": {
                    "aws:RequestTag/eks:eks-cluster-name": "${aws:PrincipalTag/eks:eks-
cluster-name}"
                }
            }
        }
    ]
}
```

既存のクラスターロールの確認

次の手順を使用して、アカウントに Amazon EKS クラスターロールが既に存在しているかどうかが確認できます。

1. IAM コンソール (https://console.aws.amazon.com/iam/) を開きます。
2. 左のナビゲーションペインで、[ロール] を選択してください。
3. ロールのリストで AmazonEKSAutoClusterRole を検索します。AmazonEKSAutoClusterRole を含むロールが存在しない場合は次のセッションの手順を参照してロールを作成します。AmazonEKSAutoClusterRole が含まれているロールが存在する場合はこのロールを選択してアタッチされているポリシーを表示します。
4. [許可] を選択してください。
5. AmazonEKSClusterPolicy 管理ポリシーがロールにアタッチされていることを確認します。ポリシーがアタッチされている場合、Amazon EKS クラスターロールは適切に設定されています。
6. [Trust relationships] (信頼関係）を 選択し、[Edit trust policy] (信頼ポリシーの編集）を選択してください。
7. 信頼関係に以下のポリシーが含まれていることを確認します。信頼関係が以下のポリシーと一致する場合、[キャンセル] を選択してください。信頼関係が一致しない場合、ポリシーを [信頼ポリシーの編集] ウィンドウにコピーし、[ポリシーの更新] を選択してください。

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "eks.amazonaws.com"
      },
      "Action": [
```

```
        "sts:AssumeRole",
        "sts:TagSession"
      ]
    }
  ]
}
```

> **Note**
>
> AWS ではこのロールの名前 AmazonEKSAutoClusterRole は必要ありません。

Amazon EKS でのクラスターロールの作成

クラスターロールを作成するにはAWS マネジメントコンソール または AWS CLI を使用できます。

AWS マネジメントコンソール

1. https://console.aws.amazon.com/iam/ で IAM コンソールを開きます。
2. [ロール] を選択してから [ロールの作成] を選びます。
3. [信頼できるエンティティタイプ] の下で、[AWS サービス] を選択してください。
4. [他の AWS サービスのユースケース] ドロップダウンリストから、[EKS] を選択してください。
5. ユースケースに [EKS - Cluster] (EKS - クラスター) を 選択し、[次へ] (次へ) を 選択してください。
6. [アクセス許可を追加] タブで、ポリシーを選択し、[次へ] を選択してください。
   - Amazon EKSコンピュートポリシー
   - AmazonEKSBlockStoragePolicyV2
   - Amazon EKSロードバランシングポリシー
   - Amazon EKSネットワーキングポリシー
   - AmazonEKSクラスターポリシー
7. [ロール名] に、AmazonEKSAutoClusterRole などのロールの一意の名前を入力します。
8. [Description] (説明) にはAmazon EKS - Cluster role のような説明文を入力してください。
9. [ロールの作成] を選択してください。

AWS CLI

1. 次の内容を *cluster-trust-policy.jsonl* という名前のファイルにコピーします。

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "eks.amazonaws.com"
      },
      "Action": [
        "sts:AssumeRole",
        "sts:TagSession"
      ]
    }
  ]
}
```

2. ロールを作成します。*AmazonEKSAutoClusterRole* は任意の名前に置き換えることができます。

```
aws iam create-role \
  --role-name AmazonEKSAutoClusterRole \
  --assume-role-policy-document file://"cluster-trust-policy.json"
```

3. 必要な IAM ポリシーをロールにアタッチします：

AmazonEKSClusterPolicy:

```
aws iam attach-role-policy \
    --role-name AmazonEKSAutoClusterRole \
    --policy-arn arn:aws:iam::aws:policy/AmazonEKSClusterPolicy
```

AmazonEKSコンピュートポリシー:

```
aws iam attach-role-policy \
    --role-name AmazonEKSAutoClusterRole \
    --policy-arn arn:aws:iam::aws:policy/AmazonEKSComputePolicy
```

AmazonEKSBlockStoragePolicyV2:

```
aws iam attach-role-policy \
    --role-name AmazonEKSAutoClusterRole \
    --policy-arn arn:aws:iam::aws:policy/AmazonEKSBlockStoragePolicyV2
```

AmazonEKSロードバランシングポリシー:

```
aws iam attach-role-policy \
    --role-name AmazonEKSAutoClusterRole \
    --policy-arn arn:aws:iam::aws:policy/AmazonEKSLoadBalancingPolicy
```

Amazon EKSネットワーキングポリシー:

```
aws iam attach-role-policy \
    --role-name AmazonEKSAutoClusterRole \
    --policy-arn arn:aws:iam::aws:policy/AmazonEKSNetworkingPolicy
```

Amazon EKS 自動モードI ノードの IAM ロール

> **Note**
>
> クラスターの作成に使用したロールは使用できません。

ノードを作成する前に、次のパイプラインまたは同等のアクセス許可を持つ IAM ロールを作成する必要があります：

- Amazon EKSワーカーノードミニマルポリシー
- Amazon EC2コンテナレジストリプルオンリー

既存のノードロールの確認

以下の手順を使用して、アカウントに既に Amazon EKS ノードロールがあるかどうかが確認できます。

1. IAM コンソール (https://console.aws.amazon.com/iam/) を開きます。
2. 左のナビゲーションペインで、[ロール] を選択してください。

3. ロールのリストで AmazonEKSAutoNodeRole を検索します。これらの名前のいずれかを持つロールが存在しない場合は次のセクションの手順を参照してロールを作成します。AmazonEKSAutoNodeRole を含むロールがある場合はロールを選択して、アタッチされたポリシーを表示します。
4. [許可] を選択してください。
5. 上記の必要なポリシーまたは同等のカスタムポリシーがアタッチされていることを確認します。
6. [信頼関係] を選択し、[信頼ポリシーの編集] を選択してください。
7. 信頼関係に以下のポリシーが含まれていることを確認します。信頼関係が以下のポリシーと一致する場合、[キャンセル] を選択してください。信頼関係が一致しない場合、ポリシーを [信頼ポリシーの編集] ウィンドウにコピーし、[ポリシーの更新] を選択してください。

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "ec2.amazonaws.com"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
```

Amazon EKS ノードでの IAM ロールの作成

AWS マネジメントコンソール または AWS CLI を使用して、ノードの IAM ロールを作成できます。

AWS マネジメントコンソール

1. IAM コンソール (https://console.aws.amazon.com/iam/) を開きます。
2. 左のナビゲーションペインで、[ロール] を選択してください。
3. [ロール] ページで、[ロールの作成] を選択してください。
4. [信頼されたエンティティを選択] ページで、以下の操作を実行します。
   a. [信頼するエンティティのタイプ] で [AWS サービス] を選択してください。
   b. [ユースケース] で、[EC2] を選択してください。
   c. [次へ] を選択してください。

5. [アクセス許可を追加] ページで、次のポリシーをアタッチします：
   - Amazon EKSワーカーノードミニマルポリシー
   - Amazon EC2コンテナレジストリプルオンリー
6. [名前を付けて、レビューし、作成する] ページで、以下の操作を実行してください：
   a. [ロール名] に、AmazonEKSAutoNodeRole などのロールの一意の名前を入力します。
   b. [説明] では現在のテキストを「Amazon EKS - Node role」などの説明文に置き換えます。
   c. [タグの追加 (オプション)] で、タグをキーバリューのペアとして添付して、メタデータをロールに追加します。IAM でのタグの使用に関する詳細については『IAM ユーザーガイド』の「IAM リソースにタグを付ける」を参照してください。
   d. [ロールの作成] を選択してください。

AWS CLI

ノード IAM ロールを作成する

前のステップの node-trust-policy.json ファイルを使用して、ロールを引き受けることができるエンティティを定義します。次のコマンドを実行してノード IAM ロールを作成します：

```
aws iam create-role \
    --role-name AmazonEKSAutoNodeRole \
    --assume-role-policy-document file://node-trust-policy.json
```

ロールの ARN をメモする

ロールを作成したら、ノード IAM ロールの ARN を取得して保存します。以降のステップでこの ARN が必要になります。次のコマンドを使用して ARN を取得します：

```
aws iam get-role --role-name AmazonEKSAutoNodeRole --query "Role.Arn" --output text
```

必要なポリシーをアタッチする

次の AWS マネージドポリシーをノード IAM ロールにアタッチして、必要なアクセス許可を付与します：

Amazon EKSワーカーノードミニマルポリシー をアタッチするには:

```
aws iam attach-role-policy \
```

```
    --role-name AmazonEKSAutoNodeRole \
    --policy-arn arn:aws:iam::aws:policy/AmazonEKSWorkerNodeMinimalPolicy
```

Amazon EC2コンテナレジストリプルオンリー をアタッチするには:

```
aws iam attach-role-policy \
    --role-name AmazonEKSAutoNodeRole \
    --policy-arn arn:aws:iam::aws:policy/AmazonEC2ContainerRegistryPullOnly
```

オプションポリシーをアタッチする

パブリック ECR からコンテナイメージをプルする場合は、ノードがパブリック ECR に対して認証し、スロットリングなしでイメージをプルできるように、以下の AWS 管理ポリシーをノード IAM ロールにアタッチする必要があります。

AmazonElasticContainerRegistryPublicReadOnly をアタッチするには:

```
aws iam attach-role-policy \
    --role-name AmazonEKSAutoNodeRole \
    --policy-arn arn:aws:iam::aws:policy/AmazonElasticContainerRegistryPublicReadOnly
```

Amazon EKS 機能 IAM ロール

EKS 機能には、機能 IAM ロール (または機能ロール) を設定する必要があります。機能は、このロールを使用して AWS サービスに対してアクションを実行し、自動的に作成されたアクセスエントリを介してクラスター内の Kubernetes リソースにアクセスします。

機能の作成時に機能ロールを指定する場合は、事前に IAM ロールを作成してその機能のタイプに適した信頼ポリシーとアクセス許可を付与する必要があります。このように作成した IAM ロールは、任意の数の機能リソースで再利用できます。

機能ロールの要件

機能ロールは、以下の要件を満たしている必要があります。

- クラスターおよび機能リソースと同じ AWS アカウントに存在する必要があります
- 信頼ポリシーにより、EKS 機能サービスがロールを引き受けることを許可する必要があります。
- 機能タイプとユースケースの要件に適したアクセス許可を付与する必要があります (「the section called “機能タイプ別のアクセス許可”」を参照)。

機能ロールの信頼ポリシー

すべての機能ロールに、次の信頼ポリシーを含める必要があります。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "capabilities.eks.amazonaws.com"
      },
      "Action": [
        "sts:AssumeRole",
        "sts:TagSession"
      ]
    }
  ]
}
```

この信頼ポリシーにより、EKS は以下のことを行うことができます。

- AWS API オペレーションを実行するロールを引き受けます。
- 監査と追跡の目的でセッションにタグを付けます。

機能タイプ別のアクセス許可

必要な IAM アクセス許可は、使用している機能と現在のデプロイモデルによって異なります。

> **Note**
>
> 本番稼働でのデプロイに IAM ロールセレクターと ACK を使用する場合や、AWS サービスを統合することなく kro または Argo CD を使用する場合、機能ロールには信頼ポリシー以外に IAM アクセス許可は必要ありません。

kro (Kube Resource Orchestrator)

必須の IAM アクセス許可はありません。ポリシーをアタッチせずに機能ロールを作成できます。kro に必要なのは Kubernetes リソースを作成および管理するための Kubernetes RBAC アクセス許可のみです。

Kubernetes 用 AWS コントローラー (ACK)

ACK は、2 つのアクセス許可モデルをサポートしています。

- シンプルなセットアップ (開発/テスト): AWS サービスアクセス許可を機能ロールに直接追加します。これは、使用を開始する場合、シングルアカウントでデプロイする場合、またはすべてのユーザーに同じアクセス許可が必要な場合に適しています。
- 本番稼働のベストプラクティス: IAM ロールセレクターを使用して最小特権のアクセスを実装します。このアプローチで機能ロールに必要なのは、サービス固有のロールを引き受ける `sts:AssumeRole` アクセス許可のみです。機能ロール自体に AWS サービスアクセス許可 (S3 や RDS など) を追加しないでください。そうしたアクセス許可は、特定の名前空間にマッピングされた個々の IAM ロールに付与します。

  IAM ロールセレクターにより、以下のことを行うことができます。

  - 名前空間レベルでのアクセス許可の分離
  - クロスアカウントリソース管理
  - チームに固有の IAM ロール
  - 最小特権セキュリティモデル

  IAM ロールセレクターによるアプローチでの機能ロールポリシーの例:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "sts:AssumeRole",
      "Resource": [
        "arn:aws:iam::111122223333:role/ACK-S3-Role",
        "arn:aws:iam::111122223333:role/ACK-RDS-Role",
        "arn:aws:iam::444455556666:role/ACKCrossAccountRole"
      ]
    }
  ]
}
```

  IAM ロールセレクターなど ACK アクセス許可の詳細な設定については、「the section called “アクセス許可の設定”」を参照してください。

Argo CD

デフォルトでは、必須の IAM アクセス許可はありません。以下のものには、オプションでアクセス許可が必要になる場合があります。

- AWS Secrets Manager: Secrets Manager を使用して Git リポジトリ認証情報を保存する場合
- AWS CodeConnections: Git リポジトリの認証に CodeConnections を使用する場合

  Secrets Manager と CodeConnections のポリシーの例:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "secretsmanager:GetSecretValue",
        "secretsmanager:DescribeSecret"
      ],
      "Resource": "arn:aws:secretsmanager:region:account-id:secret:argocd/*"
    },
    {
      "Effect": "Allow",
      "Action": [
        "codeconnections:UseConnection",
        "codeconnections:GetConnection"
      ],
      "Resource": "arn:aws:codeconnections:region:account-id:connection/*"
    }
  ]
}
```

  Argo CD アクセス許可要件の詳細については、「the section called “Argo CD に関する考慮事項”」を参照してください。

既存の機能ロールを確認する

アカウントに既にユースケースに適した機能 IAM ロールがあるかどうかを確認するには、次の手順を使用できます。

1. IAM コンソール (https://console.aws.amazon.com/iam/) を開きます。

2. 左のナビゲーションペインで、[ロール] を選択してください。
3. ロールのリストを検索して、機能ロール名 (ACKCapabilityRole や ArgoCDCapabilityRole など) を確認します。
4. ロールが存在する場合は、そのロールを選択して、アタッチされたポリシーおよび信頼関係を表示します。
5. [信頼関係] を選択し、[信頼ポリシーの編集] を選択してください。
6. 信頼関係が機能信頼ポリシーと一致することを確認します。一致しない場合は、信頼ポリシーを更新します。
7. [アクセス許可] を選択し、ロールに機能タイプとユースケースに適したアクセス許可が付与されていることを確認します。

機能 IAM ロールを作成する

AWS マネジメントコンソール または AWS CLI を使用して、機能ロールを作成できます。

AWS マネジメントコンソール

a. https://console.aws.amazon.com/iam/ で IAM コンソールを開きます。
b. [ロール] を選択してから [ロールの作成] を選びます。
c. [信頼されたエンティティタイプ] で、[カスタム信頼ポリシー] を選択します。
d. 機能信頼ポリシーをコピーして、信頼ポリシーエディタに貼り付けます。
e. [次へ] を選択します。
f. [アクセス許可を追加] タブで、機能タイプに適したポリシーを選択または作成します (「the section called “機能タイプ別のアクセス許可”」を参照)。kro では、このステップをスキップできます。
g. [次へ] を選択します。
h. [ロール名] に、ACKCapabilityRole、ArgoCDCapabilityRole、kroCapabilityRole などロールの一意の名前を入力します。
i. [Description] (説明) にはAmazon EKS - ACK capability role のような説明文を入力してください。
j. [ロールの作成] を選択してください。

AWS CLI

a. 機能信頼ポリシーを capability-trust-policy.json という名前のファイルにコピーします。

b. ロールを作成します。ACKCapabilityRole を目的のロール名で置き換えます。

```
aws iam create-role \
  --role-name ACKCapabilityRole \
  --assume-role-policy-document file://capability-trust-policy.json
```

c. 必要な IAM ポリシーをロールにアタッチします。ACK の場合、管理する AWS サービスのポリシーをアタッチします。Argo CD の場合、必要に応じて Secrets Manager または CodeConnections のポリシーをアタッチします。kro では、このステップをスキップできます。

ACK に S3 アクセス許可を付与した例:

```
aws iam put-role-policy \
  --role-name ACKCapabilityRole \
  --policy-name S3Management \
  --policy-document file://s3-policy.json
```

機能ロールの問題をトラブルシューティングする

機能の作成が「無効な IAM ロール」で失敗する

以下を確認してください。

- ロールがクラスターと同じアカウントに存在する
- 信頼ポリシーが機能信頼ポリシーと一致する
- ロールに iam:PassRole アクセス許可が付与されている

機能でアクセス許可エラーが表示される

以下を確認してください。

- ロールに、機能タイプに必要な IAM アクセス許可が付与されている
- ロールのアクセスエントリがクラスターに存在する
- 必要に応じて追加の Kubernetes アクセス許可が設定される (「the section called “その他の Kubernetes アクセス許可”」を参照)

ACK リソースが「アクセス許可が拒否されました」というエラーで失敗する

以下を確認してください。

- ロールに、ユースケースに必要な IAM アクセス許可が付与されている

- ACK コントローラーがシークレットを参照する場合、適切な名前空間に範囲が限定された `AmazonEKSSecretReaderPolicy` アクセスエントリポリシーを関連付けていることを確認します。

トラブルシューティングの詳しいガイダンスについては、「the section called “EKS 機能に関する考慮事項”」を参照してください。

クラスターのパフォーマンスをモニタリングし、ログを表示する

Amazon EKS では、利用可能な多くのモニタリングツールまたはログ記録ツールを使用して、データを観測できます。Amazon EKS ログデータを AWS サービスまたはパートナーツールにストリーミングしてデータ分析することができます。Amazon EKS の問題をトラブルシューティングするためのデータを提供する多くのサービスが、AWS マネジメントコンソール で利用可能です。また、AWS でサポートされているオープンソースソリューションを使用して、Amazon EKS インフラストラクチャをモニタリング することもできます。

Amazon EKS コンソールの左側のナビゲーションペインで [クラスター] を選択した後、クラスターの名前を選択して [オブザーバビリティ] タブを選択すると、クラスターのヘルスと詳細を表示できます。クラスターにデプロイされている既存の Kubernetes リソースの詳細を表示するには、the section called “クラスターリソースにアクセスする” を参照してください。

モニタリングは、Amazon EKS と AWS ソリューションの信頼性、可用性、パフォーマンスを維持する上で重要なパートです。モニタリングのデータを AWS ソリューションのすべての部分から収集することをお勧めします。これにより、マルチポイント障害が発生した場合は、その障害をより簡単にデバッグできます。Amazon EKS のモニタリングを開始する前に、モニタリング計画が以下の質問に対処していることを確認してください。

- 目標は何ですか。クラスターが劇的に拡大する場合、リアルタイムの通知が必要ですか。
- どのリソースを観測する必要がありますか。
- これらのリソースをどのくらいの頻度で観測する必要がありますか。あなたの会社はリスクに迅速に対応したいですか。
- どのツールを使用する予定ですか。起動の一環として既に AWS Fargate を実行している場合は、組み込みログルーターを使用できます。
- 誰がモニタリングタスクを実行する予定ですか。
- 何か問題が発生したときに通知を誰に送信したいですか。

Amazon EKS でのモニタリングとログ記録

Amazon EKS は、モニタリングとログ記録用の組み込みツールを提供します。サポートされているバージョンでは、オブザーバビリティダッシュボードは、クラスターのパフォーマンスに関する可視性を提供します。これは、問題を迅速に検出、トラブルシューティング、是正するのに役立ちま

す。モニタリング機能に加えて、コントロールプレーンの監査ログに基づくリストも含まれています。Kubernetes コントロールプレーンは、コンソールの外部でのスクレイピングも可能な多数のメトリクスを公開します。

コントロールプレーンのログ記録は、クラスターへのすべての API 呼び出し、クラスターに対してどのユーザーがどのアクションを実行したかをキャプチャする監査情報、およびロールベースの情報を記録します。詳細については、AWS 規範的ガイダンスの「Amazon EKS でのロギングとモニタリング」を参照してください。

Amazon EKS コントロールプレーンのログ記録により、アカウント内で Amazon EKS コントロールプレーンから CloudWatch ログに対し、監査および診断ログを直接送れるようになります。これらのログを使用すると、クラスターの保護と実行が容易になります。必要なログタイプを正確に選択することで、CloudWatch 内で各 Amazon EKS クラスターのためのグループに対し、ログストリームの形態でログを送信できます。詳細については、the section called “コントロールプレーンのログ” を参照してください。

> **Note**
>
> Amazon CloudWatch で Amazon EKS の認証ログをチェックすると、次のサンプルテキストのようなテキストを含むエントリが表示されます。
>
> ```
> level=info msg="mapping IAM role" groups="[]"
>  role="arn:aws:iam::111122223333:role/XXXXXXXXXXXXXXXXX-NodeManagerRole-
> XXXXXXXX" username="eks:node-manager"
> ```
>
> このテキストを含むエントリが必要です。username は、マネージド型ノードグループと Fargate の特定のオペレーションを実行する、Amazon EKS の内部サービスロールです。低レベルでカスタマイズ可能なログ記録には、Kubernetes ログ記録 が利用可能です。

Amazon EKS は、AWS CloudTrail と統合されています。このサービスにより、Amazon EKS のユーザー、ロールまたは AWS のサービスによって実行されたアクションを記録します。CloudTrail は、Amazon EKS へのすべての API コールをイベントとしてキャプチャします。キャプチャされた呼び出しには、Amazon EKS コンソールからの呼び出しと、Amazon EKS API オペレーションへのコード呼び出しが含まれます。詳細については、the section called “ AWS CloudTrail” を参照してください。

Kubernetes API サーバーは、モニタリングと分析に役立つ多数のメトリクスを公開します。詳細については、the section called “Prometheus メトリクス” を参照してください。

Fluent Bit をカスタムの Amazon CloudWatch Logs に設定するには、「Amazon CloudWatch ユーザーガイド」の「Fluent Bit の設定」を参照してください。

Amazon EKS のモニタリングとログ記録のツール

Amazon Web Services では、Amazon EKS のモニタリングに使用できるさまざまなツールを提供しています。一部のツールは自動モニタリングを設定できますが、手動呼び出しが必要なツールもあります。環境および既存のツールセットで許容される範囲で、できるだけモニタリングタスクを自動化することをお勧めします。

次の表に、さまざまなモニタリングツールのオプションを示します。

エリア	ツール	説明	セットアップ
コントロールプレーン	オブザーバビリティダッシュボード	サポートされているバージョンでは、オブザーバビリティダッシュボードは、クラスターのパフォーマンスに関する可視性を提供します。これは、問題を迅速に検出、トラブルシューティング、是正するのに役立ちます。	設定手順
アプリケーション/コントロールプレーン	Prometheus	Prometheus は、アプリケーションとコントロールプレーンのメトリクスとアラートをモニタリングするために使用できます。	設定手順
アプリケーション	CloudWatch Container Insights	CloudWatch Container Insights は、コンテナ化されたアプリケーション	設定手順

エリア	ツール	説明	セットアップ
		とマイクロサービスのメトリクスとログを収集、集約、要約します。	
アプリケーション	AWS Distro for OpenTelemetry (ADOT)	ADOT は、相関メトリクス、トレースデータ、メタデータを収集し、AWS モニタリングサービスまたはパートナーに送信できます。これは、CloudWatch Container Insights を介して設定できます。	設定手順
アプリケーション	Amazon DevOps Guru	Amazon DevOps Guru は、ノードレベルの運用パフォーマンスと可用性を検出します。	設定手順

エリア	ツール	説明	セットアップ
アプリケーション	AWS X-Ray	AWS X-Ray はアプリケーションに関するトレースデータを受信します。このトレースデータには、着信と送信のリクエスト、およびリクエストに関するメタデータが含まれます。Amazon EKS の場合、実装には OpenTelemetry アドオンが必要です。	設定手順
アプリケーション	Amazon CloudWatch	CloudWatch は、サポートされているバージョンでいくつかの基本的な Amazon EKS メトリクスを無料で提供します。CloudWatch Observability Operator を使用してこの機能を拡張し、メトリクス、ログ、トレースデータの収集を処理できます。	設定手順

次の表に、さまざまなログ記録ツールのオプションを示します。

エリア	ツール	説明	セットアップ
コントロールプレーン	オブザーバビリティダッシュボード	サポートされているバージョンでは、オ	設定手順

エリア	ツール	説明	セットアップ
		ブザーバビリティダッシュボードは、コントロールプレーンの監査ログに基づくリストを表示します。また、Amazon CloudWatch のコントロールプレーンログへのリンクも含まれています。	
アプリケーション	Amazon CloudWatch Container Insights	Amazon CloudWatch Container Insights は、コンテナ化されたアプリケーションおよびマイクロサービスのメトリクスとログを収集、集約、要約します。	設定手順
コントロールプレーン	Amazon CloudWatch Logs	監査ログと診断ログを Amazon EKS コントロールプレーンからアカウントの CloudWatch Logs に直接送信できます。	設定手順
コントロールプレーン	AWS CloudTrail	ユーザー、ロール、またはサービスによる API 呼び出しをログに記録します。	設定手順

エリア	ツール	説明	セットアップ
AWS Fargate インスタンスの複数のエリア	AWS Fargate ログルーター	AWS Fargate インスタンスの場合、ログルーターはログを AWS サービスまたはパートナーツールにストリーミングします。AWS for Fluent Bit を使用します。ログは他の AWS サービスまたはパートナーツールにストリーミングできます。	設定手順

オブザーバビリティダッシュボードを使用してクラスターをモニタリングする

Amazon EKS コンソールにはクラスターのパフォーマンスに関する可視性を提供するオブザーバビリティダッシュボードが含まれています。このダッシュボードの情報は、問題を迅速に検出、トラブルシューティング、是正するのに役立ちます。オブザーバビリティダッシュボードの該当するセクションを開くには、[ヘルスとパフォーマンスの概要] で項目を選択してください。この概要は[オブザーバビリティ] タブなど、いくつかの場所に含まれています。

オブザーバビリティダッシュボードは、複数のタブに分かれています。

概要

[ヘルスとパフォーマンスの概要] にはさまざまなカテゴリの項目数が一覧表示されます。各数値はそのカテゴリのリストを含むオブザーバビリティダッシュボード内の場所へのハイパーリンクとして機能します。

クラスターヘルス

クラスターヘルスは注意すべき重要な通知であり、その中には可能な限り早急に対処しなければならない可能性のあるものも含まれています。このリストでは説明と、影響を受けるリソースを確認でき

ます。クラスターヘルスには、[ヘルスの問題] と [設定情報] という 2 つのテーブルが含まれています。[ヘルスの問題] のステータスを更新するには、更新ボタン (↻) を選択してください。[設定情報] は、24 時間に 1 回自動的に更新されます。手動で更新することはできません。

[ヘルスの問題] の詳細については、「the section called “クラスターの正常性に関するよくある質問およびエラーコードと解決パス”」を参照してください。[設定情報] の詳細については、「the section called “クラスターのインサイト”」を参照してください。

コントロールプレーンのモニタリング

[コントロールプレーンのモニタリング] タブは 3 つのセクションに分かれており、それぞれがクラスターのコントロールプレーンのモニタリングとトラブルシューティングに役立ちます。

メトリクス

Kubernetes バージョン 1.28 以降のクラスターの場合、[メトリクス] セクションにはさまざまなコントロールプレーンコンポーネントについて収集された複数のメトリクスのグラフが表示されます。

セクションの上部で選択することで、すべてのグラフの X 軸で使用される期間を設定できます。更新ボタン (↻) を使用してデータを更新できます。個別のグラフごとに、縦三点リーダーのボタン (⋮) をクリックすると、CloudWatch のオプションを含むメニューが開きます。

これらのメトリクスなどはCloudWatch における AWS/EKS 名前空間の基本的なモニタリングメトリクスとして自動的に使用できます。詳細については「Amazon CloudWatch ユーザーガイド」の「基本モニタリングと詳細モニタリング」を参照してください。より詳細なメトリクス、ビジュアライゼーション、インサイトを取得するには「Amazon CloudWatch ユーザーガイド」の「Container Insights」を参照してください。あるいは、Prometheus ベースのモニタリングが必要な場合は「the section called “Prometheus メトリクス”」を参照してください。

次の表には使用可能なメトリクスが記載されています。

メトリクス	説明
APIServer リクエスト	API サーバーに対して実行された 1 分あたりのリクエスト。
APIServer 合計リクエスト 4XX	HTTP 4XX レスポンスコード (クライアント側のエラー) が発生した、1 分あたりの API サーバーリクエストの数。

メトリクス	説明
APIServer 合計リクエスト 5XX	HTTP 5XX レスポンスコード (サーバー側のエラー) が発生した、1 分あたりの API サーバーリクエストの数。
APIServer 合計リクエスト 429	HTTP 429 レスポンスコード (リクエストが多すぎます) が発生した、1 分あたりの API サーバーリクエストの数。
ストレージサイズ	ストレージデータベース (etcd) のサイズ。
スケジューラーの試行	「unschedulable」、「error」、「scheduled」の結果による、ポッドのスケジュールの試行回数。
保留中のポッド	キュータイプが「active」、「backoff」、「unschedulable」、「gated」の保留中のポッドの数。
API サーバーリクエストのレイテンシー	API サーバーリクエストのレイテンシー。
API サーバーの現在処理中のリクエスト	API サーバーの現在処理中のリクエスト。
ウェブフックリクエスト	1 分あたりのウェブフックリクエスト。
ウェブフックリクエストの拒否	拒否されたウェブフックリクエストの数。
ウェブフックリクエストのレイテンシー P99	外部サードパーティーのウェブフックリクエストの 99 パーセンタイルレイテンシー。

CloudWatch Log Insights

[CloudWatch Log Insights] セクションにはコントロールプレーンの監査ログに基づくさまざまなリストが表示されます。この機能を使用するにはAmazon EKS コントロールプレーンログをオンにする必要があります。これは[CloudWatch のコントロールプレーンログを表示] セクションから実行できます。

データを収集するのに十分な時間が経過したら、[すべてのクエリを実行] したり、一回に 1 つのリストについて [クエリを実行] を選択したりできます。クエリを実行するたびに、CloudWatch からの追加料金が発生します。セクションの上部に表示する結果の期間を選択してください。クエリをより高度に制御したい場合は[CloudWatch で表示] を選択できます。これにより、ニーズに合わせて CloudWatch でクエリを更新できます。

詳細については「Amazon CloudWatch Logs ユーザーガイド」の「CloudWatch Logs Insights を使用したログデータの分析」を参照してください。

CloudWatch でコントロールプレーンのログを表示する

[ログ記録を管理] を選択して、使用可能なログタイプを更新します。ログ記録を有効にした後、CloudWatch Logs にログが表示されるまでに数分かかります。十分な時間が経過したら、このセクションのリンクを [表示] を選択して、該当するログに移動します。

詳細については「the section called “コントロールプレーンのログ”」を参照してください。

クラスターのインサイト

[インサイトをアップグレード] の表では問題が表示されるほか、是正措置が推奨されます。これにより、新しい Kubernetes バージョンにアップグレードするための検証プロセスが加速されます。Amazon EKS はKubernetes バージョンのアップグレードに影響を及ぼす可能性のある問題のリストに照らしてクラスターを自動的にスキャンします。[インサイトをアップグレード] の表にはこのクラスターに対して Amazon EKS によって実行されるインサイトチェックと、関連付けられたステータスが一覧表示されます。

Amazon EKS はKubernetes プロジェクトの変更と、新しいバージョンに関連する Amazon EKS サービスの変更の評価に基づいて実行されるインサイトチェックのリストを維持し、定期的に更新します。Amazon EKS コンソールは各インサイトのステータスを自動的に更新します。これは最終更新時刻の列で確認できます。

詳細については「the section called “クラスターのインサイト”」を参照してください。

ノードのヘルスの問題

Amazon EKS ノードモニタリングエージェントはノードログを自動的に読み取り、ヘルスの問題を検出します。自動修復設定に関係なく、必要に応じて調査できるように、すべてのノードのヘルスの問題が報告されます。問題タイプが説明なしで一覧にされている場合はそのポップオーバー要素で説明を読み取ることができます。

ページを更新すると、解決された問題はリストから消えます。自動修復が有効になっている場合、アクションを起こさなくても解決されるヘルスの問題が一時的に表示されます。自動修復でサポートされていない問題の場合はタイプによっては手動アクションが必要になる場合があります。

ノードのヘルスの問題を報告するにはクラスターで Amazon EKS 自動モード を使用するか、ノードモニタリングエージェントのアドオンが必要です。詳細については、「the section called “ノードのヘルスと修復”」を参照してください。

EKS 機能

[機能] セクションには、クラスター内の EKS 機能リソースのステータスとヘルスが表示されます。クラスター内の機能とそのマネージド Kubernetes リソースの両方に関するヘルスとステータスの通知は、ここでモニタリングできます。ページを更新すると、解決された問題はリストから消えます。

詳細については、「the section called “機能の使用”」を参照してください。

コンテナネットワークのオブザーバビリティを使用して Kubernetes ワークロードのトラフィックをモニタリングする

Amazon EKS は、コンテナネットワーク環境に関するより深いインサイトを提供する拡張ネットワークのオブザーバビリティ機能を提供します。これらの機能は、AWS における Kubernetes ネットワーク環境をよりよく理解し、モニタリングし、トラブルシューティングするのに役立ちます。強化されたコンテナネットワークのオブザーバビリティでは、クラスタートラフィック、AZ 間フロー、AWS サービス全体にわたって、事前の異常検出をより効果的に行うために、ネットワーク関連のきめ細かなメトリクスを活用できます。これらのメトリクスを使用して、システムパフォーマンスを測定し、任意のオブザーバビリティスタックで基盤となるメトリクスを視覚化できます。

さらに、Amazon EKS は、より迅速な根本原因分析のために、正確なトラブルシューティングを加速および強化するネットワークモニタリングの視覚化を AWS コンソールで提供するようになりました。これらの視覚化機能を活用することで、再送信や再送信タイムアウトを引き起こしているトップトーカーやネットワークフローを特定し、インシデント中の死角を排除することもできます。

これらの機能は、Amazon CloudWatch Network Flow Monitor によって有効化されています。

ユースケース

ネットワークパフォーマンスを測定して異常を検出する

いくつかのチームは、システムのパフォーマンスを測定し、システムメトリクスを視覚化し、特定のしきい値を超えた場合にアラームを受け取れるオブザーバビリティスタックを標準化していま

す。EKS のコンテナネットワークのオブザーバビリティは、スクレイピング可能な主要なシステムメトリクスを公開することで、ポッドおよびワーカーノードレベルでシステムのネットワークパフォーマンスのオブザーバビリティを拡張し、これと整合します。

コンソールの視覚化を活用してより正確なトラブルシューティングを行う

モニタリングシステムからアラームが発生した場合は、問題が発生したクラスターとワークロードに焦点を絞り込む必要があります。これを支援するために、EKS コンソールの視覚化を活用すると、クラスターレベルで調査範囲を絞り込み、再送信、再送信タイムアウト、転送データ量の原因となっているネットワークフローの特定を高速化できます。

Amazon EKS 環境でトップトーカーを追跡する

多くのチームは、プラットフォームの基盤として EKS を実行し、EKS をアプリケーション環境におけるネットワークアクティビティの中心にしています。この機能のネットワークモニタリング機能を使用すると、クラスター内、AZ 間、さらに AWS 内の外部宛先 (DynamoDB と S3) や AWS クラウド外 (インターネットまたはオンプレミス) へのトラフィックのうち、(データ量で測定した場合に) 最も多くのトラフィックを発生させているワークロードを追跡できます。さらに、再送信、再送信タイムアウト、および転送データ量に基づいて、これらの各フローのパフォーマンスをモニタリングできます。

機能

1. パフォーマンスメトリクス - この機能を使用すると、EKS クラスター内で実行されている Network Flow Monitor (NFM) エージェントから、ポッドとワーカーノードのネットワーク関連のシステムメトリクスを直接スクレイピングできます。
2. サービスマップ - この機能では、クラスター内のワークロード間の相互通信を動的に視覚化し、通信しているポッド間のネットワークフローに関連する主要なメトリクス (再送信 - RT、再送信タイムアウト - RTO、転送データ量 - DT) を迅速に把握できます。
3. フローテーブル - このテーブルでは、AWS サービスビュー、クラスタービュー、外部ビューの 3 つの異なる観点から、クラスター内の Kubernetes ワークロード全体におけるトップトーカーをモニタリングできます。ビューごとに、送信元ポッドとその送信先の間における再送信、再送信タイムアウト、転送データ量を確認できます。
   - AWS サービスビュー: AWS サービス (DynamoDB および S3) へのトップトーカーを表示します
   - クラスタービュー: クラスター内のトップトーカーを表示します (東西方向)
   - 外部ビュー: AWS 外のクラスター外宛先へのトップトーカーを表示します

はじめに

開始するには、新規または既存のクラスターの EKS コンソールでコンテナネットワークオブザーバビリティを有効にします。これにより、Network Flow Monitor (NFM) の依存関係 (スコープとモニターリソース) の作成が自動化されます。さらに、Network Flow Monitor Agent アドオンをインストールする必要があります。または、 `AWS CLI`、EKS API (アドオン用)、NFM API、あるいは Infrastructure as Code (Terraform など) を使用して、これらの依存関係をインストールすることもできます。これらの依存関係を設定したら、NFM エージェントからポッドとワーカーノードのネットワークパフォーマンスメトリクスをスクレイピングするように、任意のモニタリングツールを設定できます。ワークロードのネットワークアクティビティとパフォーマンスを視覚化するには、クラスターのオブザーバビリティダッシュボードの [ネットワーク] タブから EKS コンソールに移動します。

EKS で Network Flow Monitor を使用すると、既存のオブザーバビリティワークフローとテクノロジースタックを維持しながら、EKS 環境のネットワークレイヤーの理解と最適化をさらに促進する一連の追加機能を活用できます。詳細については、こちらの Network Flow Monitor の料金を参照してください。

前提条件と重要な注意事項

1. 上記のように、EKS コンソールからコンテナネットワークオブザーバビリティを有効にすると、基盤となる NFM リソースの依存関係 (スコープとモニター) が自動的に作成され、NFM 用 EKS アドオンのインストールプロセスが案内されます。
2. Terraform などの Infrastructure as Code (IaC) を使用してこの機能を有効にする場合は、NFM スコープ、NFM モニター、NFM 用 EKS アドオンの依存関係を IaC で定義する必要があります。さらに、Pod Identity またはサービスアカウントの IAM ロール (IRSA) を使用して、関連するアクセス許可を EKS アドオンに付与する必要があります。
3. NFM エージェントの EKS アドオンでは、最小バージョン 1.1.0 を実行している必要があります。
4. Network Flow Monitor リソースをサポートするには、Terraform AWS Provider の v6.21.0 以降を使用する必要があります。

必要な IAM 許可

NFM エージェント用 EKS アドオン

`CloudWatchNetworkFlowMonitorAgentPublishPolicy` の AWS 管理ポリシーは、Pod Identity で使用できます。このポリシーには、NFM エージェントがテレメトリレポート (メトリクス) を Network Flow Monitor エンドポイントに送信するためのアクセス許可が含まれています。

```
{
  "Version" : "2012-10-17",
  "Statement" : [
    {
      "Effect" : "Allow",
      "Action" : [
        "networkflowmonitor:Publish"
      ],
      "Resource" : "*"
    }
  ]
}
```

EKS コンソールでのコンテナネットワークのオブザーバビリティ

機能を有効にし、コンソールでサービスマップとフローテーブルを視覚化するには、次のアクセス許可が必要です。

```
{
  "Version" : "2012-10-17",
  "Statement" : [
    {
      "Effect": "Allow",
      "Action": [
        "networkflowmonitor:ListScopes",
        "networkflowmonitor:ListMonitors",
        "networkflowmonitor:GetScope",
        "networkflowmonitor:GetMonitor",
        "networkflowmonitor:CreateScope",
        "networkflowmonitor:CreateMonitor",
        "networkflowmonitor:TagResource",
        "networkflowmonitor:StartQueryMonitorTopContributors",
        "networkflowmonitor:StopQueryMonitorTopContributors",
        "networkflowmonitor:GetQueryStatusMonitorTopContributors",
```

```
        "networkflowmonitor:GetQueryResultsMonitorTopContributors"
      ],
      "Resource": "*"
    }
  ]
}
```

AWS CLI、EKS API、NFM API の使用

```
#!/bin/bash

# Script to create required Network Flow Monitor resources
set -e

CLUSTER_NAME="my-eks-cluster"
CLUSTER_ARN="arn:aws:eks:{Region}:{Account}:cluster/{ClusterName}"
REGION="us-west-2"
AGENT_NAMESPACE="amazon-network-flow-monitor"

echo "Creating Network Flow Monitor resources..."

# Check if Network Flow Monitor agent is running in the cluster
echo "Checking for Network Flow Monitor agent in cluster..."
if kubectl get pods -n "$AGENT_NAMESPACE" --no-headers 2>/dev/null | grep -q "Running";
 then
    echo "Network Flow Monitor agent exists and is running in the cluster"
else
    echo "Network Flow Monitor agent not found. Installing as EKS addon..."
    aws eks create-addon \
        --cluster-name "$CLUSTER_NAME" \
        --addon-name "$AGENT_NAMESPACE" \
        --region "$REGION"
    echo "Network Flow Monitor addon installation initiated"
fi

# Get Account ID
ACCOUNT_ID=$(aws sts get-caller-identity --query Account --output text)

echo "Cluster ARN: $CLUSTER_ARN"
echo "Account ID: $ACCOUNT_ID"

# Check for existing scope
echo "Checking for existing Network Flow Monitor Scope..."
```

```
EXISTING_SCOPE=$(aws networkflowmonitor list-scopes --region $REGION --query
 'scopes[0].scopeArn' --output text 2>/dev/null || echo "None")

if [ "$EXISTING_SCOPE" != "None" ] && [ "$EXISTING_SCOPE" != "null" ]; then
    echo "Using existing scope: $EXISTING_SCOPE"
    SCOPE_ARN=$EXISTING_SCOPE
else
    echo "Creating new Network Flow Monitor Scope..."
    SCOPE_RESPONSE=$(aws networkflowmonitor create-scope \
        --targets "[{\"targetIdentifier\":{\"targetId\":{\"accountId\":
\"${ACCOUNT_ID}\"},\"targetType\":\"ACCOUNT\"},\"region\":\"${REGION}\"}]" \
        --region $REGION \
        --output json)

    SCOPE_ARN=$(echo $SCOPE_RESPONSE | jq -r '.scopeArn')
    echo "Scope created: $SCOPE_ARN"
fi

# Create Network Flow Monitor with EKS Cluster as local resource
echo "Creating Network Flow Monitor..."
MONITOR_RESPONSE=$(aws networkflowmonitor create-monitor \
    --monitor-name "${CLUSTER_NAME}-monitor" \
    --local-resources "type=AWS::EKS::Cluster,identifier=${CLUSTER_ARN}" \
    --scope-arn "$SCOPE_ARN" \
    --region $REGION \
    --output json)

MONITOR_ARN=$(echo $MONITOR_RESPONSE | jq -r '.monitorArn')

echo "Monitor created: $MONITOR_ARN"

echo "Network Flow Monitor setup complete!"
echo "Monitor ARN: $MONITOR_ARN"
echo "Scope ARN: $SCOPE_ARN"
echo "Local Resource: AWS::EKS::Cluster (${CLUSTER_ARN})"
```

Infrastructure as Code (IaC) の使用

Terraform

Terraform を使用して AWS クラウドインフラストラクチャを管理している場合は、次のリソース設定を含めることで、クラスターのコンテナネットワークオブザーバビリティを有効にできます。

NFM スコープ

```
data "aws_caller_identity" "current" {}

resource "aws_networkflowmonitor_scope" "example" {
  target {
    region = "us-east-1"
    target_identifier {
      target_type = "ACCOUNT"
      target_id {
        account_id = data.aws_caller_identity.current.account_id
      }
    }
  }

  tags = {
    Name = "example"
  }
}
```

NFM モニター

```
resource "aws_networkflowmonitor_monitor" "example" {
  monitor_name = "eks-cluster-name-monitor"
  scope_arn    = aws_networkflowmonitor_scope.example.scope_arn

  local_resource {
    type       = "AWS::EKS::Cluster"
    identifier = aws_eks_cluster.example.arn
  }

  remote_resource {
    type       = "AWS::Region"
    identifier = "us-east-1" # this must be the same region that the cluster is in
  }

  tags = {
    Name = "example"
  }
}
```

NFM 用 EKS アドオン

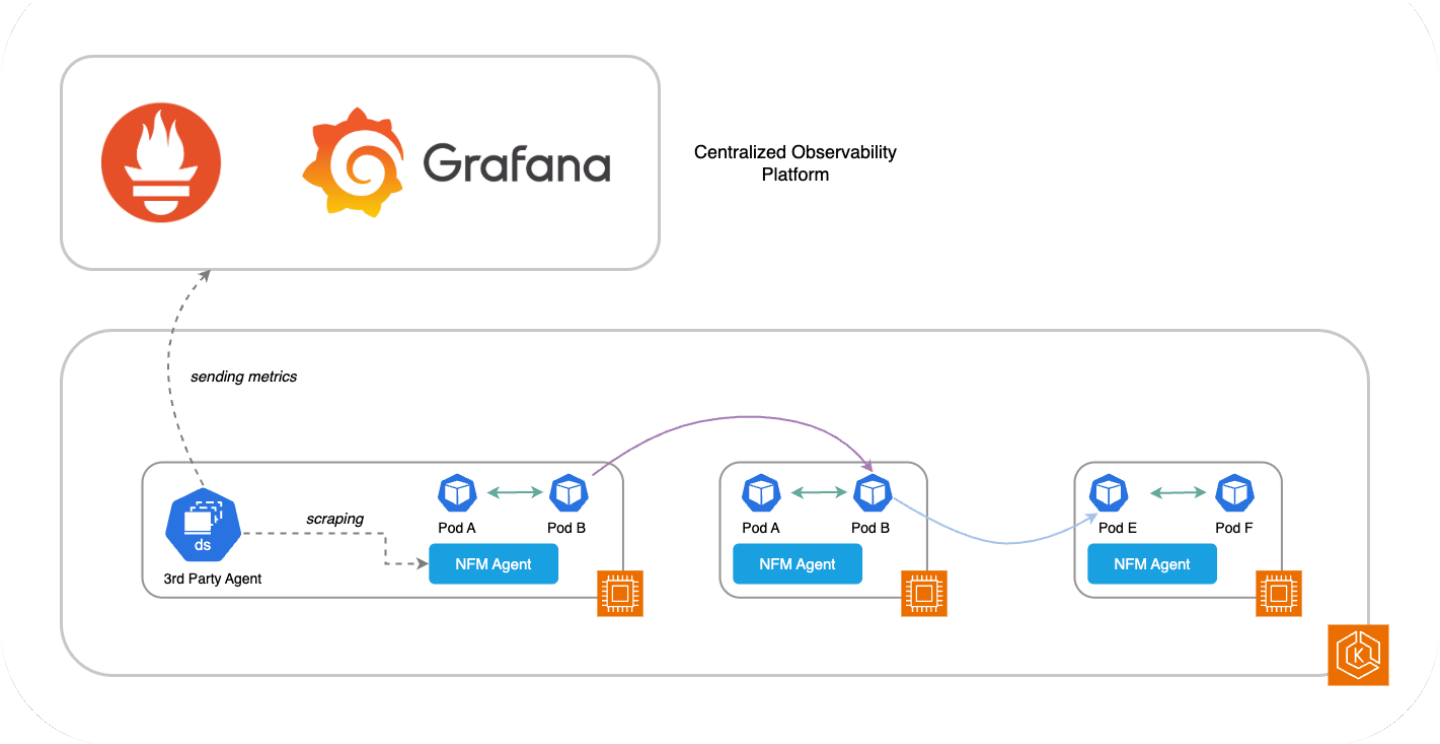
```
resource "aws_eks_addon" "example" {
  cluster_name                = aws_eks_cluster.example.name
  addon_name                  = "aws-network-flow-monitoring-agent"
}
```

動作の仕組み

パフォーマンスメトリクス

システムメトリクス

(Prometheus や Grafana などの) サードパーティ (3P) ツールを使用して EKS 環境を監視している場合、サポートされているシステムメトリクスを Network Flow Monitor エージェントから直接スクレイピングできます。これらのメトリクスをモニタリングスタックに送信することで、ポッドおよびワーカーノードレベルでシステムのネットワークパフォーマンスの測定を拡張できます。使用可能なメトリクスは、サポートされているシステムメトリクスの表に一覧表示されています。



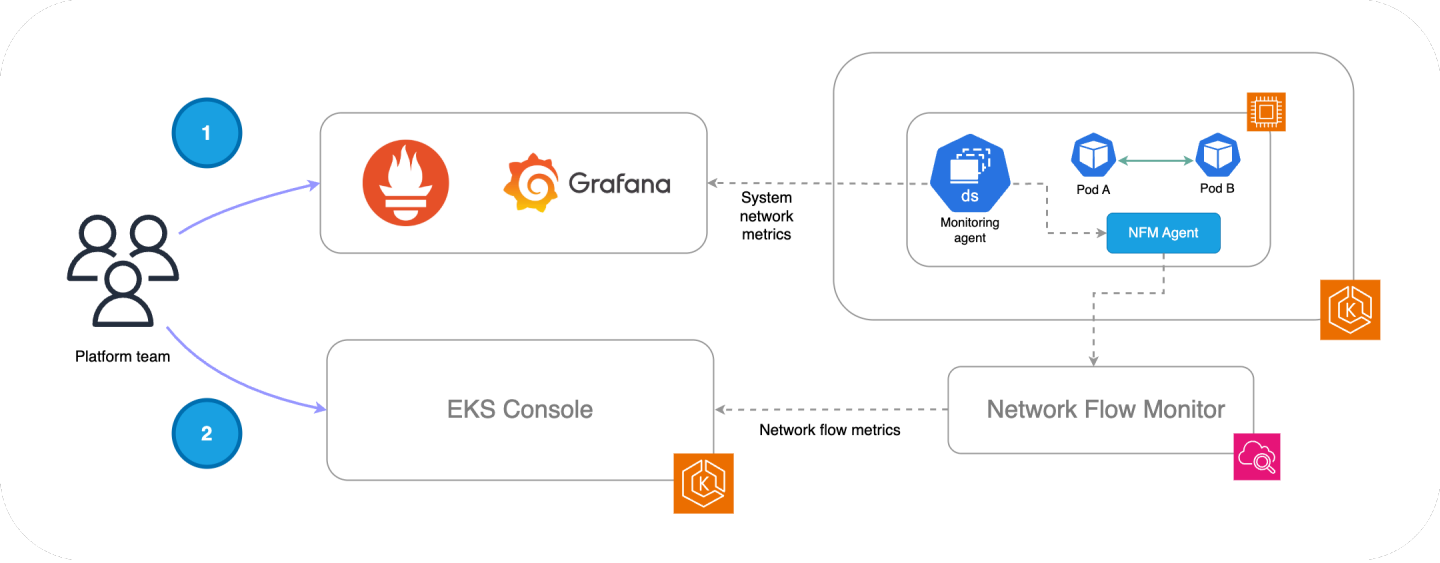
これらのメトリクスを有効にするには、インストールプロセス中に、設定変数を使用して次の環境変数を上書きします (https://aws.amazon.com/blogs/containers/amazon-eks-add-ons-advanced-configuration/ を参照)。

```
OPEN_METRICS:
    Enable or disable open metrics. Disabled if not supplied
    Type: String
    Values: [“on”, “off”]
OPEN_METRICS_ADDRESS:
    Listening IP address for open metrics endpoint. Defaults to 127.0.0.1 if not
 supplied
    Type: String
OPEN_METRICS_PORT:
    Listening port for open metrics endpoint. Defaults to 80 if not supplied
    Type: Integer
    Range: [0..65535]
```

フローレベルのメトリクス

さらに、Network Flow Monitor は、再送信、再送信タイムアウト、転送データ量といったフローレベルのメトリクスと共にネットワークフローデータをキャプチャします。このデータは Network Flow Monitor によって処理され、EKS コンソールで視覚化され、クラスターの環境内のトラフィックと、これらのフローレベルのメトリクスに基づいてパフォーマンスが示されます。

以下の図は、両方のタイプのメトリクス (システムレベルとフローレベル) を活用して、運用インテリジェンスを高めるワークフローを示しています。



1. プラットフォームチームは、モニタリングスタック内でシステムメトリクスを収集して視覚化できます。アラートを設定することで、NFM エージェントのシステムメトリクスを使用して、ポッドまたはワーカーノードに影響を与えるネットワークの異常や問題を検出できます。

2. 次のステップとして、プラットフォームチームは EKS コンソールのネイティブな視覚化を活用し、フローの表現とそれに関連するメトリクスに基づいて、調査範囲をさらに絞り込み、トラブルシューティングを高速化できます。

重要な注意点: NFM エージェントからのシステムメトリクスのスクレイピングと、NFM エージェントがフローレベルのメトリクスを NFM バックエンドにプッシュするプロセスは、互いに独立したプロセスです。

サポートされているシステムメトリクス

重要な注意点: システムメトリクスは OpenMetrics 形式でエクスポートされます。

メトリクス名	タイプ	ディメンション	説明
ingress_flow	ゲージ	instance_id、iface、ポッド、名前空間、ノード	Ingress TCP フローカウント (TcpPassiveOpens)
egress_flow	ゲージ	instance_id、iface、ポッド、名前空間、ノード	Egress TCP フローカウント (TcpActiveOpens)
ingress_packets	ゲージ	instance_id、iface、ポッド、名前空間、ノード	受信パケット数 (デルタ)
egress_packets	ゲージ	instance_id、iface、ポッド、名前空間、ノード	送信パケット数 (デルタ)
ingress_bytes	ゲージ	instance_id、iface、ポッド、名前空間、ノード	入力バイト数 (デルタ)
egress_bytes	ゲージ	instance_id、iface、ポッド、名前空間、ノード	出力バイト数 (デルタ)

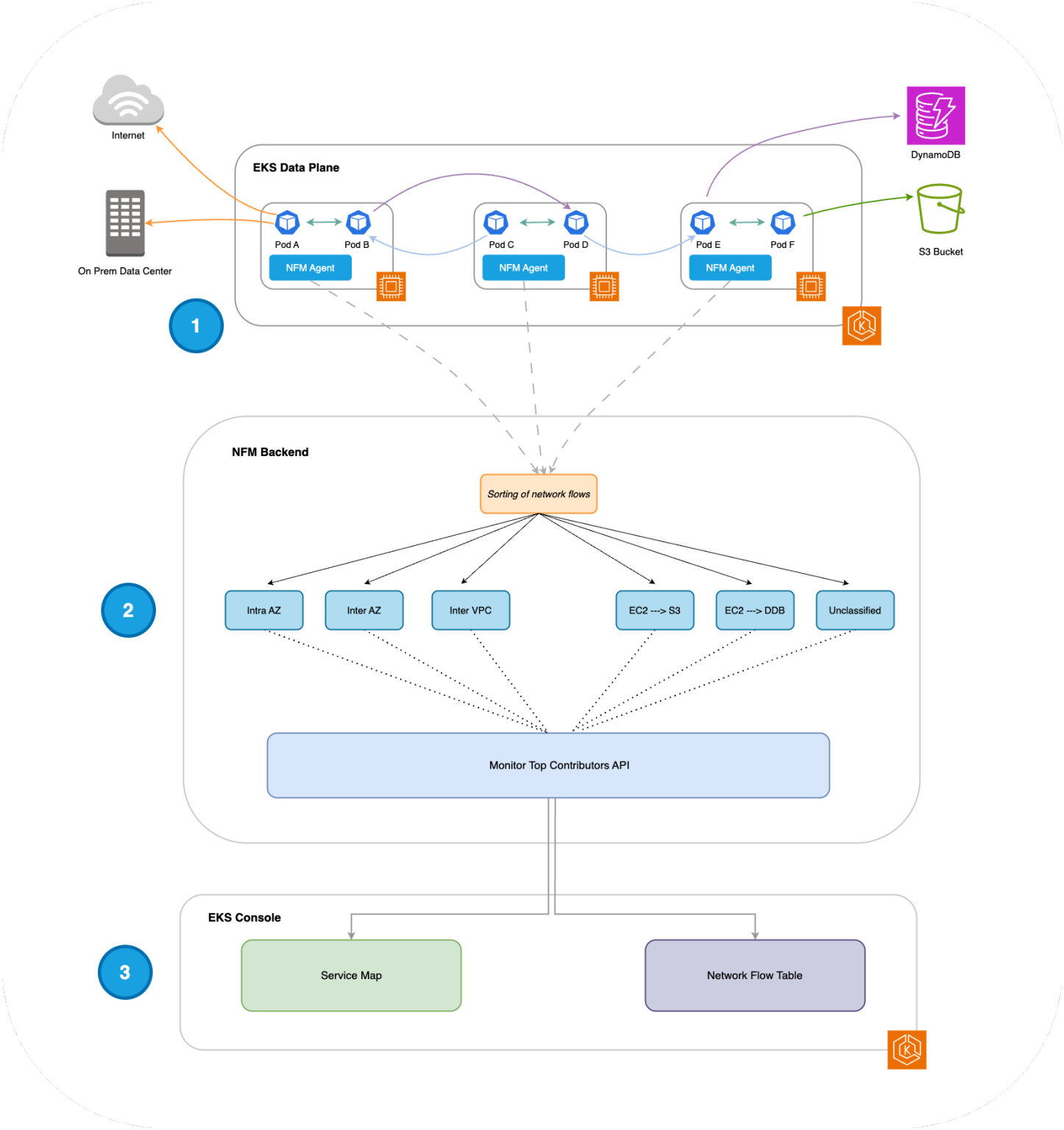
メトリクス名	タイプ	ディメンション	説明
bw_in_allowance_exceeded	ゲージ	instance_id、eni、ノード	インバウンド帯域幅の制限によりキューに入れられた/ドロップされたパケット
bw_out_allowance_exceeded	ゲージ	instance_id、eni、ノード	アウトバウンド帯域幅の制限によりキューに入れられた/ドロップされたパケット
pps_allowance_exceeded	ゲージ	instance_id、eni、ノード	双方向 PPS 制限によりキューに入れられた/ドロップされたパケット
conntrack_allowance_exceeded	ゲージ	instance_id、eni、ノード	接続追跡制限によりドロップされたパケット
linklocal_allowance_exceeded	ゲージ	instance_id、eni、ノード	ローカルプロキシサービスの PPS 制限によりドロップされたパケット

サポートされているフローレベルのメトリクス

メトリクス名	型	説明
TCP 再送信	Counter	送信中に失われた、または破損したパケットを送信者が再送信した回数。
TCP 再送信タイムアウト	Counter	送信中にパケットが失われたかどうかを判断するために、

メトリクス名	型	説明
		送信者が待機期間を開始した回数。
転送データ量 (バイト)	Counter	特定のフローの送信元と送信先の間で転送されるデータの量。

サービスマップとフローテーブル

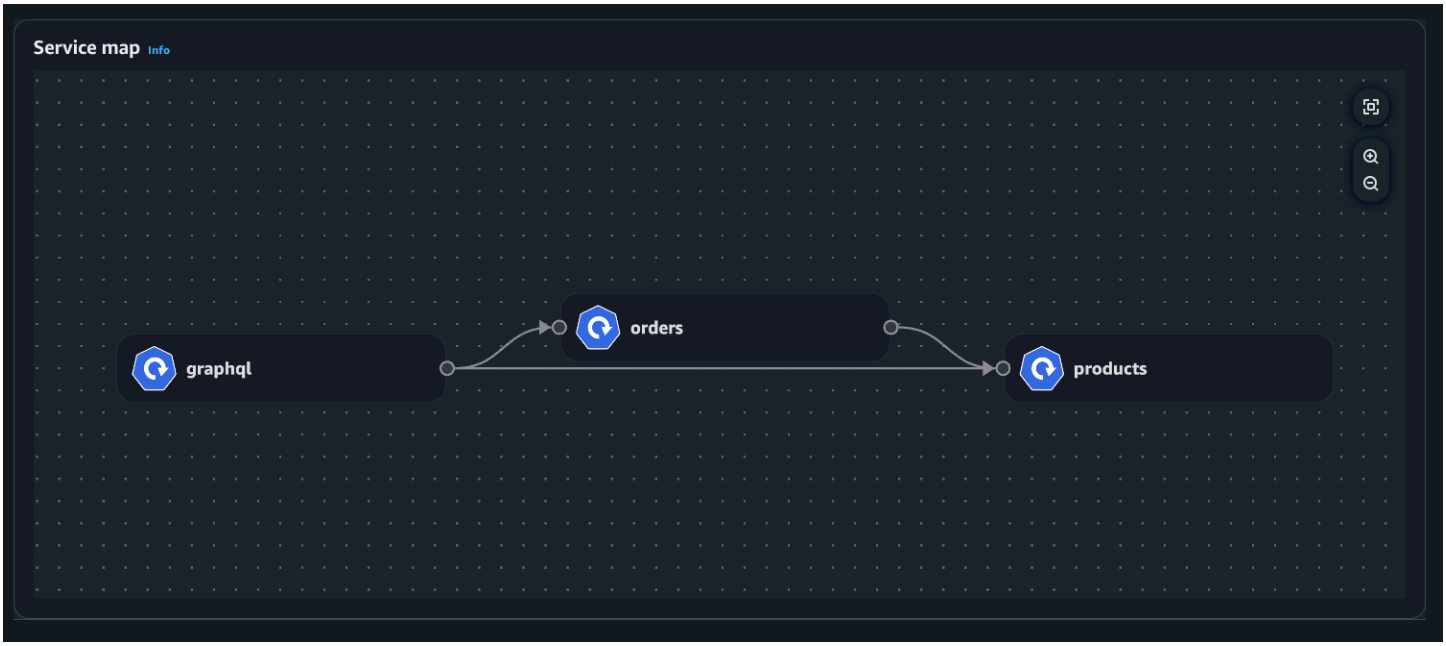


1. Network Flow Monitor エージェントは、インストールされると、すべてのワーカーノードで DaemonSet として実行され、30 秒ごとに、(転送データ量に基づいて) 上位 500 件のネットワークフローを収集します。
2. これらのネットワークフローは、AZ 内、AZ 間、EC2 → S3, EC2 → DynamoDB (DDB)、未分類のカテゴリに分類されます。各フローには、再送信、再送信タイムアウト、転送データ量 (バイト単位) の 3 つのメトリクスが関連付けられています。
   - AZ 内 - 同じ AZ 内のポッド間のネットワークフロー
   - AZ 間 - 異なる AZ 間のポッド間のネットワークフロー
   - EC2 → S3 - ポッドから S3 へのネットワークフロー
   - EC2 → DDB - ポッドから DDB へのネットワークフロー
   - 未分類 - ポッドからインターネットまたはオンプレミスへのネットワークフロー
3. Network Flow Monitor Top Contributors API からのネットワークフローは、EKS コンソールで次のエクスペリエンスを強化するために使用されます。
   - サービスマップ: クラスター内のネットワークフロー (AZ 内および AZ 間) の視覚化。
   - フローテーブル: クラスター内のネットワークフロー (AZ 内および AZ 間)、ポッドから AWS サービス (EC2 → S3 および EC2 → DDB) へのフロー、ポッドから外部宛先 (未分類) へのフローの表形式表示。

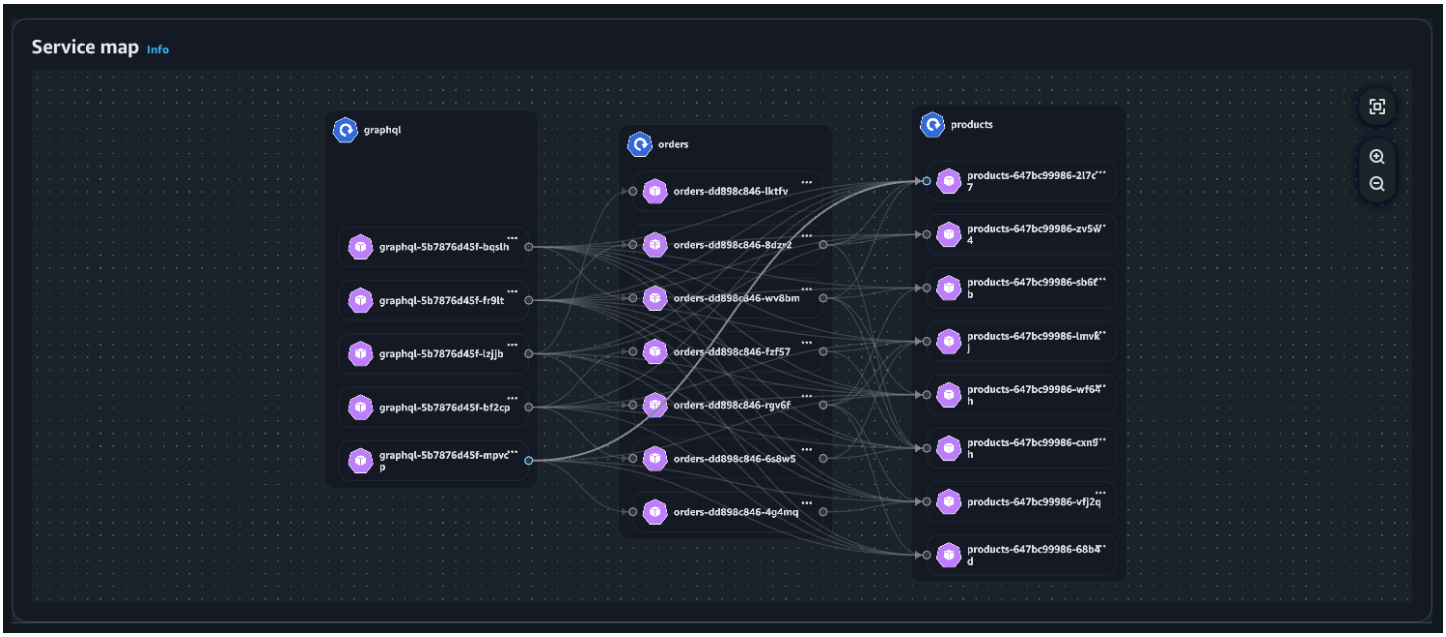
Top Contributors API からプルされたネットワークフローは、1 時間の時間範囲に限定されており、各カテゴリにつき最大 500 件のフローを含めることができます。サービスマップでは、1 時間の時間範囲において、AZ 内および AZ 間のフローカテゴリから最大 1000 件のフローを取得して表示できます。フローテーブルでは、2 時間の時間範囲において、5 つのネットワークフローカテゴリすべてから最大 2500 件のネットワークフローを取得して表示できます。

例: サービスマップ

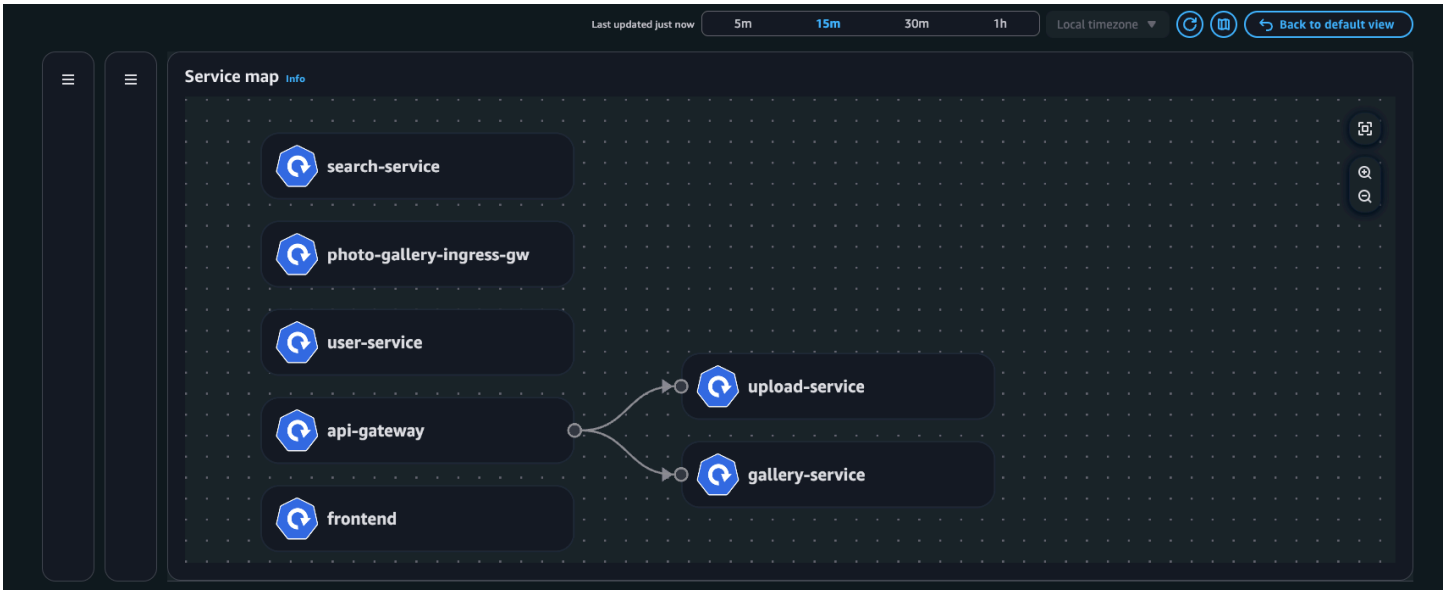
デプロイビュー



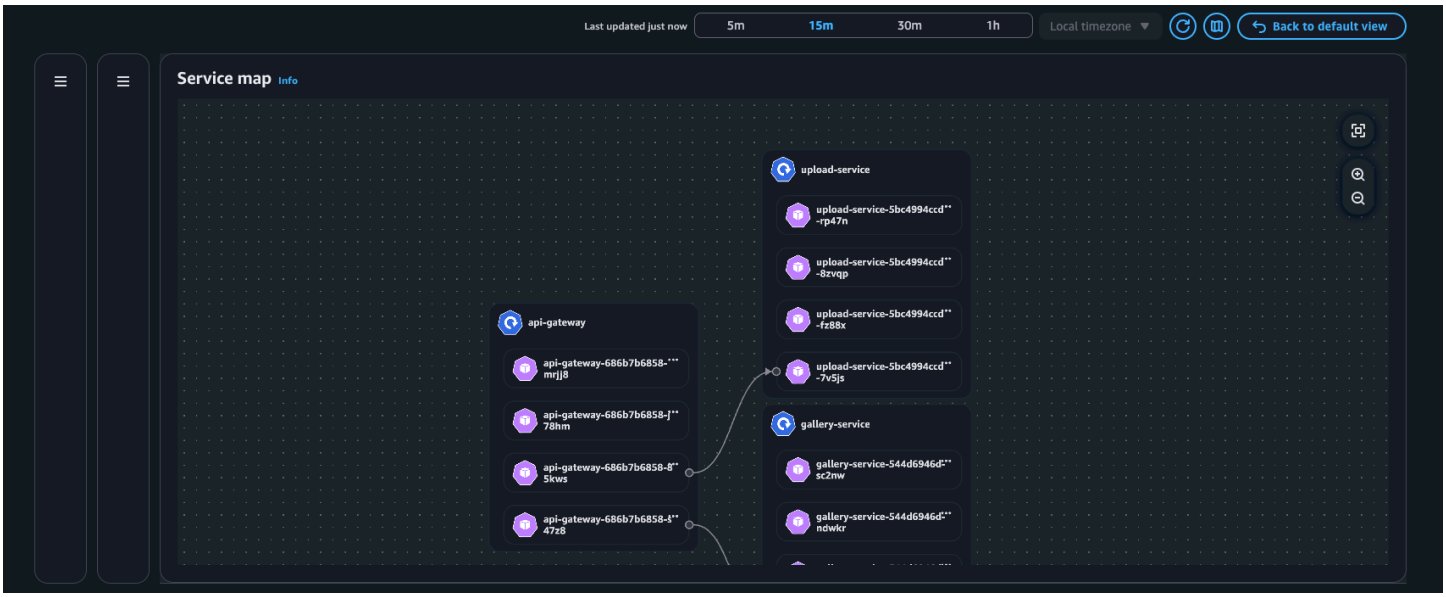
ポッドビュー



デプロイビュー

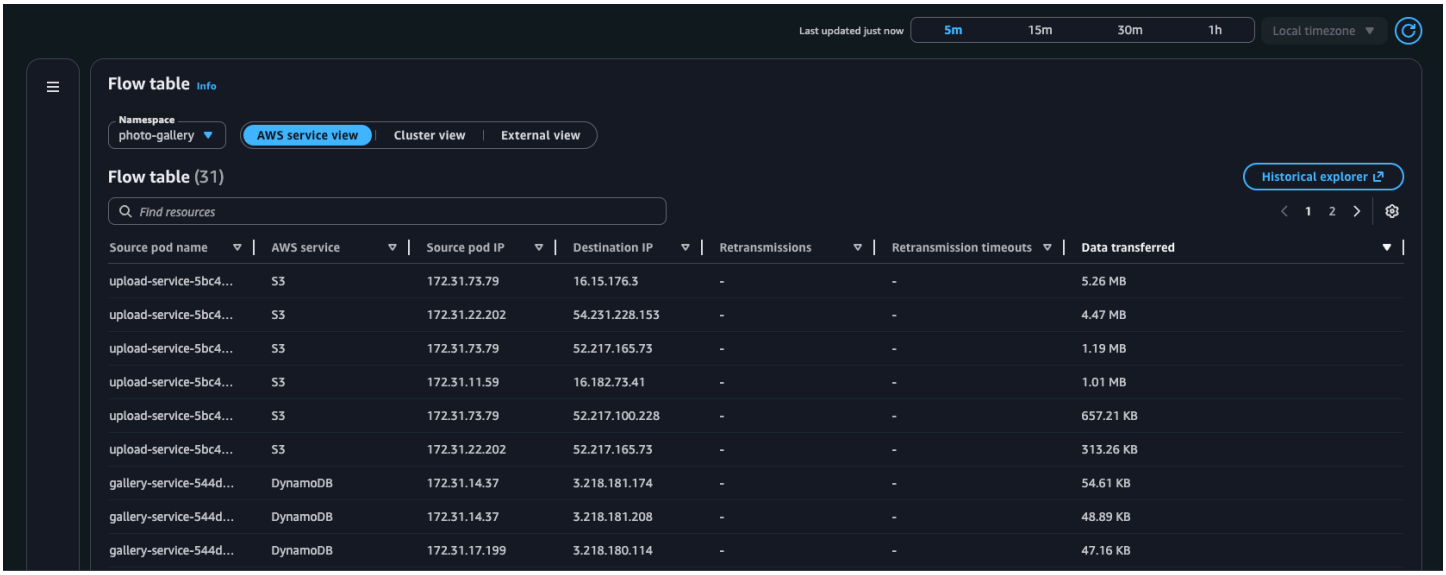


ポッドビュー

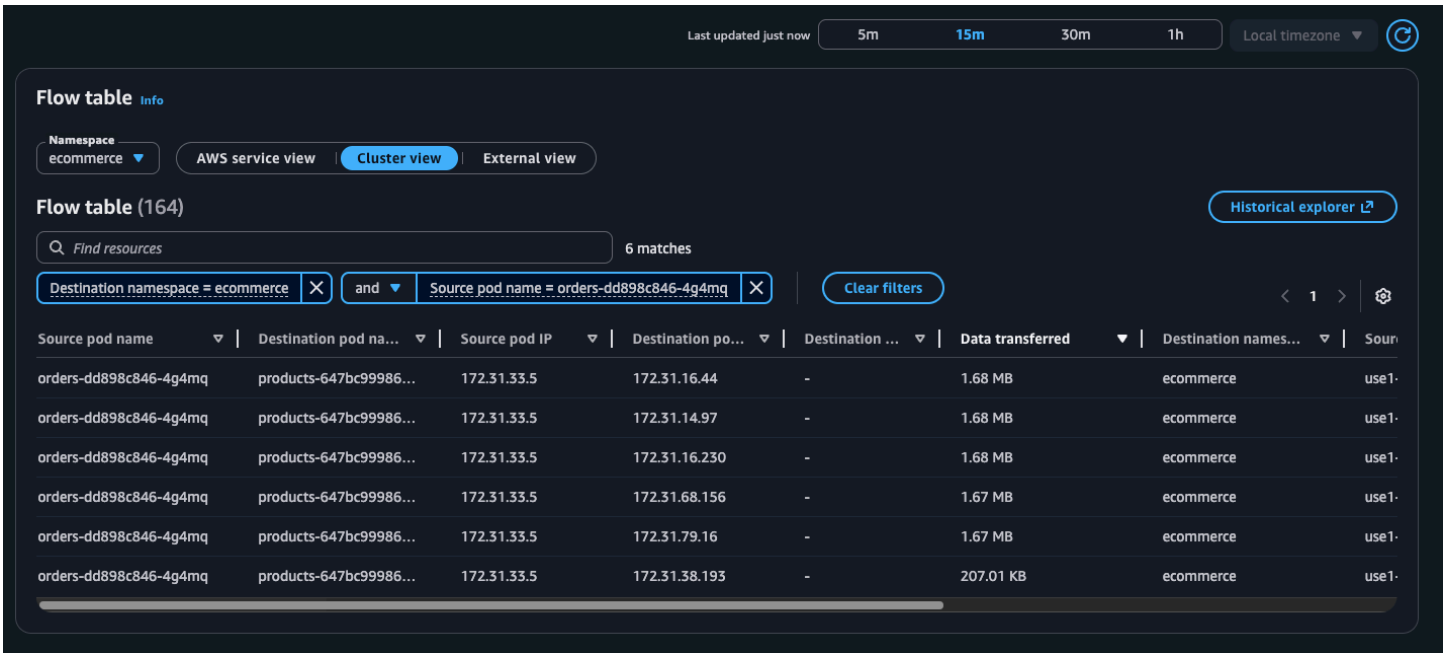


例: フローテーブル

AWS サービスビュー



クラスタービュー



考慮事項と制限事項

- EKS のコンテナネットワークオブザーバビリティは、Network Flow Monitor がサポートされているリージョンでのみ使用できます。
- サポートされているシステムメトリクスは OpenMetrics 形式で、Network Flow Monitor (NFM) エージェントから直接スクレイピングできます。

- Terraform などの Infrastructure as Code (IaC) を使用して EKS でコンテナネットワークオブザーバビリティを有効にするには、NFM スコープ、NFM モニター、NFM エージェントなどの依存関係を設定で定義して作成しておく必要があります。
- Network Flow Monitor は、1 分あたり最大約 500 万件のフローをサポートします。これは、Network Flow Monitor エージェントがインストールされた約 5,000 台の EC2 インスタンス (EKS ワーカーノード) に相当します。5,000 台を超えるインスタンスにエージェントをインストールすると、追加の容量が利用可能になるまで、モニタリングパフォーマンスに影響する可能性があります。
- NFM エージェントの EKS アドオンでは、最小バージョン 1.1.0 を実行している必要があります。
- Network Flow Monitor リソースをサポートするには、Terraform AWS Provider の v6.21.0 以降を使用する必要があります。
- ネットワークフローにポッドメタデータを付加するには、ポッドはホストのネットワーク名前空間ではなく、独自の独立したネットワーク名前空間で実行されている必要があります。

Prometheus を使用してクラスターのメトリクスをモニタリングする

Prometheus は、エンドポイントをスクレイピングするモニタリングおよび時系列データベースです。収集したデータをクエリ、集計、保存することができます。また、アラートやアラートの集計にも使用できます。このトピックでは、Prometheus をマネージドオプションまたはオープンソースオプションとして設定する方法について説明します。Amazon EKS コントロールプレーンメトリクスのモニタリングは一般的なユースケースです。

Amazon Managed Service for Prometheus は Prometheus 互換のモニタリングおよびアラートサービスで、コンテナ化されたアプリケーションやインフラストラクチャを大規模に監視することが容易になります。これは、メトリクスの取り込み、ストレージ、クエリ、アラートを自動的にスケーリングするフルマネージド型サービスです。また、AWS セキュリティサービスと統合して、データへの高速かつ安全なアクセスを可能にします。オープンソースの PromQL クエリ言語を使用して、メトリクスをクエリし、それらに関するアラートを作成できます。また、Amazon Managed Service for Prometheus のアラートマネージャーを使用して、重要なアラートのアラートルールを設定することもできます。その後、これらの重要なアラートを Amazon SNS トピックに通知として送信できます。

Amazon EKS で Prometheus を使用するにあたっては、数種類のオプションがあります。

- Amazon EKS クラスターを最初に作成するときに Prometheus メトリクスをオンにできます。また、既存のクラスター用に独自の Prometheus スクレイパーを作成することもできます。このトピックでは、これらのオプションの両方を説明します。
- Helm を使用して Prometheus をデプロイできます。詳細については、「the section called “Helm を使用してデプロイする”」を参照してください。
- コントロールプレーンの未加工メトリクスを Prometheus 形式で表示できます。詳細については、「the section called “コントロールプレーン”」を参照してください。

ステップ 1: Prometheus メトリクスを有効にする

> **Important**
>
> Amazon Managed Service for Prometheus リソースはクラスターのライフサイクル外にあるため、クラスターとは別途維持する必要があります。クラスターを削除するときは、対象コストを抑えるために、該当するスクレイパーも削除してください。詳細については、Amazon Managed Service for Prometheus ユーザーガイドの「スクレイパーの検出と作成」を参照してください。

Prometheus は、スクレイピングと呼ばれるプルベースのモデルを通じて、クラスターからメトリクスを検出して収集します。スクレイパーは、クラスターインフラストラクチャーとコンテナー化されたアプリケーションからデータを収集するように設定されています。Prometheus メトリクスを送信するオプションをオンにすると、Amazon Managed Service for Prometheus は完全マネージド型のエージェントレススクレイパーを提供します。

クラスターをまだ作成していない場合は、クラスターを初めて作成するときにこのオプションをオンにしてメトリクスを Prometheus に送信できます。Amazon EKS コンソールでは、このオプションはクラスターの新規作成の [オブザーバビリティの設定] ステップにあります。詳細については、「the section called “クラスターを作成する”」を参照してください。

既存のクラスターが既にある場合は、独自の Prometheus スクレイパーを作成できます。Amazon EKS コンソールでこれを行うには、クラスターの [オブザーバビリティ] タブに移動し、[スクレイパーの追加] ボタンを選択します。AWS API または AWS CLI でこれを行う場合は、「Amazon Managed Service for Prometheus ユーザーガイド」の「Create a scraper」を参照してください。

Amazon EKS コンソールでスクレイパーを作成する場合、次のオプションを使用できます。

スクレイパーエイリアスの名前

(オプション) スクレイパーの一意のエイリアスの入力。

送信先

Amazon Managed Service for Prometheus ワークスペースを選択します。ワークスペースは、Prometheus メトリクスの保存とクエリに使用される専用の論理スペースです。このワークスペースでは、そのワークスペースにアクセスできるアカウントの Prometheus メトリクスを表示できます。[新しいワークスペースを作成する] オプションでは、指定した [ワークスペースエイリアス] を使用してユーザーに代わってワークスペースを作成するよう Amazon EKS に指示します。[既存のワークスペースを選択] オプションでは、ドロップダウンリストから既存のワークスペースを選択できます。ワークスペースの詳細については、「Amazon Managed Service for Prometheus ユーザーガイド」の「ワークスペースの管理」を参照してください。

サービスアクセス

このセクションでは、Prometheus メトリクスを送信する際に付与するアクセス許可の概要を説明します。

- Amazon Managed Service for Prometheus に、スクレイピングされた Amazon EKS クラスターを記述できるようにします
- Amazon マネージド Prometheus ワークスペースへのリモート書き込みを許可する

`AmazonManagedScraperRole` がすでに存在する場合、スクレイパーはそれを使用します。`AmazonManagedScraperRole` リンクを選択すると、[アクセス許可の詳細] が表示されます。`AmazonManagedScraperRole` がまだ存在しない場合は、[アクセス許可の詳細を表示] リンクを選択すると、Prometheus メトリクスを送信して付与している特定のアクセス許可を確認できます。

サブネット

必要に応じてスクレイパーが継承するサブネットを変更します。グレー表示するサブネットオプションを追加する必要がある場合は、クラスターの作成でネットワークを指定する手順に戻ってください。

スクレイパー設定

必要に応じて YAML 形式でスクレイパー設定を変更します。これを行うには、フォームを使用するか、代替の YAML ファイルをアップロードします。詳細については、「Amazon Managed Service for Prometheus ユーザーガイド」の「スクレイパー設定」を参照してください。

Amazon Managed Service for Prometheus は、AWS マネージドコレクターとしてクラスターと一緒に作成されるエージェントレススクレイパーを指します。AWS マネージドコレクターの詳細については、「Amazon Managed Service for Prometheus ユーザーガイド」の「AWS マネージドコレクターによるメトリクスの取り込み」を参照してください。

> **Important**
>
> - AWS CLI または AWS API を使用して Prometheus スクレイパーを作成する場合は、その設定を調整して、スクレイパーにクラスター内アクセス許可を付与する必要があります。詳細については、「Amazon Managed Service for Prometheus ユーザーガイド」の「Amazon EKS クラスターの設定」を参照してください。
> - アクセスエントリの代わりに `aws-auth ConfigMap` を使用する Prometheus スクレイパーが 2024 年 11 月 11 日よりも前に作成されている場合は、Amazon EKS クラスターコントロールプレーンから追加のメトリクスにアクセスするように更新する必要があります。更新された設定については、「Amazon Managed Service for Prometheus ユーザーガイド」の「Manually configuring Amazon EKS for scraper access」を参照してください。

ステップ 2: Prometheus メトリクスを使用する

クラスターで Prometheus メトリクスをオンにした後の使用方法の詳細については、「Amazon Managed Service for Prometheus ユーザーガイド」を参照してください。

ステップ 3: Prometheus スクレイパーを管理する

スクレイパーを管理するには、Amazon EKS コンソールで [オブザーバビリティ] タブを選択します。表には、スクレイパー ID、エイリアス、ステータス、作成日などの情報を含む、クラスターのスクレイパーのリストが表示されます。スクレイパーの追加、編集、削除、または現在のスクレイパーに関する詳細情報の表示を行うことができます。

スクレイパーの詳細を表示するには、スクレイパー ID リンクを選択します。例えば、ARN、環境、ワークスペース ID、IAM ロール、設定、ネットワーク情報を表示できます。スクレイパー ID は、`DescribeScraper`、`UpdateScraper`、`DeleteScraper` などの Amazon Managed Service for Prometheus API オペレーションへの入力として使用できます。Prometheus API の使用に関する詳細については、「Amazon Managed Service for Prometheus API リファレンス」を参照してください。

Helm を使用して Prometheus をデプロイする

Amazon Managed Service for Prometheus の代わりに、Helm を使用してクラスターに Prometheus をデプロイできます。既に Helm がインストールされている場合は、`helm version` コマンドでバージョンを確認できます。Helm は Kubernetes クラスターのパッケージマネージャーです。Helm の詳細およびインストール方法については、「the section called “Helm を使用してアプリケーションをデプロイする”」を参照してください。

Amazon EKS クラスター用に Helm を設定した後、次の手順で、それを使用して Prometheus をデプロイできます。

1. Prometheus 名前空間を作成します。

```
kubectl create namespace prometheus
```

2. `prometheus-community` グラフリポジトリの追加

```
helm repo add prometheus-community https://prometheus-community.github.io/helm-charts
```

3. Prometheus をデプロイします。

```
helm upgrade -i prometheus prometheus-community/prometheus \
    --namespace prometheus \
    --set alertmanager.persistence.storageClass="gp2" \
    --set server.persistentVolume.storageClass="gp2"
```

> **Note**
>
> このコマンドの実行時に `Error: failed to download "stable/prometheus" (hint: running helm repo update may help)` エラーが発生した場合、`helm repo update prometheus-community` を実行し、ステップ 3 のコマンドをもう一度実行してみてください。

`Error: rendered manifests contain a resource that already exists` エラーが発生した場合、`helm uninstall` *`your-release-name`* `-n` *`namespace`* を実行し、ステップ 3 のコマンドをもう一度実行してみてください。

4. `prometheus` 名前空間のすべての Pod が `READY` 状態になっていることを確認します。

```
kubectl get pods -n prometheus
```

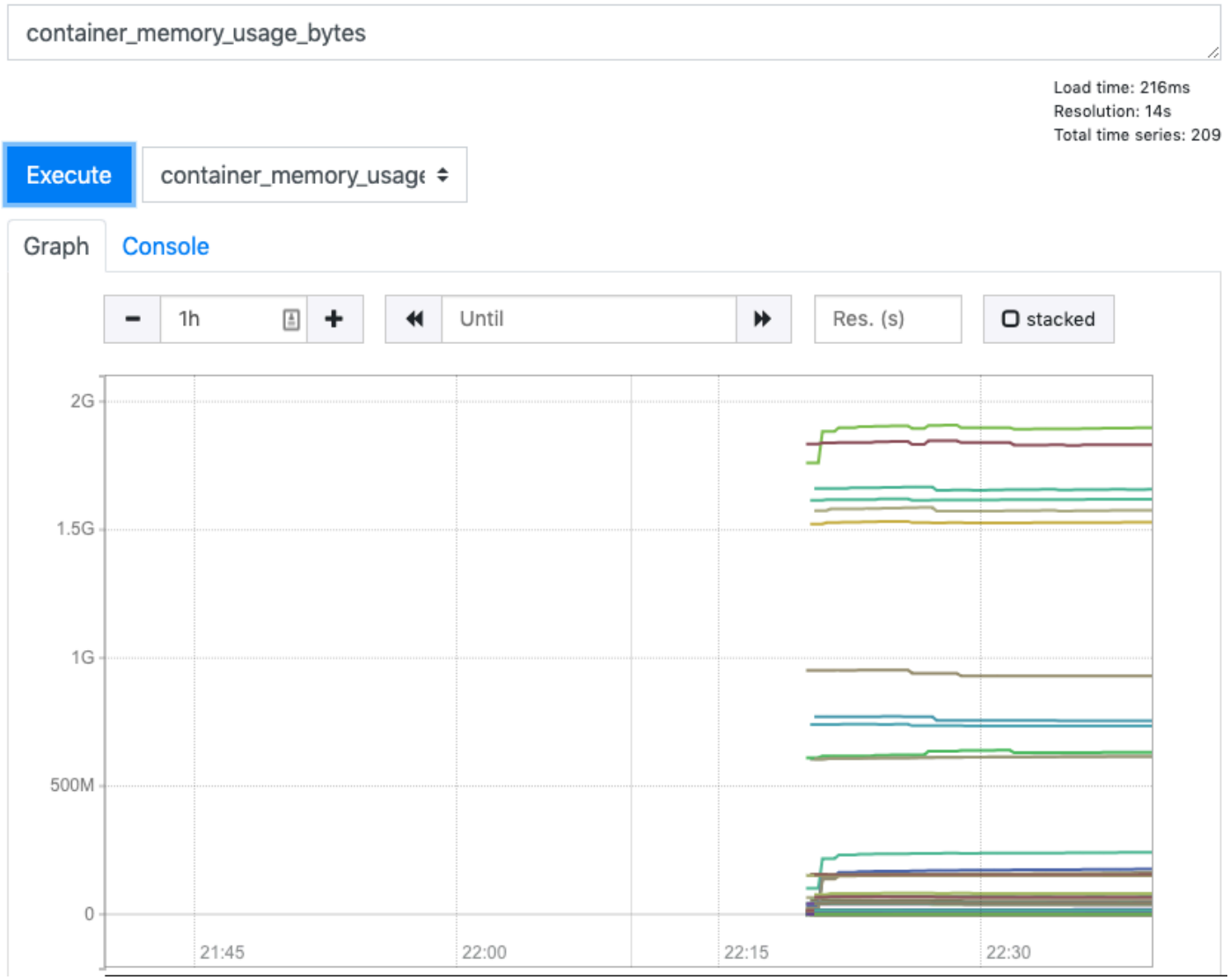
出力例は次のとおりです。

```
NAME                                             READY   STATUS    RESTARTS   AGE
prometheus-alertmanager-59b4c8c744-r7bgp         1/2     Running   0          48s
prometheus-kube-state-metrics-7cfd87cf99-jkz2f   1/1     Running   0          48s
prometheus-node-exporter-jcjqz                   1/1     Running   0          48s
prometheus-node-exporter-jxv2h                   1/1     Running   0          48s
prometheus-node-exporter-vbdks                   1/1     Running   0          48s
prometheus-pushgateway-76c444b68c-82tnw          1/1     Running   0          48s
prometheus-server-775957f748-mmht9               1/2     Running   0          48s
```

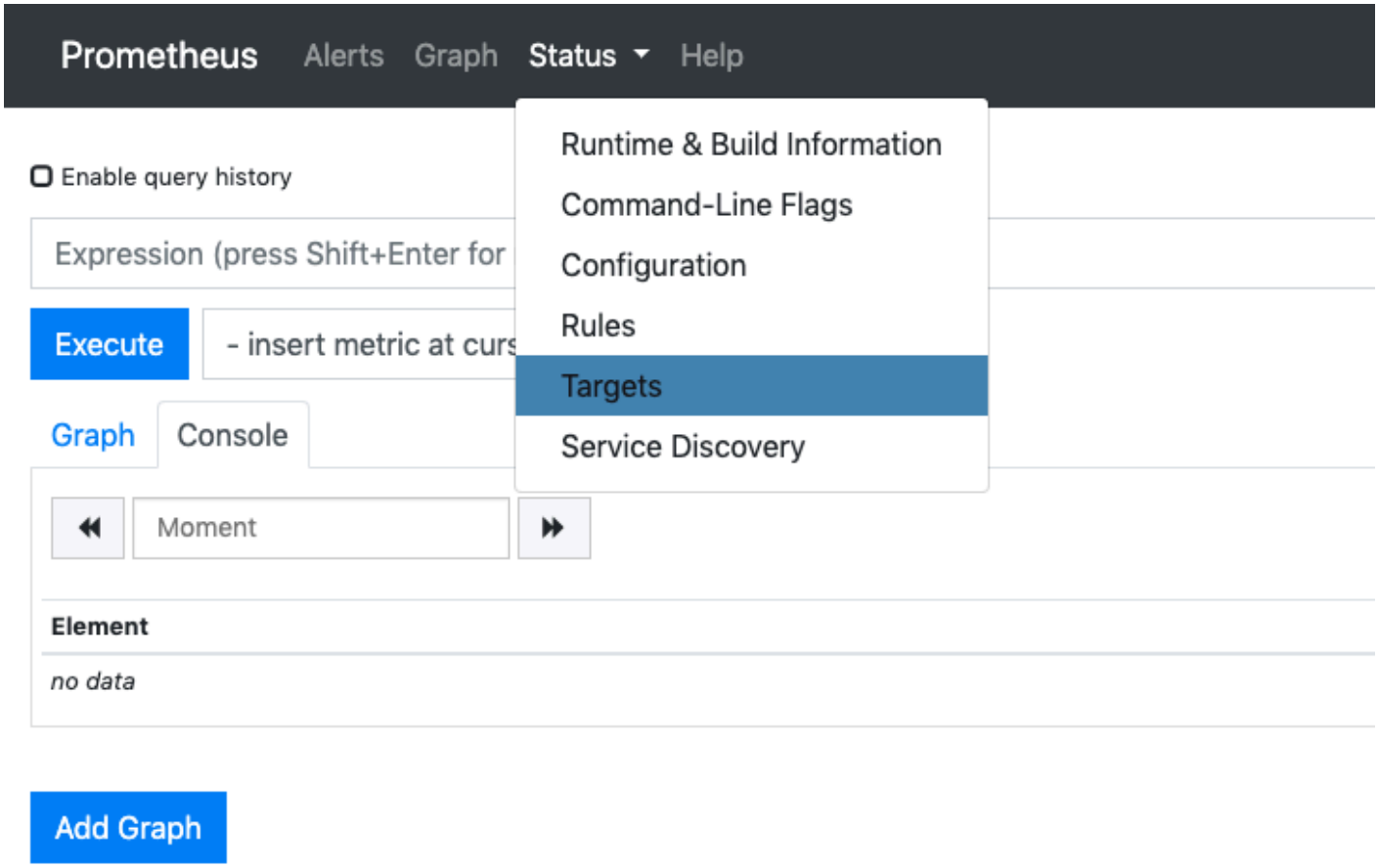
5. kubectl を使用して、Prometheus コンソールをローカルマシンにポート転送します。

```
kubectl --namespace=prometheus port-forward deploy/prometheus-server 9090
```

6. ウェブブラウザで http://localhost:9090 にアクセスして、Prometheus コンソールを表示します。
7. [- カーソルの位置のメトリクスを挿入] メニューからメトリクスを選択し、[実行] を選択します。[グラフ] タブを選択して、一定期間のメトリクスを表示します。次の図では、一定期間の container_memory_usage_bytes を示します。



8. 上部のナビゲーションバーから、[ステータス]、[ターゲット] の順に選択します。



サービス検出を使用して Prometheus に接続されているすべての Kubernetes エンドポイントが表示されます。

コントロールプレーンの未加工メトリクスを Prometheus 形式で取得する

Kubernetes コントロールプレーンは、Prometheus 形式で表される多数のメトリクスを公開します。これらのメトリクスはモニタリングおよび分析に役立ちます。これらはメトリクスエンドポイントを通じて内部的に公開され、Prometheus を完全にデプロイしなくてもアクセスできます。ただし、Prometheus をデプロイすることで、時間の経過に伴うメトリクスの分析がより簡単になります。

未加工メトリクス出力を表示するには、`endpoint`を置き換えて次のコマンドを実行してください。

```
kubectl get --raw endpoint
```

このコマンドを使用すると、任意のエンドポイントパスを渡して未加工のレスポンスを返すことができます。出力にはさまざまなメトリクスが行ごとにリストされ、各行にはメトリクス名、タグ、および値が含まれます。

```
metric_name{tag="value"[,...]} value
```

API サーバーからメトリクスを取得する

一般的な API サーバーエンドポイントはAmazon EKS コントロールプレーンで公開されます。このエンドポイントは主に特定のメトリクスを調べる際に役立ちます。

```
kubectl get --raw /metrics
```

出力例は次のとおりです。

```
[...]
# HELP rest_client_requests_total Number of HTTP requests, partitioned by status code,
 method, and host.
# TYPE rest_client_requests_total counter
rest_client_requests_total{code="200",host="127.0.0.1:21362",method="POST"} 4994
rest_client_requests_total{code="200",host="127.0.0.1:443",method="DELETE"} 1
rest_client_requests_total{code="200",host="127.0.0.1:443",method="GET"} 1.326086e+06
rest_client_requests_total{code="200",host="127.0.0.1:443",method="PUT"} 862173
rest_client_requests_total{code="404",host="127.0.0.1:443",method="GET"} 2
rest_client_requests_total{code="409",host="127.0.0.1:443",method="POST"} 3
rest_client_requests_total{code="409",host="127.0.0.1:443",method="PUT"} 8
# HELP ssh_tunnel_open_count Counter of ssh tunnel total open attempts
# TYPE ssh_tunnel_open_count counter
ssh_tunnel_open_count 0
# HELP ssh_tunnel_open_fail_count Counter of ssh tunnel failed open attempts
# TYPE ssh_tunnel_open_fail_count counter
ssh_tunnel_open_fail_count 0
```

この未加工出力はAPI サーバーが公開する内容をそのまま返します。

metrics.eks.amazonaws.com を使用してコントロールプレーンのメトリクスを取得する

Kubernete バージョン 1.28 以降のクラスターの場合、Amazon EKS は API グループ metrics.eks.amazonaws.com のメトリクスも公開します。これらのメトリクスにはkube-

scheduler や kube-controller-manager などのコントロールプレーンコンポーネントが含まれます。

> **Note**
>
> クラスターに新しい APIService リソース v1.metrics.eks.amazonaws.com の作成をブロックするウェブフック設定がある場合、メトリクスエンドポイント機能が使用できない可能性があります。v1.metrics.eks.amazonaws.com キーワードを検索することで、kube-apiserver 監査ログで確認することができます。

kube-scheduler メトリクスを取得する

kube-scheduler メトリクスを取得するには次のコマンドを使用します。

```
kubectl get --raw "/apis/metrics.eks.amazonaws.com/v1/ksh/container/metrics"
```

出力例は次のとおりです。

```
# TYPE scheduler_pending_pods gauge
scheduler_pending_pods{queue="active"} 0
scheduler_pending_pods{queue="backoff"} 0
scheduler_pending_pods{queue="gated"} 0
scheduler_pending_pods{queue="unschedulable"} 18
# HELP scheduler_pod_scheduling_attempts [STABLE] Number of attempts to successfully
 schedule a pod.
# TYPE scheduler_pod_scheduling_attempts histogram
scheduler_pod_scheduling_attempts_bucket{le="1"} 79
scheduler_pod_scheduling_attempts_bucket{le="2"} 79
scheduler_pod_scheduling_attempts_bucket{le="4"} 79
scheduler_pod_scheduling_attempts_bucket{le="8"} 79
scheduler_pod_scheduling_attempts_bucket{le="16"} 79
scheduler_pod_scheduling_attempts_bucket{le="+Inf"} 81
[...]
```

大量のデータが返される可能性があるため、kube_pod_resource_request メトリクスと kube_pod_resource_limit メトリクスを取得するための追加の kube-scheduler メトリクスエンドポイントがあります。これらのメトリクスを取得するには次のコマンドを使用します。

```
kubectl get --raw "/apis/metrics.eks.amazonaws.com/v1/ksh/container/resourcemetrics"
```

出力例は次のとおりです。

```
# HELP kube_pod_resource_limit [STABLE] Resources limit for workloads on the cluster,
 broken down by pod. This shows the resource usage the scheduler and kubelet expect per
 pod for resources along with the unit for the resource if any.
# TYPE kube_pod_resource_limit gauge
kube_pod_resource_limit{namespace="kube-system",node="ip-192-168-87-3.us-
west-2.compute.internal",pod="coredns-7bf648ff5d-
dj4ss",priority="2000000000",resource="memory",scheduler="default-
scheduler",unit="bytes"} 1.7825792e+08
# HELP kube_pod_resource_request [STABLE] Resources requested by workloads on the
 cluster, broken down by pod. This shows the resource usage the scheduler and kubelet
 expect per pod for resources along with the unit for the resource if any.
# TYPE kube_pod_resource_request gauge
kube_pod_resource_request{namespace="kube-system",node="ip-192-168-87-3.us-
west-2.compute.internal",pod="aws-node-
x7znh",priority="2000001000",resource="cpu",scheduler="default-scheduler",unit="cores"}
 0.05
kube_pod_resource_request{namespace="kube-system",node="ip-192-168-87-3.us-
west-2.compute.internal",pod="coredns-7bf648ff5d-
dj4ss",priority="2000000000",resource="cpu",scheduler="default-scheduler",unit="cores"}
 0.1
[...]
```

kube-controller-manager メトリクスを取得する

kube-controller-manager メトリクスを取得するには次のコマンドを使用します。

```
kubectl get --raw "/apis/metrics.eks.amazonaws.com/v1/kcm/container/metrics"
```

出力例は次のとおりです。

```
[...]
workqueue_work_duration_seconds_sum{name="pvprotection"} 0
workqueue_work_duration_seconds_count{name="pvprotection"} 0
workqueue_work_duration_seconds_bucket{name="replicaset",le="1e-08"} 0
workqueue_work_duration_seconds_bucket{name="replicaset",le="1e-07"} 0
workqueue_work_duration_seconds_bucket{name="replicaset",le="1e-06"} 0
workqueue_work_duration_seconds_bucket{name="replicaset",le="9.999999999999999e-06"} 0
workqueue_work_duration_seconds_bucket{name="replicaset",le="9.999999999999999e-05"} 19
workqueue_work_duration_seconds_bucket{name="replicaset",le="0.001"} 109
workqueue_work_duration_seconds_bucket{name="replicaset",le="0.01"} 139
workqueue_work_duration_seconds_bucket{name="replicaset",le="0.1"} 181
```

```
workqueue_work_duration_seconds_bucket{name="replicaset",le="1"} 191
workqueue_work_duration_seconds_bucket{name="replicaset",le="10"} 191
workqueue_work_duration_seconds_bucket{name="replicaset",le="+Inf"} 191
workqueue_work_duration_seconds_sum{name="replicaset"} 4.265655885000002
[...]
```

スケジューラーとコントローラーマネージャーのメトリクスを理解します

Prometheus スタイルスクレイピングに使用できるスケジューラとコントローラーマネージャーのメトリクスを次の表に示します。これらのメトリクスの詳細については、Kubernetes ドキュメントの「Kubernetes Metrics Reference」を参照してください。

メトリクス	コントロールプレーンのコンポーネント	説明
scheduler_pending_pods	スケジューラ	実行のためにノードにスケジュールされるのを待っているポッドの数です。
scheduler_schedule_attempts_total	スケジューラ	ポッドのスケジュールの試行回数です。
scheduler_preemption_attempts_total	スケジューラ	優先度の低いポッドを エビクションさせて、優先度の高いポッドをスケジュールするためにスケジューラーによって実行された試行回数です。
scheduler_preemption_victims	スケジューラ	優先順位の高いポッドのスペースを確保するために エビクション対象として選択されたポッドの数です。
scheduler_pod_scheduling_attempts	スケジューラ	ポッドの正常なスケジュールの試行回数です。
scheduler_scheduling_attempt_duration_seconds	スケジューラ	リソースの可用性やスケジューリングルールなどのさまざまな要因に基づいて、ス

メトリクス	コントロールプレーンのコンポーネント	説明
		ケジューラーがポッドを実行するのに適した場所をどの程度迅速に見つけることができるかを示します。
scheduler_pod_scheduling_sli_duration_seconds	スケジューラ	ポッドがスケジューリングキューに入る時点からの、スケジュールされているポッドのエンドツーエンドのレイテンシーです。これには複数のスケジューリング試行が含まれる場合があります。
kube_pod_resource_limit	スケジューラ	クラスター上のワークロードに対するリソース制限のポッド別の内訳です。これはスケジューラーと kubelet がリソースのポッドごとに想定するリソースの使用状況と、リソースの単位がある場合はそれを示します。
kube_pod_resource_request	スケジューラ	クラスター上のワークロードによってリクエストされリソースのポッド別の内訳です。これはスケジューラーと kubelet がリソースのポッドごとに想定するリソースの使用状況と、リソースの単位がある場合はそれを示します。
cronjob_controller_job_creation_skew_duration_seconds	コントローラーマネージャー	cronjob の実行がスケジュールされてから、対応するジョブが作成されるまでの時間です。

メトリクス	コントロールプレーンのコンポーネント	説明
workqueue_depth	コントローラーマネージャー	現在のキューの深さです。
workqueue_adds_total	コントローラーマネージャー	ワークキューが処理する合計追加数。
workqueue_queue_duration_seconds	コントローラーマネージャー	項目がリクエストされる前にワークキューに留まる時間 (秒)です。
workqueue_work_duration_seconds	コントローラーマネージャー	ワークキューからの項目の処理にかかる時間 (秒)です。

プロメテウス スクレイパーをデプロイしてメトリクスを一貫してスクレイピングする

プロメテウス スクレイパーをデプロイしてメトリクスを一貫してスクレイピングするには次の設定を使用します:

```
---
apiVersion: v1
kind: ConfigMap
metadata:
  name: prometheus-conf
data:
  prometheus.yml: |-
    global:
      scrape_interval: 30s
    scrape_configs:
    # apiserver metrics
    - job_name: apiserver-metrics
      kubernetes_sd_configs:
      - role: endpoints
      scheme: https
      tls_config:
        ca_file: /var/run/secrets/kubernetes.io/serviceaccount/ca.crt
        insecure_skip_verify: true
      bearer_token_file: /var/run/secrets/kubernetes.io/serviceaccount/token
      relabel_configs:
      - source_labels:
```

```
        [
          __meta_kubernetes_namespace,
          __meta_kubernetes_service_name,
          __meta_kubernetes_endpoint_port_name,
        ]
      action: keep
      regex: default;kubernetes;https
  # Scheduler metrics
  - job_name: 'ksh-metrics'
    kubernetes_sd_configs:
    - role: endpoints
    metrics_path: /apis/metrics.eks.amazonaws.com/v1/ksh/container/metrics
    scheme: https
    tls_config:
      ca_file: /var/run/secrets/kubernetes.io/serviceaccount/ca.crt
      insecure_skip_verify: true
    bearer_token_file: /var/run/secrets/kubernetes.io/serviceaccount/token
    relabel_configs:
    - source_labels:
        [
          __meta_kubernetes_namespace,
          __meta_kubernetes_service_name,
          __meta_kubernetes_endpoint_port_name,
        ]
      action: keep
      regex: default;kubernetes;https
  # Controller Manager metrics
  - job_name: 'kcm-metrics'
    kubernetes_sd_configs:
    - role: endpoints
    metrics_path: /apis/metrics.eks.amazonaws.com/v1/kcm/container/metrics
    scheme: https
    tls_config:
      ca_file: /var/run/secrets/kubernetes.io/serviceaccount/ca.crt
      insecure_skip_verify: true
    bearer_token_file: /var/run/secrets/kubernetes.io/serviceaccount/token
    relabel_configs:
    - source_labels:
        [
          __meta_kubernetes_namespace,
          __meta_kubernetes_service_name,
          __meta_kubernetes_endpoint_port_name,
        ]
      action: keep
```

```
        regex: default;kubernetes;https
---
apiVersion: v1
kind: Pod
metadata:
  name: prom-pod
spec:
  containers:
  - name: prom-container
    image: prom/prometheus
    ports:
    - containerPort: 9090
    volumeMounts:
    - name: config-volume
      mountPath: /etc/prometheus/
  volumes:
  - name: config-volume
    configMap:
      name: prometheus-conf
```

ポッドが新しいメトリクスエンドポイントにアクセスするには次の許可が必要です。

```
{
  "effect": "allow",
  "apiGroups": [
    "metrics.eks.amazonaws.com"
  ],
  "resources": [
    "kcm/metrics",
    "ksh/metrics"
  ],
  "verbs": [
    "get"
  ] },
```

使用中のロールにパッチを適用するには次のコマンドを使用できます。

```
kubectl patch clusterrole <role-name> --type=json -p='[
  {
    "op": "add",
    "path": "/rules/-",
    "value": {
      "verbs": ["get"],
```

```
      "apiGroups": ["metrics.eks.amazonaws.com"],
      "resources": ["kcm/metrics", "ksh/metrics"]
    }
  }
]'
```

その後、Prometheus スクレイパーのポートをローカルポートにプロキシすることで、Prometheus ダッシュボードを表示できます。

```
kubectl port-forward pods/prom-pod 9090:9090
```

Amazon EKS クラスターの場合、コア Kubernetes コントロールプレーンメトリクスも AWS/EKS 名前空間の Amazon CloudWatch メトリクスに取り込まれます。これらを表示するにはCloudWatch コンソールを開き、左側のナビゲーションペインから [すべてのメトリクス] を選択してください。[メトリクス] の選択ページで、クラスターの AWS/EKS 名前空間とメトリクスディメンションを選択してください。

Amazon CloudWatch でクラスターデータをモニタリングする

Amazon CloudWatch はクラウドリソースからメトリクスとログを収集するモニタリングサービスです。バージョン 1.28 以降の新しいクラスターを使用する場合、CloudWatch は基本的な Amazon EKS メトリクスを無料で提供します。ただし、CloudWatch オベサビリティオペレーターを Amazon EKS アドオンとして使用すると、オブザーバビリティ機能を強化できます。

Amazon CloudWatch の基本メトリクス

Kubernetes バージョン 1.28 以降のクラスターの場合、CloudWatch の Vended メトリクスを AWS/EKS 名前空間で無料で取得できます。次の表はサポートされているバージョンで使用できる基本的なメトリクスのリストです。リストされているあらゆるメトリクスの頻度は 1 分です。

メトリクス名	説明
apiserver_flowcontrol_current_executing_seats	API リクエストの実行に現在使用されているシートの数。シート割り当ては、Kubernetes API Priority and Fairness 機能の priority_level と flow_schema の設定によって決まります。 単位: カウント

メトリクス名	説明
	有効な統計: Sum
scheduler_schedule_attempts_total	一定期間にクラスター内の Pod をスケジュールするための、スケジューラによる合計試行回数。このメトリクスは、スケジューラのワークロードのモニタリングに役立ち、スケジューリングのプレッシャーや Pod 配置の潜在的な問題を示している可能性があります。 単位: カウント 有効な統計: Sum
scheduler_schedule_attempts_SCHEDULED	一定期間にクラスター内のノードに Pod をスケジュールするための、スケジューラによる成功した試行回数。 単位: カウント 有効な統計: Sum
scheduler_schedule_attempts_UNSCHEDULABLE	ノードの CPU やメモリの不足などの有効な制約により、一定期間スケジュールできなかった Pod をスケジュールするための試行回数。 単位: カウント 有効な統計: Sum
scheduler_schedule_attempts_ERROR	API サーバーの接続の問題などの、スケジューラ自体の内部的な問題により、一定期間失敗していた Pod をスケジュールするための試行回数。 単位: カウント 有効な統計: Sum

メトリクス名	説明
scheduler_pending_pods	一定期間にクラスター内のスケジューラによってスケジュールされる、保留中の Pod の合計数。 単位: カウント 有効な統計: Sum
scheduler_pending_pods_ACTIVEQ	一定期間にクラスターでスケジュールされることを待機している、activeQ で保留中の Pod の数。 単位: カウント 有効な統計: Sum
scheduler_pending_pods_UNSCHEDULABLE	スケジューラがスケジュールを試みて失敗し、再試行のためにスケジュールできない状態で保持されている、保留中の Pod の数。 単位: カウント 有効な統計: Sum
scheduler_pending_pods_BACKOFF	バックオフ期間の終了を待機している、バックオフ状態で backoffQ で保留中の Pod の数。 単位: カウント 有効な統計: Sum
scheduler_pending_pods_GATED	必要な条件を満たすまでスケジュールできないため、制限あり状態で現在待機している保留中の Pod の数。 単位: カウント 有効な統計: Sum

メトリクス名	説明
apiserver_request_total	クラスター内のすべての API サーバーで行われた HTTP リクエストの数。 単位: カウント 有効な統計: Sum
apiserver_request_total_4XX	4XX (クライアントエラー) ステータスコードが発生した、クラスター内のすべての API サーバーに対して行われた HTTP リクエストの数。 単位: カウント 有効な統計: Sum
apiserver_request_total_429	クライアントがレート制限のしきい値を超えたときに発生する 429 ステータスコードが発生した、クラスター内のすべての API サーバーに対して行われた HTTP リクエストの数。 単位: カウント 有効な統計: Sum
apiserver_request_total_5XX	5XX (サーバーエラー) ステータスコードが発生した、クラスター内のすべての API サーバーに対して行われた HTTP リクエストの数。 単位: カウント 有効な統計: Sum
apiserver_request_total_LIST_PODS	クラスター内のすべての API サーバーに対して行われた Pod の LIST リクエストの数。 単位: カウント 有効な統計: Sum

メトリクス名	説明
apiserver_request_duration_seconds_PUT_P99	クラスター内のすべての API サーバーでのすべてのリクエストから計算された、PUT リクエストのレイテンシーの 99 パーセンタイル。すべての PUT リクエストの 99% が完了する応答時間を表しています。 単位: 秒 有効な統計: 平均
apiserver_request_duration_seconds_PATCH_P99	クラスター内のすべての API サーバーでのすべてのリクエストから計算された、PATCH リクエストのレイテンシーの 99 パーセンタイル。すべての PATCH リクエストの 99% が完了する応答時間を表しています。 単位: 秒 有効な統計: 平均
apiserver_request_duration_seconds_POST_P99	クラスター内のすべての API サーバーでのすべてのリクエストから計算された、POST リクエストのレイテンシーの 99 パーセンタイル。すべての POST リクエストの 99% が完了する応答時間を表しています。 単位: 秒 有効な統計: 平均

メトリクス名	説明
apiserver_request_duration_seconds_GET_P99	クラスター内のすべての API サーバーでのすべてのリクエストから計算された、GET リクエストのレイテンシーの 99 パーセンタイル。すべての GET リクエストの 99% が完了する応答時間を表しています。 単位: 秒 有効な統計: 平均
apiserver_request_duration_seconds_LIST_P99	クラスター内のすべての API サーバーでのすべてのリクエストから計算された、LIST リクエストのレイテンシーの 99 パーセンタイル。すべての LIST リクエストの 99% が完了する応答時間を表しています。 単位: 秒 有効な統計: 平均
apiserver_request_duration_seconds_DELETE_P99	クラスター内のすべての API サーバーでのすべてのリクエストから計算された、DELETE リクエストのレイテンシーの 99 パーセンタイル。すべての DELETE リクエストの 99% が完了する応答時間を表しています。 単位: 秒 有効な統計: 平均

メトリクス名	説明
apiserver_current_inflight_requests_MUTATING	クラスター内のすべての API サーバーで現在処理されている、変更リクエスト (POST、PUT、DELETE、PATCH) の数。このメトリクスは、処理中であり、まだ処理を完了していないリクエストを表しています。 単位: カウント 有効な統計: Sum
apiserver_current_inflight_requests_READONLY	クラスター内のすべての API サーバーで現在処理されている、読み取り専用リクエスト (GET、LIST) の数。このメトリクスは、処理中であり、まだ処理を完了していないリクエストを表しています。 単位: カウント 有効な統計: Sum
apiserver_admission_webhook_request_total	クラスター内のすべての API サーバーで行われたアドミッションウェブフックリクエストの数。 単位: カウント 有効な統計: Sum
apiserver_admission_webhook_request_total_ADMIT	クラスター内のすべての API サーバーで行われた変更アドミッションウェブフックリクエストの数。 単位: カウント 有効な統計: Sum

メトリクス名	説明
apiserver_admission_webhook_request_total_VALIDATING	クラスター内のすべての API サーバーで行われた検証アドミッションウェブフックリクエストの数。 単位: カウント 有効な統計: Sum
apiserver_admission_webhook_rejection_count	拒否されたクラスター内のすべての API サーバーで行われたアドミッションウェブフックリクエストの数。 単位: カウント 有効な統計: Sum
apiserver_admission_webhook_rejection_count_ADMIT	拒否されたクラスター内のすべての API サーバーで行われた変更アドミッションウェブフックリクエストの数。 単位: カウント 有効な統計: Sum
apiserver_admission_webhook_rejection_count_VALIDATING	拒否されたクラスター内のすべての API サーバーで行われた検証アドミッションウェブフックリクエストの数。 単位: カウント 有効な統計: Sum

メトリクス名	説明
apiserver_admission_webhook_admission_duration_seconds	クラスター内のすべての API サーバーでのすべてのリクエストから計算された、サードパーティーのアドミッションウェブフックリクエストのレイテンシーの 99 パーセンタイル。すべてのサードパーティーのアドミッションウェブフックリクエストの 99% が完了する応答時間を表しています。 単位: 秒 有効な統計: 平均
apiserver_admission_webhook_admission_duration_seconds_ADMIT_P99	クラスター内のすべての API サーバーでのすべてのリクエストから計算された、サードパーティーの変更アドミッションウェブフックリクエストのレイテンシーの 99 パーセンタイル。すべてのサードパーティーの変更アドミッションウェブフックリクエストの 99% が完了する応答時間を表しています。 単位: 秒 有効な統計: 平均
apiserver_admission_webhook_admission_duration_seconds_VALIDATING_P99	クラスター内のすべての API サーバーでのすべてのリクエストから計算された、サードパーティーの検証アドミッションウェブフックリクエストのレイテンシーの 99 パーセンタイル。すべてのサードパーティーの検証アドミッションウェブフックリクエストの 99% が完了する応答時間を表しています。 単位: 秒 有効な統計: 平均

メトリクス名	説明
etcd_mvcc_db_total_size_in_bytes	物理的に割り当てられた基盤となるデータベースの合計サイズ (バイト単位)。このメトリクスは物理ディスクの割り当てを測定し、クォータの適用には使用されないことに注意してください。(apiserver_storage_size_bytes とも呼ばれます) 単位: バイト 有効な統計: 最大
etcd_mvcc_db_total_size_in_use_in_bytes	デフラグメンテーションを待っている空き領域を除く、etcd データベースの実際のデータサイズ。このメトリクスは、現在のデータベース使用状況を示し、クラスターがデータベースサイズのクォータを超え、読み取り専用モードになるかどうかを決定します。 単位: バイト 有効な統計: 最大

Amazon CloudWatch オブザーバビリティオペレータ

Amazon CloudWatch Observability は、リアルタイムログ、メトリクス、トレースデータを収集します。それらを Amazon CloudWatch と AWS X-Ray に送信します。このアドオンをインストールすると、CloudWatch アプリケーションシグナルと Amazon EKS のオブザーバビリティが強化された CloudWatch コンテナインサイトの両方を有効にできます。これにより、インフラストラクチャとコンテナ化されたアプリケーションのヘルスとパフォーマンスをモニタリングできます。Amazon CloudWatch オベサビリティオペレーター は必要なコンポーネントをインストールして設定するように設計されています。

Amazon EKS は Amazon EKS アドオンとして CloudWatch オベサビリティオペレーター をサポートしています。このアドオンは、クラスターの Linux と Windows の両方のワーカーノードで Container Insights を有効にします。Container Insights を Windows で有効にするには、Amazon

EKS アドオンのバージョンが 1.5.0 以上である必要があります。現在、CloudWatch Application Signals は Amazon EKS Windows ではサポートされていません。

以下のトピックでは Amazon EKS クラスターのために CloudWatch オベサビリティオペレーター の使用を開始する方法について説明します。

- このアドオンのインストール手順については「Amazon CloudWatch ユーザーガイド」の「Amazon CloudWatch オベサビリティ EKS アドオンまたは Helm チャートを使用して CloudWatch エージェントをインストールする」を参照してください。
- CloudWatch アプリケーションシグナルの詳細については「Amazon CloudWatch ユーザーガイド」の「Application Signals」を参照してください。
- 詳細については、「Amazon CloudWatch ユーザーガイド」の「Container Insights の使用」を参照してください。

コントロールプレーンログを CloudWatch Logs に送信する

Amazon EKS コントロールプレーンのログ記録により、アカウント内で Amazon EKS コントロールプレーンから CloudWatch ログに対し、監査および診断ログを直接送れるようになります。これらのログを使用すると、クラスターの保護と実行が容易になります。必要なログタイプを正確に選択することで、CloudWatch 内で各 Amazon EKS クラスターのためのグループに対し、ログストリームの形態でログを送信できます。CloudWatch サブスクリプションフィルタを使用して、ログをリアルタイムで分析したり、他のサービスに転送したりできます (ログは Base64 でエンコードされ、gzip 形式で圧縮されます)。詳細については、「Amazon CloudWatch logging」(Amazon CloudWatch ログ記録) を参照してください。

> **Note**
>
> Amazon EKS コントロールプレーンログは、数分以内に各 CloudWatch Log に配信されます。ただし、ログ配信はベストエフォートです。

Amazon EKS コントロールプレーンのログ記録の使用を開始するには、新規または既存の Amazon EKS クラスターごとに有効にするログタイプを選択します。クラスターごとに各ログタイプを有効または無効にするには、AWS マネジメントコンソール、AWS CLI (バージョン 1.16.139 以降)、または Amazon EKS API を使用します。有効化したログタイプのログが、Amazon EKS クラスターから同じアカウントの CloudWatch Logs に自動的に送信されるようになります。

Amazon EKS コントロールプレーンのログ記録を使用すると、実行しているクラスターごとに Amazon EKS の標準料金が発生します。クラスターから CloudWatch Logs に送信されるすべてのログに対して、CloudWatch Logs のデータ取り込みおよび保存に関する標準料金が発生します。また、Amazon EC2 インスタンスや Amazon EBS ボリュームなど、クラスターの一部としてプロビジョニングしている AWS リソースに対しても料金が発生します。

以下のクラスターコントロールプレーンのログタイプが使用可能です。各ログタイプは、Kubernetes コントロールプレーンのコンポーネントに対応しています。これらのコンポーネントの詳細については、Kubernetes ドキュメントの Kubernetes Components (Kubernetes コンポーネント) を参照してください。

API サーバー (**`api`**)

クラスターの API サーバーは、Kubernetes API を公開するコントロールプレーンコンポーネントです。クラスターを起動する際、またはその直後に API サーバーのログを有効にすると、ログには、API サーバーの起動に使用された API サーバーフラグが含まれます。詳細については、Kubernetes ドキュメントの「kube-apiserver」と監査ポリシーを参照してください。

監査 (**`audit`**)

Kubernetes 監査ログは、クラスターに影響を与えた個々のユーザー、管理者、またはシステムコンポーネントの記録を提供します。詳細については、Kubernetes ドキュメントの「監査」を参照してください。

Authenticator (**`authenticator`**)

Authenticator ログは、Amazon EKS に固有です。これらのログは、Amazon EKS が使用するコントロールプレーンコンポーネントに対応します。各コンポーネントは、Amazon EKS が、Kubernetes の (IAM 認証情報を使用する) ロールベースのアクセスコントロール (RBAC) 認証のために使用しています。詳細については、「クラスターの管理」を参照してください。

コントローラーマネージャー (**`controllerManager`**)

コントローラーマネージャーは、Kubernetes に付属するコアコントロールループを管理します。詳細については、Kubernetes ドキュメントの kube-controller-manager を参照してください。

スケジューラ (**`scheduler`**)

スケジューラコンポーネントは、クラスター内で Pod を実行するタイミングと場所を管理します。詳細については、Kubernetes ドキュメントの kube-scheduler を参照してください。

コントロールプレーンログを有効または無効にする

デフォルトでは、クラスターコントロールプレーンのログは CloudWatch Logs に送信されません。クラスターのログを送信するには、各ログタイプを個別に有効にする必要があります。CloudWatch Logs で設定された取り込み、アーカイブストレージ、およびデータスキャンレートが、有効化されたコントロールプレーンログに適用されます。詳細については、「CloudWatch 料金表」を参照してください。

コントロールプレーンのログ記録設定を更新するために、Amazon EKS は各サブネットで最大 5 つの使用可能な IP アドレスを必要とします。ログタイプを有効にすると、ログの詳細レベル 2 でログが送信されます。

AWS マネジメントコンソール または AWS CLI のいずれかを使用して、コントロールプレーンログを有効または無効にできます。

AWS マネジメントコンソール

1. Amazon EKS コンソールを開きます。
2. クラスター名を選択すると、そのクラスターの情報を表示されます。
3. [オブザーバビリティ] タブを選択します。
4. [コントロールプレーンのロギング] セクションで、[ロギングの管理] を選択します。
5. ログタイプごとに、そのログタイプを [有効] にするか [無効] にするかを選択します。デフォルトでは、それぞれのログタイプは無効化されています。
6. [変更を保存] を選択して終了します。

AWS CLI

1. 以下のコマンドを使用して、AWS CLI のバージョンを確認します。

```
aws --version
```

AWS CLI のバージョンが 1.16.139 より前である場合は、まず最新バージョンに更新する必要があります。AWS CLI をインストールまたはアップグレードするには、「AWS コマンドラインインターフェイスユーザーガイド」の「AWS コマンドラインインターフェイスのインストール」を参照してください。

2. 次の AWS CLI コマンドを使用して、クラスターのコントロールプレーンログのエクスポート設定を更新します。*my-cluster* をクラスター名に置き換え、目的のエンドポイントアクセス値を指定します。

> **Note**
>
> 次のコマンドは、使用可能なすべてのログタイプを CloudWatch Logs に送信します。

```
aws eks update-cluster-config \
    --region region-code \
    --name my-cluster \
    --logging '{"clusterLogging":[{"types":
["api","audit","authenticator","controllerManager","scheduler"],"enabled":true}]}'
```

出力例は次のとおりです。

```
{
    "update": {
        "id": "883405c8-65c6-4758-8cee-2a7c1340a6d9",
        "status": "InProgress",
        "type": "LoggingUpdate",
        "params": [
            {
                "type": "ClusterLogging",
                "value": "{\"clusterLogging\":[{\"types\":[\"api\",\"audit\",
\"authenticator\",\"controllerManager\",\"scheduler\"],\"enabled\":true}]}"
            }
        ],
        "createdAt": 1553271814.684,
        "errors": []
    }
}
```

3. 次のコマンドでログ設定更新のステータスをモニタリングします。その際、以前のコマンドで返されたクラスター名と更新 ID を使用します。ステータスが Successful と表示されたら、更新は完了です。

```
aws eks describe-update \
    --region region-code \
    --name my-cluster \
```

```
    --update-id 883405c8-65c6-4758-8cee-2a7c1340a6d9
```

出力例は次のとおりです。

```
{
    "update": {
        "id": "883405c8-65c6-4758-8cee-2a7c1340a6d9",
        "status": "Successful",
        "type": "LoggingUpdate",
        "params": [
            {
                "type": "ClusterLogging",
                "value": "{\"clusterLogging\":[{\"types\":[\"api\",\"audit\",
\"authenticator\",\"controllerManager\",\"scheduler\"],\"enabled\":true}]}"
            }
        ],
        "createdAt": 1553271814.684,
        "errors": []
    }
}
```

クラスターのコントロールプレーンログを表示する

Amazon EKS クラスターのいずれかのコントロールプレーンログタイプを有効にすると、それらを CloudWatch コンソールで表示できるようになります。

CloudWatch でログを表示、分析、および管理する方法については、Amazon CloudWatch Logs ユーザーガイドを参照してください。

1. CloudWatch コンソールを開きます。このリンクでは、現在使用可能なロググループが表示され、プレフィックス /aws/eks でフィルタリングされます。
2. ログを表示するクラスターを選択します。ロググループの名前の形式は /aws/eks/*my-cluster*/cluster です。
3. 表示するログストリームを選択します。次のリストで、各ログタイプのログストリーム名の形式について説明します。

> **Note**
>
> ログストリームデータが大きくなるにつれて、ログストリーム名のローテーションが行われます。特定のログタイプのログストリームが複数存在する場合、最新のログストリームを表示するには、[最終のイベント時刻] が最新のログストリーム名を見つけ出します。

- Kubernetes API サーバーコンポーネントログ (**api**) – kube-apiserver-*1234567890abcdef01234567890abcde*
- 監査 (**audit**) – kube-apiserver-audit-*1234567890abcdef01234567890abcde*
- 認証システム (**authenticator**) – authenticator-*1234567890abcdef01234567890abcde*
- コントローラーマネージャー (**controllerManager**) – kube-controller-manager-*1234567890abcdef01234567890abcde*
- スケジューラ (**scheduler**) – kube-scheduler-*1234567890abcdef01234567890abcde*

4. ログストリームのイベントを確認してください。

   たとえば、kube-apiserver-*1234567890abcdef01234567890abcde* のトップを表示すると、クラスターの初期 API サーバーが表示されます。

> **Note**
>
> ログストリームの先頭に API サーバーログが表示されない場合は、サーバーで API サーバーログ記録を有効にする前に API サーバーのログファイルがローテーションされた可能性があります。API サーバーのログ記録が有効になる前にローテーションされたログファイルは、CloudWatch にエクスポートできません。

ただし、同じ Kubernetes バージョンで新しいクラスターを作成し、クラスターの作成時に API サーバーのログ記録を有効にすることはできます。同じプラットフォームバージョンを持つクラスターでは、同じフラグが有効になるため、フラグは新しいクラスターのフラグと一致する必要があります。CloudWatch で新しいクラスターのフラグを確認し終えたら、新しいクラスターを削除します。

API コールを AWS CloudTrail イベントとしてログします。

Amazon EKS は AWS CloudTrail と統合します。CloudTrail は、Amazon EKS のユーザー、ロール、または AWS サービスによるアクションを記録するサービスです。CloudTrail は、Amazon EKS へのすべての API コールをイベントとしてキャプチャします。これには、Amazon EKS コンソールからの呼び出しと、Amazon EKS API オペレーションへのコード呼び出しが含まれます。

証跡を作成する場合は、 のイベントなど、Amazon S3 バケットへの CloudTrail イベントの継続的な配信を有効にすることができます これには、Amazon EKS のイベントが含まれます。追跡を設定しない場合でも、CloudTrail コンソールのイベント履歴で最新のイベントを表示できます。CloudTrail で収集された情報に基づいて、リクエストに関する詳細を確認できます。例えば、Amazon EKS に対するリクエストの日時や、リクエスト元の IP アドレス、リクエストの実行者を確認できます。

CloudTrail に関する詳細は、AWS CloudTrail ユーザーガイドを参照してください。

トピック

- AWS CloudTrail に役立つリファレンスを表示する
- AWS CloudTrail ログファイルエントリを分析する
- Amazon EC2 Auto Scaling グループのメトリクスを表示する

AWS CloudTrail に役立つリファレンスを表示する

AWS アカウントを作成すると、AWS アカウントで CloudTrail も有効になります。Amazon EKS でイベントアクティビティが発生すると、そのアクティビティは [Event history] (イベント履歴) で他の AWS サービスのイベントとともに CloudTrail イベントに記録されます。最近のイベントは、AWSアカウントで表示、検索、ダウンロードできます。詳細については、CloudTrail イベント履歴でのイベントの表示を参照してください。

Amazon EKS のイベントなど、AWS アカウントのイベントの継続的な記録については、証跡を作成します。証跡により、ログファイルを CloudTrail で Amazon S3 バケットに配信できます。デフォルトでは、コンソールで証跡を作成すると、すべての AWS リージョンに証跡が適用されます。追跡では、AWS パーティション内のすべての AWS リージョンからのイベントを口グに記録し、指定した Simple Storage Service (Amazon S3)バケットにログファイルを配信します。さらに、CloudTrail ログで収集したイベントデータをより詳細に分析し、それに基づいて対応するようにその他の AWS サービスを設定できます。詳細については、以下のリソースを参照してください。

- 証跡作成の概要
- 「CloudTrail がサポートされているサービスと統合」

- 「CloudTrail の Amazon SNS 通知の設定」
- 「複数のリージョンから CloudTrail ログファイルを受け取る」および「複数のアカウントから CloudTrail ログファイルを受け取る」

すべての Amazon EKS アクションは、CloudTrail により記録され、Amazon EKS API リファレンスで文書化されます。例えば、CreateCluster、ListClusters および DeleteCluster の各セクションを呼び出すと、 CloudTrail ログファイルにエントリが生成されます。

すべてのイベントまたはログエントリには、リクエストを行った IAM アイデンティティのタイプとどの認証情報が使用されたかに関する詳細が含まれます。一時的認証情報が使用された場合、エレメントは、認証情報がどのように取得されたかを示します。

詳細については、「CloudTrail userIdentity エレメント」を参照してください。

AWS CloudTrail ログファイルエントリを分析する

「トレイル」は、指定した Amazon S3 バケットにイベントをログファイルとして配信するように設定できます。CloudTrail のログファイルには、ログエントリが 1 つ以上あります。イベントでは、あらゆるソースからの単一のリクエストが示され、リクエストされたアクションに関する情報が含まれています。これには、アクションの日時、使用されたリクエストパラメータなどの情報が含まれます。CloudTrail ログファイルは、公開 API コールの順序付けられたスタックトレースではないため、特定の順序では表示されません。

以下の例は、`CreateCluster` アクションを示す CloudTrail ログエントリです。


```
{
  "eventVersion": "1.05",
  "userIdentity": {
    "type": "IAMUser",
    "principalId": "AKIAIOSFODNN7EXAMPLE",
    "arn": "arn:aws:iam::111122223333:user/username",
    "accountId": "111122223333",
    "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
    "userName": "username"
  },
  "eventTime": "2018-05-28T19:16:43Z",
  "eventSource": "eks.amazonaws.com",
  "eventName": "CreateCluster",
  "awsRegion": "region-code",
  "sourceIPAddress": "205.251.233.178",
  "userAgent": "PostmanRuntime/6.4.0",
```

```
  "requestParameters": {
    "resourcesVpcConfig": {
      "subnetIds": [
        "subnet-a670c2df",
        "subnet-4f8c5004"
      ]
    },
    "roleArn": "arn:aws:iam::111122223333:role/AWSServiceRoleForAmazonEKS-
CAC1G1VH3ZKZ",
    "clusterName": "test"
  },
  "responseElements": {
    "cluster": {
      "clusterName": "test",
      "status": "CREATING",
      "createdAt": 1527535003.208,
      "certificateAuthority": {},
      "arn": "arn:aws:eks:region-code:111122223333:cluster/test",
      "roleArn": "arn:aws:iam::111122223333:role/AWSServiceRoleForAmazonEKS-
CAC1G1VH3ZKZ",
      "version": "1.10",
      "resourcesVpcConfig": {
        "securityGroupIds": [],
        "vpcId": "vpc-21277358",
        "subnetIds": [
          "subnet-a670c2df",
          "subnet-4f8c5004"
        ]
      }
    }
  },
  "requestID": "a7a0735d-62ab-11e8-9f79-81ce5b2b7d37",
  "eventID": "eab22523-174a-499c-9dd6-91e7be3ff8e3",
  "readOnly": false,
  "eventType": "AwsApiCall",
  "recipientAccountId": "111122223333"
}
```


Amazon EKS サービスにリンクされたロールのログエントリ

Amazon EKS サービスにリンクされたロールは、AWS リソースへの API コールを行います。Amazon EKS サービスにリンクされたロールによって行われた呼び出しには、CloudTrail ログエントリが `username: AWSServiceRoleForAmazonEKS` と `username:`

AWSServiceRoleForAmazonEKSNodegroup で表示されます。Amazon EKS およびサービスにリンクされたロールの詳細については、「the section called “サービスにリンクされた役割”」を参照してください。

次の例では、sessionContext に記録され、AWSServiceRoleForAmazonEKSNodegroup サービスにリンクされたロールによって行われた DeleteInstanceProfile アクションを示す CloudTrail ログエントリが示されています。

```
{
    "eventVersion": "1.05",
    "userIdentity": {
        "type": "AssumedRole",
        "principalId": "AROA3WHGPEZ7SJ2CW55C5:EKS",
        "arn": "arn:aws:sts::111122223333:assumed-role/
AWSServiceRoleForAmazonEKSNodegroup/EKS",
        "accountId": "111122223333",
        "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
        "sessionContext": {
            "sessionIssuer": {
                "type": "Role",
                "principalId": "AROA3WHGPEZ7SJ2CW55C5",
                "arn": "arn:aws:iam::111122223333:role/aws-service-role/eks-
nodegroup.amazonaws.com/AWSServiceRoleForAmazonEKSNodegroup",
                "accountId": "111122223333",
                "userName": "AWSServiceRoleForAmazonEKSNodegroup"
            },
            "webIdFederationData": {},
            "attributes": {
                "mfaAuthenticated": "false",
                "creationDate": "2020-02-26T00:56:33Z"
            }
        },
        "invokedBy": "eks-nodegroup.amazonaws.com"
    },
    "eventTime": "2020-02-26T00:56:34Z",
    "eventSource": "iam.amazonaws.com",
    "eventName": "DeleteInstanceProfile",
    "awsRegion": "region-code",
    "sourceIPAddress": "eks-nodegroup.amazonaws.com",
    "userAgent": "eks-nodegroup.amazonaws.com",
    "requestParameters": {
        "instanceProfileName": "eks-11111111-2222-3333-4444-abcdef123456"
    },
```

```
    "responseElements": null,
    "requestID": "11111111-2222-3333-4444-abcdef123456",
    "eventID": "11111111-2222-3333-4444-abcdef123456",
    "eventType": "AwsApiCall",
    "recipientAccountId": "111122223333"
}
```

Amazon EC2 Auto Scaling グループのメトリクスを表示する

Amazon EKS マネージドノードグループでは、Amazon EC2 Auto Scaling グループのメトリクスがデフォルトで有効になっており、追加料金はかかりません。Auto Scaling グループは、サンプリングされたデータを毎分 Amazon CloudWatch に送信します。これらのメトリクスは、Auto Scaling グループの名前で絞り込むことができます。これらにより、時間の経過に伴うグループのサイズの変化など、マネージドノードグループで使用する Auto Scaling グループの履歴を継続的に把握できます。Auto Scaling グループのメトリクスは、Amazon CloudWatch または Auto Scaling コンソールで使用できます。詳細については、「Monitor CloudWatch metrics for your Auto Scaling groups and instances」を参照してください。

Auto Scaling グループのメトリクス収集を使用することで、マネージドノードグループのスケーリングをモニタリングできます。Auto Scaling グループのメトリクスは、Auto Scaling グループの最小サイズ、最大サイズ、および必要なサイズを報告します。ノードグループ内のノード数が最小サイズを下回った場合にアラームを作成できます。これは、ノードグループに異常があることを示しています。ノードグループサイズを追跡することは、データプレーンの容量が不足しないように最大数を調整する場合にも役立ちます。

これらのメトリクスを収集しない場合は、全部または一部のメトリクスのみを無効にすることを選択できます。例えば、これを実行して、CloudWatch ダッシュボードのノイズを避けることができます。詳細については、「Amazon CloudWatch metrics for Amazon EC2 Auto Scaling」を参照してください。

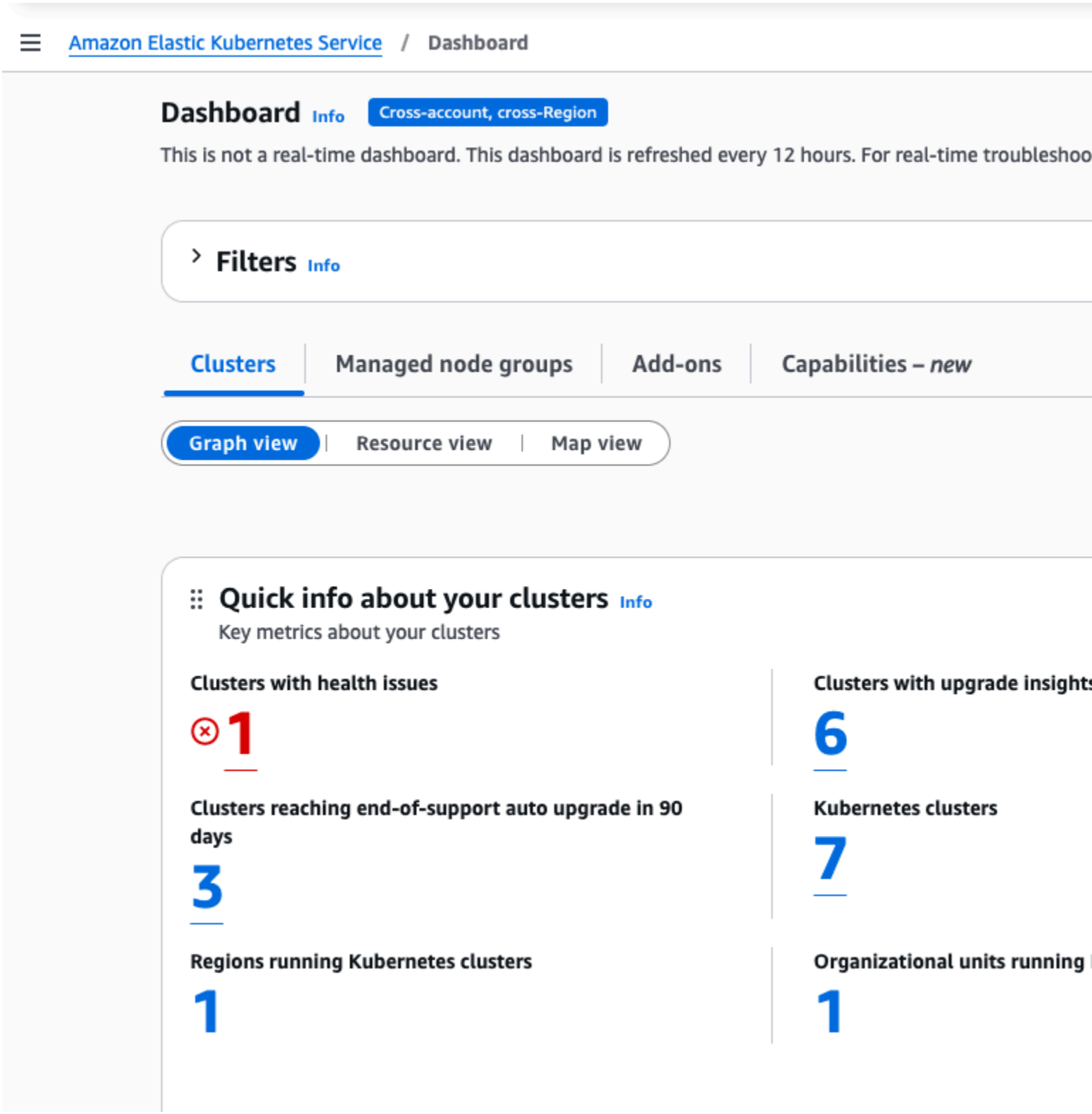
ADOT Operator を使用してメトリクスとトレースデータを送信する

Amazon EKS は、AWS マネジメントコンソール、AWS CLI、および Amazon EKS API を使用して、AWS Distro for OpenTelemetry (ADOT) 演算子をインストールおよび管理することをサポートしています。これにより、Amazon EKS 上で実行されているアプリケーションが、メトリクスやトレースデータを Amazon CloudWatch、Prometheus、X-Ray などの複数のモニタリングサービスオプションに送信することが容易になります。

詳細については、AWS Distro for OpenTelemetry ドキュメントの「Getting Started with AWS Distro for OpenTelemetry using EKS Add-Ons」を参照してください。

EKS ダッシュボードを使用してクラスターリソースに関する集計データを表示する

Amazon EKS ダッシュボードとは



Amazon EKS ダッシュボードは、複数の AWS リージョンと AWS アカウントにまたがる Kubernetes クラスターの統合可視性を提供します。このダッシュボードでは、次のことができます。

- 今後 90 日以内にサポート終了の自動アップグレードが予定されているクラスターを追跡する。
- 延長サポートのクラスターにおける EKS コントロールプレーンのコストを予測する。
- アップグレード前に注意が必要なインサイトを含むクラスターを確認する。
- 特定の AMI バージョンを実行しているマネージド型ノードグループを特定する。
- クラスターサポートタイプの分布 (標準サポートと拡張サポート) をモニタリングする。

EKS ダッシュボードは EKS クラスターインサイトと統合され、非推奨の Kubernetes API の使用など、クラスターの問題を表示します。詳細については、「the section called “クラスターのインサイト”」を参照してください。

> **Note**
>
> EKS ダッシュボードはリアルタイムではなく、12 時間ごとに更新されます。クラスターのリアルタイムモニタリングについては、「クラスターをモニタリングする」を参照してください。

ダッシュボードは AWS Organizations をどのように使用しますか?

Amazon EKS ダッシュボードの機能には AWS Organizations の統合が必要です。AWS Organizations を活用して、アカウント間でクラスター情報を安全に収集します。この統合により、AWS インフラストラクチャのスケールに応じた一元管理とガバナンスが可能になります。

AWS Organizations がインフラストラクチャに対して有効になっていない場合は、セットアップ手順について「AWS Organizations ユーザーガイド」を参照してください。

クロスリージョンおよびクロスアカウントアクセス

EKS ダッシュボードは、AWS Organization のメンバーである任意のアカウントのクラスターリソースを表示できます。組織内の AWS アカウントのリストを生成するには、「Export details for all accounts in an organization」を参照してください。

us-east-1 AWS リージョンはダッシュボードを生成します。ダッシュボードを表示するには、このリージョンにログインする必要があります。ダッシュボードは AWS リージョン間でデータを集約しますが、これには GovCloud または中国リージョンは含まれません。

重要な用語

- AWS Organization: 複数の AWS アカウントの統一された管理構造。
- 管理アカウント: AWS Organization を制御するプライマリアカウント。
- メンバーアカウント: 管理アカウントを除く組織内の任意のアカウント。
- 委任管理者: 特定のクロスアカウント管理権限を付与されたメンバーアカウント。管理アカウント内で、AWS サービスごとに 1 つの委任管理者アカウントを選択できます。
- 信頼されたアクセス: 組織アカウント間でクラスター情報にアクセスするための EKS ダッシュボードの承認。
- サービスリンクロール (SLR): AWS サービスに直接リンクされた一意のタイプの IAM ロール。EKS ダッシュボードは SLR を使用して、アカウントと組織に関する情報を読み込みます。
- 詳細については、AWS Organizations ユーザーガイドの「用語およびコンセプト」を参照してください。

全般的な概要

1. AWS Organization の管理アカウントにアクセスします。
   - 管理アカウントにアクセスする手順は、AWS Organization の設定方法によって異なります。例えば、AWS Identity Center または Okta を介して管理アカウントにアクセスできます。
2. EKS コンソールを介して信頼されたアクセスを有効にします。
3. AWS アカウント ID を使用して委任管理者を割り当てます。
4. 委任管理者アカウントに切り替えます。
5. 組織全体の可視性を備えた拡張 EKS コンソールにアクセスします。

AWS コンソールを使用して EKS ダッシュボードを有効にする

> **Important**
>
> EKS ダッシュボードを有効にするには、AWS Organization の管理アカウントにログインする必要があります。

EKS ダッシュボード設定にアクセスする

1. 次の点を確認します。
   a. AWS Organizations が有効で設定されている。
   b. 組織の管理アカウントにログインしている。
   c. us-east-1 リージョンで AWS マネジメントコンソールを表示している。
2. EKS コンソールに移動します。
3. 左側のサイドバーで、ダッシュボード設定を開きます。

Amazon EKS ダッシュボードへのアクセスを設定する

1. EKS ダッシュボードの表示を許可する AWS アカウントの AWS アカウント ID を見つけます。
   a. このステップはオプションですが、推奨されます。このステップを実行しない場合は、管理アカウントからのみダッシュボードにアクセスできます。ベストプラクティスとして、管理アカウントへのアクセスを制限する必要があります。
2. [トラステッドアクセスを有効化] をクリックします。
   a. これで、管理アカウントからダッシュボードを表示できるようになりました。
3. [委任管理者を登録] をクリックし、ダッシュボードの表示に使用する AWS アカウントのアカウント ID を入力します。
   a. これで、委任管理者アカウントまたは管理アカウントからダッシュボードを表示できるようになりました。

ダッシュボードを有効にするために必要なアクセス許可の詳細については、「最低限必要な IAM ポリシー」を参照してください。

EKS ダッシュボードを表示する

1. 委任管理者アカウント (推奨) または管理アカウントにログインします。
2. us-east-1 リージョンにログインします。
3. EKS サービスに移動し、左側のサイドバーからダッシュボードを選択します。

> **Note**
>
> EKS ダッシュボードの表示に必要な IAM アクセス許可を確認します。

ダッシュボードを設定する

ダッシュボードのビューを設定し、リソースをフィルタリングできます。

利用可能なリソース

- クラスター: EKS クラスターのステータスと場所に関する集計情報を確認します。
  - ヘルスの問題があるクラスター。
  - EKS のクラスター延長サポート。
  - Kubernetes バージョン別のクラスターの内訳。
- マネージド型ノードグループ: マネージド型ノードグループと EC2 インスタンスを確認します。
  - Amazon Linux や Bottlerocket などの AMI タイプ別のノードグループ。
  - ノードグループヘルスの問題。
  - インスタンスタイプの分散。
- アドオン: インストールした Amazon EKS アドオンとそのステータスについて確認します。
  - アドオンあたりのインストール数。
  - ヘルスの問題があるアドオン。
  - アドオンあたりのバージョン分布。
- 機能: EKS クラスターで有効になっている機能とそのステータスを確認します。
  - 機能の分布 (タイプ別)。
  - ヘルスの問題がある機能。
  - ヘルスの問題の内訳 (コード別)。

利用可能なビュー

- グラフビュー
  - 選択したリソースのグラフと視覚化を表示するカスタマイズ可能なウィジェットビュー。
  - ウィジェットの削除など、グラフビューへの変更は、EKS ダッシュボードのすべてのユーザーに表示されます。

- リソースビュー
  - 選択したリソースのリストビュー。フィルターをサポートします。
- マップビュー
  - 選択したリソースの地理的分布を表示します。

EKS ダッシュボードをフィルタリングする

EKS ダッシュボードは、次の方法でフィルタリングできます。

- AWS[アカウント]
- AWS Organizations で定義される組織単位
- AWS リージョン

AWS コンソールを使用して EKS ダッシュボードを無効にする

1. 次の点を確認します。
   a. AWS Organizations が有効で設定されている。
   b. 組織の管理アカウントにログインしている。
   c. us-east-1 リージョンで AWS マネジメントコンソールを表示している。
2. EKS コンソールに移動します。
3. 左側のサイドバーで、ダッシュボード設定を開きます。
4. [トラステッドアクセスを無効化] をクリックします。

EKS ダッシュボードのトラブルシューティング

EKS ダッシュボードの有効化に関する問題

- AWS Organization の管理アカウントにログインする必要があります。
  - AWS Organization がない場合は、作成します。「Create and configure an organization」ではその方法を説明しています。
  - AWS アカウントが既に AWS Organization のメンバーである場合は、組織の管理者を特定します。
- AWS Organizations リソースを作成および更新するには、十分な IAM アクセス許可で AWS アカウントにログインする必要があります。

EKS ダッシュボードの表示に関する問題

- 次のいずれかの AWS アカウントにログインする必要があります。
  - AWS Organization の管理アカウント
  - 管理アカウントの EKS ダッシュボード設定で識別される委任管理者アカウント。
- EKS ダッシュボードを最近有効にした場合、最初のデータセットには最大 12 時間かかる場合があることに注意してください。
- サービスにリンクされたロールの作成を含め、CLI を使用してダッシュボードの再有効化を試みます。

ダッシュボードウィジェットが予期せず移動する

- EKS ダッシュボードは、設定可能なウィジェットビューを AWS アカウントレベルで保存します。ウィジェットビューを変更すると、同じ AWS アカウントを使用している他のユーザーに変更が表示されます。

EKS ダッシュボードと AWS Organizations の統合を設定する

このセクションでは、EKS ダッシュボードの AWS Organizations との統合を設定するための手順を説明します。サービス間の信頼されたアクセスを有効または無効にする方法と、委任管理者アカウントを登録および登録解除する方法について説明します。各設定タスクは、AWS コンソールまたは AWS CLI のいずれかを使用して実行できます。

信頼されたアクセスを有効にする

信頼されたアクセスは、EKS ダッシュボードが組織内のすべてのアカウントでクラスター情報に安全にアクセスすることを許可します。

AWS コンソールを使用する場合

1. AWS Organization の管理アカウントにログインします。
2. us-east-1 リージョンの EKS コンソールに移動します。
3. 左側のサイドバーで、ダッシュボード設定を選択します。
4. [トラステッドアクセスを有効化] をクリックします。

> **Note**
>
> EKS コンソールを使用して信頼されたアクセスを有効にすると、`AWSServiceRoleForAmazonEKSDashboard` サービスにリンクされたロールが自動的に作成されます。AWS CLI または AWS Organizations コンソールを使用して信頼されたアクセスを有効にしても、この自動作成は行われません。

AWS CLI の使用

1. AWS Organization の管理アカウントにログインします。
2. 以下のコマンドを実行します。

```
aws iam create-service-linked-role --aws-service-name dashboard.eks.amazonaws.com
aws organizations enable-aws-service-access --service-principal eks.amazonaws.com
```

信頼されたアクセスを無効にする

信頼されたアクセスを無効にすると、組織のアカウント全体でクラスター情報にアクセスするための EKS ダッシュボードのアクセス許可が取り消されます。

AWS コンソールを使用する場合

1. AWS Organization の管理アカウントにログインします。
2. us-east-1 リージョンの EKS コンソールに移動します。
3. 左側のサイドバーで、ダッシュボード設定を選択します。
4. [トラステッドアクセスを無効化] をクリックします。

AWS CLI の使用

1. AWS Organization の管理アカウントにログインします。
2. 次のコマンドを実行します。

```
aws organizations disable-aws-service-access --service-principal eks.amazonaws.com
```

委任管理者アカウントを有効にする

委任管理者は、EKS ダッシュボードへのアクセス許可が付与されたメンバーアカウントです。

AWS コンソールを使用する場合

1. AWS Organization の管理アカウントにログインします。
2. us-east-1 リージョンの EKS コンソールに移動します。
3. 左側のサイドバーで、ダッシュボード設定を選択します。
4. [委任管理者を登録] をクリックします。
5. 委任管理者として選択する AWS アカウントのアカウント ID を入力します。
6. 登録を確認します。

AWS CLI の使用

1. AWS Organization の管理アカウントにログインします。
2. 123456789012 を自分のアカウント ID に置き換えて、以下のコマンドを実行します。

```
aws organizations register-delegated-administrator --account-id 123456789012 --
service-principal eks.amazonaws.com
```

委任管理者アカウントを無効にする

委任管理者を無効にすると、EKS ダッシュボードにアクセスするためのアカウントのアクセス許可が削除されます。

AWS コンソールを使用する場合

1. AWS Organization の管理アカウントにログインします。
2. us-east-1 リージョンの EKS コンソールに移動します。
3. 左側のサイドバーで、ダッシュボード設定を選択します。
4. リストで委任管理者を見つけます。
5. 委任管理者として削除するアカウントの横にある [登録解除] をクリックします。

AWS CLI の使用

1. AWS Organization の管理アカウントにログインします。

2. 委任管理者のアカウント ID を 123456789012 に置き換えて、次のコマンドを実行します。

```
aws organizations deregister-delegated-administrator --account-id 123456789012 --
service-principal eks.amazonaws.com
```

最低限必要な IAM ポリシー

このセクションでは、信頼されたアクセスを有効にし、EKS ダッシュボードと AWS Organizations の統合のために管理者を委任するために必要な最小限の IAM ポリシーの概要を説明します。

信頼されたアクセスを有効にするためのポリシー

EKS ダッシュボードと AWS Organizations 間の信頼されたアクセスを有効にするには、以下のアクセス許可が必要です。

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "organizations:EnableAWSServiceAccess",
                "organizations:DescribeOrganization",
                "organizations:ListAWSServiceAccessForOrganization"
            ],
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "iam:CreateServiceLinkedRole"
            ],
            "Resource": "arn:aws:iam::*:role/aws-service-role/
dashboard.eks.amazonaws.com/AWSServiceRoleForAmazonEKSDashboard"
        }
    ]
}
```

管理者を委任するためのポリシー

EKS ダッシュボードの委任管理者を登録または登録解除するには、以下のアクセス許可が必要です。

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "organizations:RegisterDelegatedAdministrator",
                "organizations:DeregisterDelegatedAdministrator",
                "organizations:ListDelegatedAdministrators"
            ],
            "Resource": "*"
        }
    ]
}
```

EKS ダッシュボードを表示するポリシー

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "AmazonEKSDashboardReadOnly",
            "Effect": "Allow",
            "Action": [
                "eks:ListDashboardData",
                "eks:ListDashboardResources",
                "eks:DescribeClusterVersions"
            ],
            "Resource": "*"
        },
        {
            "Sid": "AmazonOrganizationsReadOnly",
            "Effect": "Allow",
            "Action": [
                "organizations:DescribeOrganization",
                "organizations:ListAWSServiceAccessForOrganization",
                "organizations:ListRoots",
                "organizations:ListAccountsForParent",
                "organizations:ListOrganizationalUnitsForParent"
            ],
            "Resource": "*"
        },
        {
```

```
            "Sid": "AmazonOrganizationsDelegatedAdmin",
            "Effect": "Allow",
            "Action": [
                "organizations:ListDelegatedAdministrators"
            ],
            "Resource": [
                "*"
            ],
            "Condition": {
                "StringEquals": {
                    "organizations:ServicePrincipal": "eks.amazonaws.com"
                }
            }
        }
    ]
}
```

> **Note**
>
> これらのポリシーは、AWS Organization の管理アカウントの IAM プリンシパル (ユーザーまたはロール) にアタッチする必要があります。メンバーアカウントは、信頼されたアクセスを有効にしたり、管理者を委任したりすることはできません。

統合 AWS サービスによる EKS の強化

他のセクションで説明されているサービスに加えて、Amazon EK はより多くの AWS サービスと連携して追加のソリューションを提供しています。このトピックで説明するサービスは、Amazon EKS を使用して機能を追加するか、タスクを実行するために Amazon EKS に使用されるかのいずれかです。

トピック

- AWS Backup を使用して EKS クラスターをバックアップする
- AWS CloudFormation を使用して Amazon EKS リソースを作成する
- AWS CodeConnections を使用して Git リポジトリに接続する
- Amazon Detective を使用して EKS のセキュリティイベントを分析する
- Amazon GuardDuty で脅威を検出する
- AWS Local Zones を使用して低レイテンシーの EKS クラスターを起動する
- AWS Resilience Hub で EKS クラスターの耐障害性を評価する
- AWS Secrets Manager を使用してアプリケーションシークレットを管理する
- Security Lake を使用して EKS セキュリティデータを一元化して分析する
- Amazon VPC Lattice との安全なクラスター間接続を有効にする

AWS Backup を使用して EKS クラスターをバックアップする

AWS バックアップは、Kubernetes クラスターの状態と、永続ボリュームクレームを介して Amazon EKS クラスターにアタッチされた永続ストレージ (EBS ボリューム、EFS ファイルシステム、S3 バケット) を含む、Amazon EKS クラスターのバックアップをサポートします。Amazon EKS バックアップでは複合復旧ポイントが作成され、バックアップされたリソースごとに子復旧ポイントが作成されます。

- リソースをバックアップする方法: AWS バックアップの開始方法
- リソースタイプ別のバックアップの作成: Amazon EKS バックアップ
- Amazon EKS クラスターを復元する方法: Amazon EKS バックアップの復元

EKS コンソール、AWS バックアップコンソール、または CLI を使用して開始するには、IAM ロールに次のアクセス許可があることを確認してください。

- これらは、AWS バックアップのマネージドポリシー AWSBackupServiceRolePolicyForBackup にあります。これには、Amazon EKS クラスターと EBS および EFS の永続ストレージをバックアップするために必要なアクセス許可が含まれています
- EKS クラスターに S3 バケットが含まれている場合は、文書化されているとおりに、S3 バケットの以下のポリシーと前提条件が追加され、有効になっていることを確認する必要があります。
- AWSBackupServiceRolePolicyForS3Backup
- S3 バックアップ の前提条件

EKS クラスターに次の設定があることを確認します。

- AWS バックアップが EKS クラスターにアクセスするための アクセスエントリ を作成できるよう、EKS クラスターの 認可モード を API または API_AND_CONFIG_MAP に設定します。

AWS CloudFormation を使用して Amazon EKS リソースを作成する

Amazon EKS は AWS CloudFormation と統合されています。これは、リソースとインフラストラクチャの作成と管理の所要時間を短縮できるように AWS リソースをモデル化して設定するためのサービスです。すべての必要な AWS リソース (Amazon EKS クラスターなど) を記述するテンプレートを作成すれば、これらのリソースのプロビジョニングと設定は AWS CloudFormation で代行されます。

AWS CloudFormation を使用すると、テンプレートを再利用して Amazon EKS リソースを同じように繰り返してセットアップできます。リソースを一度記述するだけで、複数の AWS アカウントとリージョンに同じリソースを何度もプロビジョニングすることができます。

Amazon EKS と AWS CloudFormation のテンプレート

Amazon EKS および関連サービスのリソースをプロビジョニングして設定するには、AWS CloudFormation テンプレートについて理解しておく必要があります。テンプレートは、JSON や YAML でフォーマットされたテキストファイルです。これらのテンプレートは、AWS CloudFormation スタックでプロビジョニングするリソースについて記述します。JSON または YAML に詳しくない場合は、AWS CloudFormation Designer を使用して AWS CloudFormation テンプレートの使用を開始できます。詳細については、AWS CloudFormation ユーザーガイド の「AWS CloudFormation デザイナーとは」を参照してください。

Amazon EKS では、AWS CloudFormation でのクラスターとノードグループの作成がサポートされています。Amazon EKS リソースの JSON テンプレートと YAML テンプレートの例を含む詳細情報については、AWS CloudFormation ユーザーガイドの「Amazon EKS リソースタイプのリファレンス」を参照してください。

AWS CloudFormation の詳細はこちら

AWS CloudFormation の詳細については、以下のリソースを参照してください。

- AWS CloudFormation
- AWS CloudFormation ユーザーガイド
- AWS CloudFormation コマンドラインインターフェイスユーザーガイド

AWS CodeConnections を使用して Git リポジトリに接続する

AWS CodeConnections は、AWS サービスをサードパーティーのソースコードリポジトリに接続するための安全な方法を提供します。AWSCodeConnections は、GitHub、GitLab、Bitbucket、およびその他のプロバイダーをサポートしています。詳細と開始方法については、「接続の使用」を参照してください。

Argo CD で AWS CodeConnections を使用する

EKS Capability for Argo CD を使用する場合、存続期間の長い認証情報や個人用アクセストークンを管理することなく、AWS CodeConnections を使用して Git リポジトリへの安全な認証を有効にすることができます。AWSCodeConnections は OAuth 認証フローを処理し、Git プロバイダーへの接続を管理することで、サードパーティーの Git プロバイダーに保存されている GitOps リポジトリとアプリケーションマニフェストにアクセスするための安全で管理しやすいアプローチを提供します。

前提条件

- Amazon EKS クラスターに Argo CD 機能が作成されていること
- Git プロバイダーへの接続が AWS CodeConnections で作成されていること
- Argo CD が接続を使用するように IAM アクセス許可が設定されていること

Argo CD リポジトリアクセス用の CodeConnections を設定するには

1. CodeConnections コンソールで接続を作成します。

a. CodeConnections コンソールを開きます。

b. [接続を作成] を選択します。

c. プロバイダー (GitHub、GitLab、または Bitbucket) を選択し、認証フローに従います。

d. Argo CD の設定で使用するため、接続 ARN を書き留めます。

2. Argo CD 機能ロールが、リソースベースのポリシーにより接続を使用するためのアクセス許可を持っていることを確認します。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "codeconnections:UseConnection",
        "codeconnections:GetConnection"
      ],
      "Resource": "arn:aws:codeconnections:region:account-id:connection/connection-
id"
    }
  ]
}
```

3. CodeConnections リソースエンドポイントをリポジトリ URL として使用して、リポジトリを追加するときに、CodeConnections リソースを参照するように Argo CD を設定します。Argo CD 機能は、存続期間の長い認証情報を必要とせず、接続を使用して Git プロバイダーを認証します。

Argo CD を用いた CodeConnections を使用する際の考慮事項

EKS Capability for Argo CD で AWS CodeConnections を使用する場合は、次の点に注意してください。

- CodeConnections 接続は、EKS クラスターと同じ AWS リージョンにある必要がある
- Argo CD 機能ロールには、`codeconnections:UseConnection` および `codeconnections:GetConnection` のアクセス許可がある必要がある
- CodeConnections は、OAuth フローと認証情報のライフサイクルを自動的に管理する

Argo CD でリポジトリアクセスを設定する方法の詳細については、「the section called “リポジトリを設定する”」を参照してください。

Amazon Detective を使用して EKS のセキュリティイベントを分析する

Amazon Detective を使用すると、セキュリティに関する検出結果や疑わしいアクティビティの根本原因を分析、調査、および迅速に特定できます。Detective は、AWS リソースからログデータを自動的に収集します。その後、機械学習、統計分析、グラフ理論を使用して、セキュリティ調査をより迅速かつ効率的に行うのに役立つビジュアライゼーションを生成します。Detective の事前に作成されたデータの集計、要約、およびコンテキストは、考えられるセキュリティ問題の性質と範囲を迅速に分析および特定するのに役立ちます。詳細については、「Amazon Detective ユーザーガイド」を参照してください。

Detective は、Kubernetes と AWS のデータを次のような結果に整理します。

- Amazon EKS クラスターの詳細 (クラスターやクラスターのサービスロールを作成した IAM ID など)。Detective を使用して、これらの IAM ID の AWS および Kubernetes API アクティビティを調査できます。
- コンテナの詳細 (イメージやセキュリティコンテキストなど)。終了済みの Pod の詳細を確認することもできます。
- Kubernetes API アクティビティ (API アクティビティの全体的な傾向と特定の API 呼び出しの詳細の両方を含む)。例えば、選択した時間範囲中に発行されて、成功および失敗した Kubernetes API 呼び出しの数を表示できます。また、新たに確認された API 呼び出しに関するセクションは、疑わしいアクティビティを特定するのに役立つ場合があります。

Amazon EKS 監査ログは、Detective の動作グラフに追加できるオプションのデータソースパッケージです。利用可能なオプションのソースパッケージとそのステータスは、アカウント内で確認できます。詳細については、「Amazon Detective ユーザーガイド」の「Amazon EKS audit logs for Detective」を参照してください。

Amazon EKS との連携で Amazon Detective を使用する

結果を確認できるようにするには、クラスターが配置されているのと同じ AWS リージョンで、Detective を少なくとも 48 時間有効にする必要があります。詳細については、「Amazon Detective ユーザーガイド」の「Amazon Detective の設定」を参照してください。

1. https://console.aws.amazon.com/detective/ で Detective コンソールを開きます。
2. 左のナビゲーションペインで [検索] を選択します。

3. [タイプを選択] を選択し、[EKS クラスター] を選択します。
4. クラスター名または ARN を入力し、[検索] を選択します。
5. 検索結果で、アクティビティを表示するクラスターの名前を選択します。表示できる内容の詳細については、「Amazon Detective ユーザーガイド」の「Overall Kubernetes API activity involving an Amazon EKS cluster」を参照してください。

Amazon GuardDuty で脅威を検出する

Amazon GuardDuty は、AWS 環境内のアカウント、コンテナ、ワークロード、データを保護する脅威検知サービスです。GuardDuty は、機械学習 (ML) モデル、異常および脅威検出機能を使用して、さまざまなログソースとランタイムアクティビティを継続的に監視し、環境内の潜在的なセキュリティリスクと悪意のあるアクティビティを特定して優先順位を付けます。

GuardDuty には、EKS クラスターに対する潜在的な脅威を検出する EKS Protection と ランタイムモニタリング の 2 つの機能があります。

> **Note**
>
> 新規: Amazon EKS Auto Mode は GuardDuty と統合されています。

EKS Protection

脅威検出を備えた機能であり、関連付けられた Kubernetes 監査ログをモニタリングすることで、Amazon EKS クラスターを保護できるようになります。Kubernetes 監査ログは、ユーザー、Kubernetes API を使用するアプリケーション、コントロールプレーンからのアクティビティなど、クラスター内のシーケンシャルアクションをキャプチャします。例えば、Kubernetes クラスター内のリソースを改ざんするために呼び出された可能性のある API が、認証されていないユーザーによって呼び出された場合、GuardDuty はそのことを識別できます。

EKS Protection を有効にすると、GuardDuty は継続的な脅威検出のためにのみ Amazon EKS 監査ログにアクセスできるようになります。GuardDuty は、クラスターに対する潜在的な脅威を識別すると、特定のタイプの関連する Kubernetes 監査ログの検出結果を生成します。Kubernetes 監査ログから得られる検出結果のタイプの詳細については、「Amazon GuardDuty ユーザーガイド」の「Amazon GuardDuty User Guide」を参照してください。

詳細については、「Amazon GuardDuty ユーザーガイド」の「EKS Protection」を参照してください。

ランタイムモニタリング

この機能は、オペレーティングシステムレベル、ネットワーク、ファイルのイベントを監視して分析し、環境内の特定の AWS ワークロードにおける潜在的な脅威を検出するのに役立ちます。

ランタイムモニタリング を有効にして Amazon EKS クラスターに GuardDuty エージェントをインストールすると、GuardDuty はこのクラスターに関連付けられたランタイムイベントのモニタリングを開始します。GuardDuty エージェントと ランタイムモニタリング は Amazon EKS Hybrid Nodes では使用できないため、ランタイムモニタリング はハイブリッドノードで発生するランタイムイベントでは使用できないことにご注意ください。GuardDuty がクラスターに対する潜在的な脅威を特定すると、関連する ランタイムモニタリングの結果 を生成します。例えば、脅威は、脆弱なウェブアプリケーションを実行している 1 つのコンテナを危険にさらすことから始まる可能性があります。このウェブアプリケーションには、基盤となるコンテナとワークロードへのアクセス権限が存在する可能性があります。この場合、認証情報が正しく設定されていないと、アカウントとその中に保存されているデータへのアクセスを制御できない可能性があります。

ランタイムモニタリング を設定するには Amazon EKS アドオン としてクラスターに GuardDuty エージェントをインストールします。アドオンの詳細については、「the section called “ AWS アドオン”」を参照してください。

詳細については、「Amazon GuardDuty ユーザーガイド」の「ランタイムモニタリング」を参照してください。

AWS Local Zones を使用して低レイテンシーの EKS クラスターを起動する

AWS Local Zones は AWS リージョンの拡張であり、ユーザーに近い場所に配置されます。Local Zones は、インターネットへの独自の接続を持ち、AWS Direct Connect をサポートします。Local Zone で作成したリソースは、低いレイテンシーの通信をローカルユーザーに提供できます。詳細については、「AWS Local Zones ユーザーガイド」と、「Amazon EC2 ユーザーガイド」の「Local Zones」を参照してください。

Amazon EKS では、Local Zones の特定のリソースをサポートしています。これには、マネージドノードグループ、セルフマネージド Amazon EC2 ノード、Amazon EBS ボリューム、および Application Load Balancer (ALB) が含まれます。Amazon EKS クラスターの一部として Local Zones を使用する場合は、次を考慮することをお勧めします。

- Amazon EKS を使用して Local Zones に Fargate ノードを作成することはできません。
- Amazon EKS が管理する Kubernetes コントロールプレーンは、常に AWS リージョンで実行されます。Amazon EKS が管理する Kubernetes コントロールプレーンを Local Zone で実行することはできません。Local Zones は VPC 内のサブネットとして表示されるため、Kubernetes は Local Zone のリソースをそのサブネットの一部として認識します。
- Amazon EKS Kubernetes クラスターは、Amazon EKS が管理する Elastic Network Interface を使用して、AWS リージョンまたは Local Zone で実行する Amazon EC2 インスタンスと通信します。Amazon EKS ネットワークアーキテクチャの詳細については、「ネットワークを設定する」を参照してください。。
- リージョン別のサブネットとは異なり、Amazon EKS はクロスアカウントの Elastic Network Interface をローカルゾーンサブネットに配置することはできません。このようなインターフェイスは、コントロールプレーンとノード間の通信に使用されます。つまり、クラスターの作成時に Local Zone サブネットを指定することはできません。ただし、複数のローカルゾーンにあるワーカーノードを同じクラスターに接続することは可能です。

AWS Resilience Hub で EKS クラスターの耐障害性を評価する

AWS は、Amazon EKS クラスターのインフラストラクチャを分析して、その耐障害性を評価します。AWSResilience Hub は、Kubernetes のロールベースのアクセスコントロール (RBAC) 設定を使用して、クラスターにデプロイされた Kubernetes のワークロードを評価します。詳細については、「AWS Resilience Hub ユーザーガイド」の「Amazon EKS クラスターへの AWS Resilience Hub のアクセスを有効にする」を参照してください。

AWS Secrets Manager を使用してアプリケーションシークレットを管理する

AWS Secrets Manager は、認証情報、API キー、その他のシークレットをそのライフサイクルを通して管理、アクセス、ローテーションするのに役立ちます。Secrets Manager を使用すると、AWS、サードパーティーサービス、オンプレミスのリソースへのアクセスに使うシークレットにセキュリティを確保し、またこれを管理できます。詳細については、AWS Secrets Manager ユーザーガイドを参照してください。

EKS Capability for Argo CD を使用する場合、Secrets Manager は、Argo CD 設定とリソースで機密データをハードコーディングすることなく、Git リポジトリ認証情報を保存および取得するための安全な方法を提供します。この統合は、Argo CD が Git リポジトリからアプリケーションを同期するた

めに使用する、プライベートリポジトリのアクセストークンと SSH キーを管理する場合に特に便利です。

Argo CD で AWS Secrets Manager を使用する

EKS Capability for Argo CD を使用する場合、Git リポジトリ認証情報を Secrets Manager に保存し、それらを取得するように Argo CD を設定できます。このアプローチは、認証情報を Argo CD 設定に直接保存したり、存続期間の長い個人用アクセストークンを使用したりするよりも安全です。

前提条件

- Amazon EKS クラスターで Argo CD 機能が有効になっていること
- Git リポジトリ認証情報が AWS Secrets Manager に保存されていること
- Argo CD が Secrets Manager にアクセスするように IAM アクセス許可が設定されていること

リポジトリ認証情報に Secrets Manager を使用するように Argo CD を設定するには

1. Git 認証情報を Secrets Manager に保存します。例えば、GitHub の個人用アクセストークンを保存するには、次のようにします。

```
aws secretsmanager create-secret \
  --name argocd/github-token \
  --secret-string '{"username":"git","password":"ghp_xxxxxxxxxxxx"}'
```

2. Argo CD 機能ロールにシークレットを取得するアクセス許可があることを確認します。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "secretsmanager:GetSecretValue",
        "secretsmanager:DescribeSecret"
      ],
      "Resource": "arn:aws:secretsmanager:region:account-id:secret:argocd/github-
token*"
    },
    {
      "Effect": "Allow",
```

```
      "Action": [
        "kms:Decrypt"
      ],
      "Resource": "arn:aws:kms:region:account-id:key/*",
      "Condition": {
        "StringLike": {
          "kms:EncryptionContext:SecretARN": "arn:aws:secretsmanager:region:account-
id:secret:argocd/*",
          "kms:ViaService": "secretsmanager.*.amazonaws.com"
        }
      }
    }
  ]
}
```

> **Note**
>
> Secrets Manager は AWS KMS を使用してすべてのシークレットを暗号化するため、KMS 復号アクセス許可が必要です。この条件により、復号は `argocd/` プレフィックスを持つシークレットのみに制限されます。Secrets Manager でデフォルトの AWS マネージドキーを使用している場合、このアクセス許可で十分です。カスタマーマネージド KMS キーの場合は、使用しているキーの ARN で `Resource` フィールドを更新します。

3. Secrets Manager の認証情報を使用するように Argo CD を設定します。Secrets Manager から Argo CD が参照できる Kubernetes シークレットにシークレットを同期する方法の詳細については、Argo CD ドキュメントの「Secret Management」を参照してください。
4. シークレット ARN を参照する Argo CD リポジトリ設定を作成します。

```
apiVersion: v1
kind: Secret
metadata:
  name: private-repo
  namespace: argocd
  labels:
    argocd.argoproj.io/secret-type: repository
stringData:
  type: git
  url: https://github.com/org/repo
  secretArn: arn:aws:secretsmanager:region:account-id:secret:argocd/github-token
```

Argo CD でリポジトリアクセスを設定する方法の詳細については、「the section called “リポジトリを設定する”」を参照してください。

Security Lake を使用して EKS セキュリティデータを一元化して分析する

Amazon Security Lake は、Amazon EKS を含むさまざまなソースからのセキュリティデータを一元化するフルマネージドのセキュリティデータレイクサービスです。Amazon EKS を Security Lake と統合することで、Kubernetes リソースで実行されるアクティビティに関するより深いインサイトを入手し、Amazon EKS クラスターのセキュリティ体制を強化することができます。

> **Note**
>
> Amazon EKS で Security Lake を使用する方法、およびデータソースを設定する方法の詳細については、Amazon Security Lake のドキュメントを参照してください。

Security Lake を Amazon EKS で使用する利点

セキュリティデータを一元化 - Security Lake は、Amazon EKS クラスターのセキュリティデータを、他の AWS のサービス、SaaS プロバイダー、オンプレミスソース、サードパーティーソースのデータとともに、自動的に収集して一元化します。これにより、組織全体のセキュリティ体制を包括的に把握することができます。

標準化されたデータ形式 — Security Lake は、収集したデータを標準的なオープンソースのスキーマである Open Cybersecurity Schema Framework (OCSF) 形式 に変換します。この標準化により、分析が容易になり、他のセキュリティツールやサービスとの統合が可能になります。

脅威検出の向上 — Amazon EKS コントロールプレーンのログを含む一元化されたセキュリティデータを分析することで、Amazon EKS クラスター内の疑わしいアクティビティをより効果的に検出することができます。これにより、セキュリティインシデントをすみやかに特定し、対応することができます。

データ管理の簡素化 — Security Lake は、カスタマイズ可能な保持とレプリケーションの設定によって、セキュリティデータのライフサイクルを管理します。これにより、データ管理のタスクが簡素化され、コンプライアンスや監査に必要なデータを確実に保持することができます。

Amazon EKS の Security Lake の有効化

1. EKS クラスターの Amazon EKS コントロールプレーンログ記録を有効にします。詳細な手順については、「コントロールプレーンログの有効化と無効化」を参照してください。
2. Amazon EKS 監査ログを Security Lake のソースとして追加します。次に、Security Lake が、EKS クラスターで実行中の Kubernetes リソースで行われたアクティビティの詳細情報の収集を開始します。
3. 要件に基づいて、Security Lake のセキュリティデータの保持とレプリケーションを設定します。
4. Security Lake に保存されている標準化された OCSF データは、インシデント対応、セキュリティ分析、その他の AWS のサービスやサードパーティーツールとの統合に使用できます。例えば、Amazon OpenSearch Ingestion を使用して Amazon Security Lake のデータからセキュリティインサイトを生成 することができます。

Security Lake の EKS ログの分析

Security Lake は EKS ログイベントを OCSF 形式に標準化し、データの分析や、他のセキュリティイベントとの関連付けを簡単に行えるようにします。標準化されたデータは、Amazon Athena、Amazon QuickSight、サードパーティーのセキュリティ分析ツールなどさまざまなツールやサービスを使ってクエリしたり、可視化したりできます。

EKS ログイベントの OCSF マッピングに関する詳細は、OCSF GitHub リポジトリの「mapping reference」を参照してください。

Amazon VPC Lattice との安全なクラスター間接続を有効にする

Amazon VPC Lattice は、AWS ネットワークインフラストラクチャに直接組み込まれた完全マネージド型のアプリケーションネットワーキングサービスで、複数のアカウントや仮想プライベートクラウド (VPC) を通してサービスの接続、保護、モニタリングに使用できます。Amazon EKS では、Kubernetes「ゲートウェイ API」の実装である AWS ゲートウェイ API コントローラーを使用して Amazon VPC Lattice を活用できます。Amazon VPC Lattice を使用すると、標準的な Kubernetes セマンティクスでクラスター間の接続を簡単かつ一貫した方法でセットアップできます。Amazon EKS で Amazon VPC Lattice の使用を開始するには、「AWS ゲートウェイ API コントローラのユーザーガイド」を参照してください。

Amazon EKS Connector を使用して Kubernetes クラスターを Amazon EKS マネジメントコンソールに接続する

Amazon EKS Connector を使用して、準拠する Kubernetes クラスターを AWS に登録および接続し、Amazon EKS コンソールで視覚化できます。クラスターが接続されると、Amazon EKS コンソールでクラスターのステータス、設定、およびワークロードを確認できます。この機能を使用して Amazon EKS コンソールで接続されたクラスターを表示できますが、管理することはできません。Amazon EKS Connector には、GitHub 上のオープンソースプロジェクトであるエージェントが必要です。よくある質問やトラブルシューティングなどのその他の技術コンテンツについては、「the section called “EKS Connector をトラブルシューティングする”」を参照してください。

Amazon EKS Connector は、以下のタイプの Kubernetes クラスターを Amazon EKS に接続できます。

- オンプレミス Kubernetes クラスター
- Amazon EC2 で実行されるセルフマネージド型クラスター
- 他のクラウドプロバイダーのマネージド型クラスター

Amazon EKS Connector の考慮事項

Amazon EKS Connector を使用する前に、次のことを理解してください。

- Amazon EKS にクラスターを接続するためには、Kubernetes クラスターに対する管理者権限が必要です。
- Kubernetes クラスターには、接続する前に Linux 64 ビット (x86) ワーカーノードが存在している必要があります。ARM ワーカーノードはサポートされていません。
- Kubernetes クラスターには、`ssm.` および `ssmmessages.` Systems Manager エンドポイントへのアウトバウンドアクセスを持つワーカーノードが必要です。詳細については、 AWS 全般のリファレンスの「Systems Manager エンドポイント」を参照してください。
- デフォルトでは、1 つのリージョンで最大 10 個のクラスターを接続できます。Service Quotas コンソールを使用して引き上げをリクエストできます。詳細については、「クォータの引き上げのリクエスト」を参照してください。
- Amazon EKS の みRegisterCluster、ListClusters、DescribeCluster、DeregisterClusterAPI は外部 Kubernetes クラスターでサポートされています。

- クラスターを登録するには、次のアクセス許可が必要です。
  - eks:RegisterCluster
  - ssm:CreateActivation
  - ssm:DeleteActivation
  - iam:PassRole
- クラスターを登録解除するには、次のアクセス許可が必要です。
  - eks:DeregisterCluster
  - ssm:DeleteActivation
  - SSM: DeregisterManagedInstance

Amazon EKS コネクターの必要な IAM ロール

Amazon EKS Connector を使用するには、次の 2 つの IAM ロールが必要です。

- 初めてクラスターを登録すると、Amazon EKS Connector サービスにリンクされたロールが作成されます。
- Amazon EKS Connector エージェントの IAM ロールを作成する必要があります。詳細については、「the section called “コネクタ IAM ロール”」を参照してください。

IAM プリンシパルのクラスターおよびワークロードの表示許可を有効にするには、`eks-connector` および Amazon EKS Connector クラスターのロールをクラスターに適用します。「Amazon EKS コンソールで Kubernetes クラスターのリソースを表示するためのアクセスを付与する」の手順を行います。

外部 Kubernetes クラスターを Amazon EKS マネジメントコンソールに接続する

以下のプロセスで複数のメソッドを使用して、外部の Kubernetes クラスターを Amazon EKS に接続できます。このプロセスには 2 つのステップがあります。Amazon EKS へのクラスターの登録と、クラスターへの `eks-connector` エージェントのインストールです。

> **Important**
>
> 最初のステップを完了してから 3 日以内、つまり登録の有効期限が切れる前に 2 番目のステップを完了する必要があります。

考慮事項

エージェントをインストールする際には YAML マニフェストを使用できます。または、AWS マネジメントコンソール または AWS コマンドラインインターフェイスでクラスターを登録する場合は、Helm を使用できます。ただし、`eksctl` でクラスターを登録する場合は、Helm を使用してエージェントをインストールすることはできません。

前提条件

- Amazon EKS Connector エージェントロールが作成されたことを確認します。Amazon EKS コネクタエージェントロールの作成のステップに従います。
- クラスターを登録するには、次のアクセス許可が必要です。
  - `eks:RegisterCluster`
  - `ssm:CreateActivation`
  - `ssm:DeleteActivation`
  - `iam:PassRole`

ステップ 1: クラスターの登録

Amazon EKS Connector にクラスターを登録するには、次のいずれかのツールを使用できます。

- the section called “AWS CLI”
- the section called “AWS マネジメントコンソール”
- the section called “`eksctl`”

AWS CLI

1. AWS CLI がインストールされている必要があります。インストールまたはアップグレードするには、「AWS CLI のインストール」を参照してください。

2. コネクタ設定には、Amazon EKS Connector エージェントの IAM ロールを指定します。詳細については、「the section called “Amazon EKS コネクターの必要な IAM ロール”」を参照してください。

```
aws eks register-cluster \
     --name my-first-registered-cluster \
     --connector-config roleArn=arn:aws:iam::111122223333:role/
AmazonEKSConnectorAgentRole,provider="OTHER" \
     --region aws-region
```

出力例は次のとおりです。

```
{
    "cluster": {
        "name": "my-first-registered-cluster",
        "arn": "arn:aws:eks:region:111122223333:cluster/my-first-registered-cluster",
        "createdAt": 1627669203.531,
        "ConnectorConfig": {
            "activationId": "xxxxxxxxACTIVATION_IDxxxxxxxx",
            "activationCode": "xxxxxxxxACTIVATION_CODExxxxxxxx",
            "activationExpiry": 1627672543.0,
            "provider": "OTHER",
            "roleArn": "arn:aws:iam::111122223333:role/AmazonEKSConnectorAgentRole"
        },
        "status": "CREATING"
    }
}
```

次のステップで `aws-region`、`activationId`、`activationCode` の値を使用します。

AWS マネジメントコンソール

1. Amazon EKS コンソールを開きます。
2. [クラスターを追加] を選択し、[登録] を選択して設定ページを表示します。
3. [クラスターの設定] セクションで、次のフィールドに入力します。
   - [名前] – クラスターの一意の名前。
   - [プロバイダー] 選択すると、Kubsernetes クラスタープロバイダーのドロップダウンリストが表示されます。特定のプロバイダが不明な場合は、[その他] を選択します。

- EKS Connector のロール – クラスターの接続に使用するロールを選択します。

4. [クラスターを登録] を選択します。
5. [クラスター] 概要ページが表示されます。Helm チャートを使用する場合は、`helm install` コマンドをコピーして次のステップに進みます。YAML マニフェストを使用する場合は、[YAML ファイルをダウンロード] を選択して、マニフェストファイルをローカルドライブにダウンロードします。

> **Important**
>
> これは、`helm install` コマンドをコピーする、またはこのファイルをダウンロードする唯一の機会です。このページから移動しないでください。リンクにアクセスできないので、クラスターの登録を解除し、最初から手順を開始する必要があります。

コマンドまたはマニフェストファイルは、登録されたクラスターに対して一度だけ使用できます。Kubernetes クラスターからリソースを削除する場合は、クラスターを再登録し、新しいマニフェストファイルを取得する必要があります。

次のステップに進み、Kubernetes クラスターにマニフェストファイルを適用します。

`eksctl`

1. `eksctl` バージョン `0.68` 以降がインストールされている必要があります。インストール、またはアップグレードをする場合は「the section called “クラスターを作成する (eksctl)”」を参照してください。
2. 名前、プロバイダー、およびリージョンを指定して、クラスターを登録します。

```
eksctl register cluster --name my-cluster --provider my-provider --region region-code
```

出力例:

```
2021-08-19 13:47:26 [#]  creating IAM role "eksctl-20210819194112186040"
2021-08-19 13:47:26 [#]  registered cluster "<name>" successfully
2021-08-19 13:47:26 [#]  wrote file eks-connector.yaml to <current directory>
2021-08-19 13:47:26 [#]  wrote file eks-connector-clusterrole.yaml to <current
 directory>
```

```
2021-08-19 13:47:26 [#]  wrote file eks-connector-console-dashboard-full-access-
group.yaml to <current directory>
2021-08-19 13:47:26 [!]  note: "eks-connector-clusterrole.yaml" and "eks-connector-
console-dashboard-full-access-group.yaml" give full EKS Console access to IAM
 identity "<aws-arn>", edit if required; read https://eksctl.io/usage/eks-connector
 for more info
2021-08-19 13:47:26 [#]  run `kubectl apply -f eks-connector.yaml,eks-connector-
clusterrole.yaml,eks-connector-console-dashboard-full-access-group.yaml` before
 expiry to connect the cluster
```

これにより、ローカルコンピュータ上にファイルが作成されます。これらのファイルは 3 日以内に外部クラスターに適用する必要があります。そうしないと登録の有効期限が切れます。

3. クラスターにアクセスできるターミナルで、生成されたファイルを適用します。

```
kubectl apply -f eks-connector.yaml,eks-connector-clusterrole.yaml,eks-connector-
console-dashboard-full-access-group.yaml
```

ステップ 2: **eks-connector** エージェントをインストールする

eks-connector エージェントをインストールするには、次のいずれかのツールを使用します。

- the section called “Helm”
- the section called “YAML”

Helm

> **Note**
>
> eksctl でクラスターを登録した場合は、Helm チャートの方法の代わりに YAML マニフェストの方法を使用します。

1. 前のステップで AWS CLI を使用した場合は、次のコマンドで ACTIVATION_CODE と ACTIVATION_ID をそれぞれ activationId と activationCode の値に置き換えます。aws-region を前のステップで使用した AWS リージョンに置き換えます。その後、次のコマンドを実行して、eks-connector エージェントを登録済みのクラスターにインストールします。

```
$ helm install eks-connector \
  --namespace eks-connector \
  oci://public.ecr.aws/eks-connector/eks-connector-chart \
  --set eks.activationCode=ACTIVATION_CODE \
  --set eks.activationId=ACTIVATION_ID \
  --set eks.agentRegion=aws-region
```

前のステップで AWS マネジメントコンソールを使用した場合は、前のステップからコピーした、以下の値が入力されたコマンドを使用します。

2. インストールされている `eks-connector` デプロイの正常性をチェックし、Amazon EKS に登録されたクラスターのステータスが `ACTIVE` になるのを待ちます。

YAML

Kubernetes クラスターに Amazon EKS コネクタマニフェストファイルを適用して、接続を完了します。これを行うには、前述のメソッドを使用する必要があります。マニフェストが 3 日以内に適用されない場合、Amazon EKS Connector の登録の有効期限が切れます。クラスター接続が期限切れになった場合は、クラスターを再度接続する前にクラスターの登録を解除する必要があります。

1. Amazon EKS コネクタ YAML ファイルをダウンロードします。

```
curl -O https://amazon-eks.s3.us-west-2.amazonaws.com/eks-connector/manifests/eks-connector/latest/eks-connector.yaml
```

2. Amazon EKS Connector の YAML ファイルを編集して、`%AWS_REGION%`、`%EKS_ACTIVATION_ID%`、`%EKS_ACTIVATION_CODE%` のすべてのリファレンスを、前のステップの出力から得られた `aws-region`、`activationId`、`activationCode` に置き換えます。

   次のコマンド例では、これらの値を置き換えることができます。

```
sed -i "s~%AWS_REGION%~$aws-region~g; s~%EKS_ACTIVATION_ID%~$EKS_ACTIVATION_ID~g; s~%EKS_ACTIVATION_CODE%~$(echo -n $EKS_ACTIVATION_CODE | base64)~g" eks-connector.yaml
```

> **Important**
>
> アクティベーションコードが base64 形式であることを確認します。

3. クラスターにアクセスできるターミナルで、次のコマンドを実行して、更新されたマニフェストファイルを適用できます。

```
kubectl apply -f eks-connector.yaml
```

4. Amazon EKS Connector のマニフェストとロールバインディング YAML ファイルが Kubernetes クラスターに適用されたら、クラスターが接続されたことを確認します。

```
aws eks describe-cluster \
     --name "my-first-registered-cluster" \
     --region AWS_REGION
```

   出力には `status=ACTIVE` が含まれている必要があります。

5. (オプション) クラスターにタグを追加します。詳細については、「the section called “リソースのタグ付け”」を参照してください。

次のステップ

これらのステップで問題が発生した場合は、「the section called “EKS Connector をトラブルシューティングする”」を参照してください。

追加の IAM プリンシパルに Amazon EKS コンソールへのアクセスを許可して、接続されているクラスター内の Kubernetes リソースを表示するには、「the section called “クラスターに対するアクセスを許可する”」を参照してください。

Amazon EKS コンソールで Kubernetes クラスターのリソースを表示するためのアクセスを付与する

IAM プリンシパルに Amazon EKS コンソールへのアクセスを許可して、接続されたクラスターで実行されている Kubernetes リソースに関する情報を表示します。

前提条件

AWS マネジメントコンソール にアクセスするために使用する IAM プリンシパルは、次の要件を満たしている必要があります。

- `eks:AccessKubernetesApi` IAM アクセス許可が必要です。

- Amazon EKS Connector サービスアカウントが、クラスター内の IAM プリンシパルを偽装できる。これにより、Amazon EKS Connector は IAM プリンシパルを Kubernetes ユーザーにマッピングできます。

Amazon EKS Connector クラスターロールを作成して適用するには

1. `eks-connector` クラスターのロールテンプレートをダウンロードします。

```
curl -O https://s3.us-west-2.amazonaws.com/amazon-eks/eks-connector/manifests/eks-connector-console-roles/eks-connector-clusterrole.yaml
```

2. クラスターロールテンプレート YAML ファイルを編集します。`%IAM_ARN%` のリファレンスを IAM プリンシパルの Amazon リソースネーム (ARN) に置き換えます。
3. Amazon EKS コネクタクラスターロール YAML を Kubernetes クラスターに適用します。

```
kubectl apply -f eks-connector-clusterrole.yaml
```

IAM プリンシパルが Amazon EKS コンソールで Kubernetes リソースを確認するには、そのプリンシパルが、リソースを読み取るために必要な許可を持つ Kubernetes `role` または `clusterrole` に関連付けられている必要があります。詳細についてはKubernetes ドキュメントの「RBAC 認可を使用する」を参照してください。

接続されたクラスターにアクセスするように IAM プリンシパルを設定するには

1. 以下のいずれかのサンプルマニフェストファイルをダウンロードして、それぞれ `clusterrole` および `clusterrolebinding`、または `role` および `rolebinding` を作成できます。

   すべての名前空間の Kubernetes リソースを表示する

   - クラスターのロール `eks-connector-console-dashboard-full-access-clusterrole` は、コンソールで視覚化できるすべての名前空間とリソースへのアクセスを提供します。クラスターに適用する前に `role`、`clusterrole`、および対応するバインディングの名前を変更することができます。次のコマンドを使用して、サンプルファイルをダウンロードします。

     ```
     curl -O https://s3.us-west-2.amazonaws.com/amazon-eks/eks-connector/manifests/eks-connector-console-roles/eks-connector-console-dashboard-full-access-group.yaml
     ```

特定の名前空間の Kubernetes リソースを表示する

- このファイルの名前空間は `default` です。別の名前空間を指定する場合は、クラスターに適用する前にファイルを編集します。次のコマンドを使用して、サンプルファイルをダウンロードします。

```
curl -O https://s3.us-west-2.amazonaws.com/amazon-eks/eks-connector/manifests/eks-connector-console-roles/eks-connector-console-dashboard-restricted-access-group.yaml
```

2. フルアクセスまたは制限付きアクセスの YAML ファイルを編集して、`%IAM_ARN%` のリファレンスを IAM プリンシパルの Amazon リソースネーム (ARN) に置き換えます。
3. フルアクセスまたは制限付きアクセスの YAML ファイルを Kubernetes クラスターに適用します。YAML ファイルの値は独自の値に置き換えます。

```
kubectl apply -f eks-connector-console-dashboard-full-access-group.yaml
```

接続されたクラスター内の Kubernetes リソースを表示するには、「the section called “クラスターリソースにアクセスする”」を参照してください。[リソース] タブ上の一部のリソースタイプのデータは、接続されたクラスターでは使用できません。

Amazon EKS コンソールから Kubernetes クラスターの登録を解除する

接続されたクラスターの使用が終了したら、登録を解除できます。登録解除されると、クラスターは Amazon EKS コンソールに表示されなくなります。

deregisterCluster API を呼び出すには、次のアクセス許可が必要です。

- `eks:DeregisterCluster`
- `ssm:DeleteActivation`
- `ssm:DeregisterManagedInstance`

このプロセスには 2 つのステップがあります。Amazon EKS でのクラスターの登録解除と、クラスター内の eks-connector エージェントのアンインストールです。

Kubernetes クラスターの登録を解除する

Amazon EKS Connector からクラスターの登録を解除するには、次のいずれかのツールを使用できます。

- the section called “AWS CLI”
- the section called “AWS マネジメントコンソール”
- the section called “`eksctl`”

AWS CLI

1. AWS CLI がインストールされている必要があります。インストールまたはアップグレードするには、「AWS CLI のインストール」を参照してください。
2. Amazon EKS Connector エージェントロールが作成されたことを確認します。
3. 接続されているクラスタの登録を解除します。

```
aws eks deregister-cluster \
    --name my-cluster \
    --region region-code
```

AWS マネジメントコンソール

1. Amazon EKS コンソールを開きます。
2. [クラスター] を選択します。
3. リポジトリの[クラスター]ページで、接続されているクラスターを選択し、[登録解除]を選択します。
4. クラスターの登録を解除することを確認します。

`eksctl`

1. `eksctl` バージョン `0.68` 以降をインストールします。インストール、またはアップグレードをする場合は「the section called “クラスターを作成する (eksctl)”」を参照してください。
2. Amazon EKS Connector エージェントロールが作成されたことを確認します。
3. 接続されているクラスターの登録を解除します。

```
eksctl deregister cluster --name my-cluster
```

Kubernetes クラスターのリソースをクリーンアップする

eks-connector エージェントをアンインストールするには、次のいずれかのツールを使用します。

- the section called “Helm”
- the section called “YAML”

Helm

次のコマンドを実行して、エージェントをアンインストールします。

```
helm -n eks-connector uninstall eks-connector
```

YAML

1. Kubernetes クラスターから Amazon EKS コネクタ YAML ファイルを削除します。

   ```
   kubectl delete -f eks-connector.yaml
   ```

2. 追加の IAM プリンシパルがクラスターにアクセスするために clusterrole または clusterrolebindings を作成した場合は、それらを Kubernetes クラスターから削除します。

Amazon EKS Connector の問題をトラブルシューティングする

このトピックでは、Amazon EKS Connector の使用中に発生する可能性があるいくつかの一般的なエラーについて説明します。これには、解決方法および回避策も含まれています。

基本的なトラブルシューティング

このセクションでは、Amazon EKS Connector の問題を診断する手順について説明します。

Amazon EKS Connector の状態を確認する

Amazon EKS Connector の状態を確認するには、次のように入力します。

```
kubectl get pods -n eks-connector
```

Amazon EKS Connector のログを検査します

Amazon EKS Connector のポッドは 3 つのコンテナで構成されています。これらすべてのコンテナの完全なログを取得して検査できるようにするには、次のコマンドを実行します。

- `connector-init`

```
kubectl logs eks-connector-0 --container connector-init -n eks-connector
kubectl logs eks-connector-1 --container connector-init -n eks-connector
```

- `connector-proxy`

```
kubectl logs eks-connector-0 --container connector-proxy -n eks-connector
kubectl logs eks-connector-1 --container connector-proxy -n eks-connector
```

- `connector-agent`

```
kubectl exec eks-connector-0 --container connector-agent -n eks-connector -- cat /
var/log/amazon/ssm/amazon-ssm-agent.log
kubectl exec eks-connector-1 --container connector-agent -n eks-connector -- cat /
var/log/amazon/ssm/amazon-ssm-agent.log
```

有効なクラスター名を取得する

Amazon EKS クラスターは、単一の AWS アカウントおよび AWS リージョン内で `clusterName` により一意に識別されます。Amazon EKS に接続されたクラスターが複数ある場合、現在の Kubernetes クラスターが登録されている Amazon EKS クラスターを確認できます。これを行うには、次のコマンドを入力して、現在のクラスターの `clusterName` を確認します。

```
kubectl exec eks-connector-0 --container connector-agent -n eks-connector \
  -- cat /var/log/amazon/ssm/amazon-ssm-agent.log | grep -m1 -oE "eks_c:[a-zA-Z0-9_-]+"
 | sed -E "s/^.*eks_c:([a-zA-Z0-9_-]+)_[a-zA-Z0-9]+.*$/\1/"
kubectl exec eks-connector-1 --container connector-agent -n eks-connector \
  -- cat /var/log/amazon/ssm/amazon-ssm-agent.log | grep -m1 -oE "eks_c:[a-zA-Z0-9_-]+"
 | sed -E "s/^.*eks_c:([a-zA-Z0-9_-]+)_[a-zA-Z0-9]+.*$/\1/"
```

その他のコマンド

次のコマンドは、問題のトラブルシューティングに必要な情報の取得に役立ちます。

- Amazon EKS Connector のポッドで使用されるイメージを収集するには、以下のコマンドを使用します。

```
kubectl get pods -n eks-connector -o jsonpath="{.items[*].spec.containers[*].image}"
 | tr -s '[[:space:]]' '\n'
```

- Amazon EKS Connector が実行されているノード名を確認するには、次のコマンドを使用します。

```
kubectl get pods -n eks-connector -o jsonpath="{.items[*].spec.nodeName}" | tr -s
 '[[:space:]]' '\n'
```

- Kubernetes クライアントおよびサーバーのバージョンを取得するには、次のコマンドを実行します。

```
kubectl version
```

- ノードに関する情報を取得するには、次のコマンドを実行します。

```
kubectl get nodes -o wide --show-labels
```

Helm の問題: 403 Forbidden

Helm インストールコマンドの実行中に以下のエラーが表示された場合:

```
Error: INSTALLATION FAILED: unexpected status from HEAD request to https://
public.ecr.aws/v2/eks-connector/eks-connector-chart/manifests/0.0.6: 403 Forbidden
```

以下の行を実行して修正できます。

```
docker logout public.ecr.aws
```

コンソールのエラー: クラスターが Pending 状態でスタックしています

クラスターの登録後、クラスターが Amazon EKS コンソールで Pending 状態のままスタックしている場合、Amazon EKS Connector がクラスターを AWS に正常に接続しなかったことが原因である可能性があります。登録済みのクラスターの場合、Pending 状態は接続がまだ正常に確立されていないことを意味します。この問題を解決するには、ターゲットの Kubernetes クラスターにマニフェストを適用していることを確認します。クラスターにマニュフェストを適用したにもかかわらず、まだ Pending 状態である場合、eks-connector statefulset に異常がある可能性があります。この問題をトラブルシューティングするには、このトピックの the section called “Amazon EKS Connector Pod がクラッシュループしている” を参照してください。

コンソールのエラー: ユーザーシステム:serviceaccount:eks-connector:eks-connector can’t impersonate resource users in API group at cluster scope (eks-connector はクラススコープで API グループのリソースユーザーを偽装できません)

Amazon EKS Connector は、Kubernetes ユーザー偽装を使用して AWS マネジメントコンソール から IAM プリンシパルに代わってアクションを実行します。AWS eks-connector のサービスアカウントから Kubernetes API にアクセスする各プリンシパルについては、Kubernetes ユーザー名として IAM ARN で対応する Kubernetes ユーザーを偽装するために許可を付与する必要があります。次の例では、Kubernetes ユーザーに IAM ARN がマッピングされています。

- AWS アカウント *111122223333* からの IAM ユーザー *john* は、Kubernetes ユーザーにマッピングされています。IAM のベストプラクティスでは、ユーザーではなくロールに許可を付与することが推奨されています。

```
arn:aws:iam::111122223333:user/john
```

- AWS アカウント *111122223333* からの IAM ロール*###*は、Kubernetes ユーザーにマッピングされています。

```
arn:aws:iam::111122223333:role/admin
```

結果は、AWS STS セッションの ARN ではなく IAM ロールの ARN になります。

マッピングされたユーザーを偽装するアカウント権限を eks-connector サービスに付与するために ClusterRole および ClusterRoleBinding を設定する方法については、「the section called “クラスターに対するアクセスを許可する”」を参照してください。テンプレートで、%IAM_ARN% が AWS マネジメントコンソール IAM プリンシパルの IAM ARN に置き換えられていることを確認してください。

コンソールのエラー: […] is forbidden: User […] cannot list resource […] in API group at the cluster scope ([…] は禁止されています: ユーザー […] が、クラススコープで API グループのリソース […] をリストできません)

次の問題を検討します。Amazon EKS Connector により、ターゲットの Kubernetes クラスターでリクエストしている AWS マネジメントコンソール IAM プリンシパルが正常に偽装されました。ただし、偽装されたプリンシパルには Kubernetes API オペレーション用の RBAC 許可がありません。

この問題を解決するには、追加のユーザーに権限を付与する方法が 2 つあります。以前に Helm チャート 経由で eks-connector をインストールしたことがある場合は、以下のコマンドを実行することでユーザーにアクセス権を簡単に付与できます。userARN1 と userARN2 を IAM ロールの ARN のリストに置き換えて、Kubernetes リソース表示のアクセス権を付与します。

```
helm upgrade eks-connector oci://public.ecr.aws/eks-connector/eks-connector-chart \
    --reuse-values \
    --set 'authentication.allowedUserARNs={userARN1,userARN2}'
```

または、クラスターの管理者として個々の Kubernetes ユーザーに適切なレベルの RBAC 権限を付与します。詳細な説明と例については、the section called “クラスターに対するアクセスを許可する” を参照してください。

コンソールのエラー: Amazon EKS can't communicate with your Kubernetes cluster API server. 正常に接続するには、クラスターが ACTIVE 状態である必要があります。数分後にもう一度お試しください。

Amazon EKS サービスがターゲットクラスター内の Amazon EKS Connector と通信できない場合は、次のいずれかの原因が考えられます。

- ターゲットクラスターの Amazon EKS Connector に異常があります。
- ターゲットクラスターと AWS リージョン間の接続が悪い、または中断している。

この問題を解決するには、Amazon EKS Connector ログを確認してください。Amazon EKS Connector のエラーが表示されない場合、数分後に接続を再試行してください。ターゲットクラスターで高レイテンシーや接続が断続的になる状況が定期的に発生する場合は、近くにある AWS リストにクラスターを再登録することを検討してください。

Amazon EKS Connector Pod がクラッシュループしている

Amazon EKS Connector Pod が `CrashLoopBackOff` 状態になる原因は数多くあります。この問題は `connector-init` コンテナに関係している可能性があります。Amazon EKS Connector Pod のステータスを確認します。

```
kubectl get pods -n eks-connector
```

出力例は次のとおりです。

```
NAME              READY   STATUS                  RESTARTS   AGE
eks-connector-0   0/2     Init:CrashLoopBackOff   1          7s
```

出力が前の出力と似ている場合、the section called “Amazon EKS Connector の口グを検査します” を参照して問題をトラブルシューティングしてください。

eks-connector の起動の失敗: InvalidActivation

Amazon EKS Connector を初めて起動する場合、Amazon Web Services に `activationId` および `activationCode` を登録します。登録が失敗することがあり、次のエラーと同様なエラーで `connector-init` コンテナがクラッシュする原因になります。

```
F1116 20:30:47.261469       1 init.go:43] failed to initiate eks-connector:
 InvalidActivation:
```

この問題のトラブルシューティングを行うには、次の原因および推奨される修正を検討してください。

- `activationId` および `activationCode` がマニフェストファイルにないために、登録が失敗した可能性があります。このような場合、値が `RegisterCluster` API オペレーションから返された正しいものであり、`activationCode` がマニフェストファイルに含まれていることを確認してください。`activationCode` は Kubernetes シークレットに追加されるため、base64 でエンコードする必要があります。詳細については、「the section called “ステップ 1: クラスターの登録”」を参照してください。

- アクティベーションの有効期限が切れたため、登録が失敗した可能性があります。これは、セキュリティ上の理由により、クラスターを登録してから 3 日以内に Amazon EKS Connector をアクティブ化する必要があるためです。この問題を解決するには、Amazon EKS Connector のマニフェストが有効期限の日時よりも前にターゲットの Kubernetes クラスターに適用されていることを確認してください。アクティブ化の有効期限を確認するには、`DescribeCluster` API オペレーションを呼び出します。

```
aws eks describe-cluster --name my-cluster
```

次のレスポンスの例では、有効期限の日時は `2021-11-12T22:28:51.101000-08:00` として記録されています。

```
{
    "cluster": {
        "name": "my-cluster",
        "arn": "arn:aws:eks:region:111122223333:cluster/my-cluster",
        "createdAt": "2021-11-09T22:28:51.449000-08:00",
        "status": "FAILED",
        "tags": {
        },
        "connectorConfig": {
            "activationId": "00000000-0000-0000-0000-000000000000",
            "activationExpiry": "2021-11-12T22:28:51.101000-08:00",
            "provider": "OTHER",
            "roleArn": "arn:aws:iam::111122223333:role/my-connector-role"
        }
    }
}
```

`activationExpiry` が渡されたら、クラスターの登録を解除し、再度登録します。これにより、新しいアクティベーションが生成されます。

クラスターノードにアウトバウンド接続がありません

正常に動作するには、Amazon EKS Connector に複数の AWS エンドポイントへのアウトバウンド接続が必要です。ターゲットの AWS リージョンへのアウトバウンド接続が使用可能でないと、プライベートクラスターを接続することはできません。この問題を解決するには、必要なアウトバウンド接続を追加する必要があります。コネクタの要件については、「the section called “Amazon EKS Connector の考慮事項”」を参照してください。

Amazon EKS Connector Pod が **ImagePullBackOff** 状態になっている

`get pods` コマンドを実行しているときに Pod が `ImagePullBackOff` 状態にある場合、正常に動作しません。Amazon EKS Connector Pod が `ImagePullBackOff` 状態の場合、正常に作動できません。Amazon EKS Connector Pod の状態を確認してください。

```
kubectl get pods -n eks-connector
```

出力例は次のとおりです。

```
NAME              READY   STATUS                  RESTARTS   AGE
eks-connector-0   0/2     Init:ImagePullBackOff   0          4s
```

デフォルトの Amazon EKS Connector のマニフェストファイルは「Amazon ECR Public Gallery」からイメージを参照します。ターゲットの Kubernetes クラスターが Amazon ECR Public Gallery からイメージをプルできない可能性があります。Amazon ECR Public Gallery のイメージのプルに関する問題を解決するか、選択したプライベートコンテナレジストリ内のイメージのミラーリングを検討してください。

Amazon EKS Connector に関するよくある質問

Q: Amazon EKS Connector の基盤となるテクノロジーはどのように機能しますか?

A: Amazon EKS Connector は AWS Systems Manager (Systems Manager) エージェントに基づいています。Amazon EKS Connector は Kubernetes クラスター上で `StatefulSet` として実行されます。これにより接続が確立され、クラスターの API サーバーと Amazon Web Services 間の通信がプロキシされます。これは、AWS からクラスターが切断されるまで Amazon EKS コンソールにクラスターデータを表示しておくためです。Systems Manager エージェントはオープンソースのプロジェクトです。このプロジェクトについての詳細は、「GitHub project page」を参照してください。

Q: 接続したいオンプレミスの Kubernetes クラスターがあります。接続するにはファイアウォールポートを開く必要がありますか?

A: いいえ。ファイアウォールポートを開く必要はありません。Kubernetes クラスターは、AWS リージョンへのアウトバウンド接続のみを必要とします。AWS のサービスがオンプレミスネットワークのリソースにアクセスすることはありません。Amazon EKS Connector はクラスターで実行され、AWS への接続を開始します。クラスターの登録が完了すると、クラスター上の Kubernetes

API サーバーからの情報を必要とする Amazon EKS コンソールからアクションを開始した後にのみ、AWS により Amazon EKS Connector にコマンドが発行されます。

Q: Amazon EKS Connector によってクラスターから AWS に送信されるデータはどのようなものですか?

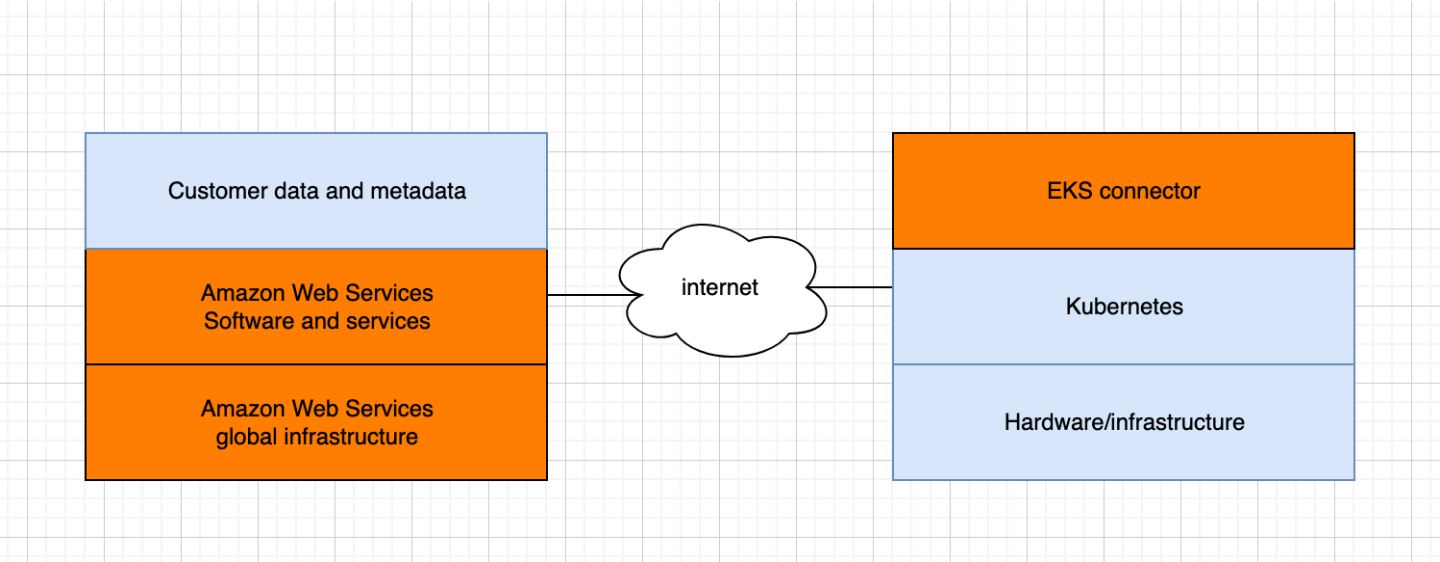
A: Amazon EKS Connector により送信されるのは、AWS でのクラスターの登録に必要な技術情報です。また、お客様がリクエストする Amazon EKS コンソールにおける機能のクラスターとワークロードのメタデータも送信されます。Amazon EKS Connector は、AWS へデータの送信を必要とする Amazon EKS コンソールまたは Amazon EKS API からアクションを開始した場合にのみ、このデータを収集または送信します。デフォルトでは、AWS で Kubernetes のバージョン番号以外のデータが保存されることはありません。許可した場合にのみデータを保存します。

Q: AWS リージョンの外部のクラスターを接続できますか?

A: はい、任意の場所からクラスターを Amazon EKS に接続できます。さらに、Amazon EKS サービスはどのようなパブリック商用 AWS リージョンにも配置できます。これは、クラスターからターゲットの AWS リージョンへの有効なネットワーク接続により動作します。UI パフォーマンスを最適化するために、クラスターの場所に最も近い AWS リージョンを選択することをお勧めします。例えば、東京で実行しているクラスターがある場合、レイテンシーを低くするためにクラスターを東京の AWS リージョン (つまり `ap-northeast-1` AWS リージョン) に接続します。任意の場所から、中国リージョンまたは GovCloud AWS リージョンを除くパブリックな商用 AWS リージョンの Amazon EKS にクラスターを接続できます。

Amazon EKS Connector のセキュリティの説明

Amazon EKS Connector は、Kubernetes クラスターで実行されるオープンソースコンポーネントです。このクラスターは、AWS 環境の外部に配置できます。これにより、セキュリティ上の責任に関する追加の考慮事項が作成されます。この設定については、次の図で示されています。オレンジ色は AWS の責任、青色はお客様の責任を表します。



このトピックでは、接続されているクラスターが AWS の外部にある場合の責任モデルの違いについて説明します。

AWS の責任

- お客様の Kubernetes クラスターで実行され、AWS と通信するオープンソースコンポーネントである Amazon EKS Connector の保守、構築および提供。
- 接続されている Kubernetes クラスターと AWS のサービス間のトランスポートおよびアプリケーションレイヤーの通信セキュリティの維持。

お客様の責任

- Kubernetes クラスター固有のセキュリティの具体例:
  - Kubernetes シークレットが適切に暗号化され、保護されている必要があります。
  - `eks-connector` 名前空間へのアクセスをロックダウンします。
- AWS からの IAM プリンシパルのアクセスを管理するロールベースのアクセスコントロール (RBAC) 許可の設定。手順については、the section called “クラスターに対するアクセスを許可する” を参照してください。
- Amazon EKS Connector のインストールおよびアップグレード。
- 接続されている Kubernetes クラスターをサポートするハードウェア、ソフトウェア、およびインフラストラクチャの保守。
- (例えばルートユーザーの認証情報を保護することを通じて) AWS アカウントを保護します。

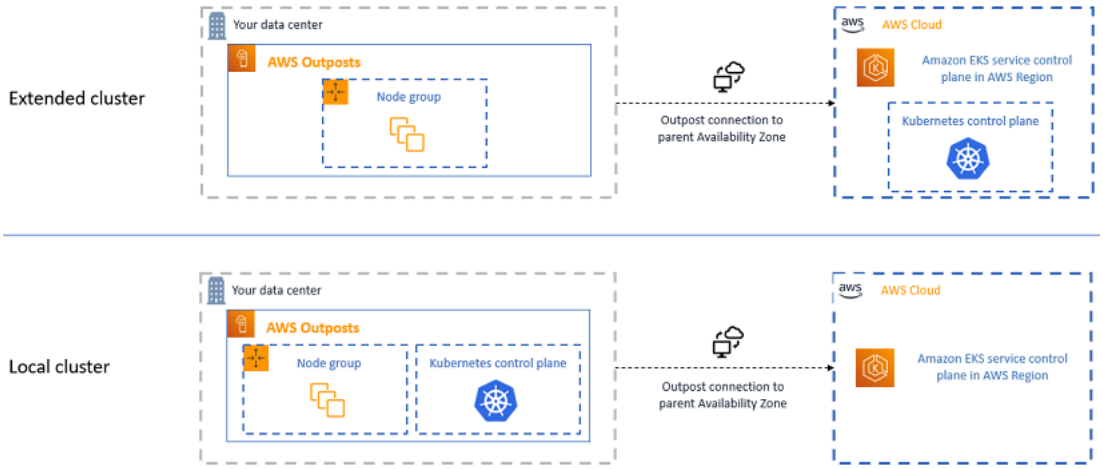
AWS Outposts を使用して Amazon EKS をオンプレミスにデプロイする

Amazon EKS を使用して、オンプレミス Kubernetes アプリケーションを AWS Outposts で実行できます。次の方法で Amazon EKS を Outposts にデプロイできます。

- 拡張クラスター - Outpost の AWS リージョンとノードで Kubernetes コントロールプレーンを実行します。
- ローカルクラスター - Outpost で Kubernetes コントロールプレーンとノードを実行します。

どちらのデプロイオプションでも、Kubernetes コントロールプレーンは、AWS によって完全に管理されています。クラウドで使用するのと同じ Amazon EKS API、ツール、コンソールを使用して Outposts で Amazon EKS を作成および実行することができます。

以下の図に、これらのデプロイのオプションを示します。



各デプロイオプションを使用する時

ローカルクラスターと拡張クラスターはどちらも汎用のデプロイオプションであり、さまざまなアプリケーションに使用できます。

拡張クラスターでは、Kubernetes コントロールプレーンが親 AWS リージョンで動作するため、Outpost の容量を節約できます。このオプションは、Outposts から AWS リージョンに対して信頼性の高いネットワーク接続を確立している場合に適しています。AWS リージョンからネットワークを切断することで静的安定性を確保するようにアプリケーションを設計することをお勧めします。Kubernetes が Kubernetes コントロールプレーンとノード間のネットワーク切断を処理する方

法によって、アプリケーションのダウンタイムが発生する可能性があります。Kubernetes の動作の詳細については、Kubernetes ドキュメントの「Scheduling, Preemption, and Eviction」を参照してください。

ローカルクラスターでは、Amazon EKS クラスター全てを Outposts でローカルに実行できます。この方法により、クラウドへの一時的なネットワーク切断によって生じる可能性のあるアプリケーションダウンタイムのリスクを軽減できます。このようなネットワーク接断は、ファイバーの切断や天候によって発生する可能性があります。すべての Amazon EKS クラスター全体が Outposts でローカルに実行されるため、アプリケーションは引き続き使用できます。クラウドへのネットワーク切断中にクラスターオペレーションを実行できます。ネットワークを切断することで Kubernetes を継続的に運用する必要がある場合や、データレジデンシーが義務づけられている場合は、ローカルクラスターを選択します。

2 つのローカルクラスターの実装

Amazon EKS では、Outposts ラックにローカルクラスターを 2 つ実装できるようになっています。Outposts で Amazon EC2 インスタンスを実行する場合は、Amazon EBS または Amazon EC2 インスタンスストアをルートボリュームタイプとして使用するように Outposts を設定します。Amazon EKS がローカルクラスターのコントロールプレーンに使用する実装は、この設定によって異なります。

- AWS Outposts に Amazon EBS を設定します。Kubernetes コントロールプレーンは、Outposts の AWS アカウントで実行されます。`etcd` は、Amazon EBS がサポートする 3 つの Amazon EC2 インスタンスにスタックされます。詳細については、「the section called “ローカルクラスタを実行する”」を参照してください。
- AWS Outposts に EC2 インスタンスストアを設定します。Amazon EKS は、最新のローカルクラスターアーキテクチャをデプロイします。Kubernetes コントロールプレーンは、Outposts の AWS マネージドサービスアカウントで動作する 6 つの Amazon EC2 インスタンスで実行されます。このうち、3 つのインスタンスは `etcd` 専用で、残り 3 つのインスタンスは他のコントロールプレーンコンポーネント専用です。クラウド内の Amazon EKS との同等性を向上させた実装であり、Amazon EKS Kubernetes バージョンのライフサイクルに準拠し、AL2023 の他に Amazon EKS アドオンや Bottlerocket ベースのワーカーノードなどの追加機能にも対応しています。AWS Outposts が利用可能なすべての標準 AWS リージョンで使用できます。詳細については、「the section called “ローカルクラスターを実行する (EC2 インスタンスストア)”」を参照してください。

次の表に、各実装で使用できる機能をまとめます。

デプロイオプションの比較

次の表は、拡張クラスターの実装と 2 つのローカルクラスターの実装との違いを比較したものです。

機能	拡張クラスター	Outposts に EBS があるローカルクラスター	Outposts に EC2 インスタンスストアがあるローカルクラスター
Kubernetes コントロールプレーンの場所	AWS リージョン	Outpost	Outpost
Kubernetes コントロールプレーンのアカウント	AWS アカウント	AWS アカウント	AWS マネージドアカウント
Kubernetes コントロールプレーントポロジ	クラウドマネージド	3 つの Amazon EC2 インスタンス、etcd をスタック	6 つの Amazon EC2 インスタンス (3 つの `etcd` + 3 つの API サーバー)
コントロールプレーンストレージ	クラウドマネージド	Amazon EBS	Amazon EC2 インスタンスストア
リージョン別の可用性	「サービスエンドポイント」を参照してください	米国東部 (バージニア北部)、米国東部 (オハイオ)、米国西部 (北カリフォルニア)、米国西部 (オレゴン)、アジアパシフィック (ソウル)、アジアパシフィック (シンガポール)、アジアパシフィック (東京)、アジアパシフィック (シドニー)、カナダ (中	AWS Outposts が利用可能なすべての商用 AWS リージョン

機能	拡張クラスター	Outposts に EBS があるローカルクラスター	Outposts に EC2 インスタンスストアがあるローカルクラスター
		部)、欧州 (フランクフルト)、欧州 (アイルランド)、欧州 (ロンドン)、中東 (バーレーン)、南米 (サンパウロ)	
Kubernetes マイナーバージョン	サポートされている Amazon EKS バージョン	「the section called “EKS プラットフォームのバージョン”」を参照してください。	サポートされている Amazon EKS バージョン
プラットフォームバージョン	「Amazon EKS のプラットフォームバージョン」を参照	「the section called “EKS プラットフォームのバージョン”」を参照してください。	「Amazon EKS のプラットフォームバージョン」を参照
Outpost フォームファクター	Outpost ラック	Outpost ラック	Outpost ラック
ユーザーインターフェイス	AWS マネジメントコンソール、AWS CLI、Amazon EKS API、`eksctl`、AWS CloudFormation、および Terraform	AWS マネジメントコンソール、AWS CLI、Amazon EKS API、`eksctl`、AWS CloudFormation、および Terraform	AWS マネジメントコンソール、AWS CLI、Amazon EKS API、AWS CloudFormation

機能	拡張クラスター	Outposts に EBS があるローカルクラスター	Outposts に EC2 インスタンスストアがあるローカルクラスター
マネージドポリシー	AmazonEKSClusterPolicy および the section called “AWS マネージドポリシー: AmazonEKSServiceRolePolicy”	AmazonEKSLocalOutpostClusterPolicy および the section called “AWS マネージドポリシー: AmazonEKSLocalOutpostServiceRolePolicy”	AmazonEKSClusterPolicy および the section called “AWS マネージドポリシー: AmazonEKSServiceRolePolicy”
クラスター VPC およびサブネット	「the section called “VPC およびサブネットの要件”」を参照してください。	「the section called “VPC とサブネットを作成する”」を参照してください。	「the section called “VPC とサブネットを作成する (EC2 インスタンスストア)”」を参照してください。
クラスターエンドポイントのアクセス	公共または個人、あるいはその両方	非公開のみ	プライベート、またはパブリックかつプライベート
Kubernetes API サーバー認証	AWS Identity and Access Management (IAM)、OIDC、およびアクセスエントリ	IAM および x.509 証明書	IAM、OIDC、アクセスエントリ、aws-auth ConfigMap、x.509 証明書 (ネットワーク切断の場合)
ノードタイプ	セルフマネージド型のみ	セルフマネージド型のみ	セルフマネージド型のみ
ノードのコンピュートタイプ	Amazon EC2 オンデマンド	Amazon EC2 オンデマンド	Amazon EC2 オンデマンド
ノードストレージタイプ	Amazon EBS gp2 とローカル NVMe SSD	Amazon EBS gp2 とローカル NVMe SSD	ローカル NVMe SSD

機能	拡張クラスター	Outposts に EBS があるローカルクラスター	Outposts に EC2 インスタンスストアがあるローカルクラスター
Amazon EKS 最適化 AMI	Amazon Linux、Windows、Bottlerocket	Amazon Linux	Amazon Linux と Bottlerocket
IP バージョン	IPv4 のみ	IPv4 のみ	IPv4 のみ
アドオン	Amazon EKS アドオンまたはセルフマネージドアドオン	セルフマネージドアドオン	Amazon EKS アドオン (検証済み) またはセルフマネージドアドオン
デフォルトの Container Network Interface	Amazon VPC CNI Kubernetes 用プラグイン	Amazon VPC CNI Kubernetes 用プラグイン	Amazon VPC CNI Kubernetes 用プラグイン
Kubernetes コントロールプレーンのログ	Amazon CloudWatch Logs	Amazon CloudWatch Logs	Amazon CloudWatch Logs
etcd バックアップ	クラウドマネージド	カスタマーマネージド (etcdctl または Amazon EBS ボリュームバックアップ)	Amazon EKS による管理
負荷分散	AWS Load Balancer Controller を使用して Application Load Balancer のみをプロビジョニングします (Network Load Balancer はプロビジョニングしない)	AWS Load Balancer Controller を使用して Application Load Balancer のみをプロビジョニングします (Network Load Balancer はプロビジョニングしない)	AWS Load Balancer Controller を使用して Application Load Balancer のみをプロビジョニングします (Network Load Balancer はプロビジョニングしない)

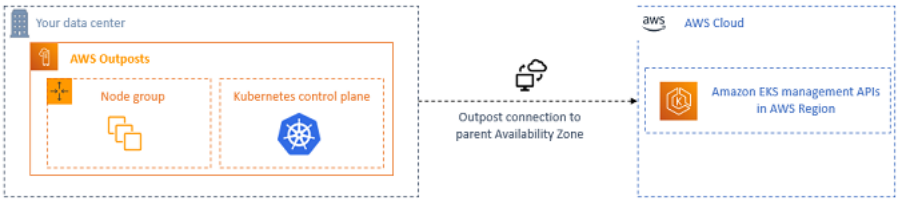
機能	拡張クラスター	Outposts に EBS があるローカルクラスター	Outposts に EC2 インスタンスストアがあるローカルクラスター
シークレットエンベロープ暗号化	サポート対象、「the section called “シークレット暗号化を有効にする”」を参照。	サポートされません	サポートされません
サービスアカウント用 IAM ロール (IRSA、IAM roles for service accounts)	サポート対象、「the section called “IRSA による認証情報”」を参照。	サポートされていません	サポート対象 (AWS リージョンへの接続が必要。ネットワーク切断中は利用不可)
EKS Pod Identity	サポート対象、「the section called “Pod Identity”」を参照。	サポートされていません	サポート対象 (AWS リージョンへの接続が必要。ネットワーク切断中は利用不可)
トラブルシューティング	「トラブルシューティング」を参照してください	「the section called “クラスターのトラブルシューティングを行う”」を参照してください。	「the section called “クラスターをトラブルシューティングする (EC2 インスタンスストア)”」を参照してください。

Important

- RAM 共有の Outposts にローカルクラスターを作成すると、クラスターのコントロールプレーンが Outposts 所有者の物理的アクセスの下に配置されます。
- EC2 インスタンスストア Outposts では、IRSA と EKS Pod Identity は AWS リージョンの AWS STS に依存します。ネットワークの切断中に、IRSA または Pod Identity を使用するワークロードが新しい認証情報を取得することはできません。詳細については、「the section called “切断への備え (EC2 インスタンスストア)”」を参照してください。

高可用性を実現するために AWS Outposts でローカル Amazon EKS クラスターを作成する

ローカルクラスターを使用すると、Amazon EKS クラスターすべてを AWS Outposts でローカルに実行できます。これにより、クラウドへの一時的なネットワーク切断によって生じる可能性のあるアプリケーションダウンタイムのリスクを軽減できます。このような接断は、ファイバーの切断や天候によって発生する可能性があります。Kubernetes クラスター全体が Outposts でローカルに実行されるため、アプリケーションは引き続き使用できます。クラウドへのネットワーク切断中にクラスターオペレーションを実行できます。詳細については、「the section called “切断の準備をする”」を参照してください。次の図は、ローカルクラスターのデプロイを示しています。



ローカルクラスターは一般的に Outposts ラックで使用できます。

サポート対象の AWS リージョン

ローカルクラスターを作成できる AWS リージョンは、米国東部 (オハイオ)、米国東部 (バージニア北部)、米国西部 (北カリフォルニア)、米国西部 (オレゴン)、アジアパシフィック (ソウル)、アジアパシフィック (シンガポール)、アジアパシフィック (シドニー)、アジアパシフィック (東京)、カナダ (中部)、欧州 (フランクフルト)、欧州 (アイルランド)、欧州 (ロンドン)、中東 (バーレーン)、南米 (サンパウロ) です。サポートされる機能の詳細については、「the section called “デプロイオプションの比較”」を参照してください。

AWS Outposts で Amazon EKS クラスターをデプロイする

このトピックでは、Outpost でローカルクラスターを実行する際に考慮すべき事項について、概要を説明します。また、Outpost でローカルクラスターをデプロイする方法についても説明します。

> **Important**
>
> - これらの考慮事項は、関連する Amazon EKS ドキュメントには記載されていません。Amazon EKS ドキュメントに記載されている他のトピックの情報が、ここに紹介する考慮事項と競合する場合は、こちらの考慮事項を優先してください。

- これらの考慮事項は変更される可能性があり、変更は頻繁に行われる場合があります。そのため、このトピックは定期的に確認することをお勧めします。
- 考慮事項の多くは、AWS Cloud でクラスターを作成する場合の考慮事項とは異なります。

- ローカルクラスターは Outpost ラックのみをサポートします。単一のローカルクラスターは、単一の論理 Outpost を構成する複数の物理 Outpost ラックにわたって実行できます。単一のローカルクラスターを複数の論理 Outposts にわたって実行することはできません。各論理 Outpost には、単一の Outpost ARN があります。
- ローカルクラスターは、Outpost のアカウントで Kubernetes コントロールプレーンを実行および管理します。Kubernetes コントロールプレーンインスタンスでワークロードを実行したり、Kubernetes コントロールプレーンのコンポーネントを変更することはできません。これらのノードは Amazon EKS サービスによって管理されます。Kubernetes コントロールプレーンへの変更は、パッチ適用などの自動 Amazon EKS 管理アクションでは存続しません。
- ローカルクラスターは、セルフマネージド型アドオンとセルフマネージド型 Amazon Linux ノードグループをサポートしています。Amazon VPC CNI plugin for Kubernetes、kube-proxy、CoreDNS アドオンはローカルクラスターに自動的にインストールされます。
- ローカルクラスターでは、Outposts で Amazon EBS を使用する必要があります。お客様の Outpost では、Kubernetes コントロールプレーンストレージに使用できる Amazon EBS が必要です。Outposts は、Amazon EBS gp2 ボリュームのみをサポートします。
- Amazon EBS のバックアップ対象である Kubernetes `PersistentVolumes` は、Amazon EBS CSI ドライバーを使用してサポートされています。
- ローカルクラスターのコントロールプレーンインスタンスは、可用性の高いスタックされたトポロジ で設定されます。クォーラムを維持するには、3 つのコントロールプレーンインスタンスのうち 2 つが常に正常である必要があります。クォーラムが失われた場合、新しいマネージドインスタンスを有効にするにはいくつかのサービス側のアクションが必要になるため、AWS サポートにお問い合わせください。

前提条件

- Outposts デプロイオプション、キャパシティに関する考慮事項に基づく AWS Outposts での Amazon EKS クラスターのインスタンスタイプとプレイスメントグループの選択、および VPC 要件と考慮事項に精通していること。
- 既存の アウトポスト。詳細については、「AWS Outposts とは」を参照してください。

- コンピュータまたは AWS CloudShell に、kubectl コマンドラインツールがインストールされていること。バージョンはご使用のクラスターの Kubernetes バージョンと同じか、1 つ前のマイナーバージョン以前、あるいはそれより新しいバージョンが使用できます。例えば、クラスターのバージョンが 1.33 である場合、kubectl のバージョン 1.32、1.33、または 1.34 が使用できます。kubectl をインストールまたはアップグレードする方法については、the section called “kubectl と eksctl のセットアップ” を参照してください。
- ご使用のデバイスまたは AWS CloudShell で、バージョン 2.12.3 以降、または AWS コマンドラインインターフェイス (AWS CLI) のバージョン 1.27.160 以降がインストールおよび設定されていること。現在のバージョンを確認するには、aws --version | cut -d / -f2 | cut -d ' ' -f1 を参照してください。yum、apt-get、macOS 用の Homebrew などのパッケージマネージャーは、多くの場合 AWS CLI の最新バージョンより数バージョン古くなっています。最新バージョンをインストールするには、 AWS コマンドラインインターフェイスユーザーガイドの「インストール」および「aws configure を使用したクイック設定」を参照してください。AWS CloudShell にインストールされている AWS CLI バージョンも最新バージョンより数バージョン遅れることがあります。更新するには、「AWS CloudShell ユーザーガイド」の「ホームディレクトリへの AWS CLI のインストール」を参照してください。
- Amazon EKS クラスターへの create および describe のアクセス許可を持つ IAM プリンシパル (ユーザーまたはロール)。詳細については、the section called “Outpost にローカル Kubernetes クラスターを作成する” および the section called “すべてのクラスターの一覧表示または説明” を参照してください。

ローカルの Amazon EKS クラスターが作成される時点で、クラスターを作成する IAM プリンシパル が恒久的に追加されています。とりわけ、プリンシパルは管理者として、Kubernetes RBAC 承認テーブルに追加されます。このエンティティには system:masters アクセス許可が付与されています。このエンティティの ID は、クラスター設定には表示されません。したがって、クラスターを作成したエンティティをメモし、削除しないようにすることが重要です。初期状態では、サーバーを作成したプリンシパルのみが、kubectl を使用して Kubernetes API サーバーへの呼び出しを実行できます。コンソールを使用してクラスターを作成する場合は、クラスター上で kubectl コマンドを実行する際、同じ IAM 認証情報が AWS SDK 認証情報チェーンにあることを確認する必要があります。クラスターの作成が完了したら、他の IAM プリンシパルにクラスターへのアクセスを付与できます。

Amazon EKS ローカルクラスターを作成します。

このページで説明されている以下のツールを使用して、ローカルクラスターを作成できます。

- the section called “eksctl”

- the section called “AWS マネジメントコンソール”

AWS CLI、Amazon EKS API、AWS SDK、AWS CloudFormation、または Terraform を使用して Outposts にクラスターを作成することもできます。

eksctl

を使用してローカルクラスターを作成するには**eksctl**

1. デバイスまたは AWS CloudShell に `eksctl` コマンドラインツールのバージョン `0.215.0` 以降をインストールします。`eksctl` をインストールまたはアップグレードするには、`eksctl` ドキュメントの「インストール」を参照してください。
2. 次のコンテンツをデバイスにコピーします。次の値を置き換えたら、変更したコマンドを実行して `outpost-control-plane.yaml` ファイルを作成します。
   - *`region-code`* は、クラスターを作成する サポートされている AWS リージョンに置き換えます。
   - *`my-cluster`* の部分は、自分のクラスター名に置き換えます。この名前には英数字 (大文字と小文字が区別されます) とハイフンのみを使用できます。先頭の文字は英数字である必要があります。また、100 文字より長くすることはできません。名前はクラスターを作成する AWS リージョンおよび AWS アカウント内で一意である必要があります。
   - *`vpc-ExampleID1`* と *`subnet-ExampleID1`* を既存の VPC とサブネットの ID に置き換えます。VPC とサブネットは、AWS Outposts で Amazon EKS クラスターの VPC とサブネットを作成する要件を満たす必要があります。
   - *`uniqueid`* を Outpost の ID に置き換えます。
   - *`m5.large`* を Outpost で使用可能なインスタンスタイプに置き換えます。インスタンスタイプを選択する前に、「the section called “容量に関する考慮事項”」を参照してください。3 つのコントロールプレーンインスタンスがデプロイされます。この番号を変更することはできません。

```
cat >outpost-control-plane.yaml <<EOF
apiVersion: eksctl.io/v1alpha5
kind: ClusterConfig

metadata:
  name: my-cluster
  region: region-code
  version: "1.37"
```

```
vpc:
  clusterEndpoints:
    privateAccess: true
  id: "vpc-vpc-ExampleID1"
  subnets:
    private:
      outpost-subnet-1:
        id: "subnet-subnet-ExampleID1"

outpost:
  controlPlaneOutpostARN: arn:aws:outposts:region-code:111122223333:outpost/op-
uniqueid
  controlPlaneInstanceType: m5.large
EOF
```

使用可能なすべてのオプションとデフォルトの完全なリストについては、eksctl ドキュメントの「AWS Outposts サポート」および「設定ファイルスキーマ」を参照してください。

3. 前のステップで作成した設定ファイルを使用してクラスターを作成します。eksctl は、クラスターをデプロイするために VPC と 1 つのサブネットを Outpost に作成します。

```
eksctl create cluster -f outpost-control-plane.yaml
```

クラスターのプロビジョニングには数分かかります。クラスターの作成中は数行の出力が表示されます。出力の最後の行は次のサンプル行のようになります。

```
[#]  EKS cluster "my-cluster" in "region-code" region is ready
```

> **Tip**
>
> eksctl を使用してクラスターを作成するときに指定できるほとんどのオプションを表示するには、eksctl create cluster --help コマンドを使用します。使用可能なオプションをすべて表示するには、config ファイルを使用します。詳細については、eksctl ドキュメントの「設定ファイルの使用」と「設定ファイルのスキーマ」を参照してください。設定ファイルの例は、GitHub で見つけることができます。

eksctl コマンドはクラスターを作成した IAM プリンシパル (ユーザーまたはロール) のアクセスエントリを自動的に作成し、クラスター上の Kubernetes オブジェクトに対するアクセス許可を

IAM プリンシパル管理者に付与しました。クラスター作成者にクラスター上の Kubernetes オブジェクトへの管理者アクセス権を付与したくない場合は、前の設定ファイルに次のテキストを追加します: `bootstrapClusterCreatorAdminPermissions: false` (`metadata`、`vpc`、および `outpost` と同じレベル)。このオプションを追加した場合、クラスターの作成後に少なくとも 1 つの IAM プリンシパルのアクセスエントリを作成する必要があります。そうしないと、IAM プリンシパルはクラスター上の Kubernetes オブジェクトにアクセスできなくなります。

AWS マネジメントコンソール

を使用してクラスターを作成するにはAWS マネジメントコンソール

1. Amazon EKS の要件を満たす既存の VPC とサブネットが必要です。詳細については、the section called “VPC とサブネットを作成する” を参照してください。
2. 既にローカルクラスター IAM ロールがある場合、または `eksctl` を使用してクラスターを作成する場合は、このステップはスキップできます。デフォルトでは `eksctl` により、ロールが自動的に作成されます。

   a. IAM 信頼ポリシー用の JSON ファイルを作成するには次のコマンドを実行してください。

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "ec2.amazonaws.com"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
```

   b. Amazon EKS クラスターの IAM ロールを作成します。IAM ロールを作成するには、ロールを作成する IAM プリンシパル に `iam:CreateRole` アクション (許可) を割り当てる必要があります。

```
aws iam create-role --role-name myAmazonEKSLocalClusterRole --assume-role-policy-
document file://"eks-local-cluster-role-trust-policy.json"
```

c. AmazonEKSLocalOutpostClusterPolicy という名前の Amazon EKS マネージド型ポリシーをロールにアタッチします。IAM ポリシーを IAM プリンシパル にアタッチするにはポリシーのアタッチを行っているプリンシパルに、次のいずれかの IAM アクション (許可) を割り当てる必要があります: `iam:AttachUserPolicy` または `iam:AttachRolePolicy`。

```
aws iam attach-role-policy --policy-arn arn:aws:iam::aws:policy/
AmazonEKSLocalOutpostClusterPolicy --role-name myAmazonEKSLocalClusterRole
```

3. Amazon EKS コンソール を開きます。
4. コンソール画面の上部で、サポートされている AWS リージョン を選択したことを確認します。
5. [クラスターを追加]、[作成] の順にクリックします。
6. [クラスターの設定] ページで、次のフィールドの値を入力または選択します。
   - [Kubernetes コントロールプレーンの場所] – AWS Outposts を選択します。
   - [Outpost ID] - コントロールプレーンを作成する Outpost の ID を選択します。
   - [インスタンスタイプ] - インスタンスタイプを選択します。Outpost で使用可能なインスタンスタイプのみが表示されます。ドロップダウンリストでは、各インスタンスタイプにそのインスタンスタイプに推奨されるノード数が示されています。インスタンスタイプを選択する前に、the section called “容量に関する考慮事項” を参照してください。すべてのレプリカは、同じインスタンスタイプを使用してデプロイされます。クラスターの作成後にインスタンスタイプを変更することはできません。3 つのコントロールプレーンインスタンスがデプロイされます。この番号を変更することはできません。
   - [名前] – クラスターの名前。AWS アカウント内で一意にする必要があります。この名前には英数字 (大文字と小文字が区別されます) とハイフンのみを使用できます。先頭の文字は英数字である必要があります。また、100 文字より長くすることはできません。名前はクラスターを作成する AWS リージョンおよび AWS アカウント内で一意である必要があります。
   - [Kubernetes バージョン] – クラスターで使用する Kubernetes バージョンを選択します。以前のバージョンを使用する必要がない限り、最新バージョンを選択することをお勧めします。
   - [クラスターサービスロール] - AWS リソースを管理することを Kubernetes コントロールプレーンに許可するために、前のステップで作成した Amazon EKS クラスター IAM ロールを選択します。
   - [Kubernetes クラスター管理者アクセス] — クラスターを作成する IAM プリンシパル (ロールまたはユーザー) にクラスター上の Kubernetes オブジェクトへの管理者アクセス権を付与する場合は、デフォルト (許可) をそのまま使用します。Amazon EKS は IAM プリンシパルのアクセスエントリを作成し、クラスター管理者にそのアクセスエントリに対するアクセス許可を付与し

ます。アクセスエントリの詳細については、the section called “アクセスエントリ” を参照してください。

クラスターを作成するプリンシパルとは異なる IAM プリンシパルに Kubernetes クラスターオブジェクトへの管理者アクセス権を付与したい場合は、拒否のオプションを選択します。クラスターの作成後、アクセスエントリを作成する IAM アクセス許可を持つすべての IAM プリンシパルは、Kubernetes クラスターオブジェクトへのアクセスを必要とする任意の IAM プリンシパルのアクセスエントリを追加できます。必要な IAM アクセス許可の詳細については、サービス認可リファレンスの「Amazon Elastic Kubernetes Service で定義されるアクション」を参照してください。拒否のオプションを選択し、アクセスエントリを作成しない場合、IAM プリンシパルはクラスター上の Kubernetes オブジェクトにアクセスできなくなります。

- [タグ] – (オプション) クラスターにタグを追加します。詳細については、the section called “リソースのタグ付け” を参照してください。このページを読み終えたら、[次へ] を選択してください。

7. [ネットワーキングの指定] ページで、次のフィールドの値を選択してください:
   - [VPC] - 既存の VPC を選択します。VPC には、作成するクラスター、ノード、およびその他の Kubernetes リソースで利用できる十分な数の IP アドレスが必要です。VPC は、「VPC の要件と考慮事項」に記載されている要件を満たしている必要があります。
   - [サブネット] – デフォルトで、前のフィールドで指定した VPC 内の利用可能なすべてのサブネットがあらかじめ選択されています。選択するサブネットは、「サブネットの要件と考慮事項」に記載されている要件を満たしている必要があります。
   - [セキュリティグループ] - (オプション) Amazon EKS が作成するネットワークインターフェイスに関連付ける 1 つまたは複数のセキュリティグループを指定します。Amazon EKS は、クラスターと VPC 間の通信を可能にするセキュリティグループを自動的に作成します。Amazon EKS はこのセキュリティグループおよびユーザーが選択したセキュリティグループを、作成するネットワークインターフェイスに関連付けます。Amazon EKS が作成するクラスターセキュリティグループの詳細については、「the section called “セキュリティグループの要件”」を参照してください。Amazon EKS が作成するクラスターセキュリティグループのルールを変更できます。独自のセキュリティグループを追加することを選択した場合、クラスターの作成後に選択したセキュリティグループを変更することはできません。オンプレミスホストがクラスターエンドポイントと通信するためには、クラスターセキュリティグループからのインバウンドトラフィックを許可する必要があります。入出力のインターネット接続がないクラスター (プライベートクラスターとも呼ばれる) の場合は、次のいずれかを行う必要があります。
     - 必要となる VPC エンドポイントに関連付けられたセキュリティグループを追加します。必要となるエンドポイントの詳細については、「AWS サービスへのサブネットアクセス」の「the section called “インターフェイス VPC エンドポイントの使用”」を参照してください。

- VPC エンドポイントに関連付けられたセキュリティグループからのトラフィックを許可するように、Amazon EKS が作成したセキュリティグループを変更します。このページを読み終えたら、[次へ] を選択してください。

8. [オブザーバビリティの設定] ページでは、有効にする [メトリクス] と [コントロールプレーンのロギング] オプションをオプションで選択できます。デフォルトでは、それぞれのログタイプは無効化されています。
   - Prometheus メトリクスの詳細については、「the section called “ステップ 1: Prometheus メトリクスを有効にする”」を参照してください。
   - 「コントロールプレーン」に関する詳細に関しては、the section called “コントロールプレーンのログ” を参照してください。このページを読み終えたら、[次へ] を選択してください。
9. [確認と作成] ページで、前のページで入力または選択した情報を確認します。変更する必要がある場合は[編集] を選択してください。そのままでよければ、[作成] を選択してください。クラスターがプロビジョニングされている間、[ステータス] フィールドに [作成中] と表示されます。

   クラスターのプロビジョニングには数分かかります。

Amazon EKS ローカルクラスターを表示する

1. クラスターを作成すると、作成された Amazon EC2 コントロールプレーンのインスタンスを表示できます。

```
aws ec2 describe-instances --query 'Reservations[*].Instances[*].{Name:Tags[?
Key==`Name`]|[0].Value}' | grep my-cluster-control-plane
```

   出力例は次のとおりです。

```
"Name": "my-cluster-control-plane-id1"
"Name": "my-cluster-control-plane-id2"
"Name": "my-cluster-control-plane-id3"
```

   各インスタンスは node-role.eks-local.amazonaws.com/control-plane で汚染されているため、コントロールプレーンインスタンスでワークロードがスケジュールされることはありません。テイントの詳細については、Kubernetes ドキュメントの「Taints and Tolerations」を参照してください。Amazon EKS はローカルクラスターの状態を継続的に監視します。セキュリティパッチや問題のあるインスタンスの修復などの自動管理アクションを実行します。ローカルクラ

スターがクラウドから切断されると、再接続時にクラスターが正常な状態に修復されるようアクションが実行されます。

2. eksctl を使用してクラスターを作成した場合、このステップはスキップできます。eksctl がこのステップを完了します。新しいコンテキストを kubectl config ファイルに追加して、kubectl がクラスターと通信できるようにします。ファイルを作成し更新する手順については、the section called “kubectl を使用してクラスターにアクセスする” を参照しください。

```
aws eks update-kubeconfig --region region-code --name my-cluster
```

出力例は次のとおりです。

```
Added new context arn:aws:eks:region-code:111122223333:cluster/my-cluster to /home/
username/.kube/config
```

3. ローカルクラスターの Kubernetes API サーバーに接続するには、サブネットのローカルゲートウェイにアクセスするか、VPC 内から接続します。Outpost ラックをオンプレミスネットワークに接続する方法の詳細については、「AWS Outposts ユーザーガイド」の「How local gateways for racks work」(ラックのローカルゲートウェイの仕組み) を参照してください。ダイレクト VPC ルーティングを使用しており、Outpost サブネットにローカルゲートウェイへのルートがある場合、Kubernetes コントロールプレーンインスタンスのプライベート IP アドレスがローカルネットワークを介して自動的にブロードキャストされます。ローカルクラスターの Kubernetes API サーバーエンドポイントは Amazon Route 53 (Route 53) でホストされます。API サービスエンドポイントは、パブリック DNS サーバーによって Kubernetes API サーバーのプライベート IP アドレスに解決されます。

   ローカルクラスターの Kubernetes コントロールプレーンインスタンスは、クラスターのライフサイクルを通じて変更されない固定プライベート IP アドレスを持つ静的なエラスティックネットワークインターフェイスで構成されます。Kubernetes API サーバーとやり取りするマシンは、ネットワーク接続が切断されている間は Route 53 に接続できない場合があります。このような場合は、操作の継続性を確保するために、静的プライベート IP アドレスを使用して /etc/hosts を構成することをお勧めします。また、ローカル DNS サーバーを設定して Outpost に接続することもお勧めします。詳細については、「AWS Outposts ドキュメント」を参照してください。次のコマンドを実行して、クラスターとの通信が確立されていることを確認します。

```
kubectl get svc
```

出力例は次のとおりです。

```
NAME         TYPE        CLUSTER-IP   EXTERNAL-IP   PORT(S)   AGE
kubernetes   ClusterIP   10.100.0.1   <none>        443/TCP   28h
```

4. (オプション) AWS Cloud から切断された状態のときにローカルクラスターへの認証をテストします。手順については、the section called “切断の準備をする” を参照してください。

内部リソース

Amazon EKS はクラスターで次のリソースを作成します。リソースは、Amazon EKS 内部で使用するためのものです。クラスターが適切に機能しなくなるため、これらのリソースは編集または変更しないでください。

- 次のミラーポッド:
  - `aws-iam-authenticator-`*`node-hostname`*
  - `eks-certificates-controller-`*`node-hostname`*
  - `etcd-`*`node-hostname`*
  - `kube-apiserver-`*`node-hostname`*
  - `kube-controller-manager-`*`node-hostname`*
  - `kube-scheduler-`*`node-hostname`*
- 次のセルフマネージド型アドオン:
  - `kube-system/coredns`
  - `kube-system/ kube-proxy` (最初のノードを追加するまで作成されません)
  - `kube-system/aws-node` (最初のノードを追加するまで作成されません)。ローカルクラスターは、クラスターネットワークに Amazon VPC CNI plugin for Kubernetes プラグインを使用します。コントロールプレーンインスタンス (名前が `aws-node-controlplane-*` のポッド) の設定を変更しないでください。プラグインが新しいネットワークインターフェイスを作成したときに、デフォルト値を変更できる設定変数があります。詳細については、GitHub のドキュメントを参照してください。
- 次のサービス:
  - `default/kubernetes`
  - `kube-system/kube-dns`
- `eks.system` という名前の `PodSecurityPolicy`
- `eks:system:podsecuritypolicy` という名前の `ClusterRole`

- `eks:system` という名前の `ClusterRoleBinding`
- クラスターセキュリティグループに加えて、Amazon EKS は `eks-local-internal-do-not-use-or-edit-cluster-name-uniqueid` という名前の AWS アカウントにセキュリティグループを作成します。このセキュリティグループにより、コントロールプレーンインスタンスで実行されている Kubernetes コンポーネント間のトラフィックが自由に流れるようになります。

推奨される次のステップは、以下の通りです。

- クラスターを作成した IAM プリンシパルに、AWS マネジメントコンソール 内の Kubernetes リソースを表示するために必要な許可を付与する
- IAM エンティティにクラスターへのアクセスを許可する。エンティティに Amazon EKS コンソールで Kubernetes リソースを表示させる場合は、必要なアクセス許可をエンティティに付与します。
- クラスターのログ記録を設定する。
- ネットワークの切断中に何が起こるかをよく理解してください。
- クラスターにノードを追加する。
- `etcd` のバックアップ計画を設定することを検討してください。Amazon EKS は、ローカルクラスターの `etcd` の自動バックアップと復元をサポートしていません。詳細については、Kubernetes ドキュメントの「etcd クラスターのバックアップ」を参照してください。2 つの主なオプションは、`etcdctl` を使用してスナップショットの作成を自動化する、または Amazon EBS ストレージボリュームのバックアップを使用することです。

AWS Outposts の Kubernetes および Amazon EKS プラットフォームバージョンについて説明します。

ローカルクラスターのプラットフォームバージョンはAWS Outposts 上の Amazon EKS クラスターの機能を表します。バージョンには Kubernetes コントロールプレーン上で実行されるコンポーネントが含まれており、Kubernetes API サーバーフラグが有効になっています。また、現在の Kubernetes パッチバージョンも含まれています。各 Kubernetes マイナーバージョンには、 プラットフォームバージョンが 1 つ以上関連付けられています。別の Kubernetes マイナーバージョンのプラットフォームバージョンは独立しています。クラウド内のローカルクラスターと Amazon EKS クラスターのプラットフォームバージョンは独立しています。

`1.33` など、新しい Kubernetes マイナーバージョンがローカルクラスターで使用可能になると、その Kubernetes マイナーバージョンの最初のプラットフォームバージョンは `eks-local-`

outposts.1 から始まります。また、Amazon EKS では、新しいプラットフォームバージョンを定期的にリリースすることで、新しい Kubernetes コントロールプレーン設定を有効化し、セキュリティ修正プログラムを提供します。

マイナーバージョンで新しいローカルクラスタープラットフォームのバージョンが使用可能になった場合:

- プラットフォームのバージョン番号がインクリメントされます (eks-local-outposts.n+1)。
- Amazon EKS は既存のすべての既存のローカルクラスターを、対応する Kubernetes マイナーバージョン用の最新のプラットフォームバージョンに自動的に更新します。既存のプラットフォームバージョンの自動更新は段階的にロールアウトされます。ロールアウトプロセスは、3 つのインスタンスすべてが新しいインスタンスに置き換えられるまで、Outpost で実行されているマネージド型 Kubernetes コントロールプレーンインスタンスを一度に 1 つずつ置き換えます。
- Kubernetes コントロールプレーンインスタンスの置換プロセスは、サービスの中断のリスクがある場合、進行を停止します。Amazon EKS は、他の 2 つの Kubernetes コントロールプレーンインスタンスが正常で、クラスターノードとしての準備状況条件にすべて合格した場合にのみ、インスタンスの置き換えを試みます。
- プラットフォームバージョンのロールアウトが完了するまでに通常 30 分もかかりません。クラスターが長時間 UPDATING 状態のままである場合は、「the section called “クラスターのトラブルシューティングを行う”」を参照し、AWS サポートに問い合わせてください。AWS サポートからの指示がない限り、Kubernetes コントロールプレーンインスタンスを手動で終了しないでください。
- Amazon EKS は対応するパッチバージョンを適用して新しいノード AMI を発行する可能性があります。すべてのパッチバージョンは単一の Kubernetes マイナーバージョンに対して、Kubernetes コントロールプレーンとノード AMI との間で互換性があります。

新しいプラットフォームバージョンでは重大な変更が発生したり、サービスが中断されたりすることはありません。

ローカルクラスターは常に、指定された Kubernetes バージョン用に入手可能な最新のプラットフォームバージョン (eks-local-outposts.n) で作成されます。

現在および最新の プラットフォームバージョンを以下の表に示します。

この特定のドキュメントページへのすべてのソースファイルの変更通知を受け取るには、RSS リーダーを使用して次の URL をサブスクライブできます。

```
https://github.com/awsdocs/amazon-eks-user-guide/commits/mainline/latest/ug/outposts/
eks-outposts-platform-versions.adoc.atom
```

Kubernetes バージョン **1.33**

以下のアドミッションコントローラーは、すべての 1.33 プラットフォームバージョンで有効です: CertificateApproval、CertificateSigning、CertificateSubjectRestriction、ClusterTr

Kubernetes バージョン	Amazon EKS プラットフォームバージョン	リリースノート	リリース日
1.33.13	eks-local-outposts.4	セキュリティの修正および機能強化が行われた新しいプラットフォームバージョン。Kubernetes 向け Amazon VPC CNI プラグインは v1.23.1 に更新されました。	2026 年 9 月 29 日
1.33.13	eks-local-outposts.3	セキュリティの修正および機能強化が行われた新しいプラットフォームバージョン。Bottlerocket のバージョンを v1.65.0 に更新しました。	2026 年 9 月 22 日
1.33.13	eks-local-outposts.2	セキュリティの修正および機能強化が行われた新しいプラットフォームバージョン。kube-proxy は v1.33.13 に更新さ	2026 年 7 月 29 日

Kubernetes バージョン	Amazon EKS プラットフォームバージョン	リリースノート	リリース日
		れました。AWSIAM オーセンティケーター が v0.7.18 に更新されました。Kubernetes 向け Amazon VPC CNI プラグインが v1.22.4 に更新されました。AWSクラウドコントローラーマネージャーが v1.33.4 に更新されました。Bottlerocket のバージョンを v1.63.0 に更新しました。	
1.33.11	eks-local-outposts.1	Outposts のローカル Amazon EKS クラスター用 Kubernetes バージョン v1.33 の初回リリース。kube-proxy v1.33.11。AWSIAM Authenticator v0.7.13。Amazon VPC CNI plugin for Kubernetes v1.22.1。CoreDNS v1.11.4。AWSCloud Controller Manager v1.33.3。Bottlerocket バージョン v1.62.0。	2026 年 6 月 15 日

Kubernetes バージョン 1.32

以下のアドミッションコントローラーは、すべての 1.32 プラットフォームバージョンで有効です: CertificateApproval、CertificateSigning、CertificateSubjectRestriction、ClusterTr

Kubernetes バージョン	Amazon EKS プラットフォームバージョン	リリースノート	リリース日
1.32.13	eks-local-outposts.7	セキュリティの修正および機能強化が行われた新しいプラットフォームバージョン。Kubernetes 向け Amazon VPC CNI プラグインは v1.23.1 に更新されました。	2026 年 9 月 29 日
1.32.13	eks-local-outposts.6	セキュリティの修正および機能強化が行われた新しいプラットフォームバージョン。Bottlerocket のバージョンを v1.65.0 に更新しました。	2026 年 9 月 22 日
1.32.13	eks-local-outposts.5	セキュリティの修正および機能強化が行われた新しいプラットフォームバージョン。AWSIAM オーセンティケーター が v0.7.18 に更新されました。Kubernetes 向け Amazon VPC	2026 年 7 月 29 日

Kubernetes バージョン	Amazon EKS プラットフォームバージョン	リリースノート	リリース日
		CNI プラグインが v1.22.4 に更新されました。AWSクラウドコントローラーマネージャーが v1.32.8 に更新されました。Bottlerocket のバージョンを v1.63.0 に更新しました。	
1.32.13	eks-local-outposts.4	セキュリティの修正および機能強化が行われた新しいプラットフォームバージョン。kube-proxy は v1.32.13 に更新されました。AWSIAM オーセンティケーター が v0.7.13 に更新されました。Kubernetes 向け Amazon VPC CNI プラグインが v1.22.1 に更新されました。AWSクラウドコントローラーマネージャーが v1.32.7 に更新されました。Bottlerocket のバージョンを v1.62.0 に更新しました。	2026 年 6 月 10 日

Kubernetes バージョン	Amazon EKS プラットフォームバージョン	リリースノート	リリース日
1.32.12	eks-local-outposts.3	Outposts のローカル Amazon EKS クラスター用 Kubernetes バージョン v1.32 の初回リリース。kube-proxy v1.32.12。AWSIAM Authenticator v0.7.10。Amazon VPC CNI plugin for Kubernetes v1.20.4。CoreDNS v1.11.4。AWSCloud Controller Manager v1.32.5。Bottlerocket バージョン v1.57.0。	2026 年 4 月 10 日

Kubernetes バージョン 1.31

以下のアドミッションコントローラーは、すべての 1.31 プラットフォームバージョンで有効です: CertificateApproval、CertificateSigning、CertificateSubjectRestriction、ClusterTr

Kubernetes バージョン	Amazon EKS プラットフォームバージョン	リリースノート	リリース日
1.31.14	eks-local-outposts.14	セキュリティの修正および機能強化が行われた新しいプラットフォームバージョン。Kuberne	2026 年 9 月 29 日

Kubernetes バージョン	Amazon EKS プラットフォームバージョン	リリースノート	リリース日
		tes 向け Amazon VPC CNI プラグインは v1.23.1 に更新されました。	
1.31.14	eks-local-outposts.13	セキュリティの修正および機能強化が行われた新しいプラットフォームバージョン。Bottlerocket のバージョンを v1.65.0 に更新しました。	2026 年 9 月 22 日
1.31.14	eks-local-outposts.12	セキュリティの修正および機能強化が行われた新しいプラットフォームバージョン。AWSIAM オーセンティケーター が v0.7.18 に更新されました。Kubernetes 向け Amazon VPC CNI プラグインは v1.22.4 に更新されました。Bottlerocket のバージョンを v1.63.0 に更新しました。	2026 年 7 月 29 日

Kubernetes バージョン	Amazon EKS プラットフォームバージョン	リリースノート	リリース日
1.31.14	eks-local-outposts.11	セキュリティの修正および機能強化が行われた新しいプラットフォームバージョン。AWSIAM オーセンティケーター が v0.7.13 に更新されました。Kubernetes 向け Amazon VPC CNI プラグイン は v1.22.1 に更新されました。Bottlerocket のバージョンを v1.62.0 に更新しました。	2026 年 6 月 10 日
1.31.14	eks-local-outposts.10	セキュリティの修正および機能強化が行われた新しいプラットフォームバージョン。Bottlerocket のバージョンを v1.57.0 に更新しました。	2026 年 4 月 6 日

Kubernetes バージョン	Amazon EKS プラットフォームバージョン	リリースノート	リリース日
1.31.14	eks-local-outposts.9	セキュリティの修正および機能強化が行われた新しいプラットフォームバージョン。AWSIAM オーセンティケーターが v0.7.10 に更新されました。AWS クラウドコントローラーマネージャーが v1.31.9 に更新されました。Bottlerocket のバージョンを v1.56.0 に更新しました。	2026 年 3 月 25 日

Kubernetes バージョン	Amazon EKS プラットフォームバージョン	リリースノート	リリース日
1.31.14	eks-local-outposts.8	セキュリティの修正および機能強化が行われた新しいプラットフォームバージョン。kube-proxy は v1.31.14 に更新されました。AWSIAM オーセンティケーター が v0.7.8 に更新されました。Kubernetes 向け Amazon VPC CNI プラグインは v1.20.4 に更新されました。Bottlerocket のバージョンを v1.52.0 に更新しました。	2025 年 12 月 23 日

Kubernetes バージョン	Amazon EKS プラットフォームバージョン	リリースノート	リリース日
1.31.12	eks-local-outposts.5	セキュリティの修正および機能強化が行われた新しいプラットフォームバージョン。kube-proxy は v1.31.10 に更新されました。AWSIAM オーセンティケーター が v0.7.4 に更新されました。Kubernetes 向け Amazon VPC CNI プラグインは v1.20.2 に更新されました。Bottlerocket のバージョンを v1.47.0 に更新しました。	2025 年 10 月 3 日

Kubernetes バージョン	Amazon EKS プラットフォームバージョン	リリースノート	リリース日
1.31.9	eks-local-outposts.4	セキュリティの修正および機能強化が行われた新しいプラットフォームバージョン。kube-proxy は v1.31.9 に更新されました。AWSIAM オーセンティケーター が v0.7.2 に更新されました。Kubernetes 向け Amazon VPC CNI プラグインは v1.20.0 に更新されました。Bottlerocket のバージョンを v1.43.0 に更新しました。	2025 年 8 月 9 日

Kubernetes バージョン	Amazon EKS プラットフォームバージョン	リリースノート	リリース日
1.31.7	eks-local-outposts.3	セキュリティの修正および機能強化が行われた新しいプラットフォームバージョン。kube-proxy は v1.31.9 に更新されました。AWSIAM オーセンティケーター が v0.7.1 に更新されました。Kubernetes 向け Amazon VPC CNI プラグインは v1.19.5 に更新されました。Bottlerocket のバージョンを v1.40.0 に更新しました。	2025 年 6 月 19 日
1.31.6	eks-local-outposts.2	セキュリティの修正および機能強化が行われた新しいプラットフォームバージョン。Bottlerocket のバージョンを v1.36.0 に更新しました。	2025 年 4 月 24 日
1.31.6	eks-local-outposts.1	アウトポスト のローカル Amazon EKS クラスター用 Kubernetes バージョン v1.31 の初回リリース。	2025 年 4 月 9 日

Kubernetes バージョン **1.30**

以下のアドミッションコントローラーは、すべての 1.30 プラットフォームバージョンで有効です: CertificateApproval、CertificateSigning、CertificateSubjectRestriction、ClusterTr

Kubernetes バージョン	Amazon EKS プラットフォームバージョン	リリースノート	リリース日
1.30.14	eks-local-outposts.15	セキュリティの修正および機能強化が行われた新しいプラットフォームバージョン。	2026 年 9 月 22 日
1.30.14	eks-local-outposts.14	セキュリティの修正および機能強化が行われた新しいプラットフォームバージョン。AWSIAM オーセンティケーター が v0.7.18 に更新されました。Kubernetes 向け Amazon VPC CNI プラグイン は v1.22.4 に更新されました。Bottlerocket のバージョンを v1.63.0 に更新しました。	2026 年 7 月 29 日
1.30.14	eks-local-outposts.13	セキュリティの修正および機能強化が行われた新しいプラットフォームバージョン。AWSIAM オーセンティケーター が	2026 年 6 月 10 日

Kubernetes バージョン	Amazon EKS プラットフォームバージョン	リリースノート	リリース日
		v0.7.13 に更新されました。Kubernetes 向け Amazon VPC CNI プラグインは v1.22.1 に更新されました。Bottlerocket のバージョンを v1.62.0 に更新しました。	
1.30.14	eks-local-outposts.12	セキュリティの修正および機能強化が行われた新しいプラットフォームバージョン。Bottlerocket のバージョンを v1.57.0 に更新しました。	2026 年 4 月 6 日

Kubernetes バージョン	Amazon EKS プラットフォームバージョン	リリースノート	リリース日
1.30.14	eks-local-outposts.11	セキュリティの修正および機能強化が行われた新しいプラットフォームバージョン。AWSIAM オーセンティケーターが v0.7.10 に更新されました。AWS クラウドコントローラーマネージャーが v1.30.10 に更新されました。Bottlerocket のバージョンを v1.56.0 に更新しました。	2026 年 3 月 25 日
1.30.14	eks-local-outposts.10	セキュリティの修正および機能強化が行われた新しいプラットフォームバージョン。AWSIAM オーセンティケーター が v0.7.8 に更新されました。Kubernetes 向け Amazon VPC CNI プラグインは v1.20.4 に更新されました。Bottlerocket のバージョンを v1.52.0 に更新しました。	2025 年 12 月 23 日

Kubernetes バージョン	Amazon EKS プラットフォームバージョン	リリースノート	リリース日
1.30.14	eks-local-outposts.7	セキュリティの修正および機能強化が行われた新しいプラットフォームバージョン。kube-proxy は v1.30.14 に更新されました。AWSIAM オーセンティケーター が v0.7.4 に更新されました。Kubernetes 向け Amazon VPC CNI プラグインは v1.20.2 に更新されました。Bottlerocket のバージョンを v1.47.0 に更新しました。	2025 年 10 月 3 日

Kubernetes バージョン	Amazon EKS プラットフォームバージョン	リリースノート	リリース日
1.30.13	eks-local-outposts.6	セキュリティの修正および機能強化が行われた新しいプラットフォームバージョン。kube-proxy は v1.30.13 に更新されました。AWSIAM オーセンティケーター が v0.7.2 に更新されました。Kubernetes 向け Amazon VPC CNI プラグインは v1.20.0 に更新されました。Bottlerocket のバージョンを v1.43.0 に更新しました。	2025 年 8 月 9 日

Kubernetes バージョン	Amazon EKS プラットフォームバージョン	リリースノート	リリース日
1.30.11	eks-local-outposts.5	セキュリティの修正および機能強化が行われた新しいプラットフォームバージョン。kube-proxy は v1.30.11 に更新されました。AWSIAM オーセンティケーター が v0.7.1 に更新されました。Kubernetes 向け Amazon VPC CNI プラグインは v1.19.5 に更新されました。Bottlerocket のバージョンを v1.40.0 に更新しました。	2025 年 6 月 19 日
1.30.10	eks-local-outposts.4	セキュリティの修正および機能強化が行われた新しいプラットフォームバージョン。Bottlerocket のバージョンを v1.36.0 に更新しました。	2025 年 4 月 24 日

Kubernetes バージョン	Amazon EKS プラットフォームバージョン	リリースノート	リリース日
1.30.10	eks-local-outposts.3	セキュリティの修正および機能強化が行われた新しいプラットフォームバージョン。kube-proxy は v1.30.10 に更新されました。AWSIAM オーセンティケーター が v0.6.29 に更新されました。Kubernetes 向け Amazon VPC CNI プラグインは v1.19.2 に更新されました。CoreDNS が v1.11.4 に更新されました。AWS クラウドコントローラーマネージャーが v1.30.8 に更新されました。Bottlerocket のバージョンを v1.34.0 に更新しました。	2025 年 3 月 27 日

Kubernetes バージョン	Amazon EKS プラットフォームバージョン	リリースノート	リリース日
1.30.7	eks-local-outposts.2	セキュリティの修正および機能強化が行われた新しいプラットフォームバージョン。kube-proxy は v1.30.7 に更新されました。AWSIAM オーセンティケーター が v0.6.28 に更新されました。Kubernetes 向け Amazon VPC CNI プラグインは v1.19.0 に更新されました。Bottlerocket のバージョンを v1.29.0 に更新しました。	2025 年 1 月 10 日
1.30.5	eks-local-outposts.1	アウトポスト のローカル Amazon EKS クラスター用 Kubernetes バージョン v1.30 の初回リリース。	2024 年 11 月 13 日

AWS Outposts で Amazon EKS クラスターの VPC とサブネットを作成する

ローカルクラスターを作成する際には、VPC と、Outposts で実行されるプライベートサブネットを少なくとも 1 つ指定します。このトピックでは、ローカルクラスターの VPC およびサブネットの要件と考慮事項について概要を説明します。

VPC の要件と考慮事項

ローカルクラスターを作成する際には、指定する VPC が次の要件と考慮事項を満たす必要があります。

- VPC には、作成するローカルクラスター、任意のノード、およびその他の Kubernetes リソースで利用できるだけの十分な数の IP アドレスが必要です。使用する VPC に十分な IP アドレスがない場合は、使用可能な IP アドレスの数を増やしてください。この操作を行うには、VPC への追加の Classless Inter-Domain Routing (CIDR) ブロックの関連付けが必要になります。クラスターの作成前または作成後に、プライベート (RFC 1918) とパブリック (非 RFC 1918) の CIDR ブロックを VPC に関連付けることができます。クラスターで VPC に関連付けた CIDR ブロックが認識されるまでに、最大 5 時間かかることがあります。
- VPC に IP プレフィックスを割り当てたり、IPv6 CIDR ブロックを持たせることはできません。これらの制約があるため、 プレフィックスを使用して Amazon EKS ノードに割り当てる IP アドレスを増やす および the section called “IPv6” でカバーされた情報は VPC には適用されません。
- VPC では DNS ホスト名と DNS 解決が有効になっています。これらの機能がないと、ローカルクラスターの作成は失敗し、機能を有効にしてクラスターを再作成する必要があります。詳細については、「Amazon VPC ユーザーガイド」の「DNS attributes for your VPC」(VPC の DNS 属性) を参照してください。
- ローカルネットワーク経由でローカルクラスターにアクセスするには、VPC が Outpost のローカルゲートウェイルートテーブルに関連付けられている必要があります。詳細については、「AWS Outposts ユーザーガイド」の「VPC アソシエーション」を参照してください。

サブネットの要件と考慮事項

クラスターを作成する際には、少なくとも 1 つのプライベートサブネットを指定します。複数のサブネットを指定すると、Kubernetes コントロールプレーンインスタンスはサブネット全体に均等に分散されます。1 つ以上のサブネットを指定する場合、サブネットは同じ Outpost 上に存在する必要があります。また、相互に通信するために、サブネットには適切なルートとセキュリティグループ権限も必要です。ローカルクラスターの作成時に、指定するサブネットは次の要件を満たす必要があります。

- サブネットはすべて、同じロジカル Outpost 上に存在すること。
- サブネットには、Kubernetes コントロールプレーンインスタンスに対して、合計で 3 つ以上の使用可能な IP アドレスを用意してください。3 つのサブネットを指定する場合、各サブネットには少なくても 1 つ以上の使用可能な IP アドレスが必要です。2 つのサブネットを指定する場合、各

サブネットには少なくても 2 つ以上の利用可能な IP アドレスが必要です。1 つのサブネットを指定する場合、サブネットには少なくても 3 つ以上の利用可能な IP アドレスが必要です。

- ローカルネットワーク経由で Kubernetes API サーバーにアクセスするため、サブネットには Outpost ラックのローカルゲートウェイへのルートがあります。サブネットに Outpost ラックのローカルゲートウェイへのルートがない場合は、VPC 内から Kubernetes API サーバーと通信する必要があります。
- サブネットでは IP アドレスベースの命名を使用する必要があります。Amazon EC2 のリソースベースの命名は、Amazon EKS でサポートされていません。

AWS サービスへのサブネットアクセス

Outposts 上のローカルクラスターのプライベートサブネットは、リージョンレベルの AWS サービスと通信できる必要があります。アウトバウンドインターネットアクセスに NAT ゲートウェイを使用するか、VPC 内ですべてのトラフィックをプライベートに保ちたい場合は、インターフェイス VPC エンドポイントを使用することによって、これを実現できます。

NAT ゲートウェイの使用

Outposts 上のローカルクラスターのプライベートサブネットには、Outposts の親アベイラビリティーゾーンにあるパブリックサブネット内の NAT ゲートウェイへのルートを持つ、関連付けられたルートテーブルが必要です。パブリックサブネットには、インターネットゲートウェイへのルートが必要です。NAT ゲートウェイはアウトバウンドのインターネットアクセスを可能にし、インターネットから Outpost のインスタンスへの未承諾なインバウンド接続を防ぎます。

インターフェイス VPC エンドポイントの使用

Outposts のローカルクラスターのプライベートサブネットにアウトバウンドインターネット接続がない場合、または VPC 内ですべてのトラフィックをプライベートに保ちたい場合は、クラスターを作成する前に、リージョンレベルのサブネットで次のインターフェイス VPC エンドポイントとゲートウェイエンドポイントを作成する必要があります。

Endpoint	エンドポイントタイプ
`com.amazonaws.` *`region-code`* `.ssm`	インターフェイス
`com.amazonaws.` *`region-code`* `.ssmmessages`	インターフェイス

Endpoint	エンドポイントタイプ
com.amazonaws. *region-code* .ec2messages	インターフェイス
com.amazonaws. *region-code* .ec2	インターフェイス
com.amazonaws. *region-code* .secretsmanager	インターフェイス
com.amazonaws. *region-code* .logs	インターフェイス
com.amazonaws. *region-code* .sts	インターフェイス
com.amazonaws. *region-code* .ecr.api	インターフェイス
com.amazonaws. *region-code* .ecr.dkr	インターフェイス
com.amazonaws. *region-code* .s3	ゲートウェイ

エンドポイントは次の要件を満たしている必要があります。

- Outpost の親アベイラビリティーゾーンにあるプライベートサブネットに作成されている
- プライベートDNS 名が有効になっている
- プライベート Outpost サブネットの CIDR 範囲からのインバウンド HTTPS トラフィックを許可するセキュリティグループがアタッチされている

エンドポイントを作成すると料金が発生します。詳細については「AWS プライベートリンク の料金」を参照してください。Pod が他の AWS サービスにアクセスする必要がある場合は、追加のエンドポイントを作成する必要があります。エンドポイントの包括的なリストについては、「AWS PrivateLink と統合する AWS のサービス」を参照してください。

「VPC を作成する」

以下の AWS CloudFormation テンプレートのいずれかを使用して、前の要件を満たす VPC を作成できます。

- テンプレート 1 — このテンプレートは、Outpost に 1 つのプライベートサブネット、また AWS リージョンに 1 つのパブリックサブネットを持つ VPC を作成します。プライベートサブネットに

は、AWS リージョンのパブリックサブネットにある NAT Gateway 経由でのインターネットへのルートがあります。このテンプレートを使用して、出力インターネットアクセスを持つサブネットにローカルクラスターを作成できます。

- テンプレート 2 — このテンプレートは、Outpost に 1 つのプライベートサブネットと、入力または出力のインターネットアクセスがないサブネット (プライベートサブネットとも呼ばれる) にローカルクラスターを作成するために必要な最小限の VPC エンドポイントセットを含む VPC を作成します。

AWS Outposts でローカル Amazon EKS クラスターを準備して、ネットワークの切断に備える

ローカルネットワークが AWS Cloud との接続を失った場合でも、Outpost にあるローカル Amazon EKS クラスターは引き続き使用できます。このトピックでは、ネットワークの切断とそれに関連する考慮事項に備えて、ローカルクラスターを準備する方法について説明します。

- ローカルクラスターにより、計画外の一時的なネットワーク切断中も安定してオペレーションを継続できます。AWSOutposts は、データセンター内の AWS Cloud の拡張機能として機能する完全に接続された製品であることに変わりはありません。Outpost と AWS Cloud 間のネットワークが切断された場合には、接続の復元を試みることをお勧めします。手順については、「AWS Outposts ユーザーガイド」の「AWS Outposts ラックネットワークのトラブルシューティングチェックリスト」を参照してください。ローカルクラスター上の問題をトラブルシューティングする方法については、「the section called “クラスターのトラブルシューティングを行う”」を参照してください。
- Outposts は、Outpost の接続状態を監視するために使用できる `ConnectedStatus` 指標を出力します。詳細については、「AWS Outposts ユーザーガイド」の「Outposts メトリクス」を参照してください。
- ローカルクラスターでは、AWS Identity and Access Management authenticator for Kubernetes を使用する、デフォルトの認証メカニズムとして IAM を使用します。ネットワークの切断中には、IAM は使用できません。ですから、ローカルクラスターが、ネットワークの切断中にクラスターの接続に使用できる `x.509` 証明書を使用する代替認証メカニズムをサポートします。クラスターの `x.509` 証明書を取得して使用する方法については、「the section called “ネットワーク切断中のローカルクラスターへの認証”」を参照してください。
- ネットワークの切断中に Route 53 にアクセスできない場合は、オンプレミス環境上にあるローカル DNS サーバーを使用することを検討してください。Kubernetes コントロールプレーンインスタンスは静的 IP アドレスを使用します。ローカル DNS サーバーを使用する代わりに、エンドポ

イントのホスト名と IP アドレスを使用してクラスターに接続するホストを設定できます。詳細については、AWS Outposts ユーザーガイドの DNS を参照してください。

- ネットワークの切断中にアプリケーショントラフィックの増加が予想する場合は、クラウドに接続したときにクラスターに予備のコンピューティング容量をプロビジョニングできます。Amazon EC2 インスタンスは AWS Outposts の料金に含まれています。そのため、スペアインスタンスを実行しても AWS の使用コストには影響しません。
- ネットワークが切断されている間、ワークロードの作成、更新、スケーリングオペレーションを有効にするには、アプリケーションのコンテナイメージにローカルネットワークを介してアクセスでき、クラスターに十分な容量がある必要があります。ローカルクラスターはコンテナレジストリをホストしません。Pod がそれらのノードで以前に実行されていた場合、コンテナイメージはノードにキャッシュされます。アプリケーションのコンテナイメージをクラウドの Amazon ECR から通常にプルする場合は、ローカルキャッシュまたはレジストリの実行を検討してください。ローカルキャッシュまたはレジストリは、ネットワーク切断中にワークロードリソースの作成、更新、スケーリングオペレーションが必要になった場合に便利です。
- ローカルクラスターは、Amazon EBS を永続ボリュームのデフォルトストレージクラスとして使用し、Amazon EBS 永続ボリュームのライフサイクルを管理するために Amazon EBS CSI ドライバーを使用しています。ネットワークの切断中は、Amazon EBS によってバックアップされる Pod を作成、更新、またはスケーリングすることはできません。これらのオペレーションにはクラウド内の Amazon EBS API への呼び出しが必要だからです。ステートフルワークロードをローカルクラスターにデプロイしていて、ネットワークの切断中に作成、更新、またはスケーリングオペレーションが必要な場合は、別のストレージメカニズムの使用を検討してください。
- AWS Outposts が、関連する AWS リージョン内 API (Amazon EBS または Amazon S3 の API など) にアクセスできない場合、Amazon EBS スナップショットを作成または削除することはできません。
- ALB (Ingress) と AWS Certificate Manager (ACM) を統合すると、証明書がプッシュされ、AWS Outposts ALB コンピューティングインスタンスのメモリに保存されます。現在の TLS ターミネーションは、AWS リージョンとの接続が切断された場合でも引き続き機能します。このコンテキストでの変更操作は失敗します (新しい入力定義、新しい ACM ベースの証明書 API 操作、ALB コンピューティングスケール、証明書のローテーションなど)。詳細については、「AWS Certificate Manager ユーザーガイド」の「マネージド型の証明書の更新のトラブルシューティング」を参照してください。
- Amazon EKS コントロールプレーンのログは、ネットワークの切断中に Kubernetes コントロールプレーンインスタンスでローカルにキャッシュされます。再接続すると、ログは親 AWS リージョンの CloudWatch Logs に送信されます。Prometheus、Grafana、または Amazon EKS パートナー

ソリューションを使用して、Kubernetes API サーバーのメトリクスエンドポイントを使用して、またはログに Fluent Bit を使用してクラスターをローカルで監視できます。

- アプリケーショントラフィックに Outposts で AWS Load Balancer Controller を使用している場合、AWS Load Balancer Controller が前面にある既存の Pod は、ネットワークの切断中もトラフィックを受信し続けます。ネットワーク切断中に作成された新しい Pod は、Outpost が AWS クラウドに再接続されるまでトラフィックを受信しません。ネットワーク切断中のスケーリングのニーズに対応するために、AWS Cloud に接続している間はアプリケーションのレプリカ数を設定することを検討してください。
- Amazon VPC CNI plugin for Kubernetes は、デフォルトでセカンダリ IP モードになります。`WARM_ENI_TARGET=1` で構成されているため、プラグインは使用可能な IP アドレスの「完全な伸縮性ネットワークインターフェイス」を維持できます。接続されていない状態でのスケーリングのニーズに応じて、`WARM_ENI_TARGET`、`WARM_IP_TARGET`、`MINIMUM_IP_TARGET` の値を変更することを検討してください。詳細については、「GitHub」の「プラグインの readme ファイル」を参照してください。各インスタンスタイプによりサポートされる Pod の最大数のリストについては、GitHub の eni-max-pods.txt ファイルを参照してください。

Kubernetes ポッドのフェイルオーバー動作を調整する

ネットワークの切断中、Kubernetes ノードライフサイクルコントローラーでは、到達できないノードを効果が `NoExecute` の `node.kubernetes.io/unreachable` テイントでマークします。デフォルトでは、許容範囲が一致しないポッドは 300 秒 (5 分) 後に削除されます。この動作は、DaemonSets を使用するか、アプリケーションポッドで `tolerationSeconds` を設定するか、カスタムコントローラーを実装することで調整できます。これにより、一時的な切断時に不要なエビクションなしでポッドをノードに留めることができます。詳細なガイダンスと例については、「Amazon EKS ベストプラクティスガイド」の「Kubernetes ポッドのフェイルオーバー動作を調整する」(https://docs.aws.amazon.com/eks/latest/best-practices/hybrid-nodes-network-disconnection-best-practices.html#tune_kubernetes_pod_failover_behavior) を参照してください。

ネットワーク切断中のローカルクラスターへの認証

AWS Identity and Access Management (IAM) は、ネットワークの切断中は使用できません。切断されている間は、IAM 認証情報を使用してローカルクラスターを認証することはできません。しかしながら、切断時に `x509` 証明書を使用して、ローカルネットワークを介してクラスターに接続できます。切断時に使用するクライアント `X509` 証明書をダウンロードして保存する必要があります。このトピックでは、証明書を作成して使用し、クラスターが接続されていない状態のときにクラスターを認証する方法を説明します。

1. 証明書署名リクエストを作成します。

   a. 証明書署名リクエストを生成します。

   ```
   openssl req -new -newkey rsa:4096 -nodes -days 365 \
       -keyout admin.key -out admin.csr -subj "/CN=admin"
   ```

   b. Kubernetes で証明書署名リクエストを作成します。

   ```
   BASE64_CSR=$(cat admin.csr | base64 -w 0)
   cat << EOF > admin-csr.yaml
   apiVersion: certificates.k8s.io/v1
   kind: CertificateSigningRequest
   metadata:
     name: admin-csr
   spec:
     signerName: kubernetes.io/kube-apiserver-client
     request: ${BASE64_CSR}
     usages:
     - client auth
   EOF
   ```

2. kubectl を使用して証明書署名リクエストを作成します。

   ```
   kubectl create -f admin-csr.yaml
   ```

3. 証明書登録リクエストのステータスを確認します。

   ```
   kubectl get csr admin-csr
   ```

   出力例は次のとおりです。

   ```
   NAME       AGE   REQUESTOR                      CONDITION
   admin-csr  11m   kubernetes-admin               Pending
   ```

   Kubernetes が証明書署名リクエストを作成しました。

4. 証明書署名リクエストを承認します。

   ```
   kubectl certificate approve admin-csr
   ```

5. 証明書署名リクエストの承認のステータスを再確認します。

```
kubectl get csr admin-csr
```

出力例は次のとおりです。

```
NAME       AGE   REQUESTOR                     CONDITION
admin-csr  11m   kubernetes-admin              Approved
```

6. 証明書を取得して検証します。

   a. 証明書を取得する。

   ```
   kubectl get csr admin-csr -o jsonpath='{.status.certificate}' | base64 --decode >
    admin.crt
   ```

   b. 証明書を検証する。

   ```
   cat admin.crt
   ```

7. admin ユーザーのクラスターロールバインディングを作成します。

```
kubectl create clusterrolebinding admin --clusterrole=cluster-admin \
    --user=admin --group=system:masters
```

8. 接続されていない状態のユーザースコープの kubeconfig を生成します。

ダウンロードした admin 証明書を使用して kubeconfig ファイルを生成できます。次のコマンドの *my-cluster* と *apiserver-endpoint* を置き換えます。

```
aws eks describe-cluster --name my-cluster \
    --query "cluster.certificateAuthority" \
    --output text | base64 --decode > ca.crt
```

```
kubectl config --kubeconfig admin.kubeconfig set-cluster my-cluster \
    --certificate-authority=ca.crt --server apiserver-endpoint --embed-certs
```

```
kubectl config --kubeconfig admin.kubeconfig set-credentials admin \
    --client-certificate=admin.crt --client-key=admin.key --embed-certs
```

```
kubectl config --kubeconfig admin.kubeconfig set-context admin@my-cluster \
```

```
    --cluster my-cluster --user admin
```

```
kubectl config --kubeconfig admin.kubeconfig use-context admin@my-cluster
```

9. kubeconfig ファイルを表示する。

```
kubectl get nodes --kubeconfig admin.kubeconfig
```

10. Outpost で既にサービスが稼働している場合は、この手順をスキップしてください。Outpost で実行されているサービスが Amazon EKS のみで、その Outpost が現在本番環境ではない場合は、ネットワークの切断をシミュレートできます。ローカルクラスターで本番環境に入る前に、切断をシミュレートして、接続されていない状態でもクラスターにアクセスできることを確認できます。

    a. Outpostを AWS リージョンに接続するネットワークデバイスにファイアウォールルールを適用します。これにより、Outpost のサービスリンクが切断されます。新しいインスタンスを作成することはできません。現在実行中のインスタンスは、AWS リージョンおよびインターネットへの接続を失います。

    b. x509 証明書を使用して、切断時にローカルクラスターへの接続をテストできます。kubeconfig を前の手順で作成した admin.kubeconfig に必ず変更してください。*my-cluster* の部分は、お客様のローカルクラスター名に置き換えます。

    ```
    kubectl config use-context admin@my-cluster --kubeconfig admin.kubeconfig
    ```

    ローカルクラスターが接続されていない状態で問題が発生した場合は、サポートチケットを開くことをお勧めします。

キャパシティに関する考慮事項に基づいて、AWS Outposts で Amazon EKS クラスターのインスタンスタイプとプレイスメントグループを選択する

このトピックでは、Kubernetes コントロールプレーンインスタンスタイプを選択し、(オプションで) プレイスメントグループを使用して Outpost のローカル Amazon EKS クラスターの高可用性要件を満たすためのガイダンスを提供します。

Outposts でローカルクラスターの Kubernetes コントロールプレーンに使用するインスタンスタイプ (m5、c5、r5 など) を選択する前に、Outpost 設定で使用可能なインスタンスタイプを確認します。

使用可能なインスタンスタイプを特定したら、ワークロードに必要なノード数に基づいてインスタンスサイズ (`large`、`xlarge`、`2xlarge` など) を選択します。次のテーブルは、インスタンスサイズを選択する際のレコメンデーションを示しています。

> **Note**
>
> インスタンスサイズは Outposts にスロットされている必要があります。ローカルクラスターの存続期間中、Outposts で使用可能なサイズの 3 つのインスタンス用に十分な容量を確保してください。使用可能な Amazon EC2 インスタンスタイプのリストについては、「AWS Outposts ラックの特徴」の「コンピューティングとストレージ」セクションを参照してください。

ノードの数。	Kubernetes コントロールプレーンのインスタンスサイズ
1 ~ 20	`large`
21 ~ 100	`xlarge`
101 ~ 250	`2xlarge`
251 ~ 500	`4xlarge`

Kubernetes コントロールプレーンのストレージには、`etcd` に要求される IOPS を満たすために、ローカルクラスターごとに 246 GB の Amazon EBS ストレージが必要です。ローカルクラスターの作成時に、Amazon EBS ボリュームは自動的にプロビジョニングされます。

コントロールプレーンの配置

`OutpostConfig.ControlPlanePlacement.GroupName` プロパティでプレイスメントグループを指定しない場合、Kubernetes コントロールプレーン用にプロビジョニングされた Amazon EC2 インスタンスは、Outpost で利用可能な基盤となる容量全体にわたって、特定のハードウェア配置の強制を受けません。

プレイスメントグループを使用すると、Outpost のローカル Amazon EKS クラスターの高可用性要件を満たすことができます。クラスターの作成中にプレイスメントグループを指定することで、Kubernetes コントロールプレーンインスタンスの配置に影響を与えます。インスタンスは基盤

となる独立したハードウェア (ラックまたはホスト) に分散されるため、ハードウェア障害発生時のインスタンス間の影響が最小限に抑えられます。

設定できるスプレッドのタイプは、デプロイの Outpost ラックの数によって異なります。

- 単一の論理的な Outpost に 1 つまたは 2 つの物理ラックを使用するデプロイ - Kubernetes コントロールプレーンインスタンス用に選択したインスタンスタイプで構成されたホストが少なくとも 3 つ必要です。ホストレベルのスプレッドを使用するスプレッドプレイスメントグループにより、すべての Kubernetes コントロールプレーンインスタンスが、Outpost デプロイで使用可能な基盤となるラック内の個別のホストで実行されるようになります。
- 単一の論理的な Outpost に 3 つ以上の物理ラックを使用するデプロイ - Kubernetes コントロールプレーンインスタンス用に選択したインスタンスタイプで構成されたホストが少なくとも 3 つ必要です。ラックレベルスプレッドを使用するスプレッドプレイスメントグループにより、すべての Kubernetes コントロールプレーンインスタンスが、Outpost デプロイ内の個別のラックで実行されるようになります。または、前のオプションで説明したように、ホストレベルのスプレッドプレイスメントグループを使用することもできます。

希望するプレイスメントグループの作成はお客様の責任となります。`CreateCluster` API を呼び出すときにプレイスメントグループを指定します。プレイスメントグループとその作成方法の詳細については、「Amazon EC2 ユーザーガイド」の「プレイスメントグループ」を参照してください。

- プレイスメントグループを指定する場合、ローカルの Amazon EKS クラスターを正常に作成するには、Outpost に使用可能なスロット容量が必要です。容量は、ホストタイプとラックスプレッドタイプのどちらを使用するかによって異なります。十分な容量がない場合、クラスターは `Creating` 状態のままになります。DescribeCluster API レスポンスのヘルスフィールドで `Insufficient Capacity Error` を確認できます。作成プロセスを進めるには、容量を解放する必要があります。
- Amazon EKS ローカルクラスタープラットフォームとバージョンの更新中に、クラスターの Kubernetes コントロールプレーンインスタンスはローリング更新戦略を使用して新しいインスタンスに置き換えられます。この交換プロセス中に、各コントロールプレーンインスタンスは終了し、それぞれのスロットが解放されます。代わりに新しい更新済みインスタンスがプロビジョニングされます。更新されたインスタンスは、リリースされたスロットに配置される可能性があります。スロットが別の無関係なインスタンスによって消費され、必要なスプレッドトポロジー要件を満たす容量が残っていない場合、クラスターは `Updating` 状態のままになります。DescribeCluster API レスポンスのヘルスフィールドでそれぞれの `Insufficient Capacity Error` を確認できます。更新プロセスを進めて以前の高可用性レベルを再設定できるように、容量を解放する必要があります。

- AWS リージョンごとにアカウントあたり、最大 500 個のプレイスメントグループを作成できます。詳細については、「Amazon EC2 ユーザーガイド」の「一般的なルールと制限」を参照してください。

AWS Outposts でローカル Amazon EKS クラスターをトラブルシューティングする

このトピックでは、ローカルクラスターの使用中に表示される可能性がある一般的なエラーとそのトラブルシューティング方法について説明します。ローカルクラスターはクラウド内の Amazon EKS クラスターと似ていますが、Amazon EKS による管理方法にはいくつかの違いがあります。

> **Important**
>
> AWS サポートから明示的に指示されていない限り、Outpost で実行されているマネージド型 EKS ローカルクラスターの `Kubernetes` コントロールプレーンインスタンスを終了しないでください。これらのインスタンスを終了すると、複数のインスタンスが同時に終了した場合にローカルクラスターが失われるなど、ローカルクラスターサービスの可用性にリスクが伴います。EKS ローカルクラスターの `Kubernetes` コントロールプレーンインスタンスは、EC2 インスタンスコンソールでは `eks-local:controlplane-name` タグで識別します。

API の動作

ローカルクラスターは Amazon EKS API を通して作成されますが、非同期で実行されます。つまり、Amazon EKS API へのリクエストはローカルクラスターに対してすぐに返されます。ただし、これらのリクエストは成功し、入力検証エラーにより迅速に失敗するか、あるいは失敗して説明入りの検証エラーが発生する可能性があります。この動作は、Kubernetes API と同様です。

ローカルクラスターは `FAILED` ステータスに移行しません。Amazon EKS は、クラスターの状態をユーザーが要求した望ましい状態と一致させようと継続的に試みます。その結果、根本的な問題が解決されるまで、長期間にわたってローカルクラスターが `CREATING` 状態のままになる可能性があります。

クラスターヘルスフィールドについて説明します

describe-cluster Amazon EKS AWS CLI コマンドを使用して、ローカルクラスターの問題を検出できます。ローカルクラスターの問題は、`describe-cluster` コマンドの応答の `cluster.health`

フィールドによって明らかになります。このフィールドに含まれるメッセージには、エラーコード、説明メッセージ、および関連するリソース ID が含まれます。この情報は、Amazon EKS API および AWS CLI のみから利用可能です。次の例では、*my-cluster* をローカルクラスターの名前に置き換えます。

```
aws eks describe-cluster --name my-cluster --query 'cluster.health'
```

出力例は次のとおりです。

```
{
    "issues": [
        {
            "code": "ConfigurationConflict",
            "message": "The instance type 'm5.large' is not supported in Outpost 'my-
outpost-arn'.",
            "resourceIds": [
                "my-cluster-arn"
            ]
        }
    ]
}
```

問題が修復できない場合は、ローカルクラスターを削除して新しいクラスターを作成する必要がある場合があります。たとえば、Outpost で利用できないインスタンスタイプでクラスターをプロビジョニングしようとしている場合などです。下表は、一般的なヘルス関連のエラーを示しています。

エラーシナリオ	コード	メッセージ	ResourceIds
指定されたサブネットが見つかりませんでした。	ResourceNotFound	The subnet ID *subnet-id* does not exist	指定されたすべてのサブネット ID
指定されたサブネットが同じ VPC に属していません。	ConfigurationConflict	Subnets specified must belong to the same VPC	指定されたすべてのサブネット ID
指定されたサブネットの一部が、指定さ	ConfigurationConflict	Subnet *subnet-id* expected to be	問題のあるサブネット ID

エラーシナリオ	コード	メッセージ	ResourceIds
れた Outpost に属していません。		in *outpost-arn* , but is in *other-outpost-arn*	
指定されたサブネットの中には、どの Outpost にも属していないものがあります。	ConfigurationConflict	Subnet *subnet-id* is not part of any Outpost	問題のあるサブネット ID
指定されたサブネットの中には、コントロールプレーンインスタンス用のエラスティックネットワークインターフェイスを作成するだけの十分な空きアドレスがないものがあります。	ResourceLimitExceeded	The specified subnet does not have enough free addresses to satisfy the request.	問題のあるサブネット ID
指定されたコントロールプレーンインスタンスタイプは、お使いの Outpost でサポートされていません。	ConfigurationConflict	The instance type *type* is not supported in Outpost *outpost-arn*	クラスター ARN

エラーシナリオ	コード	メッセージ	ResourceIds
コントロールプレーン Amazon EC2 インスタンスを終了したか、run-instance が成功し、観察された状態が Terminated に変わりました。これは、Outpost が再接続され、Amazon EBS の内部エラーが原因で Amazon EC2 内部ワークフローが失敗した後、一定期間発生する可能性があります。	InternalFailure	EC2 instance state "Terminated" is unexpected	クラスター ARN
Outpost の容量が不足しています。これは、クラスターの作成中に Outpost が AWS リージョンから切断された場合にも発生する可能性があります。	ResourceLimitExceeded	There is not enough capacity on the Outpost to launch or start the instance.	クラスター ARN
アカウントがセキュリティグループの制限を超えています。	ResourceLimitExceeded	Amazon EC2 API から返されたエラーメッセージ	ターゲット VPC ID
アカウントがエラスティックネットワークインターフェイスの制限を超えています。	ResourceLimitExceeded	Amazon EC2 API から返されたエラーメッセージ	ターゲットサブネット ID

エラーシナリオ	コード	メッセージ	ResourceIds
AWS Systems Manager を介してコントロールプレーンインスタンスにアクセスできません。解決策については、the section called “AWS Systems Manager を介してコントロールプレーンインスタンスにアクセスできません。” を参照してください。	`ClusterUnreachable`	Amazon EKS コントロールプレーンインスタンスには SSM 経由ではアクセスできません。SSM とネットワーク設定を確認し、EKS on Outposts のトラブルシューティングドキュメントを参照してください。	Amazon EC2 インスタンス ID
マネージドセキュリティグループまたはエラスティックネットワークインターフェイスの詳細を取得中に、エラーが発生しました。	Amazon EC2 クライアントエラーコードに基づきます。	Amazon EC2 API から返されたエラーメッセージ	すべてのマネージドセキュリティグループ ID
セキュリティグループのイングレスルールを承認または取り消す際にエラーが発生しました。これは、クラスターとコントロールプレーンの両方のセキュリティグループに適用されます。	Amazon EC2 クライアントエラーコードに基づきます。	Amazon EC2 API から返されたエラーメッセージ	問題のあるセキュリティグループの ID

エラーシナリオ	コード	メッセージ	ResourceIds
コントロールプレーンインスタンスのエラスティックネットワークインターフェイスの削除中にエラーが発生しました。	Amazon EC2 クライアントエラーコードに基づきます。	Amazon EC2 API から返されたエラーメッセージ	問題のあるエラスティックネットワークインターフェイス ID

次の表は、`describe-cluster` 応答のヘルスフィールドに表示される他の AWS サービスからのエラーを示しています。

Amazon EC2 エラーコード	クラスターヘルス問題コード	説明
`AuthFailure`	`AccessDenied`	このエラーは、さまざまな理由で発生する可能性があります。最も一般的な理由は、サービスにリンクされたロールポリシーの範囲を狭めるためにサービスが使用するタグが、コントロールプレーンインスタンスから誤って削除された場合に発生するのです。この場合、Amazon EKS はこれらの AWS リソースを管理および監視できなくなります。
`UnauthorizedOperation`	`AccessDenied`	このエラーは、さまざまな理由で発生する可能性があります。最も一般的な理由は、サービスにリンクされたロールポリシーの範囲を狭めるためにサービスが使用するタグが、コントロールプレーンインスタンスから誤って削除さ

Amazon EC2 エラーコード	クラスターヘルス問題コード	説明
		れた場合に発生するのです。この場合、Amazon EKS はこれらの AWS リソースを管理および監視できなくなります。
InvalidSubnetID.NotFound	ResourceNotFound	このエラーは、セキュリティグループのイングレスルールのサブネット ID が見つからない場合に発生します。
InvalidPermission.NotFound	ResourceNotFound	このエラーは、セキュリティグループのイングレスルールの権限が正しくない場合に発生します。
InvalidGroup.NotFound	ResourceNotFound	このエラーは、セキュリティグループのイングレスルールのグループが見つからない場合に発生します。
InvalidNetworkInterfaceID.NotFound	ResourceNotFound	このエラーは、セキュリティグループのイングレスルールのネットワークインターフェイス ID が見つからない場合に発生します。
InsufficientFreeAddressesInSubnet	ResourceLimitExceeded	このエラーは、サブネットリソースのクォータを超えたときに発生します。
InsufficientCapacityOnOutpost	ResourceLimitExceeded	このエラーは、outpost のクォータを超えたときに発生します。

Amazon EC2 エラーコード	クラスターヘルス問題コード	説明
NetworkInterfaceLimitExceeded	ResourceLimitExceeded	このエラーは、エラスティックネットワークインターフェイスのクォータを超えた場合に発生します。
SecurityGroupLimitExceeded	ResourceLimitExceeded	このエラーは、セキュリティグループのクォータを超えたときに発生します。
VcpuLimitExceeded	ResourceLimitExceeded	Amazon EC2 インスタンスを新規アカウントで作成するときに発生します。エラーは次のようなものになります:「You have requested more vCPU capacity than your current vCPU limit of 32 allows for the instance bucket that the specified instance type belongs to. Please visit http://aws.amazon.com/contact-us/ec2-request to request an adjustment to this limit."
InvalidParameterValue	ConfigurationConflict	Amazon EC2 は、指定されたインスタンスタイプが Outpost でサポートされていない場合、このエラーコードを返します。
その他のすべての障害	InternalFailure	なし

クラスターを作成または変更できません

ローカルクラスターには、クラウド内にホストされている Amazon EKS クラスターとは異なるアクセス権限とポリシーが必要です。クラスターの作成に失敗し `InvalidPermissions` エラーが表示される場合は、使用しているクラスターロールに AmazonEKSLocalOutpostClusterPolicy マネージドポリシーがアタッチされているかどうかをもう一度確認します。その他すべての API 呼び出しには、クラウド内の Amazon EKS クラスターと同じ権限セットが必要です。

クラスターが **`CREATING`** の状態でスタックしています

ローカルクラスターの作成にかかる時間は、いくつかの要因によって異なります。ネットワーク設定、Outpost の設定、およびクラスターの設定などが要因として考えられます。通常、ローカルクラスターは 15 ~ 20 分以内に作成され、`ACTIVE` ステータスに変わります。ローカルクラスターが `CREATING` 状態を維持する場合は、`describe-cluster` を呼び出すと `cluster.health` 出力フィールドに原因に関する情報が表示されます。

次は、最も一般的な問題を示しています。

- クラスターが Systems Manager のある AWS リージョンからコントロールプレーンインスタンスに接続できません。地域内の踏み台ホストから `aws ssm start-session --target` *`instance-id`* を呼び出すことでこれを検証できます。このコマンドがうまくいかない場合は、Systems Manager がコントロールプレーンインスタンスで実行されているかどうかを確認します。または、別の回避策として、クラスターを削除して再度作成することもできます。
- EBS ボリュームの KMS キーのアクセス許可が原因で、コントロールプレーンインスタンスの作成に失敗します。暗号化された EBS ボリュームのユーザーマネージド KMS キーでは、キーにアクセスできない場合、コントロールプレーンインスタンスは終了します。インスタンスが終了した場合、AWS マネージド KMS キーに切り替えるか、ユーザーマネージドキーポリシーがクラスターロールに必要なアクセス許可を付与していることを確認します。
- Systems Manager のコントロールプレーンインスタンスが、インターネットにアクセスできない可能性があります。クラスターの作成時に指定したサブネットに NAT ゲートウェイとインターネットゲートウェイを備えた VPC があるかどうかを確認します。VPC 到達可能性アナライザーを使用して、コントロールプレーンインスタンスがインターネットゲートウェイに到達できることを確認します。詳細については、「VPC Reachability Analyzer の開始方法」を参照してください。
- 指定したロール ARN にポリシーがありません。AWS マネージドポリシー: AmazonEKSLocalOutpostClusterPolicy がロールから削除されたかどうかを確認します。これは、AWS CloudFormation スタックの設定が間違っている場合にも発生する可能性があります。

- 指定されているすべてのサブネットが同じ Outpost に関連付けられており、相互に到達できる必要があります。クラスターの作成時に複数のサブネットを指定すると、Amazon EKS はコントロールプレーンインスタンスを複数のサブネットに分散させようとします。
- Amazon EKS マネージド セキュリティグループは、エラスティックネットワークインターフェイスに適用されます。しかしながら、NACL ファイアウォールルールなどの他の設定要素が、エラスティックネットワークインターフェイスのルールと競合する可能性があります。
- VPC とサブネット DNS の設定が誤っているか、欠落しています。「AWS Outposts で Amazon EKS クラスターの VPC とサブネットを作成する」を確認します。

クラスターが **UPDATING** の状態でスタックしています

Amazon EKS は既存のすべてのローカルクラスターを、対応する Kubernetes マイナーバージョン用の最新のプラットフォームバージョンに自動的にアップデートします。プラットフォームバージョンの詳細については、「the section called “EKS プラットフォームのバージョン”」を参照してください。

プラットフォームバージョンの自動ロールアウト中に、クラスターのステータスが UPDATING に変わります。更新プロセスでは、すべての Kubernetes コントロールプレーンインスタンスを、それぞれの Kubernetes マイナーバージョン用にリリースされた最新のセキュリティパッチとバグ修正を含む新しいインスタンスに置き換えます。一般的に、ローカルクラスタープラットフォームの更新プロセスは 30 分以内に完了し、クラスターは ACTIVE ステータスに戻ります。ローカルクラスターが長時間 UPDATING 状態になる場合は、`describe-cluster` を呼び出すと `cluster.health` 出力フィールドで原因に関する情報を確認できます。

Amazon EKS は、ローカルクラスターの可用性を維持し、サービスの中断を防ぐために、3 つの Kubernetes コントロールプレーンインスタンスのうち少なくとも 2 つが正常で運用可能なクラスターノードであるようにします。ローカルクラスターが UPDATING 状態で停止している場合、通常は、プロセスが続行された場合に 2 つのインスタンスの最小可用性が保証されないインフラストラクチャまたは設定の問題が原因です。したがって、ローカルクラスターサービスの中断を防ぐために、更新プロセスの進行が停止します。

更新プロセスを完了し、3 つの Kubernetes コントロールプレーンインスタンスの高可用性でローカルクラスターを ACTIVE に戻すことができるように、UPDATING 状態で停止しているローカルクラスターをトラブルシューティングし、根本原因に対処することが重要です。

AWS サポートから明示的に指示されていない限り、Outposts でマネージド型 EKS ローカルクラスターの Kubernetes インスタンスを終了しないでください。これは、別のコントロールプレーンノードが完全には正常ではなく、間違ったインスタンスを終了すると、サービスの中断やローカルク

ラスターのデータ損失のリスクが生じる可能性が高いため、ローカルクラスターが UPDATING 状態で停止した場合に特に重要です。

次は、最も一般的な問題を示しています。

- ローカルクラスターが最初に作成された以降のネットワーク設定の変更により、1 つまたは複数のコントロールプレーンインスタンスが System Manager に接続できません。地域内の踏み台ホストから `aws ssm start-session --target instance-id` を呼び出すことでこれを検証できます。このコマンドがうまくいかない場合は、Systems Manager がコントロールプレーンインスタンスで実行されているかどうかを確認します。
- EBS ボリュームの KMS キーのアクセス許可が原因で、新しいコントロールプレーンインスタンスを作成できません。暗号化された EBS ボリュームのユーザーマネージド KMS キーでは、キーにアクセスできない場合、コントロールプレーンインスタンスは終了します。インスタンスが終了した場合、AWS マネージド KMS キーに切り替えるか、ユーザーマネージドキーポリシーがクラスターロールに必要なアクセス許可を付与していることを確認します。
- Systems Manager のコントロールプレーンインスタンスが、インターネットにアクセスできない可能性があります。クラスターの作成時に指定したサブネットに NAT ゲートウェイとインターネットゲートウェイを備えた VPC があるかどうかを確認します。VPC 到達可能性アナライザーを使用して、コントロールプレーンインスタンスがインターネットゲートウェイに到達できることを確認します。詳細については、「VPC Reachability Analyzer の開始方法」を参照してください。プライベートネットワークにアウトバウンドインターネット接続がない場合は、必要なすべての VPC エンドポイントとゲートウェイエンドポイントがクラスターのリージョンサブネットにまだ存在することを確認してください (「the section called “AWS サービスへのサブネットアクセス”」を参照)。
- 指定したロール ARN にポリシーがありません。AWS マネージドポリシー: AmazonEKSLocalOutpostClusterPolicy がロールから削除されたかどうかを確認します。
- 新しい Kubernetes コントロールプレーンインスタンスのいずれかで、予期しないブートストラップ障害が発生した可能性があります。この例外的なケースのトラブルシューティングと ログ収集に関する詳細なガイダンスについては、AWS サポートセンターにチケットを提出してください。

ノードをクラスターに結合できない

- AMI に関する問題:
  - 互換性のない AMI を使用しています。Amazon EKS 最適化 Amazon Linux 2023 AMI のみがサポートされています。詳細については、「AWS Outposts で Amazon Linux ノードを作成する」を参照してください。

  - AWS CloudFormation テンプレートを使用してノードを作成した場合、サポートされていない AMI を使用していないことを確認します。
- AWS IAM Authenticator `ConfigMap` が見つからない - 見つからない場合は、作成する必要があります。詳細については、the section called “aws-auth ConfigMap をクラスターに適用する” を参照してください。
- 間違ったセキュリティグループが使用されている - ワーカーノードのセキュリティグループには、必ず `eks-cluster-sg-`*`cluster-name`*`-`*`uniqueid`* を使用してください。スタックが使用されるたびに、選択したセキュリティグループは AWS CloudFormation によって変更され、新しいセキュリティグループを使用できるようになります。
- 予期しないプライベートリンクの VPC 手順に従う - CA データが間違っている (`--b64-cluster-ca`) または API エンドポイント (`--apiserver-endpoint`) が渡されました。

ログの収集

Outpost が関連付けられている AWS リージョンから切断された場合でも、Kubernetes クラスターは依然として正常に動作し続けるはずです。ただし、クラスターが正常に機能しない場合は、「AWS Outposts でローカル Amazon EKS クラスターを準備して、ネットワークの切断に備える」にあるトラブルシューティング手順に従ってください。他の問題が発生した場合は、AWS サポートにお問い合わせください。AWSサポートからログ収集ツールをダウンロードして実行する方法を説明します。この方法により、Kubernetes クラスターコントロールプレーンインスタンスからログを収集し、詳細な調査のために AWS サポートに送信できます。

AWS Systems Manager を介してコントロールプレーンインスタンスにアクセスできません。

Amazon EKS コントロールプレーンインスタンスが AWS Systems Manager (Systems Manager) を通してアクセスでない場合、Amazon EKS はクラスターに対して次のエラーを表示します。

```
Amazon EKS control plane instances are not reachable through SSM. Please verify your
 SSM and network configuration, and reference the EKS on Outposts troubleshooting
 documentation.
```

この問題を解決するには、VPC とサブネットが「AWS Outposts で Amazon EKS クラスターの VPC とサブネットを作成する」の要件を満たしており、AWS Systems Manager ユーザーガイドの「Session Manager の設定」にある手順が完了していることを確認してください。

EC2 インスタンスストアを設定した AWS Outposts にあるローカルの Amazon EKS クラスターの概要

ローカルクラスターを使用すると、Amazon EKS クラスターすべてを AWS Outposts でローカルに実行できます。これにより、クラウドへのネットワーク接続が一時的に切断された場合に生じる可能性のあるアプリケーションダウンタイムのリスクを軽減し、データレジデンシー要件に対処できるようになります。Kubernetes クラスター全体が Outposts でローカルに実行されるため、アプリケーションは引き続き利用可能で、クラウドへのネットワーク接続が切断されている間もクラスターオペレーションを実行できます。

ローカルクラスターは一般的に Outposts ラックで使用できます。

EC2 インスタンスストアを設定した Outposts にある Amazon EKS ローカルクラスター

Amazon EKS ローカルクラスターは、EC2 インスタンスストアを設定した AWS Outposts をサポートします。EC2 インスタンスストアを設定した Outposts でローカルクラスターを実行すると、Amazon EKS は最新のローカルクラスターアーキテクチャをデプロイします。これにより、Amazon EKS クラウドクラスターとの同等性が向上します。

EC2 インスタンスストアがサポートする Outposts で使用される最新のアーキテクチャには、以下が追加されています。

- マネージドコントロールプレーン。Kubernetes コントロールプレーンは、Outposts の AWS マネージドサービスアカウントで実行されます。Amazon EKS は、Amazon EKS プラットフォームバージョンの更新や `etcd` バックアップなど、コントロールプレーンのライフサイクルを管理します。
- クラウド内の Amazon EKS との機能同等性の向上。Amazon EKS アドオン、サービスアカウントに対する IAM ロール、EKS Pod Identity、OIDC 認証、および the section called “アクセスエントリ” を使用します。Amazon Linux 2023 ベースのワーカーノードに加えて、Bottlerocket をワーカーノードのオペレーティングシステムとして使用します。
- Amazon EKS Kubernetes バージョンのライフサイクルに合わせた調整。新しい Kubernetes バージョンと Amazon EKS プラットフォームバージョンは、クラウドの Amazon EKS と同じケイデンスでローカルクラスターで使用できるようになります。Kubernetes の延長サポートバージョンは、クラウドクラスターと同じ料金で利用できます。
- AWS Outposts ラックをサポートするすべての商用 AWS リージョンで利用できます。

> **Note**
>
> Outpost が EC2 インスタンスストアではなく Amazon EBS で設定されている場合、このトピックで説明されているアーキテクチャは Outpost に使用できません。EBS を設定した Outposts は、既存のローカルクラスターの実装を引き続き使用します。詳細については、「the section called “ローカルクラスタを実行する”」を参照してください。
> 更新されたローカルクラスターアーキテクチャを使用して EBS-backed Outpost にローカルクラスターを作成する場合は、AWS アカウントチームにお問い合わせください。

ローカルクラスターと拡張クラスターの機能ごとの比較については、「the section called “デプロイオプションの比較”」を参照してください。

はじめに

EC2 インスタンスストアを設定した Outposts にローカルクラスターを作成するには、「the section called “ローカルクラスターをデプロイする (EC2 インスタンスストア)”」を参照してください。

クラスターを作成する前に、次の点を確認します。

- the section called “VPC とサブネットを作成する (EC2 インスタンスストア)” – ローカルクラスターの VPC とサブネットの設定。
- the section called “キャパシティに関する考慮事項 (EC2 インスタンスストア)” – コントロールプレーンと `etcd` インスタンスタイプの選択と配置。
- the section called “切断への備え (EC2 インスタンスストア)” – 切断中の認証、オブザーバビリティ、運用に関する考慮事項。
- the section called “Amazon EKS アドオン (EC2 インスタンスストア)” – ローカルクラスター向けの検証済みの Amazon EKS アドオンセット。

EC2 インスタンスストアを設定した AWS Outposts に Amazon EKS ローカルクラスターをデプロイする

このトピックでは、EC2 インスタンスストアを設定した AWS Outposts にローカルの Amazon EKS クラスターをデプロイする手順について説明します。

> **Note**
>
> Outpost が EC2 インスタンスストアではなく Amazon EBS で設定されている場合、このトピックで説明されているアーキテクチャは Outpost に使用できません。EBS を設定した Outposts は、既存のローカルクラスターの実装を引き続き使用します。詳細については、「the section called “ローカルクラスターをデプロイする”」を参照してください。
> 更新されたローカルクラスターアーキテクチャを使用して EBS-backed Outpost にローカルクラスターを作成する場合は、AWS アカウントチームにお問い合わせください。

> **Important**
>
> - ローカルクラスターは Outpost ラックのみをサポートします。単一のローカルクラスターは、単一の論理 Outpost を構成する複数の物理 Outpost ラックにわたって実行できます。単一のローカルクラスターを複数の論理 Outposts にわたって実行することはできません。
> - Kubernetes コントロールプレーンは、Outposts の AWS マネージドサービスアカウントで実行されます。コントロールプレーンインスタンスにアクセスして、Amazon EC2 コンソールに表示したり、そのインスタンスでワークロードを実行したりすることはできません。
> - コントロールプレーンは、外部の etcd トポロジにある 6 つの EC2 インスタンス (3 つの `etcd` + 3 つの API サーバー) で実行されます。キャパシティ要件については、「the section called “キャパシティに関する考慮事項 (EC2 インスタンスストア)”」を参照してください。
> - Amazon VPC CNI plugin for Kubernetes、kube-proxy、CoreDNS はローカルクラスターに自動的にインストールされます。検証済みアドオンの詳細なリストについては、「the section called “Amazon EKS アドオン (EC2 インスタンスストア)”」を参照してください。
> - Amazon EKS は、定期的に `etcd` をバックアップします。`etcd` がクォーラムを失った場合、Amazon EKS は最新のスナップショットからクラスターの状態を回復します。

前提条件

- EC2 インスタンスストアを設定した Outposts ラックが存在し、コントロールプレーンインスタンスに十分な仮想化容量が確保されていること。「the section called “キャパシティに関する考慮事項 (EC2 インスタンスストア)”」を参照してください。

- VPC とサブネットが「the section called “VPC とサブネットを作成する (EC2 インスタンスストア)”」で説明されている要件を満たしていること。
- コンピュータまたは AWS CloudShell に、kubectl コマンドラインツールがインストールされていること。バージョンはご使用のクラスターの Kubernetes バージョンと同じか、1 つ前のマイナーバージョン以前、あるいはそれより新しいバージョンが使用できます。kubectl をインストールまたはアップグレードする方法については、the section called “kubectl と eksctl のセットアップ” を参照してください。
- ご使用のデバイスまたは AWS CloudShell で、AWS CLI のバージョン 2.35.3 以降がインストールおよび設定されていること。現在のバージョンを確認するには、「aws --version」を参照してください。インストールまたはアップグレードするには、「AWS CLI のインストール」を参照してください。
- Amazon EKS クラスターを作成および記述するアクセス許可を持つ IAM プリンシパル (ユーザーまたはロール)。詳細については、「the section called “Outpost にローカル Kubernetes クラスターを作成する”」および「the section called “すべてのクラスターの一覧表示または説明”」を参照してください。
- クラスターの IAM ロールに AmazonEKSClusterPolicy と AmazonEAmazonEKSServicePolicy という AWS マネージドポリシーがアタッチされていること。ロールの信頼ポリシーでは、eks.amazonaws.com サービスプリンシパルがロールを引き受けることを許可する必要があります。

> **Note**
>
> ローカルクラスターを作成すると、クラスターを作成する IAM プリンシパルが、system:masters アクセス許可を持つ管理者として Kubernetes RBAC 承認テーブルに恒久的に追加されます。このプリンシパルは絶対に削除しないでください。

ローカルクラスターを作成する

ローカルクラスターを作成するには、AWS マネジメントコンソール、AWS CLI、Amazon EKS API、または AWS CloudFormation を使用します。

ローカルクラスターを作成する場合は、以下の点に注意してください。

- Outposts のホームであるアベイラビリティーゾーンにサブネットを渡す必要があります。複数のアベイラビリティーゾーンにサブネットを渡すと、リクエストは失敗します。

- endpointPrivateAccess を true に設定する必要がある。必要に応じて、endpointPublicAccess を有効にすることもできます。
- outpostArns、controlPlaneInstanceType、etcdInstanceType といった outpostConfig の値を渡す必要があります。必要に応じて、配置の設定を指定できます。「the section called “キャパシティに関する考慮事項 (EC2 インスタンスストア)”」を参照してください。

AWS CLI

```
aws eks create-cluster \
  --name my-cluster \
  --role-arn arn:aws:iam::111122223333:role/myEKSClusterRole \
  --kubernetes-version 1.37 \
  --resources-vpc-config subnetIds=subnet-
ExampleID1,endpointPrivateAccess=true,endpointPublicAccess=true \
  --logging '{"clusterLogging":[{"types":
["api","audit","authenticator","controllerManager","scheduler"],"enabled":true}]}' \
  --access-config
 authenticationMode=API_AND_CONFIG_MAP,bootstrapClusterCreatorAdminPermissions=true \
  --outpost-config outpostArns=arn:aws:outposts:region-code:111122223333:outpost/op-
uniqueid,controlPlaneInstanceType=m5.large,controlPlanePlacement={spreadLevel=host},etcdInstanc
```

AWS マネジメントコンソール

1. Amazon EKS コンソールを開きます。
2. [クラスターを作成] を選択します。
3. Kubernetes コントロールプレーンの場所に対して、AWS Outposts を選択します。
4. 次を設定します。
   - [名前] - クラスターの名前。
   - [Kubernetes バージョン] - 使用するバージョン。
   - [クラスターサービスロール] - AmazonEKSClusterPolicy および AmazonEKSServicePolicy を備えた IAM ロール。
   - [Outpost ID] - デプロイ先のロジカル Outposts。
   - [コントロールプレーンインスタンスタイプ] - Kubernetes API サーバー、スケジューラ、コントローラーマネージャーのインスタンスタイプ。
   - [etcd インスタンスタイプ] - etcd のインスタンスタイプ。

- [スプレッドレベル] - コントロールプレーンと etcd の配置に対しては host または rack。

5. ネットワークの設定: VPC、サブネット、セキュリティグループ。プライベートエンドポイントアクセスを有効にします。
6. オブザーバビリティ、つまりコントロールプレーンのログ記録を設定します。
7. 確認して作成します。

CreateCluster API

```
{
    "name": "my-cluster",
    "roleArn": "arn:aws:iam::111122223333:role/myEKSClusterRole",
    "version": "1.37",
    "resourcesVpcConfig": {
        "subnetIds": ["subnet-ExampleID1"],
        "endpointPublicAccess": true,
        "endpointPrivateAccess": true
    },
    "logging": {
        "clusterLogging": [{
            "types": ["api", "audit", "authenticator", "controllerManager",
 "scheduler"],
            "enabled": true
        }]
    },
    "accessConfig": {
        "authenticationMode": "API_AND_CONFIG_MAP",
        "bootstrapClusterCreatorAdminPermissions": true
    },
    "outpostConfig": {
        "outpostArns": ["arn:aws:outposts:region-code:111122223333:outpost/op-
uniqueid"],
        "controlPlaneInstanceType": "m5.large",
        "controlPlanePlacement": {
            "spreadLevel": "host"
        },
        "etcdInstanceType": "m5.large",
        "etcdPlacement": {
            "spreadLevel": "host"
        }
    }
```

```
}
```

EC2 インスタンスストアを設定した Outposts にローカルクラスターを作成するときには、`controlPlaneInstanceType` パラメータと `etcdInstanceType` パラメータが必要です。`controlPlanePlacement` および `etcdPlacement` パラメータはオプションです。指定しないと、配置のスプレッド戦略が適用されません。

> **Note**
>
> `outpostConfig` は、作成専用のプロパティです。コントロールプレーンインスタンスタイプ、`etcd` インスタンスタイプ、配置、または Outposts を変更するには、新しいクラスターを作成する必要があります。

クラスターに接続する

クラスターと通信するように `kubectl` を設定します。

```
aws eks update-kubeconfig --region region-code --name my-cluster
```

> **Note**
>
> `2.35.3` より前の AWS CLI バージョンを使用してクラスターを作成した場合は、`exec` 引数で `--cluster-id` ではなく `--cluster-name` を指定するように `kubeconfig` を更新する必要があります。

クラスターの Kubernetes API サーバーエンドポイントは、Amazon Route 53 でホストされます。エンドポイントは、Amazon EKS がサブネットに作成するクロスアカウントの Elastic Network Interface (ENI) のプライベート IP アドレスに解決されます。こうした ENI には、クラスターのライフサイクルを通じて変更されない固定のプライベート IP アドレスがあります。

クラスターを作成すると、Amazon EKS は指定したサブネットに 3 つの Elastic Network Interface を作成します。これらのネットワークインターフェイスは、クラスターと VPC 間の通信を可能にします。各ネットワークインターフェイスの説明には、テキスト `Amazon EKS cluster-name` が書き込まれます。

ワーカーノードをデプロイする

ローカルクラスターにワーカーノードをデプロイするには、セルフマネージドノードグループを使用します。

1. ワーカーノードの AWS CloudFormation テンプレートをデプロイするか、Outposts に EC2 インスタンスを個別に作成します。

```
curl -O https://s3.us-west-2.amazonaws.com/amazon-eks/cloudformation/2025-11-24/
amazon-eks-outpost-nodegroup.yaml
```

2. クラスターにノードインスタンスロールアクセスを付与します。ノードの IAM ロールに対してアクセスエントリを作成します。

```
aws eks create-access-entry \
  --cluster-name my-cluster \
  --principal-arn arn:aws:iam::111122223333:role/myNodeRole \
  --type EC2_LINUX
```

   あるいは、認証に aws-auth ConfigMap を使用している場合は、ConfigMap にノードインスタンスロールを追加します。「the section called “Kubernetes API アクセス”」を参照してください。

3. ノードが Ready 状態であることを確認します。

```
kubectl get nodes
```

内部リソース

ローカルクラスターを作成すると、Amazon EKS は kube-system 名前空間に coredns、kube-proxy、aws-node (VPC CNI) の各アドオンのセルフマネージドバージョンを自動的にインストールします。必要に応じて、Amazon EKS アドオン API を使用して、こうしたアドオンのマネージドバージョンをインストールできます。詳細については、「the section called “Amazon EKS アドオン (EC2 インスタンスストア)”」を参照してください。

Amazon EKS はこのほか、クラスターと VPC 間の通信用にクロスアカウントの Elastic Network Interface (ENI) もサブネットに作成します。こうしたネットワークインターフェイスは削除しないでください。クロスアカウントの ENI が削除された場合や、その IP アドレスが変更された場合は、静的 IP ベースのアクセスを使用するノードと管理者をすべて手動で更新する必要があります。

EC2 インスタンスストアを設定した AWS Outposts に Amazon EKS ローカルクラスターの VPC とサブネットを作成する

ローカルクラスターを作成する際には、VPC と、Outposts で実行されるプライベートサブネットを少なくとも 1 つ指定します。このトピックでは、ローカルクラスターの VPC とサブネットの要件を簡単に説明します。

> **Note**
>
> Outpost が EC2 インスタンスストアではなく Amazon EBS で設定されている場合、このトピックで説明されているアーキテクチャは Outpost に使用できません。EBS を設定した Outposts は、既存のローカルクラスターの実装を引き続き使用します。詳細については、「the section called “VPC とサブネットを作成する”」を参照してください。
> 更新されたローカルクラスターアーキテクチャを使用して EBS-backed Outpost にローカルクラスターを作成する場合は、AWS アカウントチームにお問い合わせください。

VPC の要件

- VPC には、作成するローカルクラスター、任意のノード、およびその他の Kubernetes リソースで利用できるだけの十分な数の IP アドレスが必要です。クラスターの作成前や作成後に、追加の CIDR ブロックを VPC に関連付けることができます。新規に関連付けた CIDR ブロックが認識されるまでには最大 1 時間かかる場合があります。
- VPC には、DNS ホスト名と DNS 解決のサポートが必要です。そうではない場合、ノードはクラスターに登録されません。詳細については、「Amazon VPC ユーザーガイド」の「VPC の DNS 属性」を参照してください。
- ローカルネットワーク経由でローカルクラスターにアクセスするには、VPC が Outpost のローカルゲートウェイルートテーブルに関連付けられている必要があります。詳細については、「AWS Outposts ユーザーガイド」の「VPC アソシエーション」を参照してください。

サブネットの要件

ローカルクラスターを作成する際には、Outposts のプライベートサブネットを少なくとも 1 つ指定します。Amazon EKS は、指定のサブネットにクロスアカウントの Elastic Network Interface を 3 つ作成します。これらのネットワークインターフェイスは、クラスターと VPC 間の通信を可能にします。

指定するサブネットは、次の要件を満たしている必要があります。

- すべてのサブネットが同じロジカル Outposts に存在すること。
- すべてのサブネットが同一の VPC に存在すること。
- すべてのサブネットが、Outposts のホームであるアベイラビリティーゾーンに存在する必要があります。
- Amazon EKS でクロスアカウントの Elastic Network Interface を作成するには、各サブネットで少なくとも 3 つの IP アドレスが使用可能である必要があります。
- サブネットでは IP アドレスベースの命名を使用する必要があります。Amazon EC2 のリソースベースの命名は、Amazon EKS でサポートされていません。
- ローカルネットワーク経由で Kubernetes API サーバーにアクセスできるように、サブネットに Outposts ラックのローカルゲートウェイへのルートがあること。サブネットにローカルゲートウェイへのルートがない場合は、VPC 内から Kubernetes API サーバーと通信すること。
- サブネットは、以下のアベイラビリティーゾーンには配置できません。

AWS リージョン	リージョン名	拒否されたアベイラビリティーゾーン ID
us-east-1	米国東部 (バージニア北部)	use1-az3
us-west-1	米国西部 (北カリフォルニア)	usw1-az2
ca-central-1	カナダ (中部)	cac1-az3

AWS サービスへのサブネットアクセス

ローカルクラスターでクラスター管理の操作、etcd のバックアップ、コントロールプレーンの更新を行うには、AWS リージョンへの接続が必要です。切断された状態の場合、ローカルクラスターは引き続き動作できますが、Amazon EKS が実行できるクラスター管理の操作は制限されます。詳細については、「the section called “切断への備え (EC2 インスタンスストア)”」を参照してください。

EC2 インスタンスストアを設定した AWS Outposts にある Amazon EKS ローカルクラスターのインスタンスタイプとプレイスメントを選択する

このトピックでは、Kubernetes コントロールプレーンインスタンスタイプを選択し、プレイスメントを設定して、EC2 インスタンスストアを設定した AWS Outpost にあるローカル Amazon EKS クラスターの高可用性要件を満たすためのガイダンスを提供します。

> **Note**
>
> Outpost が EC2 インスタンスストアではなく Amazon EBS で設定されている場合、このトピックで説明されているアーキテクチャは Outpost に使用できません。EBS を設定した Outposts は、既存のローカルクラスターの実装を引き続き使用します。詳細については、「the section called “容量に関する考慮事項”」を参照してください。
> 更新されたローカルクラスターアーキテクチャを使用して EBS-backed Outpost にローカルクラスターを作成する場合は、AWS アカウントチームにお問い合わせください。

コントロールプレーンアーキテクチャ

ローカルクラスターの Kubernetes コントロールプレーンは、Outpost の 6 つの EC2 インスタンスで実行されます。

- 3 つのコントロールプレーンインスタンス — Kubernetes API サーバー、スケジューラ、コントローラーマネージャーなどの Kubernetes コントロールプレーンコンポーネントをホストします。
- 3 つの **etcd** インスタンス — Kubernetes クラスターの状態を外部 etcd トポロジーに保存します。

これらのインスタンスは、Outpost の AWS マネージドサービスアカウントで実行されます。これらは、AWS アカウントまたは Amazon EC2 コンソールでは EC2 インスタンスとして表示されません。コントロールプレーンによって消費されるキャパシティは、AWS Outposts コンソールに表示されます。

コントロールプレーンは、負荷がかかっても水平スケーリングされません。別のコントロールプレーンインスタンスサイズが必要な場合は、新しいクラスターを作成する必要があります。

インスタンスタイプの選択

AWS Outposts にある Amazon EKS ローカルクラスターの更新されたアーキテクチャでは、Kubernetes コントロールプレーンは専用の `etcd` トポロジーを使用します。つまり、API サーバーと `etcd` は個別のインスタンスで実行されます。クラスターを作成するときは、API サーバー用の `controlPlaneInstanceType` と `etcd` 用の `etcdInstanceType` の 2 つのインスタンスタイプを指定します。

選択するインスタンスタイプは Outpost で使用できる必要があります。使用可能なインスタンスファミリーは Outpost の世代と SKU に依存し、異なるインスタンスファミリーは同じインスタンスサイズで異なる vCPU とメモリ設定を持つことがあります。

API サーバー**`controlPlaneInstanceType`** ()

想定されるワーカーノード	vCPU	メモリ
1 ~ 20	2	8 GiB
21 ~ 100	4	16 GiB
101 ~ 250	8	32 GiB
251 ~ 500	16	64 GiB

`etcd` (**`etcdInstanceType`**)

想定されるワーカーノード	vCPU	メモリ
1 ~ 50	2	8 GiB
51 ~ 250	4	16 GiB
251 ~ 500	8	32 GiB

> **Important**
>
> ローカルクラスターの存続期間中、Outpost に controlPlaneInstanceType の 3 つのインスタンスと etcdInstanceType の 3 つのインスタンスのためのキャパシティがあることを確認します。

これらの表は、厳格な要件ではなくガイドラインとして使用してください。クラスターの作成後は controlPlaneInstanceType と etcdInstanceType を変更できないため、予想される増加に対応するインスタンスファミリーとサイズを選択することをお勧めします。いずれかを変更するには、新しいクラスターを作成する必要があります。

コントロールプレーンと etcd のプレイスメント

高可用性を実現するために、コントロールプレーンと etcd のインスタンスを複数のハードウェア障害ドメインに分散させることをお勧めします。この分散は、outpostConfig の controlPlanePlacement と etcdPlacement の spreadLevel プロパティを使用して制御できます。spreadLevel を設定すると、Amazon EKS は分散戦略を使用して EC2 プレイスメントグループを作成します。プレイスメントグループを事前に作成する必要はありません。

> **Note**
>
> controlPlanePlacement の groupName パラメータは、インスタンスストアベースの EC2 インスタンスを実行する Outposts では使用されません。以前にローカルクラスターでプレイスメントグループを使用していた場合は、代わりに spreadLevel を使用するようになりました。

分散レベル

spreadLevel プロパティは 2 つの値を受け入れます。

- **host** — コントロールプレーンインスタンスを異なる物理ホストに分散させます。選択したインスタンスタイプで設定されたホストが少なくとも 3 つ必要です。
- **rack** — コントロールプレーンインスタンスを異なる物理 Outpost ラックに分散させます。選択したインスタンスタイプで設定されたホストがあるラックが少なくとも 3 つ必要です。

容量プランニング

ローカルクラスターの Outpost のキャパシティを計画する場合は、次のものを考慮します。

- コントロールプレーンと **etcd** のインスタンス: クラスターあたり 6 つのインスタンス (`controlPlaneInstanceType` の 3 つと `etcdInstanceType` の 3 つ)。
- ワーカーノード: セルフマネージド型ノードグループの EC2 インスタンス。
- 仮想化ホスト: コントロールプレーンは仮想化ホストで実行する必要があります。Outpost には、十分なキャパシティを持つ仮想化ホストが必要です。

プレイスメントの問題のトラブルシューティング

分散レベルを指定した後にクラスターが長期間 `CREATING` 状態または `UPDATING` 状態のままになった場合は、分散トポロジーを満たすために選択したインスタンスタイプを持つ十分なホストまたはラックが Outpost にあることを確認します。

ローカルクラスターのトラブルシューティングの詳細については、「the section called “クラスターをトラブルシューティングする (EC2 インスタンスストア)”」を参照してください。

EC2 インスタンスストアを設定した AWS Outposts にローカルの Amazon EKS クラスターを準備して、ネットワークの切断に備える

ローカルネットワークを AWS Cloud へとつなぐ AWS Outposts サービスリンクが接続を失った場合でも、Outposts にあるローカルの Amazon EKS クラスターは引き続き使用できます。このトピックでは、ネットワーク切断に備えてローカルクラスターを準備する方法とその際の考慮事項について説明します。

- ローカルクラスターにより、計画外の一時的なネットワーク切断中も安定してオペレーションを継続できます。AWSOutposts は、データセンター内の AWS Cloud の拡張機能として機能する完全に接続された製品であることに変わりはありません。Outposts と AWS Cloud との間のネットワーク接続が切断された場合には、接続の復元を試みることをお勧めします。手順については、「AWS Outposts ユーザーガイド」の「AWS Outposts ラックネットワークのトラブルシューティングチェックリスト」を参照してください。
- Outposts は、Outpost の接続状態を監視するために使用できる `ConnectedStatus` 指標を出力します。詳細については、AWS Outposts ユーザーガイドの「Outposts メトリクス」を参照してください。

ネットワーク切断中の認証

ローカルクラスターは、複数の認証メカニズムをサポートしています。それぞれのメカニズムでネットワーク切断時の可用性が異なります。

認証メカニズム	切断中に使用可能か?
AWS IAM (アクセスエントリ、`aws-auth` ConfigMap)	不可。IAM には AWSリージョンへの接続が必要です。
OIDC (お客様提供のプロバイダー)	プロバイダーの所在地によって異なります。OIDC プロバイダーが Outposts のローカルネットワークから到達可能であれば、認証は引き続き機能します。
x.509 クライアント証明書	はい。証明書は、Kubernetes API サーバーによってローカルに検証されます。
サービスアカウント用 IAM ロール (IRSA、IAM roles for service accounts)	不可。「the section called “切断中の IRSA と Pod Identity”」を参照してください。
EKS Pod Identity	不可。「the section called “切断中の IRSA と Pod Identity”」を参照してください。

x.509 クライアント証明書

ネットワーク切断中も `kubectl` へのアクセスを維持するには、切断が発生する前にクライアント X.509 証明書を作成しておきます。

管理者証明書を作成するには:

1. プライベートキーと証明書署名リクエスト (CSR) を生成します。

```
openssl req -new -newkey rsa:4096 -nodes \
    -keyout admin.key -out admin.csr -subj "/CN=admin"
```

2. Kubernetes `CertificateSigningRequest` リソースを作成して承認します。

```
cat admin.csr | base64 | tr -d '\n' > admin.csr.b64
```

```
apiVersion: certificates.k8s.io/v1
kind: CertificateSigningRequest
metadata:
  name: admin-csr
spec:
  request: <base64-encoded-csr>
  signerName: kubernetes.io/kube-apiserver-client
  usages:
    - client auth
```

```
kubectl apply -f admin-csr.yaml
kubectl certificate approve admin-csr
```

3. 署名付き証明書を取得する:

```
kubectl get csr admin-csr -o jsonpath='{.status.certificate}' | base64 --decode >
 admin.crt
```

4. ClusterRoleBinding を作成して管理者アクセスを付与します。

```
kubectl create clusterrolebinding admin --clusterrole=cluster-admin \
    --user=admin --group=system:masters
```

5. 証明書を使用する kubeconfig を構築します。

```
kubectl config --kubeconfig admin.kubeconfig set-cluster my-cluster \
    --certificate-authority=ca.crt --server $APISERVER_ENDPOINT --embed-certs
kubectl config --kubeconfig admin.kubeconfig set-credentials admin \
    --client-certificate=admin.crt --client-key=admin.key --embed-certs
kubectl config --kubeconfig admin.kubeconfig set-context admin@my-cluster \
    --cluster my-cluster --user admin
kubectl config --kubeconfig admin.kubeconfig use-context admin@my-cluster
```

切断中のクラスターエンドポイントの DNS 解決

ローカルクラスターの Kubernetes API サーバーエンドポイントは、Amazon Route 53 にホストされ、Amazon EKS がサブネットに作成するクロスアカウントの Elastic Network Interface (ENI) のプライベート IP アドレスに解決されます。これらの ENI には、通常のクラスターオペレーション中に変更されない静的なプライベート IP アドレスがあります。

ネットワークの切断中、Outposts は Route 53 に到達できないため、ローカルな解決パスを準備しない限り、クラスターエンドポイントのホスト名は解決されません。次の 3 つのカテゴリに属するクライアントが API サーバーに到達する必要があります。

- kubectl を実行しているクラスター管理者。
- ノードハートビートを送信し、仕様をプルするワーカーノード (kubelet)。
- 各ノード上の **kube-proxy**。これがクラスターサービス IP をセットアップします。

オプション 1: ローカルな DNS ソリューション (推奨)

AWS は、Outposts が切断されている間、クラスターエンドポイントレコードをキャッシュして処理する DNS ソリューションをローカルにデプロイすることを推奨しています。オンプレミス環境で、クラスターエンドポイントレコードをキャッシュする独自の DNS サーバーを実行できます。

DNS ソリューションをローカルに使用する場合は、kubeconfig とワーカーノード AMI がクラスターエンドポイントのホスト名 (ENI IP アドレスではない) を指すようにして、ローカルの DNS ソリューションと一致した解決にすることをお勧めします。

オプション 2: 静的 IP ベースのアクセス

ローカルの DNS ソリューションを実行しない場合は、静的 IP ベースのアクセスを使用できます。

- 管理者: クロスアカウントの ENI プライベート IP アドレスを直接指すように kubeconfig を設定します。ENI を検索するには、AWS アカウント内で説明に Amazon EKS *cluster-name* が含まれているネットワークインターフェイスを検索します。各 ENI の IP アドレスは、通常の運用であればクラスターが存続する間安定しています。
- ワーカーノード (Amazon EKS 最適化 AMI): Amazon EKS 最適化 AMI からワーカーノードを起動すると、ブートストラップスクリプトが ENI の IP アドレスを使用してクラスターエンドポイントを /etc/hosts に追加します。追加の設定は必要ありません。
- ワーカーノード (カスタム AMI): クラスターエンドポイントのホスト名と ENI の IP アドレスをカスタムブートストラップの /etc/hosts に追加します。そうしないと、切断されている間、kubelet と kube-proxy は API サーバーに到達できません。

> **Important**
>
> クロスアカウントの ENI が削除されたり、その IP アドレスが変更されたりした場合、例えば、そうした削除や変更のために Amazon EKS が再アタッチできなくなった場合は、静的

IP ベースのアクセスを使用するノードと管理者をすべて手動で更新する必要があります。ローカルの DNS ソリューションでは、手動による介入は必要ありません。

切断中のポッドの DNS 解決

切断中にオペレーションを実行しても DNS に障害が発生しないようにするには、kubelet's `resolvConf 設定を上書きするようにワーカーノード起動テンプレートを設定します。ユーザーデータで、カスタム resolv.conf ファイル (/etc/kubernetes/resolv.conf など) を作成して、nameserver 10.0.0.2 のみ (VPC 検索ドメインなし) を含め、NodeConfig に spec.kubelet.config.resolvConf: /etc/kubernetes/resolv.conf を設定します。これにより、ポッドの DNS 設定から *region-code*.compute.internal 検索ドメインが削除されるため、切断中は到達不能な VPC DNS リゾルバーにクエリが転送されなくなります。

次の例に、ワーカーノードのユーザーデータを示します。

```
MIME-Version: 1.0
Content-Type: multipart/mixed; boundary="BOUNDARY"

--BOUNDARY
Content-Type: text/x-shellscript; charset="us-ascii"

#!/bin/bash
mkdir -p /etc/kubernetes
echo "nameserver 10.0.0.2" > /etc/kubernetes/resolv.conf

--BOUNDARY
Content-Type: application/node.eks.aws

---
apiVersion: node.eks.aws/v1alpha1
kind: NodeConfig
spec:
  cluster:
    name: my-cluster
    ...
  kubelet:
    config:
      resolvConf: /etc/kubernetes/resolv.conf

--BOUNDARY--
```

切断中の IRSA と Pod Identity

> **Important**
>
> IRSA と EKS Pod Identity は、AWS リージョンで動作する AWS STS に依存します。ネットワークの切断中、IRSA または Pod Identity を使用するワークロードは新しい認証情報を取得できません。既存の認証情報は、一定期間後に期限切れになります。

ネットワークの切断中もワークロードを利用可能なままにする必要がある場合は、リージョンベースの AWS サービスに対して機能面や運用面で依存関係を持つことはお勧めしません。

切断中の `etcd` の動作

ネットワークの切断中、`etcd` スナップショットをバックアップすることはできません。切断中に複数の `etcd` インスタンスが使用できなくなった場合、`etcd` はクォーラムを失い、Kubernetes API オペレーションは Outposts が再接続して `etcd` クォーラムが復元されるまで使用できません。既に実行されているワークロードは引き続き動作します。

切断中のコントロールプレーンのログ記録

ネットワークの切断中に、コントロールプレーンのログはコントロールプレーンインスタンスにローカルにキャッシュされます。接続が復元されると、ログは親 AWS リージョンの Amazon CloudWatch Logs に送信されます。コントロールプレーンにログ記録エージェントをインストールまたは維持する必要はありません。

ローカルオブザーバビリティ

切断中にクラスターをローカルでモニタリングするには、Prometheus、Grafana、またはその他のサードパーティーソリューションを使用して Kubernetes API サーバーメトリクスエンドポイントをスクレイプします。

ローカルイメージリポジトリ

追加のレプリカでデプロイをスケールしたり、切断中のポッド障害から復旧したりするには、コンテナイメージリポジトリ (Docker レジストリなど) をローカルに保持するか、切断前にイメージをノードにキャッシュしておく必要があります。ネットワークの切断中、Amazon ECR は利用できません。

Kubernetes ポッドのフェイルオーバー動作を調整する

ネットワークの切断中、Kubernetes コントロールプレーンは AWS リージョンと通信できません。ノードが到達不能になった場合、Kubernetes のデフォルトの動作はタイムアウトの期間後にポッドを削除することです。許容範囲と `tolerationSeconds` ポッド仕様に沿ってこの動作を調整して、パーティション中にポッドを再スケジュールする速度を制御できます。詳細なガイダンスと例については、「Amazon EKS ベストプラクティスガイド」の「Kubernetes ポッドのフェイルオーバー動作を調整する」(https://docs.aws.amazon.com/eks/latest/best-practices/hybrid-nodes-network-disconnection-best-practices.html#tune_kubernetes_pod_failover_behavior) を参照してください。

ネットワークの切断をシミュレートする

ローカルクラスターで本番環境に入る前に、切断をシミュレートして、接続されていない状態でもクラスターにアクセスできることを確認できます。

1. Outpostを AWS リージョンに接続するネットワークデバイスにファイアウォールルールを適用します。これにより、Outpost のサービスリンクが切断されます。
2. 作成した X.509 証明書を使用して、ローカルクラスターへの接続をテストします。

```
kubectl --kubeconfig admin.kubeconfig get nodes
```

> **Note**
>
> Outposts でサービスが既に本番稼働環境にある場合は、切断をシミュレートしないでください。サービスリンクを切断すると、Outposts で実行中のすべてのサービスに影響が及びます。

EC2 インスタンスストアを設定した AWS Outposts にあるローカルクラスターの Amazon EKS アドオン

EC2 インスタンスストアを設定した AWS Outposts にあるローカルクラスターは、Amazon EKS アドオンをサポートします。次のアドオンは、ローカルクラスターでの使用が検証されています。時間の経過に伴い追加のアドオンが検証されます。

> **Note**
>
> Outpost が EC2 インスタンスストアではなく Amazon EBS で設定されている場合、このトピックで説明されているアーキテクチャは Outpost に使用できません。EBS を設定した Outposts は、Amazon EKS アドオンをサポートしていない既存のローカルクラスターの実装を引き続き使用します。
> 更新されたローカルクラスターアーキテクチャを使用して EBS-backed Outpost にローカルクラスターを作成する場合は、AWS アカウントチームにお問い合わせください。

検証済みアドオン

アドオン	説明
Amazon VPC CNI plugin for Kubernetes	ポッドのネイティブ VPC ネットワーキングを提供します。
kube-proxy	Kubernetes サービス通信のノードのネットワークルールを維持します。
CoreDNS	Kubernetes サービスの DNS 解決を提供します。
EKS Pod Identity Agent	ポッドが EKS Pod Identity を使用して AWS サービスに対して認証できるようにします。

Amazon VPC CNI プラグイン、`kube-proxy`、および CoreDNS のカスタマーマネージドバージョンは、ローカルクラスターを作成するときに自動的にインストールされます。必要に応じて、これらのアドオンのマネージドバージョンをインストールできます。

> **Important**
>
> EKS Pod Identity と IRSA は、AWS リージョンの AWS STS に依存します。ネットワークの切断中、これらのメカニズムを使用するワークロードは新しい認証情報を取得できません。詳細については、「the section called “切断への備え (EC2 インスタンスストア)”」を参照してください。

未検証のアドオン

上記以外のアドオンは、ローカルクラスターでの使用が検証されていません。未検証のアドオンの使用はお勧めしません。

追加のアドオンが検証されると、このページが更新されます。

代替 CNI プラグイン

Amazon VPC CNI は、Outposts での Amazon EKS デプロイで正式にサポートされている CNI です。代替 CNI プラグインを使用する場合は、「the section called “代替 CNI プラグイン”」を参照してください。

EC2 インスタンスストアを設定した AWS Outposts にあるローカルの Amazon EKS クラスターをトラブルシューティングする

このトピックでは、EC2 インスタンスストアを設定した AWS Outposts でローカルの Amazon EKS クラスターの使用中に表示される可能性がある一般的なエラーとそのトラブルシューティング方法について説明します。

`kubeconfig` の設定後に `kubectl` が動作しない

`aws eks update-kubeconfig` の実行後に次のエラーが表示される場合:

```
Either parameter --cluster-name or --cluster-id must be specified.
```

または:

```
couldn't get current server API group list: getting credentials: exec: executable aws
 failed with exit code 1
```

kubeconfig (`~/.kube/config`) では、`--cluster-id` ではなく引数 `exec` に `--cluster-name` を指定していることを確認します。

```
      args:
        - --region
        - region-code
        - eks
```

```
      - get-token
      - --cluster-name
      - my-cluster
      - --output
      - json
    command: aws
```

ノードがクラスターに参加していない

`kubectl get nodes` の実行時にノードが表示されない場合:

1. ノードで **kubelet** ログを確認します。SSM または SSH でノードにアクセスし、以下を実行します。

   ```
   systemctl status kubelet -l
   ```

2. 不正なエラーがないか確認します。不正なエラーがある場合、ノードロールにアクセス許可がないか、`aws-auth` ConfigMap に問題が発生している可能性があります。ノードのインスタンスロールが `aws-auth` ConfigMap に正しくマッピングされていることを確認します。
3. CSR ステータスを確認します。ノードの証明書署名リクエストが承認済みであることを確認します。

   ```
   kubectl get csr
   ```

   CSR が `Pending` 状態の場合は、承認します。

   ```
   kubectl certificate approve node-csr-EXAMPLE
   ```

クラスターが **CREATING** 状態でスタックしています

クラスターが FAILED ステータスになった場合:

- 容量不足: 指定されたインスタンスタイプの 6 つのコントロールプレーンインスタンス (3 つの `etcd` + 3 つの API サーバー) に対応できるように、Outposts の仮想化容量が十分に確保されていることを確認します。インスタンスタイプは、Outposts にスロットされている必要があります。
- スプレッドトポロジを満たせない: `host` の `spreadLevel` を指定した場合は、選択したインスタンスタイプで少なくとも 3 つのホストが設定されていることを確認します。`rack` を指定した場合は、選択したインスタンスタイプのホストが少なくとも 3 つのラックにあることを確認します。

- サブネットの問題: 指定したサブネットが Outposts のホームであるアベイラビリティーゾーンにあり、各サブネットに使用可能な IP アドレスが少なくとも 3 つあることを確認します。「the section called “VPC とサブネットを作成する (EC2 インスタンスストア)”」を参照してください。

クラスターが **UPDATING** 状態でスタックしています

Kubernetes バージョンまたはプラットフォームバージョンの更新中にクラスターが UPDATING 状態のままである場合:

- 容量不足: 更新中に、コントロールプレーンインスタンスは再インスタンス化の前に削除されます。置換先インスタンスをプロビジョニングする前に、解放された容量が別のワークロードによって消費された場合、更新は停止します。コントロールプレーンインスタンスタイプに使用可能な容量が Outposts にあることを確認します。

etcd のクォーラム損失

Kubernetes API オペレーションを実行できなくなり、Outposts との接続が切断された場合:

- `etcd` のクォーラムが失われた可能性があります。これが発生するのは、切断中に複数の `etcd` インスタンスが使用できなくなった場合です。
- 既に実行中のワークロードは引き続き動作しますが、Kubernetes API オペレーションは使用できなくなります。
- 接続が復元されると、Amazon EKS は最新の `etcd` スナップショットからクラスターの状態を復旧します。

AWS サポートセンターにお問い合わせください。

AWS Outposts で Amazon Linux ノードを作成する

> **Important**
>
> Outposts 上の Amazon EKS Local Clusters は、次の Amazon EKS 最適化 Amazon Linux 2023 AMI から作成されたノードのみをサポートします。
>
> - 標準 Amazon Linux 2023 (`amazon-linux-2023/x86_64/standard`)
> - 高速 Nvidia Amazon Linux 2023 (`amazon-linux-2023/x86_64/nvidia`)
> - 高速 Neuron Amazon Linux 2023 (`amazon-linux-2023/x86_64/neuron`)

- Bottlerocket (EC2 インスタンスストアがサポートする Outposts 専用)

AWS は、EKS AL2 最適化 AMI と AL2 高速化 AMI に対するサポートを、2025 年 11 月 26 日に終了しました。お客様は、こちらのサポート終了 (EOS) の日付 (2025 年 11 月 26 日) 以降も EKS AL2 AMI をご使用いただけますが、EKS はこの日以降、新しい Kubernetes バージョンや AL2 AMI の更新 (マイナーリリース、パッチ、バグ修正を含む) をリリースすることはありません。AL2 の非推奨の詳細については、こちらを参照してください。

このトピックは Amazon EKS クラスターに登録されている Outpost 上の Amazon Linux ノードの 自動スケーリング グループを起動する方法を説明します。クラスターはAWS クラウド または アウトポスト に置くことができます。

- 既存の アウトポスト。詳細については「AWS アウトポスト とは」を参照してください。
- 既存の Amazon EKS クラスター。AWS クラウド にクラスターをデプロイするには「the section called “クラスターを作成する”」を参照してください。Outpost にクラスターをデプロイするには「the section called “ローカルクラスタを実行する”」を参照してください。
- AWS クラウド のクラスターにノードを作成しており、AWS Outposts、AWS 波長、または AWS ローカルゾーン が有効になっている AWS リージョンにサブネットがあるとします。この場合、クラスターを作成したときに、これらのサブネットが渡されていない必要があります。Outpost 上のクラスターにノードを作成する場合はクラスターの作成時に Outpost サブネットを渡しておく必要があります。
- (AWS クラウド上のクラスターに推奨) 必要な IAM ポリシーがアタッチされた独自の IAM ロールで設定された Amazon VPC CNI plugin for Kubernetes アドオン。詳細については、「the section called “IRSA 用の設定”」を参照してください。ローカルクラスターはサービスアカウントの IAM ロールをサポートしていません。

`eksctl` または AWS マネジメントコンソール (AWS クラウドFormation テンプレートを使用) で、セルフマネージド型の Amazon Linux ノードグループを作成できます。Terraform を使用することもできます。

このページで説明されている以下のツールを使用して、ローカルクラスターのセルフマネージドノードグループを作成できます。

- the section called “`eksctl`”
- the section called “AWS マネジメントコンソール”

> **Important**
>
> - セルフマネージド型ノードグループにはアカウント内の Amazon EC2 インスタンスが含まれています。これらのインスタンスはお客様または代わりに Amazon EKS がコント役割プレーンのバージョンを更新しても、自動的にはアップグレードされません。セルフマネージド型ノードグループは更新が必要であることをコンソールに表示しません。更新が必要なノードを確認するにはクラスターの [概要] タブにある [ノード] のリストからノードを選択して、そこにインストールされている `kubelet` バージョンを表示します。ノードは手動で更新する必要があります。詳細については、「the section called “更新方式”」を参照してください。
> - セルフマネージド型ノードで kubelet が使用する証明書は、1 年の有効期限で発行されます。デフォルトでは、証明書のローテーションは有効になっていません (https://kubernetes.io/docs/reference/config-api/kubelet-config.v1beta1/#kubelet-config-k8s-io-v1beta1-KubeletConfiguration を参照)。つまり、セルフマネージド型ノードを 1 年以上にわたって実行し続けると、Kubernetes API に対して認証できなくなります。
> - ベストプラクティスとして、お客様は定期的にセルフマネージド型ノードグループを更新して、最新の Amazon EKS 最適化 AMI から CVE およびセキュリティパッチを受け取ることをお勧めします。セルフマネージド型ノードグループで使用される AMI を更新すると、ノードの再作成もトリガーされ、kubelet 証明書の有効期限が切れたことによる問題が発生しないようにします。
> - または、セルフマネージド型ノードグループの作成時にクライアント証明書のローテーション (https://kubernetes.io/docs/tasks/tls/certificate-rotation/ を参照) を有効にして、現在の証明書の有効期限が近づくと kubelet 証明書が更新されるようにすることもできます。

`eksctl`

`eksctl` を使用して自己管理型Linuxノードを起動するには

1. デバイスまたは AWS クラウドシェル にインストールされている `eksctl` コマンドラインツールのバージョン `0.215.0` 以降をインストールします。`eksctl` をインストールまたはアップグレードするには`eksctl` ドキュメントの「インストール」を参照してください。
2. クラスターが AWS クラウド にあり、AmazonEKS_CNI_Policy マネージド IAM ポリシーが Amazon EKS ノードの IAM ロールへアタッチされている場合は代わりに Kubernetes `aws-node` サービスアカウントに関連付けた IAM ロールに割り当てることをお勧めします。詳細について

は、「the section called “IRSA 用の設定”」を参照してください。クラスターが Outposts にある場合は、ポリシーをノードロールにアタッチする必要があります。

3. 次のコマンドは既存のクラスターにノードグループを作成します。クラスターはeksctl を使用して作成されている必要があります。*al-nodes* を、ノードグループの名前に置き換えます。ノードグループ名は 63 文字以下である必要があります。先頭は文字または数字でなければなりませんが、残りの文字にはハイフンおよびアンダースコアを含めることもできます。*#######* の部分は自分のクラスター名に置き換えます。この名前には英数字 (大文字と小文字が区別されます) とハイフンのみを使用できます。先頭の文字は英数字である必要があります。また、100 文字より長くすることはできません。名前はクラスターを作成する AWS リージョンおよび AWS アカウント内で一意である必要があります。クラスターが Outpost に存在する場合は*id* を Outpost サブネットの ID に置き換えます。AWS クラウド にクラスターが存在する場合、*id* をクラスターの作成時に指定しなかったサブネットの ID に置き換えます。残りのサンプル値は独自の値に置き換えます。デフォルトでは、ノードはコントロールプレーンと同じ Kubernetes バージョンで作成されます。

   ####### を Outpost で利用可能なインスタンスタイプに置き換えます。

   #### を Amazon EC2 キーペアまたはパブリックキーの名前に置き換えます。このキーは起動後のノードに SSH 接続するために使用されます。Amazon EC2 キーペアをまだ持っていない場合はAWS マネジメントコンソール で作成できます。詳細については「Amazon EC2 ユーザーガイド」の「Amazon EC2 キーペア」を参照してください。

   次のコマンドを使用して、ノードグループを作成します。

```
eksctl create nodegroup --cluster my-cluster --name al-nodes --node-type instance-
type \
    --nodes 3 --nodes-min 1 --nodes-max 4 --managed=false \
    --node-volume-type gp2 --subnet-ids subnet-id \
    --node-ami-family AmazonLinux2023
```

   クラスターを AWS クラウド 上にデプロイしている場合:

   - デプロイするノードグループではインスタンスのブロックとは異なる CIDR ブロックから IPv4 アドレスを Pod に割り当てることができます。詳細については、「the section called “カスタムネットワーキング”」を参照してください。
   - デプロイするノードグループはアウトバウンドインターネットアクセスを必要としません。詳細については「the section called “プライベートクラスター”」を参照してください。

利用できるすべてのオプションとデフォルトの詳細なリストについてはeksctl ドキュメントの「AWS Outposts サポート」を参照してください。

- ノードがクラスターに参加できない場合は「Amazon EKS クラスターとノードに関する問題をトラブルシューティングする」の「the section called “ノードがクラスターへの参加に失敗します”」および「AWS アウトポスト でローカル Amazon EKS クラスターをトラブルシューティングする」の「the section called “ノードをクラスターに結合できない”」を参照してください。
- 出力例は次のとおりです。ノードの作成中に、複数の行が出力されます。出力の最後の行は次のサンプル行が表示されます。

```
[#]  created 1 nodegroup(s) in cluster "my-cluster"
```

4. (オプション) サンプルアプリケーションをデプロイして、クラスターと Linux ノードをテストします。

AWS マネジメントコンソール

ステップ 1: AWS マネジメントコンソール を使用してセルフマネージド型の Linux ノードを起動する`

1. AWS クラウドフォーメーション テンプレートの最新バージョンをダウンロードします。

```
curl -O https://s3.us-west-2.amazonaws.com/amazon-eks/cloudformation/2025-11-24/
amazon-eks-outpost-nodegroup.yaml
```

2. AWS クラウドFormation コンソールを開きます。
3. [スタックの作成] を選択し、[新しいリソースを使用 (標準)] を選択してください。
4. [テンプレートの指定] で、[テンプレートファイルのアップロード] を選択し、[ファイルを選択] を選択してください。前のステップでダウンロードした amazon-eks-outpost-nodegroup.yaml ファイルを選択し、[次へ] を選択してください。
5. [スタックの詳細の指定] ページで、必要に応じて次のパラメータを入力し、[次へ] を選択してください：
   - スタック名: AWS クラウドFormation スタックのスタック名を選択してください：例えば、*al-nodes* という名前にすることができます。この名前には英数字 (大文字と小文字が区別されます) とハイフンのみを使用できます。先頭の文字は英数字である必要があります。また、100 文字より長くすることはできません。名前はクラスターを作成する AWS リージョンおよび AWS アカウント内で一意である必要があります。

- ApiServerEndpoint: EKS コンソール、または DescribeCluster API で表示される Kubernetes API Server エンドポイントを入力します。
- [クラスター名]: クラスターの名前を入力してください：この名前が、クラスター名と一致しない場合、ノードはクラスターに参加できません。
- ClusterId: EKS サービスによってクラスターに割り当てられた ID を入力します。DescribeCluster API で表示されます。この ID が、クラスター ID と一致しない場合、ノードはクラスターに参加できません。
- CertificateAuthority: Kubernetes の証明機関の base64 エンコード文字列を入力します。EKS コンソール、または DescribeCluster API で表示されます。
- ServiceCidr: Kubernetes サービス CIDR を入力します。EKS コンソール、または DescribeCluster API で表示されます。
- [クラスター制御プレーンセキュリティグループ]: VPC の作成時に生成した AWS クラウドフォーメーション 出力の [セキュリティグループs] 値を選択してください。

  次のステップでは該当するグループを取得する 1 つのオペレーションを説明します。

  i. Amazon EKS コンソール を開きます。

  ii. クラスターの名前を選択してください。

  iii. [ネットワーキング] タブを選択してください。

  iv. [クラスター制御プレーンセキュリティグループ] ドロップダウンリストから選択する場合は[追加のセキュリティグループ] の値をリファレンスとして使用します。
- [ノードグループ名]: ノードグループの名前を入力してください。この名前はノードに対して作成される自動スケーリングノードグループを識別するために後で使用できます。
- [ノード自動スケーリンググループ最小サイズ]: ノードの 自動スケーリング グループがスケールインできる最小ノード数を入力してください：
- ノード自動スケーリンググループ希望容量: スタック作成時にスケーリングする必要のあるノード数を入力してください。
- [NodeAutoScalingGroupMaxSize]: ノードのAuto Scaling グループがスケールアウトできる最大ノード数を入力してください。
- [ノードインスタンス型]: ノードのインスタンスタイプを選択してください：クラスターが AWS クラウド で動作している場合は詳細については「the section called “Amazon EC2 インスタンスタイプ”」を参照してください。クラスターが Outpost で実行されている場合、Outpost で使用できるインスタンスタイプのみを選択できます。
- [NodeImageIdSSMParam]: 最新の Amazon EKS 最適化 AMI の Amazon EC2 Systems Manager のパラメータが、可変 Kubernetes バージョン用に事前設定されています。Amazon EKS でサ

ポートされている別の Kubernetes マイナーバージョンを使用するには、*1.XX* を別のサポートされているバージョンに置き換えます。クラスターと同じ Kubernetes バージョンを指定することをお勧めします。

Amazon EKS 最適化高速 AMI を使用するには、*NodeImageIdSSMParam* 値を目的の SSM パラメータに更新します。SSM から EKS AMI ID を取得する方法については、こちらを参照してください。

> **Note**
>
> Amazon EKS ノード AMI は Amazon Linux をベースとしています。Amazon Linux のセキュリティまたはプライバシーに関するイベントを、Amazon Linux セキュリティセンターで追跡できます。そのためには目的のバージョンのタブを選択してください。該当する RSS フィードをサブスクライブすることもできます。セキュリティおよびプライバシーイベントには問題の概要、影響を受けるパッケージ、および問題を修正するためにインスタンスを更新する方法などがあります。

- ノードイメージId: (オプション) (Amazon EKS 最適化 AMI の代わりに) 独自のカスタム AMI を使用している場合はAWS リージョンのノード AMI ID を入力してください。ここで値を指定すると、[ノードイメージIdSSMParam] フィールドの値はすべて上書きされます。
- [ノードボリュームサイズ]: ノードのルートボリュームのサイズを GiB 単位で指定します。
- [ノードボリューム型]: ノードのルートボリュームタイプを指定します。
- [キー名]: 起動後に、SSH を使用してノードに接続するときに使用できる Amazon EC2 SSH キーペアの名前を入力してください。Amazon EC2 キーペアをまだ持っていない場合はAWS マネジメントコンソール で作成できます。詳細については「Amazon EC2 ユーザーガイド」の「Amazon EC2 キーペア」を参照してください。

> **Note**
>
> ここでキーペアを指定しないと、AWS クラウドフォーメーション スタックの作成は失敗します。

- [無効IMDSv1]: 各ノードはデフォルトでインスタンスメタデータサービスバージョン 1 (IMDSv1) および IMDSv2 をサポートします。IMDSv1 は無効にできます。ノードグループ内の将来のノードおよび Pod が MDSv1 を使用しないようにするには DisableIMDSv1 を true に設定します。IDMS の詳細については「インスタンスメタデータサービスの設定」を参照してくだ

さい。ノードでのそれへのアクセス制限について詳しくは[ワーカーノードに割り当てられたインスタンスプロファイルへのアクセスを制限する]を参照してください。

- [VpcId]: 作成した VPC の ID を入力します：VPC を選択する前に、「VPC の要件と考慮事項」を確認してください。
- [サブネット]: クラスターが Outpost にある場合、VPC 内で少なくとも 1 つのプライベートサブネットを選択してください：サブネットを選択する前に、「サブネットの要件と考慮事項」を確認してください。クラスターの [ネットワーキング] タブから、各サブネットリンクを開き、プライベートのサブネットを確認できます。

6. [スタックオプションの設定] ページで、希望する設定を選択し、[次へ] を選択してください。
7. [AWS クラウドフォーメーション が IAM リソースを作成する可能性を認識しています] の左にあるチェックボックスを選択して、[スタックの作成] を選択してください。
8. スタックの作成が完了したら、コンソールで選択し、[出力] を選択してください。
9. 作成されたノードグループの [ノードインスタンス役割] を記録します。これはAmazon EKS ノードを設定する際、必要になります。

ステップ 2: ノードを有効にしてクラスターに参加する

1. `aws-auth ConfigMap` がすでにあるかどうかを確認します。

```
kubectl describe configmap -n kube-system aws-auth
```

2. `aws-auth ConfigMap` が表示されている場合は必要に応じて更新してください。

   a. 編集する `ConfigMap` を開きます。

```
kubectl edit -n kube-system configmap/aws-auth
```

   b. 必要に応じて新しい `mapRoles` エントリを追加します。`rolearn` 値を、前の手順で記録した [ノードインスタンス役割] 値に設定します。

```
[...]
data:
  mapRoles: |
    - rolearn: <ARN of instance role (not instance profile)>
      username: system:node:{{EC2PrivateDNSName}}
      groups:
        - system:bootstrappers
        - system:nodes
```

```
[...]
```

c. ファイルを保存し、テキストエディタを終了します。

3. 「Error from server (NotFound): configmaps "aws-auth" not found」というエラーが表示されたら、ストック ConfigMap を適用してください。

a. 設定マップをダウンロードします。

```
curl -O https://s3.us-west-2.amazonaws.com/amazon-eks/cloudformation/2020-10-29/
aws-auth-cm.yaml
```

b. aws-auth-cm.yaml ファイルで、rolearn を前の手順で記録した [ノードインスタンス役割] 値に設定します。これを行うにはテキストエディタを使用するか、############ を置き換えて次のコマンドを実行してください：

```
sed -i.bak -e 's|<ARN of instance role (not instance profile)>|my-node-instance-
role|' aws-auth-cm.yaml
```

c. 設定を適用します。このコマンドが完了するまで数分かかることがあります。

```
kubectl apply -f aws-auth-cm.yaml
```

4. ノードのステータスを監視し、Ready ステータスになるまで待機します。

```
kubectl get nodes --watch
```

Ctrl+C を入力して、シェルプロンプトに戻ります。

> **Note**
>
> 認可またはリソースタイプのエラーが発生した場合はトラブルシューティングトピックの「the section called “許可されていないか、アクセスが拒否されました (kubectl)”」を参照してください。

ノードがクラスターに参加できない場合は「Amazon EKS クラスターとノードに関する問題をトラブルシューティングする」の「the section called “ノードがクラスターへの参加に失敗します”」および「AWS アウトポスト でローカル Amazon EKS クラスターをトラブルシューティングする」の「the section called “ノードをクラスターに結合できない”」を参照してください。

5. Amazon EBS CSI ドライバーをインストールします。詳細についてはGitHub の Installation を参照してください。[ドライバーのアクセス許可を設定] セクションでは[IAM インスタンスプロファイルの使用] オプションの指示に従うことを確認します。gp2 ストレージクラスを使用する必要があります。gp3 ストレージクラスはサポートされていません。

   クラスターの gp2 ストレージクラスを作成するには以下のステップを実行してください。

   a. 次のコマンドを実行して、gp2-storage-class.yaml ファイルを作成します。

```
cat >gp2-storage-class.yaml <<EOF
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  annotations:
    storageclass.kubernetes.io/is-default-class: "true"
  name: ebs-sc
provisioner: ebs.csi.aws.com
volumeBindingMode: WaitForFirstConsumer
parameters:
  type: gp2
  encrypted: "true"
allowVolumeExpansion: true
EOF
```

   b. マニフェストをクラスターに適用します。

```
kubectl apply -f gp2-storage-class.yaml
```

6. (GPU ノードのみ) GPU インスタンスタイプと Amazon EKS 最適化アクセラレーション AMI を選択した場合はクラスター上の DaemonSet として NVIDIA device plugin for Kubernetes を適用する必要があります。次のコマンドを実行する前に、*vX.X.X* を必要となる NVIDIA/k8s-device-plugin バージョンに置き換えます。

```
kubectl apply -f https://raw.githubusercontent.com/NVIDIA/k8s-device-plugin/vX.X.X/
deployments/static/nvidia-device-plugin.yml
```

ステップ 3: その他のアクション

1. (オプション) サンプルアプリケーションをデプロイして、クラスターと Linux ノードをテストします。

2. クラスターが Outpost にデプロイされている場合はこのステップをスキップしてください。クラスターが AWS クラウド にデプロイされている場合、次の情報はオプションです。AmazonEKS_CNI_Policy マネージド IAM ポリシーが Amazon EKS ノードの IAM ロールにアタッチされている場合は代わりに Kubernetes aws-node サービスアカウントに関連付けた IAM ロールに割り当てることをお勧めします。詳細については、「the section called “IRSA 用の設定”」を参照してください。

Amazon EKS における AI と機械学習の概要

> **Tip**
>
> 今後開催予定の Amazon EKS AI/ML ワークショップに登録してください。

Amazon Elastic Kubernetes Service (Amazon EKS) は、比類のない柔軟性と制御性によって、組織が AI や機械学習 (ML) のワークロードをデプロイ、管理、スケールできるようにするマネージド Kubernetes サービスです。Amazon EKS はアップストリーム Kubernetes 上に構築されているため、オープンソースのツールや AWS サービスとシームレスに統合しながら、既存の Kubernetes の専門知識を適用できます。

大規模なモデルのトレーニング、リアルタイムのオンライン推論の実行、生成 AI アプリケーションのデプロイなど、どのような用途であっても、Amazon EKS は AI/ML プロジェクトが必要とするパフォーマンス、スケーラビリティ、およびコスト効率を提供します。

AI/ML に Amazon EKS を使用する理由

Amazon EKS は、AI/ML プロジェクトに必要な制御、統合、パフォーマンス、スケーラビリティを提供します。アップストリーム Kubernetes 上に構築され、AWS サービスと統合された Amazon EKS は、複雑なワークロードをオーケストレーションしながら、既存の Kubernetes の専門知識を活用するのに役立ちます。AI/ML デプロイを初めて使用するチームでも、既存の Kubernetes スキルをそのまま移行でき、学習曲線も急ではありません。

Amazon EKS は、オペレーティングシステムのカスタマイズからコンピューティングのスケーリングまで、あらゆることをサポートし、将来のインフラストラクチャに関する意思決定において選択肢を維持するための技術的な柔軟性を促進します。プラットフォームは、AI/ML ワークロードに必要なパフォーマンスとチューニングのオプションを提供し、以下の機能を含みます。

- 完全なクラスター制御: 隠れた抽象化なしで、コストと設定をファインチューニングできます。
- 1 秒未満のレイテンシー: 本番環境でリアルタイム推論ワークロードを実行できます。
- 高度なカスタマイズ: マルチインスタンス GPU、ネットワークチューニング、オペレーティングシステムレベルのチューニングを設定できます。
- 統合オーケストレーション: AI/ML パイプラインとオンプレミス、エッジ、クラウド環境間でオーケストレーションできます。

- コスト最適化: 自動スケーリング、ネイティブ GPU スケジューリング、多様な GPU およびアクセラレーターインスタンスタイプを使用できます。

主なユースケース

Amazon EKS は、以下の一般的なユースケースを含む、幅広い AI/ML ワークロードをサポートしています。

- 推論: 低レイテンシーの応答時間を必要とするユースケース向けに、Amazon EKS 上でモデルをセルフホストします。
- バッチ推論: スケジュールされたジョブを通じて大規模なデータセットを効率的に処理します。
- モデルトレーニング: 大規模なデータセットで複雑なモデルを長期間トレーニングします。
- モデルのファインチューニング: 独自のドメイン知識でオープンソースモデルを強化します。
- 検索拡張生成 (RAG) パイプライン: 取得プロセスと生成プロセスを統合します。
- エージェンティック AI: Amazon Bedrock、サードパーティー、または Amazon EKS でホストされているモデルを使用してエージェントをデプロイします。

ケーススタディ

顧客は、以下の導入事例で示されているように、GPU 使用率の最適化や 1 秒未満のレイテンシーでの推論ワークロードの実行など、さまざまな理由で Amazon EKS を選択しています。Amazon EKS のすべての導入事例のリストについては、「AWS のお客様の成功事例」を参照してください。

- BMW グループは、2,500 万台以上のコネクテッド車両を持つ世界最大規模のコネクテッドフリートの 1 つを運営し、Amazon EKS 上に Ray を用いた分散トレーニングと Karpenter による GPU 自動スケーリングを備えたコネクテッド AI プラットフォームを構築しています。これにより、モデルトレーニング時間を数時間から 30 分に短縮し、1 回のトレーニング実行あたり 5 EUR のコストを実現しつつ、60 以上の AI ユースケースに対応する 550 人以上のデベロッパーをサポートしています。
- 世界をリードする旅行プラットフォームの 1 つである Booking.com は、実験のためのスケーラビリティを実現するため、検索ランキングの ML 推論システムを Amazon EKS に移行し、1 秒あたり最大 25 万のリクエストを 40 ミリ秒 (p99.9) のレイテンシーで処理できるようにしました。
- Unitary は、コンテンツモデレーションに AI を使用して 1 日あたり 2,600 万本の動画を処理しています。同社は高スループットかつ低レイテンシーの推論を必要としており、コンテナのブート時

間を 80% 削減することに成功しました。これにより、トラフィックの変動に応じたスケーリングイベントに対して迅速に対応できるようになっています。

- Synthesia は、顧客がテキストプロンプトからリアルな動画を作成できる生成 AI 動画作成をサービスとして提供しています。ML モデルトレーニングのスループットが 30 倍向上しました。
- Ada Support は、AI を活用したカスタマーサービス自動化企業であり、コンピューティングコストの 15% の削減と、コンピューティング効率の 30% の向上を達成しました。
- Snorkel AI は、企業が基盤モデルと大規模言語モデルを構築および適応できるように支援しています。GPU リソースにインテリジェントなスケーリングメカニズムを実装することで、40% 以上のコスト削減を達成しました。
- Artera は Amazon Elastic File System (Amazon EFS) と Amazon EKS を使用し、高解像度の生検画像を用いて癌治療をパーソナライズする ML モデルをトレーニングしています。
- Anthropic は、主力製品である Claude ファミリーの基盤モデルを Amazon EKS 上で運用しており、本番環境では最大規模の EKS クラスターの一部を稼働させています。これらのクラスターは、AI ワークロード用の AWS Trainium (trn2) インスタンスおよびと NVIDIA GPU と、CPU 集約型データ処理用の AWS Graviton プロセッサで構成されています。

ガイド構造

このガイドには、Amazon EKS で AI/ML ワークロードをデプロイおよび管理するための手順をステップバイステップで説明した一連の実践的なガイドが含まれています。各ガイドには、環境に直接実装できる手順と設定が記載されています。

このガイドでは、手順に加えて、各トピックに必要な背景と基本的な概念を提供します。また、必要となるより詳細な技術情報を参照できるよう、関連する AWS ドキュメントやリソースへのリンクも含まれています。

Amazon EKS で AI/ML の使用を開始する

Amazon EKS で AI/ML プラットフォームとワークロードの計画と使用を開始するには、the section called “クラスターを設定する” セクションに従って、必要な Kubernetes コンポーネントを含む Amazon EKS クラスターを AWS アカウントに作成します。環境の準備が完了し、稼働状態になったら、次のステップに進むことができます。

- the section called “クラスターを設定する”: このセクション全体で使用する Amazon EKS クラスター、モニタリング、Amazon S3 バケットインフラストラクチャを作成します。

- the section called “推測”: Amazon EKS を使用して、大規模言語モデル (LLM) を用いた推論アプリケーションをデプロイし、設定して使用を開始します。
- the section called “クラスターの設定”: AI/ML ワークロード用に最適化された Amazon EKS クラスターを設定します。
- the section called “容量管理”: Amazon EKS 上で機械学習ワークロードのコンピューティングリソースを管理および最適化します。
- the section called “デバイスの管理”: 動的リソース割り当て (DRA) とデバイスプラグインを使用して、特殊なハードウェアデバイスを管理します。

AI/ML ワークロード用の Amazon EKS クラスターを設定する

> **Tip**
>
> 今後開催予定の Amazon EKS AI/ML ワークショップに登録してください。

このセクションでは、GPU、モデルの重み用の Amazon S3 ストレージ、必要な AWS IAM アクセス許可など、推論ワークロードを実行する準備ができている Amazon EKS クラスターを作成する方法について説明します。

アーキテクチャの概要

セットアップでは、以下のインフラストラクチャが作成されます。

- GPU 対応ノードを備えた EKS クラスター — オンデマンドフォールバックを備えたスポットキャパシティを使用して G ファミリー GPU インスタンスを動的にプロビジョニングする Karpenter マネージド NodePool。
- スタックのモニタリング — Prometheus はクラスター、ノード、GPU メトリクスをスクレイプし、Amazon Managed Service for Prometheus (AMP) にリモート書き込みます。Grafana は視覚化用のダッシュボードを提供します。NVIDIA DCGM Exporter は、使用率、メモリ、温度、消費電力、NVLink 帯域幅、テンソルアクティビティなどの GPU 固有のメトリクスを追加します。
- モデルの重み S3 バケット — モデルの重みを保存するための Amazon S3 バケット。ワークロードポッドに読み取り/書き込みアクセスを許可する EKS Pod Identity の関連付けがあります。

クラスターコンピューティングオプション

このガイドには、クラスターを設定するための 2 つのパスが記載されています。1 つを選択し、すべてのステップを一貫して実行します。

- EKS Auto Mode — EKS Auto Mode が有効になっている EKS クラスターを 1 つのコマンドでプロビジョニングします。Karpenter ベースの自動スケーリング、EKS ノードモニタリングエージェント、SOCI を使用した高速コンテナプル、NVIDIA デバイスプラグインなど、必要なコンポーネントはすべてすぐに使用できる状態で提供されます。
- セルフマネージド Karpenter — 各コンポーネントを明示的にインストールして設定します。Karpenter は `eksctl` を使用して、自動ノード修復は機能ゲートを介して、EKS ノードモニタリングエージェントは EKS アドオンとして、NVIDIA デバイスプラグインは Helm を使用して行います。また、EKS 最適化 NVIDIA AL2023 AMI を使用して SOCI を設定するカスタム `EC2NodeClass` を作成します。

設定内容

Step	説明
クラスターを作成する	GPU ワークロードに必要な EKS コントロールプレーンとクラスターレベルのコンポーネントをプロビジョニングします。
動的にプロビジョニングされた GPU ノードを作成する	ワークロードがスケジュールされたときに G ファミリー GPU インスタンスをプロビジョニングする動的 GPU NodePool を定義します。
サンプルポッドでテストする	Karpenter をトリガーして GPU 対応ノードをプロビジョニングする `nvidia-smi` ポッドを実行して、セットアップをエンドツーエンドで検証します。
リザーブドキャパシティを追加する (オプション)	リザーブドキャパシティを優先し、スポット/オンデマンドにフォールバックするために、オンデマンドキャパシティ予約 (ODCR) を NodeClass にアタッチします。
モニタリングをインストールする	AMP へのリモート書き込みと GPU メトリクス用の NVIDIA DCGM Exporter を使用して、kube-prometheus-stack (Prometheus + Grafana) をデプロイします。

Step	説明
モデルの重みバケットを作成する	S3 バケットを作成し、ワークロードポッドがモデルの重みを読み書きできるように EKS Pod Identity を設定します。

はじめに

AWS CLI を使用したステップバイステップの手順については、「the section called “コマンドラインインターフェイス (CLI)”」を参照してください。Terraform を使用したステップバイステップの手順については、the section called “Terraform” を参照してください。

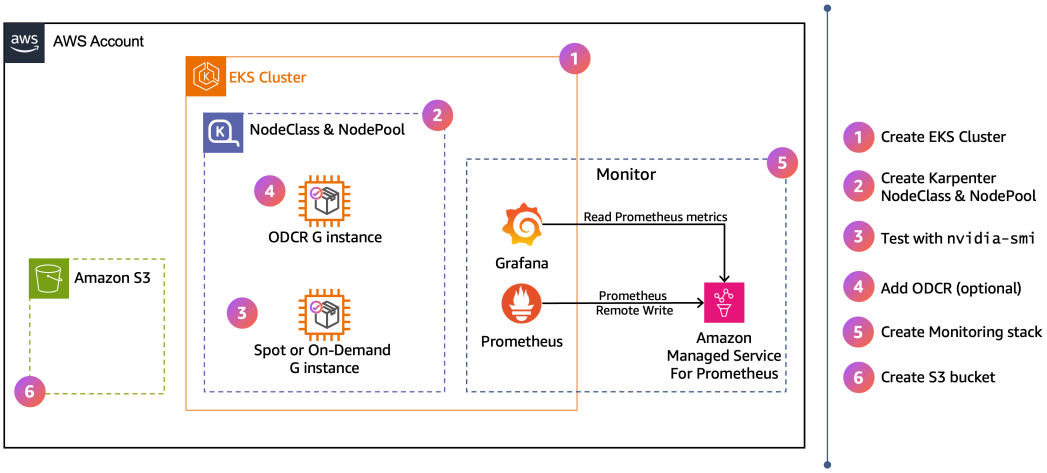
CLI を使用して AI/ML ワークロード用の Amazon EKS クラスターを設定する

> **Tip**
>
> 今後開催予定の Amazon EKS AI/ML ワークショップに登録してください。

このセクションでは、CLI コマンドを使用して Amazon EKS でトレーニングまたは推論ワークロードを実行するために必要なインフラストラクチャを作成する手順について説明します。ステップには、EKS クラスターの作成、EKS Auto Mode または Karpenter を使用した GPU 対応ノードの作成、Prometheus と Grafana を使用したモニタリングスタック、モデルの重み付け用の Amazon S3 ストレージが含まれます。

これらの機能が EKS クラスターで EC2 インスタンスをプロビジョニングおよび自動スケールする方法の詳細については、EKS Auto Mode と Karpenter のドキュメントを参照してください。

高レベルアーキテクチャとワークフロー



この図は、このセクションのセットアップの AWS 高レベルアーキテクチャを示しています。右側の番号付きステップは、以下のステップで設定を完了する順序を示しています。

前提条件

> **Important**
>
> EKS クラスター、GPU インスタンス、Application Load Balancer、Amazon Managed Service for Prometheus など、このチュートリアルで作成するリソースには料金が発生します。終了したリソースは、継続的な料金の発生を回避するために削除してください。

- `kubectl` >= 1.36。セットアップ手順については、「the section called “kubectl と eksctl のセットアップ”」を参照してください。
- AWS CLI >= 2.27。セットアップ手順については、「AWS CLI の最新バージョンのインストールまたは更新」を参照してください。
- Helm >= 3.14。セットアップの手順については、「Helm を使用して Amazon EKS にアプリケーションをデプロイする」を参照してください。
- `jq`。セットアップ手順については、「Download jq」を参照してください。
- `eksctl` >= 0.227.0。セットアップ手順については、`eksctl` ドキュメントの「インストール」を参照してください。

`eksctl` バージョンを確認します。

```
eksctl version
```

0.227.0 より前のバージョンを使用している場合は、eksctl インストールガイドに従って最新のリリースにアップグレードします。

環境変数を設定する

これらのステップ全体で、以下のクラスター名と AWS リージョンの一貫性を維持します。変更すると、後続のコマンドが間違った EKS クラスターをターゲットにする可能性があります。

```
export CLUSTER_NAME=ai-eks-docs
export AWS_REGION=us-east-2
```

利用可能なすべての AZ を使用すると、耐障害性が向上し、GPU キャパシティを取得する可能性が高くなります。

```
export AZS=$(aws ec2 describe-availability-zones \
  --region ${AWS_REGION} \
  --query "AvailabilityZones[?ZoneId!='use1-az3' && ZoneId!='usw1-az2' && ZoneId!
='cac1-az3'].ZoneName" \
  --output text | tr '\t' ',')
echo $AZS
```

> **Important**
>
> Amazon EKS はこれらのゾーンでのコントロールプレーン配置をサポートしていないため、アベイラビリティーゾーン use1-az3、usw1-az2、および cac1-az3 は除外されます。これらのゾーンのいずれかにサブネットを持つクラスターを作成すると、UnsupportedAvailabilityZoneException が発生します。

正常な出力:

```
us-east-2a,us-east-2b,us-east-2c
```

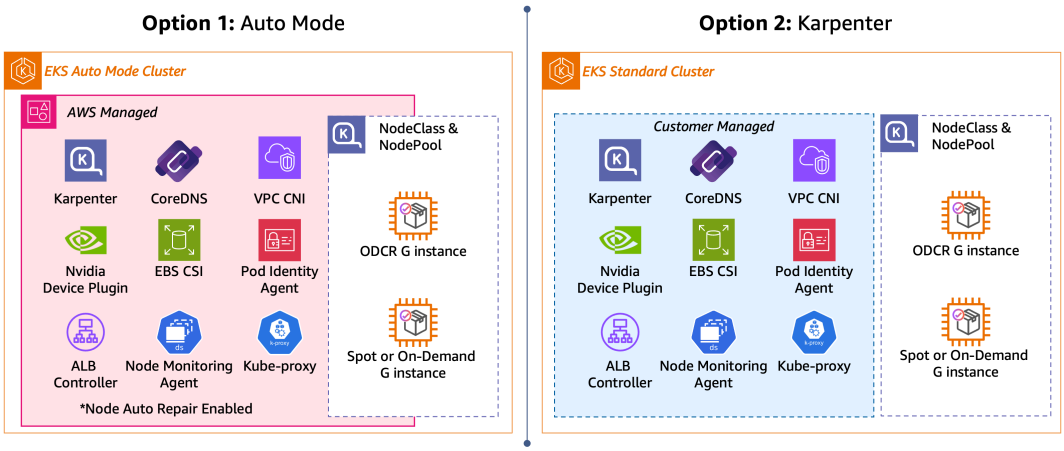
出力に表示される AZ はリージョンによって異なります。この例では、us-east-2 リージョンで利用可能な AZ を示しています。

クラスターと GPU NodePool を作成する

このセクションでは、EKS クラスターと GPU 対応ノードを作成するための 2 つのパスを以下の図に示します。ガイド全体で 1 つのオプションのみを選択してください。

- EKS Auto Mode — EKS Auto Mode には、コアネットワーク、ストレージ、ロードバランシングアドオンに加えて、EKS ノードモニタリングエージェント、自動ノード修復、高速コンテナプル用の SOCI スナップショッター、デフォルトの NodeClass の GPU 準備などのトレーニングおよび推論ワークロードの機能が含まれています。NVIDIA デバイスプラグインは、EKS Auto Mode が GPU 対応ノードに使用する Bottlerocket 高速 AMI に含まれています。
- セルフマネージド Karpenter — EKS Auto Mode のない EKS クラスターでは、トレーニングおよび推論ワークロードに必要なコンポーネントのインストールおよび設定を自分で行う必要があります。これには、ネットワーキングアドオン (VPC CNI、CoreDNS、kube-proxy)、Karpenter、EKS ノードモニタリングエージェント、NVIDIA デバイスプラグイン、高速コンテナプル用の SOCI スナップショットターが含まれます。

EKS クラスターオプション: EKS Auto Mode とセルフマネージド Karpenter



以下の各ステップで、パス (EKS Auto Mode、Karpenter) を選択し、一貫してそのパスに従います。選択したパスのステップを完了すると、GPU NodePool を備えた EKS クラスターが GPU ワークロードをスケジュールできるようになります。

ステップ 1: クラスターを作成する

まず、EKS クラスターを作成し、GPU ワークロードに必要なクラスターコンポーネントをインストールします。

EKS Auto Mode では、1 つの `eksctl create cluster --enable-auto-mode` コマンドで GPU ワークロードに対応する EKS クラスターをプロビジョニングします。

セルフマネージド Karpenter では、`eksctl create cluster` コマンドはコアネットワーキングアドオンをプロビジョニングします。その後、Karpenter 機能ゲートを介して自動ノード修復を有効に

し、EKS ノードモニタリングエージェントをインストールして、NVIDIA デバイスプラグインをインストールするための追加のステップが必要です。

EKS Auto Mode

EKS Auto Mode クラスターを作成する

```
eksctl create cluster \
  --name=$CLUSTER_NAME \
  --region=$AWS_REGION \
  --enable-auto-mode \
  --version=1.36 \
  --zones=$AZS
```

このコマンドの完了には数分かかります。完了すると、eksctl は新しくプロビジョニングされたクラスターで動作するように kubeconfig ファイルを自動的に更新します。クラスターが動作していることを確認します。

```
kubectl get pods --all-namespaces
```

正常な出力:

```
NAMESPACE     NAME                                READY   STATUS    RESTARTS   AGE
kube-system   metrics-server-55cf976ddd-cz2mw   1/1     Running   0          3m
kube-system   metrics-server-55cf976ddd-wrjvv   1/1     Running   0          3m
```

EKS Auto Mode では、VPC CNI、kube-proxy、および CoreDNS はマネージドコンポーネントとして実行され、kube-system ではポッドとして表示されません。

デフォルトの StorageClass を作成する

GPU ワークロードでは、多くの場合、モデルの重みをキャッシュするなど、永続ストレージが必要になります。これにより、Kubernetes はポッドを再起動するたびに再ダウンロードする必要がなくなります。EKS Auto Mode にはブロックストレージ機能が組み込まれているため、Amazon EBS CSI ドライバーをインストールする必要はありません。Auto Mode プロビジョナーを参照する StorageClass を作成します。

```
cat << EOF | kubectl apply -f -
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
```

```
  name: gp3
  annotations:
    storageclass.kubernetes.io/is-default-class: "true"
provisioner: ebs.csi.eks.amazonaws.com
volumeBindingMode: WaitForFirstConsumer
parameters:
  type: gp3
  encrypted: "true"
EOF
```

このコマンドは gp3 StorageClass を作成し、storageclass.kubernetes.io/is-default-class 注釈でクラスターのデフォルトとしてマークします。storageClassName を省略した PersistentVolumeClaim は、この StorageClass を使用します。volumeBindingMode: WaitForFirstConsumer 設定は、Kubernetes がポッドをスケジュールするまでボリュームの作成を遅らせるため、EBS ボリュームはポッドと同じアベイラビリティーゾーンに配置されます。

Self-managed Karpenter

Helm をパブリック ECR に認証する

eksctl は、Amazon Public ECR から Karpenter Helm チャートをプルします。Helm のインストールステップで 403 エラーが発生しないように、クラスターを作成する前に認証します。

```
aws ecr-public get-login-password --region us-east-1 \
  | helm registry login --username AWS --password-stdin public.ecr.aws
```

パブリック ECR は、us-east-1 でホストされているグローバルサービスです。EKS クラスターがどのリージョンにあるかに関係なく、ここで --region us-east-1 を使用します。

正常な出力: Login Succeeded

Karpenter で EKS クラスターを作成する

Karpenter バージョンは、後で使用するために環境変数に保存します。最新の Karpenter バージョンについては、GitHub の Karpenter リリースを参照してください。

```
export KARPENTER_VERSION=1.14.0
```

Exampleクラスター設定 YAML

```
cat << EOF > /tmp/cluster-karpenter.yaml
```

```
apiVersion: eksctl.io/v1alpha5
kind: ClusterConfig

metadata:
  name: ${CLUSTER_NAME}
  region: ${AWS_REGION}
  version: "1.36"
  tags:
    karpenter.sh/discovery: ${CLUSTER_NAME}

availabilityZones: [$(echo $AZS | sed 's/,/, /g')]

autoModeConfig:
  enabled: false

iam:
  withOIDC: true

karpenter:
  version: "${KARPENTER_VERSION}"
  withSpotInterruptionQueue: true

managedNodeGroups:
  - name: system
    instanceTypes: ["m6i.2xlarge", "m5.2xlarge", "m6a.2xlarge", "m5a.2xlarge",
 "m7i.2xlarge"]
    desiredCapacity: 2
    minSize: 2
    maxSize: 3
    labels:
      node-role: system
    tags:
      karpenter.sh/discovery: ${CLUSTER_NAME}

addons:
  - name: eks-pod-identity-agent
  - name: eks-node-monitoring-agent
  - name: aws-ebs-csi-driver
EOF
```

system マネージドノードグループには、単一のタイプではなく、同じサイズ (8 vCPU、32 GiB) の複数のインスタンスタイプが一覧表示されます。1 つのタイプがアベイラビリティーゾーンで一時的に使用できない場合、ノードグループの Auto Scaling グループは別のタイプにフォール

バックされるため、InsufficientInstanceCapacity 起動の失敗を回避できます。オンデマンドマネージドノードグループでは、ノードサイズが均一になるように、一覧表示されているすべてのインスタンスタイプに同じ vCPU とメモリがある必要があります。

設定ファイルを使用してクラスターを作成します:

```
eksctl create cluster -f /tmp/cluster-karpenter.yaml
```

このコマンドの実行には約 15 分かかります。アドオンと Karpenter コントローラーのホスティング専用のマネージドノードグループを持つ EKS クラスターを作成します。Karpenter は、スポット中断キューを有効にしてインストールされるため、スポット中断と再調整の推奨事項を処理できます。autoModeConfig.enabled: false 設定により、このクラスターが EKS Auto Mode を使用しないことが明示的になるため、このパスにインストールされた Karpenter コンポーネントがノード管理を担当します。

また、クラスターは EKS Pod Identity Agent、EKS ノードモニタリングエージェント、および Amazon EBS CSI ドライバーを EKS アドオンとしてインストールします。EKS Pod Identity はガイドの後半で使用します。EKS ノードモニタリングエージェントはすべてのノードで実行され、カーネルログを読み取り、AcceleratedHardwareReady、KernelReady、NetworkingReady などのノード条件を設定します。Karpenter 自動ノード修復は、これらのノード条件を使用して異常なノードを置き換えるタイミングを決定します。

クラスターが動作していることを確認します。

```
kubectl get pods --all-namespaces
```

予想される出力には、Karpenter、CoreDNS、kube-proxy、aws-node (VPC CNI)、EKS Pod Identity Agent、および EKS ノードモニタリングエージェントが含まれます。

```
NAMESPACE     NAME                                  READY   STATUS    RESTARTS   AGE
karpenter     karpenter-567547464c-s6vkx            1/1     Running   0          3m40s
karpenter     karpenter-567547464c-x7gmw            1/1     Running   0          3m40s
kube-system   aws-node-b6gf2                        2/2     Running   0          12m
kube-system   aws-node-lcphh                        2/2     Running   0          12m
kube-system   coredns-7d4dcbf4fb-ccvrr              1/1     Running   0          16m
kube-system   coredns-7d4dcbf4fb-qbhk2              1/1     Running   0          16m
kube-system   eks-node-monitoring-agent-h79vm       1/1     Running   0          9m45s
kube-system   eks-node-monitoring-agent-tf4dw       1/1     Running   0          9m45s
```

```
kube-system   eks-pod-identity-agent-5jbtc      1/1     Running   0          12m
kube-system   eks-pod-identity-agent-rwcrc      1/1     Running   0          12m
kube-system   kube-proxy-p4bmq                  1/1     Running   0          12m
kube-system   kube-proxy-v5nwr                  1/1     Running   0          12m
kube-system   metrics-server-5b966ff79c-hr58p   1/1     Running   0          9m22s
kube-system   metrics-server-5b966ff79c-szs2d   1/1     Running   0          9m22s
```

Karpenter 機能ゲートを有効にする

EKS Auto Mode は、デフォルトで自動ノード修復と静的キャパシティを有効にします。セルフマネージド Karpenter では、NodeRepair 機能ゲートと静的 NodePools の StaticCapacity ゲートを明示的に有効にする必要があります。これにより、保留中のポッドに応じてスケーリングするのではなく、spec.replicas を通じて固定ノード数が維持されます。Karpenter では、どちらのゲートもデフォルトでアルファとオフになっています。以下のコマンドは、Karpenter デプロイにパッチを適用して両方のゲートを有効にします。デプロイ環境を更新すると、Karpenter ポッドのロールアウトがトリガーされます。

```
kubectl set env deployment/karpenter -n karpenter \
  FEATURE_GATES=NodeRepair=true,StaticCapacity=true
```

正常な出力:

```
deployment.apps/karpenter env updated
```

Karpenter ポッドがロールアウトするのを待ちます。

```
kubectl rollout status deployment/karpenter -n karpenter
```

NVIDIA デバイスプラグインをインストールする

EKS 最適化 AL2023 AMI には NVIDIA デバイスプラグインは含まれていません (EKS Auto Mode で使用される Bottlerocket AMI とは異なります)。Helm 経由でインストールして、クラスターのポッドで GPU リソースを使用できるようにします。

```
helm repo add nvdp https://nvidia.github.io/k8s-device-plugin
helm repo update
```

```
cat << 'EOF' > /tmp/nvdp-values.yaml
mofedEnabled: false
```

```
nodeSelector:
  amiFamily: al2023
gfd:
  enabled: true
nfd:
  worker:
    tolerations:
      - operator: "Exists"
EOF
```

```
helm install nvidia-device-plugin nvdp/nvidia-device-plugin \
  --namespace kube-system \
  -f /tmp/nvdp-values.yaml
```

- `mofedEnabled: false`: AWS が使用しない Mellanox OFED (InfiniBand) のチェックを無効にします
- `nodeSelector.amiFamily: al2023`: DaemonSet を AL2023 ノードのみにスコープします (Bottlerocket には既にプラグインが組み込まれています)
- `gfd.enabled: true`: GPU Feature Discovery ラベルを有効にします (`nvidia.com/gpu.product`、`nvidia.com/gpu.memory` など)

NVIDIA デバイスプラグインがインストールされていることを確認します。一致するラベルを持つ GPU NodePool がプロビジョニングされるまで、デバイスプラグインポッドはゼロであることが想定されています。

```
kubectl get daemonset nvidia-device-plugin -n kube-system
```

正常な出力:

```
NAME                   DESIRED   CURRENT   READY   UP-TO-DATE   AVAILABLE   NODE
 SELECTOR      AGE
nvidia-device-plugin   0         0         0       0            0
 amiFamily=al2023   2m5s
```

次に、汎用 NodePool を作成します。`system` マネージドノードグループは、アドオンとコントローラーを実行します。汎用 Karpenter NodePoolを追加して、非 GPU ワークロードが `system` ノードにランディングすることなくオンデマンドでスケーリングできるようにします。GPU NodePool はステップ 2 で後ほど作成されます。

汎用 EC2NodeClass および NodePool YAML

```
cat << EOF | kubectl apply -f -
apiVersion: karpenter.k8s.aws/v1
kind: EC2NodeClass
metadata:
  name: general-purpose
spec:
  role: "eksctl-KarpenterNodeRole-${CLUSTER_NAME}"
  amiSelectorTerms:
    - alias: al2023@latest
  subnetSelectorTerms:
    - tags:
        karpenter.sh/discovery: ${CLUSTER_NAME}
  securityGroupSelectorTerms:
    - tags:
        karpenter.sh/discovery: ${CLUSTER_NAME}
  tags:
    karpenter.sh/discovery: ${CLUSTER_NAME}
---
apiVersion: karpenter.sh/v1
kind: NodePool
metadata:
  name: general-purpose
spec:
  template:
    spec:
      nodeClassRef:
        group: karpenter.k8s.aws
        kind: EC2NodeClass
        name: general-purpose
      requirements:
        - key: karpenter.sh/capacity-type
          operator: In
          values: ["on-demand"]
        - key: karpenter.k8s.aws/instance-category
          operator: In
          values: ["c", "m", "r"]
        - key: karpenter.k8s.aws/instance-generation
          operator: Gt
          values: ["4"]
        - key: kubernetes.io/arch
          operator: In
          values: ["amd64"]
```

```
  limits:
    cpu: 1000
    memory: 1000Gi
EOF
```

この NodePool にはテイントがないため、GPU テイントを許容しない Pod はこの NodePool にスケジュールできます。

デフォルトの StorageClass を作成する

GPU ワークロードでは、多くの場合、モデルの重みをキャッシュするなど、永続ストレージが必要になります。これにより、Kubernetes はポッドを再起動するたびに再ダウンロードする必要がなくなります。Amazon EBS CSI ドライバーは、このステップの前半でクラスター設定の addons: ブロックによって既にインストールされているため、StorageClass を作成するだけで済みます。

```
cat << EOF | kubectl apply -f -
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: gp3
  annotations:
    storageclass.kubernetes.io/is-default-class: "true"
provisioner: ebs.csi.aws.com
volumeBindingMode: WaitForFirstConsumer
parameters:
  type: gp3
  encrypted: "true"
EOF
```

このコマンドは gp3 StorageClass を作成し、storageclass.kubernetes.io/is-default-class 注釈でクラスターのデフォルトとしてマークします。storageClassName を省略した PersistentVolumeClaim は、この StorageClass を使用します。volumeBindingMode: WaitForFirstConsumer 設定は、Kubernetes がポッドをスケジュールするまでボリュームの作成を遅らせるため、EBS ボリュームはポッドと同じアベイラビリティーゾーンに配置されます。

> **Warning**
>
> EKS Auto Mode パスとセルフマネージド Karpenter パスの両方において、ノードの自動修復は NodePools によってプロビジョニングされたノードでも同じように動作します。EKS Auto Mode と Karpenter の自動ノード修復は、PodDisruptionBudgets、`karpenter.sh/do-not-disrupt` 注釈、および `terminationGracePeriod` をバイパスする強力な中断方法です。自動ノード修復は、`AcceleratedHardwareReady` 条件が `False` に設定されたノードについては置き換えまで 10 分待機し、他の修復条件については 30 分待機します。

ロードバランシングを設定する

このガイドの後のステップでは、AWS Application Load Balancer (ALB) を介して Grafana (および推論ウォークスルーではモデルエンドポイント) を公開します。ロードバランシングの前提条件を今すぐセットアップして、これらのセクションがそれ以上のセットアップなしで `Ingress` を作成できるようにします。

ALB は、AWS リソースタグに基づいて VPC サブネットに配置されます。インターネット向けロードバランサーに使用されるパブリックサブネットには `kubernetes.io/role/elb = 1` のタグを付け、内部ロードバランサーに使用されるプライベートサブネットには `kubernetes.io/role/internal-elb = 1` のタグを付ける必要があります。このクラスターを `eksctl` で作成したため、これらのタグは `eksctl` が作成したサブネットに既に存在します。独自の VPC またはインポートされたサブネットを持ち込む場合は、手動でそれらにタグ付けします。詳細については、「the section called “サブネットにタグを付ける”」を参照してください。

ロードバランサー IngressClass を作成する

残りのセットアップはパスによって異なります。EKS Auto Mode には組み込みの ALB コントローラーが含まれているため、`IngressClass` を自分で作成します。セルフマネージド Karpenter では、Helm で AWS Load Balancer Controller をインストールする必要があります。Helm チャートによって `IngressClass` が自動的に作成されます。

EKS Auto Mode

EKS Auto Mode には組み込みの ALB コントローラーが含まれているため、追加のコンポーネントをインストールする必要はありません。組み込みコントローラーを指す `alb` という名前の `IngressClass` を作成します。後で作成する各 `Ingress` は、このクラスを名前で参照し、注釈を使用して独自のスキームやその他のオプションを設定します。

```
cat << EOF | kubectl apply -f -
apiVersion: networking.k8s.io/v1
kind: IngressClass
metadata:
  name: alb
spec:
  controller: eks.amazonaws.com/alb
EOF
```

Self-managed Karpenter

AWS Load Balancer Controller は EKS マネージドアドオンではないため、EBS CSI ドライバーと同じように eksctl クラスター設定 addons: ブロックに入ることはできません。Helm を使用してそれをインストールします。

IAM ポリシーをダウンロードします。

コントローラーには、ALB を作成および管理するための IAM アクセス許可が必要です。公式 IAM ポリシー JSON をダウンロードします。

```
curl -o iam_policy.json https://raw.githubusercontent.com/kubernetes-sigs/aws-load-balancer-controller/main/docs/install/iam_policy.json
```

IAM ポリシーを作成します。

```
aws iam create-policy \
  --policy-name AWSLoadBalancerControllerIAMPolicy \
  --policy-document file://iam_policy.json
```

Pod Identity の関連付けを作成します。

クラスターはステップ 1 で eks-pod-identity-agent アドオンを既にインストールしているため、EKS Pod Identity を使用できます。IAM ポリシーをコントローラーのサービスアカウントにバインドする Pod Identity 関連付けを作成します。

```
eksctl create podidentityassociation \
  --cluster ${CLUSTER_NAME} \
  --region ${AWS_REGION} \
  --namespace kube-system \
  --service-account-name aws-load-balancer-controller \
```

```
  --permission-policy-arns arn:aws:iam::$(aws sts get-caller-identity --query
 Account --output text):policy/AWSLoadBalancerControllerIAMPolicy
```

Helm を使用してインストールする

EKS チャート Helm リポジトリを追加します。

```
helm repo add eks https://aws.github.io/eks-charts
helm repo update
```

クラスターの VPC ID を検索します。Helm 値は、コントローラーがサブネットを検出できるようにするために必要です。

```
VPC_ID=$(aws eks describe-cluster \
  --name ${CLUSTER_NAME} \
  --region ${AWS_REGION} \
  --query 'cluster.resourcesVpcConfig.vpcId' \
  --output text)
```

Load Balancer Controller をインストールします。nodeSelector と許容値は、GPU または汎用 Karpenter 起動ノードとキャパシティを競合するのではなく、クラスタークリティカルなインフラストラクチャとともに system マネージドノードグループにコントローラーを配置します。

```
helm install aws-load-balancer-controller eks/aws-load-balancer-controller \
  --namespace kube-system \
  --version 3.4.2 \
  --set clusterName=${CLUSTER_NAME} \
  --set region=${AWS_REGION} \
  --set vpcId=${VPC_ID} \
  --set serviceAccount.create=true \
  --set serviceAccount.name=aws-load-balancer-controller \
  --set "nodeSelector.node-role=system" \
  --set "tolerations[0].key=CriticalAddonsOnly" \
  --set "tolerations[0].operator=Exists"
```

コントローラーが実行されていることを確認します。使用可能な 2 つのレプリカをレポートする必要があります。

```
kubectl get deployment aws-load-balancer-controller -n kube-system
```

IngressClass を検証する

Helm チャートは、alb という名前の IngressClass をコントローラーのインストールの一部として作成し、セルフマネージドコントローラー ingress.k8s.aws/alb を指しています。手動で作成する必要はありません。それが存在することを確認します:

```
kubectl get ingressclass alb
```

正常な出力:

```
NAME   CONTROLLER            PARAMETERS   AGE
alb    ingress.k8s.aws/alb   <none>       22s
```

両方のパスに alb IngressClass が追加されました。Grafana および推論セクションは ingressClassName: alb を参照し、Ingress ごとの注釈を通じて ALB スキーム、送信元 IP 許可リスト、およびその他の動作を設定します。

ステップ 2: 動的 GPU NodePool を作成する

On-Demand をフォールバックとした Spot キャパシティを使用して、世代が 4 より大きい G ファミリー GPU インスタンスを動的にプロビジョニングする NodePool を定義します。EKS Auto Mode パスと Karpenter パスはどちらも同じ NodePool API を使用しますが、唯一の違いはそれが指す NodeClass です。EKS Auto Mode では、バンドルされた default NodeClass は既に適切な AMI を選択し、SOCI 並列プルを設定しているため、NodePool は作成する唯一のオブジェクトです。セルフマネージド Karpenter では、AMI を固定して SOCI を調整するカスタム EC2NodeClass も必要です。これら 2 つのパスでは異なる AMI が使用されるため、各パスに記載されているように、インスタンスの制約も異なってきます。

EKS Auto Mode

> **Important**
>
> このパスでは EKS 最適化 Bottlerocket NVIDIA AMI が使用されますが、これには g7 インスタンスファミリーをサポートするドライバーバージョンが含まれていません (g7 には NVIDIA ドライバーバージョン 595 以降が必要です)。そのため、このパスの NodePool からは g7 ファミリーが除外されています。その他の第 7 世代インスタンス (g7e など) は NVIDIA ドライバー 580 で動作し、サポートされています。g7 のサポートが必要な場合は、EKS 最適化 AL2023 NVIDIA AMI を使用するセルフマネージド Karpenter パスを使用してください。詳細については、「EKS 最適化 NVIDIA AMI」を参照してください。

EKS Auto Mode では、バンドルされた `default` NodeClass は GPU インスタンスの Bottlerocket AMI を自動的に選択します。これには、プリインストールされた NVIDIA ドライバー、NVIDIA デバイスプラグイン、SOCI 並列プルが含まれます。`default` NodeClass を参照する NodePool を適用するだけです。

```
cat << 'EOF' | kubectl apply -f -
apiVersion: karpenter.sh/v1
kind: NodePool
metadata:
  name: gpu-inf
spec:
  template:
    metadata:
      labels:
        guide: ai-eks-docs
    spec:
      nodeClassRef:
        group: eks.amazonaws.com
        kind: NodeClass
        name: default
      taints:
        - key: nvidia.com/gpu
          effect: NoSchedule
      requirements:
        - key: karpenter.sh/capacity-type
          operator: In
          values: ["spot", "on-demand"]
        - key: eks.amazonaws.com/instance-category
          operator: In
          values: ["g"]
        - key: eks.amazonaws.com/instance-generation
          operator: Gt
          values: ["4"]
        - key: eks.amazonaws.com/instance-family
          operator: NotIn
          values: ["g7"]
        - key: kubernetes.io/arch
          operator: In
          values: ["amd64"]
  limits:
    cpu: 1000
    memory: 5000Gi
```

```
EOF
```

この NodePool は、g7 ファミリーを除く、世代が 4 より大きい G ファミリー GPU インスタンス (例: G5、G6e、G7e) をプロビジョニングします。EKS Auto Mode は、これらの制約内でインスタンスタイプを選択します。

Self-managed Karpenter

セルフマネージド Karpenter には、デフォルトの NodeClass は含まれません。まず、EKS 最適化 NVIDIA AL2023 AMI エイリアスを固定し、`FastImagePull` 機能ゲートを介して SOCI を有効にし、コンテナ化されたイメージキャッシュをローカル NVMe に移動するように `instanceStorePolicy: RAID0` を設定する `EC2NodeClass` を作成します。次に、それを参照する NodePool を作成します。

EC2NodeClass を作成する

Example

```
cat << EOF | kubectl apply -f -
apiVersion: karpenter.k8s.aws/v1
kind: EC2NodeClass
metadata:
  name: gpu-inf
  labels:
    guide: ai-eks-docs
spec:
  role: "eksctl-KarpenterNodeRole-${CLUSTER_NAME}"
  amiSelectorTerms:
    - alias: al2023@latest
  subnetSelectorTerms:
    - tags:
        karpenter.sh/discovery: ${CLUSTER_NAME}
  securityGroupSelectorTerms:
    - tags:
        karpenter.sh/discovery: ${CLUSTER_NAME}
  tags:
    karpenter.sh/discovery: ${CLUSTER_NAME}
  instanceStorePolicy: RAID0
  userData: |
    apiVersion: node.eks.aws/v1alpha1
    kind: NodeConfig
    spec:
      featureGates:
```

```
        FastImagePull: true
EOF
```

`instanceStorePolicy: RAID0` はローカル NVMe ディスクを RAID-0 アレイにアセンブルします。`al2023@latest` AMI エイリアスは EKS 最適化 AL2023 AMI に解決されます。Karpenter が GPU インスタンスタイプを起動すると、プリインストールされている NVIDIA ドライバーを含む AL2023_x86_64_NVIDIA 高速バリアントが自動的に選択されます。

`FastImagePull` 機能ゲートは、イメージレイヤーを同時にダウンロードおよび解凍する SOCI スナップショッター並列プルモードを有効にします。これは、G、P、および Trn インスタンスファミリーの EKS Auto Mode の動作と一致します。

その他の SOCI チューニングオプション (イメージあたりの同時ダウンロード数、チャンクサイズなど) については、Karpenter SOCI 設計図を参照してください。

EC2NodeClass を検証します。

```
kubectl get ec2nodeclass gpu-inf
```

正常な出力: `READY True`。`False` の場合は、`kubectl describe ec2nodeclass gpu-inf` を実行し、サブネットタグまたはセキュリティグループタグがないかどうか条件を確認します。

GPU NodePool を作成する

Example

```
cat << EOF | kubectl apply -f -
apiVersion: karpenter.sh/v1
kind: NodePool
metadata:
  name: gpu-inf
spec:
  template:
    metadata:
      labels:
        guide: ai-eks-docs
        amiFamily: al2023
    spec:
      nodeClassRef:
        group: karpenter.k8s.aws
        kind: EC2NodeClass
        name: gpu-inf
```

```
      taints:
        - key: nvidia.com/gpu
          effect: NoSchedule
      requirements:
        - key: karpenter.sh/capacity-type
          operator: In
          values: ["spot", "on-demand"]
        - key: karpenter.k8s.aws/instance-category
          operator: In
          values: ["g"]
        - key: karpenter.k8s.aws/instance-generation
          operator: Gt
          values: ["4"]
        - key: kubernetes.io/arch
          operator: In
          values: ["amd64"]
  limits:
    cpu: 1000
    memory: 5000Gi
EOF
```

ノードテンプレートの `amiFamily: al2023` ラベルは、NVIDIA デバイスプラグイン DaemonSet がこれらのノードを選択するために使用するものです。Karpenter は、これらの制約内でインスタンスタイプを選択します。`nvidia.com/gpu:NoSchedule` テイントにより、GPU 対応ポッドのみがこれらのノードでスケジュールされます。

EKS Auto Mode パスとは異なり、この NodePool ではインスタンスの世代に制限が設けられていません。これは、EKS 最適化 AL2023 NVIDIA AMI に g7 インスタンスをサポートするドライバーバージョンが含まれているためです。EKS AL2023 AMI は現在、g7 インスタンス (NVIDIA ドライバー 595 を使用) を除くすべての EC2 インスタンスで NVIDIA ドライバー 580 を使用しています。詳細については、「EKS 最適化 NVIDIA AMI」を参照してください。

NodePool が作成されたことを確認します。

```
kubectl get nodepool gpu-inf
```

正常な出力:

```
NAME      NODECLASS   NODES   READY   AGE
gpu-inf   default     0       True    8s
```

セルフマネージド Karpenter パスでは、NODECLASS 列に `default` ではなく `gpu-inf` が表示されます。

ステップ 3: サンプルポッドでテストする

`nvidia-smi` ポッドを使用して GPU NodePool のセットアップをテストします。

```
cat << EOF | kubectl apply -f -
apiVersion: v1
kind: Pod
metadata:
  name: nvidia-smi
  labels:
    guide: ai-eks-docs
spec:
  tolerations:
    - key: "nvidia.com/gpu"
      operator: "Exists"
      effect: "NoSchedule"
  containers:
    - name: nvidia-smi
      image: public.ecr.aws/amazonlinux/amazonlinux:2023-minimal
      command: ["nvidia-smi"]
      resources:
        limits:
          nvidia.com/gpu: 1
  restartPolicy: OnFailure
EOF
```

ポッドがスケジュールされ、正常に完了していることを確認します。

```
kubectl get pods
```

正常な出力:

```
NAME         READY   STATUS      RESTARTS   AGE
nvidia-smi   0/1     Completed   0          67s
```

STATUS: Completed は、nvidia-smi コマンドが実行され、終了したことを意味します。ポッドログをチェックして、ノードによって検出された GPU を確認します。

```
kubectl logs nvidia-smi
```

正常な出力:

```
+-----------------------------------------------------------------------------------------
+
| NVIDIA-SMI 580.159.03             Driver Version: 580.159.03     CUDA Version: 13.0
   |
+-----------------------------------------+------------------------
+----------------------+
| GPU  Name                 Persistence-M | Bus-Id          Disp.A | Volatile Uncorr.
 ECC |
| Fan  Temp   Perf          Pwr:Usage/Cap |           Memory-Usage | GPU-Util  Compute
 M. |
|                                         |                        |               MIG
 M. |
|=========================================+========================
+======================|
|   0  NVIDIA L4                      On  |   00000000:35:00.0 Off |
 0 |
| N/A   42C    P8             16W /   72W |       0MiB /  23034MiB |      0%
 Default |
|                                         |                        |
 N/A |
+-----------------------------------------+------------------------
+----------------------+
```

出力には、GPU モデル、ドライバーバージョン、CUDA バージョン、利用可能なメモリが表示されます。この例では、Karpenter は 24 GB のメモリを持つ NVIDIA L4 GPU を持つ G6 インスタンスをプロビジョニングしました。42C は現在の GPU 温度で、P8 は低電力アイドル状態です。これは、ワークロードが GPU を使用していないため、予想される状態です。16W/72W は現在の電力消費量と最大電力容量を示し、0MiB/23034MiB は現在の GPU メモリ使用量と利用可能な合計を示します。ポッドは nvidia-smi を実行して終了しただけなので、GPU を使用するワークロードはありません。そのため、メモリは 0 で、電力はアイドル状態です。NVIDIA GPU ドライバーバージョン (580.159.03) は Bottlerocket AMI から、CUDA バージョン (13.0) はコンテナイメージから取得されます。GPU モデルとメモリは、Karpenter が選択するインスタンスタイプによって異なります。G5 インスタンスには NVIDIA A10G GPU (24 GB)、G6 インスタンスには NVIDIA L4 GPU (24 GB)、G6e インスタンスには NVIDIA L40S GPU (48 GB) があります。

Karpenter と Kubernetes スケジューラがどのように連携してノードをプロビジョニングし、ポッドを配置したかを理解するには、ポッドのライフサイクルイベントを確認します。

```
kubectl describe po nvidia-smi
```

正常な出力:

```
Events:
  Type     Reason                  Age   From                  Message
  ----     ------                  ----  ----                  -------
  Warning  FailedScheduling        60s   default-scheduler     0/2 nodes are
 available: 2 node(s) had untolerated taint(s). no new claims to deallocate,
 preemption: 0/2 nodes are available: 2 Preemption is not helpful for scheduling.
  Normal   Nominated               59s   eks-auto-mode/compute  Pod should schedule on:
 nodeclaim/gpu-inf-vxcnj
  Normal   Scheduled               24s   default-scheduler     Successfully assigned
 default/nvidia-smi to i-0fb17a09bc4203164
  Warning  FailedCreatePodSandBox  21s   kubelet               Failed to create
 pod sandbox: rpc error: code = Unknown desc = failed to setup network for sandbox
 "7f85e25b220c8fb245187758dbbbc8efb3d40f3e49e13054404880daf4c3b2f0": plugin type="aws-
cni" name="aws-cni" failed (add): add cmd: failed to setup network policy
  Normal   Pulling                  7s   kubelet               spec.containers{nvidia-
smi}: Pulling image "public.ecr.aws/amazonlinux/amazonlinux:2023-minimal"
  Normal   Pulled                   5s   kubelet               spec.containers{nvidia-
smi}: Successfully pulled image "public.ecr.aws/amazonlinux/amazonlinux:2023-minimal"
 in 1.237s (1.237s including waiting). Image size: 37442701 bytes.
  Normal   Created                  5s   kubelet               spec.containers{nvidia-
smi}: Container created
  Normal   Started                  5s   kubelet               spec.containers{nvidia-
smi}: Container started
```

これらのイベントは、ポッドスケジューリングシーケンスを示しています。まず、GPU ノードが存在しないため、ポッドはスケジュールに失敗します (FailedScheduling)。Karpenter が新しい NodeClaim をノミネートします (Nominated)。ノードが準備完了になるとスケジューラがポッドを割り当てます (Scheduled)。その後、コンテナイメージがプルされて起動します。EKS Auto Mode には、SOCI (Seekable OCI) 並列プルが最初からインストールされ、G、P、および Trn インスタンスでそのまま使用できるように設定されています。SOCI 並列プルのため、コンテナイメージは 2 秒未満 (1.237 秒) で ECR からプルされたことに注意してください。

NodeClaim は、Karpenter が特定のノードをプロビジョニングするために作成するリクエストです。インスタンスタイプ、キャパシティタイプ、AZ、ノードの準備が整っているかどうかが表示されます。

```
kubectl get nodeclaims
```

予想される NodeClaim 出力:

```
NAME            TYPE         CAPACITY   ZONE         NODE                  READY   AGE
gpu-inf-vxcnj   g6.4xlarge   spot       us-east-2c   i-0fb17a09bc4203164   True    51s
```

インスタンスタイプとアベイラビリティーゾーンは異なります。NodePool の世代制約を満たす G ファミリーインスタンスであれば、すべて対象となります。両方のパスで世代が 4 より大きいことが条件ですが、EKS Auto Mode パスでは g7 ファミリーは除外されます。

`kubectl describe pod nvidia-smi` における `FailedCreatePodSandBox` 警告は一時的なものであり、想定された動作です。VPC CNI はノードの結合後に非同期的に初期化し、kubelet は自動的に再試行します。ポッドが `ContainerCreating` の状態のままである場合は、`kubectl describe node <node-name>` でノードイベントを確認します。

> **Tip**
>
> ノードが表示されない場合は、容量不足エラーを確認します。
>
> ```
> kubectl get events | grep InsufficientCapacityError
> ```
>
> Karpenter は、使用できないサービスを 3 分間キャッシュします。NodePool で許可されたインスタンスタイプと AZ を拡張すると、ランディングキャパシティの可能性が高まります。

> **Note**
>
> Karpenter によって起動されたスポットインスタンスは EC2 スポットリクエストコンソールに表示されません。Karpenter は `type: instant` で EC2 `CreateFleet` API を使用します。インスタンスは、`spot` ライフサイクルとともに EC2 インスタンスコンソールに表示されます。

ステップ 4: NodePool にリザーブドキャパシティを追加する (オプション)

スポット/オンデマンドフォールバックで最初にリザーブドキャパシティを使用するには、オンデマンドキャパシティ予約 (ODCR) を作成して NodeClass にアタッチし、ステップ 2 の動的 NodePool を更新して `reserved` キャパシティも許可します。予約 API コールは両方のパスで同じです。EKS Auto Mode とセルフマネージド Karpenter は異なる NodeClass タイプを使用するため、NodeClass アタッチメントは異なります。

> **Warning**
>
> 以下のコマンドを実行すると、`aws ec2 cancel-capacity-reservation --capacity-reservation-id <id>` でキャンセルするまで、リザーブドインスタンスタイプの料金が発生します。

キャパシティ予約を作成する:

```
CR_AZ="us-east-2a"
INSTANCE_TYPE="g6e.4xlarge"

aws ec2 create-capacity-reservation \
  --instance-type $INSTANCE_TYPE \
  --instance-platform Linux/UNIX \
  --availability-zone "$CR_AZ" \
  --instance-count 1 \
  --instance-match-criteria open \
  --end-date-type unlimited
```

`InsufficientInstanceCapacity` エラーが発生した場合は、`CR_AZ` を別の AZ に変更して再試行します。

キャパシティ予約 ID を検索し、以下のステップでシェル変数に保存します。

```
CAPACITY_RESERVATION_ID=$(aws ec2 describe-capacity-reservations \
  --filters "Name=state,Values=active" "Name=instance-type,Values=${INSTANCE_TYPE}" \
  --query 'CapacityReservations[0].CapacityReservationId' \
  --output text \
  --region ${AWS_REGION})
echo "Capacity reservation ID: ${CAPACITY_RESERVATION_ID}"
```

次に、パスに NodeClass と NodePool の変更を適用します。

EKS Auto Mode

EKS Auto Mode では、バンドルされた `default` NodeClass は読み取り専用であるため、予約を参照するカスタム NodeClass を作成し、NodeClass を指すように NodePool を更新して、`capacity-type` リストに `reserved` キャパシティを追加します。

Exampleカスタム NodeClass YAML

```
NODE_ROLE=$(kubectl get nodeclass default -o jsonpath='{.spec.role}')

cat << EOF | kubectl apply -f -
apiVersion: eks.amazonaws.com/v1
kind: NodeClass
metadata:
  name: gpu-inf
  labels:
    guide: ai-eks-docs
spec:
  role: "$NODE_ROLE"
  subnetSelectorTerms:
    - tags:
        alpha.eksctl.io/cluster-name: "$CLUSTER_NAME"
        kubernetes.io/role/internal-elb: "1"
  securityGroupSelectorTerms:
    - tags:
        aws:eks:cluster-name: "$CLUSTER_NAME"
  capacityReservationSelectorTerms:
    - id: "$CAPACITY_RESERVATION_ID"
EOF
```

`kubernetes.io/role/internal-elb: "1"` タグにより、ノードはプライベートサブネットでのみ起動されます。

NodePool を更新して ODCR-backed NodeClass を使用し、キャパシティタイプとして `reserved` を含めます。

Example更新された NodePool YAML

```
cat << EOF | kubectl apply -f -
apiVersion: karpenter.sh/v1
kind: NodePool
metadata:
  name: gpu-inf
```

```
spec:
  template:
    metadata:
      labels:
        guide: ai-eks-docs
    spec:
      nodeClassRef:
        group: eks.amazonaws.com
        kind: NodeClass
        name: gpu-inf
      taints:
        - key: nvidia.com/gpu
          effect: NoSchedule
      requirements:
        - key: karpenter.sh/capacity-type
          operator: In
          values: ["spot", "on-demand", "reserved"]
        - key: eks.amazonaws.com/instance-category
          operator: In
          values: ["g"]
        - key: eks.amazonaws.com/instance-generation
          operator: Gt
          values: ["4"]
        - key: eks.amazonaws.com/instance-family
          operator: NotIn
          values: ["g7"]
        - key: kubernetes.io/arch
          operator: In
          values: ["amd64"]
  limits:
    cpu: 1000
    memory: 5000Gi
EOF
```

Self-managed Karpenter

セルフマネージド Karpenter の場合は、ステップ 2 で作成した `EC2NodeClass` に `capacityReservationSelectorTerms` を追加して再適用します。フィールド名と構造は、もう一方のタブに表示されている EKS Auto Mode `NodeClass` と一致します。

Example更新された EC2NodeClass YAML

```
cat << EOF | kubectl apply -f -
apiVersion: karpenter.k8s.aws/v1
```

```
kind: EC2NodeClass
metadata:
  name: gpu-inf
  labels:
    guide: ai-eks-docs
spec:
  role: "eksctl-KarpenterNodeRole-${CLUSTER_NAME}"
  amiSelectorTerms:
    - alias: al2023@latest
  subnetSelectorTerms:
    - tags:
        karpenter.sh/discovery: ${CLUSTER_NAME}
  securityGroupSelectorTerms:
    - tags:
        karpenter.sh/discovery: ${CLUSTER_NAME}
  tags:
    karpenter.sh/discovery: ${CLUSTER_NAME}
  instanceStorePolicy: RAID0
  capacityReservationSelectorTerms:
    - id: "$CAPACITY_RESERVATION_ID"
  userData: |
    apiVersion: node.eks.aws/v1alpha1
    kind: NodeConfig
    spec:
      featureGates:
        FastImagePull: true
EOF
```

ステップ 2 からの変更点は、新しい capacityReservationSelectorTerms フィールドのみです。他のすべてのフィールドは同じままです。

NodePool を更新して、キャパシティタイプとして reserved を含めます。

Example更新された NodePool YAML

```
cat << EOF | kubectl apply -f -
apiVersion: karpenter.sh/v1
kind: NodePool
metadata:
  name: gpu-inf
spec:
  template:
    metadata:
```

```
      labels:
        guide: ai-eks-docs
        amiFamily: al2023
    spec:
      nodeClassRef:
        group: karpenter.k8s.aws
        kind: EC2NodeClass
        name: gpu-inf
      taints:
        - key: nvidia.com/gpu
          effect: NoSchedule
      requirements:
        - key: karpenter.sh/capacity-type
          operator: In
          values: ["spot", "on-demand", "reserved"]
        - key: karpenter.k8s.aws/instance-category
          operator: In
          values: ["g"]
        - key: karpenter.k8s.aws/instance-generation
          operator: Gt
          values: ["4"]
        - key: kubernetes.io/arch
          operator: In
          values: ["amd64"]
  limits:
    cpu: 1000
    memory: 5000Gi
EOF
```

Karpenter は、reserved を最もコスト効率の高いオプションとして扱い、最初に起動します。予約がいっぱいになると、スポットまたはオンデマンドにフォールバックします。

予約された優先度とスポットフォールバックを検証する

変更を適用したら、Karpenter がリザーブドキャパシティを優先し、スポットまたはオンデマンドにフォールバックすることを確認します。ポッドごとに 1 GPU をリクエストする 2 レプリカデプロイをデプロイします。ODCR は 1 インスタンス用であるため、最初のポッドは Karpenter をトリガーしてリザーブドノードを起動します。2 番目のポッドはリザーブドノードに収まらず、Karpenter がスポットキャパシティまたはオンデマンドキャパシティから別のノードを起動するようにトリガーします。

```
cat << 'EOF' | kubectl apply -f -
```

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: gpu-overflow-test
  labels:
    guide: ai-eks-docs
spec:
  replicas: 2
  selector:
    matchLabels:
      app: gpu-overflow-test
  template:
    metadata:
      labels:
        app: gpu-overflow-test
        guide: ai-eks-docs
    spec:
      tolerations:
        - key: nvidia.com/gpu
          operator: Exists
          effect: NoSchedule
      containers:
        - name: nvidia-smi
          image: public.ecr.aws/amazonlinux/amazonlinux:2023-minimal
          command: ["sh", "-c", "nvidia-smi && sleep infinity"]
          resources:
            limits:
              nvidia.com/gpu: 1
EOF
```

実行して終了したステップ 3 の nvidia-smi テストポッドとは異なり、このデプロイではポッドの実行 (sleep infinity) が維持されるため、ポッドは GPU を保持し、ノードを解放しません。

異なるノードでスケジュールされたポッドを確認します。

```
kubectl get pods -l app=gpu-overflow-test -o wide
```

正常な出力:

```
NAME                                READY   STATUS    RESTARTS   AGE     IP
     NODE                   NOMINATED NODE   READINESS GATES
gpu-overflow-test-59b97944fb-lq56c   1/1     Running   0          2m42s
 192.168.186.240   i-057692590480155da   <none>           <none>
```

```
gpu-overflow-test-59b97944fb-z4zcx   1/1     Running   0          2m42s
 192.168.130.64    i-0521ecd1849fa0578   <none>           <none>
```

2 つのポッドがそれぞれ異なるノードで実行されています。

NodeClaims でキャパシティタイプを確認します。

```
kubectl get nodeclaims
```

正常な出力:

```
NAME            TYPE          CAPACITY    ZONE         NODE                   READY
 AGE
gpu-inf-shg5w   g6e.xlarge    reserved    us-east-2a   i-0ea91fdeef65b8cb6   True
 2m2s
gpu-inf-ssnqf   g6e.2xlarge   spot        us-east-2b   i-00ccf7ce65cf3f6ca   True
 112s
```

予約ノードが最初に起動され、その後に予約がいっぱいになるとスポットノードまたはオンデマンドノードが起動されます。

テストデプロイをクリーンアップする:

```
kubectl delete deployment gpu-overflow-test
```

モニタリング

クラスター、ノード、GPU メトリクスを収集するモニタリングスタックを Amazon Managed Service for Prometheus (AMP) にインストールし、Grafana で視覚化します。kube-prometheus-stack Helm チャートは、Prometheus をデプロイしてメトリクスを AMP にスクレイプおよびリモート書き込みし、ダッシュボード用のセルフマネージド Grafana もデプロイします。NVIDIA DCGM Exporter は GPU 固有のメトリクス (使用率、メモリ、温度、電力、NVLink、テンソルアクティビティ) を追加します。

Prometheus、Grafana、およびオペレーターは、GPU ノードに `nvidia.com/gpu:NoSchedule` テイントがあるため、デフォルトで非 GPU ノードに配置されます。Node-exporter と DCGM Exporter はどちらも GPU ノードで実行されるため、ホストメトリクスと GPU メトリクスをフリート全体でスクレイプできます。

新しいターミナルを開いた場合は、クラスター名とリージョンを設定します。

```
export CLUSTER_NAME=ai-eks-docs
export AWS_REGION=us-east-2
```

AMP ワークスペースを作成する

メトリクスを保存する AMP ワークスペースを作成します。

```
aws amp create-workspace \
  --alias "amp-ws-${CLUSTER_NAME}" \
  --region ${AWS_REGION}
```

ワークスペース ID を取得します。

```
AMP_WORKSPACE_ID=$(aws amp list-workspaces \
  --alias "amp-ws-${CLUSTER_NAME}" \
  --query 'workspaces[0].workspaceId' \
  --output text \
  --region ${AWS_REGION})

echo "AMP Workspace ID: ${AMP_WORKSPACE_ID}"
```

リモート書き込みエンドポイントを取得します。

```
AMP_ENDPOINT=$(aws amp describe-workspace \
  --workspace-id ${AMP_WORKSPACE_ID} \
  --query 'workspace.prometheusEndpoint' \
  --output text \
  --region ${AWS_REGION})

echo "AMP Endpoint: ${AMP_ENDPOINT}"
```

IAM ポリシーと EKS Pod Identity の関連付けを作成する

Prometheus がメトリクスをリモート書き込みし、Grafana がメトリクスをクエリできるようにする IAM ポリシーを作成します。

```
ACCOUNT_ID=$(aws sts get-caller-identity --query Account --output text)

AMP_POLICY_ARN=$(aws iam create-policy \
  --policy-name "${CLUSTER_NAME}-amp-grafana-policy" \
```

```
  --policy-document "{\"Version\": \"2012-10-17\", \"Statement\": [{\"Sid\":
 \"AllowAMPReadWrite\", \"Effect\": \"Allow\", \"Action\": [\"aps:ListWorkspaces
\", \"aps:DescribeWorkspace\", \"aps:GetMetricMetadata\", \"aps:GetSeries
\", \"aps:QueryMetrics\", \"aps:RemoteWrite\", \"aps:GetLabels\"],
 \"Resource\": \"arn:aws:aps:${AWS_REGION}:${ACCOUNT_ID}:workspace/*\"},
 {\"Sid\": \"AllowCloudWatchMetrics\", \"Effect\": \"Allow\", \"Action
\": [\"cloudwatch:DescribeAlarmsForMetric\", \"cloudwatch:ListMetrics\",
 \"cloudwatch:GetMetricData\", \"cloudwatch:GetMetricStatistics\"], \"Resource\": \"*
\"}]}" \
  --query 'Policy.Arn' \
  --output text)

echo "AMP Policy ARN: ${AMP_POLICY_ARN}"
```

Prometheus と Grafana のモニタリング名前空間とサービスアカウントを作成します。

```
kubectl create namespace monitoring
kubectl create serviceaccount amp-iamproxy-ingest-service-account -n monitoring
kubectl create serviceaccount grafana-sa -n monitoring
```

EKS Pod Identity の関連付けを作成して、サービスアカウントを IAM ポリシーにリンクします。

```
eksctl create podidentityassociation \
  --cluster ${CLUSTER_NAME} \
  --namespace monitoring \
  --service-account-name amp-iamproxy-ingest-service-account \
  --role-name "${CLUSTER_NAME}-amp-ingest-role" \
  --permission-policy-arns ${AMP_POLICY_ARN} \
  --region ${AWS_REGION}

eksctl create podidentityassociation \
  --cluster ${CLUSTER_NAME} \
  --namespace monitoring \
  --service-account-name grafana-sa \
  --role-name "${CLUSTER_NAME}-grafana-role" \
  --permission-policy-arns ${AMP_POLICY_ARN} \
  --region ${AWS_REGION}
```

両方の EKS Pod Identity の関連付けが作成されていることを確認します。

```
eksctl get podidentityassociation --cluster ${CLUSTER_NAME} --region ${AWS_REGION}
```

予想される出力には、monitoring 名前空間に amp-iamproxy-ingest-service-account と grafana-sa の両方を含める必要があります。

kube-prometheus-stack をインストールする

Helm リポジトリを追加します。

```
helm repo add prometheus-community https://prometheus-community.github.io/helm-charts
helm repo update
```

この値ファイルは、Prometheus、Grafana、およびオペレーターの nodeSelector を省略します。GPU ノードの nvidia.com/gpu:NoSchedule テイントは GPU ノードからそれらを保持するため、デフォルトでシステムまたは汎用プールに配置されます。Node-exporter はワイルドカード許容範囲を使用するため、GPU ノードを含むすべてのノードで実行され、フリート全体でメトリクスを収集します。

値ファイルでは、[ロードバランシングを設定する] で作成した alb IngressClass を使用して、ブラウザアクセス用にインターネット向け ALB をプロビジョニングするように Grafana Ingressも設定されます。alb.ingress.kubernetes.io/inbound-cidrs 注釈はロードバランサーを独自の IP アドレスに制限するので、インストールする前に設定します。

⚠ Grafana は、デフォルトの認証情報を使用して HTTP 経由でパブリックにアクセス可能です
Grafana にはデフォルトの管理者認証情報が含まれており、モニタリングデータへの完全な管理アクセスを提供します。インターネット向けロードバランサーは、このインターフェイスをプレーンテキスト HTTP 経由でパブリックインターネットに配置します。自動スキャナーは、数分以内にパブリックロードバランサーを検出します。alb.ingress.kubernetes.io/inbound-cidrs 注釈を使用してアクセスを制限し、送信元 IP 許可リストは完全な保護ではなく、最小限の保護として扱う必要があります。また、デフォルトの Grafana 管理者パスワードも変更します。体制を強化するには、スキームを internal に設定して内部スキーム (VPC または接続された VPN 内からのみ到達可能) を使用し、TLS 証明書を追加します。

パブリック IP アドレスを検索し、/32 CIDR として保存します。値ファイルは、MY_CIDR 環境変数からこれを読み取ります:

```
export MY_CIDR="$(curl -s https://checkip.amazonaws.com)/32"
```

```
echo $MY_CIDR
```

結果は 203.0.113.4/32 のようになります。ネットワークがアドレスを動的に割り当てる場合、IP アドレスは時間の経過とともに変化する可能性があるため、203.0.113.0/24 などのより広い範囲を設定することを検討してください。

値ファイルを作成します。

Example kube-prometheus-stack 値ファイル

```
cat << EOF > /tmp/kube-prometheus-values.yaml
alertmanager:
  enabled: false

prometheus-adapter:
  enabled: false

prometheus:
  serviceAccount:
    create: false
    name: amp-iamproxy-ingest-service-account
  prometheusSpec:
    serviceAccountName: amp-iamproxy-ingest-service-account
    enableRemoteWriteReceiver: true
    retention: 2h
    scrapeInterval: 30s
    evaluationInterval: 30s
    podMonitorSelectorNilUsesHelmValues: false
    serviceMonitorSelectorNilUsesHelmValues: false
    resources:
      requests:
        cpu: 500m
        memory: 1Gi
      limits:
        memory: 8Gi
    remoteWrite:
      - url: "${AMP_ENDPOINT}api/v1/remote_write"
        sigv4:
          region: "${AWS_REGION}"
        queueConfig:
          maxSamplesPerSend: 1000
          maxShards: 200
          capacity: 2500
```

```
prometheusOperator:
  resources:
    requests:
      cpu: 100m
      memory: 128Mi
    limits:
      memory: 256Mi

kube-state-metrics:
  resources:
    requests:
      cpu: 50m
      memory: 128Mi
    limits:
      memory: 512Mi

grafana:
  enabled: true
  ingress:
    enabled: true
    ingressClassName: alb
    labels:
      guide: ai-eks-docs
    annotations:
      alb.ingress.kubernetes.io/scheme: internet-facing
      alb.ingress.kubernetes.io/target-type: ip
      alb.ingress.kubernetes.io/inbound-cidrs: ${MY_CIDR}
      alb.ingress.kubernetes.io/load-balancer-name: grafana-ai-eks-docs
      alb.ingress.kubernetes.io/healthcheck-path: /api/health
  serviceAccount:
    create: false
    name: grafana-sa
  resources:
    requests:
      cpu: 100m
      memory: 256Mi
    limits:
      memory: 1Gi
  grafana.ini:
    auth.sigv4:
      enabled: true
  sidecar:
    datasources:
```

```
    defaultDatasourceEnabled: false
  plugins:
    - grafana-amazonprometheus-datasource
  additionalDataSources:
    - name: Amazon-Managed-Prometheus
      type: grafana-amazonprometheus-datasource
      access: proxy
      url: "${AMP_ENDPOINT}"
      isDefault: true
      jsonData:
        sigV4Auth: true
        defaultRegion: "${AWS_REGION}"
        sigV4Region: "${AWS_REGION}"
      editable: true
  dashboardProviders:
    dashboardproviders.yaml:
      apiVersion: 1
      providers:
        - name: default
          orgId: 1
          folder: 'GPU Monitoring'
          type: file
          disableDeletion: false
          editable: true
          options:
            path: /var/lib/grafana/dashboards/default
  dashboards:
    default:
      nvidia-dcgm:
        gnetId: 25261
        revision: 1
        datasource:
          - name: DS_PROMETHEUS
            value: Amazon-Managed-Prometheus
      vllm:
        gnetId: 25263
        revision: 1
        datasource:
          - name: DS_PROMETHEUS
            value: Amazon-Managed-Prometheus
      vllm-load-analysis:
        gnetId: 25494
        revision: 1
        datasource:
```

```
          - name: DS_PROMETHEUS
            value: Amazon-Managed-Prometheus

prometheus-node-exporter:
  resources:
    requests:
      cpu: 50m
      memory: 64Mi
    limits:
      memory: 128Mi
  tolerations:
    - operator: Exists
EOF
```

変数が正しく入力されたことを確認します。

```
grep -E "url:|region:" /tmp/kube-prometheus-values.yaml
```

完全な AMP エンドポイント URL (https://aps-workspaces… で始まる) とリージョンが表示されます。いずれかが空の場合は、変数を再エクスポートしてファイルを再作成します。

チャートをインストールします。

```
helm install kube-prometheus-stack prometheus-community/kube-prometheus-stack \
  --namespace monitoring \
  -f /tmp/kube-prometheus-values.yaml
```

ポッドが実行中であることを確認します。

```
kubectl get pods -n monitoring
```

正常な出力:

```
NAME                                                READY   STATUS    RESTARTS
 AGE
kube-prometheus-stack-grafana-7c58f54f77-rftrj      3/3     Running   0
 4m
kube-prometheus-stack-kube-state-metrics-d68dcbc84-5smxq   1/1     Running   0
 4m
kube-prometheus-stack-operator-5895df479f-ttm47     1/1     Running   0
 4m
```

```
kube-prometheus-stack-prometheus-node-exporter-t9q7s       1/1     Running   0
 4m
kube-prometheus-stack-prometheus-node-exporter-x6vfb       1/1     Running   0
 4m
prometheus-kube-prometheus-stack-prometheus-0              2/2     Running   0
 4m
```

スタックは、以下のコンポーネントをデプロイします。

- Prometheus (StatefulSet): メトリクスをスクレイプし、AMP にリモート書き込みします
- Grafana: ダッシュボードと視覚化、AMP データソースで事前設定済み
- kube-state-metrics: Kubernetes オブジェクトの状態 (ポッドのステータス、リソースリクエスト/制限、NodeClaim の状態) に関するメトリクスを生成します
- node-exporter (DaemonSet、ノードごとに 1 つ): ホストレベルのメトリクス (CPU、メモリ、ディスク、ネットワーク) を収集します
- オペレーター: Prometheus とアラートマネージャーのカスタムリソースを管理します

この設定ではアラートマネージャーが無効になっています。

Grafana にアクセスする

インターネット向け AWS Application Load Balancer (ALB) を介して Grafana にアクセスします。ここでは作成しません。kube-prometheus-stack チャートは、Set up load balancing の `alb` `IngressClass` を使用して、値ファイル の `grafana.ingress` 設定からプロビジョニングします。Grafana Helm チャートには組み込みの `Ingress` サポートがあるため、個別のマニフェストは必要ありません。ALB は `grafana-ai-eks-docs` という名前で、ラベル `guide: ai-eks-docs` を持ち、インストール時に `MY_CIDR` で設定した IP アドレスに制限されます。

> **Grafana ALB ターゲットグループにはヘルスチェックパスが必要です**
>
> 値ファイルの `alb.ingress.kubernetes.io/healthcheck-path` 注釈は、ロードバランサーのヘルスチェックを Grafana の `/api/health` エンドポイントに向け、`200` を返します。これがないと、ALB ヘルスチェックはデフォルトで `/` になり、Grafana は `/login` への `302` リダイレクトで応答し、ターゲットグループはエンドポイントを異常としてレポートします。

ロードバランサー URL を指します。ロードバランサーは非同期で作成されるため、1～2 分かかります。

```
echo "http://$(kubectl get ingress kube-prometheus-stack-grafana -n monitoring -o
 jsonpath='{.status.loadBalancer.ingress[0].hostname}')"
```

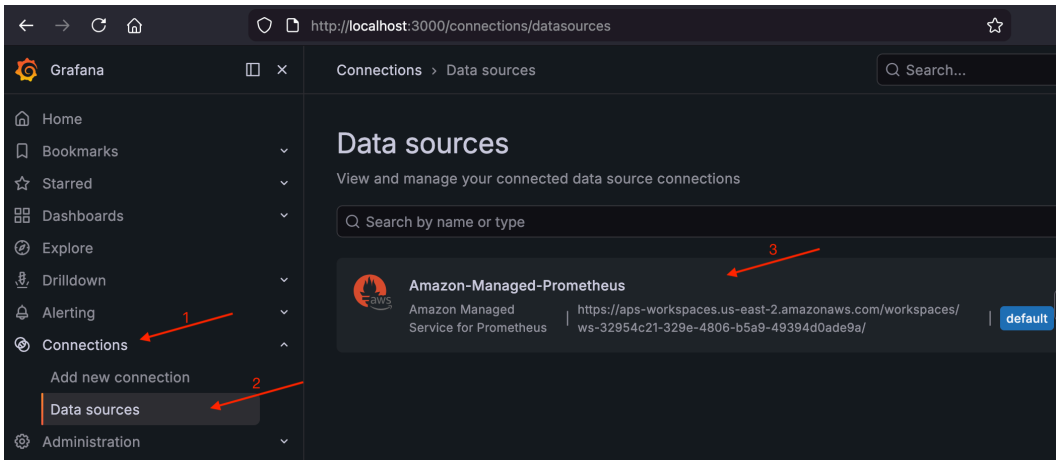
ブラウザで URL を開きます。以下のコマンドからユーザー名 admin とパスワードを使用してログインします。

```
kubectl --namespace monitoring get secrets kube-prometheus-stack-grafana -o
 jsonpath="{.data.admin-password}" | base64 -d ; echo
```

メトリクスパイプラインがエンドツーエンドで機能していることを確認するには:

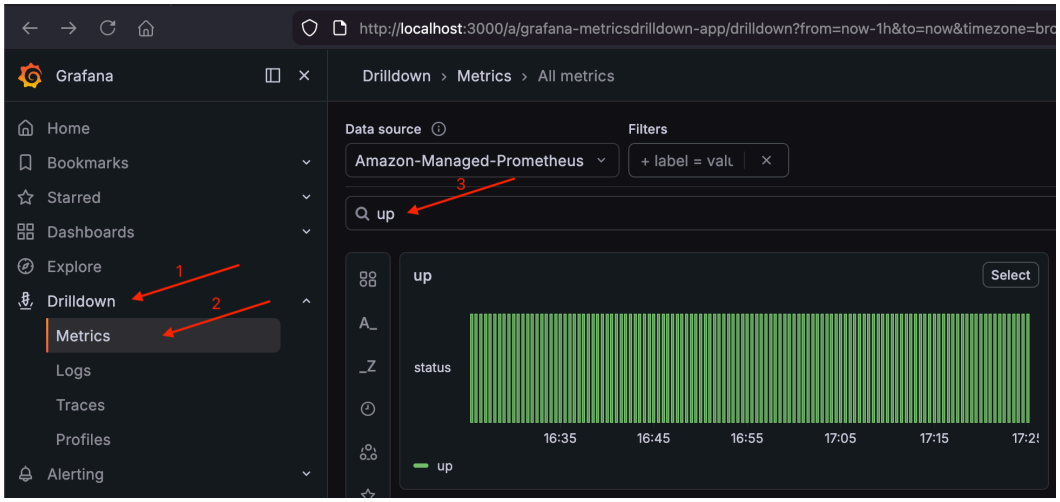
1. [接続] > [データソース] に移動し、Amazon-Managed-Prometheus がデフォルトのデータソースとしてリストされていることを確認します。

   Grafana で AMP データソースを検証する



2. [ドリルダウン] > [メトリクス] に移動し、up メトリクスを検索します。クラスターのスクレイプターゲットの結果が表示されます。

   Grafana で **up** メトリクスを検証する



up が結果を表示する場合、パイプライン (クラスター → Prometheus → AMP → Grafana) は機能しています。

GPU メトリクス用の DCGM Exporter をデプロイする

kube-prometheus-stack はノードレベルの CPU メトリクスとメモリメトリクスを収集しますが、GPU メトリクスは収集しません。NVIDIA DCGM Exporter は、GPU 使用率、メモリ使用量、温度、消費電力、NVLink 帯域幅、テンソルアクティビティを追加します。

```
helm repo add gpu-helm-charts https://nvidia.github.io/dcgm-exporter/helm-charts
helm repo update
```

パスの GPU ノードセレクタキーを設定します。EKS Auto Mode とセルフマネージド Karpenter は、GPU 製造元に異なるラベルキーを使用します。

EKS Auto Mode

```
GPU_NODE_SELECTOR_KEY="eks.amazonaws.com/instance-gpu-manufacturer"
```

Self-managed Karpenter

```
GPU_NODE_SELECTOR_KEY="karpenter.k8s.aws/instance-gpu-manufacturer"
```

DCGM Exporter 値ファイルを作成します。

Example dcgm-exporter 値ファイル

```
cat << EOF > /tmp/dcgm-exporter-values.yaml
resources:
  requests:
    memory: "512Mi"
    cpu: "100m"
  limits:
    memory: "1Gi"
    cpu: "500m"

serviceMonitor:
  enabled: true
  additionalLabels:
    release: kube-prometheus-stack

nodeSelector:
  ${GPU_NODE_SELECTOR_KEY}: nvidia

tolerations:
  - key: "nvidia.com/gpu"
    operator: "Exists"
    effect: "NoSchedule"

customMetrics: |
  # Clocks
  DCGM_FI_DEV_SM_CLOCK,  gauge, SM clock frequency (in MHz).
  DCGM_FI_DEV_MEM_CLOCK, gauge, Memory clock frequency (in MHz).
  # Temperature
  DCGM_FI_DEV_MEMORY_TEMP, gauge, Memory temperature (in C).
  DCGM_FI_DEV_GPU_TEMP,    gauge, GPU temperature (in C).
  # Power
  DCGM_FI_DEV_POWER_USAGE,              gauge, Power draw (in W).
  DCGM_FI_DEV_TOTAL_ENERGY_CONSUMPTION, counter, Total energy consumption since boot
 (in mJ).
  # PCIe
  DCGM_FI_PROF_PCIE_TX_BYTES,  counter, Number of bytes transmitted through PCIe TX (in
 KB) via NVML.
  DCGM_FI_PROF_PCIE_RX_BYTES,  counter, Number of bytes received through PCIe RX (in
 KB) via NVML.
  DCGM_FI_DEV_PCIE_REPLAY_COUNTER, counter, Total number of PCIe retries.
  # Utilization (the sample period varies depending on the product)
  DCGM_FI_DEV_GPU_UTIL,      gauge, GPU utilization (in %).
  DCGM_FI_DEV_MEM_COPY_UTIL, gauge, Memory utilization (in %).
```

```
 DCGM_FI_DEV_ENC_UTIL,      gauge, Encoder utilization (in %).
 DCGM_FI_DEV_DEC_UTIL,      gauge, Decoder utilization (in %).
 # Errors and violations
 DCGM_FI_DEV_XID_ERRORS,            gauge, Value of the last XID error encountered.
 DCGM_EXP_XID_ERRORS_COUNT,         gauge, Value of count of XID errors encountered.
 DCGM_FI_DEV_POWER_VIOLATION,       counter, Throttling duration due to power
constraints (in us).
 DCGM_FI_DEV_THERMAL_VIOLATION,     counter, Throttling duration due to thermal
constraints (in us).
 DCGM_FI_DEV_SYNC_BOOST_VIOLATION,  counter, Throttling duration due to sync-boost
constraints (in us).
 DCGM_FI_DEV_BOARD_LIMIT_VIOLATION, counter, Throttling duration due to board limit
constraints (in us).
 DCGM_FI_DEV_LOW_UTIL_VIOLATION,    counter, Throttling duration due to low
utilization (in us).
 DCGM_FI_DEV_RELIABILITY_VIOLATION, counter, Throttling duration due to reliability
constraints (in us).
 # Memory usage
 DCGM_FI_DEV_FB_FREE, gauge, Framebuffer memory free (in MiB).
 DCGM_FI_DEV_FB_USED, gauge, Framebuffer memory used (in MiB).
 # Retired pages
 DCGM_FI_DEV_RETIRED_SBE,     counter, Total number of retired pages due to single-bit
errors.
 DCGM_FI_DEV_RETIRED_DBE,     counter, Total number of retired pages due to double-bit
errors.
 DCGM_FI_DEV_RETIRED_PENDING, counter, Total number of pages pending retirement.
 # NVLink
 DCGM_FI_DEV_NVLINK_BANDWIDTH_TOTAL, counter, Total number of NVLink bandwidth
counters for all lanes.
 DCGM_FI_PROF_NVLINK_TX_BYTES,       counter, The rate of data transmitted over NVLink
not including protocol headers in bytes per second.
 DCGM_FI_PROF_NVLINK_RX_BYTES,       counter, The rate of data received over NVLink
not including protocol headers in bytes per second.
 # DCP metrics
 DCGM_FI_PROF_GR_ENGINE_ACTIVE,   gauge, Ratio of time the graphics engine is active
(in %).
 DCGM_FI_PROF_SM_ACTIVE,          gauge, The ratio of cycles an SM has at least 1 warp
assigned (in %).
 DCGM_FI_PROF_SM_OCCUPANCY,       gauge, The ratio of number of warps resident on an
SM (in %).
 DCGM_FI_PROF_PIPE_TENSOR_ACTIVE, gauge, Ratio of cycles the tensor (HMMA) pipe is
active (in %).
 DCGM_FI_PROF_DRAM_ACTIVE,        gauge, Ratio of cycles the device memory interface
is active sending or receiving data (in %).
```

```
  DCGM_FI_DEV_CLOCK_THROTTLE_REASONS, gauge, Current clock throttle reasons (bitmask of
 DCGM_CLOCKS_THROTTLE_REASON_*).
  DCGM_FI_DEV_GPU_NVLINK_ERRORS,      gauge, Identifies a GPU NVLink error type
 returned by DCGM_FI_DEV_GPU_NVLINK_ERRORS.
  ## NVLink
  DCGM_FI_DEV_NVLINK_BANDWIDTH_L0, counter, The number of bytes of active NVLink rx or
 tx data including both header and payload.
  ## Remapped rows
  DCGM_FI_DEV_UNCORRECTABLE_REMAPPED_ROWS, counter, Number of remapped rows for
 uncorrectable errors.
  DCGM_FI_DEV_CORRECTABLE_REMAPPED_ROWS, counter, Number of remapped rows for
 correctable errors.
  DCGM_FI_DEV_ROW_REMAP_FAILURE, gauge, whether remapping of rows has failed.
  ## Profiling metrics
  DCGM_FI_PROF_PIPE_FP64_ACTIVE, gauge, Ratio of cycles the fp64 pipes are active (in
 %).
  DCGM_FI_PROF_PIPE_FP32_ACTIVE, gauge, Ratio of cycles the fp32 pipes are active (in
 %).
  DCGM_FI_PROF_PIPE_FP16_ACTIVE, gauge, Ratio of cycles the fp16 pipes are active (in
 %).
  # ECC
  DCGM_FI_DEV_ECC_SBE_VOL_TOTAL, counter, Total number of single-bit volatile ECC
 errors.
  DCGM_FI_DEV_ECC_DBE_VOL_TOTAL, counter, Total number of double-bit volatile ECC
 errors.
EOF
```

customMetrics フィールドは、DCGM Exporter のデフォルトメトリクスセットを、NVLink 帯域幅、テンソルアクティビティ、PCIe スループット、ECC エラー、および熱スロットリングを含む拡張メトリクスセットで上書きします。推論ワークロードの場合、これらは GPU コンピューティングユニットが十分に使用されているかどうか、バッチサイズが低いためにリクエスト間で GPU がアイドル状態になっているかどうか、CPU と GPU 間のデータ転送がボトルネックになっているかどうか、熱スロットリングがレイテンシーの急増を引き起こしているかどうか、より大きなバッチを処理するための GPU メモリのヘッドルームがどれだけ残っているかを理解するのに役立ちます。

DCGM Exporter をインストールします。

```
helm install dcgm-exporter gpu-helm-charts/dcgm-exporter \
  --namespace monitoring \
  -f /tmp/dcgm-exporter-values.yaml
```

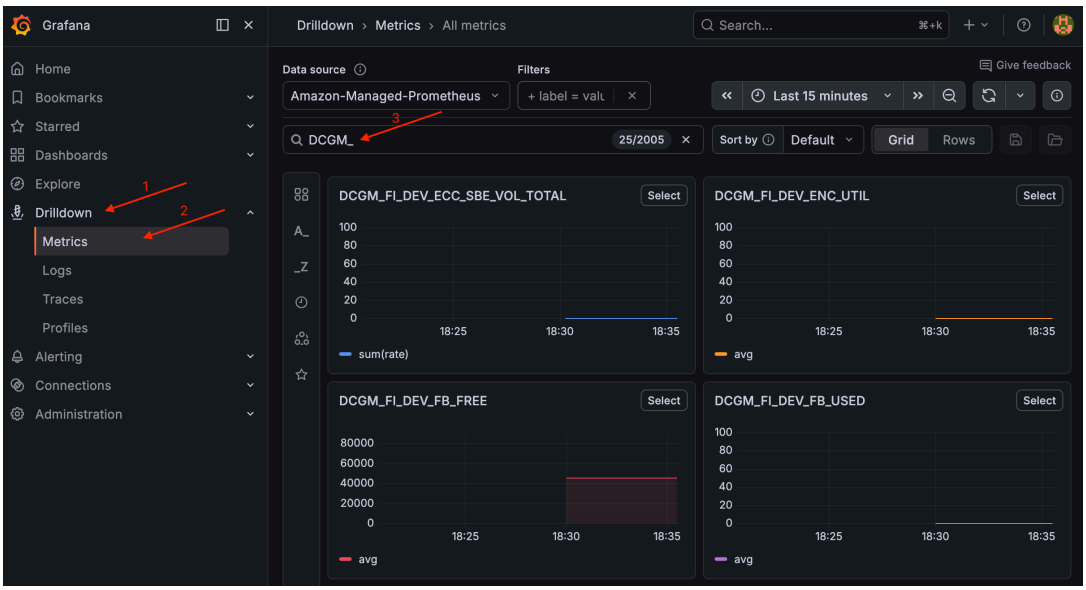
`tolerations` は、ステップ 2 でプロビジョニングした GPU テイントノードでエクスポーターを実行できるようにします。`release: kube-prometheus-stack` ラベル付きの `serviceMonitor` により、Prometheus はエクスポーターを自動的に検出してスクレイプします。

DCGM Exporter DaemonSet を確認します。

```
kubectl get daemonset dcgm-exporter -n monitoring
```

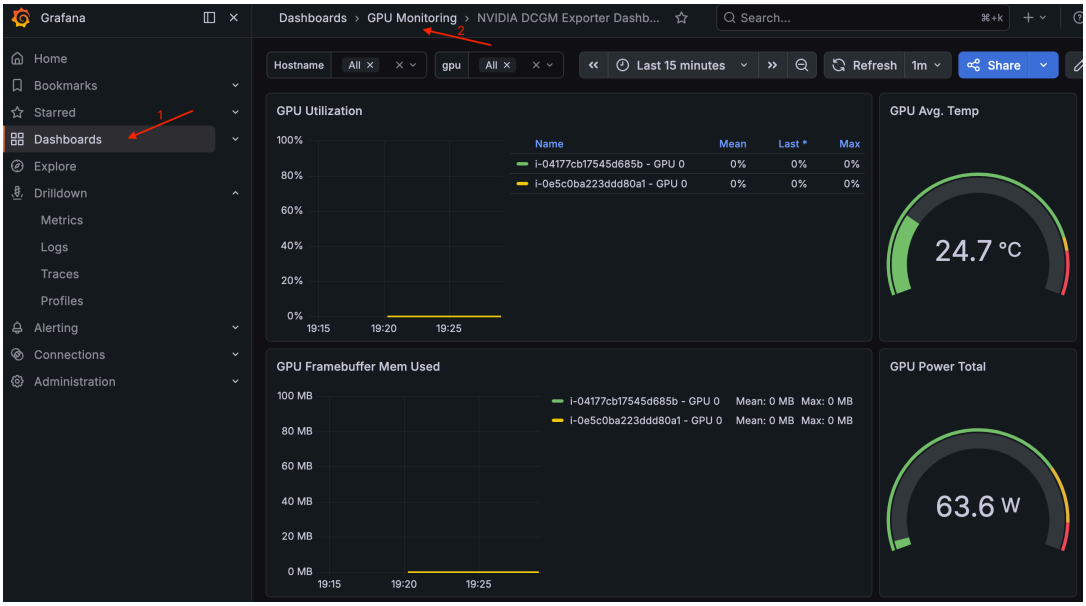
GPU ノードが実行されると、準備完了状態のポッドが 1 つ表示されます。DCGM メトリクスを検証するには、Grafana で [ドリルダウン] > [メトリクス] に移動し、`DCGM_` を検索します。

Grafana で DCGM メトリクスを検証する



ダッシュボードを表示するには、[ダッシュボード] > [GPU モニタリング] > [NVIDIA DCGM Exporter ダッシュボード] に移動します。

Grafana の NVIDIA DCGM Exporter ダッシュボード



モデルの重み S3 バケット

モデルの重みを保存するための Amazon S3 バケットを作成し、ワークロードポッドがそのバケットに対して読み書きできるように EKS Pod Identity の関連付けを設定します。

新しいターミナルを開いた場合は、クラスター名とリージョンを設定します。

```
export CLUSTER_NAME=ai-eks-docs
export AWS_REGION=us-east-2
```

S3 バケットの作成

名前の競合を避けるために、ランダムなサフィックスでバケットを作成します。

```
BUCKET_SUFFIX=$(head -c 4 /dev/urandom | od -An -tx1 | tr -d ' \n')
MODEL_BUCKET="${CLUSTER_NAME}-models-${BUCKET_SUFFIX}"

aws s3 mb s3://${MODEL_BUCKET} --region ${AWS_REGION}
```

2023 年 1 月以降に作成された S3 バケットでは、サーバー側の暗号化 (AES256) とパブリックアクセスのブロックがデフォルトで有効になっています。

S3 アクセス用の EKS Pod Identity を設定する

`default` 名前空間に `model-storage-sa` ServiceAccount、モデルバケットを対象とする IAM ポリシー、およびそれらをリンクする EKS Pod Identity の関連付けを作成しま

す。serviceAccountName: model-storage-sa を設定したワークロードポッドは、バケットを読み書きできます。

```
kubectl create serviceaccount model-storage-sa
```

IAM ポリシーを作成します。

```
POLICY_ARN=$(aws iam create-policy \
  --policy-name "${CLUSTER_NAME}-model-storage-policy" \
  --policy-document "{\"Version\": \"2012-10-17\", \"Statement\": [{\"Effect\":
 \"Allow\", \"Action\": [\"s3:GetObject\", \"s3:PutObject\", \"s3:ListBucket\",
 \"s3:DeleteObject\"], \"Resource\": [\"arn:aws:s3:::${MODEL_BUCKET}\", \"arn:aws:s3:::
${MODEL_BUCKET}/*\"]}]}" \
  --query 'Policy.Arn' \
  --output text)

echo "Policy ARN: ${POLICY_ARN}"
```

> **Note**
>
> このポリシーは、検証ステップに s3:DeleteObject と s3:PutObject を付与します。モデルの重みのみを読み取る本番推論ポッドの場合は、s3:PutObject と s3:DeleteObject を削除して最小特権に従います。

EKS Pod Identity の関連付けを作成します。eksctl は正しい信頼ポリシーを使用して IAM ロールを作成し、ServiceAccount にリンクします。

```
eksctl create podidentityassociation \
  --cluster ${CLUSTER_NAME} \
  --namespace default \
  --service-account-name model-storage-sa \
  --role-name "${CLUSTER_NAME}-model-storage-role" \
  --permission-policy-arns ${POLICY_ARN} \
  --region ${AWS_REGION}
```

関連付けを確認します。

```
eksctl get podidentityassociation --cluster ${CLUSTER_NAME} --region ${AWS_REGION}
```

出力には、default 名前空間の model-storage-sa 関連付けを含める必要があります。

ポッドからの S3 アクセスを検証する

model-storage-sa ServiceAccount を使用して AWS CLI イメージで 1 回限りのポッドを実行し、EKS Pod Identity が正しく設定され、S3 アクセスが機能することを確認します。

```
cat << EOF | kubectl apply -f -
apiVersion: v1
kind: Pod
metadata:
  name: s3-test
  labels:
    guide: ai-eks-docs
spec:
  serviceAccountName: model-storage-sa
  containers:
    - name: aws-cli
      image: public.ecr.aws/aws-cli/aws-cli:2.27.0
      command:
        - sh
        - -c
        - |
          echo "=== Caller Identity ==="
          aws sts get-caller-identity
          echo ""
          echo "=== S3 Write Test ==="
          echo "pod identity works" | aws s3 cp - s3://${MODEL_BUCKET}/test.txt
          echo ""
          echo "=== S3 List Test ==="
          aws s3 ls s3://${MODEL_BUCKET}/
          echo ""
          echo "=== S3 Delete Test ==="
          aws s3 rm s3://${MODEL_BUCKET}/test.txt
  restartPolicy: Never
EOF
```

ポッドが完了するのを待ち、ログを確認します。

```
kubectl wait --for=jsonpath='{.status.phase}'=Succeeded pod/s3-test --timeout=300s
kubectl logs s3-test
```

正常な出力:

```
=== Caller Identity ===
{
    "UserId": "AROA...:eks-ai-eks-docs-model-s-...",
    "Account": "123456789012",
    "Arn": "arn:aws:sts::123456789012:assumed-role/ai-eks-docs-model-storage-role/eks-
ai-eks-docs-model-s-..."
}

=== S3 Write Test ===
upload: - to s3://ai-eks-docs-models-01234567/test.txt

=== S3 List Test ===
2026-05-04 12:00:00         19 test.txt

=== S3 Delete Test ===
delete: s3://ai-eks-docs-models-01234567/test.txt
```

発信者 ID は、EKS Pod Identity を介してポッドが `${CLUSTER_NAME}-model-storage-role` ロールを引き受けたことを確認します。S3 コマンドは読み取りおよび書き込みアクセスを確認します。

テストポッドをクリーンアップします。

```
kubectl delete pod s3-test
```

次のステップ

クラスターの準備ができたら、「モデルのロードと提供」に進み、大規模な言語モデルをデプロイし、推論エンドポイントとやり取りできます。

クリーンアップ

> **Tip**
>
> このガイドの次のセクションに進む場合は、完全なクリーンアップをスキップします。完了時にのみ実行します。

```
export CLUSTER_NAME=ai-eks-docs
export AWS_REGION=us-east-2
```

```
kubectl delete pod nvidia-smi --ignore-not-found
kubectl delete deployment gpu-overflow-test --ignore-not-found
```

キャパシティ予約をキャンセルする

ODCR を作成した場合は、最初にキャンセルします。予約 ID を検索します。

```
INSTANCE_TYPE="g6e.4xlarge"
CAPACITY_RESERVATION_ID=$(aws ec2 describe-capacity-reservations \
  --filters "Name=state,Values=active" "Name=instance-type,Values=${INSTANCE_TYPE}" \
  --query 'CapacityReservations[0].CapacityReservationId' \
  --output text \
  --region ${AWS_REGION})
echo "Capacity reservation ID: ${CAPACITY_RESERVATION_ID}"
```

予約をキャンセルします。

```
aws ec2 cancel-capacity-reservation --capacity-reservation-id
 ${CAPACITY_RESERVATION_ID}
```

> **Important**
>
> 予約をキャンセルしても、実行中のインスタンスは終了しません。これらは、終了するまで標準のオンデマンド料金で継続されます。キャンセルする前に、最初にデプロイを削除してリザーブドノードをドレインします。

モニタリングをクリーンアップする

IAM ポリシー ARN を検索します。

```
AMP_POLICY_ARN=$(aws iam list-policies \
  --scope Local \
  --query "Policies[?PolicyName=='${CLUSTER_NAME}-amp-grafana-policy'].Arn" \
  --output text)
echo "AMP Policy ARN: ${AMP_POLICY_ARN}"
```

AMP ワークスペース ID を検索します。

```
AMP_WORKSPACE_ID=$(aws amp list-workspaces \
```

```
  --alias "amp-ws-${CLUSTER_NAME}" \
  --query 'workspaces[0].workspaceId' \
  --output text \
  --region ${AWS_REGION})
echo "AMP Workspace ID: ${AMP_WORKSPACE_ID}"
```

DCGM Exporter Helm リリースをアンインストールします。

```
helm uninstall dcgm-exporter -n monitoring
```

kube-prometheus-stack Helm リリースをアンインストールします。

```
helm uninstall kube-prometheus-stack -n monitoring
```

Prometheus 取り込みサービスアカウントの EKS Pod Identity の関連付けを削除します。

```
eksctl delete podidentityassociation \
  --cluster ${CLUSTER_NAME} \
  --namespace monitoring \
  --service-account-name amp-iamproxy-ingest-service-account \
  --region ${AWS_REGION}
```

Grafana サービスアカウントの EKS Pod Identity の関連付けを削除します。

```
eksctl delete podidentityassociation \
  --cluster ${CLUSTER_NAME} \
  --namespace monitoring \
  --service-account-name grafana-sa \
  --region ${AWS_REGION}
```

Prometheus と Grafana で使用される IAM ポリシーを削除します。

```
aws iam delete-policy --policy-arn ${AMP_POLICY_ARN}
```

AMP ワークスペースを削除します。

```
aws amp delete-workspace --workspace-id ${AMP_WORKSPACE_ID} --region ${AWS_REGION}
```

モニタリング名前空間を削除します。

```
kubectl delete namespace monitoring
```

モデルの重み S3 バケットをクリーンアップする

モデルバケット名を検索します。

```
MODEL_BUCKET=$(aws s3api list-buckets \
  --query "Buckets[?starts_with(Name, '${CLUSTER_NAME}-models-')].Name | [0]" \
  --output text)
echo "Model bucket: ${MODEL_BUCKET}"
```

IAM ポリシー ARN を検索します。

```
POLICY_ARN=$(aws iam list-policies \
  --scope Local \
  --query "Policies[?PolicyName=='${CLUSTER_NAME}-model-storage-policy'].Arn" \
  --output text)
echo "Policy ARN: ${POLICY_ARN}"
```

S3 モデルバケットとそのすべてのオブジェクトを削除します。

```
aws s3 rb s3://${MODEL_BUCKET} --force
```

EKS Pod Identity の関連付けを削除します。

```
eksctl delete podidentityassociation \
  --cluster ${CLUSTER_NAME} \
  --namespace default \
  --service-account-name model-storage-sa \
  --region ${AWS_REGION}
```

IAM ポリシーを削除します。

```
aws iam delete-policy --policy-arn ${POLICY_ARN}
```

Kubernetes ServiceAccount を削除します。

```
kubectl delete serviceaccount model-storage-sa
```

残りのリソースとクラスターを削除する

```
kubectl delete nodepool gpu-inf --ignore-not-found
kubectl delete nodeclass gpu-inf --ignore-not-found
kubectl delete ec2nodeclass gpu-inf --ignore-not-found
eksctl delete cluster --name=$CLUSTER_NAME --region=$AWS_REGION
```

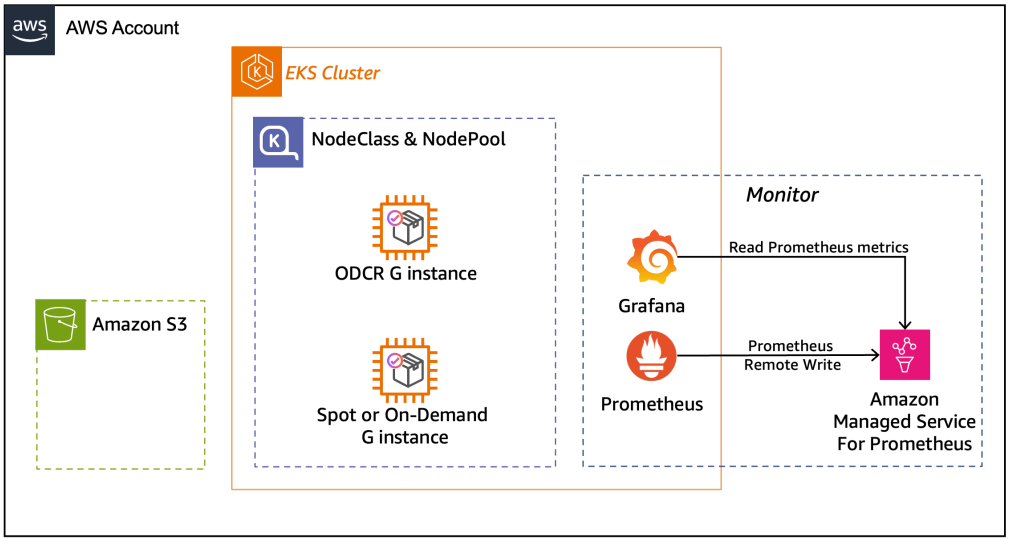
Terraform を使用して AI/ML ワークロード用の Amazon EKS クラスターを設定する

> **Tip**
>
> 今後開催予定の Amazon EKS AI/ML ワークショップに登録してください。

このセクションでは、Terraform を使用して Amazon EKS でトレーニングまたは推論ワークロードを実行するために必要なインフラストラクチャを作成する手順について説明します。ステップには、EKS クラスターの作成、EKS Auto Mode または Karpenter を使用した GPU 対応ノードの作成、Prometheus と Grafana を使用したモニタリングスタック、モデルの重み付け用の Amazon S3 ストレージが含まれます。

これらの機能が EKS クラスターで EC2 インスタンスをプロビジョニングおよび自動スケールする方法の詳細については、EKS Auto Mode と Karpenter のドキュメントを参照してください。

高レベルアーキテクチャとワークフロー



この図は、このセクションのセットアップの AWS 高レベルアーキテクチャを示しています。

前提条件

> **Important**
>
> EKS クラスター、GPU インスタンス、Application Load Balancer、Amazon Managed Service for Prometheus など、このチュートリアルで作成するリソースには料金が発生します。終了したリソースは、継続的な料金の発生を回避するために削除してください。

- Terraform >= 1.15.0。セットアップ手順については、「Terraform のインストール」を参照してください。
- `kubectl` >= 1.36。セットアップ手順については、「the section called “kubectl と eksctl のセットアップ”」を参照してください。
- AWS CLI >= 2.27。セットアップ手順については、「AWS CLI の最新バージョンのインストールまたは更新」を参照してください。
- `jq`。セットアップ手順については、「Download jq」を参照してください。

ツールのバージョンを確認します。

```
terraform --version
aws --version
kubectl version --client
jq --version
```

ステップ 1: Terraform コードをダウンロードしてデプロイする

このチュートリアルでは、sample-eks-docs AWS Samples GitHub リポジトリの Terraform コードを使用します。作業ディレクトリにリポジトリのクローンを作成します:

```
git clone git@github.com:aws-samples/sample-eks-docs.git
cd sample-eks-docs/ai-ml/set-up-cluster
```

リポジトリには、先ほど変更した `ai-ml/set-up-cluster/` ディレクトリの下に次の構造があります。

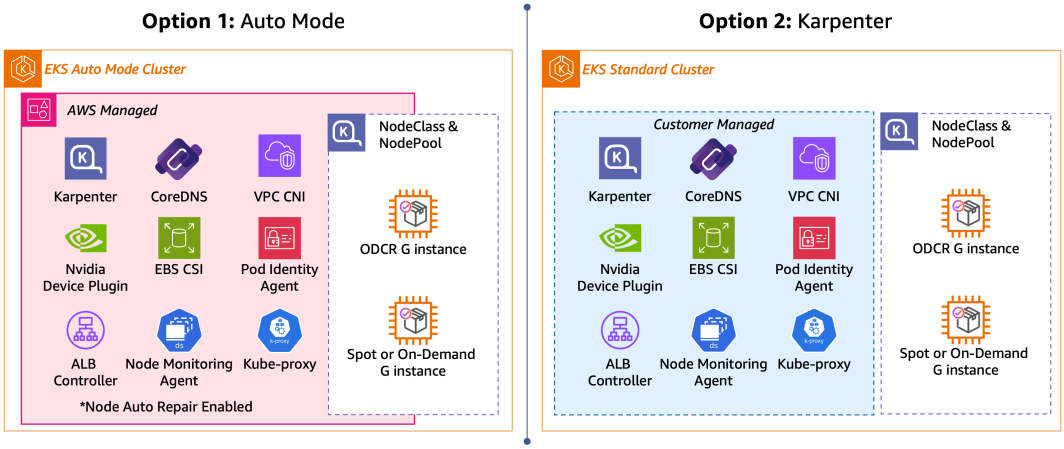
```
set-up-cluster/
### scripts/
#   ### cleanup.sh
### terraform/
    ### auto-mode/
    ### karpenter/
```

リポジトリには 2 つのデプロイパスがあります。ガイド全体で 1 つのオプションのみを選択し、それを使用します。

- EKS Auto Mode (`terraform/auto-mode/`) – コアネットワーク、ストレージ、ロードバランシングアドオンに加えて、EKS Auto Mode には、EKS ノードモニタリングエージェント、自動ノード修復、高速コンテナプル用の SOCI スナップショッター、デフォルトの NodeClass の GPU 準備など、トレーニングおよび推論ワークロード向けの以下の機能が含まれており、管理されています。NVIDIA デバイスプラグインは、EKS Auto Mode が GPU 対応ノードに使用する Bottlerocket 高速 AMI に含まれています。
- セルフマネージド Karpenter (`terraform/karpenter/`) – EKS Auto Mode のない EKS クラスターでは、Terraform コードがトレーニングおよび推論ワークロードに必要なコンポーネントをインストールおよび設定します。これには、ネットワーキングアドオン (VPC CNI、CoreDNS、kube-proxy)、Karpenter、EKS ノードモニタリングエージェント、NVIDIA デバイスプラグイン、高速コンテナプル用の SOCI スナップショットターが含まれます。

> **Important**
>
> EKS Auto Mode またはセルフマネージド Karpenter を選択し、ガイド全体で使用します。ストリームの途中で切り替えるには、クラスターを破棄して最初からやり直す必要があります。

EKS クラスターオプション: EKS Auto Mode とセルフマネージド Karpenter



⚠ Grafana は、デフォルトの認証情報を使用して HTTP 経由でパブリックにアクセス可能です

Grafana ALB Ingress ではデフォルトで `var.my_cidr` が `0.0.0.0/0` になります。これにより、Grafana はデフォルトの管理者認証情報を使用してプレーン HTTP 経由でパブリックインターネットに公開されます。自動スキャナーは、数分以内にパブリックロードバランサーを検出します。独自の IP アドレスで `var.my_cidr` を上書きしてアクセスを制限する必要があります。

```
export MY_CIDR="$(curl -s https://checkip.amazonaws.com)/32"
terraform apply -var "my_cidr=${MY_CIDR}"
```

送信元 IP 許可リストは、完全な保護ではなく、最小限の保護手段として扱います。また、最初のログイン後にデフォルトの Grafana 管理者パスワードを変更します。体制を強化するには、`alb.ingress.kubernetes.io/scheme` を `internal` (VPC または接続された VPN 内からのみ到達可能) に変更し、TLS 証明書を追加します。

クラスターをデプロイする

選択したパスのディレクトリに移動し、Terraform を初期化して適用します。

どちらのバリアントもデフォルトで `us-east-2` リージョンになります。別のリージョンにデプロイするには、次のステップで `terraform apply` コマンドに `-var "region=region-code"` を追加します。ここで、*region-code* はデプロイする AWS リージョンです。

Terraform コードは、ターゲットリージョン内の利用可能なすべてのアベイラビリティーゾーンを使用しますが、Amazon EKS はこれらのゾーンでのコントロールプレーン配置をサポートしていないため、`use1-az3`、`usw1-az2`、および `cac1-az3` は除外されます。

EKS Auto Mode

```
cd terraform/auto-mode
terraform init
terraform apply
```

このコマンドの完了には数分かかります。

Self-managed Karpenter

```
cd terraform/karpenter
terraform init
terraform apply
```

このコマンドの実行には約 15 分かかります。アドオンと Karpenter コントローラーのホスティング専用のマネージドノードグループを持つ EKS クラスターを作成します。Terraform は、スポット中断キューが有効で、NodeRepair および StaticCapacity 機能ゲートがオンになっている Karpenter をインストールします。また、NVIDIA デバイスプラグイン、AWS Load Balancer Controller、モニタリングスタックもインストールします。

Terraform 出力を確認する

適用が完了すると、Terraform は次の出力を表示します (値は設定によって異なります)。

```
Apply complete! Resources: 74 added, 0 changed, 0 destroyed.

Outputs:

cluster_name         = "ai-eks-docs"
configure_kubectl    = "aws eks update-kubeconfig --region us-east-2 --name ai-eks-docs
 --alias ai-eks-docs"
configure_model_bucket = "export MODEL_BUCKET=ai-eks-docs-models-20250612abc1"
model_bucket         = "ai-eks-docs-models-20250612abc1"
node_iam_role_name   = "ai-eks-docs-eks-auto-20250612..."
region               = "us-east-2"
```

configure_kubectl 出力は、クラスターに kubectl を向けるすぐに実行できるコマンドです。model_bucket 出力には、モデルの重みの S3 バケット名が含まれます。node_iam_role_name 出力には、ノードが使用する IAM ロールが表示されます。

kubectl を設定する

新しいクラスターに kubectl を向けます。configure_kubectl 出力はすぐに実行できるコマンドです。

```
eval "$(terraform output -raw configure_kubectl)"
```

クラスターを確認する

EKS Auto Mode

```
kubectl get pods --all-namespaces
```

正常な出力:

```
NAMESPACE     NAME                                                  READY
 STATUS    RESTARTS   AGE
kube-system   metrics-server-5db89f9ffd-h4mlr                       1/1
 Running   0          3m
kube-system   metrics-server-5db89f9ffd-nd748                       1/1
 Running   0          3m
monitoring    kube-prometheus-stack-grafana-ff9b5fd57-dh562         3/3
 Running   0          3m
monitoring    kube-prometheus-stack-kube-state-metrics-5dcbfdf69b-wd6qn  1/1
 Running   0          3m
monitoring    kube-prometheus-stack-operator-548c4f4485-m5wh2       1/1
 Running   0          3m
monitoring    kube-prometheus-stack-prometheus-node-exporter-p2z7l  1/1
 Running   0          3m
monitoring    prometheus-kube-prometheus-stack-prometheus-0         2/2
 Running   0          3m
```

EKS Auto Mode では、VPC CNI、kube-proxy、および CoreDNS はマネージドコンポーネントとして実行され、kube-system ではポッドとして表示されません。

Self-managed Karpenter

```
kubectl get pods --all-namespaces
```

予想される出力には、Karpenter、CoreDNS、kube-proxy、aws-node (VPC CNI)、EKS Pod Identity Agent、EKS ノードモニタリングエージェント、および NVIDIA デバイスプラグインが含まれます:

```
NAMESPACE     NAME
 READY   STATUS    RESTARTS   AGE
kube-system   aws-node-bzdcz                                                   2/2
    Running   0          5m
kube-system   aws-node-vkbhb                                                   2/2
    Running   0          5m
kube-system   coredns-7dbb8998cf-9b9wk                                         1/1
    Running   0          5m
kube-system   coredns-7dbb8998cf-pwtjd                                         1/1
    Running   0          5m
kube-system   ebs-csi-controller-748f54b69-8h7jv                               6/6
    Running   0          5m
kube-system   ebs-csi-controller-748f54b69-v2mv8                               6/6
    Running   0          5m
kube-system   eks-node-monitoring-agent-5qw4l                                  1/1
    Running   0          5m
kube-system   eks-node-monitoring-agent-7lrtm                                  1/1
    Running   0          5m
kube-system   eks-pod-identity-agent-ddvlv                                     1/1
    Running   0          5m
kube-system   eks-pod-identity-agent-q4g29                                     1/1
    Running   0          5m
kube-system   karpenter-898ff78-cndbd                                          1/1
    Running   0          5m
kube-system   karpenter-898ff78-hfbhn                                          1/1
    Running   0          5m
kube-system   kube-proxy-gcfnh                                                 1/1
    Running   0          5m
kube-system   kube-proxy-tktcf                                                 1/1
    Running   0          5m
kube-system   metrics-server-5b789db597-cm9qd                                  1/1
    Running   0          5m
kube-system   metrics-server-5b789db597-w57b6                                  1/1
    Running   0          5m
kube-system   nvidia-device-plugin-node-feature-discovery-gc-66cb7f5dc-xvftc    1/1
    Running   0          5m
kube-system   nvidia-device-plugin-node-feature-discovery-master-854d6b5hnqtp   1/1
    Running   0          5m
monitoring    kube-prometheus-stack-grafana-6797bcb59f-2zlwg                   3/3
    Running   0          5m
monitoring    kube-prometheus-stack-kube-state-metrics-8446bd549c-q6gjc        1/1
    Running   0          5m
```

```
monitoring    kube-prometheus-stack-operator-5f499b784-vf5xb                    1/1
    Running   0          5m
monitoring    kube-prometheus-stack-prometheus-node-exporter-4c6wq              1/1
    Running   0          5m
monitoring    kube-prometheus-stack-prometheus-node-exporter-l5t25              1/1
    Running   0          5m
monitoring    prometheus-kube-prometheus-stack-prometheus-0                     2/2
    Running   0          5m
```

NVIDIA デバイスプラグインがインストールされていることを確認します。一致する amiFamily=al2023 ラベルを持つ GPU NodePool がプロビジョニングされるまで、ゼロデバイスプラグイン Pod は表示されません:

```
kubectl get daemonset nvidia-device-plugin -n kube-system
```

正常な出力:

```
NAME                   DESIRED   CURRENT   READY   UP-TO-DATE   AVAILABLE   NODE
 SELECTOR      AGE
nvidia-device-plugin   0         0         0       0            0
 amiFamily=al2023   5m
```

ステップ 2: 動的 GPU NodePool を作成する

GPU NodePools はオプトインです。デフォルトでは、terraform apply は GPU キャパシティや GPU 請求なしでクラスターとモニタリングスタックを作成します。GPU ノードをプロビジョニングするには、戦略名を付けて nodepools 変数を渡します。

On-Demand をフォールバックとした Spot キャパシティを使用して、世代が 4 より大きい G ファミリー GPU インスタンスをプロビジョニングする spot-ondemand 戦略を有効にします:

```
terraform apply -var 'nodepools={"spot-ondemand"={}}'
```

このコマンドは、nodepools/spot-ondemand/ ディレクトリから NodePool テンプレートと NodeClass テンプレートを適用します。どちらのパスも同じ NodePool API を使用しますが、NodePool が参照する NodeClass が異なります。

EKS Auto Mode

NodePool は、Bottlerocket 高速 AMI、NVIDIA ドライバー、NVIDIA デバイスプラグイン、SOCI 並列プルを既に選択しているマネージド `default` NodeClass を参照します。この`spot-ondemand` 戦略は、このパスに独自の NodeClass を提供しません。

NodePool を検証します:

```
kubectl get nodepools,nodeclasses
```

正常な出力 `gpu-inf` NodePool は組み込みの `general-purpose` と `system` NodePool を結合し、3 つすべてがマネージド `default` NodeClass を参照します:

```
NAME                                    NODECLASS   NODES   READY   AGE
nodepool.karpenter.sh/general-purpose   default     0       True    12m
nodepool.karpenter.sh/gpu-inf           default     0       True    20s
nodepool.karpenter.sh/system            default     1       True    12m

NAME                                  ROLE                         READY   AGE
nodeclass.eks.amazonaws.com/default   ai-eks-docs-eks-auto-...     True    12m
```

Self-managed Karpenter

Terraform は、NodePool と一緒にカスタム `gpu-inf` `EC2NodeClass` を適用します。`EC2NodeClass` は、EKS 最適化 AL2023 AMI エイリアスをピン留めし、`FastImagePull` 機能ゲートを介して SOCI を有効にし、コンテナ化されたイメージキャッシュをローカル NVMe に移動するように `instanceStorePolicy: RAID0` を設定します。

NodePool と EC2NodeClass を検証します:

```
kubectl get nodepools,ec2nodeclasses
```

正常な出力 `gpu-inf` ペアは、Terraform が GPU 以外のワークロード用に作成する `general-purpose` NodePool と EC2NodeClass を結合します:

```
NAME                                    NODECLASS          NODES   READY   AGE
nodepool.karpenter.sh/general-purpose   general-purpose    1       True    14m
nodepool.karpenter.sh/gpu-inf           gpu-inf            0       True    25s

NAME                                              READY   AGE
ec2nodeclass.karpenter.k8s.aws/general-purpose    True    14m
```

```
ec2nodeclass.karpenter.k8s.aws/gpu-inf           True    25s
```

GPU ワークロードがスケジュールされるまで、両方のパスに gpu-inf の 0 ノードが表示されます。EKS Auto Mode と Karpenter は、保留中の Pod がノードを必要とする場合にのみノードを起動します。

ステップ 3: サンプル Pod でテストする

nvidia-smi Pod を使用して GPU NodePool のセットアップをテストします:

```
cat << EOF | kubectl apply -f -
apiVersion: v1
kind: Pod
metadata:
  name: nvidia-smi
  labels:
    guide: ai-eks-docs
spec:
  restartPolicy: OnFailure
  tolerations:
    - key: "nvidia.com/gpu"
      operator: "Exists"
      effect: "NoSchedule"
  containers:
    - name: nvidia-smi
      image: public.ecr.aws/amazonlinux/amazonlinux:2023-minimal
      command: ["nvidia-smi"]
      resources:
        limits:
          nvidia.com/gpu: 1
EOF
```

Pod がスケジュールされ、正常に完了していることを確認します:

```
kubectl get pods nvidia-smi
```

正常な出力:

```
NAME         READY   STATUS      RESTARTS   AGE
nvidia-smi   0/1     Completed   0          67s
```

STATUS: Completed は、nvidia-smi コマンドが実行され、終了したことを意味します。Pod ログをチェックして、ノードによって検出された GPU を確認します:

```
kubectl logs nvidia-smi
```

正常な出力:

```
+-----------------------------------------------------------------------------------------
+
| NVIDIA-SMI 580.159.03             Driver Version: 580.159.03     CUDA Version: 13.0
   |
+-----------------------------------------+------------------------
+----------------------+
| GPU  Name                 Persistence-M | Bus-Id          Disp.A | Volatile Uncorr.
 ECC |
| Fan  Temp   Perf          Pwr:Usage/Cap |           Memory-Usage | GPU-Util  Compute
 M. |
|                                         |                        |               MIG
 M. |
|=========================================+========================
+======================|
|   0  NVIDIA L4                      On  |   00000000:31:00.0 Off |
 0 |
| N/A   41C    P8             13W /   72W |       0MiB /  23034MiB |      0%
 Default |
|                                         |                        |
 N/A |
+-----------------------------------------+------------------------
+----------------------+
```

出力には、GPU モデル、ドライバーバージョン、CUDA バージョン、利用可能なメモリが表示されます。この例では、Karpenter は 24 GB のメモリを持つ NVIDIA L4 GPU を持つ G6 インスタンスをプロビジョニングしました。GPU モデルとメモリは、Karpenter が選択するインスタンスタイプによって異なります。G5 インスタンスには NVIDIA A10G GPU (24 GB)、G6 インスタンスには NVIDIA L4 GPU (24 GB)、G6e インスタンスには NVIDIA L40S GPU (48 GB) があります。

Karpenter と Kubernetes スケジューラがどのように連携してノードをプロビジョニングし、ポッドを配置したかを理解するには、ポッドのライフサイクルイベントを確認します。

```
kubectl describe pod nvidia-smi
```

正常な出力:

```
Events:
  Type     Reason            Age   From                   Message
  ----     ------            ----  ----                   -------
  Warning  FailedScheduling  75s   default-scheduler      0/2 nodes are available: 2
 node(s) had untolerated taint(s).
  Normal   Nominated         74s   eks-auto-mode/compute  Pod should schedule on:
 nodeclaim/gpu-inf-z6q75
  Normal   Scheduled         35s   default-scheduler      Successfully assigned
 default/nvidia-smi to i-0eb897a8302551589
  Normal   Pulling           27s   kubelet                spec.containers{nvidia-smi}:
 Pulling image "public.ecr.aws/amazonlinux/amazonlinux:2023-minimal"
  Normal   Pulled            22s   kubelet                spec.containers{nvidia-smi}:
 Successfully pulled image "public.ecr.aws/amazonlinux/amazonlinux:2023-minimal" in
 5.625s (5.626s including waiting). Image size: 37440620 bytes.
  Normal   Created           22s   kubelet                spec.containers{nvidia-smi}:
 Container created
  Normal   Started           21s   kubelet                spec.containers{nvidia-smi}:
 Container started
```

これらのイベントは、Pod スケジューリングシーケンスを示しています。まず、GPU ノードが存在しないため、Pod はスケジュールに失敗します (`FailedScheduling`)。Karpenter が新しい NodeClaim をノミネートします (`Nominated`)。ノードが準備完了になるとスケジューラが Pod を割り当てます (`Scheduled`)。その後、コンテナイメージがプルされて起動します。EKS Auto Mode には、G、P、Trn インスタンスに SOCI (Seekable OCI) 並列プルがデフォルトでインストールおよび設定されており、セルフマネージド Karpenter パスは `FastImagePull` 機能ゲートを介してそれを明示的に設定します。

> **Note**
>
> セルフマネージド Karpenter クラスターでは、`Nominated` イベントは `eks-auto-mode/compute` の代わりに `karpenter/compute` を表示します。

NodeClaim は、Karpenter が特定のノードをプロビジョニングするために作成するリクエストです。インスタンスタイプ、キャパシティタイプ、AZ、ノードの準備が整っているかどうかが表示されます:

```
kubectl get nodeclaims
```

正常な出力:

```
NAME            TYPE        CAPACITY   ZONE         NODE                  READY   AGE
gpu-inf-z6q75   g6.xlarge   spot       us-east-2a   i-0eb897a8302551589   True    5m
```

インスタンスタイプと AZ は異なります。世代が 4 より大きい G ファミリーインスタンスが対象です。

> **Tip**
>
> ノードが表示されない場合は、容量不足エラーを確認します。
>
> ```
> kubectl get events | grep InsufficientCapacityError
> ```
>
> Karpenter は、使用できないサービスを 3 分間キャッシュします。NodePool で許可されたインスタンスタイプと AZ を拡張すると、ランディングキャパシティの可能性が高まります。

> **Note**
>
> Karpenter によって起動された Spot インスタンスは EC2 Spot Requests コンソールに表示されません。Karpenter は `type: instant` で EC2 `CreateFleet` API を使用します。インスタンスは、`spot` ライフサイクルとともに EC2 インスタンスコンソールに表示されます。

ステップ 4: NodePool にリザーブドキャパシティを追加する (オプション)

ステップ 2 の GPU NodePool は Spot インスタンスまたは On-Demand インスタンスを動的にプロビジョニングしますが、一部のユースケースでは保証されたキャパシティが必要です。オンデマンドキャパシティ予約 (ODCR) を作成して、GPU キャパシティが必要なときに使用可能になるようにできます。

Terraform では、1 つのコマンドで ODCR が作成され、予約をタグで参照するカスタム NodeClass が作成され、NodePool が更新されてキャパシティタイプ `reserved` として含まれます。Terraform は `nodepool=reserved-spot-ondemand` で ODCR にタグ付けし、NodeClass はそのタグでそれを選択します。

> **Warning**
>
> 次のコマンドは、ノードが実行されているかどうかにかかわらず、terraform destroy またはクリーンアップスクリプトを使用して破棄するまで、すぐに請求が開始され、請求が継続される ODCR を作成します。

デフォルト (g6e.4xlarge、1 個のインスタンス、最初のクラスター AZ) を使用します。

```
terraform apply -var 'nodepools={"reserved-spot-ondemand"={reservation={}}}'
```

インスタンスタイプ、カウント、AZ を選択します。

```
terraform apply -var 'nodepools={"reserved-spot-
ondemand"={reservation={instance_type="g6e.2xlarge",instance_count=1,az="us-
east-2a"}}}'
```

reservation オブジェクトは以下のフィールドをサポートします:

- instance_type – 予約する GPU インスタンスタイプ。デフォルト: g6e.4xlarge。
- instance_count – 予約するインスタンスの数。デフォルト: 1。
- az – 予約のアベイラビリティーゾーン。デフォルト: "" (最初のクラスター AZ を使用します)。

> **Important**
>
> spot-ondemand および reserved-spot-ondemand 戦略は相互に排他的です。nodepools 変数では、最大 1 個を有効にできます。以前にステップ 2 で spot-ondemand を使用した場合は、どちらも同じ gpu-inf NodePool を管理するため、reserved-spot-ondemand コマンドによって置き換えられます。

InsufficientInstanceCapacity エラーが発生した場合、指定された AZ で予約を満たすことはできません。Terraform オペレーション (Ctrl+C) をキャンセルしてから、別の az 値で再実行します:

```
terraform apply -var 'nodepools={"reserved-spot-
ondemand"={reservation={instance_type="g6e.4xlarge",az="us-east-2b"}}}'
```

適用後、Terraform は NodePool を更新して、キャパシティタイプの要件に `reserved`、`spot`、`on-demand` を含めます。Karpenter は、`reserved` を最もコスト効率の高いオプションとして扱い、最初に起動します。予約がいっぱいになると、スポットまたはオンデマンドにフォールバックします。

EKS Auto Mode パスで、Terraform は `capacityReservationSelectorTerms` を介してタグで ODCR を参照するカスタム `gpu-inf` NodeClass (バンドルされた `default` NodeClass が読み取り専用であるため) を作成します。セルフマネージド Karpenter パスで、Terraform は `gpu-inf` EC2NodeClass を `capacityReservationSelectorTerms` で再適用し、`reserved` を含めるように NodePool を更新します。

ODCR が作成されたことを確認します:

```
aws ec2 describe-capacity-reservations \
  --filters "Name=state,Values=active" "Name=tag:nodepool,Values=reserved-spot-
ondemand" \
  --query 'CapacityReservations[0].
{Id:CapacityReservationId,State:State,InstanceType:InstanceType,AvailableCount:AvailableInstanc
 \
  --output table \
  --region $(terraform output -raw region)
```

NodeClass が ODCR を参照していることを確認します:

EKS Auto Mode

```
kubectl get nodeclasses gpu-inf -o yaml | grep 'id: cr-.*'
```

正常な出力:

```
        id: cr-xxxxxxxxxxxxxxxxx
```

Self-managed Karpenter

```
kubectl get ec2nodeclasses gpu-inf -o yaml | grep 'id: cr-.*'
```

正常な出力:

```
        id: cr-xxxxxxxxxxxxxxxxx
```

NodePool の準備が整っていることを確認します:

```
kubectl get nodepools gpu-inf
```

正常な出力:

```
NAME      NODECLASS   NODES   READY   AGE
gpu-inf   gpu-inf     0       True    30s
```

リザーブドキャパシティが最初に使用され、Spot フォールバックがあることを確認する

変更を適用したら、Karpenter がリザーブドキャパシティを優先し、スポットまたはオンデマンドにフォールバックすることを確認します。ポッドごとに 1 GPU をリクエストする 2 レプリカデプロイをデプロイします。ODCR は 1 個のインスタンス (1 GPU) 用であるため、最初のポッドは Karpenter をトリガーしてリザーブドノードを起動します。2 番目のポッドはリザーブドノードに収まらず、Karpenter がスポットキャパシティまたはオンデマンドキャパシティから別のノードを起動するようにトリガーします。

```
cat << 'EOF' | kubectl apply -f -
apiVersion: apps/v1
kind: Deployment
metadata:
  name: gpu-overflow-test
  labels:
    guide: ai-eks-docs
spec:
  replicas: 2
  selector:
    matchLabels:
      app: gpu-overflow-test
  template:
    metadata:
      labels:
        app: gpu-overflow-test
        guide: ai-eks-docs
    spec:
      tolerations:
        - key: nvidia.com/gpu
          operator: Exists
          effect: NoSchedule
      containers:
        - name: nvidia-smi
```

```
          image: public.ecr.aws/amazonlinux/amazonlinux:2023-minimal
          command: ["sh", "-c", "nvidia-smi && sleep infinity"]
          resources:
            limits:
              nvidia.com/gpu: 1
EOF
```

実行して終了したステップ 3 の nvidia-smi テスト Pod とは異なり、このデプロイでは Pod の実行 (sleep infinity) が維持されるため、Pod は GPU を保持し、ノードが統合されるのを防ぎます。

異なるノードでスケジュールされたポッドを確認します。

```
kubectl get pods -l app=gpu-overflow-test -o wide
```

正常な出力:

```
NAME                                READY   STATUS    RESTARTS   AGE     IP
 NODE                   NOMINATED NODE   READINESS GATES
gpu-overflow-test-55d55ff5b9-dvg52   1/1     Running   0          4m42s   10.0.75.210
 i-08741a36089ff2088   <none>           <none>
gpu-overflow-test-55d55ff5b9-hw4m9   1/1     Running   0          4m43s   10.0.82.49
 i-0f50cdbacb2017202   <none>           <none>
```

NodeClaims でキャパシティタイプを確認します。

```
kubectl get nodeclaims
```

正常な出力:

```
NAME            TYPE          CAPACITY   ZONE         NODE                    READY   AGE
gpu-inf-vw99m   g6e.4xlarge   reserved   us-east-2c   i-0f50cdbacb2017202   True    6m
gpu-inf-s65s6   g6.xlarge     spot       us-east-2b   i-08741a36089ff2088   True
 5m59s
```

予約ノードが最初に起動され、その後に予約がいっぱいになるとスポットノードまたはオンデマンドノードが起動されます。

テストデプロイをクリーンアップする:

```
kubectl delete deployment gpu-overflow-test
```

モニタリング

Terraform は、ステップ 1 の `terraform apply` 中に完全なモニタリングスタックをプロビジョニング済みです。スタックには、Amazon Managed Service for Prometheus (AMP) ワークスペース、IAM ポリシー、EKS Pod Identity Associations for Prometheus リモート書き込みおよび Grafana クエリアクセス、kube-prometheus-stack Helm チャート (Prometheus、Grafana、kube-state-metrics、node-exporter)、GPU メトリクス用の NVIDIA DCGM Exporter が含まれます。

このセクションでは、デプロイされたモニタリングコンポーネントの検証について説明します。

モニタリングポッドの検証

すべてのモニタリングポッドの準備が整うまで待ちます。

```
kubectl wait --for=condition=Ready pod --all -n monitoring --timeout=300s
kubectl get pods -n monitoring
```

正常な出力:

```
NAME                                                     READY   STATUS    RESTARTS
 AGE
kube-prometheus-stack-grafana-7c58f54f77-rftrj           3/3     Running   0
 5m
kube-prometheus-stack-kube-state-metrics-d68dcbc84-5smxq 1/1     Running   0
 5m
kube-prometheus-stack-operator-5895df479f-ttm47          1/1     Running   0
 5m
kube-prometheus-stack-prometheus-node-exporter-t9q7s     1/1     Running   0
 5m
kube-prometheus-stack-prometheus-node-exporter-x6vfb     1/1     Running   0
 5m
prometheus-kube-prometheus-stack-prometheus-0            2/2     Running   0
 5m
```

Grafana にアクセスする

Grafana は、`var.my_cidr` で設定した CIDR に制限された、インターネット向け AWS Application Load Balancer (ALB) を介して公開されます。ロードバランサー URL を出力します (ALB がプロビジョニングするには 1〜2 分かかります):

```
echo "http://$(kubectl get ingress kube-prometheus-stack-grafana -n monitoring -o
 jsonpath='{.status.loadBalancer.ingress[0].hostname}')"
```

ブラウザで URL を開きます。以下のコマンドからユーザー名 admin とパスワードを使用してログインします。

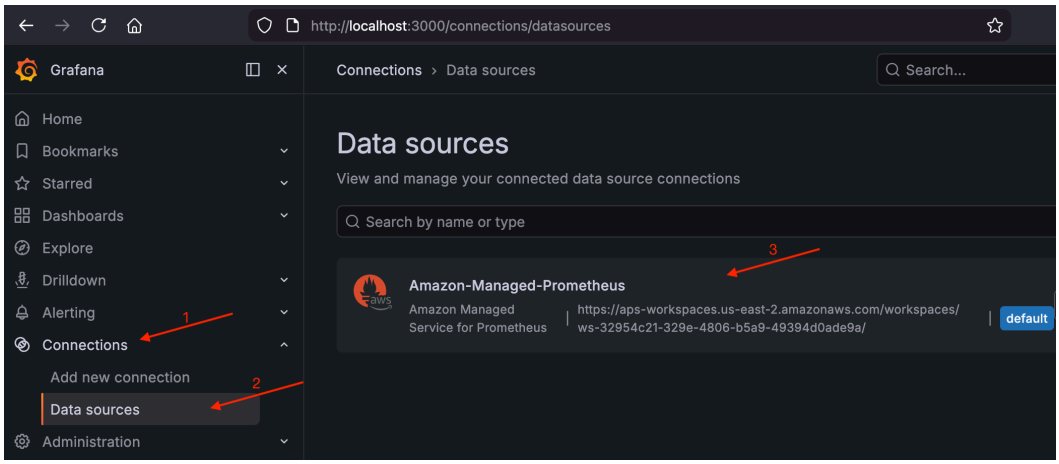
```
kubectl --namespace monitoring get secrets kube-prometheus-stack-grafana -o
 jsonpath="{.data.admin-password}" | base64 -d ; echo
```

メトリクスパイプラインを検証する

メトリクスパイプラインがエンドツーエンドで機能していることを確認するには:

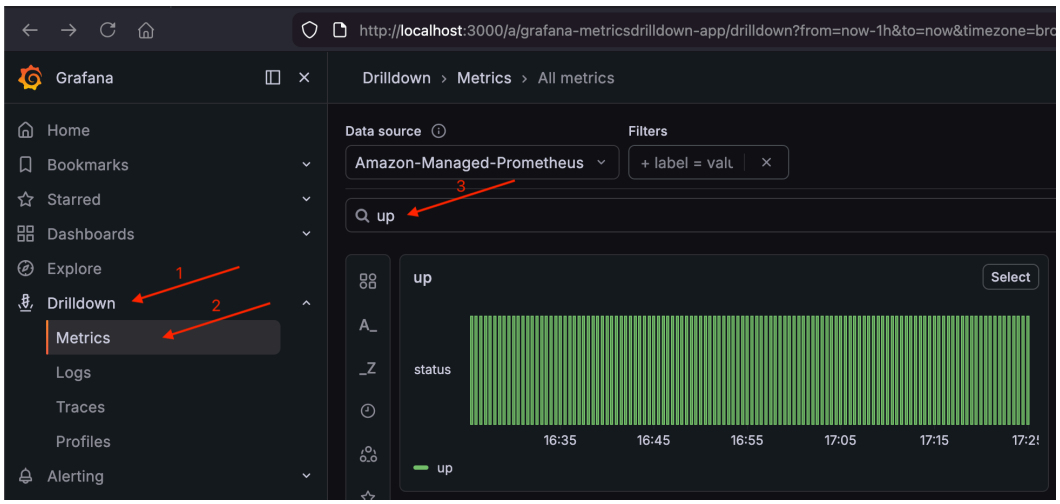
1. [接続] > [データソース] に移動し、Amazon-Managed-Prometheus がデフォルトのデータソースとしてリストされていることを確認します。

   Grafana で AMP データソースを検証する



2. [ドリルダウン] > [メトリクス] に移動し、up メトリクスを検索します。クラスターのスクレイプターゲットの結果が表示されます。

   Grafana で **up** メトリクスを検証する



up が結果を表示する場合、パイプライン (クラスター → Prometheus → AMP → Grafana) は機能しています。

DCGM GPU メトリクスを検証する

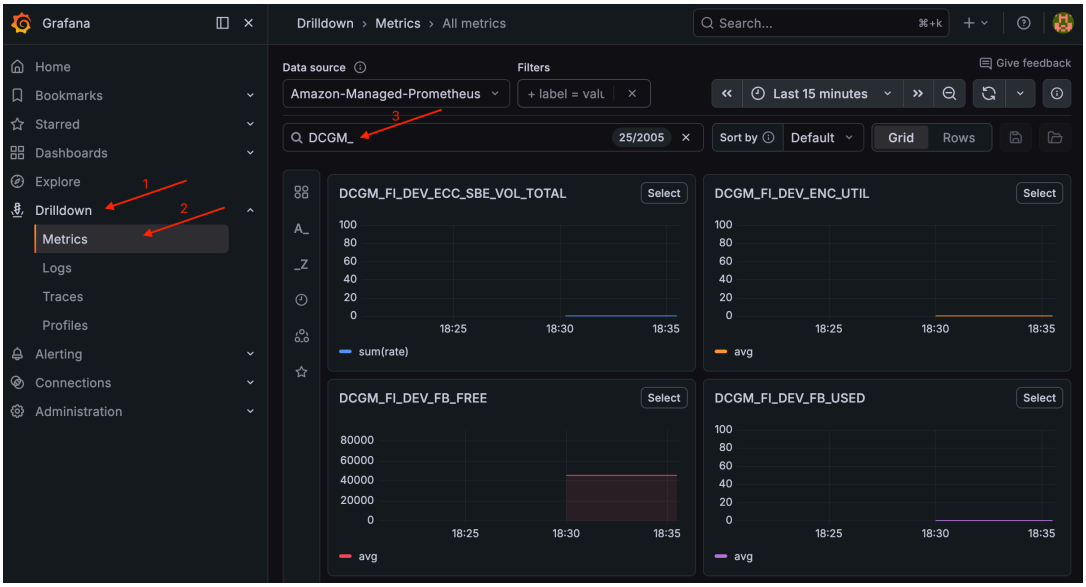
DCGM Exporter DaemonSet は GPU ノードで実行され、GPU 使用率、メモリ、温度、消費電力、NVLink 帯域幅、テンソルアクティビティメトリクスをレポートします。

DCGM Exporter DaemonSet を確認します。

```
kubectl get daemonset dcgm-exporter -n monitoring
```

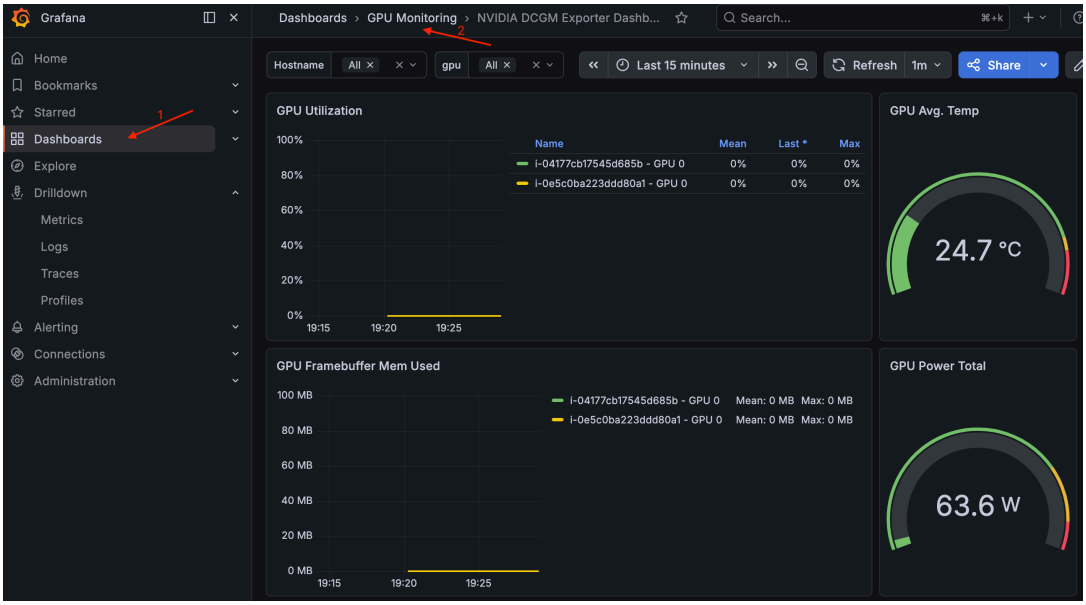
GPU ノードが実行されると (ステップ 2 またはステップ 4 から)、1 つ以上の準備完了の Pod が表示されます。DCGM メトリクスを検証するには、Grafana で [ドリルダウン] > [メトリクス] に移動し、`DCGM_` を検索します。

Grafana で DCGM メトリクスを検証する



ダッシュボードを表示するには、[ダッシュボード] > [GPU モニタリング] > [NVIDIA DCGM Exporter ダッシュボード] に移動します。

Grafana の NVIDIA DCGM Exporter ダッシュボード



モデルの重み S3 バケット

Terraform は、モデルの重みを保存するための Amazon S3 バケット、`default` 名前空間の `model-storage-sa` ServiceAccount、バケットを対象とする IAM ポリシー、およびそれらをリンクする EKS Pod Identity の関連付けを既に作成しています。`serviceAccountName: model-storage-sa` を設定したワークロード Pod は、バケットを読み書きできます。

バケットを検証する

Terraform 出力からバケット名を取得します:

```
MODEL_BUCKET=$(terraform output -raw model_bucket)
echo ${MODEL_BUCKET}
```

バケットが存在することを確認します:

```
aws s3api head-bucket --bucket ${MODEL_BUCKET}
```

正常な出力:

```
{
    "BucketArn": "arn:aws:s3:::ai-eks-docs-models-20250612abc1",
    "BucketRegion": "us-east-2",
    "AccessPointAlias": false
}
```

ポッドからの S3 アクセスを検証する

model-storage-sa ServiceAccount を使用して AWS CLI イメージで 1 回限りのポッドを実行し、EKS Pod Identity が正しく設定され、S3 アクセスが機能することを確認します。

```
cat << EOF | kubectl apply -f -
apiVersion: v1
kind: Pod
metadata:
  name: s3-test
  labels:
    guide: ai-eks-docs
spec:
  serviceAccountName: model-storage-sa
  containers:
    - name: aws-cli
      image: public.ecr.aws/aws-cli/aws-cli:2.27.0
      command:
        - sh
        - -c
        - |
          echo "=== Caller Identity ==="
          aws sts get-caller-identity
```

```
          echo ""
          echo "=== S3 Write Test ==="
          echo "pod identity works" | aws s3 cp - s3://${MODEL_BUCKET}/test.txt
          echo ""
          echo "=== S3 List Test ==="
          aws s3 ls s3://${MODEL_BUCKET}/
          echo ""
          echo "=== S3 Delete Test ==="
          aws s3 rm s3://${MODEL_BUCKET}/test.txt
  restartPolicy: Never
EOF
```

ポッドが完了するのを待ち、ログを確認します。

```
kubectl wait --for=jsonpath='{.status.phase}'=Succeeded pod/s3-test --timeout=300s
kubectl logs s3-test
```

正常な出力:

```
=== Caller Identity ===
{
    "UserId": "AROA...:eks-ai-eks-docs-model-s-...",
    "Account": "123456789012",
    "Arn": "arn:aws:sts::123456789012:assumed-role/ai-eks-docs-models-.../eks-ai-eks-
docs-..."
}

=== S3 Write Test ===
upload: - to s3://ai-eks-docs-models-20250612abc1/test.txt

=== S3 List Test ===
2026-07-15 12:00:00         19 test.txt

=== S3 Delete Test ===
delete: s3://ai-eks-docs-models-20250612abc1/test.txt
```

発信者 ID は、EKS Pod Identity を介してポッドがモデルストレージロールを引き受けたことを確認します。S3 コマンドは読み取りおよび書き込みアクセスを確認します。

テストポッドをクリーンアップします。

```
kubectl delete pod s3-test
```

次のステップ

クラスターの準備ができたら、「モデルのロードと提供」に進み、大規模な言語モデルをデプロイし、推論エンドポイントとやり取りできます。

クリーンアップ

> **Tip**
> このガイドの次のセクションに進む場合は、完全なクリーンアップをスキップします。完了時にのみ実行します。

テストワークロードを削除して、Pod が GPU ノードを保持しないようにします:

```
kubectl delete pod nvidia-smi --ignore-not-found
kubectl delete deployment gpu-overflow-test --ignore-not-found
```

クラスターを破棄せずにキャパシティ予約をキャンセルする

ODCR をリリースして Spot キャパシティと On-Demand キャパシティにフォールバックするだけの場合は、nodepools 変数を spot-ondemand 戦略に切り替えます:

```
terraform apply -var 'nodepools={"spot-ondemand"={}}'
```

これにより、NodePool のキャパシティタイプの要件から reserved がドロップされ、ODCR が破棄され、クラスター、モニタリングスタック、S3 バケットがそのまま残ります。

> **Important**
> 予約をキャンセルしても、既に実行中のインスタンスは終了しません。これらのインスタンスは、終了するまで標準の On-Demand レートで実行され続けます。予約がリリースされる前に予約済みノードがドレインされるよう、前述のとおり GPU ワークロードを最初に削除します。

クラスターとすべての Terraform マネージドリソースを破棄する

破棄する前に Karpenter マネージドノードをドレインして、処理中のノードのライフサイクルによって破棄がブロックされないようにします。ドレインを妨げる PodDisruptionBudgets を削除してから、NodeClaims を削除します。

```
kubectl delete pdb -A --all --ignore-not-found
kubectl delete nodeclaim --all --wait=true --timeout=900s
```

次に、EKS クラスター、VPC、モニタリングスタック、NodePools と NodeClasses、S3 モデルバケット、ODCR など、Terraform が作成したすべてのリソースを破棄します。

```
terraform destroy
```

> **Warning**
>
> モデルの重み S3 バケットは `force_destroy = true` で作成されるため、`terraform destroy` はアップロードしたモデルの重みとともにバケットを削除します。保持したいものは、最初に別の場所にコピーしてください。

> **Note**
>
> リポジトリには、前述のドレインおよび破棄手順を実行した後、クラスター名でタグ付けされた孤立した EBS ボリュームをスイープする `scripts/cleanup.sh` ヘルパーも含まれています。適用元の `terraform/<mode>/` ディレクトリ内から実行し、`--auto-approve` を渡して Terraform の確認プロンプトをスキップします。

予約が削除されたことを確認する

クラスターにアクティブなキャパシティ予約が残っていないことを確認します。

```
aws ec2 describe-capacity-reservations \
  --filters "Name=state,Values=active" "Name=tag:nodepool,Values=reserved-spot-
ondemand" \
  --query 'CapacityReservations[].CapacityReservationId' \
  --output text
```

結果が空の場合、予約はアクティブではなく、それ以上の料金はかかりません。

Amazon EKS で AI/ML 推論ワークロードを実行する

> **Tip**
>
> 今後開催予定の Amazon EKS AI/ML ワークショップに登録してください。

推論は、トレーニングされた AI モデルを実行して、入力データから予測または出力を生成するプロセスです。これには、テキストまたはコードを生成する大規模言語モデル (LLM)、画像を生成する拡散モデル、音声合成および文字起こし用の音声モデル、リアルタイム分析または生成用のビデオモデルの提供が含まれます。Amazon EKS は、NVIDIA GPU または AWS Trainium アクセラレーターを使用した推論ワークロードのデプロイ、動的スケーリング、より広範な Kubernetes や AWS のツールとサービスのランドスケープとの統合を提供する、スケーラブルでポータブルなソリューションです。

推論に Amazon EKS を使用する理由

Amazon EKS は、Kubernetes オーケストレーションと AWS インフラストラクチャを組み合わせて、大規模な推論の実行における主要な課題に対処します。

- 動的 GPU スケーリング — Karpenter は、ポッドリソースリクエストに基づいて適切なサイズの GPU インスタンスをオンデマンドでプロビジョニングし、アイドル時にゼロからスケーリングし、トラフィックの増加に応じてキャパシティを追加します。これにより、高価な GPU リソースの過剰プロビジョニングを回避できます。
- 高速コールドスタート — SOCI (Seekable OCI) 並列プルは、大きなコンテナイメージレイヤーを同時にダウンロードして解凍し、イメージのプル時間を数分から数秒に短縮します。Amazon S3 から GPU メモリへの直接モデルストリーミングと組み合わせると、ポッドは 2 分未満でサービスの提供を開始できます。
- GPU 障害の自動復旧 — EKS ノードモニタリングエージェントは GPU ハードウェア障害を検出し、ノードの自動交換をトリガーして、手動による介入なしでダウンタイムを最小限に抑えます。
- コストの最適化 — オンデマンドフォールバック、適切なサイズのインスタンス選択、ゼロへのスケーリング動作を備えたスポットインスタンスは、GPU コストを削減します。リザーブドキャパシティ (ODCR) は、定常状態のワークロードのコストをさらに削減できます。

- オープンスタンダードと移植性 — 推論ワークロードは、vLLM や SGLang などのオープンソースモデルサーバーを備えた標準の Kubernetes API (デプロイ、サービス、HPA) で実行され、環境間で移植性を提供します。
- 統合モニタリング — vLLM または SGLang と NVIDIA DCGM Exporter からの Prometheus メトリクスは、Grafana ダッシュボードを通じてリクエストのレイテンシー、トークンスループット、GPU 使用率、メモリ使用量を可視化します。

用語集

このセクションでは、以下の用語を使用します。

- 推論 — トレーニング済みモデルを実行して、入力データから出力 (テキスト、埋め込み、分類) を生成するプロセス。
- モデルサーバー — モデルをメモリにロードし、推論リクエストを受け取り、予測を返すコンテナ化されたサービス。例としては、vLLM、SGLang、Triton Inference Server、Text Generation Inference (TGI) などがあります。
- モデルの重み — モデルサーバーが GPU メモリにロードするファイル (通常は SafeTensors または GGUF 形式) として保存される、トレーニング済みモデルの学習済みパラメータ。
- アクセラレーター — 推論に必要なマトリックス操作を高速化する NVIDIA GPU や AWS Trainium/Inferentia チップなどの特殊なハードウェア。
- テンソル並列処理 — 同じノード上の複数の GPU にモデルを分割して、1 つの GPU のメモリを超えるモデルを提供します。
- KV キャッシュ — テキスト生成中に以前に計算されたキーと値のペアを保存し、新しいトークンごとに冗長計算を行うことを回避するメモリバッファ。

Amazon EKS での推論の仕組み

大まかに言うと、Amazon EKS に推論ワークロードをデプロイするには、以下のステップが必要です。

Step	説明
クラスターを設定する	GPU 対応ノード、モニタリング、ストレージを備えた EKS クラスターを作成します。EKS の AI/ML に関するドキュメントの「クラスターを設定する」を参照してください。

Step	説明
モデルの重みを保存する	モデルレジストリ (Hugging Face など) からモデルの重みをダウンロードし、コールドスタートまたはスケールアップイベント中にモデルを迅速にロードできるように Amazon S3 に保存します。
モデルサーバーをデプロイする	GPU ノードでモデルサーバー (vLLM など) を実行する Kubernetes デプロイを作成します。モデルサーバーは S3 から GPU メモリに重みをストリーミングし、OpenAI 互換 API を公開します。
推論エンドポイントを公開する	安定したネットワークエンドポイントを提供する Kubernetes サービスを作成します。テストにはポートフォワーディングを使用し、本番トラフィックには AWS Application Load Balancer (ALB) などのロードバランサーを使用します。
モニタリングとスケール	Prometheus メトリクス (リクエストレート、トークンスループット、レイテンシー、KV キャッシュ使用率) を使用してパフォーマンスをモニタリングし、自動スケーリングを設定します。

推論パフォーマンスメトリクス

推論パフォーマンスを理解するには、レイテンシーとスループット全体でメトリクスを追跡する必要があります。

レイテンシーメトリクス

- Time to First Token (TTFT) — リクエストの到着から最初に生成されたトークンまでの時間。インタラクティブアプリケーションには不可欠です。
- Time Per Output Token (TPOT) — 最初のトークンから後続の各トークンを生成する平均時間。
- End-to-end request latency — リクエストの送信からフルレスポンスの完了までの合計時間。

スループットメトリクス

- Requests per second — すべてのレプリカで 1 秒あたりに処理された推論リクエストの合計。
- Output tokens per second — 合計出力トークンを経過時間で割ったものとして測定されるトークン生成のレート。
- GPU utilization — 推論にアクティブに使用される GPU コンピューティングサイクルの割合。

- KV cache utilization — 割り当てられた KV キャッシュメモリが使用されている割合。サーバーがキャパシティ上限にどれだけ近いかを示します。

コールドスタート時間の短縮

大きな推論コンテナ (8 ~ 15 GB) とモデルの重み (10 ~ 100 GB 以上) では、ポッドの起動が遅くなる可能性があります。以下の手法は、コールドスタートの遅延を最小限に抑えます。

- SOCI 並列プル — コンテナイメージレイヤーをシーケンシャルではなく同時にダウンロードして解凍します。GPU インスタンスの EKS Auto Mode でデフォルトで有効になっています。
- S3 からのモデルストリーミング — Run:ai Model Streamer などのツールは、モデルの重みを Amazon S3 から GPU メモリに直接ストリーミングし、ローカルディスクをバイパスしてロード時間を数分から数秒に短縮します。
- Amazon ECR にイメージを保存する — リージョン ECR リポジトリから VPC エンドポイント経由でプルすることで、大規模なコンテナイメージのインターネットレイテンシーを回避できます。
- インスタンスストアキャッシュ — ローカル NVMe ディスクを持つ G ファミリーインスタンスはコンテナレイヤーとモデルアーティファクトをキャッシュできるため、同じノードで後続のプルを高速化できます。

デプロイするもの

「モデルのロードと提供」のチュートリアルでは、エンドツーエンドの推論アプリケーションをデプロイする方法について説明します。

1. モデル重みのダウンロード — Kubernetes ジョブは、Hugging Face から Ministral-3-8B-Instruct モデルをダウンロードし、S3 バケットにアップロードします。
2. vLLM のデプロイ — デプロイは Run:ai Model Streamer を使用して vLLM を実行し、S3 から GPU メモリに直接重みをストリーミングして、OpenAI 互換 API を提供します。
3. Grafana でモニタリング — ServiceMonitor は vLLM Prometheus メトリクスをモニタリングスタックに接続し、リアルタイムダッシュボードを提供します。
4. チャットフロントエンドのデプロイ — Open WebUI は、vLLM エンドポイントに接続されたブラウザベースのチャットインターフェイスを提供します。

このチュートリアルでは、the section called “クラスターを設定する” セクションのクラスターインフラストラクチャを使用し、EKS Auto Mode とセルフマネージド Karpenter パスの両方に対応しています。

Amazon EKS でのモデルのロードと提供

> **Tip**
>
> 今後開催予定の Amazon EKS AI/ML ワークショップに登録してください。

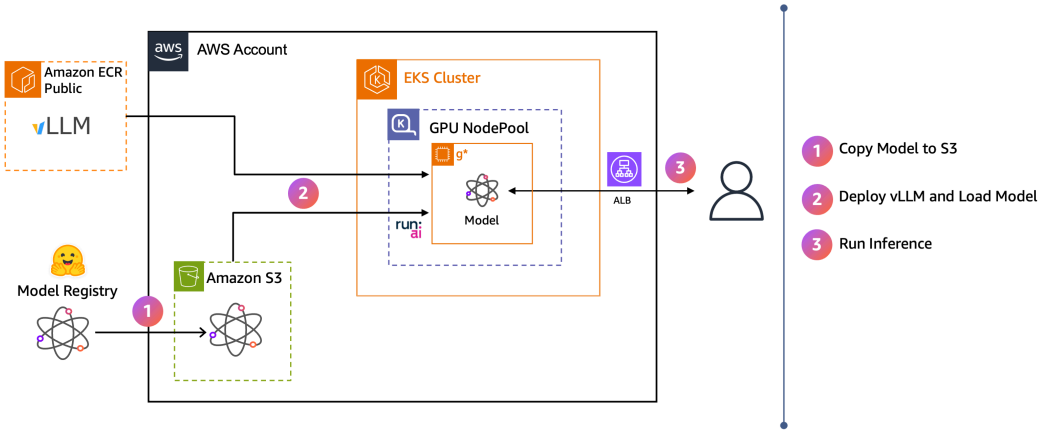
このセクションのステップでは、Amazon EKS に大規模言語モデル (LLM) をデプロイし、vLLM で提供して、推論エンドポイントとやり取りします。

このチュートリアルでは、以下のツールを使用します。

- vLLM — LLM サービスおよび GPU メモリ管理用に最適化された高スループットの推論エンジン。
- Run:ai Model Streamer — モデルの重みを Amazon S3 から GPU メモリに直接ストリーミングし、ロード時間を数分から数秒に短縮します。
- Open WebUI — vLLM の OpenAI 互換 API に接続するセルフホスト型チャットフロントエンド。

このセクションでは、Ministral-3-8B-Instruct-2512 モデルを使用しますが、vLLM がサポートする任意の AI モデルをデプロイできます。サポートされているモデルのリストについては、vLLM ドキュメントの「Supported models」を参照してください。

> **Important**
>
> the section called “クラスターを設定する” のセクションで作成したクラスターを使用します。このチュートリアルの手順は、EKS Auto Mode とセルフマネージド Karpenter の両方で機能します。



アーキテクチャ図はエンドツーエンドのフローを示しています。

1. モデルの重みは Hugging Face から Amazon S3 にダウンロードされます。
2. vLLM は Run:ai Model Streamer を使用して、モデルを S3 から GPU メモリに直接ストリーミングします。
3. ユーザーは vLLM エンドポイントに推論リクエストを送信します。

これらのステップを完了すると、チャットフロントエンドアプリケーションを介して Ministral モデルを操作するために使用できる vLLM 推論エンドポイントが利用可能になります。Amazon EKS 上のモデルロード時間の最適化の詳細については、「the section called “モデルのロードを高速化する”」を参照してください。

前提条件

「クラスターのセットアップセクション」のステップを完了します。

新しいターミナルを開いた場合は、「CLI によるクラスターのセットアップ」セクションで使用したクラスター名とリージョンを設定します。

```
export CLUSTER_NAME=ai-eks-docs
export AWS_REGION=us-east-2
```

「モデル重み S3 バケット」ステップで作成したモデル重みバケットを検索します。

```
MODEL_BUCKET=$(aws s3api list-buckets \
  --query "Buckets[?starts_with(Name, '${CLUSTER_NAME}-models-')].Name | [0]" \
  --output text)
echo "Model bucket: ${MODEL_BUCKET}"
```

ステップ 1: Hugging Face からモデルをダウンロードする

このステップでは、Hugging Face からモデルをダウンロードし、前提条件セクションで作成した S3 バケットにアップロードする Kubernetes ジョブをデプロイします。

モデルをダウンロードするには、以下のジョブマニフェストを適用します。

Exampleモデルダウンロードジョブマニフェスト

```
cat << EOF | kubectl apply -f -
apiVersion: batch/v1
kind: Job
metadata:
  name: model-download
  namespace: default
  labels:
    guide: ai-eks-docs
spec:
  backoffLimit: 10
  activeDeadlineSeconds: 3600
  ttlSecondsAfterFinished: 86400
  template:
    spec:
      restartPolicy: Never
      serviceAccountName: model-storage-sa
      containers:
      - name: downloader
        image: python:3.11-slim
        command: ["/bin/bash", "-c"]
        args:
        - |
          set -e
          pip install -q huggingface_hub boto3
          echo "Downloading Ministral-3-8B-Instruct-2512 from Hugging Face..."
          python3 -c "from huggingface_hub import snapshot_download;
 snapshot_download('mistralai/Ministral-3-8B-Instruct-2512', local_dir='/tmp/
mistral', allow_patterns=['*.json', '*.txt', '*.md', 'consolidated.safetensors'],
 ignore_patterns=['model-*.safetensors', 'model.safetensors.index.json'])"
          echo "Uploading to S3 bucket: \${MODEL_BUCKET}"
          python3 << 'PYTHON'
          import boto3
          import os
          from pathlib import Path
```

```
          s3 = boto3.client('s3')
          bucket = os.environ.get('MODEL_BUCKET')
          local_dir = Path("/tmp/mistral")

          for file_path in local_dir.rglob("*"):
              if file_path.is_file():
                  if '.cache' in file_path.parts:
                      continue
                  s3_key = f"Ministral-3-8B-Instruct-2512/
{file_path.relative_to(local_dir)}"
                  print(f"Uploading {file_path.name}...")
                  s3.upload_file(str(file_path), bucket, s3_key)
          print("Upload complete!")
          PYTHON
        env:
        - name: MODEL_BUCKET
          value: "${MODEL_BUCKET}"
        - name: HF_HUB_DISABLE_XET
          value: "1"
        resources:
          requests:
            memory: "2Gi"
            cpu: "1"
          limits:
            memory: "4Gi"
            cpu: "2"
EOF
```

ジョブが完了するまで待ちます。モデルの重み (consolidated.safetensors) は約 10.4 GB で、このステップには通常 3～5 分かかります。

```
kubectl wait --for=condition=complete job/model-download --timeout=600s
```

正常な出力:

```
job.batch/model-download condition met
```

モデルの重みが S3 にアップロードされたことを確認します。

```
aws s3 ls s3://$(kubectl get job model-download -o
 jsonpath='{.spec.template.spec.containers[0].env[?(@.name=="MODEL_BUCKET")].value}')/
Ministral-3-8B-Instruct-2512/ --recursive
```

正常な出力:

```
2026-05-18 10:29:53      20311 Ministral-3-8B-Instruct-2512/README.md
2026-05-18 10:29:53       2361 Ministral-3-8B-Instruct-2512/SYSTEM_PROMPT.txt
2026-05-18 10:29:53       1903 Ministral-3-8B-Instruct-2512/config.json
2026-05-18 10:29:54 10420633176 Ministral-3-8B-Instruct-2512/consolidated.safetensors
2026-05-18 10:29:53        131 Ministral-3-8B-Instruct-2512/generation_config.json
2026-05-18 10:29:53       1185 Ministral-3-8B-Instruct-2512/params.json
2026-05-18 10:29:53        976 Ministral-3-8B-Instruct-2512/processor_config.json
2026-05-18 10:29:53   16753777 Ministral-3-8B-Instruct-2512/tekken.json
2026-05-18 10:29:53   17077402 Ministral-3-8B-Instruct-2512/tokenizer.json
2026-05-18 10:29:53      21168 Ministral-3-8B-Instruct-2512/tokenizer_config.json
```

consolidated.safetensors ファイルには、モデルの重み (約 10.4 GB) が含まれています。残りのファイルは、vLLM がモデルを提供するために必要とする設定ファイルとトークナイザファイルです。

ステップ 2: 推論コンテナをデプロイする

このセクションでは、vLLM を Kubernetes デプロイとしてデプロイして、Amazon S3 にアップロードしたモデルを提供します。

このセクションでは、深層学習フレームワークがプリインストールされ、AWS インフラストラクチャのパフォーマンスに最適化された Docker イメージである AWS Deep Learning Containers (DLC) を使用します。DLC には、セキュリティパッチ、検証済みフレームワークバージョン、最適化された GPU ドライバー設定が含まれます。

このデプロイでは、SOCI サポートで vLLM 0.21.0 に以下の DLC を使用します: `public.ecr.aws/deep-learning-containers/vllm:0.21.0-gpu-py312-cu130-ubuntu22.04-ec2-v1.0-soci`。

イメージタグは、GPU をサポートする vLLM 0.21.0、Python 3.12、CUDA 13.0、Ubuntu 22.04 を示し、EC2 ベースのワークロード用に最適化され、コンテナの起動を高速化するための SOCI が有効化されています。

このマニフェストは、GPU ノードで vLLM を実行するデプロイを作成し、Run:ai Model Streamer を使用して S3 から GPU メモリに直接モデルをストリーミングします。マニフェストは、クラスター内アクセスのためにポート 8000 で vLLM エンドポイントを公開する ClusterIP サービスも作成します。

Amazon EKS 上のモデルロード時間の最適化の詳細については、「the section called “モデルのロードを高速化する”」を参照してください。次の例では、`--enforce-eager` を使用して、入門シナリ

オでロード時間を短縮します。the section called “--enforce-eager トレードオフ” で説明されているように、他の手法を使用してモデルのロード時間を短縮することをお勧めします。

マニフェストを適用します。

Example vLLM デプロイとサービス YAML

```
cat << EOF | kubectl apply -f -
apiVersion: apps/v1
kind: Deployment
metadata:
  name: vllm-inference-app
  labels:
    guide: ai-eks-docs
spec:
  replicas: 1
  selector:
    matchLabels:
      app: vllm-inference-app
  template:
    metadata:
      labels:
        app: vllm-inference-app
        guide: ai-eks-docs
    spec:
      serviceAccountName: model-storage-sa
      tolerations:
      - key: nvidia.com/gpu
        operator: Exists
        effect: NoSchedule
      nodeSelector:
        karpenter.sh/nodepool: gpu-inf
      containers:
      - name: vllm-inference
        image: public.ecr.aws/deep-learning-containers/vllm:0.21.0-gpu-py312-cu130-
ubuntu22.04-ec2-v1.0-soci
        ports:
        - containerPort: 8000
        args:
        - "--model=s3://${MODEL_BUCKET}/Ministral-3-8B-Instruct-2512/"
        - "--host=0.0.0.0"
        - "--port=8000"
        - "--tensor-parallel-size=1"
        - "--gpu-memory-utilization=0.9"
```

```
        - "--max-model-len=8192"
        - "--max-num-seqs=128"
        - "--load-format=runai_streamer"
        - "--enforce-eager"
        - "--tokenizer_mode=mistral"
        - "--config_format=mistral"
        - "--enable-auto-tool-choice"
        - "--tool-call-parser=mistral"
        resources:
          limits:
            nvidia.com/gpu: 1
          requests:
            memory: "40Gi"
            cpu: "8"
---
apiVersion: v1
kind: Service
metadata:
  name: vllm-inference-svc
  namespace: default
  labels:
    app: vllm-inference-app
spec:
  selector:
    app: vllm-inference-app
  ports:
  - name: http
    port: 8000
    targetPort: 8000
    protocol: TCP
EOF
```

vLLM ポッドが準備完了状態になっていることを確認してください。

```
kubectl get pod -l app=vllm-inference-app -w
```

正常な出力:

```
NAME                                  READY   STATUS    RESTARTS   AGE
vllm-inference-app-65df5fddc8-5kmjm   1/1     Running   0          86s
```

コンテナイメージがプルされ、vLLM がモデルの重みを S3 から GPU メモリにストリーミングするまでに約 2 分かかる場合があります。ポッドの READY 列に 1/1 が表示されるまで待ってから続行します。

EKS、SOCI、Run:ai Model Streamer の組み合わせにより、ポッドの高速起動が可能になります。各ステージの起動時間を確認するには、ポッドイベントを表示します。

```
kubectl describe pod -l app=vllm-inference-app | grep -A 20 "Events:"
```

正常な出力:

```
Events:
  Type     Reason            Age   From                   Message
  ----     ------            ----  ----                   -------
  Warning  FailedScheduling  86s   default-scheduler      0/2 nodes are available: 2
 node(s) had untolerated taint(s).
  Normal   Nominated         85s   eks-auto-mode/compute  Pod should schedule on:
 nodeclaim/gpu-inf-kqkq6
  Normal   Scheduled         55s   default-scheduler      Successfully assigned
 default/vllm-inference-app-d9d54586d-csmd7 to i-04f8792414384d2d3
  Normal   Pulling           52s   kubelet                Pulling image
 "public.ecr.aws/deep-learning-containers/vllm:0.21.0-gpu-py312-cu130-ubuntu22.04-ec2-
v1.0-soci"
  Normal   Pulled            4s    kubelet                Successfully pulled image
 "public.ecr.aws/deep-learning-containers/vllm:0.21.0-gpu-py312-cu130-ubuntu22.04-ec2-
v1.0-soci" in 48.376s (48.376s including waiting). Image size: 8802823997 bytes.
  Normal   Created           4s    kubelet                Created container vllm-
inference
  Normal   Started           4s    kubelet                Started container vllm-
inference
```

この例では、GPU ノードが 30 秒でプロビジョニングされ、8.8 GB コンテナイメージが SOCI を使用して約 48 秒でプルされました。高速イメージプルは、大規模な推論コンテナのコールドスタート時間を短縮します。これにより、アイドル状態の GPU キャパシティを過剰にプロビジョニングするのではなく、GPU ポッドを動的にスケールできます。

次に、vLLM ログをチェックしてモデルのロード時間を確認します。

```
kubectl logs $(kubectl get pod -l app=vllm-inference-app -o
 jsonpath='{.items[0].metadata.name}') | grep -i 'Model loading took'
```

正常な出力:

```
INFO 05-18 18:41:49 [gpu_model_runner.py:4959] Model loading took 9.81 GiB memory and
 5.023344 seconds
```

ログでは、Run:ai Model Streamer が 10.4 GB モデルの重みを S3 から直接 GPU メモリに約 5 秒でロードし、9.8 GiB の GPU メモリを消費したことを確認できます。

この例のイメージのダウンロード時間は、20 Gbps の持続的なネットワーク帯域幅を持つ g6e.4xlarge インスタンスを使用したものです。他のインスタンスタイプでは、利用可能なネットワーク帯域幅に応じて、イメージのプルとモデルのロード時間が異なります。

ステップ 3: 推論を実行する

vLLM デプロイが実行されている状態で、推論エンドポイントを検証し、チャットフロントエンドをデプロイしてモデルとやり取りします。

モデル検証テストを実行する

ポートフォワードを介して推論エンドポイントを公開します。

```
kubectl port-forward svc/vllm-inference-svc 8000:8000
```

新しいターミナルウィンドウを開き、推論コンテナが応答していることを確認します。

```
curl -sI -X GET http://localhost:8000/health
```

正常な出力:

```
HTTP/1.1 200 OK
date: Fri, 18 May 2026 00:39:23 GMT
server: uvicorn
content-length: 0
```

ステップ 4: vLLM をモニタリングする

vLLM は、リクエストレート、トークンスループット、エンドツーエンドのレイテンシー、GPU KV キャッシュ使用率など、Prometheus メトリクスをすぐに公開します。このセクションでは、「AI/ML ワークロード用の Amazon EKS クラスターを設定する」の手順で設定したモニタリングスタッ

クでこれらのメトリクスを使用し、事前プロビジョニングされた Grafana ダッシュボードで表示します。

> **Important**
>
> 続行する前に、「CLI を使用して AI/ML ワークロード用の Amazon EKS クラスターを設定する」の「モニタリング」サブセクションを完了する必要があります。このステップは、インストールされている kube-prometheus-stack と、値ファイルに既にプロビジョニングされている vLLM Grafana ダッシュボードによって異なります。

vLLM ServiceMonitor を適用する

ServiceMonitor は、vLLM メトリクスをスクレイプする場所を Prometheus に指示します。

```
cat << EOF | kubectl apply -f -
apiVersion: monitoring.coreos.com/v1
kind: ServiceMonitor
metadata:
  name: vllm-inference-app
  namespace: default
  labels:
    release: kube-prometheus-stack
spec:
  selector:
    matchLabels:
      app: vllm-inference-app
  endpoints:
  - port: http
    path: /metrics
    interval: 15s
EOF
```

ServiceMonitor が作成されたことを確認します。

```
kubectl get servicemonitor vllm-inference-app
```

正常な出力:

```
NAME                  AGE
vllm-inference-app    5s
```

推論トラフィックを生成する

ダッシュボードにメトリクスを入力するには、検証ステップでポートフォワードを介して既に公開している vLLM エンドポイントに対して推論トラフィックを生成します。

提供されているモデル名を見つけます。

```
MODEL_NAME=$(curl -s http://localhost:8000/v1/models | jq -r '.data[0].id')
echo "Using model: $MODEL_NAME"
```

50 件のチャット完了リクエストを並行して送信します。

```
for i in $(seq 1 50); do
  curl -s -X POST http://localhost:8000/v1/chat/completions \
    -H "Content-Type: application/json" \
    -d "{\"model\": \"$MODEL_NAME\", \"messages\": [{\"role\": \"user\", \"content\":
 \"Write a short poem about Kubernetes.\"}], \"max_tokens\": 128}" \
    > /dev/null &
done
wait
```

トラフィックが流れる間 (またはその直後)、vLLM `/metrics` エンドポイントから直接トークンスループットメトリクスを確認します。

```
curl -s http://localhost:8000/metrics | grep -E '^vllm:(prompt_tokens_total|
generation_tokens_total|avg_generation_throughput_toks_per_s|
avg_prompt_throughput_toks_per_s)' | head
```

`vllm:prompt_tokens_total` および `vllm:generation_tokens_total` メトリクスは、提供された入力トークンと出力トークンの単調増加カウンターです。`vllm:avg_prompt_throughput_toks_per_s` および `vllm:avg_generation_throughput_toks_per_s` メトリクスは、ローリング平均スループットゲージです。これらの同じメトリクスは、以下のサブセクションで開いた Grafana ダッシュボードの基盤となるものです。

vLLM Grafana ダッシュボードを表示する

「モニタリング」セクションの kube-prometheus-stack 値ファイルには、[GPU モニタリング] フォルダの下にコミュニティ vLLM ダッシュボード (gnetId 25263) が既にプロビジョニングされているため、追加のインポートは必要ありません。

「Grafana へのアクセス」セクションでセットアップしたロードバランサーから Grafana にアクセスします。URL を出力します。

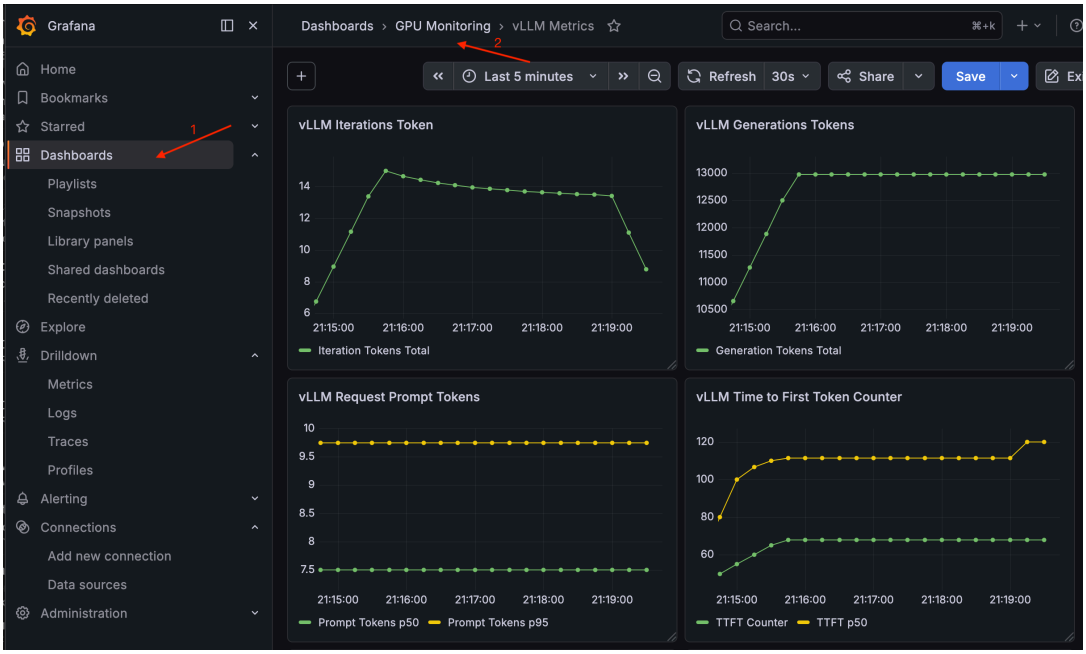
```
echo "http://$(kubectl get ingress kube-prometheus-stack-grafana -n monitoring -o
 jsonpath='{.status.loadBalancer.ingress[0].hostname}')"
```

ブラウザで URL を開き、ユーザー名 admin と以下のコマンドで取得したパスワードを使用してログインします:

```
kubectl --namespace monitoring get secrets kube-prometheus-stack-grafana -o
 jsonpath="{.data.admin-password}" | base64 -d ; echo
```

ダッシュボード > GPU モニタリング > vLLM メトリクスの順に移動します。

vLLM Grafana ダッシュボード



ダッシュボードには、vLLM 推論エンドポイントのリクエストレート、プロンプトと生成トークンのスループット、レイテンシーパーセンタイル、GPU KV キャッシュ使用率が表示されます。

ステップ 5: チャットアプリケーションをデプロイする

このステップでは、Open WebUI をチャットフロントエンドとしてデプロイし、モデルとやり取りします。Open WebUI は、オープンソースのセルフホスト型 AI インターフェイスであり、OpenAI 互換 API をサポートし、会話履歴とマークダウンレンダリングを備えたチャットインターフェイス

を提供します。vLLM は OpenAI 互換 API を公開するため、Open WebUI はバックエンドとして直接接続します。

Open WebUI アプリケーションをデプロイするには、以下のマニフェストを適用します。

Example WebUI デプロイとサービス YAML を開く

```
cat << 'EOF' | kubectl apply -f -
apiVersion: apps/v1
kind: Deployment
metadata:
  name: open-webui
  namespace: default
  labels:
    app: open-webui
    guide: ai-eks-docs
spec:
  replicas: 1
  selector:
    matchLabels:
      app: open-webui
  template:
    metadata:
      labels:
        app: open-webui
        guide: ai-eks-docs
    spec:
      containers:
      - name: open-webui
        image: ghcr.io/open-webui/open-webui:v0.9.2
        ports:
        - containerPort: 8080
        resources:
          requests:
            cpu: "500m"
            memory: "500Mi"
          limits:
            cpu: "1000m"
            memory: "2Gi"
        env:
        - name: OPENAI_API_BASE_URLS
          value: "http://vllm-inference-svc:8000/v1"
        - name: OPENAI_API_KEY
          value: "dummy"
```

```
        - name: WEBUI_AUTH
          value: "False"
        - name: ENABLE_OLLAMA_API
          value: "False"
        - name: ENABLE_EVALUATION_ARENA_MODELS
          value: "False"
        - name: RAG_EMBEDDING_ENGINE
          value: ""
        volumeMounts:
        - name: webui-volume
          mountPath: /app/backend/data
      volumes:
      - name: webui-volume
        emptyDir: {}
---
apiVersion: v1
kind: Service
metadata:
  name: open-webui
  namespace: default
  labels:
    app: open-webui
spec:
  type: ClusterIP
  selector:
    app: open-webui
  ports:
  - protocol: TCP
    port: 80
    targetPort: 8080
EOF
```

Open WebUI ポッドの準備ができるまで待ちます。

```
kubectl wait --for=condition=ready pod -l app=open-webui --timeout=300s
```

正常な出力:

```
pod/open-webui-6cbfc9867f-jf9w9 condition met
```

アプリケーションにアクセスするには、「ロードバランシングを設定する」で作成した alb IngressClass を使用して、インターネット向け AWS Application Load Balancer (ALB) を介して Open WebUI を公開します。

> ⚠ Open WebUI は認証なしでパブリックにアクセス可能です
>
> Open WebUI は認証が無効 (WEBUI_AUTH: "False") で実行されるため、ロードバランサーに到達するすべてのユーザーは GPU 推論エンドポイントでバックアップされた認証されていないチャットインターフェイスを取得し、GPU キャパシティを消費できます。自動スキャナーは、数分でパブリックロードバランサーを検索します。alb.ingress.kubernetes.io/inbound-cidrs 注釈を使用してアクセスを制限し、送信元 IP 許可リストを完全な保護ではなく最小限の保護として扱う必要があります。体制を強化するには、内部スキームを使用して TLS 証明書を追加し、Open WebUI 認証を有効にします。

パブリック IP アドレスを検索し、/32 CIDR として保存します:

```
export MY_CIDR="$(curl -s https://checkip.amazonaws.com)/32"
echo $MY_CIDR
```

結果は 203.0.113.4/32 のようになります。ネットワークがアドレスを動的に割り当てる場合、IP アドレスは変更される可能性があります。その場合、203.0.113.0/24 などのより広い範囲が必要になる場合があります。

以下の Ingress を適用して ALB を作成します。

```
cat << EOF | kubectl apply -f -
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: open-webui
  namespace: default
  labels:
    guide: ai-eks-docs
  annotations:
    alb.ingress.kubernetes.io/load-balancer-name: ai-eks-docs-chat-app
    alb.ingress.kubernetes.io/scheme: internet-facing
    alb.ingress.kubernetes.io/target-type: ip
```

```
    alb.ingress.kubernetes.io/inbound-cidrs: ${MY_CIDR}
    alb.ingress.kubernetes.io/healthcheck-path: /health
spec:
  ingressClassName: alb
  rules:
  - http:
      paths:
      - path: /
        pathType: Prefix
        backend:
          service:
            name: open-webui
            port:
              number: 80
EOF
```

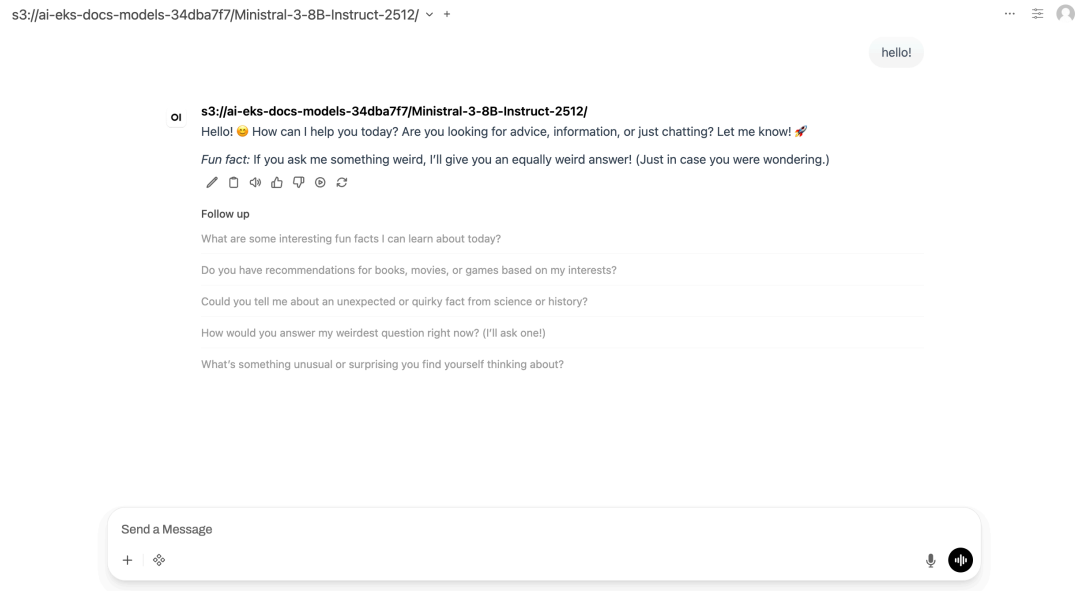
> **ヘルスチェックパス**
>
> ALB デフォルトのヘルスチェックマッチャーは 200 レスポンスを想定しているため、`healthcheck-path` 注釈はロードバランサーのヘルスチェックを Open WebUI の `/health` エンドポイントにポイントします。

ロードバランサーは非同期で作成され、1～2 分かかります。URL を出力します。

```
echo "http://$(kubectl get ingress open-webui -o
 jsonpath='{.status.loadBalancer.ingress[0].hostname}')"
```

ブラウザで URL を開きます。チャットインターフェイスが表示され、Ministral モデルを操作できます。

> **代替手段としてのポート転送**
>
> ポート転送 (`kubectl port-forward svc/open-webui 8080:80`) は、ロードバランサーをプロビジョニングせずにローカルテストを行うための選択肢として引き続き利用できます。



クリーンアップ

このセクションで作成したワークロードリソースを削除するには、Open WebUI アプリケーション、vLLM 推論サーバー、およびモデルダウンロードジョブを削除します。

```
kubectl delete ingress open-webui
kubectl delete deployment open-webui
kubectl delete service open-webui
kubectl delete deployment vllm-inference-app
kubectl delete service vllm-inference-svc
kubectl delete servicemonitor vllm-inference-app
kubectl delete job model-download
```

クラスター、NodePool、S3 バケットなどのインフラストラクチャリソースを削除する手順については、「クラスターセットアップのクリーンアップ」を参照してください。

Amazon EKS を使用して GPU で AI モデル推論を自動スケーリングする

> **Tip**
>
> 今後開催予定の Amazon EKS AI/ML ワークショップに登録してください。

このセクションでは、Amazon EKS を使用して GPU で AI 推論ワークロードを自動スケーリングする概念と手順について説明します。

前の Load & Serve Models セクションでデプロイした単一の vLLM モデルレプリカは、新しいリクエストが待機を開始する前に、限られた数の同時リクエストを処理できるため、レイテンシーが増加し、リクエストのタイムアウトが発生する可能性があります。トラフィックに対応するために、デプロイを水平方向に自動的にスケーリングできます。自動スケーリングは、需要の増加に応じてワークロードレプリカを追加し、需要の減少に応じてワークロードレプリカを削除するため、GPU キャパシティのオーバープロビジョニングを回避できます。

推論の自動スケーリングは 2 つの段階で行われます。

- ノードスケーリング – Kubernetes スケジューラに既存のノードに新しいポッドを配置するスペースがない場合、ポッドは `Pending` のままになります。Karpenter は、`Pending` ポッドに合わせて GPU ノードをプロビジョニングすることでポッドに応答し、ワークロードレプリカがスケールダウンするとノードを削除します。
- ポッドスケーリング – Horizontal Pod Autoscaler (HPA) は、設定されたメトリクスを監視します。そのメトリクスが設定されたしきい値を超えると、HPA はワークロードレプリカ数を増やし、新しい vLLM ポッドが作成されます。

ノードスケーリングステージは、ワークロードレプリカの実行に必要な GPU キャパシティを提供します。ポッドスケーリングステージは、選択したメトリクスをターゲットの近くに保持するようにワークロードレプリカの数を調整します。

Karpenter を使用した GPU ノードのプロビジョニング

Karpenter はノードのライフサイクルを管理し、ワークロードが必要なときにノードをプロビジョニングするか、必要でないときにノードを統合または削除します。Kubernetes スケジューラを置き換えるのではなく、それと連携して動作します。スケジューラはノードにポッドを配置します。Karpenter は、スケジューラが適合できないポッドのキャパシティをプロビジョニングし、ノードが `Ready` になるとスケジューラがそれらをバインドできるようにします。

Karpenter は、CPU ノードのフリートとは異なる方法で GPU ノードを処理します。

GPU はユニット全体でリクエストされます

> タイムスライシングや MIG などの GPU 共有手法を使用しない限り、ポッドは CPU やメモリのように一部ではなく、1 つ以上の GPU 全体を取得します。各 GPU には独自の専用メモリがあるため、同じノード上のレプリカは GPU メモリを共有しません。これがノードにどのようにマッピングされるかは、インスタンスタイプによって異なります。

- 単一 GPU インスタンス (g6.xlarge など) – ノードは 1 つのレプリカのみを保持します。レプリカ、GPU、ノードは 1 対 1 でスケールし、各レプリカは独自のインスタンスで分離されます。
- マルチ GPU インスタンス (4 つの GPU を持つ g6.12xlarge など) – デバイスプラグインは全 GPU 数をアドバタイズするため、Karpenter は複数のレプリカを GPU ごとに 1 つのノードにパックできます。
- マルチ GPU ポッド — ポッドは複数の GPU をリクエストできます。例えば、大きなモデルを 2 つの GPU 全体にシャードするように nvidia.com/gpu: 2 を設定すると、Karpenter はその数の GPU が空いているノードにポッドを配置します。

Karpenter が GPU の可用性とコストを最適化

Karpenter は、不十分な GPU キャパシティのランディングを費用対効果の高い実行に対してバランスを取ります。

- インスタンスの選択 柔軟な NodePool の場合、Karpenter は Pending ポッドに適した最も低コストのインスタンスタイプを選択します。起動がキャパシティ不足になると、他のインスタンスタイプとアベイラビリティーゾーンが自動的に再試行されるため、GPU キャパシティを確保できる可能性が高まります。
- 統合 – 需要が低下し、レプリカがスケールインすると、Karpenter は解放されたノードを再利用します。WhenEmptyOrUnderutilized は空のノードを削除し、レプリカをより少ないノードに再パックしますが、WhenEmpty はスケジュールされていないノードのみを再利用します。
- 進行中のポッドの保護 – 統合中にポッドが中断されないようにするには、karpenter.sh/do-not-disrupt 注釈を追加するか、PodDisruptionBudget を設定するか、consolidateAfter 設定を使用して統合を遅延させます。

Karpenter に GPU キャパシティが設定されると、ポッドスケーリングステージが引き継がれます。HPA は、設定したメトリクスに基づいてレプリカを追加および削除します。このセクションの残りの部分では、推論ワークロードのスケーリングメトリクスを選択する方法について説明します。

GPU と CPU の自動スケーリング

Kubernetes では、従来の CPU ベースのワークロードはすぐに自動スケーリングされますが、GPU ベースの推論は自動スケーリングされません。主な違いは 3 つあります。

自動スケーリングのメトリクス可用性

Kubernetes は CPU とメモリのメトリクスを HPA にすぐに提供します

各ノードの kubelet は CPU とメモリの使用状況を Kubernetes メトリクス API にレポートするため、HPA はこれらの値を直接読み取り、追加のコンポーネントなしでワークロードをスケーリングできます。

高速ワークロードには、追加のメトリクスのセットアップ、エクスポーター、アダプターが必要です

kubelet は GPU または推論メトリクスをレポートしません。GPU ベースの推論をスケールするには、DCGM エクスポーターからの vLLM アプリケーションメトリクスや GPU メトリクスなどの関連シグナルを発行するエクスポーターと、それらのメトリクスを HPA に公開するアダプターを追加する必要があります。

リソースの割り当てと共有

従来のワークロードは CPU とメモリを分数で共有します

Linux cgroups は CPU とメモリを分数リソースに分割し、複数のポッドが同じノードを共有できるようにします。ポッドは、100 ミリコアの CPU や 512 MiB のメモリなど、ノードのキャパシティの一部をリクエストできます。

高速ワークロードは、GPU 割り当てフレームワークを通じて GPU を割り当て、共有します。

GPU は CPU やメモリなどの cgroup を通じては共有されません。現在の最も一般的な Kubernetes セットアップでは、NVIDIA デバイスプラグインを使用して GPU 全体をポッド (`nvidia.com/gpu: 1` など) に割り当てます。Dynamic Resource Allocation (DRA) は、リソースクレームを通じて GPU を割り当てる新しい Kubernetes API であり、より柔軟なスケジューリングとリソース管理を提供します。デバイスプラグインと DRA の両方が GPU 共有手法をサポートできます。タイムスライシングを使用すると、複数のポッドが時間の経過とともにアクセスをインターリーブすることで GPU を共有できます。一方、マルチインスタンス GPU (MIG) は NVIDIA A100 や H100 などのサポートされている GPU をハードウェア分離された GPU インスタンスにパーティション化します。

スケーリングシグナル

従来のワークロードは通常、CPU とメモリの使用率に応じてスケールします。

> 使用率は、CPU およびメモリワークロードの標準的な自動スケーリングシグナルです。これは、ワークロードが既に負荷を受けた後にのみ上昇するため、遅行指標ですが、多くの従来のアプリケーションではスケーリングの決定を促進するのに十分です。

高速ワークロードは GPU 使用率ではなく、先行指標メトリクスに基づいてスケールする必要があります

> GPU 使用率 (DCGM_FI_DEV_GPU_UTIL) は、GPU がビジー状態かどうかを示すものであり、どの程度ビジー状態かを示すものではありません。vLLM などの推論サーバーは、さまざまな受信リクエストで 100% 近くに留まる可能性があるため、使用率は信頼性の低い自動スケーリングシグナルです。代わりに、リクエストキューの深度、最初のトークンまでの時間 (TTFT)、リクエストレイテンシーなど、需要がキャパシティに近づくにつれて増加する先行指標に基づいてスケールします。

上記の 3 つの違いのうち、スケーリングシグナルを慎重に検討してください。GPU 使用率は残りの推論キャパシティの信頼性の高いシグナルではないため、モデル推論サーバーへの実際の需要を追跡するシグナルを特定して収集する必要があります。

次のセクションでは、スケーリングメトリクスとして使用するシグナルと、それらがどのように連携するかについて説明します。

スケーリングメトリクスの選択

モデル推論レプリカをスケールするタイミングを伝える単一のメトリクスはないため、推論サーバーからのシグナルの組み合わせに基づいてスケールします。これらのシグナルは連携して機能し、それぞれが前のシグナルを基に構築されることで、単一のメトリクスよりも早く確実に需要を捉えます。

シグナル	メトリクス	スケーリングにおける役割
キュー	リクエストキューの深度 (処理を待機しているリクエストの数)	キューは需要がキャパシティを超えた瞬間に形成されるため、先行指標と最適なデフォルトトリガーです。
レイテンシー	リクエストレイテンシー (p95 レイテンシーまたは TTFT)	キューがまだ構築されていない場合でもレスポンスが遅いときにスケー

シグナル	メトリクス	スケーリングにおける役割
		リングをトリガーし、ユーザー側のレイテンシーをターゲット内に維持します。
GPU メモリ	KV キャッシュ使用率 (GPU のリクエストメモリの使用状況)	GPU メモリの不足から保護します。制限に近づくと、サーバーはリクエストの拒否またはキューイングを開始します。

これらを組み合わせて防御レイヤーを形成します。リクエストキューの深度は、需要がキャパシティを上回った瞬間に発生する主なトリガーであり、リクエストレイテンシーはキューが形成される前にレスポンスが遅いかどうかをチェックし、KV キャッシュ使用率は GPU メモリの不足を防ぐバックストップです。

すべてを実践する

以下のサブセクションでは、これらのシグナルを特定のオートスケーラーと連携させる方法を示します。

- スケーリングメトリクスのしきい値を検索する。単一の vLLM レプリカの負荷テストを行い、キャパシティの上限とスケールオンするキューの深度とレイテンシーのしきい値を検索します。ここから開始します。
- HPA と KEDA を使用してスケールする。スケーリングシグナルとしてリクエストキューの深度とエンドツーエンドレイテンシーを使用して、KEDA と Horizontal Pod Autoscaler で vLLM レプリカを自動スケーリングします。

AI 推論のスケーリングメトリクスのしきい値を特定する

> **Tip**
>
> 今後開催予定の Amazon EKS AI/ML ワークショップに登録してください。

このセクションでは、Amazon EKS で vLLM 推論サーバーの負荷テストを行い、いつ飽和状態になるかを特定する方法について説明します。負荷テスト結果を使用して、次の the section called “HPA

と KEDA" セクションで水平自動スケーリングのキューの深度とレイテンシーのしきい値を設定します。

このチュートリアルでは、以下のツールを使用します。

- k6 (Grafana k6) はオープンソースの負荷テストツールで、`grafana/k6` コンテナイメージとして実行され、制御されたリクエストレートで推論リクエストをレプリカに送信します。
- `g6e.4xlarge` インスタンス上の Ministral-3-8B-Instruct-2512 モデル。オープンソースモデルと GPU タイプに同じ方法を適用できます。

前提条件

このセクションは、ドキュメントの 2 つの前のセクションに基づいています。開始する前に、両方を完了してください:

- Cluster Setup は、GPU NodePool、kube-prometheus-stack Monitoring スタック (Prometheus および Grafana)、モデルの重み用の Amazon S3 バケットを備えた EKS クラスターを提供します。
- Load & Serve Models は vLLM 推論サーバーをデプロイし、`vllm-inference-app` ServiceMonitor を作成します。

開始する前に以下の各項目を確認し、チェックが失敗した場合は、まずリンクされたセクションを完了してください。

新しいターミナルを開いた場合は、先ほど使用したクラスター名とリージョンを設定します。

```
export CLUSTER_NAME=ai-eks-docs
export AWS_REGION=us-east-2
```

クラスターのセットアップステップで作成された Amazon S3 モデルバケットの名前をキャプチャします。負荷ジェネレーターはこの値を vLLM に渡します。

```
MODEL_BUCKET=$(aws s3api list-buckets \
  --query "Buckets[?starts_with(Name, '${CLUSTER_NAME}-models-')].Name | [0]" \
  --output text)
echo "Model bucket: ${MODEL_BUCKET}"
```

モニタリングスタックが実行されていることを確認する

Monitoring セットアップの kube-prometheus-stack は、`monitoring` 名前空間で実行されます。

```
kubectl get pods -n monitoring
```

Prometheus ポッドと Grafana ポッドは Running である必要があります:

```
NAME                                                     READY   STATUS    RESTARTS   AGE
kube-prometheus-stack-grafana-6f9c8b7d5c-2xk9p           3/3     Running   0          3h
kube-prometheus-stack-operator-7b8c9d6f4-q4m7n           1/1     Running   0          3h
prometheus-kube-prometheus-stack-prometheus-0            2/2     Running   0          3h
```

これらが見つからない場合は、続行する前に Monitoring セットアップを完了してください。

vLLM モデルが実行されていることを確認する

Load & Serve Models からの vllm-inference-app デプロイと vllm-inference-svc サービスは、default 名前空間で実行されます。

```
kubectl get deployment vllm-inference-app
```

デプロイは 1/1 Ready と報告されます:

```
NAME                READY   UP-TO-DATE   AVAILABLE   AGE
vllm-inference-app  1/1     1            1           3h
```

デプロイが欠落しているか、Ready ではない場合は、続行する前に Load & Serve Models を完了してください。

Grafana にアクセスする

負荷テストはプリロードされた vLLM ダッシュボードで監視するため、Access Grafana セクションでセットアップした Grafana ロードバランサーにアクセスできることを確認します。URL を出力します。

```
echo "http://$(kubectl get ingress kube-prometheus-stack-grafana -n monitoring -o
 jsonpath='{.status.loadBalancer.ingress[0].hostname}')"
```

ブラウザで URL を開き、ユーザー名 admin と以下のコマンドで取得したパスワードを使用してログインします:

```
kubectl --namespace monitoring get secrets kube-prometheus-stack-grafana -o
 jsonpath="{.data.admin-password}" | base64 -d ; echo
```

vLLM ServiceMonitor が存在することを確認する

[vLLM をモニタリングする] ステップで作成された `vllm-inference-app` ServiceMonitor は、`default` 名前空間で実行され、vLLM メトリクスをスクレイプする場所を Prometheus に伝えます。

```
kubectl get servicemonitor vllm-inference-app
```

正常な出力:

```
NAME                 AGE
vllm-inference-app   3h
```

見つからない場合は、[vLLM をモニタリングする] ステップを完了します。

ステップ 1: GPU をウォームアップする

モデルが GPU メモリにロードされ、レプリカが Ready とレポートしても、vLLM と GPU が最初の推論で初期化を完了するので、最初のリクエストにはそれ以降のリクエストよりも時間がかかります。vLLM は CUDA グラフをキャプチャし、GPU カーネルをコンパイルして自動チューニングし、KV キャッシュメモリプールを割り当てます。一方、GPU はクロック速度をアイドル状態からブースト範囲に上げます。

これらは 1 回限りのコストであるため、ウォームアップ前にしきい値を測定すると、定常状態のサービングの代わりにスタートアップ作業が記録されます。次のウォームアップでは、この初期化を完了するために 100 個のシーケンシャルリクエストを送信するので、次の負荷テストでは定常状態のキャパシティを測定します。

まず、ウォームアップスクリプト (`warmup.js`) を保持する ConfigMap を作成します。スクリプトは、先にキャプチャした `MODEL_BUCKET` 値からジョブが設定する `MODEL` 環境変数からターゲットモデルを読み取ります:

Example k6 ウォームアップ ConfigMap

```
cat << 'EOF' | kubectl apply -f -
apiVersion: v1
kind: ConfigMap
metadata:
  name: vllm-warmup-script
```

```
  namespace: default
data:
  warmup.js: |
    import http from 'k6/http';

    export default function () {
      const payload = JSON.stringify({
        model: __ENV.MODEL,
        prompt: 'Write a brief technical explanation about cloud computing concepts.',
        max_tokens: 100,
        temperature: 0.7,
      });
      const params = { headers: { 'Content-Type': 'application/json' }, timeout:
 '240s' };
      http.post('http://vllm-inference-svc:8000/v1/completions', payload, params);
    }
EOF
```

次に、ウォームアップジョブを実行します。単一のクライアントが 100 件のシーケンシャルリクエストを送信します:

Exampleウォームアップジョブ

```
cat << EOF | kubectl apply -f -
apiVersion: batch/v1
kind: Job
metadata:
  name: vllm-warmup
  namespace: default
spec:
  backoffLimit: 0
  template:
    spec:
      restartPolicy: Never
      containers:
        - name: k6
          image: grafana/k6:latest
          env:
            - name: MODEL
              value: "s3://${MODEL_BUCKET}/Ministral-3-8B-Instruct-2512/"
          args: ["run", "--vus", "1", "--iterations", "100", "/scripts/warmup.js"]
          volumeMounts:
            - name: script
```

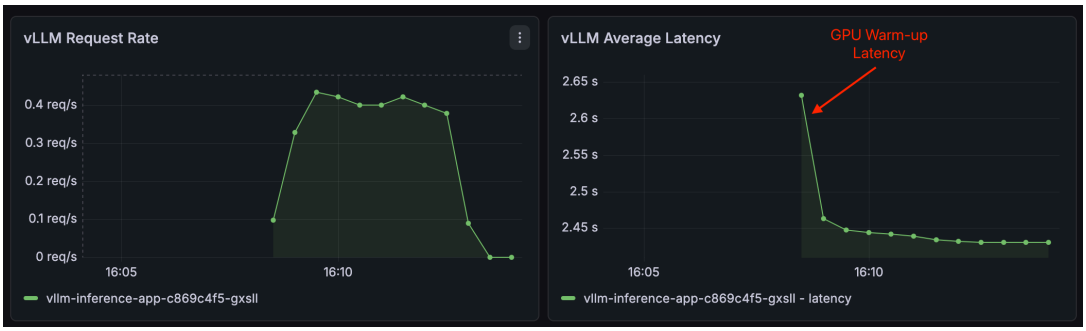
```
              mountPath: /scripts
      volumes:
        - name: script
          configMap:
            name: vllm-warmup-script
EOF
```

ウォームアップが完了するまで待ちます (数分かかります):

```
kubectl wait --for=condition=complete job/vllm-warmup --timeout=600s
```

Grafana を開き (ロードバランサーのホスト名を開きます。詳細については、「Grafana にアクセスする」を参照してください)、ダッシュボード > GPU モニタリング > パフォーマンステスト - vLLM 負荷分析の順に移動します。vLLM リクエストレートパネルには、ウォームアップトラフィックのバーストが表示されます。vLLM 平均レイテンシーパネルは、最初のリクエストが遅いことを示しています。これは 1 回限りの GPU ウォームアップコストです。

レイテンシーに対する GPU ウォームアップ効果を示す vLLM ダッシュボード



次に、ウォームアップジョブを削除します。

```
kubectl delete job vllm-warmup
```

ステップ 2: リクエストレートを上げて負荷を送信する

このステップでは、1 秒あたり 10、20、30、40、50、60、および 70 リクエストとレートを段階的に上げながら、単一の vLLM レプリカにリクエストを送信する負荷テストを実行します。これにより、ユーザー需要の増加がシミュレートされます。テストでは、constant-arrival-rate モデルで k6 を使用します。ここで、TARGET_RPS 環境変数はターゲットリクエストレートを設定します。メトリクスが安定するように、各レートは 60 秒間実行され、次のレート増加の前に 30 秒間一時停止します。

このテストの目的は、飽和ポイントを特定することです。飽和ポイントは、単一のレプリカが受信リクエストに追いつくことができなくなるリクエストレートです。キュー深度 (vllm:num_requests_waiting) とエンドツーエンドレイテンシー (vllm:e2e_request_latency_seconds) をモニタリングします。キュー深度とレイテンシーの増加は、レプリカが飽和に近づいていることを示します。

まず、k6 負荷テストスクリプト (script.js) を保持する ConfigMap を作成します。このスクリプトは、各ポッドが前にキャプチャした MODEL_BUCKET 値から設定する MODEL 環境変数からターゲットモデルを読み取ります。

Example k6 負荷テスト ConfigMap

```
cat << 'EOF' | kubectl apply -f -
apiVersion: v1
kind: ConfigMap
metadata:
  name: vllm-loadtest-script
  namespace: default
data:
  script.js: |
    import http from 'k6/http';

    // Dictionary of 10-letter words for variation (prevents prompt caching)
    const words = [
      'revolution', 'technology', 'javascript', 'kubernetes', 'basketball',
      'strawberry', 'watermelon', 'blackberry', 'throughout', 'background',
      'playground', 'understand', 'everything', 'protection', 'complexity'
    ];

    export const options = {
      noConnectionReuse: true,
      scenarios: {
        constant_request_rate: {
          executor: 'constant-arrival-rate',
          rate: __ENV.TARGET_RPS || 10,  // Overridden by TARGET_RPS env var
          timeUnit: '1s',
          duration: __ENV.DURATION || '60s',  // Overridden by DURATION env var
          preAllocatedVUs: 50,
          maxVUs: 500,
        },
      },
    };
```

```
    export default function () {
      const randomWord = words[Math.floor(Math.random() * words.length)];
      const payload = JSON.stringify({
        model: __ENV.MODEL,
        prompt: `Starting with the word "${randomWord}", write a brief technical
 explanation about cloud computing concepts. Cover containerization, orchestration, and
 scalability in about 80 words.`,
        max_tokens: 100,
        temperature: 0.7,
      });
      const params = { headers: { 'Content-Type': 'application/json' }, timeout:
 '240s' };
      http.post('http://vllm-inference-svc:8000/v1/completions', payload, params);
    }
EOF
```

次に、各リクエストレートで負荷テストを実行します。次のループは、レートごとに個別のジョブを送信します。

Example負荷テストループ

```
for RPS in 10 20 30 40 50 60 70; do
  echo "=== Testing at ${RPS} req/s ==="
  cat << EOF | kubectl apply -f -
apiVersion: batch/v1
kind: Job
metadata:
  name: vllm-loadtest-${RPS}
  namespace: default
  labels:
    app: vllm-loadtest
spec:
  backoffLimit: 0
  template:
    metadata:
      labels:
        app: vllm-loadtest
    spec:
      restartPolicy: Never
      containers:
        - name: k6
          image: grafana/k6:latest
          env:
```

```
            - name: K6_PROMETHEUS_RW_SERVER_URL
              value: "http://kube-prometheus-stack-
prometheus.monitoring.svc.cluster.local:9090/api/v1/write"
            - name: MODEL
              value: "s3://${MODEL_BUCKET}/Ministral-3-8B-Instruct-2512/"
            - name: TARGET_RPS
              value: "${RPS}"
            - name: DURATION
              value: "60s"
          args: ["run", "-o", "experimental-prometheus-rw", "/scripts/script.js"]
          volumeMounts:
            - name: script
              mountPath: /scripts
      volumes:
        - name: script
          configMap:
            name: vllm-loadtest-script
EOF
  kubectl wait --for=condition=complete job/vllm-loadtest-${RPS} --timeout=180s
  sleep 30
done
```

テストの実行中に Grafana のメトリクスを監視します (ステップ 3 を参照)。各レートは独自のジョブ (`vllm-loadtest-10` から `vllm-loadtest-70`) を作成するため、`kubectl get jobs` で任意の実行を検査できます。完了したら、すべての負荷テストジョブを削除します。

```
kubectl delete jobs -l app=vllm-loadtest
```

ステップ 3: 負荷中のメトリクスを測定する

負荷の実行中に、自動スケーリングを駆動する 2 つのメトリクスを確認します。

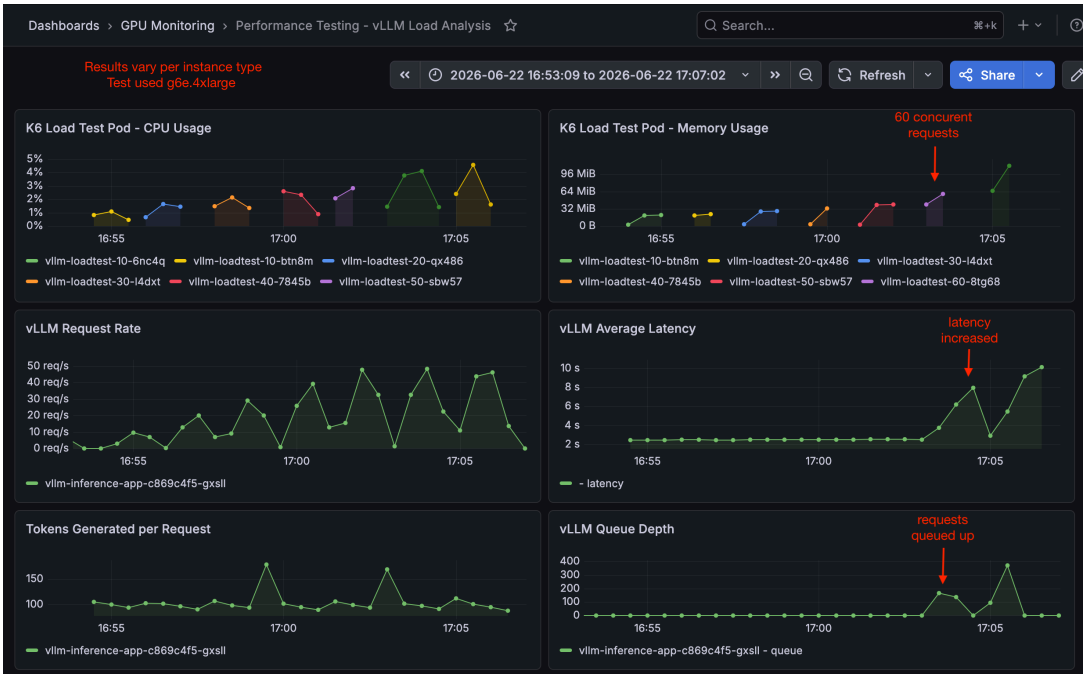
- キュー深度 (`vllm:num_requests_waiting`) – 処理を待っているリクエストの数。
- p95 エンドツーエンドレイテンシー (`vllm:e2e_request_latency_seconds`) – 95 パーセンタイルの応答時間。

Grafana でメトリクスを表示する

Grafana を開き (ロードバランサーのホスト名を開いて `admin` としてログインします。詳細については、「Grafana にアクセスする」を参照してください)、ダッシュボード > GPU モニタリング > パフォーマンステスト - vLLM 負荷分析の順に移動します。このダッシュボードは Amazon Managed

Service for Prometheus から読み取るので、メトリクスはバッチでリモートで書き込まれます。そのため、パネルはライブ状態から最大 1 分遅延する場合があります。

パフォーマンステスト – 負荷テスト中の vLLM 負荷分析ダッシュボード



負荷テスト結果に基づいて、キュー深度は同時リクエスト数が 50 件までほぼゼロに留まりました。その後、同時リクエスト数が 60 件になると急増し、キューに入れられたリクエストが約 150～350 件でピークに達しました。同時に、p95 エンドツーエンドのレイテンシーは、約 2.5 秒の安定したベースラインから 6～10 秒に上昇しました。この動作は、持続的な過負荷の兆候を示しており、レイテンシーが 10 秒に近づくレベルにキュー深度が達する前に自動スケーリングを開始する必要があることを示唆しています。実用的な開始点は、キュー深度が 30～60 秒で 25 リクエストを超えたときにスケーリングをトリガーすることです。セカンダリシグナルとして、p95 エンドツーエンドのレイテンシーが同じ間隔で 5 秒を超えたときにスケーリングをトリガーすることもできます。これにより、キュー深度が蓄積される前にレイテンシーが増加するワークロードパターンを考慮できます。

ステップ 4: スケールするメトリクスとしきい値を選択する

飽和ポイントで観察されたメトリクスを使用して、次のセクションで自動スケーリングの設定に使用するしきい値を決定します。

- キュー深度 (`vllm:num_requests_waiting`) – レプリカが飽和状態になったときに処理を待機しているリクエストの数。一時的なスパイクがスケーリングをトリガーしないように、キューが最初に正の値を維持するポイントを超えるしきい値を設定します。このセクションの負荷テストでは、キュー深度は同時リクエスト数が 50 件までほぼゼロを維持し、60 件の同時リクエストで約 150～

350 件のキューリクエストに急増しました。実用的な開始点は、キュー深度が 30 ~ 60 秒で 25 リクエストを超えたときにスケーリングをトリガーすることです。

- レイテンシー (`vllm:e2e_request_latency_seconds`) – レプリカがキャパシティーの制限に近づいたときのエンドツーエンドの応答時間。しきい値をレイテンシー SLO より低く設定することで、ユーザーに表示されるレイテンシーが許容できなくなる前に自動スケーラーがスケールアウトします。このセクションの負荷テストでは、p95 エンドツーエンドのレイテンシーは約 2.5 秒で安定し、飽和時に 6 ~ 10 秒に増加しました。セカンダリシグナルとして、実用的な開始点は、p95 エンドツーエンドのレイテンシーが 30 ~ 60 秒で 5 秒を超えたときにスケーリングをトリガーすることです。これにより、キュー深度が蓄積される前にレイテンシーが上昇するワークロードパターンや、キューイングなしにレイテンシーが増加するワークロードパターンを考慮できます。
- スケールダウン – 需要が低下した後に自動スケーラーがレプリカを削除するレート。スケールアップするより遅くスケールダウンします。この例では、キュー深度が `0` で、p95 エンドツーエンドのレイテンシーが 5 分で 3 秒未満である場合にのみスケールダウンします。これにより、自動スケーラーはトラフィックの短い一時停止中にレプリカを削除できなくなり、振動を回避できます。

これらの値は、次のセクションで自動スケーリングを設定するための開始点となります。独自のモデル、GPU タイプ、リクエストパターンに対してこのプロセスを繰り返して、デプロイに適したしきい値を決定します。

HPA と KEDA による AI 推論の自動スケーリング

> **Tip**
>
> 今後開催予定の Amazon EKS AI/ML ワークショップに登録してください。

このセクションでは、「スケーリングメトリクスのしきい値を検索する」セクションで確立されたしきい値に基づいて `vllm-inference-app` デプロイをスケールする方法を示します。

このチュートリアルでは、以下のツールを使用します。

- KEDA (Kubernetes Event-Driven Autoscaler) は、Prometheus に直接クエリを実行し、標準の Kubernetes Horizontal Pod Autoscaler (HPA) を作成する CNCF プロジェクトです。vLLM キューの深度 (プライマリ需要シグナル) と p95 エンドツーエンドレイテンシー (SLO ガードレール) でデプロイをスケーリングし、いずれかのしきい値を超えたときにレプリカを追加します。

  KEDA には、GPU 推論の自動スケーリングに関する 3 つの主な利点があります。

- ゼロへのスケーリング – KEDA は、アイドル時にデプロイをゼロレプリカにスケールダウンし、需要が戻ったときにスケールアップできるため、GPU コストを削減できます。
- アクティベーションしきい値 – KEDA は、スケーリングをアクティブ化するしきい値を、レプリカ数を決定するために使用されるターゲットしきい値から分離します。これにより、一時的なスパイクを無視し、少数のリクエストで GPU を起動しないようにできます。
- セットアップの簡素化 – GPU 推論ワークロードでは、KEDA は多くの場合、最もシンプルな自動スケーリングメカニズムであり、Prometheus 統合が組み込まれており、管理すべき個別のメトリクスアダプターはありません。

前提条件

このサブセクションは、「スケーリングメトリクスのしきい値を検索する」から続きます。Load & Serve Models と Monitoring のセットアップが完了していることを確認します。これにより、`vllm-inference-app` デプロイとサービスが `default` 名前空間で実行され、kube-prometheus-stack が vLLM メトリクスをスクレイピングします。

新しいターミナルを開いた場合は、先ほど使用したクラスター名とリージョンを設定します。

```
export CLUSTER_NAME=ai-eks-docs
export AWS_REGION=us-east-2
```

ステップ 1: KEDA をインストールする

Helm で KEDA を独自の名前空間にインストールします:

```
helm repo add kedacore https://kedacore.github.io/charts
helm repo update

helm install keda kedacore/keda --namespace keda --create-namespace
```

KEDA ポッドが実行中であることを確認します:

```
kubectl get pods -n keda
```

正常な出力:

```
NAME                                      READY   STATUS    RESTARTS   AGE
keda-admission-webhooks-7d4d6c6f9-xxxxx   1/1     Running   0          40s
keda-operator-6b8f9c5d7c-xxxxx            1/1     Running   0          40s
```

```
keda-operator-metrics-apiserver-xxxxx     1/1     Running   0          40s
```

ステップ 2: KEDA ScaledObject を作成する

ScaledObject は、スケーリングするデプロイ、レプリカの境界、およびトリガーを KEDA に伝えます。この設定では、プライマリ需要シグナルとしてキュー深度、および SLO ガードレールとして p95 エンドツーエンドレイテンシーの 2 つのトリガーを使用します。metricType: AverageValue の設定により、KEDA は各メトリクスをレプリカ間で分割するため、しきい値はポッドごとに解釈されます。

このチュートリアルでは、「スケーリングメトリクスしきい値を検索する」セクションのしきい値の例を使用します。平均キュー深度がポッドあたり 25 個の待機リクエストを超えた場合、または p95 エンドツーエンドレイテンシーが5 秒を超えた場合にスケールアップします。独自のモデル、GPU、およびリクエストシェイプで測定した値を置き換えます。

```
cat << EOF | kubectl apply -f -
apiVersion: keda.sh/v1alpha1
kind: ScaledObject
metadata:
  name: vllm-inference-app
spec:
  scaleTargetRef:
    apiVersion: apps/v1
    kind: Deployment
    name: vllm-inference-app
  minReplicaCount: 1
  maxReplicaCount: 5
  advanced:
    horizontalPodAutoscalerConfig:
      name: vllm-inference-app
      behavior:
        scaleUp:
          stabilizationWindowSeconds: 30
          policies:
            - type: Pods
              value: 2
              periodSeconds: 60
          selectPolicy: Max
        scaleDown:
          stabilizationWindowSeconds: 300
          policies:
            - type: Pods
```

```
              value: 1
              periodSeconds: 120
          selectPolicy: Min
  triggers:
    # Primary: queue depth (demand exceeds capacity)
    - type: prometheus
      metricType: AverageValue
      metadata:
        serverAddress: http://kube-prometheus-stack-prometheus.monitoring.svc:9090
        query: sum(vllm:num_requests_waiting) or vector(0)
        threshold: "25"
        activationThreshold: "1"
    # SLO guardrail: p95 end-to-end request latency (seconds)
    - type: prometheus
      metricType: AverageValue
      metadata:
        serverAddress: http://kube-prometheus-stack-prometheus.monitoring.svc:9090
        query: histogram_quantile(0.95,
 sum(rate(vllm:e2e_request_latency_seconds_bucket[1m])) by (le)) or vector(0)
        threshold: "5"
EOF
```

キー設定:

- キュー深度トリガー – query はすべての vLLM ポッド全体で待機中のリクエストの合計数を返します。or vector(0) 句は、待機中のリクエストがない場合に空の結果の代わりに 0 を返します。これにより、トリガーが非アクティブになるのを防ぎます。threshold: "25" は、「スケーリングメトリクスしきい値を検索する」で確立したポッドごとのキュー深度ターゲットであり、activationThreshold: "1" は少なくとも 1 つのリクエストが待機するまでデプロイを minReplicaCount に維持します。そのため、アイドルシグナルでスケーリングされません。
- レイテンシートリガー – query は、1 分間のウィンドウにおける vLLM レイテンシーヒストグラムから計算された p95 エンドツーエンドのリクエストレイテンシーを秒単位で返します。threshold: "5" は、「スケーリングメトリクスしきい値を検索する」からのレイテンシー SLO ガードレールです。p95 レイテンシーがそれを超えると、キューがまだ構築されていない場合でも KEDA はスケールアップします。
- トリガー評価 – KEDA は、どのトリガーが最もレプリカを要求するかを満たすようにスケールするため、どちらのシグナルに違反してもスケールアップがトリガーされます。
- スケールアップとスケールダウンの動作 – 待機または上昇レイテンシーはユーザーが既に待っていることを意味するため、スケールアップは応答的です (30 秒ウィンドウ、1 分あたり最大 2 ポッド)。GPU ポッドの起動が遅いため、スケールダウンは控えめです (5 分ウィンドウ、2 分ごとに 1

ポッド)。このアプローチにより、後で再び必要になる可能性のあるキャパシティを削除することを回避できます。

ScaledObject の準備が整っていることを確認します:

```
kubectl get scaledobject vllm-inference-app
```

正常な出力:

```
NAME                SCALETARGETKIND      SCALETARGETNAME      MIN   MAX   READY
 ACTIVE
vllm-inference-app   apps/v1.Deployment   vllm-inference-app   1     5     True
 False
```

READY: True は、トリガーが有効で、KEDA がデプロイを管理していることを意味します。ACTIVE: False は、トリガーしきい値がまだ超過していないため、トラフィックが流れていないときに予期されます。

ScaledObject を作成すると、KEDA は同じ名前のバッキング Horizontal Pod Autoscaler (HPA) を自動的に作成および管理します。この HPA は自分で作成または編集しないでください。それが存在することを確認します:

```
kubectl get hpa vllm-inference-app
```

正常な出力:

```
NAME                REFERENCE                          TARGETS                MINPODS
 MAXPODS   REPLICAS
vllm-inference-app   Deployment/vllm-inference-app   0/25 (avg), 0/5 (avg)  1         5
         1
```

TARGETS 列には、しきい値 (キュー深度と p95 レイテンシー) に対する各メトリクスの現在の値が表示されます。MINPODS と MAXPODS は ScaledObject の minReplicaCount と maxReplicaCount から取得され、REPLICAS は vLLM ポッドの現在の数です。

ステップ 3: 負荷を生成する

負荷テストでは、「スケーリングメトリクスしきい値を検索する」で作成した vllm-loadtest-script ConfigMap が再利用されます。まだそれが存在することを確認します:

```
kubectl get configmap vllm-loadtest-script
```

見つからない場合は、「スケーリングメトリクスしきい値を検索する」から再作成します。

スケールアップを促進するために、1 秒あたり 60 リクエストで 10 分間、単一の負荷テストを実行します。ジョブは、前に設定した MODEL_BUCKET 環境変数からモデルバケットを読み取り、負荷テストスクリプトが読み取る TARGET_RPS 変数と DURATION 変数を通じてレートと期間を渡します。

```
cat << EOF | kubectl apply -f -
apiVersion: batch/v1
kind: Job
metadata:
  name: vllm-loadtest-scaleup
  labels:
    app: vllm-loadtest-scaleup
spec:
  backoffLimit: 0
  template:
    spec:
      restartPolicy: Never
      containers:
        - name: k6
          image: grafana/k6:latest
          env:
            - name: MODEL
              value: "s3://${MODEL_BUCKET}/Ministral-3-8B-Instruct-2512/"
            - name: TARGET_RPS
              value: "60"
            - name: DURATION
              value: "10m"
          args: ["run", "/scripts/script.js"]
          volumeMounts:
            - name: script
              mountPath: /scripts
      volumes:
        - name: script
          configMap:
            name: vllm-loadtest-script
EOF
```

ジョブは 10 分間実行されます。次のステップに移動して、テストの実行中にデプロイがスケールアウトするのを監視します。

ステップ 4: デプロイのスケールアップを監視する

自動スケーラーが前のステップの負荷に応答するのを監視します。次の各コマンドを別々のターミナルで実行します。

HPA を監視します。TARGETS 列が設定されたターゲットを上回り、REPLICAS が増加します。

```
kubectl get hpa vllm-inference-app -w
```

vLLM ポッドを監視します。モデルが GPU メモリにロードされると、新しいレプリカが表示され、Ready に到達します。

```
kubectl get pods -l app=vllm-inference-app -w
```

ScaledObject がアクティブであることを確認します。トリガーのしきい値を超えると、ACTIVE は True に変わります:

```
kubectl get scaledobject vllm-inference-app
```

予想される動作:

- 約 30 秒以内にキュー深度がしきい値を上回り (または p95 エンドツーエンドのレイテンシーが 5 秒を超える)、HPA TARGETS 列が設定されたターゲットを上回ります。
- KEDA はデプロイをスケールアップし、利用可能な GPU ノードがない場合は Karpenter が新しい GPU ノードをプロビジョニングします。モデルが GPU メモリにロードされると、新しいレプリカは Ready に到達します。
- トラフィックがレプリカ全体に分散すると、キューがドレインし、レイテンシーが回復し、両方のメトリクスがしきい値を下回ります。
- 負荷が停止すると、デプロイは 5 分間のクールダウンを通じてより高いレプリカ数を保持し、一度に 1 つのポッドを minReplicaCount に向けてスケールダウンします。

負荷テストが完了したら、負荷テストジョブを削除します。vllm-loadtest-script ConfigMap はしきい値セクションと共有されるため、そのままにします:

```
kubectl delete jobs -l app=vllm-loadtest-scaleup --ignore-not-found
```

クリーンアップ

> **Note**
>
> 自動スケーリングの使用を継続する場合は、このクリーンアップをスキップします。完了時にのみ実行します。

このセクションで作成した自動スケーリングリソースを削除するには、`ScaledObject` を削除し、KEDA をアンインストールします。

```
kubectl delete scaledobject vllm-inference-app --ignore-not-found
helm uninstall keda -n keda
kubectl delete namespace keda
```

`ScaledObject` を削除すると、KEDA が作成した HPA も削除されます。vLLM デプロイは静的レプリカ数に戻ります。

vLLM 推論サーバーと関連するワークロードリソースを削除するには、「Load & Serve Models」を参照してください。クラスター、NodePool、S3 バケットなどのインフラストラクチャリソースを削除する手順については、「クラスターセットアップのクリーンアップ」を参照してください。

Amazon EKS 上でモデルのロードを高速化する

Amazon EKS に大規模言語モデル (LLM) をデプロイすると、モデルのロード時間は、Pod が推論リクエストの処理を開始する速度に直接影響します。これは、新しい Pod またはノードがトラフィックを処理する前にモデルをロードする必要があるスケールアップイベントに特に当てはまります。モデルのスタートアップには、チューニングとコンパイルアーティファクトのキャッシュで改善できる 2 つのフェーズがあります。

- 重みのロード – Run:ai Model Streamer を使用して Amazon S3 から GPU メモリにモデルの重みファイルをストリーミングします。
- torch.compile – モデルの計算グラフを最適化された融合 CUDA/Triton カーネルにコンパイルします。このコンパイルは最初のスタートアップ時に実行され、モデルサイズに応じて起動時間に大きな影響を与える可能性があります。

このトピックでは、Run:ai Model Streamer のパフォーマンスと torch.compile キャッシュを最適化して、モデルのロードプロセスの両方のフェーズを短縮する方法について説明します。推論のために

vLLM を Amazon EKS にデプロイする詳細な手順については、「the section called “モデルのロードと提供”」を参照してください。

EKS Auto Mode 上で S3 ネットワークパスを最適化する

プライベートサブネットに GPU ノードがある EKS Auto Mode 上で実行している場合は、S3 のゲートウェイ VPC エンドポイントを使用して、ノードと S3 間のネットワークパスを最適化することをお勧めします。Gateway VPC エンドポイントでは、S3 へのトラフィックは AWS ネットワーク上にとどまり、NAT Gateway を完全にバイパスするため、共有帯域幅の上限や GB あたりの NAT データ処理料金はありません。Gateway VPC エンドポイントがない場合、トラフィックが NAT Gateway を通過すると、複数のノードが同時にモデルをプルする際に、スケールアップ中に NAT Gateway が共有ボトルネックになります。

EKS Auto Mode は通常、プライベートサブネットにノードを配置するため、S3 へのトラフィックはデフォルトで NAT ゲートウェイを通過します。NAT Gateway は、送信先ごとに最大 100 Gbps の帯域幅と 55,000 の同時接続を提供します。ただし、その帯域幅はプライベートサブネット内のすべてのノード全体で共有されます。スケールアップイベント中、一度に完全なモデルをダウンロードしている複数のノードが同じ NAT Gateway 帯域幅に対して競合するため、すべてのノードでモデルの重みのロードが遅くなる可能性があります。

Run:ai Model Streamer のパフォーマンスチューニング

vLLM や SGLang などの推論エンジンは、推論の起動中に重みをロードするための代替メカニズムとして Run:ai Model Streamer を使用します。

デフォルトでは、Run:ai Model Streamer は、S3 からモデルの重みファイルをダウンロードするときに保守的な同時実行設定を使用します。ダウンロードの同時実行とチャンクサイズを増やすと、より多くのデータを並行してダウンロードすることで、モデルの重みのロード時間を短縮できます。

最適な同時実行を計算する

最適な同時実行値を次のように計算します:

```
concurrency = ceil(total_model_size_gb / chunk_size_gb)
```

使用する `concurrency` 値をモデルサイズとチャンクサイズに基づいて置き換えます。いくつかの例については、次の表を参照してください。例えば、67 GB のモデルと 4 GB のチャンクサイズの場合: `ceil(67 / 4) = 17`。

モデルサイズ	チャンクサイズ	同時実行
10 GB	4 GB	3
67 GB	4 GB	17
140 GB	4 GB	35

設定を適用します

推論コンテナ仕様に次の引数と環境変数を追加します:

- `--tensor-parallel-size` – テンソル並列 (TP) の度数は通常、モデルサイズに基づいてモデルを GPU メモリに収めるために必要な GPU の最小数です。例えば、`p5.48xlarge` インスタンスタイプの 67 GB モデルには少なくとも 2 GPU が必要なため、2 に設定します。
- `concurrency` および `distributed` (`--model-loader-extra-config` 内) – モデルの計算値に `concurrency` を設定します。テンソル並列処理 (TP > 1) を使用する場合にのみ `distributed` を `true` に設定します。有効にすると、各テンソル並列ランクは、ランク 0 がすべての重みをロードして他のランクにブロードキャストするのではなく、Amazon S3 から直接独自の重みシャードをストリーミングします。これにより、マルチ GPU デプロイのロードパフォーマンスが大幅に向上します。TP=1 の場合はメリットがないため、設定解除 (または `false`) のままにしておきます。このオプションには、vLLM V1 アーキテクチャが必要です。これは V0 パスを強制する `--enforce-eager` と互換性がありません。両方を使用すると、エラーになるか、分散されていないロードにサイレントにフォールバックします。
- `RUNAI_STREAMER_CHUNK_BYTESIZE` – 4 GB のチャンクサイズ。この値は、ベンチマーク全体で一貫して最高のパフォーマンスを示しています。チャンクが大きいほど、S3 リクエストの数が減少し、高帯域幅インスタンスでのスループットが向上します。
- `RUNAI_STREAMER_S3_REQUEST_TIMEOUT_MS` – ミリ秒単位のリクエストごとのタイムアウト。遅い S3 レスポンスでより高速に再試行できます。
- `RUNAI_STREAMER_S3_LOW_SPEED_LIMIT` – リクエストが遅いと判断されて再試行される前の、1 秒あたりの最小転送速度 (バイト単位)。

```
containers:
- name: vllm-inference
  image: vllm/vllm-openai:v0.21.0
  command:
```

```
    - python3
    - -m
    - vllm.entrypoints.openai.api_server
  args:
  # ... your existing args ...
  - --model=s3://<MODEL_PATH>
  - --tensor-parallel-size=2
  - --load-format=runai_streamer
  - --model-loader-extra-config={"concurrency":17,"distributed":true}
  env:
  - name: RUNAI_STREAMER_CHUNK_BYTESIZE
    value: "4294967296"
  - name: RUNAI_STREAMER_S3_REQUEST_TIMEOUT_MS
    value: "3000"
  - name: RUNAI_STREAMER_S3_LOW_SPEED_LIMIT
    value: "1048576"
```

torch.compile コールドスタートの最適化

推論処理プロセスの `torch.compile` ステップは、モデルの計算グラフ (数学演算のシーケンス) をトレースし、最適化されたフュージョン CUDA/Triton カーネルにコンパイルします。モデルの重みはコンパイルされません。モデルの重みを変換するオペレーションのみがコンパイルされます。

推論エンジンは torch.compile を使用します。これは、モデルアーキテクチャごとにカスタムカーネルエンジニアリングを行わなくても、スループットが自動的に大幅に向上するからです。

- カーネルフュージョン – 複数の小さなオペレーション (残差加算、レイヤーノーム、アクティベーション) が 1 つのカーネルにフュージョンされ、GPU メモリのラウンドトリップが削減されます。
- カーネルの起動数の低減 – トランスフォーマーレイヤーは、約 15～30 個の個別の CUDA カーネルから、約 10 個のフュージョンカーネルに削減され、起動あたりの CPU オーバーヘッドが削減されます。
- ホットパスから Python を削除 – フォワードパス全体が C++ 実行プランになり、オペレーション間の Python インタープリタのオーバーヘッドがなくなります。
- CUDA グラフの互換性の向上 – コンパイルされた静的グラフは、ほぼゼロの CPU オーバーヘッドでキャプチャおよび再生されます。
- 自動最適化 – 任意のモデルアーキテクチャで機能します。スループットは、eager モードよりも 5～30% 向上します。

次の表は、torch.compile を使用する一般的な推論サービングエンジンを示しています。

エンジン	torch.compile の使用
vLLM	デフォルトで有効 (V1 アーキテクチャ)
SGLang	`--enable-torch-compile` 経由のオプション
TensorRT-LLM	新しいパスがレガシーエンジンビルドとともにそれをサポートする

torch.compile コールドスタートの問題

torch.compile のトレードオフは、最初の推論ポッドがリクエストを処理する前にコンパイルする必要があることです。このコンパイルは、モデルサイズに応じて数分かかる場合があります。コンパイルされたアーティファクトは小さい (60 GB モデルでは約 15 MB) ですが、コールドスタート時間が大幅に増加します。アーティファクトは小さく、特定の設定に対して決定的であるため、キャッシュして再利用することで、後続の Pod とノードの起動時のコールドスタートペナルティを排除できます。

アーティファクトは以下で構成されます:

- 生成された Triton カーネルソースファイル
- コンパイルされたカーネルバイナリ (`.cubin`)
- カーネル呼び出しをシーケンスするグラフ構造

--enforce-eager トレードオフ

vLLM はデフォルトで `torch.compile` と CUDA グラフキャプチャを有効にします。`--enforce-eager` フラグは両方をオフにし、モデルを eager モードで実行します。各オペレーションは Python インタープリタを介してすぐに実行されます。eager モードはコンパイルとグラフキャプチャの両方をスキップするため、the section called “モデルのロードと提供” のベースデプロイを含むいくつかのクイックスタートガイドでは、`--enforce-eager` を使用して Pod をより迅速に開始します。

`--enforce-eager` は、デバッグ、メモリ制約のあるデプロイ、またはクリーンにコンパイルされないモデルアーキテクチャに対する有効な選択肢であり、これらの理由により本番環境で使用できます。`torch.compile` コールドスタートのペナルティは軽減できますが、本番環境でランタイムパ

フォーマンスを維持するために、このページの以降のセクションで説明されているような他のアプローチをお勧めします。

本番環境にデプロイする前に、`--enforce-eager` のスタートアップ時間とランタイムパフォーマンスのトレードオフを理解します。

側面	**`--enforce-eager`** を使用 (eager モード)	デフォルト (torch.compile + CUDA グラフ)
スタートアップ時間	速い – コンパイルまたはグラフキャプチャステップなし	遅いコールドスタート (the section called “同じノードで torch.compile アーティファクトをキャッシュする” と the section called “新しいノード上で torch.compile キャッシュを事前ウォーミングする” によって解決された問題)
定常状態スループット	ベースライン	カーネルフュージョンから最大 5～30% 高い
トークンあたりのレイテンシー (スモールバッチ)	高い CPU 起動オーバーヘッド	はるかに低い – CUDA グラフはカーネルの起動を 1 ユニットとして再生します
GPU メモリ	低く、予測可能	高い – グラフキャプチャはバッファプールを事前に割り当てます
デバッグ可能性	オペレーションごとのクリーンなスタックトレース	生成されたカーネル内に エラーが表示される

同じノードで torch.compile アーティファクトをキャッシュする

この手法は、vLLM (デフォルトで有効) や SGLang (`--enable-torch-compile` 経由で有効) など、`torch.compile` をサポートする推論エンジンを使用する場合に適用されます。`torch.compile` がアクティブなとき、つまり `--enforce-eager` が設定されていないときにのみ機能します。

> **Important**
>
> torch.compile が無効になっている場合、この改善は効果がありません。vLLM では、--enforce-eager フラグは torch.compile を完全に無効にするため、アーティファクトはコンパイルまたはキャッシュされません。the section called “モデルのロードと提供” で --enforce-eager を使用したベースデプロイに従った場合、vLLM はキャッシュディレクトリを作成しますが、それに書き込むことはありません。この手法を適用する前に、--enforce-eager を削除します。

推論エンジンが最初のスタートアップ時にモデルの計算グラフをコンパイルすると、生成された最適化されたカーネルをノードのローカルストレージにキャッシュできます。同じノード上の後続の Pod はキャッシュされたアーティファクトを再利用し、コンパイルステップを完全にスキップするため、スタートアップ時間を大幅に短縮できます。

キャッシュ環境変数の追加

次の環境変数を推論コンテナ仕様に追加して、torch.compile キャッシュと Triton キャッシュを永続的ホストパスに送信します。hostPath のルート Amazon Elastic Block Store (Amazon EBS) ボリュームではなく、ノードのローカル NVMe インスタンスストアを使用することをお勧めします。例については、次のセクションを参照してください。

```
containers:
- name: vllm-inference
  env:
  # torch.compile cache
  - name: XDG_CACHE_HOME
    value: "/compile-cache"
  - name: TORCHINDUCTOR_CACHE_DIR
    value: "/compile-cache/inductor"
  - name: TRITON_CACHE_DIR
    value: "/compile-cache/triton"
  volumeMounts:
  - name: compile-cache
    mountPath: /compile-cache
volumes:
- name: compile-cache
  hostPath:
    path: /mnt/k8s-disks/0/compile-cache
    type: DirectoryOrCreate
```

キャッシュパスを NVMe インスタンスストアに設定する

コンパイルされたアーティファクトとストリーミングされた重みは、高速ローカルストレージからメリットが得られます。NVMe インスタンスストアを持つ GPU インスタンス (G ファミリーインスタンスや P ファミリーインスタンスなど) では、ルート Amazon EBS ボリュームの約 1 GB/秒と比較して、インスタンスストアは約 30 GB/秒のスループットを提供します。キャッシュ `hostPath` を NVMe マウントポイントに誘導してスループットを最適化します。

> **Important**
>
> `hostPath` ボリュームは `type: DirectoryOrCreate` を使用します。NVMe インスタンスストアによってバックアップされていないパスを指定すると、Kubernetes は代わりにルート Amazon EBS ボリュームにディレクトリをサイレントに作成します。キャッシュは引き続き機能しますが、エラーや警告なしに NVMe パフォーマンスのメリットが失われます。

NVMe マウントポイントとその有効化方法は、EKS Auto Mode ノードとセルフマネージドノードで異なります。

コンピューティング	NVMe マウントポイント	NVMe インスタンスストアを有効にする方法
EKS 自動モード	`/mnt/.ephemeral`	リクエストされたエフェメラルストレージに基づいて動的に有効にします。EKS Auto Mode は、NodeClass でリクエストされた `ephemeralStorage.size` がインスタンスの使用可能な NVMe キャパシティよりも小さい場合にのみ、インスタンスに複数の NVMe ドライブがあるときに、NVMe インスタンスストアを RAID 0 アレイとしてフォーマットしてマウントします。リクエストされた `ephemeralStorage.size` が NVMe キャパシティ以上である場合、EKS Auto Mode はインスタンスストアを使用せず、パスはルート

コンピューティング	NVMe マウントポイント	NVMe インスタンスストアを有効にする方法
		EBS ボリュームによってバックアップされます。
セルフマネージド Karpenter	`/mnt/k8s-disks/0`	Karpenter `EC2NodeClass` で `instanceStorePolicy: RAID0` を設定します。これがないと、Karpenter はインスタンスストアボリュームを無視し、パスは NVMe によってバックアップされません。

EKS Auto Mode では、コンテナ仕様で `hostPath` を `/mnt/.ephemeral/compile-cache` に設定します。

```
volumes:
- name: compile-cache
  hostPath:
    path: /mnt/.ephemeral/compile-cache
    type: DirectoryOrCreate
```

セルフマネージド Karpenter では、コンテナ仕様で `hostPath` を `/mnt/k8s-disks/0/compile-cache` に設定します。

```
volumes:
- name: compile-cache
  hostPath:
    path: /mnt/k8s-disks/0/compile-cache
    type: DirectoryOrCreate
```

サンプル結果

1. ノード上の最初の Pod は torch.compile を実行し、コンパイルされたカーネルをホスト上の `/compile-cache` に書き込みます。
2. 同じノード上の後続の Pod は既存のキャッシュをマウントし、コンパイルを完全にスキップするため、torch.compile コールドスタートが約 50～80 秒から約 4～6 秒に短縮されます。

次の表は、異なるモデルサイズの同じノードキャッシュによる改善を示しています。

モデルサイズ	最初の Pod torch.compile (キャッシュなし)	後続の Pods torch.compile (同じノード)
60 GB	約 53 秒	約 6 秒
140 GB	約 60 秒	約 6 秒
640 GB	約 80 秒	約 6 秒

新しいノード上で torch.compile キャッシュを事前ウォーミングする

この手法は、ノード間で同種の GPU、テンソル並列処理、モデル、PyTorch バージョンを使用してマルチノード推論を実行する場合に適用されます。the section called “同じノードで torch.compile アーティファクトをキャッシュする” の手法は単一ノードのケースに焦点を当てていますが、スケールアップイベント中に追加された新しいノードは空の torch.compile キャッシュで始まります。新しく初期化されたノードのコールドスタート時間をさらに短縮するために、コンパイルされた torch.compile アーティファクトを S3 に保存し、クラスターを結合するときに新しいノードにそれらを事前ダウンロードするキャッシュメカニズムを実装できます。

一般的なアプローチは次のとおりです:

1. 最初の Pod が最初のノード上でモデルをコンパイルしたら、torch.compile アーティファクト (~15 MB) を S3 バケットにアップロードします。
2. 新しいノードがクラスターを結合したら、推論 Pod がスケジュールされる前に、キャッシュされたアーティファクトをノードのローカルストレージにダウンロードします。

例えば、torch.compile キャッシュをパッケージ化して S3 にアップロードする GPU ノード上で実行される DaemonSet を実装できます。また、推論 Pod が新しいノード上でスケジュールされる前に、そのキャッシュをローカルノードストレージに同期することもできます。

クロスノードキャッシュに関する考慮事項

torch.compile アーティファクトをノード間でキャッシュする場合、コンパイルされたカーネルは、キャッシュを生成したノードとそれを使用するノードの間でこれらのパラメータが一致する場合にのみ有効です。キャッシュメカニズムは、これらすべてを考慮する必要があります。パラメータが一致しない場合、再コンパイルを適用したり、ランタイムエラーを発生させたりする無効なキャッシュが

生成されます。キャッシュツールは、S3 オブジェクトキーやキャッシュディレクトリ構造に組み込むなど、これらのパラメータによってアーティファクトを区別する必要があります。

パラメータ	重要な理由
GPUタイプ	コンパイルされたカーネルは GPU アーキテクチャ固有です (例えば、H100 では sm_90、L4 では sm_89)。
テンソル並列性 (TP)	TP の度数が異なると、計算グラフのパーティションが異なります。
モデル	各モデルアーキテクチャとサイズは、異なるカーネルにコンパイルされます。
PyTorch バージョン	torch.compile および Triton コンパイラの内部では、バージョン間で重大な変更が発生する可能性があります。

サンプル結果

クロスノードキャッシュ事前ウォーミング (Qwen3-6-35B-A3B、67 GB、p5.48xlarge での TP=2 の 2x GPU) による指向性テストでは、新しくスケーリングされたノード上の最初の Pod は、既にウォームされたノード上の後続の Pod と同じスタートアップ時間を達成しました。

シナリオ	最初の Pod	2 番目の Pod
クロスノードキャッシュなし (新しいノード)	65 秒	16 秒
クロスノードキャッシュあり (新しいノード)	16 秒	16 秒

デプロイの例

次の例では、Run:ai Model Streamer のパフォーマンスチューニングと torch.compile キャッシュを単一の vLLM デプロイマニフェストに結合します。プレースホルダーの値を、独自の設定に置き換えます:

- `serviceAccountName` – モデルバケットへの Amazon S3 読み取りアクセス権がある IAM ロールを持つサービスアカウント。
- `nodeSelector` (`karpenter.sh/nodepool`) – GPU ノードプール名 (例: `gpu-nodepool-g6e-12xlarge`)。

- `--model` – モデルの重みへの Amazon S3 パス。
- `--model-loader-extra-config` – モデルサイズに基づいて `concurrency` を設定します: `ceil(total_model_size_gb / chunk_size_gb)` 例えば、チャンクサイズが 4 GB の 67 GB モデルでは、`ceil(67 / 4) = 17` が得られます。
- `--tensor-parallel-size` – テンソル並列 (TP) の度数を、モデルをメモリに収めるために必要な GPU の最小数に設定します。
- hostPath path – Karpenter を使用したセルフマネージドノードの NVMe インスタンスストアマウントポイント。EKS Auto Mode では、代わりに `/mnt/.ephemeral/compile-cache` を使用します。詳細については、「the section called “同じノードで torch.compile アーティファクトをキャッシュする”」を参照してください。

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: vllm-inference
  namespace: default
spec:
  replicas: 1
  selector:
    matchLabels:
      app: vllm-inference
  template:
    metadata:
      labels:
        app: vllm-inference
    spec:
      serviceAccountName: <SERVICE_ACCOUNT_NAME>
      nodeSelector:
        karpenter.sh/nodepool: <GPU_NODEPOOL>
      tolerations:
        - key: nvidia.com/gpu
          operator: Exists
          effect: NoSchedule
      containers:
        - name: vllm
          image: vllm/vllm-openai:v0.21.0
          command:
            - python3
            - -m
            - vllm.entrypoints.openai.api_server
```

```
        args:
          - --model=s3://<BUCKET_NAME>/<MODEL_PATH>
          - --load-format=runai_streamer
          - --model-loader-extra-config={"concurrency":17,"distributed":true}
          - --tensor-parallel-size=2
          - --max-model-len=8192
          - --host=0.0.0.0
          - --port=8000
        ports:
          - containerPort: 8000
            name: http
        env:
          # Run:ai streamer tuning
          - name: RUNAI_STREAMER_CHUNK_BYTESIZE
            value: "4294967296"
          - name: RUNAI_STREAMER_S3_REQUEST_TIMEOUT_MS
            value: "3000"
          - name: RUNAI_STREAMER_S3_LOW_SPEED_LIMIT
            value: "1048576"
          # torch.compile cache
          - name: XDG_CACHE_HOME
            value: "/compile-cache"
          - name: TORCHINDUCTOR_CACHE_DIR
            value: "/compile-cache/inductor"
          - name: TRITON_CACHE_DIR
            value: "/compile-cache/triton"
        resources:
          requests:
            cpu: "12"
            memory: 80Gi
            nvidia.com/gpu: "2"
          limits:
            nvidia.com/gpu: "2"
        volumeMounts:
          - name: compile-cache
            mountPath: /compile-cache
        startupProbe:
          httpGet:
            path: /health
            port: 8000
          periodSeconds: 10
          failureThreshold: 60
          initialDelaySeconds: 30
        readinessProbe:
```

```
            httpGet:
              path: /health
              port: 8000
            periodSeconds: 5
            timeoutSeconds: 3
      volumes:
        - name: compile-cache
          hostPath:
            path: /mnt/k8s-disks/0/compile-cache
            type: DirectoryOrCreate
```

AI/ML ワークロード用の Amazon EKS クラスター設定

> **Tip**
>
> 今後開催予定の Amazon EKS AI/ML ワークショップに登録してください。

このセクションは、AI/ML ワークロードに最適化された Amazon EKS クラスターを設定することを目的としています。Linux および Windows 最適化 AMI を使用した GPU アクセラレーションコンテナの実行、高性能ネットワーキング用の Elastic Fabric Adapter (EFA) を使用したクラスターのトレーニングの設定、および AWS Inferentia インスタンスを使用した推論クラスターの作成に関するガイダンスが表示されます。これには、前提条件、ステップバイステップの手順、デプロイに関する考慮事項が含まれます。

トピック

- GPU インスタンス向けに EKS 最適化高速 AMI を使用する
- GPU アクセラレーションコンテナを実行する (EC2 G シリーズでの Windows)
- Elastic Fabric Adapter を使用して Amazon EKS で機械学習トレーニングを実行する
- 機械学習のための Amazon EKS で AWS Inferentia インスタンスを使用する

GPU インスタンス向けに EKS 最適化高速 AMI を使用する

Amazon EKS は、GPU インスタンス向けに EKS 最適化 Amazon Linux および Bottlerocket AMI をサポートしています。EKS 最適化高速 AMI は、高速 Kubernetes スタック向けに事前構築および検証されたオペレーティングシステムイメージを提供することで、EKS クラスターでの AI および ML ワークロードの実行を簡素化します。標準の EKS 最適化 AMI に含まれるコア Kubernetes コンポー

ネントに加えて、EKS 最適化高速 AMI には、EKS クラスターで NVIDIA GPU G および P EC2 インスタンス、ならびに AWS GPU Inferentia および Trainium EC2 インスタンスを実行するために必要なカーネルモジュールとドライバーが含まれています。

以下の表は、EKS 最適化高速 AMI のバリアントごとにサポートされている GPU インスタンスタイプを示しています。AMI バリアントの最新の更新については、GitHub の EKS 最適化 AL2023 リリースと Bottlerocket リリースを参照してください。

EKS AMI バリアント	EC2 インスタンスタイプ
AL2023 x86_64 NVIDIA	p6-b300、p6-b200、p5、p5e、p5en、p4d、p4de、p3、p3dn、g7、g7e、gr6、g6、g6e、g6f、gr6f、g5、g4dn
AL2023 ARM NVIDIA	p6e-gb200、p6e-gb300、g5g
AL2023 x86_64 Neuron	inf1、inf2、trn1、trn2
Bottlerocket x86_64 aws-k8s-nvidia	p6-b300、p6-b200、p5、p5e、p5en、p4d、p4de、p3、p3dn、g7 (EKS 1.37 以降)、g7e、gr6、g6、g6e、g6f、gr6f、g5、g4dn
Bottlerocket aarch64/arm64 aws-k8s-nvidia	g5g
Bottlerocket x86_64 aws-k8s	inf1、inf2、trn1、trn2

EKS 最適化 NVIDIA AMI

EKS 最適化 NVIDIA AMI を使用することで、NVIDIA のクラウドエンドユーザーライセンス契約 (EULA) に同意したものとみなされます。

最新の EKS 最適化 NVIDIA AMI を確認するには、「the section called “最新の ID を取得する”」および「the section called “最新の ID を取得する”」を参照してください。

EKS 最適化 AL2023 または Bottlerocket NVIDIA AMI で Amazon Elastic Fabric Adapter (EFA) を使用する場合は、EFA デバイスプラグインを個別にインストールする必要があります。詳細については、「the section called “EFA を使用したクラスターのトレーニングの設定”」を参照してください。

EKS AL2023 NVIDIA AMI

EKS 最適化 AL2023 NVIDIA AMI で NVIDIA GPU Operator を使用する場合は、ドライバーとツールキットのオペレータインストールを無効にする必要があります。これらは既に EKS AMI に含まれているためです。EKS 最適化 AL2023 NVIDIA AMI には、NVIDIA Kubernetes デバイスプラグインまたは NVIDIA DRA ドライバーが含まれていないため、これらは個別にインストールする必要があります。詳細については、「the section called “NVIDIA GPU”」を参照してください。

標準の EKS AMI コンポーネントに加えて、EKS 最適化 AL2023 NVIDIA AMI には、以下のコンポーネントが含まれています。

- NVIDIA ドライバー
- NVIDIA CUDA ユーザーモードドライバー
- NVIDIA Container Toolkit
- NVIDIA ファブリックマネージャー
- NVIDIA 永続化
- NVIDIA IMEX ドライバー
- NVIDIA NVLink サブネットマネージャー
- NVIDIA GDRCopy ドライバー
- EFA 最小構成 (カーネルモジュールと rdma-core)

NVIDIA CUDA ユーザーモードドライバーとアプリケーションコンテナ内で使用される CUDA ランタイム/ライブラリの詳細については、NVIDIA ドキュメントを参照してください。`nvidia-smi` に表示される CUDA バージョンは、ホストにインストールされている NVIDIA CUDA ユーザーモードドライバーのバージョンであり、アプリケーションコンテナで使用される CUDA ランタイム/ライブラリと互換性がある必要があります。

EKS 最適化 AL2023 NVIDIA AMI は、Kubernetes バージョン 1.33 ~ 1.35 ではカーネル 6.12 を、Kubernetes バージョン 1.36 以降ではカーネル 6.18 をサポートしています。1 つの AMI に 2 つの NVIDIA ドライバーバージョンが含まれています。

各ノードは、初回起動時に搭載する GPU に適したドライバーを自動的に選択し、ノードの存続期間中はそのバージョンを使用し続けます。g7 インスタンスタイプには NVIDIA ドライバーバージョン 595 以降が必要ですが、サポートされている他のすべての GPU (g7e を含む) には NVIDIA ドライバーバージョン 580 が必要です。各ノードは、搭載する GPU に基づいてドライバーを自動的に選択します。ドライバーのバージョンを設定する必要はありません。

LTS (長期サポート) ドライバーのバージョンは現在 580 で、本番ブランチのドライバーバージョンは現在 595 です。CUDA 13 以降を使用するには、NVIDIA 580 ドライバーが必要です。

AMI に含まれるコンポーネントバージョンの詳細については、GitHub の EKS 最適化 AL2023 リリースを参照してください。EKS AMI が NVIDIA の依存関係を設定し、起動時にドライバーバージョンを選択する方法の詳細については、EKS AL2023 NVIDIA AMI のインストールスクリプトとドライバー解決スクリプトに関するページを参照してください。実行中の EC2 インスタンスでは、`dnf list installed` コマンドを使用して、インストールされているパッケージとそのバージョンのリストを確認できます。

EKS 最適化 AMI をベースとしてカスタム AMI を構築する場合、コンポーネントの互換性が損なわれる可能性があるため、オペレーティングシステムのアップグレード (`dnf upgrade` など) を実行したり、EKS 最適化 AMI に含まれる Kubernetes または GPU パッケージをアップグレードしたりすることは、推奨もサポートもされていません。EKS 最適化 AMI に含まれるオペレーティングシステムまたはパッケージをアップグレードする場合は、本番環境にデプロイする前に、開発環境またはステージング環境で徹底的にテストすることをお勧めします。

GPU インスタンス用のカスタム AMI を構築する場合は、実行するインスタンスタイプの世代およびファミリーごとに、個別のカスタム AMI を構築することをお勧めします。EKS 最適化高速 AMI は、基盤となるインスタンスタイプの世代およびファミリーに基づいて、ランタイム時にドライバーとパッケージを選択的にインストールします。詳細については、インストールとランタイムの EKS AMI スクリプトを参照してください。

EKS Bottlerocket NVIDIA AMI

EKS 最適化 Bottlerocket NVIDIA AMI で NVIDIA GPU Operator を使用する場合は、ドライバー、ツールキット、デバイスプラグインのオペレータインストールを無効にする必要があります。これらは既に EKS AMI に含まれているためです。

標準の EKS AMI コンポーネントに加えて、EKS 最適化 Bottlerocket NVIDIA AMI には、以下のコンポーネントが含まれています。EFA の最小限の依存関係 (カーネルモジュールと rdma-core) は、すべての Bottlerocket バリアントにインストールされます。

- NVIDIA Kubernetes デバイスプラグイン
- NVIDIA ドライバー
- NVIDIA CUDA ユーザーモードドライバー
- NVIDIA Container Toolkit
- NVIDIA ファブリックマネージャー

- NVIDIA 永続化
- NVIDIA IMEX ドライバー
- NVIDIA NVLink サブネットマネージャー
- NVIDIA MIG マネージャー

NVIDIA CUDA ユーザーモードドライバーとアプリケーションコンテナ内で使用される CUDA ランタイム/ライブラリの詳細については、NVIDIA ドキュメントを参照してください。`nvidia-smi` に表示される CUDA バージョンは、ホストにインストールされている NVIDIA CUDA ユーザーモードドライバーのバージョンであり、アプリケーションコンテナで使用される CUDA ランタイム/ライブラリと互換性がある必要があります。

インストールされているパッケージとそのバージョンの詳細については、Bottlerocket ドキュメントの Bottlerocket バージョン情報を参照してください。EKS 最適化 Bottlerocket NVIDIA AMI は、Kubernetes バージョン 1.33 ~ 1.35 ではカーネル 6.12 を、Kubernetes バージョン 1.36 以降ではカーネル 6.18 をサポートしています。これらの AMI には、Kubernetes バージョン 1.34 ~ 1.36 用の NVIDIA ドライバーバージョン 580 が含まれています。CUDA 13 以降を使用するには、NVIDIA 580 ドライバーが必要です。1.37 以降の EKS 最適化 Bottlerocket NVIDIA AMI は、g7 インスタンスタイプに必要な NVIDIA ドライバーバージョン 595 もサポートしています。

EKS 最適化 Neuron AMI

Amazon EKS で Neuron を使用するトレーニングおよび推論ワークロードの方法の詳細については、以下のリファレンスを参照してください。

- コンテナ – Kubernetes – 開始方法 (AWS Neuron ドキュメント)
- GitHub での AWS Neuron EKS サンプルによるトレーニング例
- Amazon EKS で Inferentia を使用して ML 推論ワークロードをデプロイする

最新の EKS 最適化 Neuron AMI を確認するには、「the section called “最新の ID を取得する”」および「the section called “最新の ID を取得する”」を参照してください。

EKS 最適化 AL2023 または Bottlerocket Neuron AMI で Amazon Elastic Fabric Adapter (EFA) を使用する場合は、EFA デバイスプラグインを個別にインストールする必要があります。詳細については、「the section called “EFA を使用したクラスターのトレーニングの設定”」を参照してください。

EKS AL2023 Neuron AMI

EKS 最適化 AL2023 Neuron AMI には、Neuron DRA ドライバー、Neuron Kubernetes デバイスプラグインまたは Neuron Kubernetes スケジューラ拡張機能が含まれていないため、これらは個別にインストールする必要があります。詳細については、「the section called “Neuron アクセラレーター”」を参照してください。

標準の EKS AMI コンポーネントに加えて、EKS 最適化 AL2023 Neuron AMI には、以下のコンポーネントが含まれています。

- Neuron ドライバー (aws-neuronx-dkms)
- Neuron ツール (aws-neuronx-tools)
- EFA 最小構成 (カーネルモジュールと rdma-core)

EKS AMI が Neuron 依存関係を設定する方法の詳細については、EKS AL2023 Neuron AMI のインストールスクリプトを参照してください。AMI に含まれるコンポーネントバージョンについては、GitHub の EKS 最適化 AL2023 リリースを参照してください。実行中の EC2 インスタンスでは、`dnf list installed` コマンドを使用して、インストールされているパッケージとそのバージョンのリストを確認できます。

EKS Bottlerocket Neuron AMI

標準の Bottlerocket バリアント (aws-k8s) には、AWS Inferentia または Trainium EC2 インスタンスでの実行時に自動的に検出およびロードされる Neuron 依存関係が含まれています。

EKS 最適化 Bottlerocket AMI には、Neuron DRA ドライバー、Neuron Kubernetes デバイスプラグインまたは Neuron Kubernetes スケジューラ拡張機能が含まれていないため、これらは個別にインストールする必要があります。詳細については、「the section called “Neuron アクセラレーター”」を参照してください。

標準の EKS AMI コンポーネントに加えて、EKS 最適化 Bottlerocket Neuron AMI には、以下のコンポーネントが含まれています。

- Neuron ドライバー (aws-neuronx-dkms)
- EFA 最小構成 (カーネルモジュールと rdma-core)

Neuron インスタンスで EKS 最適化 Bottlerocket AMIs を使用する場合は、Bottlerocket の user-data で以下を設定する必要があります。この設定により、コンテナは、ワークロード仕様で指定された

runAsUser および runAsGroupの値に基づいて、マウントされた Neuron デバイスの所有権を取得できます。Bottlerocket での Neuron サポートの詳細については、GitHub の EKS readme でのクイックスタートを参照してください。

```
[settings]
[settings.kubernetes]
device-ownership-from-security-context = true
```

EKS 最適化 Bottlerocket AMI に含まれる Neuron ドライバーのバージョンについては、Bottlerocket カーネルキットの変更ログを参照してください。

GPU アクセラレーションコンテナを実行する (EC2 G シリーズでの Windows)

> **Important**
>
> TensorWorks による DirectX 用 Kubernetes Device Plugin は、AWS によって承認、サポート、または保守されていないサードパーティーのツールです。AWS は、このプラグインのセキュリティ、信頼性、またはパフォーマンスについて責任を負いません。

TensorWorks の DirectX 用 Kubernetes デバイスプラグインで NVIDIA GPU を使用して、Amazon EKS (Elastic Kubernetes Service) で GPU アクセラレーション Windows コンテナワークロードを実行する方法について説明します。詳細については、「Kubernetes Device Plugin for DirectX」を参照してください。

Windows コンテナの GPU アクセラレーションをセットアップするには、主に 2 つの方法があります。

- オプション 1: 必要な GPU ドライバーがプリインストールされたカスタム EKS Windows 最適化 AMI を構築します。
  - このアプローチは、GPU アクセラレーションされた Windows コンテナをすぐに実行できる、一貫性のある事前設定済みの環境が必要で、カスタム AMI を構築して維持するために追加の労力を費やすことができる場合に使用します。
- オプション 2: インスタンスの起動後に、必要な GPU ドライバーを EKS ワーカーノードにインストールします。

- このアプローチは、より簡単なセットアッププロセスが必要で、新しいワーカーノードそれぞれに GPU ドライバーをインストールしたくない場合に使用します。GPU アクセラレーションのワークロードを評価またはプロトタイプ化する開発環境により適しています。

どちらのアプローチも、本ガイドで説明されているステップで活用することができます。

考慮事項

本ガイドでは、NVIDIA GPU、NVIDIA GRID ドライバー、および TensorWorks による DirectX 用 Kubernetes デバイスプラグインを使用して、Windows コンテナの GPU アクセラレーションをインストールしてセットアップする手順について説明します。これらのステップは、Amazon EKS の Windows コンテナワークロードに GPU アクセラレーションを提供するようにテストおよび検証されています。互換性のあるドライバーとデバイスプラグインの詳細については、「the section called “既知の制限事項”」を参照してください。先に進める前に、次の点に注意してください。

- NVIDIA GRID ドライバーを使用する G ファミリーインスタンスタイプのみが、本ガイドで動作することがテストおよび検証されています。他のインスタンスタイプとドライバーの組み合わせでも GPU アクセラレーション Windows コンテナを実行できる場合がありますが、本ガイドでは説明されていない追加の設定手順が必要になる場合があります。
- DirectX ベースのワークロードのみが、本ガイドで動作することがテストおよび検証されています。OpenGL、Vulkan、OpenCL などの他の GPU API でも、GPU アクセラレーションされた Windows コンテナの実行と互換性がある可能性がありますが、このガイドでは説明されていない追加の設定手順が必要になる場合があります。
- GPU アクセラレーション Windows コンテナを実行する前に注意すべき既知の制限がいくつかあります。詳細については、the section called “既知の制限事項” のセクションを参照してください。

前提条件

Amazon EKS で Windows コンテナの GPU アクセラレーションを有効にするには、手順を進める前に次の要件を準備する必要があります。

- Kubernetes v1.27 以降で Amazon EKS クラスターを起動します。
- Windows Server 2022 以降で Windows ノードをプロビジョニングします。
- G4 や G5 などのインスタンスタイプの G ファミリーで Windows ノードをプロビジョニングします。

- containerd `1.7.x` または `2.x.x` でコンテナランタイムを使用して Windows ノードをプロビジョニングします。(Amazon EKS 最適化 AMI 内の containerd バージョンを確認するには、the section called “バージョン情報を取得する” を参照してください。)

各 Windows ノードに GPU ドライバーをインストール

EKS ワーカーノードに NVIDIA GRID ドライバーをインストールするには、Amazon EC2 インスタンスの NVIDIA ドライバー で説明されているステップに従ってください。インストールオプション - オプション 3: GRID ドライバーに移動し、インストール手順に従ってください。

Windows Server Core のためのインストール

デスクトップエクスペリエンスのない Windows Server Core の場合は、次のコマンドを使用して NVIDIA GRID ドライバーをサイレントインストールします。

```
$nvidiaInstallerFilePath = nvidia-driver-installer.exe # Replace with path to installer
$installerArguments = "-s -clean -noreboot -noeula"
Start-Process -FilePath $nvidiaInstallerFilePath -ArgumentList $installerArguments -
Wait -NoNewWindow -PassThru
```

インストールを確認する

次の PowerShell コマンドを実行して、インスタンスの GPU に関する診断情報を表示します。

```
nvidia-smi
```

このコマンドは、NVIDIA ドライバーのバージョンと GPU ハードウェアに関する情報を表示します。このコマンドの出力が、インストールされると想定している NVIDIA GRID ドライバーのバージョンと一致することを確認します。

各ノードに GPU デバイスプラグインをデプロイする

GPU リソースの検出と Windows ノード上のコンテナへの公開を有効にするには、デバイスプラグインが必要になります。EKS クラスターで DaemonSet として実行することで、各ワーカーノードに Tensorworks による DirectX デバイスプラグインをデプロイします。README.md で指定されているインストールガイドに従ってください。これには、次のステップが求められます。以下を行うことをお勧めします。

- デバイスプラグインを `kube-system` 名前空間にデプロイします。

- DaemonSet に適切なリソース制限を設定して、ノードでリソースを過剰に消費しないようにします。

> **Note**
>
> デバイスプラグイン DaemonSet は、昇格された権限を持つホストプロセスコンテナとしてすべてのノードで実行されます。この DaemonSet へのアクセスを制限する RBAC コントロールを実装して、権限のあるユーザーのみが特権コマンドを実行できるようにすることをお勧めします。

GPU アクセラレーションコンテナを実行する場合、デバイスプラグインは 2 つのモードをサポートします。

- シングルテナンシーモード: このモードは、すべての GPU リソースをインスタンス上の 1 つのコンテナ専用にします。次のコマンドを使用して、シングルテナンシーをサポートするデバイスプラグインをインストールします。詳細については、「README.md」を参照してください。

  ```
  kubectl apply -f "https://raw.githubusercontent.com/TensorWorks/directx-device-plugins/main/deployments/default-daemonsets.yml"
  ```

- マルチテナンシーモード: このモードでは、インスタンス上の複数のコンテナ間で GPU リソースを共有できます。次のコマンドを使用して、マルチテナンシーをサポートするデバイスプラグインをインストールします。詳細については、「README.md」を参照してください。

  ```
  kubectl apply -f "https://raw.githubusercontent.com/TensorWorks/directx-device-plugins/main/deployments/multitenancy-inline.yml"
  ```

  または、ConfigMap を使用してマルチテナンシーを指定します。

  ```
  kubectl apply -f "https://raw.githubusercontent.com/TensorWorks/directx-device-plugins/main/deployments/multitenancy-configmap.yml"
  ```

デバイスプラグインのデプロイの検証

デバイスプラグインをデプロイしたら、<namespace> を置き換えてから次のコマンドを実行して、DirectX Device Plugin がすべての Windows ノードで正しく動作していることを確認します。

```
kubectl get ds device-plugin-wddm -n <namespace>
```

コンテナをデプロイする準備が整っていることの検証

デバイスプラグイン DaemonSet が GPU 搭載の Windows ワーカーノードで実行されると、次のコマンドを使用して、各ノードに割り当て可能な GPU があることを確認します。対応する番号は、各ノードの DirectX デバイスの番号と一致する必要があります。

```
kubectl get nodes "-o=custom-
columns=NAME:.metadata.name,DirectX:.status.allocatable.directx\.microsoft\.com/
display"
```

GPU アクセラレーションを使用した Windows コンテナの実行

ポッドを起動する前に、`.spec.containers[].resources` でリソース名 `directx.microsoft.com/display` を指定します。これは、コンテナに GPU が対応している機能が必要であることを示し、kube-scheduler は使用可能な GPU リソースを持つ事前設定された Windows ノードにポッドを配置しようとします。

例として、pi の値を推定するモンテカルロ・シミュレーションを実行する Job を起動する以下のサンプル・コマンドをご覧ください。この例は、DirectX 用の Kubernetes Device Plugins を持つ GitHub リポジトリのものです。このリポジトリには、Windows ノード GPU の機能をテストするために実行できる複数の例があります。

```
cat <<EOF | kubectl apply -f -
apiVersion: batch/v1
kind: Job
metadata:
  name: example-cuda-montecarlo-wddm
spec:
  template:
    spec:
      containers:
      - name: example-cuda-montecarlo-wddm
        image: "index.docker.io/tensorworks/example-cuda-montecarlo:0.0.1"
        resources:
          limits:
            directx.microsoft.com/display: 1
      nodeSelector:
        "kubernetes.io/os": windows
```

```
      restartPolicy: Never
  backoffLimit: 0
EOF
```

既知の制限事項

すべての GPU が使用可能です

インスタンスのすべての GPU は、特定のコンテナに対して特定の数の GPU をリクエストした場合でも、ホストで実行中の各コンテナで使用できるようになります。さらに、デフォルトの動作では、ノードで使用可能な複数の GPU がある場合でも、ホストで実行されているすべてのコンテナがインデックス 0 の GPU を使用します。したがって、マルチ GPU タスクを正しく動作させるには、アプリケーションのコード内で使用される特定の GPU デバイスを明示的に指定する必要があります。

アプリケーションに使用するデバイスを割り当てるための正確な実装は、使用しているプログラミング言語やフレームワークによって変わります。例えば、CUDA プログラミングを使用している場合、特定の GPU を選択するために、関数 cudaSetDevice() を使用して、アプリケーションコードで使用するデバイスを明示的に指定できます。

デバイスを明示的に指定する必要があるのは、Windows コンテナに影響する既知の問題があるからです。この問題の解決の進捗状況は、microsoft/Windows-Containers の問題 #333 で追跡できます。次の表は、この GPU 割り当て動作の視覚的な表現と実例を示しています。

EC2 インスタンスタイプ `g4dn.12xlarge` の Windows ノードが 1 つあり、4 つの GPU が搭載されているというシナリオを考えてみましょう。このインスタンスで起動されるポッドが 3 つあるシナリオを考えてみましょう。この表は、各コンテナがリクエストした GPU の数に関係なく、3 つのポッドすべてがインスタンス上の 4 つの GPU すべてにアクセスできることを示しており、デフォルトではデバイスインデックス 0 の GPU を利用できます。

ポッド	リクエストされた GPU	実際の GPU アクセス	デフォルトの GPU 使用率	使用可能な GPU インデックス	インスタンス GPU の合計
Pod 1	1 GPU	すべての 4 GPU	インデックス 0 の GPU	0、1、2、3	4
Pod 2	GPU 2 個	すべての 4 GPU	インデックス 0 の GPU	0、1、2、3	4

ポッド	リクエストされた GPU	実際の GPU アクセス	デフォルトの GPU 使用率	使用可能な GPU インデックス	インスタンス GPU の合計
Pod 3	1 GPU	すべての 4 GPU	インデックス 0 の GPU	0、1、2、3	4

Kubernetes デバイスプラグインのサポート

NVIDIA の Kubernetes デバイスプラグインの公式実装は Windows をサポートしていません。NVIDIA/k8s-device-plugin issue #419 で、Windows の公式サポート追加の進行状況を追跡できます。

GPU コンピューティングインスタンスの制限

AWS アカウント設定によっては、起動できる Amazon EC2 GPU コンピューティングインスタンスの数とタイプにサービス制限がある場合があります。追加の容量が必要な場合は、クォータの引き上げをリクエストできます。

Windows GPU 最適化 AMI を構築する必要があります

Amazon EKS が提供する EKS Windows GPU 最適化 AMI または EC2 Image Builder マネージドコンポーネントはありません。このガイドの手順に従って、必要な GPU ドライバーがプリインストールされたカスタム EKS Windows 最適化 AMI を構築するか、インスタンスの起動後に必要な GPU ドライバーを EKS ワーカーノードにインストールする必要があります。

Inferentia と Trainium はサポートされていません

AWS Inferentia および AWS Trainium ベースのワークロードは Windows ではサポートされていません。

Elastic Fabric Adapter を使用して Amazon EKS で機械学習トレーニングを実行する

このトピックでは、Amazon EKS クラスターにデプロイした Pod と Elastic Fabric Adapter (EFA) を統合する方法について説明します。Elastic Fabric Adapter (EFA) は、Amazon EC2 インスタンス向けのネットワークインターフェイスです。これにより、AWS で高レベルのノード間通信を必要とするアプリケーションを実行できます。カスタムビルドされたオペレーティングシステムのバイ

パスハードウェアインターフェイスにより、インスタンス間通信のパフォーマンスが向上します。これは、これらのアプリケーションのスケーリングに不可欠です。EFA を使用すると、Message Passing Interface (MPI) を使用するハイパフォーマンスコンピューティング (HPC) アプリケーションと NVIDIA Collective Communications Library (NCCL) を使用する Machine Learning (ML) アプリケーションを、数千の CPU または GPU にスケールできます。その結果、AWS クラウドのオンデマンドの伸縮自在性と柔軟性を備えたオンプレミスの HPC クラスターのアプリケーションパフォーマンスを実現できます。EFA と Amazon EKS クラスターで実行中のアプリケーションを統合すると、大規模な分散型トレーニングワークロードを完了する時間を短縮できます。クラスターにインスタンスを追加する必要はありません。EFA の詳細については、「Elastic Fabric Adapter」を参照してください。

EFA DRA ドライバー、EFA デバイスプラグイン、トポロジー認識割り当て、ネットワークインターフェイス設定など、EKS クラスターでの EFA デバイスのインストールと管理については、「the section called “EFA デバイス”」を参照してください。

EFA を使用するインスタンスタイプ

AWS EFA Kubernetes Device Plugin は、EFA を持つすべての Amazon EC2 インスタンスタイプをサポートします。EFA を持つすべてのインスタンスタイプのリストを見るには、「Amazon EC2 ユーザーガイド」の「サポートされるインスタンスタイプ」を参照してください。ただし、機械学習アプリケーションを迅速に実行するために、EFA に加えて、NVIDIA GPU、AWS Inferentia チップ、AWS Trainium チップなどのハードウェアアクセラレーションチップもインスタンスに搭載しておくことをお勧めします。ハードウェアアクセラレーションチップと EFA を持つインスタンスタイプのリストを見るには、「Amazon EC2 ユーザーガイド」の「高速コンピューティング」を参照してください。

インスタンスタイプを比較して選択する場合、そのインスタンスタイプで使用できる EFA ネットワークカードの数と、アクセラレータカードの数、CPU の量、メモリの量を考慮してください。ネットワークカードごとに EFA を 1 つまで割り当てることができます。EFA はネットワークインターフェイスとしてカウントされます。EFA を持つ各インスタンスタイプで使用できる EFA の数を確認するには、「Amazon EC2 ユーザーガイド」の「ネットワークカード」のリストを参照してください。

EFA および EFA 専用インターフェイス

Elastic Fabric Adapter (EFA) は、Elastic Network Adapter (ENA) と OS バイパスインターフェイスの機能を組み合わせたネットワークインターフェイスで、AWS Scalable Reliable Datagram (SRD) プロトコルを搭載しています。EFA 機能を使用すると、アプリケーションはハードウェアと直接通信して低レイテンシーの転送を実現できます。EFA 専用インターフェイスを使用して EFA 機能にのみ

アクセスすることを選択し、通信を同じアベイラビリティーゾーン内のインターフェイスに制限できます。

EFA 専用インターフェイスを持つノードを作成するには、カスタム EC2 起動テンプレートを使用して `InterfaceType` を `efa-only` に設定する必要があります。カスタム起動テンプレートでは、ネットワークカード `0` を EFA 専用インターフェイスに設定することはできません。このネットワークカードが EC2 インスタンスのプライマリネットワークカードおよびネットワークインターフェイスであるためです。EFA 専用インターフェイスには、VPC CNI バージョン `1.18.5` 以降が必要です。Amazon Linux 2 を使用している場合、EFA 専用インターフェイスの AMI バージョンは `v20240928` 以降である必要があります。

次の手順では、NVIDIA GPU と EFA インターフェイスを持つノードで `eksctl` を使用して、EKS クラスターを作成する手順を説明します。`eksctl` を使用して、EFA 専用インターフェイスを使用するノードとノードグループを作成することはできません。

前提条件

- 既存の Amazon EKS クラスター。既存のクラスターがない場合は、はじめに を使用して作成します。クラスターは、ノードをデプロイするのに十分な IP アドレスを持つ、少なくとも 1 つのプライベートサブネットがある VPC にデプロイする必要があります。プライベートサブネットには、NAT ゲートウェイなどの外部のデバイスから提供されるアウトバウンドのインターネットアクセスが必要です。

  `eksctl` を使用してノードグループを作成する予定がある場合、`eksctl` でクラスターを作成することもできます。
- ご使用のデバイスまたは AWS CloudShell で、バージョン `2.12.3` 以降、または AWS コマンドラインインターフェイス (AWS CLI) のバージョン `1.27.160` 以降がインストールおよび設定されていること。現在のバージョンを確認するには、`aws --version | cut -d / -f2 | cut -d ' ' -f1` を参照してください。`yum`、`apt-get`、macOS 用の Homebrew などのパッケージマネージャーは、多くの場合 AWS CLI の最新バージョンより数バージョン古くなっています。最新バージョンをインストールするには、「 AWS コマンドラインインターフェイスユーザーガイド」の「インストール」および「aws configure を使用したクイック設定」を参照してください。AWS クラウドシェル にインストールされている AWS CLI バージョンも最新バージョンより数バージョン遅れることがあります。更新するには、「AWS CloudShell ユーザーガイド」の「ホームディレクトリへの AWS CLI のインストール」を参照してください。
- デバイスまたは AWS CloudShell に、`kubectl` コマンドラインツールがインストールされていること。バージョンはご使用のクラスターの Kubernetes バージョンと同じか、1 つ前のマイナーバージョン以前、あるいはそれより新しいバージョンが使用できます。例えば、クラスターの

バージョンが 1.29 である場合、kubectl のバージョン 1.28、1.29、または 1.30 が使用できます。kubectl をインストールまたはアップグレードする方法については、the section called “kubectl と eksctl のセットアップ” を参照してください。

- p4d や p5 といった複数の Elastic Fabric Adapters をサポートするワーカーノードを起動する前に、Amazon VPC CNI plugin for Kubernetes バージョン 1.7.10 以降をインストールしておく必要があります。Amazon VPC CNI plugin for Kubernetes バージョンの更新の詳細については、「the section called “Amazon VPC CNI”」を参照してください。
- p6-b200 インスタンスの場合、EFA Device プラグインバージョン v0.5.6 以降を使用する必要があります。

> **Important**
>
> Kubernetes で EFA を導入するために必要となる重要な考慮事項は、クラスター内のリソースとして Huge Page を設定および管理することです。詳細については、Kubernetes ドキュメントの「huge pageを管理する」を参照してください。EFA ドライバーがインストールされた Amazon EC2 インスタンスでは、5128 2MiB Huge Page が事前に割り当てられます。このページは、ジョブの仕様で使用するリソースとしてリクエストできます。

ノードグループの作成

次の手順は、EFA インターフェイスと GPUDirect RDMA を使用して、p4d.24xlarge がバックアップされたノードグループでノードグループを作成し、EFA を使用したマルチノード NCCL パフォーマンス向けに NVIDIA Collective Communications Library (NCCL) のサンプルテストを実行するのに役立ちます。この例は、EFA を使用した Amazon EKS の分散型深層学習トレーニングのテンプレートとして使用できます。

1. ノードをデプロイしたい AWS リージョンで利用可能な、EFA をサポートする Amazon EC2 インスタンスを特定します。*region-code* を、ノードグループをデプロイする AWS リージョンに置き換えます。

```
aws ec2 describe-instance-types --region region-code \
    --filters Name=network-info.efa-supported,Values=true \
    --query "InstanceTypes[*].[InstanceType]" --output text
```

ノードをデプロイする場合、デプロイするインスタンスタイプは、ユーザーのクラスターがある AWS リージョンで利用可能である必要がります。

2. デプロイするインスタンスタイプが利用可能なアベイラビリティーゾーンを特定します。このチュートリアルでは、p5.48xlarge インスタンスタイプが使用されているため、前のステップで指定した AWS リージョンの出力が返されているはずです。本番クラスターにノードをデプロイするときは、*p5.48xlarge* を前のステップで返されたいずれかのインスタンスタイプに置き換えます。

```
aws ec2 describe-instance-type-offerings --region region-code \
    --location-type availability-zone --filters Name=instance-
type,Values=p4d.24xlarge,p5.48xlarge \
    --query 'InstanceTypeOfferings[*].Location' --output text
```

出力例は次のとおりです。

```
us-west-2a    us-west-2c    us-west-2b
```

後のステップで使用するために、返されたアベイラビリティーゾーンをメモします。ノードをクラスターにデプロイする場合、VPC には、出力で返されるアベイラビリティーゾーンの 1 つに使用可能な IP アドレスを持つサブネットが必要です。

3. eksctl を使用してノードグループを作成します。デバイスまたは AWS CloudShell にインストールされている eksctl コマンドラインツールのバージョン `0.215.0` 以降が必要です。eksctl をインストールまたはアップグレードするには、eksctl ドキュメントの「インストール」を参照してください。

   a. 次の内容を *efa-cluster.yaml* という名前のファイルにコピーします。値の例は、実際に使用する値に置き換えます。p5.48xlarge を異なるインスタンスに置き換えることができますが、その場合、availabilityZones の値がステップ 1 でインスタンスタイプに対して返されたアベイラビリティーゾーンであることを確認してください。

```
apiVersion: eksctl.io/v1alpha5
kind: ClusterConfig

metadata:
  name: my-efa-cluster
  region: region-code
  version: "1.XX"

iam:
  withOIDC: true

availabilityZones: ["us-west-2a", "us-west-2c"]
```

```
managedNodeGroups:
  - name: my-efa-ng
    instanceType: p5.48xlarge
    minSize: 1
    desiredCapacity: 2
    maxSize: 3
    availabilityZones: ["us-west-2a"]
    volumeSize: 300
    privateNetworking: true
    efaEnabled: true
```

b. 既存のクラスターにマネージド型ノードグループを作成します。

```
eksctl create nodegroup -f efa-cluster.yaml
```

既存のクラスターがない場合は、次のコマンドを実行してクラスターとノードグループを作成できます。

```
eksctl create cluster -f efa-cluster.yaml
```

> **Note**
>
> この例で使用されているインスタンスタイプには GPU が含まれているため、Amazon Linux 2 を使用するときは `eksctl` で NVIDIA Kubernetes デバイスプラグインが各インスタンスに自動的にインストールされます。NVIDIA デバイスプラグインが Bottlerocket の EKS NVIDIA バリアントに組み込まれているため、このプラグインは Bottlerocket には必要ありません。ノードグループ設定で `efaEnabled` が `true` に設定されている場合、`eksctl` では EFA デバイスプラグインもノードに自動的にデプロイされます。

EFA で Bottlerocket を使用する

Bottlerocket AMI バージョン 1.28.0 以降には、EFA の公式サポートが含まれています。EFA 対応ノードに Bottlerocket を使用するには、設定で `amiFamily: Bottlerocket` を指定します。カスタム AMI ID を使用する必要がある場合は、`managedNodeGroups` の代わりに標準の `nodeGroups` を使用する必要があります。

設定の例を次に示します。

```
apiVersion: eksctl.io/v1alpha5
kind: ClusterConfig

metadata:
  name: my-efa-bottlerocket-cluster
  region: region-code
  version: "1.XX"

iam:
  withOIDC: true

availabilityZones: ["us-west-2a", "us-west-2c"]

managedNodeGroups:
  - name: my-efa-bottlerocket-ng
    instanceType: p5.48xlarge
    minSize: 1
    desiredCapacity: 2
    maxSize: 3
    availabilityZones: ["us-west-2a"]
    volumeSize: 300
    privateNetworking: true
    efaEnabled: true
    amiFamily: Bottlerocket
    bottlerocket:
      enableAdminContainer: true
      settings:
        kernel:
          sysctl:
            "vm.nr_hugepages": "3000"  # Configures 3000 * 2Mi = 6000Mi hugepages
```

上記の vm.nr_hugepages sysctl 設定で、2Mi hugepages の数を指定します。この例で、3000 は 3000 x 2Mi = 6000Mi の hugepages を意味します。

EFA デバイスプラグインのインストールを確認する

efaEnabled: true を使用してノードグループを作成すると、eksctl によって EFA Kubernetes デバイスプラグインが自動的にデプロイされます。デバイスプラグインが正しくインストールされ、機能していることを確認できます。

1. DaemonSet のステータスを確認します。

```
kubectl get daemonsets -n kube-system
```

サンプル出力:

```
NAME                                  DESIRED   CURRENT   READY   UP-TO-DATE
 AVAILABLE   NODE SELECTOR   AGE
aws-efa-k8s-device-plugin-daemonset   2         2         2       2            2
      <none>          6m16s
...
```

ここでは、EFA デバイスプラグインの DaemonSet が 2 つのノードで実行されています。どちらも READY と AVAILABLE です。

2. 次に、DaemonSet によって作成されたポッドを確認します。

```
kubectl get pods -n kube-system -l name=aws-efa-k8s-device-plugin
```

サンプル出力:

```
NAME                                        READY   STATUS    RESTARTS   AGE
aws-efa-k8s-device-plugin-daemonset-d68bs   1/1     Running   0          6m16s
aws-efa-k8s-device-plugin-daemonset-w4l8t   1/1     Running   0          6m16s
```

EFA デバイスプラグインポッドが実行中状態であり、プラグインが正常にデプロイされ、動作していることを確認します。

3. リソース登録を確認します。

vpc.amazonaws.com/efa リソースが kubelet に登録されていることを確認するには、ノードを記述します。

```
kubectl describe nodes
```

EFA リソースが正しく登録されている場合、ノードのキャパシティと割り当て可能なリソースの下に一覧表示されます。例えば、次のようになります。

```
Capacity:
  ...
  vpc.amazonaws.com/efa:  4
```

```
Allocatable:
  ...
  vpc.amazonaws.com/efa:  4
```

この出力では、ノードが EFA リソースを認識し、それをリクエストするポッドで使用できるようになっていることが確認されます。

(オプション) EFA のパフォーマンスをテストする

EFA セットアップをテストすることをお勧めします。GitHub の `aws-samples/awsome-distributed-training` リポジトリで NCCL テストを使用できます。NCCL テストは、NVIDIA Collective Communications Library を使用してネットワークのパフォーマンスを評価します。次のステップでは、Amazon EKS で NCCL テストを送信します。

1. Kubeflow MPI Operator のデプロイ:

   NCCL テストでは、Kubeflow MPI Operator を適用できます。MPI Operator を使用すると、Kubernetes で AllReduce スタイルの分散型トレーニングを簡単に実行できます。詳細については、GitHub の「MPI Operator」を参照してください。

2. マルチノードの NCCL パフォーマンステストを実行して、GPUDirectRDMA/EFA を検証する:

   GPUDirectRDMA over EFA を使用した NCCL パフォーマンスを検証するには、標準の NCCL パフォーマンステストを実施します。詳細については、GitHub で公式の「NCCL-Tests」(NCCL テスト) を参照してください。

   2 ノード NCCLL パフォーマンステストを実行するには、次の手順を実行します。サンプルの NCCL テストジョブでは、各ワーカーが 8 つの GPU、5120Mi の `hugepages-2Mi`、4 つの EFA、8000Mi のメモリをリクエストします。これは事実上、各ワーカーが `p5.48xlarge` インスタンスのすべてのリソースを使用することを意味します。

   a. MPIJob マニフェストを作成する:

      次の JSON を `nccl-tests.yaml` という名前のファイルにコピーします:

```
apiVersion: kubeflow.org/v2beta1
kind: MPIJob
metadata:
  name: nccl-tests
spec:
  runPolicy:
```

```
    cleanPodPolicy: Running
    backoffLimit: 20
  slotsPerWorker: 8
  mpiReplicaSpecs:
    Launcher:
      replicas: 1
      template:
         spec:
          restartPolicy: OnFailure
          containers:
          - image: public.ecr.aws/hpc-cloud/nccl-tests:latest
            imagePullPolicy: IfNotPresent
            name: test-nccl-launcher
            env:
             - name: PATH
               value: $PATH:/opt/amazon/efa/bin:/usr/bin
            command:
            - /opt/amazon/openmpi/bin/mpirun
            - --allow-run-as-root
            - --tag-output
            - -np
            - "16"
            - -N
            - "8"
            - --bind-to
            - none
            - -x
            - PATH
            - -x
            - LD_LIBRARY_PATH
            - -x
            - NCCL_DEBUG=INFO
            - -x
            - NCCL_BUFFSIZE=8388608
            - -x
            - NCCL_P2P_NET_CHUNKSIZE=524288
            - -x
            - NCCL_TUNER_PLUGIN=/opt/amazon/ofi-nccl/lib/x86_64-linux-gnu/libnccl-
ofi-tuner.so
            - --mca
            - pml
            - ^cm,ucx
            - --mca
            - btl
```

```
            - tcp,self
            - --mca
            - btl_tcp_if_exclude
            - lo,docker0,veth_def_agent
            - /opt/nccl-tests/build/all_reduce_perf
            - -b
            - "8"
            - -e
            - "16G"
            - -f
            - "2"
            - -g
            - "1"
            - -c
            - "1"
            - -n
            - "100"
    Worker:
      replicas: 2
      template:
        spec:
          nodeSelector:
            node.kubernetes.io/instance-type: "p5.48xlarge"
          containers:
          - image: public.ecr.aws/hpc-cloud/nccl-tests:latest
            imagePullPolicy: IfNotPresent
            name: nccl-tests-worker
            volumeMounts:
            - name: shmem
              mountPath: /dev/shm
            resources:
              limits:
                nvidia.com/gpu: 8
                hugepages-2Mi: 5120Mi
                vpc.amazonaws.com/efa: 32
                memory: 32000Mi
              requests:
                nvidia.com/gpu: 8
                hugepages-2Mi: 5120Mi
                vpc.amazonaws.com/efa: 32
                memory: 32000Mi
          volumes:
          - name: shmem
            hostPath:
```

```
                path: /dev/shm
```

b. NCCL テスト MPIJob を適用する:

マニフェストを適用して MPIJob を送信します。これにより、2 つの p5.48xlarge Amazon EC2 インスタンスが作成されます。

```
kubectl apply -f nccl-tests.yaml
```

出力例は次のとおりです。

```
mpijob.kubeflow.org/nccl-tests created
```

c. ジョブがポッドを開始したことを検証する:

実行中の Pod を表示します。

```
kubectl get pods
```

出力例は次のとおりです。

```
NAME                             READY   STATUS     RESTARTS   AGE
nccl-tests-launcher-nbql9    0/1     Init:0/1   0          2m49s
nccl-tests-worker-0          1/1     Running    0          2m49s
nccl-tests-worker-1          1/1     Running    0          2m49s
```

MPI Operator は、ランチャー Pod と 2 つのワーカー Pod (ノードごとに 1 つ) を作成します。

d. ログを使用してジョブが正常に実行されていることを検証する:

nccl-tests-launcher Pod のログを表示します。*nbql9* を出力の値で置き換えます。

```
kubectl logs -f nccl-tests-launcher-nbql9
```

テストが正常に完了すると、NVIDIA Collective Communications Library を使用するアプリケーションをデプロイできます。

機械学習のための Amazon EKS で AWS Inferentia インスタンスを使用する

このトピックではAmazon EC2 Inf1 インスタンスを実行中のノードで Amazon EKS クラスターを作成する方法、(オプションで) サンプルアプリケーションをデプロイする方法について説明します。Amazon EC2 Inf1 インスタンスは AWS インフェクエンティア チップを利用しています。Inf1 チップはクラウド内で高パフォーマンスと低コストの推論を提供するために、AWS によってカスタムビルドされたものです。機械学習モデルはAWS Neuron を使用してコンテナにデプロイされます。Neuron はコンパイラー、ランタイム、および インフェクエンティア チップの機械学習推論パフォーマンスを最適化するプロファイリングツールで構成される software development kit (SDK です。AWSNeuron はTensorFlow、PyTorch、MXNet などの一般的な機械学習フレームワークをサポートしています。

> **Note**
>
> Neuron デバイスの論理 ID は連続している必要があります。複数の Neuron デバイスをリクエストする Pod が、(複数の Neuron デバイスを持つ) `inf1.6xlarge` または `inf1.24xlarge` インスタンスタイプでスケジュールされている場合、Kubernetes スケジューラが連続していないデバイス ID を選択すると、その Pod は起動に失敗します。詳細については、GitHub の「Device logical IDs must be contiguous」を参照してください。

前提条件

- コンピュータに `eksctl` がインストールされている。インストールされていない場合は`eksctl` ドキュメントの「インストール」を参照してください。
- コンピュータに `kubectl` がインストールされている。詳細については「the section called “kubectl と eksctl のセットアップ”」を参照してください。
- (オプション) コンピュータに `python3` がインストールされている。インストールされていない場合は「Python ダウンロード」でインストール手順を参照してください。

クラスターを作成する

1. Inf1 Amazon EC2 インスタンスを実行するノードを持つクラスターを作成します。*`inf1.2xlarge`* を任意の Inf1 インスタンスタイプ に置き換えることができます。`eksctl` ユーティリティは`Inf1` インスタンスタイプでノードグループを起動していることを検出し、アク

セラレーションされた Amazon EKS 最適化 Amazon Linux AMI のいずれかを使用してノードを起動します。

> **Note**
>
> TensorFlow Serving でサービスアカウント用の IAM 役割を使用することはできません。

```
eksctl create cluster \
    --name inferentia \
    --region region-code \
    --nodegroup-name ng-inf1 \
    --node-type inf1.2xlarge \
    --nodes 2 \
    --nodes-min 1 \
    --nodes-max 4 \
    --ssh-access \
    --ssh-public-key your-key \
    --with-oidc
```

> **Note**
>
> 出力の次の行の値をメモします。これは後の (オプションの) ステップで使用されます。

```
[9]  adding identity "arn:aws:iam::111122223333:role/eksctl-inferentia-nodegroup-ng-in-NodeInstanceRole-FI7HIYS3BS09" to auth ConfigMap
```

`Inf1` インスタンスでノードグループを起動すると、`eksctl` によって AWS Neuron Kubernetes デバイスプラグインが自動的にインストールされます。このプラグインは、Neuron デバイスをシステムリソースとして Kubernetes スケジューラにアドバタイズします。このスケジューラはコンテナによってリクエストできます。デフォルトの Amazon EKS ノードの IAM ポリシーに加えて、Amazon S3 読み取り専用のアクセスポリシーが追加されます。これにより、後のステップで説明するサンプルアプリケーションで Amazon S3 からトレーニングされたモデルをロードできるようになります。

2. すべての Pod が正しく起動していることを確認します。

```
kubectl get pods -n kube-system
```

簡略化された出力:

```
NAME                                   READY   STATUS    RESTARTS   AGE
[...]
neuron-device-plugin-daemonset-6djhp   1/1     Running   0          5m
neuron-device-plugin-daemonset-hwjsj   1/1     Running   0          5m
```

(オプション) TensorFlow Serving アプリケーションイメージのデプロイ

トレーニング済みモデルはインフェクエンティア インスタンスにデプロイする前に、インフェクエンティア ターゲットにコンパイルする必要があります。続行するにはAmazon S3 に保存されている Neuron optimized TensorFlow モデルが必要になります。SavedModel がまだない場合はNeuron 互換の ResNet50 モデルの作成についてのチュートリアルに従って、結果の SavedModel を S3 にアップロードします。ResNet-50 は画像認識のタスクに使用される一般的な機械学習モデルです。Neuron モデルのコンパイルについての詳細は「AWS ディープラーニング AMI デベロッパーガイド」の「AWS インフェクエンティア チップと DLAMI」を参照してください。

サンプルのデプロイメントマニフェストではAWS ディープラーニング Containers によって提供される TensorFlow の事前構築済みの推論サービングコンテナが管理されます。コンテナ内には、AWS Neuron ランタイムと TensorFlow Serving アプリケーションがあります。Neuron 用に最適化された事前構築済みの Deep Learning Containers の完全なリストは、GitHub で利用可能なイメージに保持されます。起動時に、DLC は Amazon S3 からモデルを取得し、保存したモデルで Neuron TensorFlow Serving を起動して、予測リクエストのために待機します。

サービングアプリケーションに割り当てられる Neuron デバイスの数はデプロイメント yaml で `aws.amazon.com/neuron` リソースを変更することで調整できます。TensorFlow Serving と Neuron ランタイム間の通信は GRPC 経由で行われることに注意してください。GRPC では `IPC_LOCK` 機能をコンテナに追加する必要があります。

1. 「クラスターを作成する」のステップ 1 で作成したノードのインスタンス役割に `AmazonS3ReadOnlyAccess` IAM ポリシーを追加します。これはサンプルアプリケーションが Amazon S3 からトレーニングされたモデルをロードできるようにするために必要です。

```
aws iam attach-role-policy \
    --policy-arn arn:aws:iam::aws:policy/AmazonS3ReadOnlyAccess \
```

```
    --role-name eksctl-inferentia-nodegroup-ng-in-NodeInstanceRole-FI7HIYS3BS09
```

2. 次の内容で、rn50_deployment.yaml という名前のファイルを作成します。希望の設定に合わせて、リージョンコードとモデルパスを更新します。モデル名はクライアントが TensorFlow サーバーにリクエストを行う際の識別を目的としています。この例ではモデル名を使用してサンプルの ResNet50 のクライアントスクリプトと照合します。このスクリプトは後のステップで予測リクエストを送信する際に使用されます。

```
aws ecr list-images --repository-name neuron-rtd --registry-id 790709498068 --region
 us-west-2
```

```
kind: Deployment
apiVersion: apps/v1
metadata:
  name: eks-neuron-test
  labels:
    app: eks-neuron-test
    role: master
spec:
  replicas: 2
  selector:
    matchLabels:
      app: eks-neuron-test
      role: master
  template:
    metadata:
      labels:
        app: eks-neuron-test
        role: master
    spec:
      containers:
        - name: eks-neuron-test
          image: 763104351884.dkr.ecr.us-east-1.amazonaws.com/tensorflow-inference-
neuron:1.15.4-neuron-py37-ubuntu18.04
          command:
            - /usr/local/bin/entrypoint.sh
          args:
            - --port=8500
            - --rest_api_port=9000
            - --model_name=resnet50_neuron
            - --model_base_path=s3://${your-bucket-of-models}/resnet50_neuron/
          ports:
```

```
            - containerPort: 8500
            - containerPort: 9000
          imagePullPolicy: IfNotPresent
          env:
            - name: AWS_REGION
              value: "us-east-1"
            - name: S3_USE_HTTPS
              value: "1"
            - name: S3_VERIFY_SSL
              value: "0"
            - name: S3_ENDPOINT
              value: s3.us-east-1.amazonaws.com
            - name: AWS_LOG_LEVEL
              value: "3"
          resources:
            limits:
              cpu: 4
              memory: 4Gi
              aws.amazon.com/neuron: 1
            requests:
              cpu: "1"
              memory: 1Gi
          securityContext:
            capabilities:
              add:
                - IPC_LOCK
```

3. モデルをデプロイします。

```
kubectl apply -f rn50_deployment.yaml
```

4. 次の内容で、rn50_service.yaml というファイルを作成します。予測リクエストを受け入れるために HTTP ポートと gRPC ポートが開かれます。

```
kind: Service
apiVersion: v1
metadata:
  name: eks-neuron-test
  labels:
    app: eks-neuron-test
spec:
  type: ClusterIP
  ports:
```

```
    - name: http-tf-serving
      port: 8500
      targetPort: 8500
    - name: grpc-tf-serving
      port: 9000
      targetPort: 9000
  selector:
    app: eks-neuron-test
    role: master
```

5. TensorFlow モデル提供アプリケーション用の Kubernetes サービスを作成します。

```
kubectl apply -f rn50_service.yaml
```

(オプション) TensorFlow Serving サービスに対する予測を行う

1. ローカルでテストするにはgRPC ポートを eks-neuron-test サービスに転送します。

```
kubectl port-forward service/eks-neuron-test 8500:8500 &
```

2. 次の内容で tensorflow-model-server-infer.py という Python スクリプトを作成します。このスクリプトはサービスフレームワークである gRPC を介して推論を実行します。

```
import numpy as np
import grpc
import tensorflow as tf
from tensorflow.keras.preprocessing import image
from tensorflow.keras.applications.resnet50 import preprocess_input
from tensorflow_serving.apis import predict_pb2
from tensorflow_serving.apis import prediction_service_pb2_grpc
from tensorflow.keras.applications.resnet50 import decode_predictions

if __name__ == '__main__':
   channel = grpc.insecure_channel('localhost:8500')
   stub = prediction_service_pb2_grpc.PredictionServiceStub(channel)
   img_file = tf.keras.utils.get_file(
       "./kitten_small.jpg",
       "https://raw.githubusercontent.com/awslabs/mxnet-model-server/master/docs/
images/kitten_small.jpg")
   img = image.load_img(img_file, target_size=(224, 224))
   img_array = preprocess_input(image.img_to_array(img)[None, ...])
```

```
    request = predict_pb2.PredictRequest()
    request.model_spec.name = 'resnet50_neuron'
    request.inputs['input'].CopyFrom(
        tf.make_tensor_proto(img_array, shape=img_array.shape))
    result = stub.Predict(request)
    prediction = tf.make_ndarray(result.outputs['output'])
    print(decode_predictions(prediction))
```

3. スクリプトを実行して、予測をサービスに送信します。

```
python3 tensorflow-model-server-infer.py
```

出力例は次のとおりです。

```
[[(u'n02123045', u'tabby', 0.68817204), (u'n02127052', u'lynx', 0.12701613),
 (u'n02123159', u'tiger_cat', 0.08736559), (u'n02124075', u'Egyptian_cat',
 0.063844085), (u'n02128757', u'snow_leopard', 0.009240591)]]
```

Amazon EKS で AI/ML ワークロードの高速コンピューティングを管理する

> **Tip**
>
> 今後開催予定の Amazon EKS AI/ML ワークショップに登録してください。

このセクションでは、Amazon EKS を使用して AI/ML トレーニングおよび推論ワークロード用の EC2 高速コンピューティングインスタンスを購入およびプロビジョニングする方法について説明します。大規模なモデルをトレーニングする場合でも、リアルタイム推論を実行する場合でも、生成 AI アプリケーションをデプロイする場合でも、適切な NVIDIA GPU または AWS Trainium キャパシティを使用することは、ワークロードのパフォーマンスの基盤となります。

EC2 インスタンスタイプから選択

利用可能な Amazon EC2 高速コンピューティングインスタンスの詳細については、「Amazon EC2 高速コンピューティングインスタンスの仕様」を参照してください。これには、P ファミリーと G ファミリーの NVIDIA GPU インスタンス、および AWS 設計のアクセラレーター Trainium と Inferentia が含まれます。

EC2 購入オプションを理解する

ワークロードに必要な高速インスタンスがわかったならば、次のステップは、これらの高速インスタンスタイプを取得するために利用できる購入オプションを理解することです。AWS には、オンデマンドインスタンス、スポットインスタンス、ML のキャパシティブロック、オンデマンドキャパシティ予約 (ODCR) の 4 つのコンピューティングキャパシティ購入オプションがあります。各オプションは、さまざまなワークロードパターン、コストプロファイル、可用性要件に対応します。Amazon EC2 インスタンス購入オプションドキュメントでは、各オプションの仕組み、その料金モデル、および使用時期について説明します。

- オンデマンドインスタンス: キャパシティが存在する場合は、コミットメントなしで秒単位で支払います。開発、プロトタイプ作成、予測不可能な推論スケーリング、および中断リスクなしで即時のコンピューティングを必要とするワークロードに最適です。
- スポットインスタンス: 予備の EC2 キャパシティを使用することで、2 分間の中断通知で、オンデマンドと比較して最大 90% の節約を実現します。耐久性のあるストレージにチェックポイントを提供する耐障害性のあるワークロードに最適です: ハイパーパラメータの調整、定期的なチェックポイントによる分散トレーニング、バッチ推論とオフライン推論、データ前処理パイプライン。
- ML のキャパシティブロック: P ファミリーインスタンスと Trainium インスタンスを固定ウィンドウ (24 時間、最大 6 か月) に予約し、最大 8 週間前に予約します。キャパシティブロックは、GPU クラスターへの予測可能なアクセスを必要とする既知のタイムラインを持つ、計画された大規模なトレーニング実行、時間制限のあるファインチューニング実験、研究プロジェクトに使用します。
- オンデマンドキャパシティ予約 (ODCR): 長期契約なしで特定のアベイラビリティーゾーンに高速キャパシティを予約します。キャパシティが使用されているかどうかにかかわらず、標準のオンデマンド料金で請求されます。本番推論、SLA バウンドサービス、ビジネスクリティカルなアプリケーションに最適です。スケジュールの遅延やキャパシティの利用不可は許容されません。キャパシティブロックとは異なり、ODCR は P ファミリーインスタンスと G ファミリーインスタンスの両方をサポートします。

購入オプションをワークロード要件に一致させる

高速インスタンスタイプと購入オプションを理解したので、次のステップは適切な購入オプションをワークロード固有の要件に一致させることです。インスタンスタイプ、リージョン、タイミングの柔軟性が高いワークロードは、より多くの購入オプションとより低い料金の対象となります。

次のような要因に基づいて決定します:

- 戦略的重要性と SLA コミットメント

- 需要の予測可能性とスケジューリングの柔軟性
- リザーブドキャパシティに事前にコミットする意思
- インスタンスタイプ、リージョン、タイミングの柔軟性
- 中断とコスト削減の許容度

実際には、チームは複数の購入オプションを組み合わせて、ワークロードポートフォリオ全体でコスト、可用性、信頼性のバランスを取るハイブリッドアプローチを採用しています。記事「AWS で GPU キャパシティを取得する方法」には、さまざまなタイプのワークロードに適した購入オプションを選択するための決定ツリー、料金比較、実例が記載されています。

EC2 サービスクォータを確認する

EKS クラスターにキャパシティ購入オプションを実装する前に、使用する計画がある GPU インスタンスファミリーに十分な vCPU クォータが AWS アカウントにあることを確認してください。適切なクォータがない場合、Karpenter NodePools、EKS Auto Mode プロビジョニング、および EKS ノードグループは、選択した購入オプションに関係なく、高速コンピューティングノードを起動できません。

AWS は、インスタンスファミリーと購入モデルごとに個別の vCPU クォータを適用します。Amazon EC2 インスタンスタイプのクォータを確認して、高速コンピューティングインスタンスのデフォルトのクォータを理解します。

これらのクォータは、インスタンス数ではなく vCPU 数に基づいています。例えば、10 個の p6-b300.48xlarge インスタンスを起動するには、1,920 個の vCPU (10 × 192) が必要です。多くの場合、新しいアカウントではデフォルトの GPU クォータが 0 に設定されているため、インスタンスをデプロイしようとする前にクォータの引き上げをリクエストしてください。

キャパシティブロック予約の作成、オンデマンドインスタンスの起動、またはスポットリクエストの送信時にクォータの制限が発生した場合は、AWS サポートまたは AWS アカウントチームに連絡して要件について話し合い、ニーズに最適な高速コンピューティングキャパシティを確保するためのオプションを検討してください。

Amazon EKS で EC2 購入オプションを使用する

EC2 高速コンピューティング購入オプションを選択したら、キャパシティを使用するように Amazon EKS クラスターを設定します。Amazon EKS には 3 つのプロビジョニング方法があり、それぞれで制御と自動化のバランスが異なります。

- Amazon EKS Auto Mode: ノードを自動的にプロビジョニング、スケーリング、パッチ適用する AWS マネージドコンピューティング。プロビジョニングに組み込み Karpenter、および NVIDIA ドライバーとデバイスプラグインを含む Bottlerocket オペレーティングシステムを使用します。運用上のオーバーヘッドを最小限に抑えたマネージドインフラストラクチャが必要な場合に最適です。静的キャパシティのプロビジョニングと動的キャパシティのプロビジョニングの両方をサポートします。
- Karpenter (セルフマネージド): Amazon EKS クラスターにインストールして運用するオープンソースのアップストリームプロジェクト。EKS Auto Mode と同じプロビジョニングモデルを提供し、オペレーティングシステム、AMI、カーネルチューニング、およびノードライフサイクルを完全に制御できます。EKS Auto Mode がすぐに提供しない要件を持つプラットフォームチームに最適です。
- ノードグループ (マネージドおよびセルフマネージド): EC2 Auto Scaling Groups (ASG) によってバックアップされ、キャパシティは EC2 起動テンプレートを通じて事前に定義されます。既存の EKS マネージドノードグループまたはセルフマネージドノードグループを持つプラットフォームチーム、および既知の静的な高速コンピューティングフットプリントで予測可能なサイズ設定を持つトレーニングワークロードに最適です。

以下のページでは、各プロビジョニングオプションについて詳しく説明します。

- the section called “EKS Auto Mode と Karpenter”
- the section called “マネージドノードグループとセルフマネージドノード”

混合戦略：購入オプションを組み合わせる

単一の Amazon EKS クラスター内で複数のキャパシティ購入オプションを組み合わせるのが一般的です。このアプローチは、さまざまなワークロードを最適なキャパシティソースにルーティングすることで、コスト、可用性、信頼性を同時に最適化します。顧客は、3 つの EKS コンピューティング管理アプローチ (EKS Auto Mode、Karpenter、または Node Groups) のいずれかを使用してこのハイブリッド戦略を実装するか、同じクラスター内でそれらを組み合わせます。

EKS Auto Mode と Karpenter は常にリザーブドキャパシティ (ODCR とキャパシティブロック) を最初にプロビジョニングし、続いてスポットまたはオンデマンドをプロビジョニングします。このインスタンスプロビジョニングの優先度を組み合わせるには、重要なワークロードをリザーブドキャパシティにスケジューリングし、柔軟なワークロードをスポットインスタンスまたはオンデマンドインスタンスにスケジューリングします。Kubernetes ネイティブスケジューリングプリミティブを使用してワークロードのルーティングを制御します。`nodeSelector` は特定のキャパシティタイプを

ターゲットにし、テイントと許容値によって NVIDIA GPU ノードまたは AWS Trainium ノードを分離し、topologySpreadConstraints は高可用性のためにアベイラビリティーゾーン全体でワークロードを分散します。

適切に設計された Amazon EKS クラスターは、高速コンピューティング NodePools またはノードグループをリザーブドとバーストの 2 つのカテゴリに整理し、それぞれをキャパシティ戦略に最適なワークロードパターンに合わせます。以下に例を示します。

- リザーブドキャパシティ: gpu-reserved NodePool またはノードグループは、SLA バインドサービスおよび計画されたコンピューティング負荷の高いジョブ用に、リザーブドキャパシティ (ODCR とキャパシティブロック) で本番推論とスケジュールされた大規模トレーニングを実行します。この NodePool またはノードグループは、リアルタイム推論エンドポイント、本番モデルサービング、および予測可能なパフォーマンスで常時オンの GPU 可用性を必要とするビジネスクリティカルなアプリケーションなど、推論ワークロードと本番ワークロードを処理します。また、計画された分散トレーニング、大規模なファインチューニング実験、期間限定の研究プロジェクト、および開始時間と期間を事前に把握しているワークロードなど、スケジュールされたコンピューティング負荷の高いジョブもサポートします。
- バーストキャパシティ: gpu-burst NodePool またはノードグループは、実験、アドホックワークロード、バッチ処理を処理します。オンデマンドフォールバックを備え、スポットインスタンスをプライマリキャパシティタイプとして使用します。この組み合わせにより、耐障害性のあるワークロードのコスト削減を最大化し、スポットが利用できない場合のキャパシティを確保できます。この NodePool またはノードグループは、バッチオフライン推論、データ前処理パイプライン、モデル評価ジョブ、開発とプロトタイピング、予測不可能な推論スケーリング、短期間のデバッグセッション、チェックポイントを実装してスポットの中断を処理できるワークロード、または予約を正当化しないがリザーブドウィンドウを待つことができないワークロードを処理します。この NodePool またはノードグループ上のワークロードは、チェックポイントとグレースフルシャットダウンを実装して、2 分間のスポット中断ウィンドウ内のノード損失を処理します。

nodeSelector を使用してワークロードに必要なキャパシティタイプを指定します。karpenter.sh/capacity-type: [spot, on-demand, reserved]重みベースのプロビジョニングは、すべてのキャパシティプール全体でクラスターを効率的にスケーリングします。このアーキテクチャでは、コストを最適化しながら、実験的なノートブックから本番推論まで、多様な AI/ML ワークロードを単一の Amazon EKS クラスター内で実行できます。

EKS Auto Mode と Karpenter を使用して AI/ML ワークロードのコンピューティングを管理する

> **Tip**
>
> 今後開催予定の Amazon EKS AI/ML ワークショップに登録してください。

このセクションでは、Amazon EKS Auto Mode またはセルフマネージド Karpenter を使用して、AI トレーニングおよび推論ワークロードの高速コンピューティング (AWS Trainium、NVIDIA GPUs) を管理する方法について説明します。

EKS Auto Mode と Karpenter は、動的プロビジョニングと静的プロビジョニングの 2 つのプロビジョニングモードをサポートしています。動的プロビジョニングでは、EKS Auto Mode と Karpenter は、ワークロードがクラスター上でスケジュールされたときに、高速コンピューティングインスタンスをプロビジョニングしてスケーリングします。静的プロビジョニングでは、EKS Auto Mode と Karpenter は固定数のノードをプロビジョニングして維持します。動的プロビジョニングと静的プロビジョニングを同じクラスターで使用して、ワークロードの需要に応じてスケーリングしながら、一定のベースラインキャパシティプールを維持できます。

EKS Auto Mode と Karpenter は、4 つのキャパシティ購入オプション (オンデマンド、スポット、キャパシティブロック、ODCR) をすべてサポートし、常にリザーブドキャパシティを最初にプロビジョニングし、続いて Spot または On-Demand をプロビジョニングします。

EKS Auto Mode と Karpenter の比較

どちらのアプローチも NodePool API を共有しますが、運用所有権、リソース API、オペレーティングシステムのサポート、Spot 中断処理、設定の柔軟性が異なります。

機能	EKS Auto Mode	セルフマネージド Karpenter
適しているチーム	運用上のオーバーヘッドを最小限に抑えたマネージドインフラストラクチャを好むチーム	ノードのライフサイクル、AMI、OS チューニング、パッチ適用を完全に制御したいチーム。
運用モデル	AWS は、Karpenter コントローラー、GPU/Trainium ドライバー、デバイスプラグイン、OS パッチ適	Karpenter コントローラーをクラスターにインストールして運用し、独自の GPU/Trainium ドライバー、デ

機能	EKS Auto Mode	セルフマネージド Karpenter
	用、Spot 中断処理をプロビジョニングおよび管理します。	バイスプラグイン、AMI ライフサイクル、パッチ適用、Spot 中断処理を行います。
コンピューティングオプション	On-Demand、Spot、ODCR、機械学習用のキャパシティブロック	On-Demand、Spot、ODCR、機械学習用のキャパシティブロック
リソース API	`NodePool` (`karpenter.sh/v1`), `NodeClass` (`eks.amazonaws.com/v1`).	`NodePool` (`karpenter.sh/v1`), `EC2NodeClass` (`karpenter.k8s.aws/v1`).
ノードオペレーティングシステム	Bottlerocket のみ。NVIDIA GPU、AWS Trainium、EFA の依存関係が含まれています。	AL2023、Bottlerocket、Windows、または独自の AMI。
ノード存続期間	セキュリティパッチ適用の最大ノード存続期間は 21 日です。ワークロードはノードのローテーションを許容する必要があります。	NodePool `expireAfter` と中断予算を使用してノードのライフサイクルを定義します。
Spot 中断処理	ネイティブ。SQS キューまたはノード終了ハンドラーは必要ありません。	設定および有効化はユーザーの責任です。
高速コンテナプル	すべての G、P、および Trn ファミリーインスタンスに含まれる SOCI 並列プル	設定および有効化はユーザーの責任です。
EC2 プレイスメントグループ	クラスター、パーティション、スプレッド	クラスター、パーティション、スプレッド
ネットワークインターフェイスの設定	タイプ `interface` または `EFA-only` のインターフェイス設定ごと	タイプ `interface` または `EFA-only` のインターフェイス設定ごと

機能	EKS Auto Mode	セルフマネージド Karpenter
ノードの修復	デフォルトで有効、EKS ノードモニタリングエージェントが含まれています	オプションで有効、EKS ノードモニタリングエージェントセルフマネージド
料金	基盤となる EC2 インスタンスコストに加えて EKS Auto Mode 管理料金。	オープンソース。基盤となる EC2 インスタンスに対して料金が発生します。

一般的な AI/ML のよく知られているラベル

EKS Auto Mode と Karpenter は、インスタンスタイプをハードコーディングせずにワークロードをターゲットにするために NodePool `requirements` と Pod `nodeSelector` または `nodeAffinity` で使用できるインスタンスラベルを公開します。ラベルプレフィックスは 2 つで異なります。EKS Auto Mode は `eks.amazonaws.com/` を使用し、セルフマネージド Karpenter は `karpenter.k8s.aws/` を使用します。

以下の表は、NodePools で使用できる関連ラベルを示しています。EKS Auto Mode と Karpenter は、ワークロードのターゲティングにさらに使用できるプロビジョニングプロセスの一環として、Karpenter ドキュメントに記載されているラベルをノードにも適用します。

EKS Auto Mode

完全なリストについては、「EKS Auto Mode でサポートされているラベル」を参照してください。

ラベル	値の例	説明
`eks.amazonaws.com/instance-family`	`p5`	プロパティが類似していてリソース量が異なるインスタンスタイプ。
`eks.amazonaws.com/instance-category`	`p`	インスタンスカテゴリ。通常は生成番号の前の文字。
`eks.amazonaws.com/instance-generation`	`5`	カテゴリ内のインスタンスタイプ生成番号。

ラベル	値の例	説明
eks.amazonaws.com/instance-gpu-name	h100	インスタンス上の GPU の名前。
eks.amazonaws.com/instance-gpu-manufacturer	nvidia	GPU メーカーの名前。
eks.amazonaws.com/instance-gpu-count	8	インスタンス上の GPU の数。
eks.amazonaws.com/instance-gpu-memory	81920	GPU あたりのメモリのメビバイト。
karpenter.sh/capacity-type	reserved	キャパシティタイプ: spot、on-demand 、または reserved。
topology.kubernetes.io/zone	us-east-1a	アベイラビリティーゾーン。

Self-managed Karpenter

詳細なリストについては、「Karpenter のよく知られているラベル」を参照してください。

ラベル	値の例	説明
karpenter.k8s.aws/instance-family	p5	プロパティが類似していてリソース量が異なるインスタンスタイプ。
karpenter.k8s.aws/instance-category	p	インスタンスカテゴリ。通常は生成番号の前の文字。
karpenter.k8s.aws/instance-generation	5	カテゴリ内のインスタンスタイプ生成番号。
karpenter.k8s.aws/instance-gpu-name	h100	インスタンス上の GPU の名前。

ラベル	値の例	説明
karpenter.k8s.aws/instance-gpu-manufacturer	nvidia	GPU メーカーの名前。
karpenter.k8s.aws/instance-gpu-count	8	インスタンス上の GPU の数。
karpenter.sh/capacity-type	reserved	キャパシティタイプ: spot、on-demand 、または reserved。
topology.kubernetes.io/zone	us-east-1a	アベイラビリティーゾーン。
kubernetes.io/arch	amd64	CPU アーキテクチャ。

リザーブドキャパシティのスケジューリングラベル

EKS Auto Mode または Karpenter が予約にノードを起動すると、次のラベルが追加されます。ワークロードをルーティングするには、nodeSelector、ノードアフィニティ、または NodePool の要件でこれらを使用します。

- karpenter.sh/capacity-type: reserved、on-demand、または spot。ノードをバックアップするキャパシティを示します。
- karpenter.k8s.aws/capacity-reservation-id: ノードが起動された特定の予約 ID。
- karpenter.k8s.aws/capacity-reservation-type: ODCR では default、キャパシティブロックでは capacity-block。

次の例は、一般的なスケジューリングパターンを示しています。

Pod を 1 つの特定の予約にピン留めする (フォールバックなし):

```
spec:
  nodeSelector:
    karpenter.sh/capacity-type: reserved
    karpenter.k8s.aws/capacity-reservation-id: "cr-0123456789abcdef0"
```

ターゲット ODCR ノードのみ (キャパシティブロックではなく任意の ODCR):

```
spec:
  nodeSelector:
    karpenter.sh/capacity-type: reserved
    karpenter.k8s.aws/capacity-reservation-type: default
```

リザーブドキャパシティ (ODCR またはキャパシティブロック) をターゲットにします。

```
spec:
  nodeSelector:
    karpenter.sh/capacity-type: reserved
```

リザーブドを優先しますが、利用できない場合は Spot または On-Demand にフォールバックします。

```
spec:
  affinity:
    nodeAffinity:
      preferredDuringSchedulingIgnoredDuringExecution:
        - weight: 100
          preference:
            matchExpressions:
              - key: karpenter.sh/capacity-type
                operator: In
                values: ["reserved"]
```

予約の有効期限の動作

予約が終了したときに、ODCR とキャパシティブロックの動作が異なります。スケジューリングとチェックポイント戦略が、ワークロードをサポートする予約のタイプと一致していることを確認します。

ODCR

ODCR に起動されたインスタンスは、その ODCR に無期限に存在するわけではありません。ODCR は期限切れになるか、キャンセルされる可能性があるか、またはインスタンスを ODCR から手動で削除される可能性があります。これらのいずれかが発生し、インスタンスが ODCR に属していないことを EKS Auto Mode/Karpenter が検出した場合、ノードの `karpenter.sh/capacity-type` ラベルが `reserved` から `on-demand` に更新されます。インスタンスは標準の On-Demand キャパシティとして実行され続け、既存の Pod は中断なく実行され続けます。

> **Note**
>
> 厳密な `nodeSelector: karpenter.sh/capacity-type: reserved` でスケジュールされた Pod は、再度ラベルが付けられている場合、ノードにスケジュールされません。ワークロードが ODCR の有効期限またはキャンセル後も存続するには、`nodeSelector` の代わりに上記の `preferredDuringSchedulingIgnoredDuringExecution` パターンを使用します。

キャパシティブロック

ODCR とは異なり、キャパシティブロックには常に終了時間があり、EC2 は終了時間の 30 分前 (UltraServer インスタンスタイプの場合は 60 分前) にキャパシティブロックインスタンスを終了します。予約ウィンドウが閉じる前に、トレーニングジョブと推論ジョブが完了するか状態を保存するように計画してください。特定の `capacity-reservation-id` に対して厳密な `nodeSelector` を使用する Pod は、ブロックの有効期限が切れると `Pending` になり、他の場所では再スケジュールされません。キャパシティブロックの有効期間中にワークロードを他のキャパシティに移動する必要がある場合は、チェックポイントの作成と上記の柔軟なアフィニティパターンを組み合わせてください。

- ほとんどのインスタンスタイプでは、キャパシティブロックの終了時間の 30 分前まで、UltraServer インスタンスタイプでは終了時間の 60 分前まで、予約済みインスタンスを使用できます。
- EKS Auto Mode と Karpenter は、EC2 が終了を開始する 10 分前にキャパシティブロック内のノードのドレインを事前に開始するため、ワークロードにはチェックポイントの作成とグレースフルシャットダウンの時間があります。

静的キャパシティ NodePool

EKS Auto Mode と Karpenter は、静的キャパシティ NodePool をサポートしており、これはワークロードの需要に関係なく決まった数のノードを維持します。静的プールは、レイテンシーの影響を受けやすい推論のコールドスタート遅延を排除し、クラスターの最小限のインフラストラクチャフットプリントを確保できます。

静的キャパシティは、NodePool 上に `replicas` フィールドを設定することで構成されます。

考慮事項

- NodePool 上に `replicas` が設定されると、それを削除することはできません。単一の NodePool で静的キャパシティのプロビジョニングと動的キャパシティのプロビジョニングを切り替えることはできません。
- 静的キャパシティ NodePool は、統合の対象と見なされません。`limits.nodes` を `replicas` より大きく設定して、AMI のドリフトまたは有効期間中に一時的なスケーリングを許可します。
- 予測可能なアベイラビリティーゾーン (AZ) ディストリビューションでは、単一のプールで複数のゾーンにまたがるのではなく、AZ ごとに 1 つの静的キャパシティ NodePool を作成します。

EKS Auto Mode

以下の例は、デフォルトの EKS Auto Mode NodeClass を使用し、最大 6 個のノード (`limits.nodes`) にできる 4 個のノード (`replicas`) を持つ静的 NodePool を作成する静的キャパシティ NodePool を示しています。

```
apiVersion: karpenter.sh/v1
kind: NodePool
metadata:
  name: gpu-static-inference
spec:
  replicas: 4
  template:
    spec:
      nodeClassRef:
        group: eks.amazonaws.com
        kind: NodeClass
        name: default
      requirements:
        - key: "karpenter.sh/capacity-type"
          operator: In
          values: ["on-demand"]
        - key: "eks.amazonaws.com/instance-family"
          operator: In
          values: ["g6e"]
        - key: "topology.kubernetes.io/zone"
          operator: In
          values: ["us-east-1a"]
      taints:
        - key: nvidia.com/gpu
          value: "true"
          effect: NoSchedule
  limits:
```

```
    nodes: 6   # Allow temporary headroom during node replacement
```

Self-managed Karpenter

セルフマネージド Karpenter では、静的キャパシティはアルファ StaticCapacity 機能 (Karpenter バージョン v1.8 で導入) によってゲートされます。これは Helm 値で有効にする必要があります。

```
settings:
  featureGates:
    staticCapacity: true
```

NodePool は my-nodeclass という名前のカスタム EC2NodeClass を参照し、最大 6 個のノード (limits.nodes) にできる 4 個のノード (replicas) を持つ静的 NodePool を作成します。

```
apiVersion: karpenter.sh/v1
kind: NodePool
metadata:
  name: gpu-static-inference
spec:
  replicas: 4
  template:
    spec:
      nodeClassRef:
        group: karpenter.k8s.aws
        kind: EC2NodeClass
        name: my-nodeclass
      requirements:
        - key: "karpenter.sh/capacity-type"
          operator: In
          values: ["on-demand"]
        - key: "karpenter.k8s.aws/instance-family"
          operator: In
          values: ["g6e"]
        - key: "topology.kubernetes.io/zone"
          operator: In
          values: ["us-east-1a"]
      taints:
        - key: nvidia.com/gpu
          value: "true"
          effect: NoSchedule
  limits:
```

```
    nodes: 6   # Allow temporary headroom during node replacement
```

機械学習用のキャパシティブロック

機械学習用のキャパシティブロックを使用すると、定義された今後のウィンドウ用に P ファミリーインスタンスと Trainium インスタンスを予約できます。これらは前払いであるため、EKS Auto Mode と Karpenter はそれらを無料としてモデル化し、On-Demand と Spot よりも優先します。機械学習用のキャパシティブロックには、1 ~ 14 日、または最大 182 日 (6 か月) の 7 日間の倍数のキャパシティブロックの予約期間があります。

EKS Auto Mode または Karpenter で機械学習用のキャパシティブロックを使用するには、NodeClass のキャパシティ予約 ID を使用して `capacityReservationSelectorTerms` を設定します。機械学習用のキャパシティブロックでオープン予約一致を使用することはできません。条件では、選択する ID、タグのセット、またはインスタンス一致条件を指定できます。タグを指定すると、一致するタグを持つアカウントからアクセス可能なすべてのキャパシティ予約が選択されます。これは、所有者アカウント ID を指定することでさらに制限できます。

詳細な例については、Karpenter のドキュメントを参照してください。

EKS Auto Mode

キャパシティブロックの予約を参照する `NodeClass` を作成してから、それを使用する NodePool を作成します。

`consolidateAfter: Never` を設定すると、Karpenter はコストを削減したりワークロードをより効率的にパックしたりするために、ノードの置き換え、マージ、終了を試みません。これは、キャパシティが既に前払いされているため、キャパシティブロックに推奨されます。

```
apiVersion: eks.amazonaws.com/v1
kind: NodeClass
metadata:
  name: capacity-block-gpu
spec:
  capacityReservationSelectorTerms:
    - id: "cr-0123456789abcdef0"   # Your Capacity Block reservation ID
    # Alternative: select by tags
    # - tags:
    #     role: "production-inference"
    #   owner: "012345678901"
---
apiVersion: karpenter.sh/v1
```

```
kind: NodePool
metadata:
  name: gpu-capacity-block
spec:
  disruption:
    consolidationPolicy: WhenEmpty
    consolidateAfter: Never
  template:
    spec:
      nodeClassRef:
        group: eks.amazonaws.com
        kind: NodeClass
        name: capacity-block-gpu
      requirements:
        - key: "karpenter.sh/capacity-type"
          operator: In
          values: ["reserved"]
        - key: "eks.amazonaws.com/instance-family"
          operator: In
          values: ["p5", "p5e", "p5en", "p4d"]
      taints:
        - key: nvidia.com/gpu
          value: "true"
          effect: NoSchedule
```

Self-managed Karpenter

capacityReservationSelectorTerms に加えて、AMI、サブネット、セキュリティグループセレクタを含む EC2NodeClass を作成してから、それを使用する NodePool を作成します。

consolidateAfter: Never を設定すると、Karpenter はコストを削減したりワークロードをより効率的にパックしたりするために、ノードの置き換え、マージ、終了を試みません。これは、キャパシティが既に前払いされているため、キャパシティブロックに推奨されます。

```
apiVersion: karpenter.k8s.aws/v1
kind: EC2NodeClass
metadata:
  name: capacity-block-gpu
spec:
  amiSelectorTerms:
    - alias: al2023@latest
  subnetSelectorTerms:
    - tags:
```

```
        karpenter.sh/discovery: ml-cluster
  securityGroupSelectorTerms:
    - tags:
        karpenter.sh/discovery: ml-cluster
  capacityReservationSelectorTerms:
    - id: "cr-0123456789abcdef0" # Your Capacity Block reservation ID
    # Alternative: select by tags
    # - tags:
    #     role: "production-inference"
    #   owner: "012345678901"
---
apiVersion: karpenter.sh/v1
kind: NodePool
metadata:
  name: gpu-capacity-block
spec:
  disruption:
    consolidationPolicy: WhenEmpty
    consolidateAfter: Never
  template:
    spec:
      nodeClassRef:
        group: karpenter.k8s.aws
        kind: EC2NodeClass
        name: capacity-block-gpu
      requirements:
        - key: "karpenter.sh/capacity-type"
          operator: In
          values: ["reserved"]
        - key: "karpenter.k8s.aws/instance-family"
          operator: In
          values: ["p5", "p5e", "p5en", "p4d"]
      taints:
        - key: nvidia.com/gpu
          value: "true"
          effect: NoSchedule
```

On-Demand キャパシティ予約 (ODCR)

ODCR は、長期的なコミットメントなしに、特定のアベイラビリティーゾーン (AZ) のキャパシティを保証します。キャパシティが使用されているかどうかにかかわらず、標準 On-Demand 料金で請求されます。ODCR は、機械学習用のキャパシティブロックでサポートされていない G ファミリーイ

ンスタンスを含む、すべての NVIDIA GPU ファミリーをサポートします。ODCR は前払いであるため、EKS Auto Mode と Karpenter はそれらを無料としてモデル化し、On-Demand と Spot よりも優先します。

ODCR は予約の終了時に機械学習用のキャパシティブロックとは異なる動作をします。ODCR の有効期限が切れるかキャンセルされると、インスタンスは標準 On-Demand として実行され続けます。詳細については、「the section called “予約の有効期限の動作”」を参照してください。

EKS Auto Mode または Karpenter で ODCR を使用するには、NodeClass のキャパシティ予約条件を使用して `capacityReservationSelectorTerms` を設定します。条件では、選択する ID、タグのセット、またはインスタンス一致条件を指定できます。タグを指定すると、一致するタグを持つアカウントからアクセス可能なすべてのキャパシティ予約が選択されます。インスタンス一致条件を指定すると、一致する動作で予約が選択されます: open (互換性のあるすべてのインスタンスに一致) または targeted (明示的にターゲットされたインスタンスにのみ一致)。これは、所有者アカウント ID を指定することでさらに制限できます。

詳細な例については、Karpenter のドキュメントを参照してください。

EKS Auto Mode

`capacityReservationSelectorTerms` を使用して NodeClass を作成し、on-demand フォールバックで `reserved` を優先する NodePool を作成します。`topology.kubernetes.io/zone` を ODCR の AZ にピン留めします。

```
apiVersion: eks.amazonaws.com/v1
kind: NodeClass
metadata:
  name: odcr-gpu-production
spec:
  capacityReservationSelectorTerms:
    - id: "cr-0987654321fedcba0"
    # Alternative: select by tags
    # - tags:
    #     Purpose: "production-inference"
    #   owner: "012345678901"
---
apiVersion: karpenter.sh/v1
kind: NodePool
metadata:
  name: gpu-reserved-production
spec:
```

```
  disruption:
    consolidationPolicy: WhenEmpty
    consolidateAfter: Never
  template:
    spec:
      nodeClassRef:
        group: eks.amazonaws.com
        kind: NodeClass
        name: odcr-gpu-production
      requirements:
        - key: "karpenter.sh/capacity-type"
          operator: In
          values: ["reserved", "on-demand"]
        - key: "eks.amazonaws.com/instance-family"
          operator: In
          values: ["p5", "g6e"]
        - key: "topology.kubernetes.io/zone"
          operator: In
          values: ["us-east-1a"]
      taints:
        - key: nvidia.com/gpu
          value: "true"
          effect: NoSchedule
```

Self-managed Karpenter

`capacityReservationSelectorTerms` に加えて、`EC2NodeClass` を AMI、サブネット、セキュリティグループセレクタを使用して作成してから、NodePool を作成します。

```
apiVersion: karpenter.k8s.aws/v1
kind: EC2NodeClass
metadata:
  name: odcr-gpu-production
spec:
  amiSelectorTerms:
    - alias: al2023@latest
  subnetSelectorTerms:
    - tags:
        karpenter.sh/discovery: ml-cluster
  securityGroupSelectorTerms:
    - tags:
        karpenter.sh/discovery: ml-cluster
  capacityReservationSelectorTerms:
```

```
    - id: "cr-0987654321fedcba0"
---
apiVersion: karpenter.sh/v1
kind: NodePool
metadata:
  name: gpu-reserved-production
spec:
  disruption:
    consolidationPolicy: WhenEmpty
    consolidateAfter: Never
  template:
    spec:
      nodeClassRef:
        group: karpenter.k8s.aws
        kind: EC2NodeClass
        name: odcr-gpu-production
      requirements:
        - key: "karpenter.sh/capacity-type"
          operator: In
          values: ["reserved", "on-demand"]
        - key: "karpenter.k8s.aws/instance-family"
          operator: In
          values: ["p5", "g6e"]
        - key: "topology.kubernetes.io/zone"
          operator: In
          values: ["us-east-1a"]
      taints:
        - key: nvidia.com/gpu
          value: "true"
          effect: NoSchedule
```

On-Demand

On-Demand はデフォルトのキャパシティタイプであり、EKS Auto Mode および Karpenter の静的または動的なプロビジョニングで使用できます。NodePool で `karpenter.sh/capacity-type: on-demand` を設定することで、オンデマンドインスタンスを明示的にリクエストできます。EKS Auto Mode と Karpenter は、Pod のリソースリクエストを満たす最低価格のインスタンスを選択します。開発、プロトタイプ作成、予測不可能な推論スケーリング、および中断リスクなしで即時の可用性を必要とするワークロードには、On-Demand を使用します。

EKS Auto Mode

```
apiVersion: karpenter.sh/v1
kind: NodePool
metadata:
  name: gpu-ondemand
spec:
  template:
    spec:
      nodeClassRef:
        group: eks.amazonaws.com
        kind: NodeClass
        name: default
      requirements:
        - key: "karpenter.sh/capacity-type"
          operator: In
          values: ["on-demand"]
        - key: "eks.amazonaws.com/instance-family"
          operator: In
          values: ["g6", "g6e", "g7e"]
        - key: "eks.amazonaws.com/instance-gpu-manufacturer"
          operator: In
          values: ["nvidia"]
      taints:
        - key: nvidia.com/gpu
          value: "true"
          effect: NoSchedule
```

Self-managed Karpenter

```
apiVersion: karpenter.sh/v1
kind: NodePool
metadata:
  name: gpu-ondemand
spec:
  template:
    spec:
      nodeClassRef:
        group: karpenter.k8s.aws
        kind: EC2NodeClass
        name: default
      requirements:
        - key: "karpenter.sh/capacity-type"
```

```
        operator: In
        values: ["on-demand"]
      - key: "karpenter.k8s.aws/instance-family"
        operator: In
        values: ["g6", "g6e", "g7e"]
      - key: "karpenter.k8s.aws/instance-gpu-manufacturer"
        operator: In
        values: ["nvidia"]
    taints:
      - key: nvidia.com/gpu
        value: "true"
        effect: NoSchedule
```

Spot

Spot は、予備の EC2 キャパシティを使用することで、On-Demand と比較して最大 90% のコスト削減を実現します。AWS は、2 分間の中断通知でスポットインスタンスを再利用できます。NodePool 上で複数のインスタンスファミリーを一覧表示して可用性を最大化します。Spot ワークロードを PodDisruptionBudget とペアリングし、耐久性のあるストレージ (Amazon S3 または Amazon EFS) に定期的にチェックポイントを保存して、Pod がドレインウィンドウ中に状態を保存できるようにします。

Spot スポットは、耐障害性があり、再開可能なトレーニングや推論のワークロードに適しています。ここでは、大幅なコスト削減と引き換えに、一時的な中断が許容されます。

一般的な候補には、以下が含まれます。

- ハイパーパラメータのチューニングとスイープ: 中断した場合に再試行できる多くの短い並列トライアル。
- チェックポイントを使用した分散トレーニング: 状態を S3 または FSx に定期的に保存し、ノードの損失後に最後のチェックポイントから再開できる長時間実行されるジョブ。
- バッチ推論とオフライン推論: エンドツーエンドのレイテンシーが秒単位ではなく時間単位で測定されるデータセットに対する大規模なスコアリングジョブ。
- データ前処理と特徴量エンジニアリングパイプライン: 大規模なデータセットでの並列変換。
- モデル評価とベンチマーク: べき等の結果を生成する反復可能なジョブ。
- 開発、プロトタイプ、ノートブック: ユーザーがときどき再起動を許容できるインタラクティブな実験。

レイテンシーの影響を受けやすいリアルタイム推論、SLA バインドされた本番稼働用エンドポイント、チェックポイントがない、または再起動を許容できないワークロードには Spot の使用を避けてください。

NodePool で `karpenter.sh/capacity-type: spot` を設定することで、スポットインスタンスを明示的にリクエストできます。

EKS Auto Mode

EKS Auto Mode は Spot 中断をネイティブに処理します。SQS キューまたはノード終了ハンドラーは必要ありません。

```
apiVersion: karpenter.sh/v1
kind: NodePool
metadata:
  name: gpu-spot
spec:
  disruption:
    budgets:
      - nodes: 10%
    consolidationPolicy: WhenEmpty
    consolidateAfter: 1h
  template:
    spec:
      nodeClassRef:
        group: eks.amazonaws.com
        kind: NodeClass
        name: default
      requirements:
        - key: "karpenter.sh/capacity-type"
          operator: In
          values: ["spot"]
        - key: "eks.amazonaws.com/instance-family"
          operator: In
          values: ["g6", "g6e", "g7e"]
        - key: "eks.amazonaws.com/instance-gpu-manufacturer"
          operator: In
          values: ["nvidia"]
      taints:
        - key: nvidia.com/gpu
          value: "true"
          effect: NoSchedule
  limits:
```

```
    resources:
      nvidia.com/gpu: "64"
```

Self-managed Karpenter

セルフマネージド Karpenter では、EC2 Spot 中断イベントと再調整レコメンデーションイベントを受信する SQS キューである中断キューを設定することで、Karpenter コントローラー上で (NodePool 上ではなく) ネイティブ中断処理を有効にする必要があります。これをインストール時に 1 回設定します。

Helm で Karpenter を直接インストールする場合は、`settings.interruptionQueue` で `values.yaml` を設定します。

```
# karpenter values.yaml (Helm)
settings:
  clusterName: my-cluster
  interruptionQueue: my-queue   # Name of the SQS queue receiving Spot events
```

`eksctl` で Karpenter をブートストラップする場合は、クラスター設定ファイルに `withSpotInterruptionQueue: true` を設定します。`eksctl` は SQS キューと EventBridge ルールを作成し、それらを使用するように Karpenter コントローラーを設定します。

```
# eksctl ClusterConfig
karpenter:
  version: "${KARPENTER_VERSION}"
  withSpotInterruptionQueue: true
```

コントローラーがキューを使用するようにセットアップされると、個々の NodePool リソースに追加の設定は必要ありません。中断処理はクラスター全体に適用されます。

```
apiVersion: karpenter.sh/v1
kind: NodePool
metadata:
  name: gpu-spot
spec:
  disruption:
    budgets:
      - nodes: 10%
    consolidationPolicy: WhenEmpty
    consolidateAfter: 1h
```

```
  template:
    spec:
      nodeClassRef:
        group: karpenter.k8s.aws
        kind: EC2NodeClass
        name: default
      requirements:
        - key: "karpenter.sh/capacity-type"
          operator: In
          values: ["spot"]
        - key: "karpenter.k8s.aws/instance-family"
          operator: In
          values: ["g6", "g6e", "g7e"]
        - key: "karpenter.k8s.aws/instance-gpu-manufacturer"
          operator: In
          values: ["nvidia"]
      taints:
        - key: nvidia.com/gpu
          value: "true"
          effect: NoSchedule
  limits:
    resources:
      nvidia.com/gpu: "64"
```

ノードグループを使用して Amazon EKS 上で AI/ML ワークロードのコンピューティングリソースを管理する

> **Tip**
>
> 今後開催予定の Amazon EKS AI/ML ワークショップに登録してください。

このセクションでは、Amazon EKS マネージドノードグループまたはセルフマネージドノードを使用して、AI トレーニングおよび推論ワークロードの高速コンピューティング (AWS Trainium、NVIDIA GPUs) を管理する方法について説明します。

EKS マネージドノードグループとセルフマネージドノードは EC2 Auto Scaling Groups (ASG) を使用します。EKS マネージドノードグループには、ノードを作成、更新、削除するための専用の EKS API があり、ノード修復機能とライフサイクル終了フックも組み込まれています。EKS セルフマネージドノードは EC2 API を通じて直接デプロイおよび管理されます。

これらのオプションでは、インスタンスタイプ、必要数、スケーリング境界、および EC2 起動テンプレートを事前に定義します。EKS 以外のワークロードがあり、EC2 起動テンプレートを介した設定の一貫性を希望する場合は、EKS マネージドノードグループまたはセルフマネージドノードの使用を検討してください。EKS ノードグループは、高速コンピューティングフットプリントが事前にわかっているトレーニングやファインチューニングワークロードに適しています。EKS Auto Mode と Karpenter はどちらも静的キャパシティのプロビジョニングをサポートしていることに注意してください。詳細については、the section called “EKS Auto Mode と Karpenter” を参照してください。

EKS マネージドノードグループとセルフマネージドノードは、すべての高速コンピューティング購入オプション (オンデマンド、スポット、オンデマンドキャパシティ予約、ML のキャパシティブロック) をサポートします。キャパシティタイプごとに、それぞれに独自の起動テンプレート、インスタンスタイプ、スケーリング設定がある、個別のマネージドまたはセルフマネージドノードグループを作成します。これにより、異種動的プロビジョニングロジックを使用せずに、各キャパシティプールの明示的で ASG でバックアップされた制御を提供できます。

EKS マネージドノードグループとセルフマネージドノードの比較

EKS マネージドノードグループとセルフマネージドノードのどちらを選択するかは、必要なカスタマイズと制御のレベルによって異なります。EKS マネージドノードグループは EC2 起動テンプレートのカスタマイズのサブセットを許可しますが、セルフマネージドノードは EC2 起動テンプレートの全範囲をサポートします。ノードライフサイクルを自分でカスタマイズおよび管理する理由が特にない場合は、EKS マネージドノードグループから始めて、特定の要件によって必要になった場合にのみセルフマネージドノードに移行してください。

次の場合、マネージドノードグループを使用します: EKS がユーザーに代わって AMI 選択、ノードブートストラップ、ローリング更新、ノード修復、および正常なドレインワークフローを処理することを希望している。トレーニングおよび推論ワークロードに EKS Auto Mode または Karpenter を使用することを希望しない場合は、EKS マネージドノードグループが推奨の開始点です。ML にキャパシティブロックを使用する場合、EKS マネージドノードグループは、予約が終了する 40 分前にノードグループをドレインするスケジュールされたスケーリングポリシーを自動的に作成し、AWS ノード終了ハンドラーまたは独自のスケールダウンオートメーションを使用する必要がなくなります。サポートされている EKS 最適化 AMI を使用している場合、カーネルレベルまたは詳細な EC2 起動テンプレートのカスタマイズが不要な場合、および Kubernetes バージョンのノードアップグレードパスをよりシンプルにしたい場合は、EKS マネージドノードグループを使用してください。

次の場合、セルフマネージドノードグループを使用します: EC2 起動テンプレート、AMI、カーネルパラメータ、コンテナランタイム設定、またはカスタムブートストラップスクリプトを完全に制御する必要がある。一般的な ML シナリオには、Elastic Fabric Adapter (EFA) による分散トレーニング

のカーネルと NIC 設定のチューニングや、カスタムノードライフサイクルコントローラーとの統合などがあります。セルフマネージドノードを使用すると、必要なユーザーデータと IAM インスタンスプロファイルを柔軟に提供できますが、ユーザーは、更新、スケジュールされたスケーリングポリシー、AWS ノード終了ハンドラーなどのライフサイクルフックに対して責任を負います。

機械学習用のキャパシティブロックを使用して GPU を予約する

機械学習 (ML) 用のキャパシティブロックを使用すると、将来の日付で GPU インスタンスを時間制限付きトレーニングまたは推論ワークロード用に予約できます。詳細については、「Amazon EC2 ユーザーガイド」の「機械学習用のキャパシティブロック」を参照してください。

EKS マネージドノードグループとセルフマネージドノードを介してキャパシティブロック予約を使用できます。EC2 起動テンプレートの設定は、どちらの場合も同じです。ノード作成ワークフロー、スケールダウン動作、およびワークロード終了のライフサイクルフックは、プロビジョニングオプションによって異なります。

考慮事項

- キャパシティブロックは、特定の Amazon EC2 インスタンスタイプおよび AWS リージョンでのみ使用できます。詳細については、「キャパシティブロックの操作の前提条件」を参照してください。
- キャパシティブロックはゾーン単位です。ノードグループの作成時に、キャパシティブロックの予約と同じアベイラビリティーゾーン (AZ) のサブネットを使用する必要があります。
- キャパシティブロックの予約がアクティブになる前にノードグループを作成する場合は、ノードグループの作成時に希望するキャパシティを `0` に設定します。
- 正常なワークロードドレイニングに時間をかけるには、キャパシティブロックの予約が終了する 30 分以上前にゼロへのスケーリングをスケジュールします。EC2 は、予約終了時間の 30 分前にインスタンスのシャットダウンを開始します。

機械学習用のキャパシティブロックを使用してノードグループを作成する

EKS マネージドノードグループとセルフマネージドノードでは、キャパシティブロックの予約をターゲットとするカスタム EC2 起動テンプレートを使用する必要があります。以下は、EKS マネージドノードグループとセルフマネージドノードの最小必須フィールドを示しています。以下のセルフマネージドノードのステップに示すように、セルフマネージドノードには追加のフィールドが必要です。

`LaunchTemplateData` には以下が含まれている必要があります:

- MarketType が "capacity-block" に設定された InstanceMarketOptions
- CapacityReservationId がキャパシティブロック ID に設定された CapacityReservationSpecification: CapacityReservationTarget。例えば、*cr-0123456789abcdef0*。
- キャパシティブロックの予約のインスタンスタイプに設定された InstanceType。例えば、*p5.48xlarge*。

これらの要件は、EKS マネージドノードグループとセルフマネージドノードの起動テンプレートを作成するための以下の例に示されています。

Managed node groups

1. 次の内容で、eks-capacity-block-lt.json という名前のファイルを作成します。

   CapacityReservationId と InstanceType の内容をキャパシティブロックの値に置き換えます。追加の EC2 起動テンプレートフィールドの詳細については、「the section called “起動テンプレート”」および「機械学習用のキャパシティブロックワークロードを使用する」を参照してください。

   ```
   {
     "LaunchTemplateData": {
       "InstanceMarketOptions": {
         "MarketType": "capacity-block"
       },
       "CapacityReservationSpecification": {
         "CapacityReservationTarget": {
           "CapacityReservationId": "cr-0123456789abcdef0"
         }
       },
       "InstanceType": "p5.48xlarge"
     }
   }
   ```

2. 起動テンプレートを作成します。

   ```
   aws ec2 create-launch-template \
       --launch-template-name EKS-Capacity-Block-Launch-Template \
       --launch-template-data file://eks-capacity-block-lt.json
   ```

3. 起動テンプレートを使用して、EKS マネージドノードグループを作成します。以下のコマンドのプレースホルダーを、環境に適した値に置き換えます。以下のコマンドは、`--ami-type` を AL2023 EKS 最適化 NVIDIA AMI に設定します。使用可能な EKS 最適化 AMI の詳細については、「the section called “EKS Linux GPU AMI を使用する”」を参照してください。EKS マネージドノードグループでカスタム AMI を使用するには、起動テンプレートで AMI ID を指定します。

   キャパシティブロックを使用する EKS マネージドノードグループを作成するときは、以下の操作を行います。

   - `--capacity-type` を `"CAPACITY_BLOCK"` に設定します。
   - キャパシティ予約と同じアベイラビリティーゾーンにあるサブネットのみを指定します。
   - 予約がアクティブになる前にゼロ以外の `desiredSize` を指定した場合、Auto Scaling グループは予約がアクティブになるまで起動エラーをレポートします。アクティブになると、インスタンスが起動し、ASG はリクエストされた `desiredSize` までスケールアップします。

   ```
   aws eks create-nodegroup \
       --cluster-name my-eks-cluster \
       --nodegroup-name eks-cb-nodes \
       --node-role "arn:aws:iam::111122223333:role/myNodeRole" \
       --region region-code \
       --subnets subnet-ExampleID1 \
       --ami-type "AL2023_x86_64_NVIDIA" \
       --scaling-config minSize=0,maxSize=2,desiredSize=0 \
       --capacity-type "CAPACITY_BLOCK" \
       --launch-template name="EKS-Capacity-Block-Launch-Template"
   ```

4. 作成時に `desiredSize` を `0` に設定すると、予約がアクティブになって以下のいずれかを使用できるようになったたときに、ノードグループをスケールアップします。

   - 予約開始時間に合わせて ASG でスケジュールされたスケーリングポリシー。詳細については、「Amazon EC2 Auto Scaling ユーザーガイド」の「Amazon EC2 Auto Scaling のスケジュールされたスケーリング」を参照してください。
   - スケーリング設定を更新するための Amazon EKS コンソールまたは `aws eks update-nodegroup-config`。

5. スケールアップ後にノードがクラスターを結合することを確認します。

6. EKS は、Amazon EKS Node Group Capacity Scaledown Before Reservation End という名前のスケジュールされたスケーリングポリシーを自動的に作成し、予約が終了する 40 分前に

ノードグループを 0 にスケールダウンします。これにより、EC2 が 30 分のマークでインスタンスの終了を開始する前に、Pod を適切にドレインする時間を確保できます。このスケジュールされたアクションを編集または削除しないでください。

Self-managed nodes

1. 次の内容で、eks-capacity-block-lt.json という名前のファイルを作成します。

   CapacityReservationId と InstanceType の内容をキャパシティブロックの値に置き換えます。追加の EC2 起動テンプレートフィールドの詳細については、「the section called “起動テンプレート”」および「機械学習用のキャパシティブロックワークロードを使用する」を参照してください。

   IamInstanceProfile、ImageId、SecurityGroupIds、UserData、KeyName の内容を、お使いの環境の値で置き換えます。

   ```
   {
     "LaunchTemplateData": {
       "InstanceMarketOptions": {
         "MarketType": "capacity-block"
       },
       "CapacityReservationSpecification": {
         "CapacityReservationTarget": {
           "CapacityReservationId": "cr-0123456789abcdef0"
         }
       },
       "IamInstanceProfile": {
         "Arn": "arn:aws:iam::111122223333:role/myNodeRole"
       },
       "ImageId": "image-id",
       "InstanceType": "p5.48xlarge",
       "KeyName": "key-name",
       "SecurityGroupIds": "sg-05b1d815d1EXAMPLE"
       ],
       "UserData": "user-data"
     }
   }
   ```

2. 起動テンプレートを作成します。

   ```
   aws ec2 create-launch-template \
   ```

```
    --launch-template-name EKS-Capacity-Block-Launch-Template \
    --launch-template-data file://eks-capacity-block-lt.json
```

3. 起動テンプレートを使用して、the section called “Amazon Linux” の手順に従って Auto Scaling グループを作成します。予約がまだアクティブでない場合は、`DesiredCapacity` を `0` に設定します。キャパシティが予約されているアベイラビリティーゾーンのサブネットのみを指定します。
4. `DesiredCapacity` を `0` に設定してセルフマネージドノードを作成したら、キャパシティブロックの予約時間に合わせて Auto Scaling グループにスケジュールされたスケーリングポリシーを作成します。詳細については、「Amazon EC2 Auto Scaling のスケジュールされたスケーリング」を参照してください。

   予約したインスタンスは、予約終了時刻の 30 分前まで使用できます。Pod がドレインする時間を確保できるように、終了時刻の 30 分以上前にゼロへのスケーリングをスケジュールします。

   手動でスケールする場合は、予約の開始時刻に ASG の希望するキャパシティを更新し、終了時刻の 30 分以上前に再度更新します。
5. Pod を適切にドレインするには、AWS ノード終了ハンドラーをセットアップします。このハンドラーは、EventBridge を使用して Amazon EC2 Auto Scaling からの ASG スケールインライフサイクルイベントを監視し、インスタンスが使用できなくなる前に Kubernetes コントロールプレーンが必要なアクションを実行できるようにします。そうしないと、Pod と Kubernetes のオブジェクトが保留状態でスタックする可能性があります。詳細については、GitHub の「AWS ノード終了ハンドラー」を参照してください。

   ノード終了ハンドラーをセットアップしない場合は、正常にドレインさせる時間を十分に確保できるように、30 分のウィンドウに入る前に Pod のドレインを手動で行います。

Amazon EKS で P6e-GB200 UltraServer を使用する

このトピックでは、P6e-GB200 UltraServer で Amazon EKS を設定して使用する方法について説明します。4 基の NVIDIA Blackwell GPU を搭載した `p6e-gb200.36xlarge` インスタンスタイプは、P6e-GB200 UltraServer としてのみ使用できます。P6e-GB200 UltraServer には 2 つのタイプがあります。`u-p6e-gb200x36` UltraServer には 9 個の `p6e-gb200.36xlarge` インスタンスがあり、`u-p6e-gb200x72` UltraServer には 18 個の `p6e-gb200.36xlarge` インスタンスがあります。

詳細については、Amazon EC2 P6e-GB200 UltraServer のウェブページを参照してください。

考慮事項

- Amazon EKS は、Kubernetes バージョン 1.33 以降で P6e-GB200 UltraServer をサポートしています。この Kubernetes バージョンリリースでは、EKS および AL2023 EKS 最適化高速 AMI でデフォルトで有効になっている動的リソース割り当て (DRA) がサポートされています。DRA は、P6e-GB200 UltraServer を EKS で使用するための要件です。DRA は、Karpenter 静的キャパシティプロビジョニング、EKS マネージド型ノードグループ、およびセルフマネージドノードでサポートされていますが、EKS Auto Mode または Karpenter 動的キャパシティプロビジョニングではサポートされていません。
- P6e-GB200 UltraServer は、EC2 Capacity Blocks for ML を通じて利用できます。キャパシティブロックを使用して EKS ノードを起動する方法については、「the section called “容量管理”」を参照してください。
- キャパシティブロックを使用して EKS マネージド型ノードグループを使用する場合は、カスタム起動テンプレートを使用する必要があります。P6e-GB200 UltraServer で EKS マネージド型ノードグループをアップグレードする場合は、アップグレードする前にノードグループの希望するサイズを `0` に設定する必要があります。
- EKS 最適化高速 AMI の AL2023 ARM NVIDIA バリアントを使用することをお勧めします。この AMI には、P6e-GB200 UltraServer を操作するために必要なノードコンポーネントと設定が含まれています。独自の AMI を構築する場合は、ドライバーを含むノードとシステムソフトウェアの互換性をインストールして検証する責任があります。詳細については、the section called “EKS Linux GPU AMI を使用する” を参照してください。
- NVIDIA ドライバーバージョン 580 を含む EKS 最適化 AMI リリース `v20251103` 以降を使用することをお勧めします。この NVIDIA ドライバーバージョンでは、潜在的なメモリの過剰レポートに対処するために、コヒーレントなドライバーベースのメモリ管理 (CDMM) が有効になります。CDMM が有効になっている場合、NVIDIA マルチインスタンス GPU (MIG) と vGPU の機能はサポートされていません。CDMM の詳細については、「NVIDIA Coherent Driver-based Memory Management (CDMM)」を参照してください。
- EKS 最適化 AL2023 NVIDIA AMI で NVIDIA GPU Operator を使用する場合は、ドライバーとツールキットのオペレータインストールを無効にする必要があります。これらは既に AMI に含まれているためです。EKS 最適化 AL2023 NVIDIA AMI には、NVIDIA Kubernetes デバイスプラグインまたは NVIDIA DRA ドライバーが含まれていないため、これらは個別にインストールする必要があります。
- 各 `p6e-gb200.36xlarge` インスタンスは、最大 17 枚のネットワークカードで構成でき、UltraServer 間の通信に EFA を活用できます。ワークロードのネットワークトラフィックは UltraServer をまたぐことができますが、最高のパフォーマンスを得るには、UltraServer 内の

GPU 通信に IMEX を活用し、同じ UltraServer にワークロードをスケジュールすることをお勧めします。詳細については、「P6e-GB200 インスタンスの EFA 設定」を参照してください。

- 各 `p6e-gb200.36xlarge` インスタンスには、7.5 TB のインスタンスストアストレージが 3 つあります。デフォルトでは、EKS 最適化 AMI はインスタンスストアをフォーマットおよびマウントしません。ノードのエフェメラルストレージは、エフェメラルストレージをリクエストするポッドと、ノードにダウンロードされるコンテナイメージとの間で共有できます。AL2023 EKS 最適化 AMI を使用している場合は、NodeConfig のインスタンスローカルストレージポリシーを RAID0 に設定することで、ユーザーデータのノードブートストラップの一部として設定できます。RAID0 に設定すると、インスタンスストアがストライプ化され、コンテナランタイムおよび kubelet がこのエフェメラルストレージを使用するように設定されます。

コンポーネント

P6e-GB200 UltraServer を使用して EKS でワークロードを実行するには、次のコンポーネントをお勧めします。オプションで、NVIDIA GPU Operator を使用して NVIDIA ノードコンポーネントをインストールできます。EKS 最適化 AL2023 NVIDIA AMI で NVIDIA GPU Operator を使用する場合は、ドライバーとツールキットのオペレータインストールを無効にする必要があります。これらは既に AMI に含まれているためです。

スタック	コンポーネント
EKS 最適化高速 AMI	カーネル 6.12 または 6.18
	NVIDIA GPU ドライバー
	NVIDIA CUDA ユーザーモードドライバー
	NVIDIA Container Toolkit
	NVIDIA ファブリックマネージャー
	NVIDIA IMEX ドライバー
	NVIDIA NVLink サブネットマネージャー
	EFA driver
ノードで実行されているコンポーネント	VPC CNI

スタック	コンポーネント
	EFA DRA ドライバーまたは EFA デバイスプラグイン
	NVIDIA K8s デバイスプラグイン
	NVIDIA DRA ドライバー
	NVIDIA Node Feature Discovery (NFD)
	NVIDIA GPU Feature Discovery (GFD)

上記の表のノードコンポーネントは、次の機能を実行します。

- VPC CNI: EKS で実行されているポッドのプライマリネットワークインターフェイスとして VPC IP を割り当てます
- EFA DRA ドライバーまたは EFA デバイスプラグイン: EKS で実行されているポッドのセカンダリネットワークとして EFA デバイスを割り当てます。P6e-GB200 UltraServer 間のネットワークトラフィックを担当します。マルチノードワークロードの場合、UltraServer 内の GPU 間トラフィックはマルチノード NVLink を介して送信されます。Kubernetes 1.34 以降では EFA DRA ドライバーが推奨されます。これは、トポロジー認識割り当てとデバイス共有を提供します。EFA デバイスプラグインは、すべての Kubernetes バージョンでサポートされます。詳細については、「the section called “EFA デバイス”」を参照してください。
- NVIDIA Kubernetes デバイスプラグイン: EKS で実行されているポッドのデバイスとして GPU を割り当てます。NVIDIA DRA ドライバー の GPU 割り当て機能が実験段階を終えるまで、NVIDIA Kubernetes デバイスプラグインを使用することをお勧めします。最新情報については、NVIDIA DRA ドライバーのリリースを参照してください。
- NVIDIA DRA ドライバー: P6e-GB200 UltraServer で実行されているワークロードに従う IMEX ドメインの作成を容易にする ComputeDomain カスタムリソースを有効にします。
  - ComputeDomain リソースは、Internode Memory Exchange (IMEX) ドメインを記述します。ComputeDomain の ResourceClaim を持つワークロードがクラスターにデプロイされると、NVIDIA DRA ドライバーは、一致するノードで実行される IMEX DaemonSet を自動的に作成し、ワークロードが開始される前にノード間で IMEX チャネル (複数可) を確立します。IMEX の詳細については、「マルチノード NVLink システム向け NVIDIA IMEX の概要」を参照してください。

  - NVIDIA DRA ドライバーは、ネットワークトポロジと NVLink ドメインに関する情報を伝達する NVIDIA GFD によって適用される clique ID ラベル (`nvidia.com/gpu.clique`) を使用します。
  - ワークロードジョブごとに ComputeDomain を作成するのがベストプラクティスです。
- NVIDIA Node Feature Discovery (NFD): 検出されたノードレベルの属性に基づいて GFD がノードラベルを適用するために必要な依存関係。
- NVIDIA GPU Feature Discovery (GFD): `nvidia.com/gpu.clique` という NVIDIA 標準のトポロジーラベルをノードに適用します。同じ `nvidia.com/gpu.clique` 内のノードにはマルチノードの NVLink 到達可能性があり、アプリケーションのポッドアフィニティを使用して、ポッドを同じ NVlink ドメインにスケジュールできます。

手順

次のセクションでは、Kubernetes バージョン 1.33 以降を実行している EKS クラスターと、AL2023 ARM NVIDIA EKS 最適化高速 AMI を実行している P6e-GB200 UltraServer を含む 1 つ以上のノードグループがあることを前提としています。EKS セルフマネージドノードとマネージドノードグループの前提条件の手順については、「the section called “容量管理”」にあるリンクを参照してください。

次の手順では、以下のコンポーネントを使用します。

名前	バージョン	説明
NVIDIA GPU Operator	25.3.4 以降	NVIDIA Kubernetes デバイスプラグインや NFD/GFD などの必要なプラグインのライフサイクル管理用。
NVIDIA DRA ドライバー	25.8.0 以降	ComputeDomain CRD と IMEX ドメイン管理用。
EFA DRA ドライバー (DRANET)	最新	トポロジー認識割り当てによる UltraServer 間通信用。Kubernetes 1.34 以降で推奨されます。
EFA デバイスプラグイン	0.5.14 以降	UltraServer 間通信用。すべての Kubernetes バージョンでサポートされます。

NVIDIA GPU Operator をインストールする

NVIDIA GPU Operator は、Kubernetes クラスターで GPU を使用するために必要なコンポーネントの管理を簡素化します。NVIDIA GPU ドライバーとContainer Toolkit は、EKS 最適化高速 AMI の一部としてインストールされるため、Helm 値設定では、これらを `false` に設定する必要があります。

1. 以下の設定を使用して、`gpu-operator-values.yaml` という名前の Helm 値ファイルを作成します。

```
devicePlugin:
  enabled: true
nfd:
  enabled: true
gfd:
  enabled: true
driver:
  enabled: false
toolkit:
  enabled: false
migManager:
  enabled: false
```

2. 前のステップで作成した `gpu-operator-values.yaml` ファイルを使用して、クラスターに NVIDIA GPU Operator をインストールします。

```
helm repo add nvidia https://helm.ngc.nvidia.com/nvidia
helm repo update
```

```
helm install gpu-operator nvidia/gpu-operator \
 --namespace gpu-operator \
 --create-namespace \
 --version v25.3.4 \
 --values gpu-operator-values.yaml
```

NVIDIA DRA ドライバーをインストールする

NVIDIA GPU Operator バージョン `v25.3.4` 以降では、NVIDIA DRA ドライバーを個別にインストールする必要があります。今後のリリースで変更される可能性があるため、NVIDIA GPU Operator のリリースノートを追跡することをお勧めします。

1. 以下の設定を使用して、dra-values.yaml という名前の Helm 値ファイルを作成します。NVIDIA GPU を搭載したノードにのみデプロイするように DRA ドライバーを設定する nodeAffinity と tolerations に注意してください。

```
resources:
  gpus:
    enabled: false # set to false to disable experimental gpu support
  computeDomains:
    enabled: true

controller:
  nodeSelector: null
  affinity: null
  tolerations: []

kubeletPlugin:
  affinity:
    nodeAffinity:
      requiredDuringSchedulingIgnoredDuringExecution:
        nodeSelectorTerms:
        - matchExpressions:
          - key: "nvidia.com/gpu.present"
            operator: In
            values:
            - "true"
  tolerations:
    - key: "nvidia.com/gpu"
      operator: Exists
      effect: NoSchedule
```

2. 前のステップで作成した dra-values.yaml ファイルを使用して、クラスターに NVIDIA DRA ドライバーをインストールします。

```
helm repo add nvidia https://helm.ngc.nvidia.com/nvidia
helm repo update
```

```
helm install nvidia-dra-driver-gpu nvidia/nvidia-dra-driver-gpu \
  --version="25.8.0" \
  --namespace nvidia-dra-driver-gpu \
  --create-namespace \
  -f dra-values.yaml
```

3. インストール後、DRA ドライバーは Kubernetes が `ComputeDomain` リソースを理解して割り当てられるようにする `DeviceClass` リソースを作成し、P6e-GB200 UltraServer 上の分散 GPU ワークロードに対する IMEX 管理を可能にします。

   次のコマンドを使用して、DRA リソースが使用可能であることを確認します。

```
kubectl api-resources | grep resource.k8s.io
```

```
deviceclasses           resource.k8s.io/v1  false        DeviceClass
resourceclaims          resource.k8s.io/v1  true         ResourceClaim
resourceclaimtemplates  resource.k8s.io/v1  true         ResourceClaimTemplate
resourceslices          resource.k8s.io/v1  false        ResourceSlice
```

```
kubectl get deviceclasses
```

```
NAME
compute-domain-daemon.nvidia.com
compute-domain-default-channel.nvidia.com
```

UltraServer 間通信用に EFA をインストールする

UltraServer 間で EFA 通信を使用するには、EFA DRA ドライバー (DRANET) または EFA デバイスプラグインをインストールします。P6e-GB200 インスタンスは最大 17 枚のネットワークカードで設定でき、プライマリ NCI (インデックス 0) は `interface` タイプで、最大 100 Gbps の ENA 帯域幅をサポートしている必要があります。ノードのプロビジョニング中に、要件に従って EFA インターフェイスと ENA インターフェイスを設定します。EFA 設定の詳細については、「P6e-GB200 インスタンスの EFA 設定」に関する AWS ドキュメントを参照してください。

> **Important**
>
> EFA DRA ドライバーと EFA デバイスプラグインを同じノードにインストールしないでください。2 つのメカニズムを同じノードに共存させることはできません。

オプション 1: EFA DRA ドライバー (DRANET) をインストールする

1. 以下の設定を使用して、`efa-values.yaml` という名前の Helm 値ファイルを作成します。

```
tolerations:
  - key: nvidia.com/gpu
    operator: Exists
    effect: NoSchedule
```

2. EKS Helm チャートリポジトリを追加し、EFA DRA ドライバーをインストールします。

```
helm repo add eks https://aws.github.io/eks-charts
helm repo update
```

```
helm install aws-dranet eks/aws-dranet --namespace kube-system -f efa-values.yaml
```

3. DRANET DaemonSet が実行されていることを確認します。

```
kubectl get daemonset -n kube-system aws-dranet
```

```
NAME          DESIRED   CURRENT   READY   UP-TO-DATE   AVAILABLE   NODE SELECTOR
 AGE
aws-dranet    2         2         2       2            2           <none>
 60s
```

4. DeviceClass オブジェクトと ResourceSlice オブジェクトが使用可能であることを確認します。

```
kubectl get deviceclass efa.networking.k8s.aws
```

```
NAME                    AGE
efa.networking.k8s.aws  60s
```

```
kubectl get resourceslices -l resource.k8s.io/driver=dra.net
```

トポロジー認識割り当てやデバイス共有など、EFA DRA ドライバーの使用の詳細については、「the section called “EFA デバイス”」を参照してください。

オプション 2: EFA デバイスプラグインをインストールする

1. 以下の設定を使用して、efa-values.yaml という名前の Helm 値ファイルを作成します。

```
tolerations:
  - key: nvidia.com/gpu
    operator: Exists
    effect: NoSchedule
```

2. EKS Helm チャートリポジトリを追加し、EFA デバイスプラグインをインストールします。

```
helm repo add eks https://aws.github.io/eks-charts
helm repo update
```

```
helm install efa eks/aws-efa-k8s-device-plugin -n kube-system -f efa-values.yaml
```

3. DaemonSet という EFA デバイスプラグインが実行されていることを確認します。

```
kubectl get daemonset -n kube-system aws-efa-k8s-device-plugin-daemonset
```

```
NAME                                  DESIRED   CURRENT   READY   UP-TO-DATE
 AVAILABLE   NODE SELECTOR   AGE
aws-efa-k8s-device-plugin-daemonset   2         2         2       2            2
      <none>          60s
```

4. 割り当て可能な EFA デバイスがノードにあることを確認します。例えば、各 NCI グループに 1 つの efa のみのインターフェイスでインスタンスを設定した場合、ノードごとに 4 つの割り当て可能な EFA デバイスが表示されることが予想されます。

```
kubectl get nodes "-o=custom-columns=NAME:.metadata.name,EFA:.status.allocatable.vpc
\.amazonaws\.com/efa"
```

```
NAME                                          EFA
ip-192-168-11-225.us-west-2.compute.internal   4
ip-192-168-24-96.us-west-2.compute.internal    4
```

マルチノード NVLink で IMEX を検証する

マルチノード NVLINK の NCCL テストおよびその他のマイクロベンチマークについては、awesome-distributed-training GitHub リポジトリを確認してください。次の手順は、nvbandwidth を使用してマルチノード NVLink テストを実行する方法を示しています。

1. NVL72 ドメイン内の 2 つのノードにまたがるマルチノード帯域幅テストを実行するには、まず MPI Operator をインストールします。

```
kubectl create -f https://github.com/kubeflow/mpi-operator/releases/download/v0.7.0/mpi-operator.yaml
```

2. テストマニフェストを定義する nvbandwidth-test-job.yaml という名前のファイルを作成します。マルチノード NVLink 到達可能性を持つ同じ NVLink ドメインにワーカーをスケジュールするための nvidia.com/gpu.clique ポッドアフィニティに注意してください。以下のサンプルでは、cuMemcpyAsync を使用してマルチノードデバイス間 CE Read memcpy テストを実行し、結果をログに出力します。

   NVIDIA DRA ドライバーバージョン v25.8.0 以降では、ComputeDomains は伸縮自在であり、ComputeDomain 定義で .spec.numNodes を 0 に設定できます。更新については、最新の NVIDIA DRA ドライバーのリリースノートを確認してください。

   ComputeDomain (IMEX チャネル) はノードごとに 1 つだけ設定できます。ComputeDomain リソースの allocationMode は All に変更しないでください。ComputeDomain とその ComputeDomain にアクセスするポッドの割り当てとスケジュールが正しく行われない可能性があります。詳細については、NVIDIA DRA ドライバーの問題 #353 を参照してください。

```
---
apiVersion: resource.nvidia.com/v1beta1
kind: ComputeDomain
metadata:
  name: nvbandwidth-test-compute-domain
spec:
  numNodes: 0 # This can be set to 0 from NVIDIA DRA Driver version v25.8.0+
  channel:
    resourceClaimTemplate:
      name: nvbandwidth-test-compute-domain-channel

---
apiVersion: kubeflow.org/v2beta1
kind: MPIJob
metadata:
  name: nvbandwidth-test
spec:
  slotsPerWorker: 4 # 4 GPUs per worker node
  launcherCreationPolicy: WaitForWorkersReady
  runPolicy:
```

```
    cleanPodPolicy: Running
  sshAuthMountPath: /home/mpiuser/.ssh
  mpiReplicaSpecs:
    Launcher:
      replicas: 1
      template:
        metadata:
          labels:
            nvbandwidth-test-replica: mpi-launcher
        spec:
          affinity:
            nodeAffinity:
              requiredDuringSchedulingIgnoredDuringExecution:
                nodeSelectorTerms:
                - matchExpressions:
                  # Only schedule on NVIDIA GB200/GB300 nodes
                  - key: node.kubernetes.io/instance-type
                    operator: In
                    values:
                    - p6e-gb200.36xlarge
                    - p6e-gb300.36xlarge
          containers:
          - image: ghcr.io/nvidia/k8s-samples:nvbandwidth-v0.7-8d103163
            name: mpi-launcher
            securityContext:
              runAsUser: 1000
            command:
            - mpirun
            args:
            - --bind-to
            - core
            - --map-by
            - ppr:4:node
            - -np
            - "8"
            - --report-bindings
            - -q
            - nvbandwidth
            - -t
            - multinode_device_to_device_memcpy_read_ce
    Worker:
      replicas: 2 # 2 worker nodes
      template:
        metadata:
```

```
      labels:
        nvbandwidth-test-replica: mpi-worker
    spec:
      affinity:
        nodeAffinity:
          requiredDuringSchedulingIgnoredDuringExecution:
            nodeSelectorTerms:
            - matchExpressions:
              # Only schedule on NVIDIA GB200/GB300 nodes
              - key: node.kubernetes.io/instance-type
                operator: In
                values:
                - p6e-gb200.36xlarge
                - p6e-gb300.36xlarge
        podAffinity:
          requiredDuringSchedulingIgnoredDuringExecution:
          - labelSelector:
              matchExpressions:
              - key: nvbandwidth-test-replica
                operator: In
                values:
                - mpi-worker
            topologyKey: nvidia.com/gpu.clique
      containers:
      - image: ghcr.io/nvidia/k8s-samples:nvbandwidth-v0.7-8d103163
        name: mpi-worker
        securityContext:
          runAsUser: 1000
        env:
        command:
        - /usr/sbin/sshd
        args:
        - -De
        - -f
        - /home/mpiuser/.sshd_config
        resources:
          limits:
            nvidia.com/gpu: 4  # Request 4 GPUs per worker
          claims:
          - name: compute-domain-channel # Link to IMEX channel
      resourceClaims:
      - name: compute-domain-channel
        resourceClaimTemplateName: nvbandwidth-test-compute-domain-channel
```

3. 次のコマンドを使用して、ComputeDomain を作成し、ジョブを開始します。

```
kubectl apply -f nvbandwidth-test-job.yaml
```

4. ComputeDomain の作成後、ワークロードの ComputeDomain に 2 つのノードがあることを確認できます。

```
kubectl get computedomains.resource.nvidia.com -o yaml
```

```
status:
  nodes:
  - cliqueID: <ClusterUUID>.<Clique ID>
    ipAddress: <node-ip>
    name: <node-hostname>
  - cliqueID: <ClusterUUID>.<Clique ID>
    ipAddress: <node-ip>
    name: <node-hostname>
  status: Ready
```

5. 次のコマンドを使用して、ジョブの結果を確認します。

```
kubectl logs --tail=-1 -l job-name=nvbandwidth-test-launcher
```

テストが成功すると、マルチノード memcpy テストの帯域幅統計が GB/秒で表示されます。成功したテスト出力の例を以下に示します。

```
...
nvbandwidth Version: ...
Built from Git version: ...

MPI version: ...
CUDA Runtime Version: ...
CUDA Driver Version: ...
Driver Version: ...

Process 0 (nvbandwidth-test-worker-0): device 0: NVIDIA GB200 (...)
Process 1 (nvbandwidth-test-worker-0): device 1: NVIDIA GB200 (...)
Process 2 (nvbandwidth-test-worker-0): device 2: NVIDIA GB200 (...)
Process 3 (nvbandwidth-test-worker-0): device 3: NVIDIA GB200 (...)
Process 4 (nvbandwidth-test-worker-1): device 0: NVIDIA GB200 (...)
Process 5 (nvbandwidth-test-worker-1): device 1: NVIDIA GB200 (...)
```

```
Process 6 (nvbandwidth-test-worker-1): device 2: NVIDIA GB200 (...)
Process 7 (nvbandwidth-test-worker-1): device 3: NVIDIA GB200 (...)

Running multinode_device_to_device_memcpy_read_ce.
memcpy CE GPU(row) -> GPU(column) bandwidth (GB/s)
           0         1         2         3         4         5         6         7
 0       N/A    821.45    822.18    821.73    822.05    821.38    822.61    821.89
 1    822.34       N/A    821.67    822.12    821.94    820.87    821.53    822.08
 2    821.76    822.29       N/A    821.58    822.43    821.15    821.82    822.31
 3    822.19    821.84    822.05       N/A    821.67    821.23    820.95    822.47
 4    821.63    822.38    821.49    822.17       N/A    821.06    821.78    822.22
 5    822.08    821.52    821.89    822.35    821.27       N/A    821.64    822.13
 6    821.94    822.15    821.68    822.04    821.39    820.92       N/A    822.56
 7    822.27    821.73    822.11    821.86    822.38    821.04    821.49       N/A

SUM multinode_device_to_device_memcpy_read_ce ...

NOTE: The reported results may not reflect the full capabilities of the platform.
Performance can vary with software drivers, hardware clocks, and system topology.
```

6. テストが完了したら、次のコマンドを使用してテストを削除します。

```
kubectl delete -f nvbandwidth-test-job.yaml
```

Amazon EKS でハードウェアデバイスを管理する

Amazon EKS は、EKS クラスターで特殊なハードウェアデバイスを管理できるように、動的リソース割り当て (DRA) とデバイスプラグインという 2 つの Kubernetes メカニズムをサポートしています。どちらも、ワークロードが NVIDIA GPU や AWS Trainium チップなどのハードウェアアクセラレーターや、Elastic Fabric Adapter (EFA) などのハイパフォーマンスネットワークデバイスにアクセスできるようにするメカニズムです。

Karpenter、EKS マネージドノードグループ、またはセルフマネージドノードで [静的キャパシティのプロビジョニング] を使用する場合は、Kubernetes バージョン 1.34 以降の新しいデプロイに DRA ドライバーを使用することをお勧めします。DRA は、現在 EKS Auto Mode ではサポートされていません。DRA は、デバイスプラグインでは不可能な、より豊富なデバイス選択、トポロジー認識スケジューリング、デバイス共有機能を提供します。

この 2 つの Kubernetes 機能に関する全般的な情報については、動的リソース割り当て と デバイスプラグイン に関する Kubernetes ドキュメントを参照してください。

動的リソース割り当てとデバイスプラグインの比較

Kubernetes デバイスプラグインは、これまで特殊なハードウェアを Kubernetes ワークロードに公開する際に第一に使用されてきたメカニズムです。ユーザーからのコンテナリソースのリクエストや制限を受けて、デバイスを拡張リソース (`nvidia.com/gpu` や `aws.amazon.com/neuroncore` など) としてアドバタイズします。広くサポートされ、使用されていますが、次のような制限があります。

- デバイスは、属性ベースのフィルタリングがない不透明な整数カウントとしてリクエストされます。
- コンテナまたはポッド間でデバイスを共有する機能はサポートされていません。
- トポロジー認識割り当てをデバイスタイプ全体で表現力豊かに行う機能はありません。
- インテリジェントに配置するには、多くの場合、カスタムスケジューラ拡張機能が必要です。

動的リソース割り当て (DRA) は、これらの制限に対処するために Kubernetes バージョン 1.34 で一般公開された Kubernetes 機能です。DRA により、デバイスドライバーは `ResourceSlice` オブジェクトを介して豊富なデバイス属性を Kubernetes スケジューラに発行します。デバイスをリクエストするには、`DeviceClass` カテゴリを参照する `ResourceClaim` オブジェクトおよび `ResourceClaimTemplate` オブジェクトを使用します。

DRA では次のことが可能になります。

- Common Expression Language (CEL) 式を使用して属性ベースでデバイスを選択する。
- トポロジ認識割り当てにより、複数のデバイスが同じ PCIe スイッチまたは NUMA ドメインに共存できるようになる。
- `ResourceClaim` リファレンスを共有することで、複数のコンテナやポッド間でデバイスを共有する。
- MIG またはタイムスライシングを使用する場合の NVIDIA GPU の動的パーティショニングと共有

Amazon EKS 向け DRA ドライバー

次の DRA ドライバーは、Amazon EKS クラスターで特殊なハードウェアデバイスを管理する場合によく使用されます。

NVIDIA DRA ドライバー

GitHub の GPU 用 NVIDIA DRA ドライバーは、NVIDIA GPU の柔軟な割り当てと動的な設定を可能にします。NVIDIA DRA ドライバーを使用した GPU の管理の詳細については the section called “DRA & デバイスプラグイン” を、EC2 Grace-Blackwell インスタンスを使用したマルチノード NVLink (MNNVL) ワークロードでの `ComputeDomains` の使用の詳細については the section called “EKS で P6e-GB200 を使用する” を参照してください。

EFA DRA ドライバー

EFA DRA ドライバー (GitHub の DRANET) は、EFA インターフェイスをトポロジー的にローカルな GPU または Neuron デバイスとペアリングするトポロジー認識スケジューリングを使用して Elastic Fabric Adapter (EFA) デバイスの割り当てを管理し、ポッド間でのデバイス共有をサポートします。詳細については、「the section called “EFA デバイス”」を参照してください。

Neuron DRA ドライバー

Neuron DRA ドライバーは、トポロジー認識スケジューリング、コネクテッドデバイスサブセット割り当て、Logical NeuronCore (LNC) 設定により、AWS Trainium と AWS Inferentia2 のデバイス割り当てを管理します。カスタムスケジューラ拡張機能は必要ありません。詳細については、「the section called “Neuron アクセラレーター”」を参照してください。

Amazon EKS 用のデバイスプラグイン

次のデバイスプラグインは、Amazon EKS クラスターで特殊なハードウェアデバイスを管理する場合によく使用されます。

NVIDIA デバイスプラグイン

GitHub の NVIDIA デバイスプラグイン は、NVIDIA GPU を `nvidia.com/gpu` 拡張リソースとしてアドバタイズし、GPU の正常性を追跡します。

EFA デバイスプラグイン

EFA デバイスプラグインは、各ノードで使用可能な EFA デバイスをすべて検出して、`vpc.amazonaws.com/efa` 拡張リソースとしてアドバタイズします。

Neuron デバイスプラグイン

Neuron デバイスプラグイン は、Neuron ハードウェアを `aws.amazon.com/neuroncore` および `aws.amazon.com/neuron` 拡張リソースとして公開します。各ノードで使用可能な Neuron デバイスを検出して、割り当て可能なリソースとしてアドバタイズし、そのライフサイクルを管理します。

考慮事項

Amazon EKS で DRA ドライバーを使用する前に、以下の考慮事項を検討してください。

- DRA は Kubernetes バージョン 1.33 以降の Amazon EKS で使用できますが、GitHub のアップストリーム Kubernetes の問題 があるため、Kubernetes バージョン 1.34 以降をお勧めします。クラスターコントロールプレーンとノードが、DRA をサポートする Kubernetes バージョンを実行している必要があります。
- DRA は、現在 EKS Auto Mode と互換性がありません。
- 動的にプロビジョニングされたキャパシティを使用する場合、DRA は現在 Karpenter と互換性がありません。DRA ドライバーでは、Karpenter の静的キャパシティプロビジョニング、EKS マネージドノードグループ、またはセルフマネージドノードを使用する必要があります。
- 同じノードで同じデバイスタイプの DRA ドライバーとデバイスプラグインを同時に実行することはできません。対応する DRA ドライバーをインストールする前に、デバイスプラグインをアンインストールするか、別のノードにデプロイします。同じノード上の同じデバイスに対して DRA ドライバーとデバイスプラグインの両方を実行すると、基盤となるハードウェアデバイスがサイレントオーバーサブスクリプションになる可能性があります。
- DRA で使用される Kubernetes API リソース (`ResourceClaim`、`ResourceClaimTemplate`、`DeviceClass`) は、デバイスプラグインのもの (`resource.limits`、`resource.requests`) とは異なります。DRA を使用すると、ワークロードの仕様を変更することなく、デバイスプラグインの拡張リソースを管理できます。DRA の Kubernetes リソースの詳細については、Kubernetes ウェブサイトの「Kubernetes ドキュメント」を参照してください。
- NVIDIA、EFA、および Neuron 用の DRA ドライバーは、EKS 最適化 AL2023 AMI と Bottlerocket AMI の両方と互換性があります。Bottlerocket で NVIDIA DRA ドライバーを使用している場合は、Bottlerocket NVIDIA バリアントに含まれている NVIDIA デバイスプラグインを無効にします。
- デバイスプラグインは、すべての Kubernetes バージョンで引き続き完全にサポートされます。

DRA の ResourceClaim と ResourceClaimTemplate

DRA を使用する場合は、`ResourceClaim` オブジェクトまたは `ResourceClaimTemplate` オブジェクトを使用してデバイスをリクエストします。これらの 2 つのリソースタイプは、目的も異なれば、ライフサイクルの動作も異なります。

ResourceClaim

ResourceClaim は、いずれのポッドからも独立して作成する名前付き Kubernetes オブジェクトです。resourceClaimName フィールドを使用してポッド仕様で名前で参照します。ResourceClaim には以下の特徴があります。

- ポッドで参照するには、ポッドの作成前にクラスターに存在している必要があります。クレームが存在しない場合、ポッドは保留状態のままになります。
- ポッドで参照しているかどうかに関係なく、明示的に削除するまで保持されます。
- 複数のポッドで同じ ResourceClaim を参照できるため、デバイス共有が可能です。同じクレームを参照するすべてのポッドで、割り当てられた同じデバイスへのアクセスを共有し、同じノードにスケジュールされます。

  ResourceClaim は、複数のポッドで同じデバイスへのアクセスを共有する必要がある場合や、単一のポッドのライフタイムを超えてクレームを保持する必要がある場合に使用します。

ResourceClaimTemplate

ResourceClaimTemplate は、各ポッドの固有の ResourceClaim を Kubernetes で自動的に生成するために使用するテンプレートを定義します。resourceClaimTemplateName フィールドを使用してポッド仕様で参照します。ResourceClaimTemplate 自体はいずれのポッドにもバインドされません。個別に保持される再利用可能なテンプレートです。ResourceClaimTemplate には以下の特徴があります。

- テンプレートを参照するポッドごとに、新しい ResourceClaim が Kubernetes で作成されます。各ポッドに独自のデバイスのセットが割り当てられます。
- 生成された ResourceClaim は、作成をトリガーしたポッドのライフサイクルにそれぞれバインドされます。ポッドが削除されると、それに関連する生成された ResourceClaim も削除されます。ResourceClaimTemplate 自体に影響はなく、以降のポッドに対する新しいクレームの生成に引き続き使用されます。

  ResourceClaimTemplate は、ワークロードの各ポッドに同様の設定を持つ独自の専用デバイスが必要な場合に使用します。例えば、各ポッドに独自の GPU デバイスまたは EFA デバイスが必要な並列実行を使用するジョブでは、ポッドに ResourceClaimTemplate を使用します。

次の表は、ResourceClaim と ResourceClaimTemplate の違いをまとめたものです。

動作	ResourceClaim	ResourceClaimTemplate
作成	ポッドで参照する前に手動で作成します。	ポッドごとに Kubernetes で自動的にクレームが生成されます。
ライフサイクル	削除するまで保持されます。	テンプレートは削除するまで保持されます。生成された `ResourceClaim` は、作成をトリガーしたポッドにそれぞれバインドされます。
ポッド間でのデバイス共有	サポート対象。複数のポッドで同じクレームを参照できます。	サポート外。各ポッドに個別のクレームが割り当てられます。
ポッド仕様のフィールド	`resourceClaimName`	`resourceClaimTemplateName`

`ResourceClaim` オブジェクトを使用してポッド間で EFA デバイスを共有する例については、「the section called “複数のポッド間で EFA デバイスを共有する”」を参照してください。`ResourceClaimTemplate` オブジェクトでトポロジー認識割り当てを使用する例については、「the section called “EFA と GPU/Neuron デバイスのトポロジー認識割り当て”」を参照してください。

トピック

- the section called “EFA デバイス”
- the section called “Neuron アクセラレーター”
- the section called “NVIDIA GPU”
- the section called “DRA & デバイスプラグイン”
- the section called “マルチインスタンス GPU (MIG)”
- the section called “タイムスライシング”

Amazon EKS で EFA デバイスを管理する

Elastic Fabric Adapter (EFA) は Amazon EC2 インスタンス用のネットワークデバイスであり、人工知能、機械学習、およびハイパフォーマンスコンピューティング (HPC) のワークロード用にハイパフォーマンスのノード間通信と RDMA (リモートダイレクトメモリアクセス) を実現します。Amazon EKS は、EKS クラスターで EFA デバイスを管理できるように、EFA 動的リソース割り当て (DRA) ドライバー (DRANET) と EFA デバイスプラグインという 2 つのメカニズムをサポートしています。

Karpenter、EKS マネージドノードグループ、またはセルフマネージドノードで [静的キャパシティのプロビジョニング] を使用する場合は、Kubernetes バージョン 1.34 以降の新しいデプロイに DRA ドライバーを使用することをお勧めします。DRA は、現在 EKS Auto Mode ではサポートされていません。EFA DRA ドライバーを使用すると、EFA インターフェイスをトポロジー的にローカルな GPU または Neuron デバイスとペアリングするトポロジー認識割り当てを設定し、ポッド間でデバイスを共有できます。

EFA DRA ドライバーと EFA デバイスプラグイン

機能	EFA DRA ドライバー	EFA デバイスプラグイン
最小限必要な Kubernetes バージョン	1.34	EKS でサポートされているすべての Kubernetes バージョン
EKS コンピューティング	Karpenter (静的キャパシティのみ)、マネージドノードグループ、セルフマネージドノード	EKS オートモード、Karpenter、マネージドノードグループ、セルフマネージドノード
EKS 最適化 AMI	AL2023、Bottlerocket	AL2023、Bottlerocket
デバイスアドバタイズ	デバイスタイプ、トポロジー、PCIe ローカリティなど、`ResourceSlice` オブジェクトによる豊富な属性	`vpc.amazonaws.com/efa` 拡張リソースの整数カウント
GPU-EFA アフィニティ	DRA ネイティブトポロジー認識	自動トポロジー認識 (EKS 最適化 AL2023 AMI のみ)

機能	EFA DRA ドライバー	EFA デバイスプラグイン
Neuron-EFA アフィニティ	DRA ネイティブトポロジー認識	自動トポロジー認識 (EKS 最適化 AL2023 AMI のみ)
デバイス共有	共有の `ResourceClaim` を参照することにより複数のポッドで同じ EFA デバイスを共有できる	サポート外。各 EFA デバイスは 1 つのポッドにのみ割り当てられる。

EFA インターフェイスを備えた EKS ノードを作成する

EFA インターフェイスを備えた EKS ノードを作成すると、インスタンスのプロビジョニング中に EFA インターフェイスがインスタンスにアタッチされます。デバイスごとの EFA 設定をカスタマイズし、EKS Auto Mode、Karpenter、EKS マネージドノードグループ、または EKS セルフマネージドノードグループでプレイスメントグループを使用できます。EKS Auto Mode では、各ネットワークインターフェイスの設定を `advancedNetworking.networkInterfaces` の `NodeClass` 経由で渡します。Karpenter では、`EC2NodeClass` を介して各ネットワークインターフェイスの設定を渡します。EKS マネージドノードグループまたはセルフマネージドノードでは、起動テンプレートを使用して各ネットワークインターフェイスの設定を渡します。

`eksctl` を使用して EKS ノードを `efaEnabled` 設定にプロビジョニングすると、すべてのインターフェイスがインターフェイスタイプ `EFA` で設定され、EFA 固有のセキュリティグループが作成されて、EFA デバイスプラグインがクラスターにインストールされます。`eksctl` の使用時にデバイスごとの EFA 設定をカスタマイズする必要がある場合は、起動テンプレートに eksctl サポートを使用することをお勧めします。

以下の例では、EFA インターフェイスを備えた `NodeClass` および起動テンプレートを設定する方法を示しています。これは、EFA トラフィックに使用するか標準の IP ベースのトラフィックに使用するかに応じて、インターフェイスをカスタマイズするのに役立ちます。各インスタンスタイプでサポートされる EFA インターフェイスの数、およびネットワーク帯域幅を最大化するための設定方法については、「Amazon EC2 ユーザーガイド」の「EFA 対応インスタンスタイプのネットワーク帯域幅を最大化する」を参照してください。

EKS 自動モード

EKS Auto Mode では、`NodeClass` (`eks.amazonaws.com/v1`) の `advancedNetworking.networkInterfaces` フィールドを使用して EFA ネットワークインターフェイスを設定します。各エントリでは、`networkCardIndex`、`deviceIndex`、および

interfaceType を指定します。interfaceType は、標準ネットワークインターフェイスの場合は interface になり、RDMA トラフィック専用で IP アドレスの割り当てがない EFA インターフェイスの場合は efa-only になります。

networkInterfaces が設定されている場合、NodeClass を参照する NodePool によって起動されたインスタンスは、ポッドが vpc.amazonaws.com/efa リソースをリクエストするかどうかに関係なく、この設定を使用します。EKS Auto Mode は、静的ネットワークインターフェイス設定を持つノードのインスタンス起動後に、追加の IP、プレフィックス、または ENI をアタッチしません。起動時に設定されたインターフェイスと IP のみが Pod で使用できます。

この機能は、静的キャパシティノードプールとともに使用することで、分散トレーニングおよび推論ワークロード用に事前ウォーミングされた EFA 対応ノードを維持できます。

P5 の EFA 専用インターフェイスを使用した EKS Auto Mode NodeClass の例

```
apiVersion: eks.amazonaws.com/v1
kind: NodeClass
metadata:
  name: efa-node-class
spec:
  role: MyNodeRole
  subnetSelectorTerms:
    - tags:
        Name: "private-subnet"
  securityGroupSelectorTerms:
    - tags:
        Name: "efa-security-group"
  placementGroupSelector:
    name: "ml-training-pg"
  advancedNetworking:
    networkInterfaces:
    - deviceIndex: 0
      interfaceType: interface
      networkCardIndex: 0
      secondaryIPv4PrefixCount: 1
    - networkCardIndex: 0
      deviceIndex: 1
      interfaceType: efa-only
    - networkCardIndex: 1
      deviceIndex: 0
      interfaceType: efa-only
    - networkCardIndex: 2
```

```
      deviceIndex: 0
      interfaceType: efa-only
    - networkCardIndex: 3
      deviceIndex: 0
      interfaceType: efa-only
```

静的ネットワークインターフェイス設定の制約の完全なリストについては、NodeClass ドキュメントの the section called “静的ネットワークインターフェイスの設定” を参照してください。

Karpenter

networkInterfaces の各エントリで、networkCardIndex、deviceIndex、および interfaceType を指定します。interfaceType は、標準ネットワークインターフェイスの場合は interface、RDMA トラフィック専用で IP アドレスの割り当てがない EFA インターフェイスの場合は efa-only になります。networkInterfaces が設定されている場合、NodeClass を参照する NodePool によって起動されたインスタンスは、ポッドが vpc.amazonaws.com/efa リソースをリクエストするかどうかに関係なく、この設定を使用します。

NodeClass で networkInterfaces を指定せずに Karpenter を使用している場合、vpc.amazonaws.com/efa をリクエストするポッド用にインスタンスを作成すると、すべてのインターフェイスがインターフェイスタイプ EFA で設定されます。

Karpenter v1.11 で EC2NodeClass の networkInterfaces 設定が追加されました。次の例は、1 つの ENA インターフェイスと 8 つの EFA 専用インターフェイスを備えた P6-B200 インスタンス用に設定された EC2NodeClass を示しています。

EFA 専用インターフェイスを備えた P6-B200 の Karpenter EC2NodeClass の例

```
apiVersion: karpenter.k8s.aws/v1
kind: EC2NodeClass
metadata:
  name: efa-node-class
spec:
  networkInterfaces:
  - networkCardIndex: 0
    deviceIndex: 0
    interfaceType: interface
  - networkCardIndex: 0
    deviceIndex: 1
    interfaceType: efa-only
  - networkCardIndex: 1
```

```
    deviceIndex: 0
    interfaceType: efa-only
  - networkCardIndex: 2
    deviceIndex: 0
    interfaceType: efa-only
  - networkCardIndex: 3
    deviceIndex: 0
    interfaceType: efa-only
  - networkCardIndex: 4
    deviceIndex: 0
    interfaceType: efa-only
  - networkCardIndex: 5
    deviceIndex: 0
    interfaceType: efa-only
  - networkCardIndex: 6
    deviceIndex: 0
    interfaceType: efa-only
  - networkCardIndex: 7
    deviceIndex: 0
    interfaceType: efa-only
```

EKS マネージドノードグループとセルフマネージドノード

EKS マネージドノードグループまたはセルフマネージドノードでは、起動テンプレートを使用して各ネットワークインターフェイスの設定を渡します。

次の例は、1 つの ENA インターフェイスと 8 つの EFA 専用インターフェイスを備えた P6-B200 インスタンス用に設定された起動テンプレートを示しています。プライマリネットワークインターフェイス (ネットワークカード 0、デバイスインデックス 0) では IP トラフィックに標準の `interface` タイプを使用し、その他のインターフェイスでは専用の RDMA トラフィックに `efa-only` を使用しています。`efa-only` インターフェイスの数は、インスタンスタイプに基づいて調整します。各インスタンスタイプでサポートされる EFA インターフェイスの数については、「Amazon EC2 ユーザーガイド」の「EFA 対応インスタンスタイプのネットワーク帯域幅を最大化する」を参照してください。

EFA 専用インターフェイスを備えた P6-B200 の起動テンプレートの例

`security-group-id` は、実際の値に置き換えます。EFA OS バイパス機能を有効にするには、セキュリティグループで、それ自体との間のすべてのインバウンドとアウトバウンドのトラフィックを許可する必要があります。詳細については、Amazon EC2 ユーザーガイド の ステップ 1: EFA 対応のセキュリティグループを準備する を参照してください。

> **Important**
>
> EKS マネージドノードグループを使用する場合、起動テンプレートで SubnetId は指定しないでください。EKS では、すべてのサブネットを CreateNodegroup API で指定する必要があり、サブネットの設定を含む起動テンプレートは拒否されます。

```
{
  "LaunchTemplateName": "efa-launch-template",
  "LaunchTemplateData": {
    "InstanceType": "p6-b200.48xlarge",
    "NetworkInterfaces": [
      {
        "NetworkCardIndex": 0,
        "DeviceIndex": 0,
        "InterfaceType": "interface",
        "Groups": ["security-group-id"]
      },
      {
        "NetworkCardIndex": 0,
        "DeviceIndex": 1,
        "InterfaceType": "efa-only",
        "Groups": ["security-group-id"]
      },
      {
        "NetworkCardIndex": 1,
        "DeviceIndex": 0,
        "InterfaceType": "efa-only",
        "Groups": ["security-group-id"]
      },
      {
        "NetworkCardIndex": 2,
        "DeviceIndex": 0,
        "InterfaceType": "efa-only",
        "Groups": ["security-group-id"]
      },
      {
        "NetworkCardIndex": 3,
        "DeviceIndex": 0,
        "InterfaceType": "efa-only",
        "Groups": ["security-group-id"]
      },
```

```
      {
        "NetworkCardIndex": 4,
        "DeviceIndex": 0,
        "InterfaceType": "efa-only",
        "Groups": ["security-group-id"]
      },
      {
        "NetworkCardIndex": 5,
        "DeviceIndex": 0,
        "InterfaceType": "efa-only",
        "Groups": ["security-group-id"]
      },
      {
        "NetworkCardIndex": 6,
        "DeviceIndex": 0,
        "InterfaceType": "efa-only",
        "Groups": ["security-group-id"]
      },
      {
        "NetworkCardIndex": 7,
        "DeviceIndex": 0,
        "InterfaceType": "efa-only",
        "Groups": ["security-group-id"]
      }
    ]
  }
}
```

EKS 最適化 AMI を EFA と共に使用する

EKS 最適化 AL2023 AMI とすべての Bottlerocket AMI には、EFA を使用するために必要なホストレベルのコンポーネントが含まれています。これらのコンポーネントは、具体的には aws-efa-installer によってインストールされます。EFA DRA ドライバーまたは EFA デバイスプラグインは EKS AL2023 および Bottlerocket AMI には含まれていないため、ワークロードをデプロイする前にクラスターに別途インストールする必要があります。

IP アドレス割り当てを節約する

`p5.48xlarge` や `p6-b200.48xlarge` などの EFA 対応インスタンスは、多くのネットワークインターフェイスをサポートしています。デフォルトでは、Amazon VPC CNI は、IP 対応のすべてのアタッチされた ENI に IP アドレスを割り当てます。これにより、それらのアドレスがポッドで頻繁には使用されない場合でも、サブネットの IP アドレスが大量に消費される可能性があります。多数の

ネットワークインターフェイスを使用するインスタンスでは、サブネットの使用可能な IP スペースがすぐに枯渇してしまうことがあります。

EFA 対応ノードでの IP アドレスの消費を抑えるには、プライマリ以外のすべてのインターフェイスに `efa-only` を使用するようにネットワークインターフェイスを設定します。EFA 専用インターフェイスは RDMA トラフィック専用であり、IP アドレスの割り当てがないため、サブネットのアドレスを消費しません。設定例については、「the section called “Karpenter”」および「the section called “EKS マネージドノードグループとセルフマネージドノード”」を参照してください。各インスタンスタイプの推奨されるインターフェイスレイアウトについては、「Amazon EC2 ユーザーガイド」の「EFA 対応インスタンスタイプのネットワーク帯域幅を最大化する」を参照してください。

`efa-only` インターフェイスの使用に加えて、事前に割り当てられるウォーム IP アドレスとウォーム ENI の数を制限するように Amazon VPC CNI を設定することができます。デフォルトでは、ポッドの起動を高速化するために VPC CNI で ENI と IP アドレスのウォームプールが事前に割り当てられますが、大規模なインスタンスでは、これによって数百もの使用されない IP アドレスが予約されることがあります。`aws-node` DaemonSet で `WARM_IP_TARGET` および `WARM_ENI_TARGET` 環境変数を設定して、CNI で保持する予備の IP アドレスと ENI の数を制御します。これらの設定の詳細については、「Amazon VPC CNI のベストプラクティス」を参照してください。

> **Note**
>
> `WARM_ENI_TARGET` および `WARM_IP_TARGET` はクラスター全体の設定であり、VPC CNI によって管理されるすべてのノードに適用されます。現在のところ、ノードグループまたはインスタンスタイプごとに異なる値を設定する方法はありません。これらの設定をより詳細に制御する必要がある場合は、GitHub の containers-roadmap の問題 #1834 に対するフィードバックを提供してください。

EFA DRA ドライバー (DRANET) をインストールする

EFA DRA ドライバーは、Kubernetes DRA 向けのクラウド対応ネットワークデバイス管理を提供する GitHub のアップストリーム DRANET プロジェクトに組み込まれています。このドキュメントでは EFA DRA ドライバーと DRANET を同じ意味で使用しており、どちらも同じツールのことを指しています。

EFA DRA ドライバーは、ドライバー名が `dra.net` で `DeviceClass` 名が `efa.networking.k8s.aws` である `ResourceSlice` オブジェクトとして EFA デバイスをアドバタイズします。EFA DRA ドライバーは、各ノードで DaemonSet として実行され、EFA デバイスを自動的に検出します。

前提条件

- Amazon EKS クラスターで Kubernetes バージョン 1.34 以降を実行していて、Karpenter、EKS マネージドノードグループ、またはセルフマネージドノードグループによって静的キャパシティがプロビジョニングされていること。
- ノードのインスタンスタイプが EFA 対応の Amazon EC2 であること。サポートされるインスタンスタイプのリストについては、「Amazon EC2 ユーザーガイド」の「サポートされるインスタンスタイプ」を参照してください。
- ノードに EFA 用のホストレベルのコンポーネントがインストールされていること。詳細については、「EFA ソフトウェアをインストールする」を参照してください。EKS 最適化 AL2023 NVIDIA および Neuron AMI と Bottlerocket AMI には、EFA ホストレベルのコンポーネントが含まれています。
- コマンドライン環境に Helm がインストールされていること。詳細については、「Setup Helm instructions」を参照してください。
- クラスターと通信するように kubectl が設定されていること。詳細については、「the section called “kubectl をインストールまたは更新する”」を参照してください。

手順

> **Important**
>
> EFA デバイスプラグインが実行されているノードには EFA DRA ドライバーをインストールしないでください。2 つのメカニズムを同じノードに共存させることはできません。これを行うと、基盤となるデバイスが同じノード上の複数のポッドにサイレントオーバーサブスクライブされる可能性があります。

1. EKS Helm チャートリポジトリを追加します。

```
helm repo add eks https://aws.github.io/eks-charts
```

2. ローカル Helm リポジトリを更新します。

```
helm repo update
```

3. Helm を使用してクラスターに EFA DRA ドライバーをインストールします。EFA DRA ドライバーは、インスタンスメタデータサービス (IMDS) を介して EC2 インスタンスで実行されている

ことを自動的に検出し、EFA デバイス検出を有効にします。EFA DRA ドライバーは、デフォルトでは kube-system 名前空間に DaemonSet としてデプロイされます。設定可能なパラメータについては、GitHub の EKS Helm チャートリポジトリにある Helm の values.yaml を参照してください。

```
helm install aws-dranet eks/aws-dranet --namespace kube-system
```

4. DRANET DaemonSet が実行されていることを確認します。

```
kubectl get daemonset -n kube-system aws-dranet
```

```
NAME          DESIRED   CURRENT   READY   UP-TO-DATE   AVAILABLE   NODE SELECTOR
 AGE
aws-dranet    2         2         2       2            2           <none>
 60s
```

5. DeviceClass が作成されたことを確認します。

```
kubectl get deviceclass
```

```
NAME                     AGE
efa.networking.k8s.aws   60s
```

6. ノードに対して ResourceSlice オブジェクトがアドバタイズされていることを確認します。

```
kubectl get resourceslices --field-selector spec.driver=dra.net
```

上記の手順でエラーが発生する場合は、次のコマンドを使用して DRANET のログを確認できます。

```
kubectl logs -n kube-system -l app=aws-dranet
```

7. DRA ドライバーを使用して EFA デバイスをリクエストするには、EFA DeviceClass を参照する ResourceClaim または ResourceClaimTemplate を作成し、それをポッド仕様で参照します。次の例では、EFA デバイスを 1 つだけリクエストしています。

```
apiVersion: resource.k8s.io/v1
kind: ResourceClaimTemplate
metadata:
```

```
  name: single-efa-claim
spec:
  spec:
    devices:
      requests:
      - name: efa
        exactly:
          deviceClassName: efa.networking.k8s.aws
          count: 1
---
apiVersion: v1
kind: Pod
metadata:
  name: efa-workload
spec:
  containers:
  - name: app
    ...
    resources:
      claims:
      - name: efa-device
  resourceClaims:
  - name: efa-device
    resourceClaimTemplateName: single-efa-claim
```

EFA と GPU/Neuron デバイスのトポロジー認識割り当て

EFA DRA ドライバーは、EFA インターフェイスを同じ PCIe ルート上の GPU または Neuron デバイスとペアリングするトポロジー認識割り当てをサポートしています。`matchAttribute` 制約を使用して、EFA と GPU または Neuron デバイスの割り当てをアライメントします。この機能を使用するには、NVIDIA または Neuron DRA ドライバーも使用する必要があります。詳細については、「the section called “NVIDIA GPU”」および「the section called “Neuron アクセラレーター”」を参照してください。

次の例では、1 つの NVIDIA GPU とアライメントされた 1 つの EFA インターフェイスをリクエストしています。

```
apiVersion: resource.k8s.io/v1
kind: ResourceClaimTemplate
metadata:
  name: aligned-efa-nvidia
```

```
spec:
  spec:
    devices:
      requests:
      - name: 1-efa
        exactly:
          deviceClassName: efa.networking.k8s.aws
          count: 1
      - name: 1-gpu
        exactly:
          deviceClassName: gpu.nvidia.com
          count: 1
      constraints:
      - requests: ["1-gpu", "1-efa"]
        matchAttribute: "resource.kubernetes.io/pcieRoot"
```

次の例では、4 つの Neuron デバイスとアライメントされた 4 つの EFA インターフェイスをリクエストしています。

```
apiVersion: resource.k8s.io/v1
kind: ResourceClaimTemplate
metadata:
  name: aligned-efa-neuron
spec:
  spec:
    devices:
      requests:
      - name: 4-neurons
        exactly:
          deviceClassName: neuron.aws.com
          count: 4
      - name: 4-efas
        exactly:
          deviceClassName: efa.networking.k8s.aws
          count: 4
      constraints:
      - requests: ["4-neurons", "4-efas"]
        matchAttribute: "resource.aws.com/devicegroup4_id"
```

devicegroup 属性の名前に含まれる数字は、接続されたトポロジーグループ内の Neuron デバイスの数に対応しています。例えば、resource.aws.com/devicegroup1_id は単一の Neuron デバイスを識別し、resource.aws.com/devicegroup4_id は接続された 4 台のデバ

イスのグループを識別し、resource.aws.com/devicegroup8_id と resource.aws.com/devicegroup16_id は接続された 8 台と 16 台のデバイスのグループをそれぞれ識別します。割り当てられた Neuron デバイスと EFA インターフェイスが同じ接続トポロジーグループに属するように、リクエスト内のデバイスの count に一致する matchAttribute を選択します。これらの属性の詳細については、Neuron DRA ドライバーのドキュメントを参照してください。

allocationMode を使用して、EFA デバイスをアライメントされた GPU または Neuron アクセラレーターに割り当てる方法を簡素化できます。allocationMode フィールドでは 2 つの値がサポートされます。ExactCount (デフォルト) は count で指定された特定の数のデバイスをリクエストし、All はプール内のすべての一致するデバイスをリクエストします。例えば、p5.48xlarge インスタンスには、1 つの GPU で同じ PCIe ルートを共有する 4 つの EFA デバイスがあります。これらの EFA デバイスのグループをアライメントされた GPU で割り当てる場合、正確な EFA-GPU デバイスマッピングやアライメントされた EFA デバイスの数がわからなくても、ResourceClaimTemplate で EFA デバイスに allocationMode: All を設定することができます。

```
apiVersion: resource.k8s.io/v1
kind: ResourceClaimTemplate
metadata:
  name: aligned-all-efa-one-nvidia
spec:
  spec:
    devices:
      requests:
      - name: all-efas
        exactly:
          deviceClassName: efa.networking.k8s.aws
          allocationMode: All
      - name: one-gpu
        exactly:
          deviceClassName: gpu.nvidia.com
          allocationMode: ExactCount
          count: 1
      constraints:
      - requests: ["all-efas", "one-gpu"]
        matchAttribute: "resource.kubernetes.io/pcieRoot"
```

複数のポッド間で EFA デバイスを共有する

EFA DRA ドライバーは、ResourceClaim を使用して複数のポッド間での EFA デバイスの共有をサポートします。ポッドごとに個別のクレームを生成する ResourceClaimTemplate とは異な

り、ResourceClaim は、個別に作成して複数のポッドから参照する名前付きオブジェクトです。同じ ResourceClaim を参照するすべてのポッドで、割り当てられた同じ EFA デバイスへのアクセスを共有し、それらのデバイスが使用可能な同じノードにスケジュールされます。

ポッド間で EFA デバイスを共有するには、EFA デバイスをリクエストする ResourceClaim を作成し、そのクレームを各ポッドの resourceClaims フィールドで resourceClaimName を使用して名前で参照します。ResourceClaim は、それを参照するポッドの作成前にクラスターに存在している必要があります。参照している ResourceClaim が存在しない場合、ポッドはクレームが作成されるまで保留状態になります。

次の例では、4 つの EFA デバイスをリクエストする ResourceClaim と、それらのデバイスへのアクセスを共有する 2 つのポッドを作成します。

1. ResourceClaim を作成します。

```
apiVersion: resource.k8s.io/v1
kind: ResourceClaim
metadata:
  name: shared-efa
spec:
  devices:
    requests:
    - name: efa
      exactly:
        deviceClassName: efa.networking.k8s.aws
        count: 4
```

2. EFA デバイスにアクセスする必要がある各ポッドで ResourceClaim を名前で参照します。各ポッドで、resourceClaimTemplateName ではなく resourceClaimName を使用して既存のクレームを参照します。

```
apiVersion: v1
kind: Pod
metadata:
  name: training-worker
spec:
  containers:
  - name: worker
    image: my-training-image
    resources:
      claims:
      - name: efa-devices
```

```
  resourceClaims:
  - name: efa-devices
    resourceClaimName: shared-efa
---
apiVersion: v1
kind: Pod
metadata:
  name: training-monitor
spec:
  containers:
  - name: monitor
    image: my-monitor-image
    resources:
      claims:
      - name: efa-devices
  resourceClaims:
  - name: efa-devices
    resourceClaimName: shared-efa
```

両方のポッドが同じ `shared-efa` `ResourceClaim` を参照し、それらの EFA デバイスが割り当てられたノードにスケジュールされます。`ResourceClaim` のライフサイクルはポッドとは無関係です。このクレームは、それを参照するすべてのポッドが削除されても、削除するまで保持されます。

EFA Kubernetes デバイスプラグインをインストールする

EFA Kubernetes デバイスプラグインは、EFA デバイスを `vpc.amazonaws.com/efa` 拡張リソースとしてアドバタイズします。コンテナリソースのリクエストや制限の際に EFA デバイスをリクエストします。トレーニングワークロードで EFA を設定する詳細なチュートリアルについては、「the section called “EFA を使用したクラスターのトレーニングの設定”」を参照してください。

> **Important**
>
> EKS 最適化 AL2023 高速 AMI を使用している場合、EFA インターフェイスを備えた NVIDIA GPU または Neuron デバイスのトポロジーに沿った割り当てが自動的に行われます。この自動アライメントは、Bottlerocket EKS 最適化 AMI またはカスタム AMI を使用している場合は発生しません。Bottlerocket またはカスタム AMI でトポロジーに沿ったアクセラレーターと EFA デバイスの割り当てが必要な場合は、EFA DRA ドライバーと対応する Neuron DRA ドライバーを使用してください。Bottlerocket で NVIDIA DRA ドライバーを使用するには、まず Bottlerocket NVIDIA バリアントにバンドルされている NVIDIA デバイスプラグインを無効にする必要があります。これには Bottlerocket バージョン 1.63.0 以降が必要です。詳細につ

いては、「the section called “EFA と GPU/Neuron デバイスのトポロジー認識割り当て”」および「the section called “NVIDIA DRA ドライバーをインストールする”」を参照してください。

Important

NVIDIA `k8s-device-plugin` v0.19.0 以降では、`--mofed-enabled` フラグがデフォルトで `true` に設定されています。そのため、NVIDIA デバイスプラグインでは、GPU をリクエストするコンテナにすべての `/dev/infiniband/uverbs*` デバイスがマウントされます。これは、`/dev/infiniband` で EFA デバイスの割り当てを管理するコンポーネントである EFA デバイスプラグインと競合します。NVIDIA デバイスプラグインで EKS マネージドノードグループまたはセルフマネージドノードを使用している場合は、MOFED を明示的に無効にする必要があります。手順については、「the section called “NVIDIA Kubernetes デバイスプラグインをインストールする”」を参照してください。
EKS オートモードではデフォルトで MOFED が有効にならないため、この問題は影響しません。

前提条件

- Amazon EKS クラスター。
- ノードのインスタンスタイプが EFA 対応の Amazon EC2 であること。サポートされるインスタンスタイプのリストについては、「Amazon EC2 ユーザーガイド」の「サポートされるインスタンスタイプ」を参照してください。
- ノードに EFA 用のホストレベルのコンポーネントがインストールされていること。詳細については、「EFA ソフトウェアをインストールする」を参照してください。EKS 最適化 AL2023 AMI と Bottlerocket AMI には、EFA ホストレベルのコンポーネントが含まれています。
- コマンドライン環境に Helm がインストールされていること。詳細については、「Setup Helm instructions」を参照してください。
- クラスターと通信するように `kubectl` が設定されていること。詳細については、「the section called “`kubectl` をインストールまたは更新する”」を参照してください。

手順

1. EKS Helm チャートリポジトリを追加します。

```
helm repo add eks https://aws.github.io/eks-charts
```

2. ローカル Helm リポジトリを更新します。

```
helm repo update
```

3. EFA デバイスプラグインをインストールします。

```
helm install efa eks/aws-efa-k8s-device-plugin -n kube-system
```

4. DaemonSet という EFA デバイスプラグインが実行されていることを確認します。

```
kubectl get daemonset -n kube-system efa-aws-efa-k8s-device-plugin
```

```
NAME                                  DESIRED   CURRENT   READY   UP-TO-DATE
 AVAILABLE   NODE SELECTOR   AGE
efa-aws-efa-k8s-device-plugin         2         2         2       2
      2           <none>          60s
```

5. ノードを調べて割り当て可能な EFA リソースがあることを確認します。

```
kubectl get nodes "-o=custom-columns=NAME:.metadata.name,EFA:.status.allocatable.vpc
\.amazonaws\.com/efa"
```

```
NAME                                          EFA
ip-192-168-11-225.us-west-2.compute.internal   4
ip-192-168-24-96.us-west-2.compute.internal    4
```

6. デバイスプラグインを使用して EFA デバイスをリクエストするには、コンテナリソースのリクエストと制限で vpc.amazonaws.com/efa リソースを指定します。

```
apiVersion: v1
kind: Pod
metadata:
  name: efa-workload
spec:
  containers:
  - name: app
    ...
    resources:
```

```
      limits:
        vpc.amazonaws.com/efa: 4
        hugepages-2Mi: ...
      requests:
        vpc.amazonaws.com/efa: 4
        hugepages-2Mi: ...
```

Amazon EKS で Neuron デバイスを管理する

AWS Trainium と AWS Inferentia は、AWS によって設計された専用の機械学習チップです。Amazon EKS は、EKS クラスターで Neuron デバイスを管理できるように 2 つのメカニズムをサポートしています。Neuron DRA ドライバーと Neuron Kubernetes デバイスプラグインです。

Karpenter、EKS マネージドノードグループ、またはセルフマネージドノードで [静的キャパシティのプロビジョニング] を使用する場合は、Kubernetes バージョン 1.34 以降の新しいデプロイに DRA ドライバーを使用することをお勧めします。DRA は、現在 EKS Auto Mode ではサポートされていません。Neuron DRA ドライバーは、トポロジ認識割り当て、コネクテッドデバイスサブセットスケジューリング、Logical NeuronCore (LNC) 設定、UltraServer マルチノード割り当ての機能を備えています。カスタムスケジューラ拡張機能は必要ありません。

Neuron DRA ドライバーと Neuron デバイスプラグインの比較

機能	Neuron DRA ドライバー	Neuron デバイスプラグイン
最小限必要な Kubernetes バージョン	1.34	EKS でサポートされているすべての Kubernetes バージョン
EKS コンピューティング	Karpenter (静的キャパシティのみ)、マネージドノードグループ、セルフマネージドノード	EKS オートモード、Karpenter、マネージドノードグループ、セルフマネージドノード
EKS 最適化 AMI のサポート	AL2023 (Neuron)、Bottlerocket	AL2023 (Neuron)、Bottlerocket
デバイスアドバタイズ	デバイス ID、インスタンスタイプ、トポロジ、ドライバーバージョン、EFA ローカリティなど、	`aws.amazon.com/neuron` および `aws.amazon.com/neuroncore` 拡張リソースの整数カウント

機能	Neuron DRA ドライバー	Neuron デバイスプラグイン
	`ResourceSlice` オブジェクトによる豊富な属性	
コネクテッドデバイスサブセット	トポロジ制約を使用して、1、4、8、または 16 台のコネクテッド Neuron デバイスサブセットを割り当てる	連続してデバイスを割り当てるには、Neuron スケジューラ拡張機能が必要
LNC 設定	`ResourceClaimTemplate` パラメータによるワークロードごとの Logical NeuronCore 設定 (LNC=1 または LNC=2)	EC2 起動テンプレートで事前設定が必要
属性ベースの選択	CEL 式を使用してインスタンスタイプやドライバーバージョンといった属性でデバイスをフィルタリングする	サポートされていません
トポロジー認識 EFA 割り当て	DRA ネイティブトポロジー認識	自動トポロジー認識 (EKS 最適化 AL2023 AMI のみ)

Neuron DRA ドライバーをインストールする

Neuron DRA ドライバーは、`DeviceClass` 名が `neuron.aws.com` である `ResourceSlice` オブジェクトとして Neuron デバイスをアドバタイズします。DaemonSet として実行されたこのドライバーは、Neuron デバイスとそのトポロジ属性を自動的に検出します。

Neuron DRA ドライバーの詳細については、Neuron DRA のドキュメントを参照してください。

前提条件

- Amazon EKS クラスターで Kubernetes バージョン 1.34 以降を実行していて、Karpenter、EKS マネージドノードグループ、またはセルフマネージドノードグループによって静的キャパシティがプロビジョニングされていること。
- ノードのインスタンスタイプが AWS Trainium または Inferentia2 であること。
- コマンドライン環境に Helm がインストールされていること。詳細については、「Setup Helm instructions」を参照してください。

- クラスターと通信するように kubectl が設定されていること。詳細については、「the section called “kubectl をインストールまたは更新する”」を参照してください。

手順

> **Important**
>
> Neuron デバイスプラグインが実行されているノードに、Neuron DRA ドライバーをインストールしないでください。2 つのメカニズムを同じノードに共存させることはできません。これを行うと、基盤となるデバイスが同じノード上の複数のポッドにサイレントオーバーサブスクライブされる可能性があります。

1. Helm を使用して Neuron DRA ドライバーをインストールします。

```
helm upgrade --install neuron-helm-chart oci://public.ecr.aws/neuron/neuron-helm-
chart \
    --namespace neuron-dra-driver \
    --create-namespace \
    --set "devicePlugin.enabled=false" \
    --set "npd.enabled=false" \
    --set "draDriver.enabled=true"
```

   ドライバーは、デフォルトで neuron-dra-driver 名前空間に DaemonSet としてデプロイされ、その際に DeviceClass は neuron.aws.com になります。

2. DRA ドライバーが DaemonSet として実行されていることを確認します。

```
kubectl get ds -n neuron-dra-driver neuron-dra-driver-kubelet-plugin
```

```
NAME                              DESIRED   CURRENT   READY   UP-TO-DATE   AVAILABLE
  NODE SELECTOR   AGE
neuron-dra-driver-kubelet-plugin  1         1         1       1            1
  <none>          60s
```

3. DeviceClass が作成されたことを確認します。

```
kubectl get deviceclass neuron.aws.com
```

```
NAME            AGE
neuron.aws.com  60s
```

4. ノードに対して ResourceSlice オブジェクトがアドバタイズされていることを確認します。

```
kubectl get resourceslice
```

使用可能な ResourceSlice オブジェクト属性については、Neuron DRA のドキュメントを参照してください。

ポッドで Neuron デバイスをリクエストする

DRA ドライバーを使用して Neuron デバイスをリクエストするには、neuron.aws.com DeviceClass を参照する ResourceClaimTemplate を作成し、それをポッド仕様で参照します。

次の例では、trn2.48xlarge インスタンス上のすべての Neuron デバイスをリクエストしています。

```
apiVersion: resource.k8s.io/v1
kind: ResourceClaimTemplate
metadata:
  name: all-neurons
spec:
  spec:
    devices:
      requests:
      - name: neurons
        exactly:
          deviceClassName: neuron.aws.com
          selectors:
          - cel:
              expression: "device.attributes['neuron.aws.com'].instanceType ==
 'trn2.48xlarge'"
          allocationMode: All
---
apiVersion: v1
kind: Pod
metadata:
  name: neuron-workload
spec:
```

```
  containers:
  - name: app
    ...
    resources:
      claims:
      - name: neurons
  resourceClaims:
  - name: neurons
    resourceClaimTemplateName: all-neurons
```

コネクテッドデバイスサブセットを割り当てる

Neuron DRA ドライバーは、コネクテッド Neuron デバイスサブセットを割り当てることができます。Neuron スケジューラ拡張機能は必要ありません。サポートされているサブセットサイズは、1、4、8、または 16 台のデバイスです。matchAttribute 制約とトポロジグループ ID を使用して、デバイスが接続されていることを確認します。

次の例では、4 台のコネクテッド Neuron デバイスをリクエストしています。

```
apiVersion: resource.k8s.io/v1
kind: ResourceClaimTemplate
metadata:
  name: 1x4-connected-neurons
spec:
  spec:
    devices:
      requests:
      - name: neurons
        exactly:
          deviceClassName: neuron.aws.com
          allocationMode: ExactCount
          count: 4
          selectors:
          - cel:
              expression: "device.attributes['neuron.aws.com'].instanceType ==
 'trn2.48xlarge'"
      constraints:
      - requests: ["neurons"]
        matchAttribute: "resource.aws.com/devicegroup4_id"
```

コネクテッドサブセットにサポートされている matchAttribute 値は、resource.aws.com/devicegroup1_id、resource.aws.com/devicegroup4_id、resource.aws.com/

devicegroup8_id、および resource.aws.com/devicegroup16_id です。devicegroup 属性の名前に含まれる数字は、接続されたトポロジーグループ内の Neuron デバイスの数に対応しています。例えば、resource.aws.com/devicegroup1_id は単一の Neuron デバイスを識別し、resource.aws.com/devicegroup4_id は接続された 4 台のデバイスのグループを識別し、resource.aws.com/devicegroup8_id と resource.aws.com/devicegroup16_id は接続された 8 台と 16 台のデバイスのグループをそれぞれ識別します。割り当てられたデバイスが同じ接続トポロジーグループに属するように、リクエスト内のデバイスの count に一致する matchAttribute を選択します。これらの属性の詳細については、Neuron DRA ドライバーのドキュメントを参照してください。

Logical NeuronCores (LNC) を設定する

Neuron DRA ドライバーにより、ResourceClaimTemplate パラメータを使用してワークロードごとに Logical NeuronCore を設定できます。これにより、EC2 起動テンプレートで LNC を事前設定する必要がなくなります。

次の例では、LNC が 1 に設定されているすべての Neuron デバイスをリクエストしています。

```
apiVersion: resource.k8s.io/v1
kind: ResourceClaimTemplate
metadata:
  name: all-neurons-lnc-1
spec:
  spec:
    devices:
      requests:
      - name: neurons
        exactly:
          deviceClassName: neuron.aws.com
          selectors:
          - cel:
              expression: "device.attributes['neuron.aws.com'].instanceType ==
 'trn2.48xlarge'"
          allocationMode: All
      config:
      - requests: ["neurons"]
        opaque:
          driver: neuron.aws.com
          parameters:
            apiVersion: neuron.aws.com/v1
            kind: NeuronConfig
```

```
            logicalNeuronCore: 1
```

アライメントされた EFA インターフェイスを備えた Neuron デバイスを割り当てる

「the section called “EFA と GPU/Neuron デバイスのトポロジー認識割り当て”」を参照してください。

Neuron Kubernetes デバイスプラグインをインストールする

Neuron Kubernetes デバイスプラグインは、Neuron デバイスを `aws.amazon.com/neuron` としてアドバタイズし、NeuronCores を `aws.amazon.com/neuroncore` 拡張リソースとしてアドバタイズします。コンテナリソースのリクエストや制限の際に Neuron デバイスをリクエストします。

前提条件

- Amazon EKS クラスター。
- ノードに、ホストレベルのコンポーネントが AWS Trainium または AWSInferentia インスタンス用にインストールされていること。こうしたコンポーネントは、EKS AL2023 高速 AMI または EKS Bottlerocket AMI を使用している場合に含まれています。
- コマンドライン環境に Helm がインストールされていること。詳細については、「Setup Helm instructions」を参照してください。
- クラスターと通信するように `kubectl` が設定されていること。詳細については、「the section called “`kubectl` をインストールまたは更新する”」を参照してください。

手順

1. Helm を使用して Neuron Kubernetes デバイスプラグインをインストールします。

```
helm upgrade --install neuron-helm-chart oci://public.ecr.aws/neuron/neuron-helm-
chart \
    --set "npd.enabled=false"
```

2. DaemonSet という Neuron デバイスプラグインが実行されていることを確認します。

```
kubectl get ds -n kube-system neuron-device-plugin
```

```
NAME                   DESIRED   CURRENT   READY   UP-TO-DATE   AVAILABLE   NODE
 SELECTOR   AGE
```

```
neuron-device-plugin   1         1         1       1            1           <none>
       60s
```

3. ノードを調べて割り当て可能な Neuron デバイスがあることを確認します。

```
kubectl get nodes "-o=custom-
columns=NAME:.metadata.name,NeuronDevice:.status.allocatable.aws\.amazon\.com/
neuron,NeuronCore:.status.allocatable.aws\.amazon\.com/neuroncore"
```

```
NAME                                          NeuronDevice   NeuronCore
ip-192-168-47-173.us-west-2.compute.internal   1              2
```

テストポッドを使用して Neuron デバイスを確認する

Neuron デバイスがアクセス可能であることを確認するには、テストポッドで neuron-ls ツールを実行します。

1. 次の内容で、neuron-ls.yaml という名前のファイルを作成します。このマニフェストは、neuron-ls ツールがインストールされている Neuron Monitor コンテナを起動します。

```
apiVersion: v1
kind: Pod
metadata:
  name: neuron-ls
spec:
  restartPolicy: Never
  containers:
  - name: neuron-container
    image: public.ecr.aws/g4h4h0b5/neuron-monitor:1.0.0
    command: ["/bin/sh"]
    args: ["-c", "neuron-ls"]
    resources:
      limits:
        aws.amazon.com/neuron: 1
      requests:
        aws.amazon.com/neuron: 1
  tolerations:
  - key: "aws.amazon.com/neuron"
    operator: "Exists"
    effect: "NoSchedule"
```

2. マニフェストを適用します。

```
kubectl apply -f neuron-ls.yaml
```

3. ポッドの実行が終了したら、そのログを表示します。

```
kubectl logs neuron-ls
```

出力例は次のとおりです。

```
instance-type: inf2.xlarge
instance-id: ...
+--------+--------+--------+---------+
| NEURON | NEURON | NEURON |   PCI   |
| DEVICE | CORES  | MEMORY |   BDF   |
+--------+--------+--------+---------+
| 0      | 2      | 32 GB  | 00:1f.0 |
+--------+--------+--------+---------+
```

> **Note**
>
> Neuron デバイスプラグインを使用している場合に、複数の Neuron デバイスを備えたインスタンス (`trn2.48xlarge` など) で連続してデバイスを割り当てるには、Neuron Kubernetes スケジューラ拡張機能が必要です。Neuron DRA ドライバーは、トポロジ制約を利用してこれを自動的に処理します。

Amazon EKS で Neuron デバイスを使用する方法の詳細については、EKS 上で実行するための Neuron のドキュメントを参照してください。

Amazon EKS で NVIDIA GPU を管理する

NVIDIA GPU は、機械学習トレーニング、推論、ハイパフォーマンスコンピューティングワークロードに広く使用されています。Amazon EKS は、EKS クラスターで NVIDIA GPU デバイスを管理できるように 2 つのメカニズムをサポートしています。GPU 用 NVIDIA DRA ドライバーと NVIDIA Kubernetes デバイスプラグインです。

Karpenter、EKS マネージドノードグループ、またはセルフマネージドノードで [静的キャパシティのプロビジョニング] を使用する場合は、Kubernetes バージョン 1.34 以降の新しいデプロイに DRA

ドライバーを使用することをお勧めします。DRA は、現在 EKS Auto Mode ではサポートされていません。NVIDIA DRA ドライバーを使用すると、GPU を柔軟に割り当てたり、コンテナ間で GPU を共有したりできます。

EKS での DRA 機能の可用性については、EKS のKubernetes バージョンのリリースノートを参照してください。EKS で NVIDIA DRA ドライバーとデバイスプラグインを使用する方法の詳細については、the section called “DRA & デバイスプラグイン” を参照してください。

GPU 共有 (MIG & タイムスライシング)

マルチインスタンス GPU (MIG)

マルチインスタンス GPU (MIG) は、単一の物理 GPU を複数の分離されたインスタンスにパーティション化する NVIDIA GPU のハードウェア機能です。インスタンスの最大数は GPU によって異なります。A100、H100、H200、B200 などのデータセンター GPU は最大 7 個のインスタンスをサポートし、Blackwell ベースの g7 および g7e GPU はより少ないインスタンスをサポートします。各インスタンスには専用メモリ、コンピューティングユニット、メモリ帯域幅があるため、あるインスタンスで実行されるワークロードが別のインスタンスのワークロードに影響を与えることはありません。分離なしでソフトウェアのタイムマルチプレクシングを通じて GPU を共有するタイムスライシングとは異なり、MIG は Pod 間でハードウェアレベルのメモリと障害分離を提供します。

MIG は、マルチテナント推論、予測可能なサービス品質を必要とするワークロード、ハードウェア分離テナンシーのコンプライアンス要件がある環境に最適です。NVIDIA Ampere (A100)、Hopper (H100 と H200)、および Blackwell GPU で使用できます。AWS では、これらは P ファミリーインスタンスタイプと、Blackwell ベースの g7 および g7e インスタンスタイプです。

NVIDIA GPU および EKS で MIG を使用する方法と手順の詳細については、the section called “マルチインスタンス GPU (MIG)” を参照してください。

タイムスライシング

タイムスライシングでは、NVIDIA デバイスプラグインまたは DRA ドライバーを設定して、GPU ごとに複数のスケジュール可能なスロットをアドバタイズすることで、複数の Pod が単一の物理 NVIDIA GPU を共有できます。Kubernetes スケジューラは同じ GPU に複数の Pod を配置し、GPU の CUDA スケジューラはワークロードをタイムマルチプレックスします。

タイムスライシングは、最もシンプルな GPU 共有戦略です。ソフトウェアのみを使用し、特別なハードウェアを必要とせず、AWS 上のすべての NVIDIA GPU インスタンスタイプで動作します。タイムスライシングでは、GPU を共有する Pod 間でメモリやコンピューティングの分離を行うことはありません。これは、リクエスト間でアイドル状態になる推論サービスや、複数のユーザーが GPU

を共有する開発環境など、GPU 使用率が低いワークロードに最適です。ハードウェアレベルのメモリとコンピューティングの分離を必要とするワークロードでは、代わりに MIG を使用します。

NVIDIA GPU と EKS でタイムスライシングを使用する方法と手順については、the section called “タイムスライシング” を参照してください。

トピック

- the section called “DRA & デバイスプラグイン”
- the section called “マルチインスタンス GPU (MIG)”
- the section called “タイムスライシング”

Amazon EKS で NVIDIA DRA ドライバーまたはデバイスプラグインを使用する

Amazon EKS は、EKS クラスターで NVIDIA GPU デバイスを管理できるように 2 つのメカニズムをサポートしています。GPU 用 NVIDIA DRA ドライバーと NVIDIA Kubernetes デバイスプラグインです。

Karpenter、EKS マネージドノードグループ、またはセルフマネージドノードで [静的キャパシティのプロビジョニング] を使用する場合は、Kubernetes バージョン 1.34 以降の新しいデプロイに NVIDIA DRA ドライバーを使用することをお勧めします。NVIDIA DRA ドライバーは、現在 EKS Auto Mode ではサポートされていません。動的キャパシティのプロビジョニングを使用する場合は、EKS Auto Mode または Karpenter で NVIDIA デバイスプラグインを使用します。NVIDIA デバイスプラグインは、EKS マネージドノードグループとセルフマネージドノードでも引き続きサポートされます。

マルチインスタンス GPU (MIG) やタイムスライシングなどの GPU 共有機能を使用している場合は、DRA ドライバーで静的設定を使用するか、NVIDIA デバイスプラグインを使用することをお勧めします。NVIDIA DRA ドライバーでは、動的 MIG とタイムスライシングはアルファ状態です。最新の更新については、GitHub の NVIDIA DRA ドライバーリリースを参照してください。

NVIDIA DRA ドライバーと NVIDIA デバイスプラグインの比較

機能	NVIDIA DRA ドライバー	NVIDIA デバイスプラグイン
最小限必要な Kubernetes バージョン	1.34	EKS でサポートされているすべての Kubernetes バージョン

機能	NVIDIA DRA ドライバー	NVIDIA デバイスプラグイン
EKS コンピューティング	Karpenter (静的キャパシティのみ)、マネージドノードグループ、セルフマネージドノード	EKS オートモード、Karpenter、マネージドノードグループ、セルフマネージドノード
EKS 最適化 AMI	AL2023 (NVIDIA)、Bottlerocket	AL2023 (NVIDIA)、Bottlerocket
デバイスアドバタイズ	GPU モデル、メモリ、ドライバーバージョン、トポロジーなど、`ResourceSlice` オブジェクトによる豊富な属性	`nvidia.com/gpu` 拡張リソースの整数カウント
GPU 共有	(アルファ) 動的 MIG、MPS、タイムスライシング	(GA) 静的 MIG、MPS、タイムスライシング
ComputeDomains	安全なマルチノード GPU 通信を実現できるように、`ComputeDomain` リソースを介してマルチノード NVLink (MNNVL) を管理する	サポートされていません
属性ベースの選択	CEL 式を使用してモデルやメモリといった属性で GPU をフィルタリングする	サポートされていません
トポロジー認識 EFA 割り当て	DRA ネイティブトポロジー認識	自動トポロジー認識 (EKS 最適化 AL2023 AMI のみ)

NVIDIA GPU ノードラベル

NVIDIA GPU ノードには、GPU ハードウェアとその設定方法を記述する一連の `nvidia.com/*` Kubernetes ノードラベルがあります。コンポーネントごとに異なるラベルが適用されます。ラベルは、書き込むコンポーネントがノードで実行されている場合にのみ存在します。各ラベルを適用するコンポーネントを理解することで、信頼性の高いノードセレクタを記述し、ノードが報告する内容を解釈できます。

GPU Feature Discovery なしで NVIDIA デバイスプラグインのみを実行するノードには、nvidia.com/* ラベルはありません。EKS AL2023 または Bottlerocket NVIDIA AMI を使用している場合、 nvidia.com/gpu.present ラベルは自動的に適用されます。

ラベルは情報提供を目的としており、ノードのターゲティングに使用されます。GPU でワークロードを実行するには、ポッド仕様で GPU リソース (nvidia.com/gpu や nvidia.com/mig-1g.10gb など) をリクエストします。nodeSelector またはノードアフィニティのラベルを使用して、ワークロードがどの GPU ノードに配置されるかを制限します。

GPU Feature Discovery (GFD)

GPU Feature Discovery (GFD) は、スタンドアロンまたは NVIDIA GPU Operator の一部として実行される NVIDIA デバイスプラグインプロジェクト (gpu-feature-discovery デプロイメント) のコンポーネントです。GitHub ウェブサイトの Node Feature Discovery (NFD) に従って、以下を含むすべての説明的な nvidia.com/* 機能ラベルを書き込みます。

ラベル	説明
nvidia.com/gpu.product	GPU モデル。例: NVIDIA-A100-SXM4-40GB 。シングルストラテジー MIG では、GFD はアクティブな MIG プロファイルを -MIG-<profile> サフィックスとして追加します。例えば、NVIDIA-A100-SXM4-40GB-MIG-1g.5gb です。
nvidia.com/gpu.count	デバイスプラグインがアドバタイズするスケジュール可能な GPU デバイスの数。シングルストラテジー MIG の場合、これは物理的な GPU の数ではなく、MIG インスタンスの数です。
nvidia.com/gpu.memory	デバイスあたりの GPU メモリ (MiB 単位)。
nvidia.com/gpu.family , nvidia.com/gpu.compute.major , nvidia.com/gpu.compute.minor	GPU アーキテクチャファミリー (ampere、hopper、blackwell など) と CUDA コンピューティング機能。
nvidia.com/mig.capable	GPU ハードウェアが MIG をサポートしている場合、true。これは、MIG が有効になってい

ラベル	説明
	るかどうかではなく、ハードウェア機能のみを反映します。
nvidia.com/mig.strategy	有効なデバイスプラグイン MIG 戦略 (single または mixed)。「MIG 戦略」を参照してください。
nvidia.com/gpu.sharing-strategy	GPU 共有戦略 (none、time-slicing 、または mps)。
nvidia.com/mig-<profile>.*	混合戦略 MIG では、各 MIG プロファイルに対して、nvidia.com/mig-1g.10gb.count や nvidia.com/mig-1g.10gb.product などのプロファイルごとのラベルがノードに存在します。
nvidia.com/cuda.driver-version., nvidia.com/cuda.runtime-version.	インストール済みの NVIDIA ドライバーと CUDA ランタイムのバージョン。

MIG Manager (NVIDIA GPU Operator のみ)

MIG Manager は NVIDIA GPU Operator のオペランドです。次のラベルを適用します。GPU Operator をインストールしない限り存在しません。

ラベル	説明
nvidia.com/mig.config	MIG Manager が適用をリクエストされる MIG レイアウト (例: all-disabled 、または all-1g.5gb などの名前付きプロファイル)。このラベルを設定してレイアウトを選択または変更します。MIG Manager は変更を監視し、一致するように GPU をパーティション分割します。必要に応じてノードを再起動します。

ラベル	説明
nvidia.com/mig.config.state	リクエストされたレイアウトの調整状態 (success、pending、failed など)。

> **Important**
>
> nvidia.com/mig.config は、GPU のライブリードバックではなく、MIG Manager が最後に適用を求められたレイアウトを反映しています。MIG Manager は、このラベルが変更された場合にのみ調整を行います。したがって、MIG モードが有効になっているか、MIG Manager の外部 (nvidia-smi など) でパーティション分割されている場合、ラベルが all-disabled を示している間も GPU はアクティブにパーティション分割される可能性があります。ノードの Allocatable (およびホスト上の nvidia-smi) でアドバタイズされたリソースを、スケジュール可能なリソースの信頼できるソースとして扱い、nvidia.com/mig.config ラベルは MIG Manager を駆動する目的にのみ使用します。

NVIDIA DRA ドライバーをインストールする

GPU 用 NVIDIA DRA ドライバーは、GPU と ComputeDomains という 2 つのタイプのリソースを管理します。また、gpu-kubelet-plugin と compute-domain-kubelet-plugin という 2 つの DRA kubelet プラグインを実行します。それぞれ、インストール中に個別に有効または無効にできます。このガイドでは、GPU の割り当てに焦点を当てます。ComputeDomains の使用方法については、「the section called “EKS で P6e-GB200 を使用する”」を参照してください。

前提条件

- Amazon EKS クラスターで Kubernetes バージョン 1.34 以降を実行していて、Karpenter、EKS マネージドノードグループ、またはセルフマネージドノードグループによって静的キャパシティがプロビジョニングされていること。
- ノードのインスタンスタイプが NVIDIA GPU であること (P や G インスタンスなど)。
- ノードに NVIDIA GPU 用のホストレベルのコンポーネントがインストールされていること。EKS 最適化 AL2023 または Bottlerocket NVIDIA AMI を使用する場合には、ホストレベルの NVIDIA ドライバー、CUDA ユーザーモードドライバー、コンテナツールキットがプリインストールされていること。

- コマンドライン環境に Helm がインストールされていること。詳細については、「Setup Helm instructions」を参照してください。
- クラスターと通信するように kubectl が設定されていること。詳細については、「the section called “kubectl をインストールまたは更新する”」を参照してください。

手順

> **Important**
>
> GPU デバイスの管理に NVIDIA DRA ドライバーを使用する場合、同じノードに NVIDIA デバイスプラグインと一緒にデプロイしないでください。これを行うと、基盤となるデバイスが同じノード上の複数のポッドにサイレントオーバーサブスクライブされる可能性があります。

> **Bottlerocket で組み込み NVIDIA デバイスプラグインを無効にする**
>
> EKS 最適化 Bottlerocket NVIDIA バリアントには NVIDIA デバイスプラグインが含まれており、デフォルトで有効になっています。DRA ドライバーは、同じノードのデバイスプラグインと一緒に実行することはできません。DRA ドライバーを使用する前に、Bottlerocket GPU ノードの組み込みデバイスプラグインを無効にします。Bottlerocket ノードのユーザーデータで `settings.kubelet-device-plugins.nvidia.enabled` を `false` に設定します。
>
> ```
> [settings.kubelet-device-plugins.nvidia]
> enabled = false
> ```
>
> `settings.kubelet-device-plugins.nvidia.enabled` 設定は Bottlerocket バージョン 1.63.0 以降で使用できます。以前のバージョンでは、組み込み NVIDIA デバイスプラグインを無効にすることはできません。詳細については、GitHub の「bottlerocket-os/bottlerocket pull request #4856」を参照してください。

1. NVIDIA DRA ドライバーを Kubernetes SIG OCI レジストリから直接インストールします。利用可能なバージョンを検索するには、GitHub の「NVIDIA DRA ドライバーリリース」を参照してください。

```
helm install dra-driver-nvidia-gpu \
    oci://registry.k8s.io/dra-driver-nvidia/charts/dra-driver-nvidia-gpu \
    --version 0.4.1 \
    --create-namespace \
    --namespace nvidia \
    --set resources.computeDomains.enabled=false \
    --set gpuResourcesEnabledOverride=true
```

詳細な設定オプションについては、Kubernetes SIG ウェブサイトの「NVIDIA DRA ドライバーの Helm チャート値」を参照してください。特定のチャートバージョンで使用できる値を表示するには、`helm show values oci://registry.k8s.io/dra-driver-nvidia/charts/dra-driver-nvidia-gpu --version 0.4.1` を実行します。

2. (オプション) DRA ドライバーでタイムスライシングを使用するには、前のステップの `helm install` コマンドに `TimeSlicingSettings` 機能ゲートを追加します。これは、デフォルトで無効になっているアルファ機能です。詳細については、「the section called “NVIDIA DRA ドライバーで GPU タイムスライシングを使用する”」を参照してください。

```
--set featureGates.TimeSlicingSettings=true
```

3. (オプション) DRA ドライバーで動的 MIG を使用するには、前のステップの `helm install` コマンドに `DynamicMIG` 機能ゲートを追加します。これは、デフォルトで無効になっているアルファ機能です。`DynamicMIG` 機能ゲートを `PassthroughSupport`、`NVMLDeviceHealthCheck`、または `MPSSupport` 機能ゲートと組み合わせることはできません。詳細については、「the section called “NVIDIA DRA ドライバーで MIG を使用する”」を参照してください。

```
--set featureGates.DynamicMIG=true
```

4. DRA ドライバーポッドが実行中であることを確認します。

```
kubectl get pods -n nvidia
```

5. `DeviceClass` オブジェクトが作成されたことを確認します。

```
kubectl get deviceclass
```

```
NAME            AGE
gpu.nvidia.com  60s
```

6. ResourceSlice オブジェクトが GPU ノードに公開されることを確認します。

```
kubectl get resourceslice
```

DRA ドライバーを使用して NVIDIA GPU をリクエストするには、gpu.nvidia.com DeviceClass を参照する ResourceClaimTemplate を作成し、それをポッド仕様で参照します。次の例では、GPU を 1 つだけリクエストしています。トポロジーに沿った EFA インターフェイスを備えた NVIDIA GPU を割り当てる手順については、「the section called “EFA と GPU/Neuron デバイスのトポロジー認識割り当て”」を参照してください。

```
apiVersion: resource.k8s.io/v1
kind: ResourceClaimTemplate
metadata:
  name: single-gpu
spec:
  spec:
    devices:
      requests:
      - name: gpu
        exactly:
          deviceClassName: gpu.nvidia.com
          count: 1
---
apiVersion: v1
kind: Pod
metadata:
  name: gpu-workload
spec:
  containers:
  - name: gpu-demo
    image: public.ecr.aws/amazonlinux/amazonlinux:2023-minimal
    command: ["/bin/sh", "-c"]
    args: ["nvidia-smi && tail -f /dev/null"]
    resources:
      claims:
      - name: gpu
  resourceClaims:
  - name: gpu
    resourceClaimTemplateName: single-gpu
  tolerations:
  - key: "nvidia.com/gpu"
    operator: "Exists"
```

```
    effect: "NoSchedule"
```

NVIDIA Kubernetes デバイスプラグインをインストールする

NVIDIA Kubernetes デバイスプラグインは、NVIDIA GPU を nvidia.com/gpu 拡張リソースとしてアドバタイズします。コンテナリソースのリクエストや制限の際に GPU をリクエストします。

前提条件

- Amazon EKS クラスター。
- ノードのインスタンスタイプが NVIDIA GPU であること (P や G インスタンスなど)。
- ノードのインスタンスタイプが NVIDIA GPU であり、EKS 最適化 AL2023 NVIDIA AMI を使用していること。EKS 最適化 Bottlerocket AMI には NVIDIA デバイスプラグインが含まれています。別途インストールする必要はありません。
- ノードに NVIDIA GPU 用のホストレベルのコンポーネントがインストールされていること。EKS 最適化 AL2023 または Bottlerocket NVIDIA AMI を使用する場合には、ホストレベルの NVIDIA ドライバー、CUDA ユーザーモードドライバー、コンテナツールキットがプリインストールされていること。
- コマンドライン環境に Helm がインストールされていること。詳細については、「Setup Helm instructions」を参照してください。
- クラスターと通信するように kubectl が設定されていること。詳細については、「the section called “kubectl をインストールまたは更新する”」を参照してください。

手順

1. NVIDIA デバイスプラグインの Helm チャートリポジトリを追加します。

```
helm repo add nvdp https://nvidia.github.io/k8s-device-plugin
```

2. ローカル Helm リポジトリを更新します。

```
helm repo update
```

3. NVIDIA Kubernetes デバイスプラグインをインストールします。

```
helm install nvdp nvdp/nvidia-device-plugin \
    --create-namespace \
```

```
    --namespace nvidia
```

オプション: GPU Feature Discovery (GFD) を有効にする

デバイスプラグインは nvidia.com/gpu 拡張リソースをアドバタイズし、GPU Pod を独自にスケジュールします。EKS 最適化 AL2023 NVIDIA AMI では、nvidia.com/gpu.present=true ノードラベルは nodeadm による起動時に既に適用されているため、GPU Feature Discovery (GFD) は基本的な GPU スケジューリングには必要ありません。
nvidia.com/gpu.product、nvidia.com/gpu.memory、nvidia.com/gpu.count、MIG プロファイルラベル、ドライバー/CUDA バージョンなど、詳細な GPU 属性でノードにラベルを付けたい場合は、--set gfd.enabled=true で GFD を有効にします。これらのラベルを使用すると、nodeSelector またはノードアフィニティを使用して特定の GPU タイプをターゲットにできます (例えば、ワークロードを A10G GPU または特定の MIG プロファイルにのみスケジュールするなど)。タイムスライシングや MIG などの GPU 共有設定もこれらのラベルを使用します。属性ベースのノード選択や GPU 共有が必要ない場合は、フラグを省略できます。

```
helm install nvdp nvdp/nvidia-device-plugin \
    --create-namespace \
    --namespace nvidia \
    --set gfd.enabled=true
```

ワークロードが GDRCopy を使用している場合に GDRCopy を有効にする

k8s-device-plugin v0.19.0 から v0.19.2 では、GDRCopy および MOFED 機能がデフォルトで有効になっていました。このデフォルトの有効化は k8s-device-plugin v0.19.3 で元に戻されました。元に戻した結果、Pod 仕様の NVIDIA_GDRCOPY=enabled 環境変数で GDRCopy (gdrdrv) をコンテナごとに有効にできなくなり、その変数は無視されるようになりました。
ワークロードが GDRCopy (GPUDirect RDMA コピー) を使用している場合は、gdrcopyEnabled=true を設定して、インストール時にデバイスプラグインで有効にする必要があります。

```
helm upgrade --install nvdp nvdp/nvidia-device-plugin \
    --namespace nvidia \
```

```
    --create-namespace \
    --set gdrcopyEnabled=true
```

(前述のように、GPU Feature Discovery ラベルも必要な場合は、`--set gfd.enabled=true` も追加します。)
GitHub の NVIDIA GPU Operator を通じて NVIDIA デバイスプラグインを管理する場合、gdrdrv カーネルモジュールがノードにロードされると、オペレーターは動的に `GDRCOPY_ENABLED=true` を設定します。
詳細については、GitHub の「NVIDIA k8s-device-plugin の問題 #1692」を参照してください。

> **Note**
>
> GitHub の NVIDIA GPU Operator を使用して NVIDIA Kubernetes デバイスプラグインをインストールおよび管理することもできます。これにより、GPU のプロビジョニングに必要なすべての NVIDIA ソフトウェアコンポーネントの管理が自動化されます。

4. DaemonSet という NVIDIA デバイスプラグインが実行されていることを確認します。

```
kubectl get ds -n nvidia nvdp-nvidia-device-plugin
```

```
NAME                        DESIRED   CURRENT   READY   UP-TO-DATE   AVAILABLE   NODE
 SELECTOR   AGE
nvdp-nvidia-device-plugin   2         2         2       2            2
 <none>          60s
```

5. ノードを調べて割り当て可能な GPU があることを確認します。

```
kubectl get nodes "-o=custom-
columns=NAME:.metadata.name,GPU:.status.allocatable.nvidia\.com/gpu"
```

出力例は次のとおりです。

```
NAME                                           GPU
ip-192-168-11-225.us-west-2.compute.internal   1
ip-192-168-24-96.us-west-2.compute.internal    1
```

ポッド内で NVIDIA GPU をリクエストする

デバイスプラグインを使用して NVIDIA GPU をリクエストするには、コンテナリソースのリクエストと制限で nvidia.com/gpu リソースを指定します。

```
apiVersion: v1
kind: Pod
metadata:
  name: nvidia-smi
spec:
  restartPolicy: OnFailure
  containers:
  - name: gpu-demo
    image: public.ecr.aws/amazonlinux/amazonlinux:2023-minimal
    command: ["/bin/sh", "-c"]
    args: ["nvidia-smi && tail -f /dev/null"]
    resources:
      limits:
        nvidia.com/gpu: 1
      requests:
        nvidia.com/gpu: 1
  tolerations:
  - key: "nvidia.com/gpu"
    operator: "Equal"
    value: "true"
    effect: "NoSchedule"
```

このテストを実行するには、マニフェストを適用してログを表示します。

```
kubectl apply -f nvidia-smi.yaml
kubectl logs nvidia-smi
```

出力例は次のとおりです。

```
+-----------------------------------------------------------------------------------------
+
| NVIDIA-SMI XXX.XXX.XX            Driver Version: XXX.XXX.XX     CUDA Version: XX.X
   |
|-----------------------------------------+------------------------
+----------------------+
| GPU  Name                 Persistence-M | Bus-Id          Disp.A | Volatile Uncorr.
 ECC |
```

```
| Fan  Temp   Perf          Pwr:Usage/Cap |           Memory-Usage | GPU-Util  Compute
 M. |
|                                         |                        |               MIG
 M. |
|=========================================+========================
+======================|
|   0  NVIDIA L4                      On  |   00000000:31:00.0 Off |
 0 |
| N/A   27C    P8             11W /   72W |       0MiB /  23034MiB |      0%
 Default |
|                                         |                        |
 N/A |
+-----------------------------------------+------------------------
+----------------------+

+---------------------------------------------------------------------------------------
+
| Processes:
   |
|  GPU   GI   CI              PID   Type   Process name                        GPU
 Memory |
|        ID   ID                                                               Usage
   |
|
=======================================================================================|
|  No running processes found
   |
+---------------------------------------------------------------------------------------
+
```

Amazon EKS の NVIDIA GPU でマルチインスタンス GPU (MIG) を使用する

マルチインスタンス GPU (MIG) は、単一の物理 GPU を複数の分離されたインスタンスにパーティション化する NVIDIA GPU のハードウェア機能です。パーティション化されたインスタンスの最大数は GPU によって異なります。各インスタンスには専用メモリ、コンピューティングユニット、メモリ帯域幅があるため、あるインスタンスで実行されるワークロードが別のインスタンスのワークロードに影響を与えることはありません。分離なしでソフトウェアのタイムマルチプレクシングを通じて GPU を共有するタイムスライシングとは異なり、MIG は Pod 間でハードウェアレベルのメモリと障害分離を提供します。

MIG は、予測可能なサービスの品質を必要とするマルチテナント推論やワークロードに最適です。NVIDIA Ampere (A100)、Hopper (H100 と H200)、および Blackwell GPU で使用できま

す。AWS では、これらは P ファミリーインスタンスタイプと、Blackwell ベースの g7 および g7e インスタンスタイプです。利用できるタイムゾーンの一覧については、「the section called “MIG 対応インスタンスタイプ”」を参照してください。

MIG は、次の場合に適しています:

- あるワークロードが別のワークロードに必要な GPU メモリを消費できないようにするために、メモリの分離が必要である。
- テナントが GPU ハードウェアを共有するけれども、テナントごとのサービス品質が必要なマルチテナント推論を実行する。
- A100、H100、H200、または Blackwell インスタンスでトレーニングを既に実行しており、トレーニングがアイドル状態のときに、これらの GPU を小規模な推論ワークロードに再利用したいと考えている。

次の場合は、別のアプローチを検討してください:

- ノードが、g5、g6、g6e ファミリーなど、MIG をサポートしていない GPU インスタンスタイプを使用している。代わりにタイムスライシングを使用します。
- メモリを分離する必要はなく、最も簡単な設定を望んでいる。代わりにタイムスライシングを使用します。
- GPU パーティションを中断することなく頻繁に変更する必要がある。MIG モードまたはパーティションレイアウトを変更するには、GPU のリセットが必要です。GPU オペレータはノードを再起動してこれを実行します。
- GPU 間の集合通信またはピアツーピア通信に依存するマルチ GPU トレーニングを実行している。MIG は NCCL またはクロス GPU P2P をサポートしていません。

考慮事項

本番環境で MIG を使用する前に、以下の考慮事項を確認してください。

一般的な考慮事項

- MIG 設定を変更するには、GPU のリセットが必要です。MIG モードの有効化または無効化、またはパーティションレイアウトの変更には GPU のリセットが必要なため、インプレースで変更することはできません。例えば、NVIDIA GPU Operator の MIG Manager は、ノード上の GPU Pod を停止し、MIG モードを変更するために再起動が必要な場合にノードを再起動することで、設定変更を適用します。

- コンピューティングはインスタンスサイズに厳密には比例しません。メモリ帯域幅とキャッシュ動作はプロファイルによって異なるため、1g インスタンスはすべてのワークロードで GPU 全体のスループットを比例配分するわけではありません。パーティションのサイズを設定する前に、使用するプロファイルでワークロードをベンチマークします。
- タイムスライシングは MIG インスタンスには影響しません。MIG インスタンスはすでにハードウェア分離されており、タイムスライスを通じてそれ以上共有することはできません。MIG デバイスで `TimeSlicing` 共有戦略をリクエストしても、ハードウェアの動作は変更されません。コンテナ間で単一の MIG インスタンスを共有するには、代わりに NVIDIA Multi-Process Service (MPS) を使用します。
- クロス GPU 通信の制限。MIG が有効になっている場合、異なる GPU 上の MIG インスタンスは GPU 間のピアツーピア (P2P) 通信を使用できません。また、NCCL は MIG と連携しません。テンソル並列マルチ GPU トレーニングなど、GPU 間の集合通信または P2P に依存するマルチ GPU ワークロードには、代わりに GPU 全体が必要です。詳細については、NVIDIA ウェブサイトの NVIDIA MIG ユーザーガイドの「アプリケーションに関する考慮事項」を参照してください。

NVIDIA デバイスプラグインに関する考慮事項

- ポッドリソースリクエストは、戦略と一致する必要があります。単一戦略では、Pod は `nvidia.com/gpu` をリクエストします。混合戦略では、Pod は `nvidia.com/mig-1g.10gb` などのプロファイル固有のリソースをリクエストします。ノードがアドバタイズしないプロファイルをリクエストする Pod は `Pending` 状態のままになります。`kubectl describe node <node-name>` でアドバタイズされたリソースを確認します。
- AL2023 のスタンドアロンデバイスプラグイン。例えば、クラスターのセットアップの一部として NVIDIA デバイスプラグインを個別にインストールする場合は、MIG ノードからそれを除外して、GPU Operator が管理するデバイスプラグインと競合しないようにします。`nvidia.com/mig.config` ラベルを持つノードを除外するノードアフィニティルールをスタンドアロンデバイスプラグインに追加します。
- EKS Auto Mode ではサポートされていません。EKS Auto Mode は NVIDIA デバイスプラグインを管理し、その設定を公開しません (the section called “加速ワークロードのデプロイ” を参照)。EKS Auto Mode ノードで MIG を有効にすることはできません。AMI とデバイスプラグインの設定を制御するセルフマネージド Karpenter ノードまたはマネージドノードグループで MIG を設定します。

NVIDIA DRA ドライバーに関する考慮事項

- 静的 MIG には、事前に作成されたインスタンスが必要です。静的 MIG では、DRA ドライバーは既存の MIG インスタンスを割り当てますが、MIG モードを有効にしたり GPU をパーティション化したりしません。まず MIG モードを有効にし、インスタンスを作成する必要があります。例えば、NVIDIA GPU Operator の MIG Manager または `nvidia-smi` を使用します。詳細については、「the section called “NVIDIA DRA ドライバーで MIG を使用する”」を参照してください。
- 動的 MIG はアルファ機能です。動的 MIG を使用すると、ドライバーはワークロードリクエストに応じてオンデマンドで MIG パーティションを作成および破棄します。`DynamicMIG` 機能ゲートが必要です。これはデフォルトで無効になっています。詳細については、「the section called “NVIDIA DRA ドライバーで MIG を使用する”」を参照してください。
- Bottlerocket の組み込みデバイスプラグインを無効にします。DRA ドライバーは、同じノード上の NVIDIA デバイスプラグインと一緒に実行することはできません。Bottlerocket で、組み込みデバイスプラグインを無効にします。これには Bottlerocket バージョン 1.63.0 以降が必要です。詳細については、「the section called “NVIDIA DRA ドライバーをインストールする”」を参照してください。
- コンピューティングのサポート。NVIDIA DRA ドライバーは、Karpenter、EKS マネージドノードグループ、またはセルフマネージドノードの静的キャパシティのプロビジョニングでサポートされており、EKS Auto Mode ではサポートされていません。詳細については、Karpenter ウェブサイトの「Karpenter 静的 NodePool ドキュメント」を参照してください。

MIG 対応インスタンスタイプ

AWS では、次のインスタンスタイプが MIG 対応 GPU を提供します。

インスタンスタイプ	GPU	GPU メモリ
`p4d.24xlarge`	8 x NVIDIA A100 40 GB	320 GB
`p4de.24xlarge`	8 x NVIDIA A100 80 GB	640 GB
`p5.48xlarge`	8 x NVIDIA H100 80 GB	640 GB
`p5e.48xlarge`	8 x NVIDIA H200	1128 GB
`p5en.48xlarge`	8 x NVIDIA H200	1128 GB
`p6-b200.48xlarge`	8 x NVIDIA Blackwell B200	1432 GB

インスタンスタイプ	GPU	GPU メモリ
p6-b300.48xlarge	8 x NVIDIA Blackwell Ultra B300	2144 GB
g7.48xlarge	8 x NVIDIA RTX PRO 4500 Blackwell Server Edition	256 GB
g7e.48xlarge	8 x NVIDIA RTX PRO 6000 Blackwell Server Edition	768 GB

MIG は、g5、g6、または g6e ファミリーでは使用できません。MIG 対応 NVIDIA GB200 GPU を使用する p6e-gb200 UltraServers については、the section called “EKS で P6e-GB200 を使用する” を参照してください。

> **Note**
>
> g7 インスタンスタイプには NVIDIA ドライバーバージョン 595 以降が必要です。EKS 最適化 AL2023 NVIDIA AMI には g7 をサポートするドライバーバージョンが含まれており、各ノードは起動時に自動的に選択します。詳細については、「EKS 最適化 NVIDIA AMI」を参照してください。

MIG インスタンスは、命名パターン <slices>g.<memory>gb を使用するプロファイルによって記述されます。ここで、<slices> はコンピューティングスライスの数で、<memory> はギガバイト単位でのインスタンスのメモリです。例えば、3g.40gb プロファイルは 7 個のコンピューティングスライスのうち 3 個と 40 GB のメモリを提供します。各 GPU がサポートするプロファイルは、ハードウェアによって固定されます。完全なリストについては、NVIDIA ウェブサイトの「NVIDIA マルチインスタンス GPU ユーザーガイド」を参照してください。

インスタンスタイプあたりの MIG プロファイル

ノードで使用できる MIG プロファイルは、インスタンスタイプの GPU によって異なります。以下のセクションでは、MIG 対応の各 Amazon EC2 インスタンスタイプのプロファイルを一覧表示します。各プロファイルの最大インスタンスは、単一の GPU で作成できるそのプロファイルの最大インスタンス数で、インスタンスあたりのメモリは、それぞれに割り当てられた GPU メモリです。

p4d.24xlarge – NVIDIA A100 40 GB

プロファイル	コンピューティングスライス	インスタンスあたりのメモリ	最大インスタンス
1g.5gb	1/7	5 GB	7
1g.10gb	1/7	10 GB	4
2g.10gb	2/7	10 GB	3
3g.20gb	3/7	20 GB	2
4g.20gb	4/7	20 GB	1
7g.40gb	7/7	40 GB	1

p4de.24xlarge – NVIDIA A100 80 GB

プロファイル	コンピューティングスライス	インスタンスあたりのメモリ	最大インスタンス
1g.10gb	1/7	10 GB	7
1g.20gb	1/7	20 GB	4
2g.20gb	2/7	20 GB	3
3g.40gb	3/7	40 GB	2
4g.40gb	4/7	40 GB	1
7g.80gb	7/7	80 GB	1

p5.48xlarge – NVIDIA H100 80 GB

プロファイル	コンピューティングスライス	インスタンスあたりのメモリ	最大インスタンス
1g.10gb	1/7	10 GB	7
1g.20gb	1/7	20 GB	4
2g.20gb	2/7	20 GB	3
3g.40gb	3/7	40 GB	2
4g.40gb	4/7	40 GB	1
7g.80gb	7/7	80 GB	1

p5e.48xlarge と **p5en.48xlarge** – NVIDIA H200 141 GB

プロファイル	コンピューティングスライス	インスタンスあたりのメモリ	最大インスタンス
1g.18gb	1/7	18 GB	7
1g.35gb	1/7	35 GB	4
2g.35gb	2/7	35 GB	3
3g.71gb	3/7	71 GB	2
4g.71gb	4/7	71 GB	1
7g.141gb	7/7	141 GB	1

p6-b200.48xlarge – NVIDIA Blackwell B200 180 GB

プロファイル	コンピューティングスライス	インスタンスあたりのメモリ	最大インスタンス
1g.23gb	1/7	23 GB	7
1g.45gb	1/7	45 GB	4
2g.45gb	2/7	45 GB	3
3g.90gb	3/7	90 GB	2
4g.90gb	4/7	90 GB	1
7g.180gb	7/7	180 GB	1

p6-b300.48xlarge – NVIDIA Blackwell Ultra B300

p6-b300.48xlarge は、各 GPU を 7 個の 32 GB のインスタンス、4 個の 67 GB、2 個の 135 GB、または 1 個の 270 GB のインスタンスへのパーティション化をサポートする HGX B300 を使用します。これらのサイズは暫定的なものであり、変更される可能性があります。プロファイルの詳細については、NVIDIA ウェブサイトの「NVIDIA がサポートする MIG プロファイル」を参照してください。

g7.48xlarge – NVIDIA RTX PRO 4500 Blackwell Server Edition 32 GB

プロファイル	コンピューティングスライス	インスタンスあたりのメモリ	最大インスタンス
1g.16gb	1/2	16 GB	2
2g.32gb	2/2	32 GB	1

RTX PRO 4500 Blackwell は、グラフィックス対応 (+gfx) およびメディアエンジン (+me.all、-me) プロファイルバリアントもサポートしています。詳細なリストについては、NVIDIA ウェブサイトの「NVIDIA がサポートする MIG プロファイル」を参照してください。

g7e.48xlarge – NVIDIA RTX PRO 6000 Blackwell Server Edition 96 GB

プロファイル	コンピューティングスライス	インスタンスあたりのメモリ	最大インスタンス
1g.24gb	1/4	24 GB	4
2g.48gb	2/4	48 GB	2
4g.96gb	4/4	96 GB	1

RTX PRO 6000 Blackwell Server Edition は、グラフィックス対応 (+gfx) およびメディアエンジン (+me.all、-me) プロファイルバリアントもサポートしています。詳細なリストについては、NVIDIA ウェブサイトの「NVIDIA がサポートする MIG プロファイル」を参照してください。

MIG 戦略

NVIDIA DRA ドライバーと NVIDIA デバイスプラグインは、さまざまな方法で MIG インスタンスを Kubernetes に公開します。デバイスプラグインはノード全体の MIG 戦略設定を使用しますが、DRA ドライバーは属性によってインスタンスを選択するため、同等の設定はありません。この違いを理解することは、2 つのモデルから選択する上で重要です。

NVIDIA DRA ドライバー

NVIDIA DRA ドライバーは単一または混合戦略の概念を使用しないため、同等の設定を構成することはできません。MIG インスタンスをカウントされたリソースとしてアドバタイズする代わりに、ドライバーは各インスタンスを mig.nvidia.com DeviceClass のデバイスとして、その profile などの属性とともに発行します。Pod は、the section called “NVIDIA DRA ドライバーで MIG を使用する” に示すように、これらの属性を ResourceClaim または ResourceClaimTemplate の共通表現言語 (CEL) セレクターと照合してインスタンスを選択します。

選択はインスタンスごとに行われるため、混合プロファイルノードは戦略モードの切り替えなしで機能します。単一の GPU を複数の異なるプロファイルに分割でき、各クレームで必要なプロファイルが選択されます。DRA ドライバーにとって重要な選択は、単一戦略か混合戦略ではなく、静的 MIG

か動的 MIG であり、MIG インスタンスを事前に作成するか、ドライバーがオンデマンドで作成するかを制御します。詳細については、「the section called “NVIDIA DRA ドライバーで MIG を使用する”」を参照してください。

NVIDIA デバイスプラグイン

NVIDIA デバイスプラグインは、2 つの戦略のいずれかを使用して MIG インスタンスを Kubernetes にアドバタイズします。デバイスプラグインは MIG インスタンスをノードレベルの拡張リソースとして公開しており、整数カウントのみを持ちインスタンスごとの属性を持たないため、戦略によってそれらのリソースに名前を付ける方法が決まります。

- 単一戦略 – ノード上のすべての GPU は、同じ MIG プロファイルを使用します。デバイスプラグインは各インスタンスを `nvidia.com/gpu` リソースとしてアドバタイズし、Pod は専用 GPU の場合と同様に `nvidia.com/gpu: 1` をリクエストします。既存のマニフェストは変更されません。Bottlerocket と AL2023 はどちらも単一戦略をサポートしています。
- 混合戦略 – 同じノード上の GPU は、異なる MIG プロファイルを使用できます。デバイスプラグインは、各プロファイルを `nvidia.com/mig-1g.10gb` や `nvidia.com/mig-3g.40gb` などの個別のリソースとしてアドバタイズし、Pod は必要な特定のプロファイルをリクエストします。Bottlerocket の組み込み NVIDIA デバイスプラグインでは、混合戦略を使用できません。詳細については、GitHub の「Bottlerocket GitHub の問題 #4483」を参照してください。

NVIDIA DRA ドライバーで MIG を使用する

NVIDIA DRA ドライバーを使用して MIG インスタンスを割り当てると、Pod はデバイスプラグインの `nvidia.com/mig-<profile>` 拡張リソースではなく、`ResourceClaim` または `ResourceClaimTemplate` を介して MIG インスタンスをリクエストします。

DRA ドライバーは、カウントされたリソースとしてではなく属性でインスタンスを記述するため、デバイスプラグインが必要とする単一戦略または混合戦略を使用しません (the section called “MIG 戦略” を参照)。ドライバーは、各 MIG インスタンスを `mig.nvidia.com` `DeviceClass` 内の `gpu.nvidia.com/type` 属性が `mig` のデバイスとして公開し、`profile` (`1g.5gb` など) や物理 GPU の `parentUUID` などのインスタンスごとの属性をアドバタイズします。これらの属性を共通表現言語 (CEL) セレクターと照合して、特定のプロファイルをリクエストしたり、複数のインスタンスを同じ GPU に保持したりできます。

DRA ドライバーは、次の 2 つのモードのいずれかで MIG インスタンスを割り当てます。

- 静的 MIG – MIG モードを有効にし、ドライバーの起動前にノードに MIG インスタンスを作成します。例えば、the section called “NVIDIA デバイスプラグインで AL2023 ノードの MIG を使用する”

で説明されているように、NVIDIA GPU Operator の MIG Manager を使用します。ドライバーは既存のインスタンスを検出して Pod に割り当てますが、ノードの MIG 設定は変更しません。ドライバーの起動後に追加されたインスタンスは、GPU kubelet プラグインが再起動するまで検出されません。静的 MIG はデフォルトであり、機能ゲートは必要ありません。

- 動的 MIG – ドライバーはワークロードリクエストに応じてオンデマンドで MIG パーティションを作成および破棄するため、GPU を事前にパーティション化する必要はありません。動的 MIG は、デフォルトで無効になっているアルファ機能です。次のセクションに示すのと同じ `ResourceClaimTemplate` セレクターを使用してプロファイルをリクエストすると、ドライバーはリクエストを満たすために GPU をパーティション化します。

考慮事項

- 動的 MIG は、ノードの静的検出を置き換えます。ドライバーはすべてのパーティションを管理し、GPU kubelet プラグインの起動時に作成されなかった MIG パーティションを破棄します。保持する事前に作成されたパーティションがあるノードで動的 MIG を有効にしないでください。また、プラグインの実行中に `mig-parted` または `nvidia-smi mig` を実行しないでください。それは、手動の変更がドライバーのパーティション状態と競合し、Pod の準備またはクリーンアップが失敗する可能性があるからです。
- 動的 MIG はアルファ状態であり、NVIDIA DRA ドライバーのインストール時に機能ゲートを有効にする必要があります。手順については、the section called “NVIDIA DRA ドライバーをインストールする” を参照してください。
- Hopper (H100 および H200) 以降のアーキテクチャは、オンデマンドで MIG モードを有効にします。以前の世代では、Ampere (A100) GPU など、オンデマンドで MIG モードを有効にできません。
- 動的 MIG は、Kubernetes バージョン 1.36 以降でデフォルトで有効になっている、Kubernetes のパーティション化可能なデバイス機能 (GitHub の KEP-4815) に依存しています。以前のバージョンでは、この機能はデフォルトで有効になっていないため、スケジューラは動的に作成された MIG デバイスを割り当てることができません。

前提条件

- Amazon EKS クラスターで Kubernetes バージョン 1.34 以降を実行していて、Karpenter、EKS マネージドノードグループ、またはセルフマネージドノードグループによって静的キャパシティがプロビジョニングされていること。

- MIG モードが有効で GPU が MIG インスタンスにパーティション化された MIG 対応 P ファミリーノード。静的 MIG については、NVIDIA ウェブサイトの「NVIDIA GPU Operator の MIG Manager」を参照してください。
- the section called “NVIDIA DRA ドライバーをインストールする” で説明されているようにインストールされた NVIDIA DRA ドライバー。静的 MIG パーティショニングを使用していない場合は、オプションで動的 MIG が有効にされている。

手順

次の例は、静的または動的な MIG および NVIDIA DRA ドライバーで使用できます。

1. `mig.nvidia.com` DeviceClass から MIG インスタンスをリクエストする ResourceClaimTemplate と、それを参照する Pod を作成します。この例では、プロファイルを制限せずに、使用可能な MIG インスタンスをリクエストします。

```
cat <<EOF | kubectl apply -f -
apiVersion: resource.k8s.io/v1
kind: ResourceClaimTemplate
metadata:
  name: mig-profile-any
spec:
  spec:
    devices:
      requests:
      - name: mig
        exactly:
          deviceClassName: mig.nvidia.com
          count: 1
---
apiVersion: v1
kind: Pod
metadata:
  name: mig-dra-pod
spec:
  tolerations:
    - key: nvidia.com/gpu
      operator: Exists
      effect: NoSchedule
  containers:
    - name: cuda
      image: nvidia/cuda:12.6.0-base-ubuntu22.04
```

```
      command: ["nvidia-smi", "-L"]
      resources:
        claims:
        - name: mig
  resourceClaims:
    - name: mig
      resourceClaimTemplateName: mig-profile-any
  restartPolicy: OnFailure
EOF
```

2. Pod に単一の MIG インスタンスが割り当てられていることを確認します。

```
kubectl logs mig-dra-pod
```

出力例は次のとおりです。Pod は、パーティション化された GPU から 1 つの MIG インスタンスを確認します。

```
GPU 0: NVIDIA A100-SXM4-40GB (UUID: GPU-edd63844-8488-f76a-f6e0-1027a7319a88)
  MIG 2g.10gb     Device  0: (UUID: MIG-a5ad493e-e7e8-5675-9381-1d0f5311a456)
```

特定の MIG プロファイルをリクエストする

使用可能なインスタンスの代わりに特定のプロファイルをリクエストするには、profile 属性に一致する CEL セレクターを追加します。次の ResourceClaimTemplate は 1g.5gb インスタンスをリクエストし、Pod はそれを参照します。

```
cat <<EOF | kubectl apply -f -
apiVersion: resource.k8s.io/v1
kind: ResourceClaimTemplate
metadata:
  name: mig-profile-1g.5gb
spec:
  spec:
    devices:
      requests:
      - name: mig
        exactly:
          deviceClassName: mig.nvidia.com
          selectors:
          - cel:
              expression: "device.attributes['gpu.nvidia.com'].profile == '1g.5gb'"
```

```
---
apiVersion: v1
kind: Pod
metadata:
  name: mig-profile-pod
spec:
  tolerations:
    - key: nvidia.com/gpu
      operator: Exists
      effect: NoSchedule
  containers:
    - name: cuda
      image: nvidia/cuda:12.6.0-base-ubuntu22.04
      command: ["nvidia-smi", "-L"]
      resources:
        claims:
        - name: mig
  resourceClaims:
    - name: mig
      resourceClaimTemplateName: mig-profile-1g.5gb
  restartPolicy: OnFailure
EOF
```

同じ GPU から複数の MIG インスタンスをリクエストする

単一のクレーム内の複数の MIG インスタンスが同じ物理 GPU から取得されるようにするには、matchAttribute: "gpu.nvidia.com/parentUUID" で constraints ブロックを追加します。次の ResourceClaimTemplate は、1g.5gb インスタンスと 2g.10gb インスタンスを同じ GPU からリクエストし、Pod はクレームを参照します。コンテナは特定のリクエストに名前を付けることなくクレームを参照するため、両方の MIG インスタンスを受け取ります。

```
cat <<EOF | kubectl apply -f -
apiVersion: resource.k8s.io/v1
kind: ResourceClaimTemplate
metadata:
  name: multi-mig
spec:
  spec:
    devices:
      requests:
      - name: mig-small
        exactly:
          deviceClassName: mig.nvidia.com
```

```
          selectors:
          - cel:
              expression: "device.attributes['gpu.nvidia.com'].profile == '1g.5gb'"
      - name: mig-medium
        exactly:
          deviceClassName: mig.nvidia.com
          selectors:
          - cel:
              expression: "device.attributes['gpu.nvidia.com'].profile == '2g.10gb'"
      constraints:
      - requests: []
        matchAttribute: "gpu.nvidia.com/parentUUID"
---
apiVersion: v1
kind: Pod
metadata:
  name: multi-mig-pod
spec:
  tolerations:
    - key: nvidia.com/gpu
      operator: Exists
      effect: NoSchedule
  containers:
    - name: cuda
      image: nvidia/cuda:12.6.0-base-ubuntu22.04
      command: ["nvidia-smi", "-L"]
      resources:
        claims:
        - name: mig
  resourceClaims:
    - name: mig
      resourceClaimTemplateName: multi-mig
  restartPolicy: OnFailure
EOF
```

NVIDIA デバイスプラグインで Bottlerocket ノードの MIG を使用する

Bottlerocket では、EKS 最適化高速 AMI には NVIDIA デバイスプラグインが含まれています。`settings.kubelet-device-plugins.nvidia` 設定を通じて単一戦略で MIG を有効にします。Bottlerocket は、バージョン 1.34.0 以降で MIG をサポートしています。

前提条件

- Amazon EKS クラスター。次の手順では、EKS 最適化 Bottlerocket NVIDIA AMI バージョン 1.34.0 以降を使用して、MIG 対応 P ファミリーノードをプロビジョニングします。
- Karpenter がクラスターにインストールおよび設定されていること。それは、次の手順で、Karpenter `EC2NodeClass` を使用して Bottlerocket ノードのユーザーデータに MIG 設定を指定するからです。詳細については、Karpenter ウェブサイトの「Karpenter の開始方法」を参照してください。
- クラスターと通信するように `kubectl` が設定されていること。詳細については、「the section called “kubectl をインストールまたは更新する”」を参照してください。

手順

GPU ノードの Bottlerocket ユーザーデータに MIG パーティショニング設定を追加します。ユーザーデータの提供方法は、ノードのプロビジョニング方法によって異なります。次の例は、`p4d.24xlarge` ノード用の Karpenter `EC2NodeClass` を示しています。

```
cat <<EOF | kubectl apply -f -
apiVersion: karpenter.k8s.aws/v1
kind: EC2NodeClass
metadata:
  name: gpu-bottlerocket-mig
spec:
  amiFamily: Bottlerocket
  amiSelectorTerms:
    - alias: bottlerocket@latest
  role: eksctl-KarpenterNodeRole-<cluster-name>
  subnetSelectorTerms:
    - tags:
        karpenter.sh/discovery: <cluster-name>
  securityGroupSelectorTerms:
    - tags:
        karpenter.sh/discovery: <cluster-name>
  userData: |
    [settings.kubelet-device-plugins.nvidia]
    device-partitioning-strategy = "mig"

    [settings.kubelet-device-plugins.nvidia.mig.profile]
    "a100.40gb" = "2g.10gb"
EOF
```

これらの設定でプロビジョニングされたノードがクラスターを結合すると、GPU で MIG モードが有効になり、各 GPU が 2g.10gb インスタンスにパーティション化され、デバイスプラグインは結果のインスタンスを nvidia.com/gpu リソースとしてアドバタイズします。p4d.24xlarge には 8 個の A100 GPU があり、それぞれが 3 個の 2g.10gb インスタンスをサポートしているため、ノードは nvidia.com/gpu: 24 をアドバタイズします。

> **Note**
>
> mig.profile 設定は、a100.40gb や h100.80gb などの GPU モデルによってキー指定されます。mig.profile 設定がない場合、GPU は MIG モードを有効にし、最大のプロファイルを使用します。Bottlerocket は単一戦略を使用するため、ノード上のすべての GPU は同じプロファイルを使用します。同じノードで異なるプロファイルを使用するには (混合戦略)、NVIDIA GPU Operator で AL2023 パスを使用します。

NVIDIA デバイスプラグインで AL2023 ノードの MIG を使用する

AL2023 では、次の手順で NVIDIA GPU Operator を使用して NVIDIA デバイスプラグインと MIG Manager をインストールします。MIG Manager は MIG モードを有効にし、指定した設定に従って GPU をパーティション化します。次に NVIDIA デバイスプラグインは、結果のインスタンスを Kubernetes にアドバタイズします。GPU Operator は、単一戦略と混合戦略の両方をサポートしています。

EKS 最適化 AL2023 NVIDIA AMI には NVIDIA ドライバーとツールキットが既に含まれているため、GPU Operator でドライバー管理を無効にして、プリインストールされているドライバーとの競合を回避します。または、GPU Operator を使用せずに NVIDIA デバイスプラグインと MIG Manager を自分でインストールおよび管理することもできます。

前提条件

- Amazon EKS クラスター。次の手順では、EKS 最適化 AL2023 NVIDIA AMI を使用して MIG 対応 P ファミリーノード (p4d.24xlarge など) をプロビジョニングします。
- Karpenter がクラスターにインストールおよび設定されていること。それは、この手順で、Karpenter EC2NodeClass と NodePool を作成して GPU ノードをプロビジョニングするからです。詳細については、Karpenter ウェブサイトの「Karpenter の開始方法」を参照してください。
- コマンドライン環境に Helm がインストールされていること。詳細については、「Setup Helm instructions」を参照してください。

- クラスターと通信するように kubectl が設定されていること。詳細については、「the section called “kubectl をインストールまたは更新する”」を参照してください。

手順

1. AL2023 P ファミリー GPU ノード用の EC2NodeClass と NodePool を作成します。AL2023 では、後のステップで GPU Operator によって MIG パーティショニングが適用されるため、これは標準の AL2023 GPU ノードクラスです。次の例では、EKS 最適化 AL2023 NVIDIA AMI を使用して p4d.24xlarge ノードをプロビジョニングします。

```
cat <<EOF | kubectl apply -f -
apiVersion: karpenter.k8s.aws/v1
kind: EC2NodeClass
metadata:
  name: gpu-mig-al2023
spec:
  amiFamily: AL2023
  amiSelectorTerms:
    - alias: al2023@latest
  role: eksctl-KarpenterNodeRole-<cluster-name>
  subnetSelectorTerms:
    - tags:
        karpenter.sh/discovery: <cluster-name>
  securityGroupSelectorTerms:
    - tags:
        karpenter.sh/discovery: <cluster-name>
  tags:
    karpenter.sh/discovery: <cluster-name>
---
apiVersion: karpenter.sh/v1
kind: NodePool
metadata:
  name: gpu-mig-al2023
spec:
  template:
    spec:
      nodeClassRef:
        group: karpenter.k8s.aws
        kind: EC2NodeClass
        name: gpu-mig-al2023
      taints:
        - key: nvidia.com/gpu
```

```
          effect: NoSchedule
      requirements:
        - key: karpenter.sh/capacity-type
          operator: In
          values: ["spot", "on-demand"]
        - key: node.kubernetes.io/instance-type
          operator: In
          values: ["p4d.24xlarge"]
        - key: kubernetes.io/arch
          operator: In
          values: ["amd64"]
  limits:
    cpu: 1000
    memory: 5000Gi
EOF
```

2. NVIDIA Helm リポジトリを追加します。

```
helm repo add nvidia https://nvidia.github.io/gpu-operator
helm repo update
```

3. ドライバー管理を無効にし、混合戦略を選択し、適用する MIG プロファイルを定義する gpu-operator-values.yaml ファイルを作成します。次の例では、p4d.24xlarge ノード上の 8 個の GPU のうち 4 個をパーティション化し、残りのすべてをそのままにする p4d-half-balanced 設定を定義します。

```
cat <<EOF > gpu-operator-values.yaml
driver:
  enabled: false
toolkit:
  enabled: false
devicePlugin:
  enabled: true
nfd:
  enabled: true
gfd:
  enabled: true
mig:
  strategy: mixed
migManager:
  enabled: true
  env:
    - name: WITH_REBOOT
```

```
      value: "true"
  config:
    create: true
    name: custom-mig-parted-configs
    default: all-disabled
    data:
      config.yaml: |-
        version: v1
        mig-configs:
          all-disabled:
            - devices: all
              mig-enabled: false
          p4d-half-balanced:
            - devices: [0, 1, 2, 3]
              mig-enabled: true
              mig-devices:
                "1g.5gb": 2
                "2g.10gb": 1
                "3g.20gb": 1
            - devices: [4, 5, 6, 7]
              mig-enabled: false
EOF
```

4. 値ファイルを使用して GPU Operator をインストールします。

```
helm install gpu-operator nvidia/gpu-operator \
    --namespace gpu-operator \
    --create-namespace \
    --values gpu-operator-values.yaml
```

5. MIG 対応ノードに、適用するプロファイル設定をラベル付けします。MIG Manager コンポーネントはこのラベルを監視し、それに応じて GPU をパーティション化し、ノードを再起動して変更を適用します。

```
kubectl label nodes -l node.kubernetes.io/instance-type=p4d.24xlarge \
    nvidia.com/mig.config=p4d-half-balanced --overwrite
```

6. GPU Operator が GPU をパーティション化した後、Pod は `nvidia.com/gpu` ではなくそのリソース名で特定の MIG プロファイルをリクエストします。次の例では、1 つの `1g.5gb` インスタンスをリクエストする Pod を実行します。

```
cat <<EOF | kubectl apply -f -
```

```
apiVersion: v1
kind: Pod
metadata:
  name: mig-inference
spec:
  tolerations:
    - key: nvidia.com/gpu
      operator: Exists
      effect: NoSchedule
  containers:
    - name: cuda
      image: nvidia/cuda:12.6.0-base-ubuntu22.04
      command: ["nvidia-smi", "-L"]
      resources:
        limits:
          nvidia.com/mig-1g.5gb: 1
EOF
```

キャパシティ予約とプロファイルごとのワークロードを持つマネージドノードグループなど、P ファミリーインスタンスの完全な混合戦略の設定例については、「Amazon EKS ベストプラクティスガイド」の「MIG」セクションを参照してください。

MIG がアクティブであることを確認する

MIG 対応ノードが Ready になったら、MIG モードが GPU でアクティブであり、ノードが予想される MIG リソースをアドバタイズしていることを確認します。

1. ノードが MIG リソースをアドバタイズしていることを確認します。単一戦略では、ノードはインスタンスを nvidia.com/gpu としてレポートします。混合戦略では、ノードは nvidia.com/mig-1g.10gb などのプロファイル固有のリソースをレポートします。

   ```
   kubectl describe node <node-name> | grep nvidia.com
   ```

2. MIG 対応ノードの Pod から nvidia-smi を実行して、MIG モードがアクティブであることを確認します。

   nvidia-smi は、MIG モードがオンになっている GPU の MIG M.: Enabled をレポートし、それぞれに設定されている MIG インスタンスを一覧表示します。以下は、MIG が有効で、3g.20gb、2g.10gb、および 1g.5gb インスタンスにパーティション化された A100 40 GB GPU からの出力例です。

```
+-----------------------------------------------------------------------------------------
+
| NVIDIA-SMI 580.159.03             Driver Version: 580.159.03     CUDA Version: 13.0
     |
+-----------------------------------------+------------------------
+----------------------+
| GPU  Name                 Persistence-M | Bus-Id          Disp.A | Volatile Uncorr.
 ECC |
| Fan  Temp   Perf          Pwr:Usage/Cap |           Memory-Usage | GPU-Util
 Compute M. |
|                                         |                        |
 MIG M. |
|=========================================+========================
+======================|
|   0  NVIDIA A100-SXM4-40GB          On  |   00000000:10:1C.0 Off |
  On |
| N/A   35C    P0             65W /  400W |     213MiB /  40960MiB |     N/A
 Default |
|                                         |                        |
 Enabled |
+-----------------------------------------+------------------------
+----------------------+

+-----------------------------------------------------------------------------------------
+
| MIG devices:
     |
+------------------+----------------------------------+-----------
+-----------------------+
| GPU  GI  CI  MIG |              Shared Memory-Usage |        Vol|        Shared
     |
|      ID  ID  Dev |                Shared BAR1-Usage | SM     Unc| CE ENC  DEC  OFA
 JPG |
|                  |                                 |        ECC|
     |
|==================+==================================+===========
+======================|
|  0    1   0   0  |             107MiB / 20096MiB    | 42      0 |  3   0    2    0
   0 |
|                  |               0MiB / 12211MiB    |           |
     |
+------------------+----------------------------------+-----------
+-----------------------+
```

```
|  0    5   0   1  |              71MiB /  9984MiB    | 28      0 |  2   0    1    0
   0 |
|                  |               0MiB /  6105MiB    |           |
     |
+------------------+----------------------------------+-----------
+-----------------------+
|  0   13   0   2  |              36MiB /  4864MiB    | 14      0 |  1   0    0    0
   0 |
|                  |               0MiB /  3052MiB    |           |
     |
+------------------+----------------------------------+-----------
+-----------------------+
```

混合戦略では、Pod はリクエストした MIG インスタンスのみを確認し、ノードの完全な GPU レイアウトは確認しません。前のステップの mig-inference Pod は 1 つの nvidia.com/mig-1g.5gb インスタンスをリクエストしたため、その Pod 内の nvidia-smi -L では 1 つの MIG デバイスが一覧表示されます。

```
kubectl logs mig-inference
```

出力例は次のとおりです。

```
GPU 0: NVIDIA A100-SXM4-40GB (UUID: GPU-5b7ce860-1951-004c-4881-6ac6997df770)
  MIG 1g.5gb      Device  0: (UUID: MIG-9b065868-21b6-5b8d-8ab9-e99089ed472c)
```

ノード上のすべての GPU の完全なパーティションレイアウトを表示するには、ワークロード Pod 内の代わりにホストで nvidia-smi -L を実行します。次のコマンドは、ノードで特権デバッグ Pod を起動し、ホストの nvidia-smi を実行します。*node-name* を MIG 対応ノードの名前に置き換えます。

```
kubectl debug node/<node-name> -it --profile=sysadmin --image=nvidia/cuda:12.6.0-
base-ubuntu22.04 -- chroot /host nvidia-smi -L
```

以下は、p4d-half-balanced 設定の出力例です。最初の 4 個の GPU は MIG インスタンスにパーティション化され、残りの 4 個が全 GPU です。各 MIG 行は、独自の UUID、メモリ、コンピューティングスライスを持つハードウェア分離インスタンスです。

```
GPU 0: NVIDIA A100-SXM4-40GB (UUID: GPU-5b7ce860-1951-004c-4881-6ac6997df770)
  MIG 3g.20gb     Device  0: (UUID: MIG-7dc16162-7ba2-5894-abde-d753dc8ecf56)
```

```
  MIG 2g.10gb     Device  1: (UUID: MIG-56cfe1f0-0662-50e4-a5f7-111107e4d5e6)
  MIG 1g.5gb      Device  2: (UUID: MIG-1409727e-2ffa-5fd4-9586-204c9e2b36d5)
  MIG 1g.5gb      Device  3: (UUID: MIG-9b065868-21b6-5b8d-8ab9-e99089ed472c)
GPU 1: NVIDIA A100-SXM4-40GB (UUID: GPU-73692a43-dd2d-f1a1-b0df-2f4734e2a87d)
  MIG 3g.20gb     Device  0: (UUID: MIG-745fd93a-9582-57a6-8b3b-9782d289ca1b)
  MIG 2g.10gb     Device  1: (UUID: MIG-8440294e-c4a0-5687-add5-2cc506babb2f)
  MIG 1g.5gb      Device  2: (UUID: MIG-559810c0-f2bb-5ca2-b529-47228d99437f)
  MIG 1g.5gb      Device  3: (UUID: MIG-a91cf3f9-6459-59e9-97a8-dc69b9958eef)
GPU 2: NVIDIA A100-SXM4-40GB (UUID: GPU-4e56019e-84de-eef5-5ac3-85e468e93639)
  MIG 3g.20gb     Device  0: (UUID: MIG-c130392b-fd7c-59f8-9f4b-ecab27a2984b)
  MIG 2g.10gb     Device  1: (UUID: MIG-7d61cf16-ad08-57be-b2c2-ac6e515b28c3)
  MIG 1g.5gb      Device  2: (UUID: MIG-f5388dec-d841-506c-bb7a-6ac136ceee53)
  MIG 1g.5gb      Device  3: (UUID: MIG-02d54020-c6cb-5997-894b-e95af0f49388)
GPU 3: NVIDIA A100-SXM4-40GB (UUID: GPU-4fd894a0-b471-9e77-eb67-0ad15002ed5b)
  MIG 3g.20gb     Device  0: (UUID: MIG-10399b59-2625-5106-b3f8-76ae19da46e1)
  MIG 2g.10gb     Device  1: (UUID: MIG-ef81ee7d-fc48-56ed-8479-8ffa76eb4154)
  MIG 1g.5gb      Device  2: (UUID: MIG-ed9bf59d-6bf2-5e07-80fb-dd0d7ca41f9d)
  MIG 1g.5gb      Device  3: (UUID: MIG-a15d6f7d-661b-514e-8838-06fe4ffe7f75)
GPU 4: NVIDIA A100-SXM4-40GB (UUID: GPU-05b6b91b-da6e-3078-1f4f-a7bbf1ff7ed2)
GPU 5: NVIDIA A100-SXM4-40GB (UUID: GPU-078a8df1-f387-0315-6b0b-af12e082f6d5)
GPU 6: NVIDIA A100-SXM4-40GB (UUID: GPU-6cdeffe7-45f1-7e8e-bcc1-4634399ad877)
GPU 7: NVIDIA A100-SXM4-40GB (UUID: GPU-5f68814a-4e4a-5dec-79b4-8d70a61c7714)
```

Amazon EKS 上の NVIDIA GPU でタイムスライシングを使用する

タイムスライシングを使用すると、複数の Pod が単一の物理 NVIDIA GPU を共有できます。Kubernetes スケジューラは同じ GPU に複数の Pod を配置し、GPU の CUDA スケジューラは作業をタイムマルチプレックスします。タイムスライシングは、最もシンプルな GPU 共有戦略です。ソフトウェアのみを使用し、特別なハードウェアを必要とせず、AWS 上のすべての NVIDIA GPU インスタンスタイプで動作します。タイムスライシングでは、GPU を共有する Pod 間でメモリやコンピューティングの分離を行うことはありません。これは、リクエスト間でアイドル状態になる推論サービスや、複数のユーザーが GPU を共有する開発環境など、GPU 使用率が低いワークロードに最適です。ハードウェアレベルのメモリとコンピューティングの分離を必要とするワークロードでは、代わりにマルチインスタンス GPU (MIG) を使用します。

Amazon EKS では、NVIDIA DRA ドライバーまたは NVIDIA デバイスプラグインを使用してタイムスライシングを管理できます。

タイムスライシングは、静的キャパシティのプロビジョニングを使用している場合にのみ Karpenter で使用できます。タイムスライシングは現在、EKS Auto Mode では使用できません。

タイムスライシングは、次の場合に適しています:

- GPU 使用率が一貫して低い。例えば、リクエスト間でアイドル状態になるレイテンシーの影響を受けやすい推論サービスなど。
- 複数のデベロッパーがノートブックまたは開発作業用に単一の GPU ノードを共有している。
- ノードが、g5、g6、g6e ファミリーなど、MIG をサポートしていない GPU インスタンスタイプを使用している。
- 同じ GPU 上の別の Pod がビジー状態であるため、Pod が低速になる場合があることを許容できる。

次の場合は、別のアプローチを検討してください:

- メモリの分離が必要である。タイムスライスされた GPU 上のポッドは同じ GPU メモリを共有し、1 つのポッドは他のポッドが依存するメモリを使い果たすことができます。メモリの分離には MIG を使用します。
- 予測可能なポッドごとのレイテンシーまたはサービス品質が必要である。GPU スケジューラは、保証なしでベストエフォートベースでスロット間でコンピューティングを共有します。
- トレーニングワークロードを実行している。タイムスライシングは、トレーニング効率を低下させるコンテキスト切り替えを追加します。別の Pod によって遅くなるチェックポイントジョブは、最後のチェックポイントから再試行する必要があります。

考慮事項

本番環境でタイムスライシングを使用する前に、以下の考慮事項を確認してください。

一般的な考慮事項

- メモリ分離がない: タイムスライスされた GPU を共有する Pod はメモリを共有します。1 つの Pod が他の Pod に必要なメモリを割り当てることができるため、メモリ不足エラーが発生する可能性があります。各 GPU を共有する Pod の数をワークロードのメモリフットプリントに一致させます。Pod が定期的に大きなモデルをロードする場合は、GPU あたりの Pod 数を減らすか、MIG を使用します。
- ベストエフォートのコンピューティング共有: GPU スケジューラは、ベストエフォートベースで Pod 間でコンピューティングを共有し、各 Pod に対して比例するコンピューティングを保証するものではありません。

- タイムスライシングと MPS は同じ GPU を共有できない: タイムスライシングは GPU コンピューティングモードを DEFAULT に設定しますが、NVIDIA Multi-Process Service (MPS) には EXCLUSIVE_PROCESS が必要です。両方の戦略を同じクラスターで使用できますが、同じ物理 GPU で同時に使用することはできません。タイムスライシングは MIG インスタンスにも影響しません。コンテナ間で単一の MIG インスタンスを共有するには、代わりに MPS を使用します。
- コンテナごとのメトリクス: タイムスライシングがアクティブな場合、NVIDIA Data Center GPU Manager (DCGM) はメトリクスを個々のコンテナに帰属させることができません。GPU レベルのメトリクスは引き続き使用できますが、どの Pod が一定量の GPU リソースを消費したかを特定することはできません。

NVIDIA DRA ドライバーに関する考慮事項

- アルファ機能: DRA ドライバーを通じたタイムスライシングには、デフォルトで無効になっているアルファ機能である TimeSlicingSettings 機能ゲートが必要です。詳細については、「the section called “NVIDIA DRA ドライバーで GPU タイムスライシングを使用する”」を参照してください。
- ユーザーによる共有のみ: Pod は、名前空間スコープされている同じ ResourceClaim または ResourceClaimTemplate を参照することによってのみ GPU を共有するため、共有は名前空間を横断できません。システムによる共有は、将来の機能として提案されています。
- Bottlerocket の組み込みデバイスプラグインを無効にする: DRA ドライバーを同じノードの NVIDIA デバイスプラグインと一緒に実行することはできません。Bottlerocket で、組み込みデバイスプラグインを無効にします。これには Bottlerocket バージョン 1.63.0 以降が必要です。詳細については、「the section called “NVIDIA DRA ドライバーをインストールする”」を参照してください。
- コンピューティングサポート: NVIDIA DRA ドライバーは、Karpenter、EKS マネージド型ノードグループ、またはセルフマネージド型ノードの静的キャパシティのプロビジョニングでサポートされており、EKS Auto Mode ではサポートされていません。詳細については、Karpenter ウェブサイトの「Karpenter 静的 NodePool ドキュメント」を参照してください。

NVIDIA デバイスプラグインに関する考慮事項

- スロットを使用したベストエフォートのコンピューティング共有: デバイスプラグインは、GPU ごとに固定数のスロットをアドバタイズします。単一の Pod が複数のスロットをリクエストした場合、追加のコンピューティングは受け取りません。fail-requests-greater-than-one オプションを有効にして、複数のスロットをリクエストする Pod を拒否します。

- Karpenter を使用したプロビジョニング: Karpenter は、GPU でタイムスライシングスが有効になっている場合でも、各 `nvidia.com/gpu` リクエストを物理 GPU としてカウントします。詳細については、GitHub の「Karpenter の問題 #2140」を参照してください。
- EKS Auto Mode でのサポートがない: EKS Auto Mode は NVIDIA デバイスプラグインを管理し、その設定を公開しません。タイムスライシングにはデバイスプラグイン設定が必要であるため、EKS Auto Mode ノードでタイムスライシング設定を適用することはできません。
- タイムスライシング設定の変更: NVIDIA デバイスプラグインはタイムスライシング `ConfigMap` の変更をモニタリングしません。設定を更新したら、デバイスプラグインを再起動します。

設定オプション

EKS 最適化 AL2023 および Bottlerocket NVIDIA AMI では、NVIDIA GPU のタイムスライシングを使用できます。タイムスライシング設定は、NVIDIA DRA ドライバーと NVIDIA デバイスプラグインのどちらを使用しているかによって異なります。

NVIDIA DRA driver

EKS 最適化高速 AMI には NVIDIA DRA ドライバーは含まれていません。Bottlerocket を使用している場合は、ノードユーザーデータで `settings.kubelet-device-plugins.nvidia.enabled = false` を設定して、NVIDIA DRA ドライバーを使用する前に組み込み NVIDIA デバイスプラグインを無効にする必要があります。これには Bottlerocket バージョン 1.63.0 以降が必要です。詳細については、「the section called “NVIDIA DRA ドライバーをインストールする”」を参照してください。

NVIDIA DRA ドライバーでは、タイムスライシングは ResourceClaimTemplates を介して設定されます。`interval` フィールドは、CUDA タイムスライス期間を制御します。

Interval	説明
デフォルト	NVIDIA GPU ドライバーの組み込みデフォルト間隔を使用する
ショート	短い間隔、コンテキストの切り替え頻度が高い
中	中間の間隔
ロング	各コンテキストは、プリエンプトされる前に、ターンごとに実行時間が長くなる

NVIDIA device plugin

- Bottlerocket – AMI には、NVIDIA デバイスプラグインがプリインストールされています。Bottlerocket 設定を介してタイムスライシングを設定します。個別のデバイスプラグインのインストール、Helm チャート、または `ConfigMap` はありません。
- AL2023 – the section called “NVIDIA Kubernetes デバイスプラグインをインストールする” で説明されているように NVIDIA デバイスプラグインをインストールし、`ConfigMap` を介してタイムスライス設定を指定します。

  以下のオプションは、NVIDIA デバイスプラグインがタイムスライスされた GPU をアドバタイズおよび管理する方法を制御します。フィールド名は、以下の手順に示すように、Bottlerocket 設定と AL2023 `ConfigMap` で異なります。

オプション	推奨値	説明
レプリカ	2~8	物理 GPU ごとにアドバタイズするスケジュール可能なスロットの数。値を大きくすると、より多くの共有が可能になりますが、Pod 間の競合が増加します。
デフォルトで名前を変更	`false`	`false` の場合、Pod は既存のマニフェストとの互換性を維持する `nvidia.com/gpu` をリクエストします。`true` の場合、デバイスプラグインはリソースを `nvidia.com/gpu.shared` としてアドバタイズし、Pod はその名前をリクエストする必要があります。共有 GPU ノードプールと専用 GPU ノードプールの両方を実行し、ワークロードで共有リソースを明示的に選択する場合は、これを `true` に設定します。
複数のリクエストの失敗	`true`	複数のタイムスライスされたスロットをリクエストする Pod を拒否します。複数のスロットをリクエストする Pod は、比例コンピューティングを受け取りません。この設定を有効にすると、一般的な設定ミスを防ぐことができます。

GPU 共有オプションとそのデフォルト値の完全なリストについては、GitHub の「NVIDIA Kubernetes デバイスプラグインドキュメント」を参照してください。

NVIDIA DRA ドライバーで GPU タイムスライシングを使用する

NVIDIA DRA ドライバーを使用して、Pod はタイムスライシング `GpuConfig` を持つ `gpu.nvidia.com` `DeviceClass` をリクエストする共通の `ResourceClaim` を参照することで GPU を共有します。

NVIDIA DRA ドライバーは現在、ユーザーによるタイムスライシングを実装しています。GPU は、同じクレームを明示的に指定した Pod とコンテナ間でのみ共有されます。`ResourceClaim` と `ResourceClaimTemplate` は名前空間スコープされているため、この方法で GPU を共有する Pod は同じ名前空間にある必要があります。単一の Pod 内のコンテナ間で GPU を共有するには、各コンテナにクレーム内の同じリクエスト名を参照させます。異なるリクエスト名を参照するコンテナはそれぞれ個別の GPU を受け取ります。必要なタイムスライス間隔ごとに個別の `ResourceClaimTemplate` を作成し、GPU を共有する各名前空間にも 1 つ作成します。

システムによるタイムスライシングは、ドライバーが、設定したクレームではなく、システムが定義する基準に基づいて、独立したクレーム間 (名前空間間を含む) で GPU を共有する場合に行われます。システムによるタイムスライシングの詳細については、GitHub の「GPU のシステムによるタイムスライシング (問題 #659)」および「プルリクエスト #1257」を参照してください。

> **Important**
>
> DRA ドライバーを通じたタイムスライシングには、デフォルトで無効になっているアルファ機能である `TimeSlicingSettings` 機能ゲートが必要です。この機能ゲートを有効にせずに `TimeSlicing` 共有戦略をリクエストした場合、ドライバーはデバイスの準備に失敗し、Pod は `error validating GPU config: unknown GPU sharing strategy: TimeSlicing` をレポートする `FailedPrepareDynamicResources` イベントとともに `ContainerCreating` のままになります。アルファ機能の使用に伴うリスクを受け入れる場合にのみ、フィーチャーゲートを有効にしてください。

前提条件

- Amazon EKS クラスターで Kubernetes バージョン 1.34 以降が動作していること。NVIDIA DRA ドライバーは、Karpenter、EKS マネージド型ノードグループ、またはセルフマネージド型ノードでの静的キャパシティのプロビジョニングでサポートされています。
- ノードのインスタンスタイプが NVIDIA GPU であり、EKS 最適化 AL2023 NVIDIA AMI を使用していること。
- the section called “NVIDIA DRA ドライバーをインストールする” で説明されているように NVIDIA DRA ドライバーがインストールされ、`TimeSlicingSettings` 機能ゲートが有効になっています。

手順

1. `TimeSlicing` 共有戦略を使用して GPU をリクエストする `ResourceClaim` を作成します。このクレームを参照する複数の Pod は、同じ物理 GPU を共有します。

```
cat <<EOF | kubectl apply -f -
apiVersion: resource.k8s.io/v1
kind: ResourceClaim
metadata:
  name: shared-timeslice-gpu
spec:
  devices:
    requests:
    - name: gpu
      exactly:
        deviceClassName: gpu.nvidia.com
        count: 1
    config:
    - requests: ["gpu"]
      opaque:
        driver: gpu.nvidia.com
        parameters:
          apiVersion: resource.nvidia.com/v1beta1
          kind: GpuConfig
          sharing:
            strategy: TimeSlicing
            timeSlicingConfig:
              interval: Long
EOF
```

2. 名前で共有された ResourceClaim を参照する 2 つ以上の Pod をデプロイします。各 Pod は、resourceClaims および resources.claims を通じてクレームを参照します。

```
cat <<EOF | kubectl apply -f -
apiVersion: v1
kind: Pod
metadata:
  name: share-a
spec:
  tolerations:
    - key: nvidia.com/gpu
      operator: Exists
      effect: NoSchedule
  containers:
    - name: cuda
      image: nvidia/cuda:12.6.0-base-ubuntu22.04
      command: ["nvidia-smi", "-L"]
      resources:
        claims:
        - name: gpu
  resourceClaims:
    - name: gpu
      resourceClaimName: shared-timeslice-gpu
  restartPolicy: OnFailure
EOF
```

3. Pod が同じ物理 GPU を共有していることを確認します。各 Pod で nvidia-smi -L を実行し、同じ GPU UUID がレポートされていることを確認します。

```
kubectl logs share-a
```

出力例は次のとおりです。同じクレームを参照する 2 番目の Pod は、同一の GPU UUID をレポートし、両方の Pod が 1 つの物理 GPU を共有していることを確認します。

```
GPU 0: NVIDIA L4 (UUID: GPU-b41973ee-5d0a-cde8-6287-12b53f861f02)
```

NVIDIA デバイスプラグインで Bottlerocket ノードの GPU タイムスライシングを使用する

Bottlerocket では、EKS 最適化高速 AMI には NVIDIA デバイスプラグインが含まれています。ノードの起動時に Bottlerocket がデバイスプラグイン設定にレンダリングする `settings.kubelet-device-plugins.nvidia` 設定を通じてタイムスライシングを有効にします。

前提条件

- Amazon EKS クラスター。次の手順では、EKS 最適化 Bottlerocket NVIDIA AMI を使用して NVIDIA GPU ノードをプロビジョニングします。
- Karpenter がクラスターにインストールおよび設定されていること。それは、次の手順で、Karpenter `EC2NodeClass` を使用して Bottlerocket ノードのユーザーデータにタイムスライシング設定を指定するからです。詳細については、Karpenter ウェブサイトの「Karpenter の開始方法」を参照してください。
- クラスターと通信するように `kubectl` が設定されていること。詳細については、「the section called “kubectl をインストールまたは更新する”」を参照してください。

手順

GPU ノードの Bottlerocket ユーザーデータにタイムスライシング設定を追加します。以下の例では、GPU ごとに 4 つのスロットを設定します。ユーザーデータの提供方法は、ノードのプロビジョニング方法によって異なります。次の例は、Karpenter `EC2NodeClass` を示しています。

```
cat <<EOF | kubectl apply -f -
apiVersion: karpenter.k8s.aws/v1
kind: EC2NodeClass
metadata:
  name: gpu-bottlerocket-timeslicing
spec:
  amiFamily: Bottlerocket
  amiSelectorTerms:
    - alias: bottlerocket@latest
  role: eksctl-KarpenterNodeRole-<cluster-name>
  subnetSelectorTerms:
    - tags:
        karpenter.sh/discovery: <cluster-name>
  securityGroupSelectorTerms:
    - tags:
        karpenter.sh/discovery: <cluster-name>
  userData: |
```

```
    [settings.kubelet-device-plugins.nvidia]
    device-sharing-strategy = "time-slicing"

    [settings.kubelet-device-plugins.nvidia.time-slicing]
    replicas = 4
    rename-by-default = false
    fail-requests-greater-than-one = true
EOF
```

これらの設定でプロビジョニングされたノードがクラスターを結合すると、デバイスプラグインは物理 GPU ごとに 4 個の nvidia.com/gpu スロットをアドバタイズします。nvidia.com/gpu 拡張リソースのコンテナリソースリクエストまたは制限を使用して、ワークロード仕様でリクエストします。

NVIDIA デバイスプラグインで AL2023 ノードの GPU タイムスライシングを使用する

AL2023 では、the section called “NVIDIA Kubernetes デバイスプラグインをインストールする” で説明されているように NVIDIA デバイスプラグインをインストールし、ConfigMap でタイムスライス設定を指定します。

前提条件

- NVIDIA GPU インスタンスタイプを使用しているノードと EKS 最適化 AL2023 NVIDIA AMI がある Amazon EKS クラスター。
- コマンドライン環境に Helm がインストールされていること。詳細については、「Setup Helm instructions」を参照してください。
- クラスターと通信するように kubectl が設定されていること。詳細については、「the section called “kubectl をインストールまたは更新する”」を参照してください。

手順

1. the section called “NVIDIA Kubernetes デバイスプラグインをインストールする” にデバイスプラグインをインストールした nvidia 名前空間にタイムスライシング ConfigMap を作成します。この例では、GPU ごとに 4 個のスロットを設定します。

```
cat <<EOF | kubectl apply -f -
apiVersion: v1
kind: ConfigMap
metadata:
```

```
  name: nvidia-device-plugin-config
  namespace: nvidia
data:
  config.yaml: |
    version: v1
    sharing:
      timeSlicing:
        renameByDefault: false
        failRequestsGreaterThanOne: true
        resources:
          - name: nvidia.com/gpu
            replicas: 4
EOF
```

2. 既存の NVIDIA デバイスプラグインリリースを更新して、config.name 値で ConfigMap を参照します。--reuse-values フラグは、the section called “NVIDIA Kubernetes デバイスプラグインをインストールする” にデバイスプラグインをインストールしたときに設定した値を保持します。

```
helm upgrade nvdp nvdp/nvidia-device-plugin \
    --namespace nvidia \
    --reuse-values \
    --set config.name=nvidia-device-plugin-config
```

> **Note**
>
> ConfigMap を変更しても、デバイスプラグインは自動的に再ロードされません。タイムスライシング設定を更新したら、デバイスプラグイン Pod を再起動して変更を適用します。

GPU タイムスライシングがアクティブであることを確認する

タイムスライスされたノードが Ready になったら、デバイスプラグインが予想されるスロット数をアドバタイズし、Pod が物理 GPU を共有していることを確認します。

> **Note**
>
> この手順の共有 UUID チェックは、Pod が同じ物理 GPU に配置されたときに最も明確にタイムスライシングを示します。単一の GPU を持つノードでは、デバイスプラグインは 4 個のスロットをアドバタイズし、4 個の Pod すべてがその GPU を共有するため、それ

らは同じ UUID をレポートします。複数の物理 GPU を持つノードでは、スケジューラは異なる GPU に Pod を配置できます。これらの Pod は、タイムスライシングがアクティブであっても、異なる UUID をレポートします。1 個の GPU での共有をデモンストレーションするには、ワークロードを単一 GPU インスタンスタイプにスケジュールします。例えば、node.kubernetes.io/instance-type: g6.2xlarge などのノードセレクターを Pod 仕様に追加します。

1. 設定された数の GPU スロットをノードがアドバタイズしていることを確認します。GPU ごとに 4 個のスロットがある場合、1 個の物理 GPU を持つノードは 4 をレポートします。

```
kubectl get nodes "-o=custom-
columns=NAME:.metadata.name,GPU:.status.allocatable.nvidia\.com/gpu"
```

出力例は次のとおりです。

```
NAME                                           GPU
ip-192-168-11-225.us-west-2.compute.internal   4
```

2. 4 個のレプリカを実行するデプロイを作成し、それぞれが 1 個の GPU スロットをリクエストします。

```
cat <<EOF | kubectl apply -f -
apiVersion: apps/v1
kind: Deployment
metadata:
  name: timeslicing-demo
spec:
  replicas: 4
  selector:
    matchLabels:
      app: timeslicing-demo
  template:
    metadata:
      labels:
        app: timeslicing-demo
    spec:
      tolerations:
        - key: nvidia.com/gpu
          operator: Exists
          effect: NoSchedule
```

```
      containers:
        - name: cuda
          image: nvidia/cuda:12.6.0-base-ubuntu22.04
          command: ["bash", "-c", "nvidia-smi --query-gpu=uuid --format=csv,noheader;
 sleep infinity"]
          resources:
            limits:
              nvidia.com/gpu: 1
EOF
```

3. 4 個の Pod がすべて同じノードでスケジュールされていることを確認します。

```
kubectl get pods -l app=timeslicing-demo -o wide
```

4. 4 個の Pod がすべて同じ GPU UUID をレポートしていることを確認します。4 個の Pod すべてにまたがる単一の共有 UUID は、1 個の物理 GPU が時間多重化されていることを確認します。

```
kubectl logs -l app=timeslicing-demo --prefix
```

出力例は次のとおりです。

```
[pod/timeslicing-demo-xxxxxxxxxx-aaaaa/cuda] GPU-c0583cce-87c5-c736-db7f-6d3128c84d03
[pod/timeslicing-demo-xxxxxxxxxx-bbbbb/cuda] GPU-c0583cce-87c5-c736-db7f-6d3128c84d03
[pod/timeslicing-demo-xxxxxxxxxx-ccccc/cuda] GPU-c0583cce-87c5-c736-db7f-6d3128c84d03
[pod/timeslicing-demo-xxxxxxxxxx-ddddd/cuda] GPU-c0583cce-87c5-c736-db7f-6d3128c84d03
```

AI/ML ワークロード用に Amazon EKS クラスターを最適化するためのレシピ

> **Tip**
>
> 今後開催予定の Amazon EKS AI/ML ワークショップに登録してください。

このセクションでは、特に特殊なハードウェアに関係する AI/ML ワークロード向けに、Amazon EKS クラスターを最適化するためのわかりやすいレシピを提供します。前提条件、詳細な手順、デプロイに関する考慮事項など、マネージド型ノードグループにテイントを追加することで、ポッドが特定のノードでスケジュールされないようにするためのガイダンスを紹介します。

トピック

- レシピ: 特定のノードでポッドがスケジュールされないようにする

レシピ: 特定のノードでポッドがスケジュールされないようにする

概要

GPU などの専門的なプロセッサを搭載したノードは、標準的なマシンのノードよりも実行コストが高くなる場合があります。特別なハードウェアを必要としないワークロードからこれらのノードを保護するには、Kubernetes テイントを使用できます。テイントはノードをマークして、許容範囲が一致しないポッドを避け、互換性のあるワークロードのみがスケジュールされるようにします。詳細については、Kubernetes ドキュメントの「テイントと容認」を参照してください。

AWS マネジメントコンソールまたは Amazon EKS API を使用して、新規または既存のマネージド型ノードグループに Kubernetes ノードのテイントが適用されます。このレシピは、AWS CLI を使用して Amazon EKS マネージドノードグループにテイントを適用する方法を示しています。を使用してテイントが適用されたノードグループを作成するを使用してテイントが適用されたノードグループを作成するを使用してテイントが適用されたノードグループを作成する情報はAWS マネジメントコンソールthe section called “作成”、、、、、、、、

前提条件

- 既存の Amazon EKS クラスター。
- 適切なアクセス許可でインストールされ設定された AWS CLI。

Steps

ステップ 1: テイントを含むノードグループを作成する

`aws eks create-nodegroup` コマンドを使用して、テイントを含む新しいマネージドノードグループを作成します。この例では、キー `dedicated`、値 `gpuGroup`、および効果 `NO_SCHEDULE` を持つテイントを適用します。

```
aws eks create-nodegroup \
 --cli-input-json '
{
  "clusterName": "my-cluster",
  "nodegroupName": "node-taints-example",
  "subnets": [
```

```
      "subnet-1234567890abcdef0",
      "subnet-abcdef01234567890",
      "subnet-021345abcdef67890"
    ],
  "nodeRole": "arn:aws:iam::111122223333:role/AmazonEKSNodeRole",
  "taints": [
     {
         "key": "dedicated",
         "value": "gpuGroup",
         "effect": "NO_SCHEDULE"
     }
    ]
}'
```

詳細な説明と例については、Kubernetes リファレンスドキュメントの「テイント」を参照してください。

ステップ 2: 既存のノードグループのテイントを更新する

aws eks update-nodegroup-config AWS CLI コマンドを使用して、マネージドノードグループのテイントを追加、削除、または交換します。

```
aws eks update-nodegroup-config \
  --cluster-name my-cluster \
  --nodegroup-name node-taints-example \
  --taints 'removeTaints=[{key=dedicated,value=gpuGroup,effect=NO_SCHEDULE}]'
```

注意事項

- テイントは、UpdateNodegroupConfig API を使用してノードグループを作成した後に更新できます。
- テイントのキーは、文字または数字で始まる必要があります。英字、数字、ハイフン (-)、ピリオド (.)、およびアンダースコア (_) を使用できます。最大 63 文字です
- オプションで、Taint キーは DNS サブドメインプレフィックスと単一の / で始めることができます。このキーを DNS サブドメインプレフィックスで始める場合は、最大 253 文字まで使用できます。
- 値は省略可能で、文字または数字で始める必要があります。英字、数字、ハイフン (-)、ピリオド (.)、およびアンダースコア (_) を使用できます。最大 63 文字です
- Kubernetes を直接使用するか、AWS マネジメントコンソールを使用する場合、テイント効果は NoSchedule、PreferNoSchedule、または NoExecute でなければなりません。ただし、AWS

CLI または API を使用する場合、テイント効果は NO_SCHEDULE、PREFER_NO_SCHEDULE、または NO_EXECUTE でなければなりません。

- 1 つのノードグループには、最大 50 個のテイントを使用できます。
- マネージドノードグループを使用して作成されたテイントをノードから手動で削除した場合、Amazon EKS はテイントをノードに戻しません。これは、管理対象ノードグループ設定でテイントが指定されている場合にも当てはまります。

Amazon EKS で AI/ML の使用を開始するためのリソース

EKS での機械学習の使用を開始するには、まずこれらの規範的なパターンから選択して、EKS クラスターと ML ソフトウェアおよびハードウェアが ML ワークロードの実行を開始できるように準備します。

ワークショップ

Amazon EKS での生成 AI ワークショップ

Amazon EKS で大規模言語モデル (LLM) アプリケーションと推論を開始する方法について説明します。本稼働グレードの LLM ワークロードをデプロイおよび管理する方法を説明します。実践的なラボを通じて、Amazon EKS を AWS サービスやオープンソースツールとともに活用して、堅牢な LLM ソリューションを作成する方法について説明します。ワークショップ環境には、必要なインフラストラクチャとツールがすべて用意されているため、学習と実装に集中できます。

Neuron を使用した Amazon EKS での生成 AI

Amazon EKS で大規模言語モデル (LLM) アプリケーションと推論を開始する方法について説明します。本稼働グレードの LLM ワークロードのデプロイと管理、ベクトルデータベースを使用した高度な RAG パターンの実装、オープンソースフレームワークを使用したデータ駆動型 LLM アプリケーションの構築方法について説明します。実践的なラボを通じて、Amazon EKS を AWS サービスやオープンソースツールとともに活用して、堅牢な LLM ソリューションを作成する方法について説明します。ワークショップ環境には、必要なインフラストラクチャとツールがすべて用意されているため、学習と実装に集中できます。

ベストプラクティス

Amazon EKS ベストプラクティスガイドの AI/ML に焦点を当てたトピックでは、Amazon EKS の AI/ML ワークロードを最適化するために、以下の領域にわたって詳細な推奨事項を提供します。

AI/ML コンピューティングと自動スケーリング

このセクションでは、GPU リソース管理、ノードの耐障害性、アプリケーションのスケーリングに焦点を当て、Amazon EKS で AI/ML コンピューティングと自動スケーリングを最適化するためのベストプラクティスの概要を説明します。よく知られているラベルとノードアフィニティを使用してワークロードのスケジュール設定、ML キャパシティブロックまたはオンデマンドキャパシティ予約の使用、および EKS Node Monitoring Agent などのツールを使用したノードヘルスチェックの実装などの戦略を提供します。

AI/ML ネットワーク

このセクションでは、Amazon EKS で AI/ML ネットワークを最適化してパフォーマンスとスケーラビリティを向上させるためのベストプラクティスについて説明します。これには、分散トレーニング用のネットワーク帯域幅が高いインスタンスや Elastic Fabric Adapter (EFA) の選択、MPI や NCCL などのツールのインストール、IP アドレスを増やしてポッドの起動時間を短縮するためのプレフィックス委任の有効化などの戦略が含まれます。

AI/ML セキュリティ

このセクションでは、データストレージの保護と Amazon EKS での AI/ML ワークロードのコンプライアンスの確保に焦点を当てます。これには、Amazon S3 と AWS Key Management Service (KMS) を使用したサーバー側の暗号化 (SSE-KMS)、リージョン KMS キーと S3 バケットキーを使用したバケットの設定によるコスト削減、EKS ポッドへの復号化などの KMS アクションに対する IAM アクセス許可の付与、AWS CloudTrail ログによる監査などのプラクティスが含まれます。

AI/ML ストレージ

このセクションでは、Amazon EKS の AI/ML ワークロードのストレージを最適化するためのベストプラクティスについて説明します。これには、CSI ドライバーを使用して S3、FSx for Lustre、EFS などのサービスを永続ボリュームとしてマウントするモデルをデプロイする方法、ワークロードのニーズに基づいてストレージを選択する方法 (Scratch-SSD や Persistent-SSD などのオプションを使用した分散トレーニング用の FSx for Lustre など)、データ圧縮やストライピングなどの機能を有効にする方法が含まれます。

AI/ML のオブザーバビリティ

このセクションでは、Amazon EKS の AI/ML ワークロードの GPU 使用率をモニタリングおよび最適化して効率を向上させ、コストを削減することに重点を置いています。これには、CloudWatch Container Insights や Prometheus および Grafana と統合された NVIDIA の DCGM-Exporter などの

ツールを使用して高い GPU 使用率をターゲットにする戦略や、AI/ML ワークロードの分析におすすめのメトリクスが含まれます。

AI/ML パフォーマンス

このセクションでは、コンテナイメージ管理とスタートアップ最適化を通じて、Amazon EKS の AI/ML ワークロードのアプリケーションのスケーリングとパフォーマンスを向上させることに重点を置いています。これには、マルチステージビルドで小さな軽量ベースイメージまたは AWS 深層学習コンテナを使用する、EBS スナップショットを介してイメージをプリロードする、DaemonSets または Deployments を使用してランタイムキャッシュにプルするなどのプラクティスが含まれます。

リファレンスアーキテクチャ

リファレンスアーキテクチャ、サンプルコード、ユーティリティに関するこれらの GitHub リポジトリを調べて、Amazon EKS やその他の AWS サービスで AI/ML ワークロードの分散トレーニングと推論を実装します。

Awsome Distributed Training

このリポジトリには、AWS で大規模モデルをトレーニングするためのベストプラクティス、リファレンスアーキテクチャ、モデルトレーニングの例、ユーティリティのコレクションが用意されています。EKS クラスター用の CloudFormation テンプレート、カスタム AMI とコンテナビルド、PyTorch (DDP/FSDP、MegatronLM、NeMo) や JAX などのフレームワークのテストケース、EFA Prometheus エクスポーターや Nvidia Nsight Systems などの検証、オブザーバビリティ、パフォーマンスのモニタリングツールなど、Amazon EKS での分散トレーニングをサポートします。

AWSome 推論

このリポジトリは、Amazon EKS と高速 EC2 インスタンスに焦点を当て、AWS で推論ソリューションを最適化するためのリファレンスアーキテクチャとテストケースを提供します。これには、VPC および EKS クラスターのインフラストラクチャ設定、NVIDIA NIM、TensorRT-LLM、Triton Inference Server、RayService などのフレームワークのプロジェクト、Llama3-8B や Llama 3.1 405B などのモデルの例が含まれています。K8s LeaderWorkerSet、EKS オートスケーリング、マルチインスタンス GPU (MIG)、ASR、推論、TTS 用のオーディオボットなどの実際のユースケースを使用したマルチノードデプロイを提供します。

チュートリアル

EKS での機械学習プラットフォームとフレームワークのセットアップに関心がある場合はこのセクションに記載されているチュートリアルを詳しくご覧ください。これらのチュートリアルではGPUプロセッサを最大限に活用するためのパターンから、モデリングツールの選択、専門的な業界向けのフレームワークの構築まで、あらゆる事項について説明します。

EKS で生成 AI プラットフォームを構築する

- Deploy Generative AI Models on Amazon EKS
- Building multi-tenant JupyterHub Platforms on Amazon EKS

EKS で専門的な生成 AI フレームワークを実行する

- Accelerate your generative AI distributed training workloads with the NVIDIA NeMo Framework on Amazon EKS
- Running TorchServe on Amazon エラスティックKubernetesサービス

ML on EKS の NVIDIA GPU パフォーマンスを最大化する

- EKS クラスターで NVIDIA GPU を効率的に使用するために、GPU 共有を実装します:

  GPU sharing on Amazon EKS with NVIDIA time-slicing and accelerated EC2 instances
- EKS クラスターで GPU ごとにより多くのポッドを実行するために、マルチインスタンス GPU (MIG) と NIM マイクロサービスを使用します:

  Maximizing GPU utilization with NVIDIA's Multi-Instance GPU (MIG) on Amazon EKS: Running more pods per GPU for enhanced performance
- Build and deploy a scalable machine learning system on Kubernetes with Kubeflow on AWS

EKS で動画エンコーディングワークロードを実行する

- Delivering video content with fractional GPUs in containers on Amazon EKS

推論ワークロードのイメージロードを高速化する

- How H2O.ai optimized and secured their AI/ML infrastructure with Karpenter and Bottlerocket

ML ワークロードのモニタリング

- Monitoring GPU workloads on Amazon EKS using AWS managed open-source services
- Enable pod-based GPU metrics in Amazon CloudWatch

Amazon EKS ツール

Amazon EKS には、EKS クラスターをより効率的に管理、トラブルシューティング、操作するのに役立つ一連の AWS ツールが用意されています。これらのツールは、AI を活用した機能と標準化されたインターフェイスを活用して、クラスターオペレーションを簡素化し、トラブルシューティングを加速し、Kubernetes リソースとの自然言語によるインタラクションを可能にします。使用可能なツールは次のとおりです。

- Amazon EKS モデルコンテキストプロトコル (MCP) サーバー - EKS クラスター管理のために AI を活用したエクスペリエンスを可能にするフルマネージドサービス。EKS MCP サーバーは、MCP 互換の AI コーディングアシスタントと統合され、クラスターと Kubernetes リソースに関するコンテキストに応じた知識をリアルタイムで提供します。これを使用して、自然言語によるインタラクションを通じて、クラスターの管理、アプリケーションのデプロイ、問題のトラブルシューティング、EKS ドキュメントのクエリを行います。
- Amazon EKS コンソールの Amazon Q - AWS マネジメントコンソールで直接利用できる、AI を活用したトラブルシューティング統合。Amazon Q はクラスターの問題を自動的に分析し、EKS コンソールでエラーや警告が発生した場合に、調査結果、根本原因分析、推奨される緩和手順を提供します。

これらのツールは連携して、初期設定や設定から、継続的な運用やトラブルシューティングまで、EKS クラスターのライフサイクル全体で包括的なサポートを提供します。アプリケーションの開発、インフラストラクチャの管理、本番稼働の問題の解決など、これらの AWS ツールは EKS クラスターをより効率的に操作するのに役立ちます。

トピック

- Amazon EKS モデルコンテキストプロトコル (MCP) サーバー
- Amazon EKS コンソールで Amazon Q Developer を使用する

Amazon EKS モデルコンテキストプロトコル (MCP) サーバー

Amazon EKS MCP サーバーは、AI を活用した開発および運用のエクスペリエンスを実現するフルマネージドサービスです。モデルコンテキストプロトコル (MCP) は、EKS クラスターと Kubernetes リソースに関するリアルタイムでコンテキストに応じた知識で AI エージェントとアプリケーションを強化する標準化されたインターフェイスを提供します。これにより、初期設定から本番稼働の最適

化、トラブルシューティングまで、アプリケーションのライフサイクル全体にわたって、より正確でコンテキストに応じたレスポンスと AI を活用したワークフローが可能になります。

> **Note**
>
> Amazon EKS MCP サーバー は Amazon EKS のプレビュー版であり、変更される可能性があります。

概要

Amazon EKS MCP サーバーは、Kiro、Amazon Q Developer CLI、または Cursor、Cline などのサードパーティーツールといった、MCP 互換の AI コーディングアシスタントと簡単に統合でき、開発ワークフローを強化できます。使用を開始すると、EKS MCP サーバーがクラスターの作成をガイドし、前提条件を自動的にプロビジョニングして、AWS のベストプラクティスを適用します。開発中は、アプリケーションのデプロイとクラスター管理のための高レベルのワークフローを提供することで、EKS および Kubernetes のオペレーションを簡素化します。デバッグとトラブルシューティングでは、EKS コンソールや好みの AI アシスタントで利用可能な、統合されたトラブルシューティング支援およびナレッジベースへのアクセスを通じて、問題解決を加速します。これらの機能は自然言語インタラクションを通じてアクセスできるため、複雑な Kubernetes オペレーションをより直感的かつ効率的に実行できます。

フルマネージド EKS MCP サーバーは AWS クラウドでホストされているため、ローカルでのインストールやメンテナンスは必要ありません。自動更新とパッチ適用、AWS IAM 統合による一元化されたセキュリティ、AWS CloudTrail による包括的な監査ログ記録、AWS の実績あるスケーラビリティ、信頼性、サポートなど、エンタープライズグレードの機能を備えています。AWS クラウドでホストされているフルマネージド EKS MCP サーバーには、次の主な利点があります。

- インストールとメンテナンスが不要。EKS MCP サーバーは AWS クラウドでホストされているため、バージョン更新の管理やローカルサーバーの問題のトラブルシューティングを行う必要がありません。ホストされた EKS MCP サーバーのエンドポイントに接続するように AI アシスタントを設定するだけで、EKS クラスターの使用を開始できます。
- アクセスの一元管理。EKS MCP サーバーは IAM と統合されており、サーバーへのアクセスを制御するための一元化された安全な方法を提供します。すべてのリクエストは、軽量なプロキシを介して AWS SigV4 で署名され、既存の AWS 認証情報や IAM ポリシーとシームレスに統合できます。

- モニタリングと可視性の強化。 AWSCloudTrail との統合により、ホストされたサービスを通じて行われる初期化や完全なアクセスを伴うツール呼び出しを記録し、詳細な監査証跡とコンプライアンスレポートを実現にします。
- 常に最新の状態。ローカルインストールを更新することなく、新しいツール、機能、バグ修正を自動的に受け取れます。ホストされたサービスは、ユーザーからのフィードバックと AWS のベストプラクティスに基づいて継続的に改善されます。

統合の例

EKS MCP サーバーには、以下に使用できるツールがいくつか用意されています。

- クラスターの管理 自動化されたベストプラクティスを使用して、EKS クラスターを作成、設定、管理します。
- Kubernetes リソースの管理 アプリケーションをデプロイし、Kuberentes リソースを管理し、クラスターの状態を検査します。
- クラスターのトラブルシューティング 統合されたトラブルシューティングツールとランブックのナレッジベースを使用して、問題を診断します。
- ドキュメントのクエリ 関連する EKS ドキュメントをコンテキスト別に検索して取得します。

クラスターの調査

```
Show me all EKS clusters and their status
What insights does EKS have about my production-cluster?
Show me the VPC configuration for my staging cluster
```

Kubernetes リソースの確認

```
List all deployments in the production namespace
Show me pods that are not in Running state
Get the logs from the api-server pod in the last 30 minutes
```

問題のトラブルシューティング

```
Why is my nginx-ingress-controller pod failing to start?
Search the EKS troubleshooting guide for pod networking issues
Show me events related to the failed deployment in the staging namespace
```

リソースの作成 (「書き込み」モードが有効になっている場合)

```
Create a new EKS cluster named demo-cluster with VPC and Auto Mode
Deploy my containerized app from ECR to the production namespace with 3 replicas
Generate a Kubernetes deployment YAML for my Node.js app running on port 3000
```

はじめに

開始するには、the section called “はじめに” を参照してください。

Amazon EKS MCP サーバーの使用開始

このガイドでは、AI コードアシスタントで EKS MCP サーバーをセットアップして使用する手順について説明します。環境を設定し、サーバーに接続し、自然言語インタラクションを通じて EKS クラスターの管理を開始する方法が理解できます。

> **Note**
>
> Amazon EKS MCP サーバー は Amazon EKS のプレビュー版であり、変更される可能性があります。

前提条件

始める前に、次のタスクを実行していることを確認してください。

- Amazon EKS にアクセスできる AWS アカウントが作成されていること
- AWS CLI がインストールされ、認証情報を使用して設定されていること
- Python 3.10 以降がインストールされていること
- uv をインストールする

セットアップ

1. 前提条件を確認する

```
# Check that your Python version is 3.10 or higher
python3 --version
```

```
# Check uv installation
uv --version

# Verify CLI configuration
aws configure list
```

2. IAM 許可をセットアップする

EKS MCP サーバーに接続するには、IAM ロールに次のポリシーがアタッチされている必要があります。 **eks-mcp:InvokeMcp** (初期化および使用可能なツールに関する情報の取得に必要なアクセス許可)、 **eks-mcp:CallReadOnlyTool** (読み取り専用ツールの使用に必要なアクセス許可)、および **eks-mcp:CallPrivilegedTool** (フルアクセス (書き込み) ツールの使用に必要なアクセス許可) です。これらの eks-mcp アクセス許可は、以下に示す読み取り専用およびフルアクセスの AWS 管理ポリシーに含まれています。

- [IAM コンソール] を開きます。
- 左側のナビゲーションペインで、ポリシーをアタッチするアイデンティティに応じて [ユーザー]、[ユーザーグループ]、または [ロール] を選択し、特定のユーザー、グループ、またはロールの名前を選択します。
- [アクセス許可] タブを選択します。
- [ポリシーをアタッチ] (または、初めての場合は [アクセス許可の追加]) を選択します。
- ポリシーリストで、アタッチする管理ポリシーを検索して選択します。
- [読み取り専用オペレーション]: AmazonEKSMCPReadOnlyAccess
- [ポリシーをアタッチ] を選択します (または [次へ] を選択してから、[アクセス許可の追加] を選択して確認します)。

これによりポリシーがアタッチされ、アクセス許可はすぐに有効になります。同じアイデンティティに複数のポリシーをアタッチし、各ポリシーにさまざまなアクセス許可を含めることができます。これらのポリシーの詳細については、「Amazon Elastic Kubernetes Service に関する AWS 管理ポリシー」を参照してください。

3. AI アシスタントを選択する

次の MCP 互換 AI アシスタントのいずれか、または MCP 互換ツールを選択します。

- Amazon Q Developer CLI のインストール

- Kiro のインストール
- Cursor のインストール
- VS Code Vale 拡張機能のインストール

ステップ 1: AI アシスタントを設定する

AI コードアシスタントをセットアップするには、次のいずれかのオプションを選択します。この手順を完了すると、AWS の MCP Proxy を使用するように AI コードアシスタントがセットアップされます。これは、Amazon EKS MCP サーバーへの安全で認証されたアクセスのために必要です。これには、MCP 設定ファイル (例えば、Amazon Q Developer CLI の場合は `~/.aws/amazonq/mcp.json`) の追加または編集が含まれます。このプロキシはクライアント側のブリッジとして機能し、ローカルの AWS 認証情報を使用して AWS SigV4 認証を処理し、EKS MCP サーバーなどのバックエンド AWS MCP サーバーとやり取りするための動的なツール検出を有効にします。詳細については、「MCP Proxy for AWS」リポジトリを参照してください。

オプション A: Amazon Q Developer CLI

Q Developer CLI は、EKS MCP サーバーと最も統合されたエクスペリエンスを提供します。

1. MCP 設定ファイルを見つける

- macOS/Linux: `~/.aws/q/mcp.json`
- Windows: `%USERPROFILE%\.aws\q\mcp.json`

2. MCP サーバー設定を追加する

設定ファイルが存在しない場合は、作成します。リージョン (`{region}`) プレースホルダーは、ご使用のリージョンに置き換えてください。

Mac/Linux の場合:

```
{
  "mcpServers": {
    "eks-mcp": {
      "disabled": false,
      "type": "stdio",
      "command": "uvx",
      "args": [
```

```
        "mcp-proxy-for-aws@latest",
        "https://eks-mcp.{region}.api.aws/mcp",
        "--service",
        "eks-mcp",
        "--profile",
        "default",
        "--region",
        "{region}"
      ]
    }
  }
}
```

Windows の場合 –

```
{
  "mcpServers": {
    "eks-mcp": {
      "disabled": false,
      "type": "stdio",
      "command": "uvx",
      "args": [
        "--from",
        "mcp-proxy-for-aws@latest",
        "mcp-proxy-for-aws.exe",
        "https://eks-mcp.{region}.api.aws/mcp",
        "--service",
        "eks-mcp",
        "--profile",
        "default",
        "--region",
        "{region}"
      ]
    }
  }
}
```

セキュリティ上の注意: `--read-only` は、読み取り専用のツールオペレーションのみを許可するために使用できます。

3. 設定を確認する

Q Developer CLI を再起動し、使用可能なツールを確認します。

```
q /tools
```

オプション B: Kiro IDE

Kiro は、組み込み MCP サポートを備えた AI ファーストのコーディングワークスペースです。

1. Kiro 設定を開く

- Kiro を開く
- [Kiro] → [設定] に移動し、「MCP Config」を検索する
- または、`Cmd+Shift+P` (Mac) あるいは `Ctrl+Shift+P` (Windows/Linux) を押して、「MCP Config」を検索する

2. MCP サーバー設定を追加する

- [ワークスペース MCP Config を開く] または [ユーザー MCP Config を開く] をクリックして、MCP 設定ファイルを直接編集します。

リージョン (`{region}`) プレースホルダーは、ご使用のリージョンに置き換えてください。

Mac/Linux の場合:

```
{
  "mcpServers": {
    "eks-mcp": {
      "disabled": false,
      "type": "stdio",
      "command": "uvx",
      "args": [
        "mcp-proxy-for-aws@latest",
        "https://eks-mcp.{region}.api.aws/mcp",
        "--service",
        "eks-mcp",
        "--profile",
        "default",
        "--region",
        "{region}"
      ]
    }
  }
```

```
}
```

Windows の場合 –

```
{
  "mcpServers": {
    "eks-mcp": {
      "disabled": false,
      "type": "stdio",
      "command": "uvx",
      "args": [
        "--from",
        "mcp-proxy-for-aws@latest",
        "mcp-proxy-for-aws.exe",
        "https://eks-mcp.{region}.api.aws/mcp",
        "--service",
        "eks-mcp",
        "--profile",
        "default",
        "--region",
        "{region}"
      ]
    }
  }
}
```

セキュリティ上の注意: `--read-only` は、読み取り専用のツールオペレーションのみを許可するために使用できます。

オプション C: Cursor IDE

Cursor IDE は、グラフィカル設定インターフェイスを備えた組み込み MCP サポートを提供します。

1. Cursor 設定を開く

- Cursor を開く
- [設定] → [Cursor 設定] → [ツールと MCP] に移動する
- または `Cmd+Shift+P` (Mac) / `Ctrl+Shift+P` (Windows) を押して、「MCP」を検索する

2. MCP サーバー設定を追加する

- [新しい MCP サーバー] をクリックします。

設定ファイルが存在しない場合は、作成します。リージョン ({region}) プレースホルダーは、ご使用のリージョンに置き換えてください。

Mac/Linux の場合:

```
{
  "mcpServers": {
    "eks-mcp": {
      "disabled": false,
      "type": "stdio",
      "command": "uvx",
      "args": [
        "mcp-proxy-for-aws@latest",
        "https://eks-mcp.{region}.api.aws/mcp",
        "--service",
        "eks-mcp",
        "--profile",
        "default",
        "--region",
        "{region}"
      ]
    }
  }
}
```

Windows の場合 –

```
{
  "mcpServers": {
    "eks-mcp": {
      "disabled": false,
      "type": "stdio",
      "command": "uvx",
      "args": [
        "--from",
        "mcp-proxy-for-aws@latest",
        "mcp-proxy-for-aws.exe",
        "https://eks-mcp.{region}.api.aws/mcp",
        "--service",
        "eks-mcp",
        "--profile",
        "default",
        "--region",
```

```
        "{region}"
      ]
    }
  }
}
```

セキュリティ上の注意: `--read-only` は、読み取り専用のツールオペレーションのみを許可するために使用できます。

3. Cursor を再起動する

Cursor を閉じて再度開き、変更を有効にします。

4. Cursor チャットで検証する

チャットパネルを開き、以下を試します。

```
What EKS MCP tools are available?
```

使用可能な EKS 管理ツールのリストが表示されます。

オプション D: Cline (VS Code 拡張機能)

Cline は、AI アシスタンスを工ディタに直接提供する、一般的な VS Code 拡張機能です。

1. Cline 設定を開く

- Cline を開く
- `Cmd+Shift+P` (Mac) / `Ctrl+Shift+P` (Windows) を押して、「MCP」を検索する

2. MCP サーバー設定を追加する

- [サーバーの追加] をクリックします。
- [ユーザー設定を開く] をクリックします。

設定ファイルが存在しない場合は、作成します。リージョン (`{region}`) プレースホルダーは、ご使用のリージョンに置き換えてください。

Mac/Linux の場合:

```
{
  "mcpServers": {
    "eks-mcp": {
      "disabled": false,
      "type": "stdio",
      "command": "uvx",
      "args": [
        "mcp-proxy-for-aws@latest",
        "https://eks-mcp.{region}.api.aws/mcp",
        "--service",
        "eks-mcp",
        "--profile",
        "default",
        "--region",
        "{region}"
      ]
    }
  }
}
```

Windows の場合 –

```
{
  "mcpServers": {
    "eks-mcp": {
      "disabled": false,
      "type": "stdio",
      "command": "uvx",
      "args": [
        "--from",
        "mcp-proxy-for-aws@latest",
        "mcp-proxy-for-aws.exe",
        "https://eks-mcp.{region}.api.aws/mcp",
        "--service",
        "eks-mcp",
        "--profile",
        "default",
        "--region",
        "{region}"
      ]
    }
  }
```

```
}
```

セキュリティ上の注意: `--read-only` は、読み取り専用のツールオペレーションのみを許可するために使用できます。

3. VS コードを再ロードする

`Cmd+Shift+P` / `Ctrl+Shift+P` を押し、[デベロッパー: ウィンドウの再ロード] を選択します。

4. 設定を確認する

Cline を開き、次の質問をします。

```
List the available MCP tools for EKS
```

ステップ 2: (オプション) 「書き込み」ポリシーを作成する

必要に応じて、Amazon EKS MCP サーバーへのフルアクセスを提供するカスタマー管理の IAM ポリシーを作成できます。このポリシーは、書き込みオペレーションを伴う特権ツールと読み取り専用ツールの両方を含む、EKS MCP サーバー内のすべてのツールを使用するためのアクセス許可を付与します。manage_eks_stacks ツールにおけるクラスターリソースのセットアップ/ティアダウンに必要であるため、高リスクのアクセス許可 (Delete* を含むもの、または無制限の IAM リソース) がこのポリシーに含まれていることに注意してください。

```
aws iam create-policy \
 --policy-name EKSMcpWriteManagementPolicy \
 --policy-document "{\"Version\": \"2012-10-17\", \"Statement\": [{\"Effect
\": \"Allow\", \"Action\": [\"eks:DescribeCluster\", \"eks:ListClusters\",
 \"eks:DescribeNodegroup\", \"eks:ListNodegroups\", \"eks:DescribeAddon\",
 \"eks:ListAddons\", \"eks:DescribeAccessEntry\", \"eks:ListAccessEntries\",
 \"eks:DescribeInsight\", \"eks:ListInsights\", \"eks:AccessKubernetesApi\"],
 \"Resource\": \"*\"}, {\"Effect\": \"Allow\", \"Action\": [\"eks:CreateCluster
\", \"eks:DeleteCluster\", \"eks:CreateAccessEntry\", \"eks:TagResource\"],
 \"Resource\": \"*\"}, {\"Effect\": \"Allow\", \"Action\": [\"iam:GetRole\",
 \"iam:ListRolePolicies\", \"iam:ListAttachedRolePolicies\", \"iam:GetRolePolicy\",
 \"iam:GetPolicy\", \"iam:GetPolicyVersion\"], \"Resource\": \"*\"}, {\"Effect\":
 \"Allow\", \"Action\": [\"iam:TagRole\", \"iam:CreateRole\", \"iam:AttachRolePolicy
\", \"iam:PutRolePolicy\", \"iam:DetachRolePolicy\", \"iam:DeleteRole\"],
 \"Resource\": \"*\"}, {\"Effect\": \"Allow\", \"Action\": [\"iam:PassRole\"],
 \"Resource\": \"*\", \"Condition\": {\"StringEquals\": {\"iam:PassedToService
\": [\"eks.amazonaws.com\", \"ec2.amazonaws.com\"]}}}, {\"Effect\": \"Allow\",
```

```
 \"Action\": [\"ec2:CreateVpc\", \"ec2:CreateSubnet\", \"ec2:CreateRouteTable\",
 \"ec2:CreateRoute\", \"ec2:CreateInternetGateway\", \"ec2:CreateNatGateway\",
 \"ec2:CreateSecurityGroup\", \"ec2:AttachInternetGateway\", \"ec2:AssociateRouteTable
\", \"ec2:ModifyVpcAttribute\", \"ec2:ModifySubnetAttribute\", \"ec2:AllocateAddress
\", \"ec2:CreateTags\"], \"Resource\": \"*\"}, {\"Effect\": \"Allow\", \"Action
\": [\"ec2:DeleteVpc\", \"ec2:DeleteSubnet\", \"ec2:DisassociateRouteTable\",
 \"ec2:DeleteRouteTable\", \"ec2:DeleteRoute\", \"ec2:DetachInternetGateway\",
 \"ec2:DeleteInternetGateway\", \"ec2:DeleteNatGateway\", \"ec2:ReleaseAddress\",
 \"ec2:DeleteSecurityGroup\"], \"Resource\": \"*\"}, {\"Effect\": \"Allow\", \"Action
\": [\"ec2:DescribeVpcs\", \"ec2:DescribeSubnets\", \"ec2:DescribeRouteTables\",
 \"ec2:DescribeInternetGateways\", \"ec2:DescribeNatGateways\", \"ec2:DescribeAddresses
\", \"ec2:DescribeSecurityGroups\", \"ec2:DescribeAvailabilityZones\"], \"Resource
\": \"*\"}, {\"Effect\": \"Allow\", \"Action\": [\"cloudformation:CreateStack
\", \"cloudformation:UpdateStack\", \"cloudformation:DeleteStack\",
 \"cloudformation:DescribeStacks\", \"cloudformation:TagResource\"], \"Resource
\": \"*\"}, {\"Effect\": \"Allow\", \"Action\": [\"sts:GetCallerIdentity\"],
 \"Resource\": \"*\"}, {\"Effect\": \"Allow\", \"Action\": [\"logs:StartQuery\",
 \"logs:GetQueryResults\"], \"Resource\": \"*\"}, {\"Effect\": \"Allow\", \"Action\":
 [\"cloudwatch:GetMetricData\"], \"Resource\": \"*\"}, {\"Effect\": \"Allow\", \"Action
\": [\"eks-mcp:*\"], \"Resource\": \"*\"}]}"
```

ステップ 3: セットアップを確認する

接続をテストする

AI アシスタントに簡単な質問をして、接続を確認します。

```
List all EKS clusters in my {aws} account
```

EKS クラスターのリストが表示されます。

ステップ 4: 最初のタスクを実行する

例 1: クラスターを調べる

```
Show me all EKS clusters and their status
What insights does EKS have about my production-cluster?
Show me the VPC configuration for my staging cluster
```

例 2: Kubernetes リソースを確認する

```
Get the details of all the kubernetes resources deployed in my EKS cluster
```

```
Show me pods that are not in Running state or pods with any restarts
Get the logs from the aws-node daemonset in the last 30 minutes
```

例 3: 問題のトラブルシューティングを行う

```
Why is my nginx-ingress-controller pod failing to start?
Search the EKS troubleshooting guide for pod networking issues
Show me events related to the failed deployment in the staging namespace
```

例 4: リソースを作成する (「書き込み」モードが有効になっている場合)

```
Create a new EKS cluster named demo-cluster with VPC and Auto Mode
Deploy my containerized app from ECR to the production namespace with 3 replicas
Generate a Kubernetes deployment YAML for my Node.js app running on port 3000
```

一般的な設定

シナリオ 1: 複数の AWS プロファイル

複数の AWS アカウントを使用する場合は、個別の MCP サーバー設定を作成します。

Mac/Linux の場合:

```
{
  "mcpServers": {
    "eks-mcp-prod": {
      "disabled": false,
      "type": "stdio",
      "command": "uvx",
      "args": [
        "mcp-proxy-for-aws@latest",
        "https://eks-mcp.us-west-2.api.aws/mcp",
        "--service",
        "eks-mcp",
        "--profile",
        "production",
        "--region",
        "us-west-2"
      ]
    },
    "eks-mcp-dev": {
      "disabled": false,
```

```
      "type": "stdio",
      "command": "uvx",
      "args": [
        "mcp-proxy-for-aws@latest",
        "https://eks-mcp.us-east-1.api.aws/mcp",
        "--service",
        "eks-mcp",
        "--profile",
        "development",
        "--region",
        "us-east-1"
      ]
    }
  }
}
```

Windows の場合 –

```
{
  "mcpServers": {
    "eks-mcp-prod": {
      "disabled": false,
      "type": "stdio",
      "command": "uvx",
      "args": [
        "--from",
        "mcp-proxy-for-aws@latest",
        "mcp-proxy-for-aws.exe",
        "https://eks-mcp.us-west-2.api.aws/mcp",
        "--service",
        "eks-mcp",
        "--profile",
        "production",
        "--region",
        "us-west-2"
      ]
    },
    "eks-mcp-dev": {
      "disabled": false,
      "type": "stdio",
      "command": "uvx",
      "args": [
        "--from",
```

```
        "mcp-proxy-for-aws@latest",
        "mcp-proxy-for-aws.exe",
        "https://eks-mcp.us-east-1.api.aws/mcp",
        "--service",
        "eks-mcp",
        "--profile",
        "development",
        "--region",
        "us-east-1"
      ]
    }
  }
}
```

シナリオ 2: 本番環境の読み取り専用

本番環境の読み取り専用設定を作成します。

Mac/Linux の場合:

```
{
  "mcpServers": {
    "eks-mcp-prod-readonly": {
      "command": "uvx",
      "args": [
        "mcp-proxy-for-aws@latest",
        "https://eks-mcp.us-west-2.api.aws/mcp",
        "--service",
        "eks-mcp",
        "--profile",
        "production",
        "--region",
        "us-west-2",
        "--read-only"
      ],
      "autoApprove": [
        "list_k8s_resources",
        "get_pod_logs",
        "get_k8s_events"
      ]
    }
  }
}
```

Windows の場合 –

```
{
  "mcpServers": {
    "eks-mcp-prod-readonly": {
      "command": "uvx",
      "args": [
        "--from",
        "mcp-proxy-for-aws@latest",
        "mcp-proxy-for-aws.exe",
        "https://eks-mcp.us-west-2.api.aws/mcp",
        "--service",
        "eks-mcp",
        "--profile",
        "production",
        "--region",
        "us-west-2",
        "--read-only"
      ],
      "autoApprove": [
        "list_k8s_resources",
        "get_pod_logs",
        "get_k8s_events"
      ]
    }
  }
}
```

シナリオ 3: フルアクセスを備えた開発

フル書き込みアクセスを備えた開発環境用。

Mac/Linux の場合:

```
{
  "mcpServers": {
    "eks-mcp-dev-full": {
      "command": "uvx",
      "args": [
        "mcp-proxy-for-aws@latest",
        "https://eks-mcp.us-east-1.api.aws/mcp",
        "--service",
        "eks-mcp",
```

```
        "--profile",
        "development",
        "--region",
        "us-east-1"
      ]
    }
  }
}
```

Windows の場合 –

```
{
  "mcpServers": {
    "eks-mcp-dev-full": {
      "command": "uvx",
      "args": [
        "--from",
        "mcp-proxy-for-aws@latest",
        "mcp-proxy-for-aws.exe",
        "https://eks-mcp.us-east-1.api.aws/mcp",
        "--service",
        "eks-mcp",
        "--profile",
        "development",
        "--region",
        "us-east-1"
      ]
    }
  }
}
```

考慮事項

セキュリティ

許可された入力メカニズムを介してシークレットや機密情報を渡さない:

- apply_yaml で適用される YAML ファイルにシークレットや認証情報を含めないでください。
- 機密情報をプロンプトでモデルに直接渡さないでください。
- CloudFormation テンプレートまたはアプリケーションマニフェストにシークレットを含めないでください。

- モデルにシークレットデータを提供する必要があるため、Kubernetes Secrets の作成に MCP ツールを使用しないでください。
- Kubernetes ポッド内のアプリケーションログに、機密情報を記録しないでください。

YAML コンテンツのセキュリティ:

- 信頼できるソースの YAML ファイルのみを使用してください。
- サーバーは YAML コンテンツについて Kubernetes API の検証に依存しており、独自の検証は実行しません。
- クラスターに適用する前に、YAML ファイルを監査してください。

MCP を介してシークレットを渡す代わりに、次の操作を行います。

- 機密情報の保存には、AWS Secrets Manager または Parameter Store を使用します。
- サービスアカウントに適切な Kubernetes RBAC を設定します。
- ポッドから AWS サービスにアクセスするには、サービスアカウントの IAM ロール (IRSA) を使用します。

機密データの秘匿化:

- EKS MCP サーバーは、ツールレスポンス内のセキュリティトークン、証明書、およびその他の機密情報の一般的なパターンを自動的に秘匿化します。
- 秘匿化された値は、誤ってデータがモデルに公開されないよう、`HIDDEN_FOR_SECURITY_REASONS` に置き換えられます。
- この秘匿化は、ログ、リソースの説明、設定データを含むすべてのツールレスポンスに適用されます。

次回の予定

設定オプションについては、「the section called “設定”」を参照してください。ツールの詳細なリストについては、「the section called “ツール”」を参照してください。

Amazon EKS MCP サーバーの設定リファレンス

このガイドでは、IDE からフルマネージドの Amazon EKS MCP サーバーに接続できる、mcp-proxy-for-aws のクライアント側ツールで利用可能なすべての設定を示します。

> **Note**
>
> Amazon EKS MCP サーバー は Amazon EKS のプレビュー版であり、変更される可能性があります。

例

```
{
  "mcpServers": {
    "eks-mcp": {
      "disabled": false,
      "type": "stdio",
      "command": "uvx",
      "args": [
        "mcp-proxy-for-aws@latest",
        "https://eks-mcp.us-west-2.api.aws/mcp",
        "--service",
        "eks-mcp",
        "--profile",
        "default",
        "--region",
        "us-east-1",
        "--read-only"
      ]
    }
  }
}
```

IAM アクセス許可

MCP サーバーへの接続に使用されるロールには、初期化および使用可能なツールに関する情報の取得のために **`eks-mcp:InvokeMcp`** のアクセス許可が必要です。 **`eks-mcp:CallReadOnlyTool`** は読み取り専用ツールの使用に必要であり、 **`eks-mcp:CallPrivilegedTool`** はフルアクセス (書き込み) ツールの使用に必要です。

環境変数

`AWS_PROFILE` (オプション) AWS の認証情報プロファイル名。`--profile` コマンドライン引数で上書きできます。

- 例:`export AWS_PROFILE=production`

`AWS_REGION` (オプション) SigV4 署名用の AWS リージョン。設定されていない場合は `us-west-2` がデフォルトになります。

- 例:`export AWS_REGION=us-east-1`

引数

SigV4 MCP エンドポイント URL (必須) 接続先の MCP エンドポイント URL。

- 例:`https://eks-mcp.us-west-2.api.aws/mcp`

`--service` (オプション) SigV4 署名用の AWS のサービス名。指定されていない場合は、エンドポイントホスト名から自動検出されます。

- 例:`--service eks-mcp`

`--profile` (オプション) 使用する AWS 認証情報プロファイル。指定されていない場合は、`AWS_PROFILE` 環境変数がデフォルトになります。

- 例:`--profile production`

`--region` 使用する AWS リージョン。設定されていない場合は、`AWS_REGION` 環境変数を使用します。デフォルトは `us-east-1` です。

- 例:`--region us-west-2`

`--read-only` (オプション) 書き込みアクセス許可が必要なツールを無効にします (書き込みアクセス許可を必要としないツールには readOnlyHint=true という注釈が付けられます)。デフォルトでは、すべてのツールが有効になっています。

- 例:`--read-only`

その他の設定オプションについては、「Configuration parameters」を参照してください。

ツール

パラメータや使用例など、使用可能なすべてのツールの詳細については、「the section called “ツール”」を参照してください。

Amazon EKS MCP サーバーのツールリファレンス

サーバーは、次の MCP ツールを公開しています。

> **Note**
>
> Amazon EKS MCP サーバー は Amazon EKS のプレビュー版であり、変更される可能性があります。

読み取り専用ツール

このセクションでは、EKS MCP サーバーで使用できる読み取り専用ツールについて説明します。すべての読み取り専用の Kubernetes API オペレーションは、次の両方にアクセスできることに注意してください。

- プライベートクラスター (「クラスタープライベートエンドポイント」を参照)
- パブリッククラスター

search_eks_documentation

最新の情報とガイダンスを得るには、EKS ドキュメントを検索してください。このツールは、エージェントがまだ把握していない可能性のある新機能や最近の機能強化を含む、最新の EKS ドキュメントへのアクセスを提供します。

パラメータ:

- query (必須): EKS のドキュメント、機能、またはベストプラクティスに関連する特定の質問または検索クエリ。
- limit (オプション): 返されるドキュメント結果の最大数 (1 ~ 10)。デフォルト: 5

search_eks_troubleshooting_guide

クエリに基づいてトラブルシューティング情報を得るために、EKS トラブルシューティングガイドを検索します。一般的な問題を特定し、ステップバイステップの解決策を提供します。

パラメータ:

- query (必須): EKS のトラブルシューティングに関連する特定の質問または問題の説明。

describe_eks_resource

設定、ステータス、メタデータなど、特定の EKS クラスターリソースに関する詳細情報を取得します。

パラメータ:

- cluster_name (必須): EKS クラスターの名前 (クラススコープのリソースで必要)。
- resource_type (必須): 記述する EKS リソースタイプ。有効な値:
- accessentry (principalArn として cluster_name と resource_name が必要)
- addon (アドオン名として cluster_name と resource_name が必要)
- cluster (cluster_name が必要)、nodegroup (nodegroup 名として cluster_name と resource_name が必要)。
- resource_name (オプション): 記述する特定のリソースの名前 (ほとんどのリソースタイプで必要)。

list_eks_resources

特定のタイプの EKS リソースを一覧表示し、アクセス可能な指定されたタイプのすべてのリソースの概要を返します。

パラメータ:

- resource_type (必須): 一覧表示する EKS リソースタイプ。有効な値:
- accessentry (cluster_name が必要)
- addon (cluster_name が必要)
- クラスター (追加のパラメータは不要)
- nodegroup (cluster_name が必要)。

- cluster_name (オプション): EKS クラスターの名前 (クラススコープのリソースで必要)。

get_eks_insights

最適化のための EKS クラスターインサイトと推奨事項を取得します。AWS ベストプラクティスとクラスター分析に基づいて、セキュリティ、パフォーマンス、コスト最適化に関する実用的なインサイトを提供します。

パラメータ:

- cluster_name (必須): EKS クラスターの名前。
- category (オプション): インサイトをフィルタリングするためのオプションのカテゴリ (「MISCONFIGURATION」または「UPGRADE_READINESS」など)。
- insight_id (オプション): 詳細情報を取得するための特定のインサイトのオプション ID。
- next_token (オプション): 次の結果セットを取得するためのページ分割用のオプションのトークン。

get_eks_vpc_config

サブネット、ルートテーブル、ネットワーク接続など、EKS クラスターの VPC 設定を取得します。

パラメータ:

- cluster_name (必須): VPC 設定を取得する EKS クラスターの名前。
- vpc_id (オプション): クエリする特定の VPC の ID (オプション、指定しない場合はクラスター の VPC が使用されます)。

get_k8s_events

トラブルシューティングとモニタリングのために、特定のリソースに関連する Kubernetes イベントを取得します。

パラメータ:

- cluster_name (必須): リソースが配置されている EKS クラスターの名前。
- kind (必須): 関係するオブジェクトの種類 (「ポッド」、「デプロイ」、「サービス」など)。リソースの kind と正確に一致する必要があります。

- name (必須): イベントを取得する対象のオブジェクトの名前。
- namespace (オプション): 関係するオブジェクトの名前空間。名前空間リソース (ポッド、デプロイなど) で必要です。クラスタースコープのリソース (Nodes、PersistentVolumes など) には必要ありません。

get_pod_logs

フィルタリングオプションを使用して、EKS クラスター内のポッドからログを取得します。

パラメータ:

- cluster_name (必須): ポッドが実行されている EKS クラスターの名前。
- 名前空間 (必須): ポッドが配置されている Kubernetes 名前空間。
- pod_name (必須): ログを取得するポッドの名前。
- container_name (オプション): ログを取得する特定のコンテナの名前。ポッドに複数のコンテナが含まれている場合にのみ必要です。
- limit_bytes (オプション): 返す最大バイト数。デフォルト: 10 KB (10240 バイト)。
- previous (オプション): 以前に終了したコンテナログを返します (デフォルトは false)。再起動中のポッドのログを取得するのに役立ちます。
- since_seconds (オプション): 指定した秒数より新しいログのみを返します。履歴全体を取得せずに最近のログを取得する場合に便利です。
- tail_lines (オプション): ログの末尾から返す行数。デフォルト: 100。

list_api_versions

指定された Kubernetes クラスターで使用可能なすべての API バージョンを一覧表示します。

パラメータ:

- cluster_name (必須): EKS クラスターの名前。

list_k8s_resources

EKS クラスター内の特定の種類の Kubernetes リソースを一覧表示します。

パラメータ:

- cluster_name (必須): リソースが配置されている EKS クラスターの名前。
- kind (必須): 一覧表示する Kubernetes リソースの種類 (「ポッド」、「サービス」、「デプロイ」など)。`list_api_versions` ツールを使用して、使用可能なリソースの種類を検索します。
- api_version (必須): API バージョンの Kubernetes リソース (「v1」、「apps/v1」、「networking.k8s.io/v1」など)。`list_api_versions` ツールを使用して、使用可能な API バージョンを検索します。
- field_selector (オプション): リソースをフィルタリングするフィールドセレクタ (「metadata.name=my-pod,status.phase=Running」など)。kubectl の --field-selector フラグと同じ構文を使用します。
- label_selector (オプション): リソースをフィルタリングするラベルセレクタ (「app=nginx,tier=frontend」など)。kubectl の `--selector` フラグと同じ構文を使用します。
- namespace (オプション): 一覧表示する Kubernetes リソースの名前空間。指定しない場合、(名前空間リソースでは) すべての名前空間にわたってリソースが一覧表示されます。

read_k8s_resource

EKS クラスター内の特定の Kubernetes リソースに関する詳細情報を取得します。

パラメータ:

- api_version (必須): API バージョンの Kubernetes リソース (「v1」、「apps/v1」、「networking.k8s.io/v1」など)。
- cluster_name (必須): リソースが配置されている EKS クラスターの名前。
- kind (必須): Kubernetes リソースの種類 (「ポッド」、「サービス」、「デプロイ」など)。
- name (必須): 読み取る Kubernetes リソースの名前。
- namespace (オプション): Kubernetes リソースの名前空間。名前空間リソースに必要です。クラスタースコープのリソース (Nodes、PersistentVolumes など) には必要ありません。

generate_app_manifest

コンテナ化されたアプリケーション向けに、標準化された Kubernetes デプロイとサービスマニフェストを生成します。

パラメータ:

- app_name (必須): アプリケーションの名前。デプロイ名とサービス名、ラベルに使用されます。

- image_uri (必須): タグ付きの完全な ECR イメージ URI (123456789012.dkr.ecr.region.amazonaws.com/repo:tag など)。完全なリポジトリパスとタグを含める必要があります。
- load_balancer_scheme (オプション): AWSロードバランサースキーム。有効な値:
- 「internal」 (プライベート VPC のみ)
- 「internet-facing」 (パブリックアクセス)。
- デフォルト: 「internal」。
- cpu (オプション): 各コンテナの CPU リクエスト (例: 0.1 CPU コアの場合は「100m」、半コアの場合は「500m」)。デフォルト: 「100m」。
- memory (オプション): 各コンテナのメモリリクエスト (例: 128 MiB の場合は「128Mi」、1 GiB の場合は「1Gi」)。デフォルト: 「128Mi」
- namespace (オプション): アプリケーションをデプロイする Kubernetes 名前空間。デフォルト: 「default」。
- port (オプション): アプリケーションがリッスンするコンテナポート。デフォルト: 80
- replicas (オプション): デプロイするレプリカの数。デフォルト: 2

get_cloudwatch_logs

入力パラメータに基づいてフィルタリングし、EKS クラスターオブザーバビリティに使用される標準ロググループをサポートする CloudWatch ログをクエリします。

パラメータ:

- cluster_name (必須): リソースが配置されている EKS クラスターの名前。CloudWatch ロググループ名を構築するために使用します。
- resource_type (必須): ログを検索するリソースタイプ。有効な値: +「pod」、「node」、「container」、「cluster」。これにより、ログのフィルタリング方法が決まります。
- log_type (必須): クエリするログタイプ。有効な値:
- 「application」: コンテナ/アプリケーションログ
- 「host」: ノードレベルのシステムログ
- 「performance」: パフォーマンスメトリクスログ
- 「control-plane」: EKS コントロールプレーンログ
- 「your-log-group-name」: カスタム CloudWatch ロググループ名を直接指定します。

- resource_name (オプション): ログメッセージで検索するリソース名 (ポッド名、ノード名、コンテナ名など)。特定のリソースのログをフィルタリングするために使用されます。
- minutes (オプション): ログをさかのぼる分数。デフォルト: 15。start_time が指定されている場合は無視されます。最近の問題には小さい値を、履歴分析には大きい値を使用します。
- start_time (オプション): ISO 形式の開始時刻 (例: 「2023-01-01T00:00:00Z」)。指定した場合、minutes パラメータを上書きします。
- end_time (オプション): ISO 形式の終了時刻 (例: 「2023-01-01T01:00:00Z」)。指定しない場合、デフォルトで現在の時刻になります。
- fields (オプション): クエリ結果に含めるカスタムフィールド (デフォルトは「@timestamp, @message」)。CloudWatch Logs Insights のフィールド構文を使用します。
- filter_pattern (オプション): 適用する追加の CloudWatch Logs フィルターパターン。CloudWatch Logs Insights の構文 (「ERROR」、「field=value」など) を使用します。
- limit (オプション): 返されるログエントリの最大数。より高速なクエリには低い値 (10～50) を、より包括的な結果には高い値 (100～1000) を使用します。値を大きくすると、パフォーマンスに影響する可能性があります。

get_cloudwatch_metrics

EKS クラスターのモニタリングとパフォーマンス分析のために、CloudWatch メトリクスとデータポイントを取得します。Container Insights メトリクス、カスタムメトリクス、および設定可能な期間とディメンションを処理します。

パラメータ:

- cluster_name (必須): メトリクスを取得する EKS クラスターの名前。
- dimensions (必須): CloudWatch メトリクスクエリに JSON 文字列として使用するディメンション。リソースタイプとメトリクスに適したディメンションを含める必要があります (例: '{"ClusterName": "my-cluster", "PodName": "my-pod", "Namespace": "default"}')。
- metric_name (必須): 取得するメトリクス名。一般的な例:
- `cpu_usage_total`: 合計 CPU 使用率
- `memory_rss`: レジデントセットサイズのメモリ使用量
- `network_rx_bytes`: 受信したネットワークバイト数
- `network_tx_bytes`: 送信されたネットワークバイト数
- namespace (必須): メトリクスが保存されている CloudWatch 名前空間。共通の値:

- "ContainerInsights": コンテナメトリクス用
- "AWS/EC2": EC2 インスタンスメトリクス用
- "AWS/EKS": EKS コントロールプレーンメトリクス用
- minutes (オプション): メトリクスをさかのぼる分数。デフォルト: 15。start_time が指定されている場合は無視されます。
- start_time (オプション): ISO 形式の開始時刻 (例: 「2023-01-01T00:00:00Z」)。指定した場合、minutes パラメータを上書きします。
- end_time (オプション): ISO 形式の終了時刻 (例: 「2023-01-01T01:00:00Z」)。指定しない場合、デフォルトで現在の時刻になります。
- limit (オプション): 返されるデータポイントの最大数。値 (100 ~ 1000) が大きいほど、より詳細なデータが得られますが、パフォーマンスに影響する場合があります。デフォルト: 50。
- period (オプション): メトリクスデータポイントの秒単位の期間。デフォルト: 60 (1 分)。小さい値 (1 ~ 60) ほど解像度は高くなりますが、使用できなくなる場合があります。
- stat (オプション): メトリクスの集計に使用する統計。デフォルト: 「Average」。有効な値:
- Average: 期間中の平均値
- Sum: 期間中の合計値
- Maximum: 期間中の最大値
- Minimum: 期間中の最小値
- SampleCount: 期間中のサンプルの数。

get_eks_metrics_guidance

EKS クラスター内の特定のリソースタイプに対する CloudWatch メトリクスのガイダンスを取得します。get_cloudwatch_metrics ツールで使用する適切なディメンションを決定するときに、エージェントに役立ちます。

パラメータ:

- resource_type (必須): メトリクスを取得するリソースのタイプ (クラスター、ノード、ポッド、名前空間、サービス)。

get_policies_for_role

ロール継承ポリシー、マネージドポリシー、インラインポリシーなど、指定された IAM ロールにアタッチされているすべてのポリシーを取得します。

パラメータ:

- role_name (必須): ポリシーを取得する IAM ロールの名前。ロールは、ご自身の AWS アカウント内に存在する必要があります。

フルアクセス (書き込み) ツール

このセクションでは、EKS MCP サーバーで使用できるフルアクセス (書き込み) ツールについて説明します。(現時点で) すべての書き込み Kubernetes API オペレーションは、次のものにのみアクセスできることに注意してください。

- パブリッククラスター (endpointPublicAccess=true)

manage_k8s_resource

書き込みオペレーション (作成、更新、パッチ、削除) を使用して、単一の Kubernetes リソースを管理します。

パラメータ:

- operation (必須): リソースに対して実行するオペレーション。有効な値:
- `create`: 新しいリソースを作成する
- `replace`: 既存のリソースを置き換える
- `patch`: 既存のリソースの特定のフィールドを更新する
- `delete`: 既存のリソースを削除する
- Note: リソースの読み取りには read_k8s_resource を使用し、複数のリソースを一覧表示するには list_k8s_resources を使用します。
- cluster_name (必須): リソースが配置されている、または作成される EKS クラスターの名前。
- kind (必須): Kubernetes リソースの種類 (「ポッド」、「サービス」、「デプロイ」など)。
- api_version (必須): API バージョンの Kubernetes リソース (「v1」、「apps/v1」、「networking.k8s.io/v1」など)。
- body (オプション): JSON 文字列としてのリソース定義。作成、置換、およびパッチオペレーションに必要です。作成と置換の場合、これは完全なリソース定義である必要があります。パッチの場合、更新するフィールドのみを含める必要があります。
- name (オプション): Kubernetes リソースの名前。作成を除くすべてのオペレーションに必要です (本文で指定できる場合)。

- namespace (オプション): Kubernetes リソースの名前空間。名前空間リソースに必要です。クラスタースコープのリソース (Nodes、PersistentVolumes など) には必要ありません。

apply_yaml

Kubernetes YAML マニフェストを EKS クラスターに適用します。

パラメータ:

- cluster_name (必須): リソースが作成または更新される EKS クラスターの名前。
- namespace (必須): リソースを適用する Kubernetes 名前空間。名前空間を指定していない名前空間スコープのリソースに使用されます。
- yaml_content (必須): クラスターに適用する YAML コンテンツ。「---」で区切られた複数のドキュメントを含めることができます。
- force (オプション): リソースが既に存在する場合に更新するかどうか (kubectl apply と同様)。false に設定すると、新しいリソースのみが作成されます。

manage_eks_stacks

EKS クラスターとその基盤となるインフラストラクチャのテンプレート生成、デプロイ、記述、削除などのオペレーションにより、EKS CloudFormation スタックを管理します。クラスターの作成には通常、15～20 分かかります。デプロイおよび削除オペレーションの場合、スタックはこのツールによって作成されている必要があります (つまり、CreatedBy=EksMcpServer でタグ付けされている必要があります)。

パラメータ:

- cluster_name (必須): EKS クラスターの名前 (生成、デプロイ、記述、削除オペレーション用)。この名前は CloudFormation スタック名の取得に使用され、クラスターリソースに埋め込まれます。
- operation (必須): 実行するオペレーション。有効な値:
- `generate`: CloudFormation テンプレートを生成する
- `deploy`: CloudFormation スタックをデプロイする (`template_content` が必要)
- `describe`: CloudFormation スタックを記述/読み取りする (読み取り専用)
- `delete`: CloudFormation スタックを削除する
- template_content (オプション): CloudFormation テンプレートコンテンツ (デプロイオペレーション用)。これは、完全な YAML または JSON テンプレートコンテンツである必要があります。単一リソースと「---」で区切られたマルチドキュメント YAML コンテンツの両方をサポートします。

add_inline_policy

IAM ロールに新しいインラインポリシーを追加します。

パラメータ:

- permissions (必須): IAM ポリシーステートメントを表す JSON 文字列としてポリシーに含めるアクセス許可。単一の JSON 文字列でも、JSON 文字列の配列でもかまいません。
- policy_name (必須): 作成するインラインポリシーの名前。名前はロール内で一意でなければなりません。
- role_name (必須): ポリシーを追加する IAM ロールの名前。ロールが存在している必要があります。

Amazon EKS コンソールで Amazon Q Developer を使用する

Amazon Elastic Kubernetes Service (EKS) は、AI を活用したトラブルシューティングを AWS マネジメントコンソールで直接利用できるよう、Amazon Q と統合されました。この統合により、Amazon Q の AI アシスタントを利用して、クラスター、コントロールプレーン、ノード、ワークロードの問題を迅速に調査して解決できます。

仕組み

Amazon EKS コンソールでは、コンソールの至る所でエラーや問題の他にコンテキストに沿って [Amazon Q で検査] ボタンが表示されます。このボタンをクリックすると、Amazon Q が自動的に問題を分析し、コンソールの右側にチャットパネルを開いて調査結果、根本原因分析、推奨される緩和ステップを示します。

統合した機能は、Amazon EKS コンソール内の次の場所に表示されます。

- クラスターのヘルス - オブザーバビリティダッシュボードの [クラスターのヘルス] タブで、クラスターのヘルスの問題とステータスメッセージを調査します。
- コントロールプレーン - オブザーバビリティダッシュボードの [コントロールプレーン監視] でコントロールプレーンコンポーネントのエラーと警告をトラブルシューティングします。
- アップグレードインサイト - オブザーバビリティダッシュボードの [アップグレードインサイト] でアップグレードブロッカーと互換性に潜在的な問題がないか分析します。
- ノードヘルス - オブザーバビリティダッシュボードの [ノードヘルスの問題] でクラスターキャパシティに影響するノードレベルの問題を調査します。
- ワークロード - ポッドで Kubernetes イベントを分析して障害や問題の兆候がないか確認します。

トラブルシューティングに Amazon Q を使用する

Amazon Q で問題を調査するには

1. Amazon EKS コンソールを開きます。
2. 調査するクラスターの名前を選択します。
3. エラーメッセージや問題インジケータが表示された場合は、[Amazon Q で検査] ボタンを探してください。このボタンは、コンテキストに沿って問題の隣またはエラーステータスの詳細ビューに表示されます。
4. [Amazon Q で検査] を選択します。
5. Amazon Q は、自動的に問題を調査し、コンソールの右側にチャットパネルを開いて分析を表示します。
6. 根本原因分析や推奨される緩和ステップなど、調査結果を確認します。
7. チャットの会話を続ける場合は、問題の関連質問を Amazon Q に尋ねます。

注意 Amazon Q 統合機能は、エラー、警告、または調査を要する問題がある場合にのみ表示されます。リソースが正常である場合には表示されません。

考慮事項

Amazon EKS で Amazon Q を使用する場合は、次の点を考慮してください。

- 読み取り専用オペレーション – Amazon Q 統合機能は、クラスターリソースに対して読み取りオペレーションのみを実行します。クラスター設定やワークロードに対して変化アクションや書き込みアクションは実行することはありません。
- クロスリージョン処理 – Amazon Q は、AI を活用した分析を提供するために、AWS リージョン間でデータを処理することがあります。詳細については、「Amazon Q Developer ユーザーガイド」の「クロスリージョン処理」を参照してください。
- AWS マネジメントコンソール専用 – この統合機能は、AWS マネジメントコンソールでのみ使用できます。AWS CLI、AWS API、Infrastructure as Code ツールでは使用できません。

詳細情報

Amazon Q の使用方法の詳細については、「Amazon Q Developer ユーザーガイド」の「Amazon Q とのチャット」を参照してください。

Amazon EKS でのバージョニング

このセクションでは、Amazon Elastic Kubernetes Service (EKS) 内での Kubernetes バージョニングの動作について説明します。サポートされているバージョン、廃止およびサポートポリシー、アップグレードプロセス、クラスターバージョンを管理するための重要な考慮事項に関する詳細を確認できます。

トピック

- EKS の Kubernetes バージョンライフサイクルを理解する
- 各 Kubernetes バージョンの Amazon EKS プラットフォームバージョンを表示する

EKS の Kubernetes バージョンライフサイクルを理解する

> **Tip**
>
> 今後開催予定の Amazon EKS ワークショップに登録してください。

Kubernetes は、新しい機能、設計の更新、バグの修正と共に急速に進化しています。コミュニティでは、平均して 4 か月に一度、新しい Kubernetes のマイナーバージョン (1.37 など) をリリースしています。Amazon EKS はマイナーバージョンのアップストリームリリースと非推奨サイクルに従います。新しい Kubernetes バージョンが Amazon EKS で利用可能になったら、利用可能な最新のバージョンが使用できるように、クラスターをタイムリーに更新することをお勧めします。

マイナーバージョンはリリース後最初の 14 か月間は Amazon EKS の標準サポート対象となります。標準サポート終了日を過ぎたバージョンは次の 12 か月間の延長サポートに入ります。延長サポートでは、クラスター時間あたりの追加料金を支払えば、特定の Kubernetes バージョンを延長して使用することができます。延長サポート期間が終了する前にクラスターを更新しなかった場合、クラスターは現在サポートされている最も古い延長バージョンに自動的にアップグレードされます。

延長サポートはデフォルトで有効になっています。無効にするには、「EKS 延長サポートを無効にする」を参照してください。

Amazon EKS でサポートされている利用可能な最新の Kubernetes バージョンを使用してクラスターを作成することをお勧めします。アプリケーションに特定のバージョンの Kubernetes が必要な場合は、古いバージョンを選択できます。標準サポートまたは延長サポートが提供されているどのバージョンでも、新しい Amazon EKS クラスターを作成できます。

標準サポートで入手可能なバージョン

Amazon EKS 標準サポートでは、現在以下の Kubernetes バージョンを利用できます。

- 1.37
- 1.36
- 1.35
- 1.34

標準サポートの各バージョンで注意すべき重要な変更点については「Kubernetes バージョンの標準サポート」を参照してください。

延長サポートで利用可能なバージョン

Amazon EKS 延長サポートでは、現在以下の Kubernetes バージョンを利用できます。

- 1.33
- 1.32
- 1.31

拡張サポートの各バージョンで注意すべき重要な変更点については「Kubernetes バージョンの延長サポート」を参照してください。

バージョンのロールバック

インプレースアップグレードを通じてアップグレードされたクラスターは、アップグレード完了後の 7 日以内に以前のマイナーバージョンにロールバックできます。延長サポートでバージョンにロールバックするには、まずクラスターのアップグレードポリシーを EXTENDED に変更する必要があります。延長サポートの終了時に自動的にアップグレードされたクラスターは、ロールバックできません。詳細については、「the section called “Kubernetes バージョンのロールバック”」を参照してください。

Amazon EKS Kubernetes リリースカレンダー

次の表は、Kubernetes バージョンごとに検討すべき重要なリリース日とサポート日を示しています。延長サポートの請求はバージョンが標準サポートの終了日を迎える日の開始時刻 (UTC+0 タイムゾーン) に開始されます。次の表の日付は、UTC+0 タイムゾーンを使用しています。

> **Note**
>
> 月と年のみの日付はおおよその日付であり、確定後に正確な日付で更新されます。

この特定のドキュメントページへのすべてのソースファイルの変更通知を受け取るには、RSS リーダーを使用して次の URL をサブスクライブできます。

```
https://github.com/awsdocs/amazon-eks-user-guide/commits/mainline/latest/ug/versioning/kubernetes-versions.adoc.atom
```

Kubernetes バージョン	アップストリームのリリース	Amazon EKS リリース	標準サポート終了日	延長サポートの終了日
1.37	2026 年 8 月 26 日	2026 年 10 月 1 日	2027 年 12 月 1 日	2028 年 12 月 1 日
1.36	2026 年 4 月 22 日	2026 年 6 月 2 日	2027 年 8 月 2 日	2028 年 8 月 2 日

Kubernetes バージョン	アップストリームのリリース	Amazon EKS リリース	標準サポート終了日	延長サポートの終了日
1.35	2025 年 12 月 17 日	2026 年 1 月 27 日	2027 年 3 月 27 日	2028 年 3 月 27 日
1.34	2025 年 8 月 27 日	2025 年 10 月 2 日	2026 年 12 月 2 日	2027 年 12 月 2 日
1.33	2025 年 4 月 23 日	2025 年 5 月 29 日	2026 年 7 月 29 日	2027 年 7 月 29 日
1.32	2024 年 12 月 11 日	2025年1月23日	2026 年 3 月 23 日	2027年3月23日
1.31	2024 年 8 月 13 日	2024 年 9 月 26 日	2025 年 11 月 26 日	2026 年 11 月 26 日

AWS CLI を使用してバージョン情報を取得する

AWS CLI を使用すると、標準サポートの終了日など、EKS で利用可能な Kubernetes バージョンに関する情報を取得できます。

AWS CLI を使用して EKS で利用可能な Kubernetes バージョンに関する情報を取得するには

1. ターミナルを開きます。
2. AWS CLI がインストールされ、設定されていることを確認します。詳細については「CLI の最新バージョンのインストールまたはアップデート」を参照してください。
3. 次のコマンドを実行してください。

```
aws eks describe-cluster-versions
```

4. このコマンドは使用可能なクラスターバージョンに関する詳細を含む JSON 出力を返します。以下に出力の例を示します。

```
{
    "clusterVersions": [
```

```
        {
            "clusterVersion": "1.31",
            "clusterType": "eks",
            "defaultPlatformVersion": "eks.21",
            "defaultVersion": true,
            "releaseDate": "2024-09-25T17:00:00-07:00",
            "endOfStandardSupportDate": "2025-11-25T16:00:00-08:00",
            "endOfExtendedSupportDate": "2026-11-25T16:00:00-08:00",
            "status": "STANDARD_SUPPORT",
            "kubernetesPatchVersion": "1.31.3"
        }
    ]
}
```

この出力にはクラスターバージョンごとに次の情報が表示されます。

- `clusterVersion`: EKS クラスターの Kubernetes バージョン
- `clusterType`: クラスターのタイプ (「eks」など)
- `defaultPlatformVersion`: デフォルトの EKS プラットフォームバージョン
- `defaultVersion`: これがデフォルトのバージョンであるかどうか
- `releaseDate`: このバージョンがリリースされた日付
- `endOfStandardSupportDate`: 標準サポートが終了する日付
- `endOfExtendedSupportDate`: 延長サポートが終了する日付
- `status`: `STANDARD_SUPPORT` や `EXTENDED_SUPPORT` などの、そのバージョンの現在のサポートステータス
- `kubernetesPatchVersion`: 特定の Kubernetes パッチバージョン

Amazon EKS のバージョンのよくある質問

標準サポートで利用できる Kubernetes バージョンはいくつありますか?

Kubernetes のバージョンに対する Kubernetes コミュニティのサポートに沿って、Amazon EKS ではいつでも少なくとも 3 つの Kubernetes バージョンのサポートを提供するように努めています。特定の Kubernetes マイナーバージョンの標準サポートを終了する場合は、終了日の少なくとも 60 日前に発表します。Kubernetes の新しいバージョンの Amazon EKS における認定およびリリースプロセスにより、Amazon EKS での Kubernetes バージョンの標準サポート終了日

は、Kubernetes プロジェクトがそのバージョンでアップストリームを停止した日よりも後になります。

Kubernetes バージョンは、Amazon EKS による標準サポートをどのくらいの期間受けられますか?

Kubernetes バージョンは、Amazon EKS で最初に利用可能になってから 14 か月間標準サポートを受けていました。これは、Amazon EKS で利用可能なバージョンが Kubernetes アップストリームでサポートされなくなった場合にも当てはまります。Amazon は、Amazon EKS でサポートされている Kubernetes の各バージョンに適用されるセキュリティパッチをバックポートしています。

Kubernetes バージョンのサポートが Amazon EKS で終了する際には通知が届きますか?

はい。アカウント内のいずれかのクラスターでサポートの終了が近い Kubernetes バージョンを実行している場合、そのバージョンが Amazon EKS でリリースされてから約 12 か月後に、Amazon EKS から AWS Health Dashboard を介して通知が送信されます。通知にはサポート終了日が含まれます。これは通知の日から 60 日以上後です。

Amazon EKS では、どの Kubernetes 機能がサポートされていますか?

Amazon EKS では、Kubernetes API の一般提供されている (GA) すべての機能がサポートされています。新しいベータ版 API はデフォルトではクラスターで有効になっていません。デフォルトでは既存のベータ API と既存のベータ API の新しいバージョンは引き続き有効になっています。アルファ機能はサポートされません。

Amazon EKS のマネージド型ノードグループはクラスターのコントロールプレーンのバージョンとともに自動的に更新されますか?

いいえ。マネージドノード型グループではアカウントに Amazon EC2 インスタンスを作成します。これらのインスタンスはお客様または Amazon EKS がコントロールプレーンを更新しても、自動的にはアップグレードされません。詳細については、「the section called “更新”」を参照してください。コントロールプレーンとノードでは、Kubernetes バージョンを同じに維持することをお勧めします。

セルフマネージド型ノードグループはクラスターのコントロールプレーンのバージョンとともに自動的に更新されますか?

いいえ。セルフマネージド型ノードグループにはアカウント内の Amazon EC2 インスタンスが含まれています。これらのインスタンスはお客様または代わりに Amazon EKS がコントロールプレーンのバージョンを更新しても、自動的にはアップグレードされません。セルフマネージド型ノードグループは更新が必要であることをコンソールに表示しません。更新が必要なノードを確認するにはクラスターの [概要] タブにある [ノード] のリストからノードを選択して、そこにイン

ストールされている kubelet バージョンを表示します。ノードは手動で更新する必要があります。詳細については、「the section called “更新方式”」を参照してください。

Kubernetes では、最大 3 つ前のマイナーバージョンに対して、コントロールプレーンとノード間の互換性がテストされています。例えば、1.34 ノードは1.37 コントロールプレーンによってオーケストレーションされた場合も動作し続けます。しかし、コントロールプレーンの背後で 3 つのマイナーバージョンのノードを使用し続けながら、クラスターを実行することはお勧めしません。詳細については、Kubernetes ドキュメントの「Kubernetes のバージョンおよびバージョンスキューのサポートポリシー」を参照してください。コントロールプレーンとノードでは、同一の Kubernetes バージョンを維持することをお勧めします。

クラスターコントロールプレーンのバージョンに対する自動アップグレードによって、Fargate で実行中の Pod も自動的にアップグレードされますか?

いいえ。Fargate Pod は Kubernetes デプロイなどのレプリケーションコントローラーの一部として実行することを強くお勧めします。その後、すべての Fargate Pod のローリング再起動を実行してください。Fargate Pod の新しいバージョンは、更新されたクラスターのコントロールプレーンのバージョンと同じ kubelet バージョンでデプロイされます。詳細については、Kubernetes ドキュメントの「Deployments」を参照してください。

> **Important**
>
> コントロールプレーンを更新する場合は引き続き Fargate ノードを自分で更新する必要があります。Fargate ノードを更新するには、ノードと対応した Fargate Pod を削除した上で Pod を再デプロイします。新しい Pod は、クラスターと同じ kubelet バージョンでデプロイされます。

ハイブリッドノードではどの Kubernetes バージョンがサポートされていますか?

Amazon EKS Hybrid Nodes は標準および拡張 Kubernetes バージョンのサポートなど、他のノードコンピューティングタイプを持つ Amazon EKS クラスターと同じ Kubernetes バージョンをサポートします。コントロールプレーンのバージョンをアップグレードしても、ハイブリッドノードは自動的にアップグレードされません。また、ハイブリッドノードのアップグレードはお客様の責任となります。詳細については「the section called “ハイブリッドノードをアップグレードする”」を参照してください。

Amazon EKS 延長サポートに関するよくある質問

標準サポートと延長サポートという用語を初めて知りました。これらの用語はどういう意味ですか?

Amazon EKS での Kubernetes バージョンに対する標準サポートは、Amazon EKS で Kubernetes バージョンがリリースされた時点から始まり、リリース日の 14 か月後に終了します。Kubernetes バージョンの延長サポートは、標準サポートの終了後すぐに始まり、その 12 か月後に終了します。たとえば、バージョン `1.23` 向けの Amazon EKS 標準サポートは 2023 年 10 月 11 日に終了します。バージョン `1.23` の延長サポートは 2023 年 10 月 12 日に開始され、2024 年 10 月 11 日に終了しました。

Amazon EKS クラスターの延長サポートを受けるには何をする必要がありますか?

クラスターのアップグレードポリシーを EXTENDED に変更して、クラスターの拡張サポート (「EKS 延長サポート」を参照) を有効にする必要があります。デフォルトではすべての新規および既存のクラスターについて、特に指定がない限り、アップグレードポリシーは EXTENDED に設定されています。クラスターのアップグレードポリシーを表示するには「クラスターアップグレードポリシー」を参照してください。標準サポートは、Amazon EKS で Kubernetes バージョンがリリースされた時点で始まり、リリース日の 14 か月後に終了します。Kubernetes バージョンの延長サポートは、標準サポートの終了後すぐに始まり、その 12 か月後に終了します。

どの Kubernetes バージョンで延長サポートを受けることができますか?

クラスターはそのバージョンの標準サポートが終了してから最大 12 か月間、どのバージョンでも実行できます。これは Amazon EKS では各バージョンが 26 か月間サポートされることを意味します (14 か月間標準サポートと 12 か月間延長サポート)。

延長サポートを利用したくない場合はどうすればいいですか?

延長サポートに自動的に登録されないようにする場合は、クラスターを Amazon EKS 標準サポートの対象となる Kubernetes バージョンにアップグレードします。延長サポートを無効にするには、「EKS 延長サポートを無効にする」を参照してください。注意: 延長サポートを無効にすると、クラスターは標準サポートの終了時に自動的にアップグレードされます。

12 か月間の延長サポートが終了するとどうなりますか?

26 か月のライフサイクル (14 か月の標準サポートと 12 か月の延長サポート) が終了した Kubernetes バージョンで実行されているクラスターは、次のバージョンに自動的にアップグレードされます。自動アップグレードはKubernetes コントロールプレーンのみが対象です。EKS 自動モード ノードがある場合、ノードは自動的に更新される場合があります。セルフマネージドノードと EKS マネージドノードグループは、以前のバージョンのままになります。

延長サポート終了日には終了の対象となっているバージョンで新しい Amazon EKS クラスターを作成できなくなります。サポート終了日を過ぎると、既存のコントロールプレーンは後の段階的なデプロイプロセスを通じて Amazon EKS によりサポートされている最も初期のバージョンに自動的に更新されます。コントロールプレーンの自動更新後はクラスターアドオンと Amazon EC2 ノードを手動で更新してください。詳細については「the section called “Kubernetes バージョンを更新する”」を参照してください。

延長サポート終了後にコントロールプレーンが自動的に更新されるのは正確にはいつですか?

Amazon EKS での具体的なスケジュールは決まっていません。自動更新は延長サポート終了日以降に任意のタイミングで実行される可能性があります。更新前には通知は届きません。Amazon EKS の自動更新プロセスに頼ることなく、事前にコントロールプレーンを更新することをお勧めします。詳細については、「the section called “Kubernetes バージョンを更新する”」を参照してください。

コントロールプレーンを 1 つの Kubernetes バージョンに恒久的に維持することはできますか?

いいえ。AWS ではクラウドのセキュリティを最優先事項ととらえています。特定の時点 (通常は 1 年) を過ぎると、Kubernetes コミュニティは共通脆弱性識別子 (CVE) パッチのリリースを停止し、サポートされていないバージョンの CVE 提出を非推奨とします。つまり、Kubernetes の古いバージョンに固有の脆弱性がある場合、報告すらされない可能性があります。クラスターは脆弱性の発生時にも通知および修復オプションなしで公開され続けることになります。このため、Amazon EKS では延長サポートが終了したバージョンでコントロールプレーンを維持することはできません。

延長サポートを受けるには追加料金がかかりますか?

はい。延長サポートで稼働している Amazon EKS クラスターに対して追加料金が発生します。料金の詳細についてはAWS ブログの「Kubernetes バージョンの Amazon EKS 延長サポートの料金」または「料金ページ」を参照してください。

延長サポートには何が含まれますか?

延長サポートの Amazon EKS クラスターは、Kubernetes コントロールプレーンのセキュリティパッチを継続的に受け取ります。また、Amazon EKS は延長サポート対象バージョンに対する Amazon VPC CNI、`kube-proxy`、CoreDNS アドオンのパッチをリリースします。また、Amazon EKS は Amazon Linux、Bottlerocket、および Windows 向けに AWS が公開した Amazon EKS に最適化された AMI のパッチと、それらのバージョンの Amazon EKS Fargate ノードのパッチもリリースします。延長サポートに該当するすべてのクラスターは引き続き AWS からテクニカルサポートを受けることができます。

延長サポートでは Kubernetes 以外のコンポーネントに適用できるパッチに制限はありますか?

延長サポートは AWS から Kubernetes 固有のすべてのコンポーネントを対象としていますが、常にサポートされるのは Amazon Linux、Bottlerocket、および Windows 向けに AWS が公開した Amazon EKS の最適化された AMI のみです。つまり、延長サポートを利用している間はAmazon EKS に最適化された AMI にはより新しいコンポーネント (OS やカーネルなど) が搭載される場合があります。たとえば、Amazon Linux 2 のライフサイクルが 2025 年に終了すると、Amazon EKS に最適化された Amazon Linux AMI はより新しい Amazon Linux OS を使用して構築されるようになります。Amazon EKS は、このようなサポートライフサイクルの重要な相違点を Kubernetes バージョンごとに発表し、文書化します。

延長サポートのバージョンを使用して新しいクラスターを作成できますか?

はい。

標準サポートの Kubernetes バージョンのリリースノートを確認する

> **Tip**
> 今後開催予定の Amazon EKS ワークショップに登録してください。

このトピックでは、標準サポートで注意すべき重要な変更点を Kubernetes バージョンごとに説明します。アップグレードするときはクラスターの古いバージョンと新しいバージョン間で発生した変更を注意深く確認してください。

Kubernetes 1.37

Amazon EKS で Kubernetes `1.37` が利用可能になりました。Kubernetes`1.37` の詳細については、公式リリースのお知らせを参照してください。

> **Important**
>
> - SELinux マウントラベリングがデフォルトで有効化 (GA): Kubernetes 1.37 では、`SELinuxMount` 機能がデフォルトで有効になっています。ポッドで `seLinuxOptions` が設定され、かつそのボリュームの CSI ドライバーで `seLinuxMount: true` が設定されている場合、kubelet はすべてのファイルのラベルを付け直すのではなく、`mount -o context=<label>` でボリュームをマウントします。1

つのマウントには 1 つの SELinux ラベルしか適用できないため、同じノード上の別のポッドが異なる SELinux ラベル、異なる `seLinuxChangePolicy`、または異なる特権レベルで同じボリュームを既に使用している場合、ポッドは `ContainerCreating` 状態のままとなります。SELinux が有効になっていないノードは影響を受けません。

- 必要なアクション: アップグレードする前に、`kubectl get csidriver -o custom-columns=NAME:.metadata.name,SELINUXMOUNT:.spec.seLinuxMount` を実行し、`seLinuxMount: true` が設定されているドライバーを特定してください。これらのドライバーを介したボリュームについては、`seLinuxOptions` が設定され、かつボリュームを共有しているポッドを特定します。ボリュームを共有するポッドは、同じ `seLinuxOptions.level` と `seLinuxChangePolicy` を使用する必要があります。特権ポッドと非特権ポッド間でボリュームを共有する場合は、非特権ポッドに `spec.securityContext.seLinuxChangePolicy: Recursive` を設定します。`Recursive` は以前の再ラベル付け動作を維持する設定であり、ボリュームを共有するすべてのポッドに適用する必要があります。詳細については、「SELinux Volume Label Changes goes GA (and likely implications in v1.37)」および KEP-1710 を参照してください。

- EKS Auto Mode 統合のデフォルト設定: Kubernetes 1.37 以降、`consolidationPolicy` を指定していない新しい EKS Auto Mode の NodePool では、デフォルト設定が `WhenEmptyOrUnderutilized` ではなく `Balanced` になります。通常、`Balanced` 設定では、ほぼ同じコストでポッドのエビクションが少なくなります。組み込みの `general-purpose` および `system` NodePool でも `Balanced` が使用されており、これらの設定を変更することはできません。詳細については、EKS Auto Mode リリースノートに関するページを参照してください。

  - 必要なアクション: 特定の統合動作を維持するには、独自の NodePool に `consolidationPolicy` を明示的に設定します。ワークロードで以前の動作が必要な場合は、`consolidationPolicy: WhenEmptyOrUnderutilized` が設定されている NodePool をターゲットにします。

- 動的リソース割り当て機能 (DRA) のデバイステイントと許容範囲 (安定版): DRA ドライバーとクラスター管理者は GPU などのデバイスにテイントを付与できるため、スケジューラはテイントを許容しないポッドにデバイスを割り当てません。EKS で DRA を使用するには DRA ドライバーのインストールが必要ですが、EKS はこのドライバーをインストールしません。NVIDIA DRA ドライバーは、Karpenter 静的キャパシティ、EKS マネージド型ノードグループ、およびセルフマネージドノードでサポートされています。EKS Auto Mode ではサポートされていません。EKS Auto

Mode または Karpenter 動的キャパシティプロビジョニングでは、NVIDIA デバイスプラグインを使用します。

  - `DeviceTaintRule` を作成するときは、必ず `spec.deviceSelector` を設定します。すべてのデバイスをターゲットにするには、`deviceSelector: {}` を使用します。詳細については、「Amazon EKS で NVIDIA DRA ドライバーまたはデバイスプラグインを使用する」を参照してください。

- Horizontal Pod Autoscaler の設定可能な許容範囲 (安定版): クラスター全体のデフォルト値である 10% のみに依存するのではなく、スケールアップとスケールダウンのそれぞれについて HPA ごとに許容範囲を設定できます。
- Horizontal Pod Autoscaler を使用したゼロへのスケール (ベータ版): `HPAScaleToZero` 機能ゲートは、Kubernetes 1.37 ではデフォルトで有効になっています。オブジェクトまたは外部メトリクスに基づいてスケールする HPA では、`minReplicas: 0` を設定することで、アイドル時にポッドをゼロにスケールし、需要が戻ったときにスケールアップすることができます。オブジェクトおよび外部メトリクスにはメトリクスアダプターが必要ですが、EKS はこのアダプターをインストールしません。
- Kubelet 設定の変更: 非推奨の cAdvisor フラグ (`--housekeeping-interval` を除く) が設定されている場合、kubelet は起動しなくなり、`container_cpu_load_average_10s`、`container_cpu_load_d_average_10s`、および `container_tasks_state` メトリクスがドロップされ、`eventRecordQPS: 0` は制限なしとして扱われます。以前は、`eventRecordQPS: 0` により 1 秒あたり 5 イベントの制限が適用されていました。EKS 最適化 AMI では、非推奨の cAdvisor フラグまたは `eventRecordQPS` が設定されていないため、kubelet の起動とレート制限に変更はありません。削除されたメトリクスは、どの AMI を使用しているかにかかわらず、それらをスクレイピングするすべてのユーザーに影響します。
  - 必要なアクション: カスタム AMI または追加の kubelet 引数を使用する場合は、非推奨の cAdvisor フラグを削除します。また、`eventRecordQPS: 0` を設定している場合は `5` に変更して、以前のレート制限を維持します。削除されたメトリクスを使用するダッシュボードまたはアラートを更新します。詳細については、「kubelet configuration (v1beta1) reference」を参照してください。
- Kubelet が起動時に設定をログ記録: kubelet は起動時に有効な設定をすべてログに記録するようになりました。信頼できるユーザーのみがノードログを読み込めるようにしてください。詳細については、「kubernetes/kubernetes#139837」を参照してください。
- Amazon EC2 G7 インスタンス用の NVIDIA ドライバー 595: G7 インスタンスには NVIDIA ドライバー 595 が必要です。EKS 最適化 Amazon Linux 2023 NVIDIA AMI には、サポートされ

ているすべてのバージョンでドライバー 595 が含まれています。Bottlerocket NVIDIA AMI には、Kubernetes 1.37 以降でドライバー 595 が含まれています。

Kubernetes 1.37 の変更ログの詳細については、「https://github.com/kubernetes/kubernetes/blob/master/CHANGELOG/CHANGELOG-1.37.md」を参照してください。

Kubernetes 1.36

Amazon EKS で Kubernetes 1.36 が利用可能になりました。Kubernetes1.36 の詳細については、公式リリースのお知らせを参照してください。

Important

- gitRepo ボリュームの削除: gitRepo ボリュームタイプは Kubernetes 1.36 では完全に無効になっており、再度有効にすることはできません。Kubernetes API は引き続き gitRepo ボリュームを持つポッドを受け入れますが、kubelet はそれらの実行を拒否し、エラーを返します。
  - 必要なアクション: 1.36 にアップグレードする前に、init コンテナまたは git-sync サイドカーコンテナに移行します。詳細については、KEP-5040 を参照してください。
- SELinux ボリュームラベル付けの変更 (GA): Kubernetes 1.36 では、再帰的なファイル再ラベル付けの代わりに mount -o context が使用され、すべてのボリュームに対して、より高速な SELinux ボリュームラベル付けがデフォルトで適用されます。同じノード上の特権ポッドと非特権ポッド間、または異なる SELinux ラベルを持つポッド間でボリュームを共有すると、問題が発生する可能性があります。SELinuxMount でポッドを保持できる関連する動作 ContainerCreating がデフォルトで有効になっているのは、Kubernetes 1.36 ではなく Kubernetes 1.37 です。アップグレードする前に、Kubernetes 1.37 のセクションを確認してください。
  - 必要なアクション: アップグレードする前に、SELinux が有効なノードを実行しているクラスターを監査し、ボリュームを共有するポッドで seLinuxChangePolicy フィールドと SELinux ボリュームラベルが正しく設定されていることを確認してください。詳細については、「SELinux Volume Label Changes goes GA (and likely implications in v1.37)」を参照してください。
- 厳格な IP/CIDR 検証がデフォルトで有効: 組み込み API タイプで StrictIPCIDRValidation 機能ゲートがデフォルトで有効になりました。API フィールドは、先頭に余分な 0 が付いた IP 値または CIDR 値 (10.0.0.5 の代わりに 010.000.000.005 など)、またはあいまいなセマンティクスが付いた CIDR 値

(192.168.0.5/24 の代わりに 192.168.0.0/24 など) を受け入れなくなりました。既存の保存済みオブジェクトは検証のラッチによって保持されますが、新しい作成と更新は拒否されます。これはカスタムリソースタイプには適用されません。

- 必要なアクション: マニフェスト、Helm チャート、および先頭のゼロまたは非正規 CIDR 表記を含む IP アドレスの自動化を確認します。アップグレードする前に正規形式を使用するように更新します。詳細については、KEP-4858 を参照してください。

- ユーザー名前空間 (安定): ユーザー名前空間は、コンテナのルートユーザーをホスト上の非特権ユーザーにマッピングすることで多層防御を提供し、コンテナブレークアウトによってノードに対する管理権限が付与されないようにします。
  - 詳細については、Kubernetes ブログの「User Namespaces in Kubernetes are finally GA」を参照してください。
- リソースヘルスステータス (ベータ): ポッドステータスのデバイスごとのヘルスをレポートします。これにより、オペレーターはクラッシュループがアプリケーションの問題ではなく、異常または不明なデバイスステータスによるものかどうかを判断できます。デバイスプラグインと動的リソース割り当ての両方で動作します。
  - 詳細については、KEP-4680 を参照してください。
- 動的リソース割り当て機能 (ベータ): いくつかの DRA 機能がデフォルトで有効になりました。具体的には、GPU などのハードウェアをより詳細に共有するための Partitionable Devices と Consumable Capacity、およびスケジューリング前にデバイスの準備状況チェックを徹底するための Device Binding Conditions です。
  - 詳細については、Kubernetes ブログの「Kubernetes v1.36: More Drivers, New Features, and the Next Era of DRA」を参照してください。
- 非推奨の通知 — サービス externalIPs: Kubernetes 1.36 では、サービス `.spec` の `externalIPs` フィールドは非推奨です。このフィールドを使用するサービスを作成または更新すると、非推奨の警告が表示されます。Kubernetes 1.43 では完全な削除が予定されています。`externalIPs` を使用するお客様は、LoadBalancer Services、NodePort、または Gateway API に移行する必要があります。詳細については、KEP-5707 を参照してください。

Kubernetes `1.36` の変更ログの詳細については、「https://github.com/kubernetes/kubernetes/blob/master/CHANGELOG/CHANGELOG-1.36.md」を参照してください。

Kubernetes 1.35

Amazon EKS で Kubernetes `1.35` が利用可能になりました。Kubernetes`1.35` の詳細については、公式リリースのお知らせを参照してください。

Important

- Cgroup v1 サポートの削除: Kubernetes 1.35 では cgroup v1 サポートが廃止されました。つまり、今後 cgroup v1 を使用するノードではデフォルトで kubelet の起動が拒否されます。
  - AL2023: AL2023 はデフォルトで cgroup v2 を使用し、Kubernetes のアップストリーム動作と連動します。
    - 必要なアクション: cgroup v1 を使用するように AL2023 を手動で設定していたお客様は、cgroups v2 に移行するか、もしくは kubelet 設定で `failCgroupV1: false` を手動で設定する必要があります。
  - Bottlerocket: Bottlerocket 1.35 はデフォルトで cgroup v2 を使用しますが、kubelet 設定では `failCgroupV1: false` を設定し、下位互換性を維持してください。
  - Fargate: Fargate は引き続き cgroup v1 を使用します。
- Containerd 1.x サポート終了予定: Kubernetes 1.35 は containerd 1.x をサポートする最後のリリースです。次の Kubernetes バージョンにアップグレードする前に、containerd 2.0 以降に切り替える必要があります。
- 2026 年 3 月、アップストリーム Kubernetes プロジェクトは、多くの Kubernetes 環境において重要なインフラストラクチャコンポーネントであった Ingress NGINX を廃止します。
  - 必要なアクション: EKS のお客様は、自環境が Ingress NGINX に依存しているかどうかを評価し、Gateway API やサードパーティーの Ingress コントローラーなどの代替手段への移行の計画を開始する必要があります。これは、廃止後にはバグ修正、セキュリティパッチ、更新などのリリースが提供されなくなるためです。既存のデプロイは引き続き機能しますが、廃止後も Ingress NGINX を使い続けると、セキュリティリスクに対して脆弱になります。これは、利用可能な代替手段はどれも直接的な完全互換品ではなく、別途プランニングやエンジニアリングが必要になるためです。この Kubernetes の発表の詳細については、Kubernetes 運営およびセキュリティ対応委員会の公式ステートメント「Ingress NGINX retirement」を参照してください。

- ポッドリソースのインプレースアップグレード (安定版): ポッドリソースのインプレースアップグレードにより、ユーザーはポッドやコンテナを再起動せずに CPU リソースやメモリリソースを調整できます。以前は、このような変更にはポッドの再作成が必要であり、特にステートフルアプリケーションやバッチアプリケーションでは、ワークロードが中断される可能性がありました。新しいインプレース機能により、よりスムーズで中断のない垂直スケーリングが可能になり、効率が向上し、開発が簡素化されます。
  - 詳細については、Kubernetes ブログの「Kubernetes 1.35: In-Place Pod Resize graduates to Stable」を参照してください。
- PreferSameNode トラフィック配分 (安定版): サービスの `trafficDistribution`フィールドが更新され、トラフィックルーティングをより明示的に制御できるようになりました。新しいオプションとして `PreferSameNode` が導入されたことで、ローカルノードが利用可能な場合はローカルノード上のエンドポイントを常に優先し、それ以外の場合にリモートエンドポイントにフォールバックするようサービスを設定できるようになりました。この変更により、API で現在のノード内のトラフィックをより明確に優先化できるようになります。
- StatefulSet MaxUnavailable (ベータ): この機能は、`maxUnavailable` (3% または 10%) を設定することで並列ポッド更新を有効にします。これにより、データベースクラスターなどのステートフルアプリケーションは、順次 1 つずつの更新に比べて最大 60% 高速に更新できるようになり、メンテナンスウィンドウが大幅に短縮されます。
- Windows Server 2025 のサポート: EKS 1.35 では、Windows Server 2025のサポートを追加します。
- Kubelet フラグの削除: `--pod-infra-container-image` フラグが kubelet から削除されました。カスタム AMI ユーザーは、1.35 にアップグレードする前に、kubelet 設定からこのフラグを削除する必要があります。
- 非推奨の通知 – IPVS モード: kube-proxy の IPVS モードは非推奨となり、将来の Kubernetes リリースでは削除される予定です。nftables モードに移行します。詳細については、「Running kube-proxy in nftables Mode」を参照してください。

Kubernetes `1.35` の変更ログの詳細については、「https://github.com/kubernetes/kubernetes/blob/master/CHANGELOG/CHANGELOG-1.35.md」を参照してください。

Kubernetes 1.34

Amazon EKS で Kubernetes `1.34` が利用可能になりました。Kubernetes`1.34` の詳細については、公式リリースのお知らせを参照してください。

Important

- Containerd は、バージョン 1.34 のリリース時に 2.1 に更新されました。
  - アップグレード後に問題が発生した場合は、containerd 2.1 リリースノートを確認してください。
- AWS では、EKS で最適化された Amazon Linux 2 AMI for Kubernetes 1.34 をリリースしていません。
  - AWS では、Amazon Linux 2023 への移行を推奨しています。the section called “AL2023 にアップグレードする” では、その方法を説明しています。
  - 詳細については、「the section called “Amazon Linux 2 AMI の廃止”」を参照してください。
- AppArmor は Kubernetes 1.34 で廃止されました。
  - seccomp やポッドセキュリティ標準などの代替コンテナセキュリティソリューションに移行することをお勧めします。
- VolumeAttributesClass (VAC) は Kubernetes 1.34 で GA に移行し、ベータ API (`storage.k8s.io/v1beta1`) から安定した API (`storage.k8s.io/v1`) に移行します。
  - (ECR Gallery の CSI コンポーネントから提供される) AWS マネージドサイドカーコンテナと共に EBS CSI ドライバーを使用している場合、ボリュームの変更は EKS 1.31 ~ 1.33 クラスターでも引き続きシームレスに動作します。AWS は、EKS 1.33 の標準サポート終了日 (2026 年 7 月 29 日) まで、ベータ版 VAC API をサポートするようにサイドカーにパッチを適用します。
  - CSI サイドカーコンテナを自己管理している場合、VAC 機能を維持するために、1.34 より前のクラスターでは古いサイドカーのバージョンに固定する必要がある場合があります。
  - GA の VolumeAttributesClass 機能 (変更のロールバックなど) を使用するには、EKS 1.34 以降にアップグレードします。
- サービスアカウントトークンの外部 JWT サイナーがベータに昇格しました。外部サイナーを使用する場合、「service-account-extend-token-expiration」フラグは完全に尊重されなくなります。API サーバーは、希望される拡張期間 (1 年) と外部サイナーの制限時間 (24 時間) 間の最小有効期限を適用します。
  - Kubernetes によって自動的にマウントおよびローテーションされるバインドされたサービスアカウントトークンを使用することをお勧めします。

- 動的リソース割り当て (DRA) コア API (GA): 動的リソース割り当ては安定段階にあり、標準化された割り当てインターフェイスを通じて GPU などの特殊なハードウェアを効率的に管理できるようになりました。これにより、ハードウェアアクセラレーターのリソース管理が簡素化され、特殊なリソースの使用率が向上します。
- Kubelet 向けの投影 ServiceAccount トークン (ベータ): この機能強化では、長期間有効なシークレットの代わりに短期間有効な認証情報を使用してコンテナイメージをプルすることで、セキュリティが向上します。これにより、認証情報が公開されるリスクが軽減され、クラスター全体のセキュリティ体制が強化されます。
- ポッドレベルのリソースリクエストと制限 (ベータ): この機能では、マルチコンテナポッドに対して共有リソースプールを許可することでリソース管理が簡素化されます。これにより、複数のコンテナを持つ複雑なアプリケーションに対して、より効率的なリソース割り当てと使用率が可能になります。
- 変更可能な CSI ノード割り当て可能数 (ベータ): `MutableCSINodeAllocatableCount` 機能ゲートは EKS 1.34 でデフォルトで有効になっており、CSINode の最大アタッチ可能ボリューム数属性を変更可能にし、CSI ドライバーレベルでユーザー設定に基づいて動的に更新するメカニズムを導入します。これらの更新は、定期的な間隔または障害検出によってトリガーされ、ノードで報告されたアタッチメント容量と実際のアタッチメント容量の不一致に対処することで、ステートフルポッドスケジューリングの信頼性を高めることができます。
  - 詳細については、Kubernetes ブログの「Kubernetes v1.34: Mutable CSI Node Allocatable Count」を参照してください。
- 非推奨のお知らせ - cgroup ドライバー設定: 手動の cgroup ドライバー設定は、自動検出を優先するため非推奨となります。
  - 顧客への影響: 現在、kubelet 設定で `--cgroup-driver` フラグを手動で設定している場合は、この設定を削除する準備が必要です。
  - 必要なアクション: 将来の Kubernetes リリースで機能が削除される前に、ノードのブートストラップスクリプトとカスタム AMI 設定を更新し、手動の cgroup ドライバー設定を削除する計画を立てます。
  - 詳細については、cgroup ドライバーのドキュメントを参照してください。

Kubernetes `1.34` の変更ログの詳細については、「https://github.com/kubernetes/kubernetes/blob/master/CHANGELOG/CHANGELOG-1.34.md」を参照してください。

延長サポートの Kubernetes バージョンに関するリリースノートを確認する

Amazon EKS は、アップストリームでサポートされているよりも長く Kubernetes バージョンをサポートしており、Amazon EKS でリリースされてから 14 か月間は Kubernetes マイナーバージョンの標準サポートと、さらに 12 か月間のサポート (バージョンごとに合計 26 か月) は Kubernetes マイナーバージョンの拡張サポートを提供します。

このトピックでは延長サポートにおいて各 Kubernetes バージョンで注意すべき重要な変更点を説明します。アップグレードするときはクラスターの古いバージョンと新しいバージョン間で発生した変更を注意深く確認してください。

> **Note**
>
> 標準サポートのバージョンから拡張サポートのバージョンにロールバックすると、そのクラスターの拡張サポート料金が再開されます。

Kubernetes 1.33

Amazon EKS で Kubernetes `1.33` が利用可能になりました。Kubernetes`1.33` の詳細については、公式リリースのお知らせを参照してください。

> **Important**
>
> - 動的リソース割り当てで Kubernetes の beta API が有効になっています。
>   - この beta API によって、GPU などのリソースを必要とするワークロードのスケジューリングとモニタリングのエクスペリエンスが向上します。
>   - beta API は Kubernetes コミュニティによって定義され、今後の Kubernetes バージョンで変更される可能性があります。
>   - Kubernetes ドキュメントの「機能ステージ」を注意深く確認し、beta API の使用による影響を理解してください。
> - AWS では、EKS で最適化された Amazon Linux 2 AMI for Kubernetes 1.33 をリリースしていません。
>   - AWS では、Amazon Linux 2023 への移行を推奨しています。the section called “AL2023 にアップグレードする” では、その方法を説明しています。

    - 詳細については、「the section called “Amazon Linux 2 AMI の廃止”」を参照してください。

- インプレースポッドリソースのサイズ変更 (ベータ): Kubernetes 1.33 では、インプレースリソースのサイズ変更がベータに昇格しました。これにより、既存のポッドの CPU とメモリリソースを再起動なしで動的に更新でき、ステートフルなワークロードをダウンタイムゼロで垂直スケーリングできるようになります。
- サイドカーコンテナが安定版に: サイドカーコンテナが安定版に昇格しました。これらは `restartPolicy: Always` を備えた特別な init コンテナとして実装され、アプリケーションコンテナより前に起動し、ポッドのライフサイクル全体にわたって実行され、運用状態のシグナリング用のプローブをサポートします。
    - 詳細については、Kubernetes ドキュメントの「サイドカーコンテナ」を参照してください。
- エンドポイント API の廃止: エンドポイント API が正式に廃止され、アクセス時に警告が返されるようになりました。代わりに EndpointSlices API を使用するようにワークロードとスクリプトを移行してください。これにより、デュアルスタックネットワーキングなどの最新機能がサポートされ、サービスごとに複数の EndpointSlices が処理されます。
    - 詳細については、Kubernetes ブログの「Kubernetes v1.33: Continuing the transition from Endpoints to EndpointSlice」を参照してください。
- Elastic Fabric Adapter のサポート: Amazon EKS クラスターのデフォルトのセキュリティグループで、Elastic Fabric Adapter (EFA) トラフィックをサポートするようになりました。デフォルトのセキュリティグループには、同じセキュリティグループの宛先に対して EFA トラフィックを許可する新しいアウトバウンドルールがあります。これにより、クラスター内での EFA トラフィックが可能になります。
    - 詳細については、「Amazon Elastic Compute Cloud ユーザーガイド」の「Amazon EC2 の AI/ML および HPC ワークロード用の Elastic Fabric Adapter」を参照してください。

Kubernetes `1.33` の変更ログの詳細については、「https://github.com/kubernetes/kubernetes/blob/master/CHANGELOG/CHANGELOG-1.33.md」を参照してください。

Kubernetes 1.32

Amazon EKS で Kubernetes `1.32` が利用可能になりました。Kubernetes`1.32` の詳細については、公式リリースのお知らせを参照してください。

Important

- FlowSchema および PriorityLevelConfiguration の `flowcontrol.apiserver.k8s.io/v1beta3` API バージョンは、バージョン `1.32` で削除されました。これらの APIs を使用している場合はアップグレードする前に、サポートされている最新バージョンを使用するように設定を更新する必要があります。
- ServiceAccount `metadata.annotations[kubernetes.io/enforce-mountable-secrets]` は、バージョン `1.32` で廃止され、今後の Kubernetes マイナーバージョンリリースで削除されます。マウントされたシークレットへのアクセスを分離するには、別の名前空間を使用することをお勧めします。
- Kubernetes バージョン `1.32` は、Amazon EKS が Amazon Linux 2 (AL2) AMI をリリースする最後のバージョンです。バージョン `1.33` 以降、Amazon EKS は引き続き Amazon Linux 2023 (AL2023) および Bottlerocket ベースの AMI をリリースします。

- Memory Manager 機能は、Kubernetes バージョン `1.32` で一般提供 (GA) ステータスに移行しました。この機能強化により、コンテナ化されたアプリケーションに対してより効率的で予測可能なメモリ割り当てが可能になり、特定のメモリ要件を持つワークロードに特に役立ちます。
- StatefulSets によって作成された PersistentVolumeClaims (PVC) に自動クリーンアップ機能が含まれるようになりました。PVCs が不要になった場合はStatefulSet 更新およびノードメンテナンスオペレーション中にデータの永続性を維持しながら、自動的に削除されます。この機能により、ストレージ管理が簡素化され、クラスター内で孤立した PVC が発生するのを防ぐことができます。
- カスタム リソース フィールド セレクター機能が導入され、デベロッパーはカスタム リソースにフィールド セレクターを追加できるようになりました。この機能は、組み込みの Kubernetes オブジェクトで使用できるものと同じフィルタリング機能をカスタム リソースに提供し、より正確で効率的なリソース フィルタリングを可能にし、より優れた API 設計プラクティスを促進します。

Kubernetes `1.32` の変更ログの詳細については、「https://github.com/kubernetes/kubernetes/blob/master/CHANGELOG/CHANGELOG-1.32.md」を参照してください。

匿名認証の変更

Amazon EKS 以降`1.32`、匿名認証は次の API サーバーのヘルスチェックエンドポイントに制限されています。

- `/healthz`

- /livez
- /readyz

system:unauthenticated ユーザーを使用した他のエンドポイントへのリクエストはHTTP 401 Unauthorized レスポンスを受け取ります。このセキュリティ強化はRBAC ポリシーの設定ミスが原因で発生する可能性のある意図しないクラスターアクセスを防ぐのに役立ちます。

> **Note**
>
> public-info-viewer RBAC ロールは上記のヘルスチェックエンドポイントに引き続き適用されます。

Amazon Linux 2 AMI の廃止

Kubernetes バージョン 1.32 は Amazon EKS が AL2 AMI をリリースした最後のバージョンです。バージョン 1.33 以降、Amazon EKS は引き続き AL2023 および Bottlerocket ベースの AMI をリリースします。詳細については、the section called “AL2 AMI の廃止” を参照してください。

Kubernetes 1.31

Amazon EKS で Kubernetes 1.31 が利用可能になりました。Kubernetes1.31 の詳細については、公式リリースのお知らせを参照してください。

> **Important**
>
> - 2017 年以降廃止された kubelet フラグ --keep-terminated-pod-volumes は、バージョン 1.31 リリースの一部として削除されました。この変更は終了したポッドボリュームが kubelet によって処理される方法に影響します。ノード設定でこのフラグを使用している場合はアップグレードする前にブートストラップスクリプトと起動テンプレートを更新してこのフラグを削除する必要があります。

- ベータ VolumeAttributesClass 機能ゲートと API リソースは、Amazon EKS バージョン 1.31 で有効になっています。この機能を使用すると、クラスターオペレーターは Amazon EBS CSI ドライバーなど、互換性のある CSI ドライバーによって管理される永続ボリューム (PVs) のミュータブルプロパティを変更できます。この機能を活用するにはお使いの CSI ドライバー

が VolumeAttributesClass 機能をサポートしていることを確認します (Amazon EBS CSI ドライバーの場合はバージョン 1.35.0 以降にアップグレードして自動的に機能を有効にします)。VolumeAttributesClass オブジェクトを作成して、ボリュームタイプやスループットなどの必要なボリューム属性を定義し、永続的ボリュームクレーム (PVC) に関連付けることができます。詳細についてはKubernetes の公式ドキュメント とご使用の CSI ドライバーのドキュメントを参照してください。

  - Amazon EBS CSI ドライバーの詳細については、「the section called “Amazon EBS”」を参照してください。

- AppArmor の Kubernetes サポートは安定するように段階的に拡張され、一般公開されました。この機能を使用すると、コンテナの securityContext で appArmorProfile.type フィールドを設定することで、AppArmor を使用してコンテナを保護できます。バージョン 1.30 より前の Kubernetes では、AppArmor は注釈によって制御されていました。バージョン 1.30 以降では、フィールドを使用して制御されます。この機能を活用するには注釈から移行し、appArmorProfile.type フィールドを使用してワークロードに互換性があることを確認することをお勧めします。
- PersistentVolume の最終フェーズ移行時間機能は、安定するように段階的に拡張され、Kubernetes バージョン 1.31 で一般公開されました。この機能ではPersistentVolumeStatus に新しいフィールド .status.lastTransitionTime が導入され、PersistentVolume が最後に別のフェーズに移行したときのタイムスタンプが提供されます。この機能強化により、特にボリュームのライフサイクルを理解することが重要なシナリオではPersistentVolumes の追跡と管理が向上します。

Kubernetes 1.31 の変更ログの詳細については、「https://github.com/kubernetes/kubernetes/blob/master/CHANGELOG/CHANGELOG-1.31.md」を参照してください。

現在のクラスターサポート期間を表示する

AWS コンソールの [クラスターサポート期間] セクションには、クラスターが現在、標準サポートであるか、または延長サポートであるかが表示されます。クラスターサポート期間が [延長サポート] である場合、EKS 延長サポートについて課金されています。

標準サポートと延長サポートの詳細については、「Understand the Kubernetes version lifecycle on EKS」を参照してください。

1. AWS コンソールの EKS セクションの [クラスター] ページに移動します。コンソールが、確認するクラスターと同じ AWS リージョンに設定されていることを確認します。

2. [サポート期間] 列を確認します。値が [...まで標準サポート] である場合、現在、延長サポートについて課金されていません。標準サポート期間内です。値が [延長サポート...] である場合、このクラスターは現在、延長サポートについて課金されています。

> **Note**
>
> [サポート期間] は AWS API または CLI では取得できません。

現在のクラスターアップグレードポリシーを表示する

[クラスターのアップグレードポリシー] は、標準サポート期間を過ぎたクラスターがどうなるのかを決定します。アップグレードポリシーが EXTENDED である場合、クラスターは自動的にアップグレードされず、延長サポートに入ります。アップグレードポリシーが STANDARD である場合、自動的にアップグレードされます。

Kubernetes バージョンポリシーの Amazon EKS コントロールでは、EKS クラスターの標準サポート終了時の動作を選択できます。これらのコントロールを使用すると、Kubernetes バージョンの標準サポートの終了時に、どのクラスターが延長サポートに入るか、およびどのクラスターを自動的にアップグレードするかを決定できます。

マイナーバージョンはリリース後最初の 14 か月間は Amazon EKS の標準サポート対象となります。標準サポート終了日を過ぎたバージョンは次の 12 か月間の延長サポートに入ります。延長サポートでは、クラスター時間あたりの追加料金を支払えば、特定の Kubernetes バージョンを延長して使用することができます。EKS クラスターの延長サポートを有効または無効にすることができます。延長サポートを無効にすると、AWS は標準サポートの終了時にクラスターを次のバージョンに自動的にアップグレードします。延長サポートを有効にすると、一定期間、追加料金で最新バージョンを維持できます。延長サポートを使用している場合でも、Kubernetes クラスターを定期的にアップグレードする計画を立ててください。

`supportType` プロパティを使用して、新規クラスターと既存のクラスターの両方のバージョンポリシーを設定できます。バージョンサポートポリシーの設定に使用できるオプションは 2 つあります:

- **`STANDARD`** – EKS クラスターは、標準サポートの終了時に自動アップグレードの対象となります。この設定では延長サポート料金は発生しませんが、EKS クラスターは標準サポートで次にサポートされている Kubernetes バージョンに自動的にアップグレードされます。
- **`EXTENDED`** – Kubernetes バージョンが標準サポートの終了に達すると、EKS クラスターは延長サポートに入ります。この設定では、延長サポート料金が発生します。クラスターをサポートされ

ている標準 Kubernetes バージョンにアップグレードして、延長サポート料金の発生を停止できます。延長サポートで実行されているクラスターは、延長サポートの終了時に自動アップグレードの対象となります。

延長サポートは、新しいクラスターと既存のクラスターのためにデフォルトで有効になっています。AWS マネジメントコンソール で、または AWS CLI を使用して、クラスターのために延長サポートが有効になっているかどうかを確認できます。

> **Important**
>
> クラスターを現在の Kubernetes バージョンのままにして延長サポート期間を利用する場合は、クラスターの現在の Kubernetes バージョンに対する標準サポート期間が終了する前に延長サポートアップグレードポリシーを有効にすることを強くお勧めします。延長サポートは標準サポート期間の終了後に有効にできます。ただし、クラスターで自動アップグレードが既に始まっていると、変更が有効になる保証はありません。

クラスターが延長サポート期間に入った後は、延長サポートを無効にすることはできません。バージョンサポートポリシーの設定を変更するには、クラスターが標準サポートの Kubernetes バージョンで実行されている必要があります。

クラスターアップグレードポリシーを表示する (AWS コンソール)

1. AWS コンソールの EKS セクションの [クラスター] ページに移動します。コンソールが、確認するクラスターと同じ AWS リージョンに設定されていることを確認します。
2. [アップグレードポリシー] 列を確認します。値が [標準サポート] である場合、クラスターは延長サポートに入りません。値が [延長サポート] である場合、クラスターは延長サポートに入ります。

クラスターアップグレードポリシーを表示する (AWS CLI)

1. AWS CLI がインストールされ、ログインしていることを確認します。AWS CLI を更新およびインストールする方法について説明します。
2. EKS クラスターの名前を決定します。CLI を EKS クラスターと同じ AWS リージョンに設定します。
3. 次のコマンドを実行します。

```
aws eks describe-cluster \
--name <cluster-name> \
--query "cluster.upgradePolicy.supportType"
```

4. 値が [STANDARD] である場合、クラスターは延長サポートに入りません。値が [EXTENDED] である場合、クラスターは延長サポートに入ります。

EKS 延長サポートを有効にして、Kubernetes バージョンアップグレードの計画にさらなる柔軟性をもたらす

このトピックは、EKS クラスターのアップグレードポリシーを設定して延長サポートを有効にする方法について説明します。EKS クラスターのアップグレードポリシーは、クラスターが標準サポート期間の終了に達したときにどうなるのかを決定します。クラスターアップグレードポリシーで延長サポートが有効になっている場合、標準サポート期間が終了するのと同時に、延長サポート期間に入ります。クラスターは、標準サポート期間の終了時に自動的にアップグレードされません。

実際に延長サポート期間中のクラスターでは、より高額なコストが発生します。延長サポートを有効にするように設定されたアップグレードポリシーがクラスターにあるが、標準サポート期間中であれば、標準コストが発生します。

AWS コンソールでクラスターを作成すると、延長サポートを無効にするようにアップグレードポリシーが設定されます。別の方法でクラスターを作成すると、延長サポートを有効にするようにアップグレードポリシーが設定されます。例えば、AWS API で作成されたクラスターでは、拡張サポートが有効になっています。

アップグレードポリシーの詳細については、「クラスターアップグレードポリシー」を参照してください。

> **Important**
>
> クラスターを現在の Kubernetes バージョンのままにして延長サポート期間を利用する場合は、クラスターの現在の Kubernetes バージョンに対する標準サポート期間が終了する前に延長サポートアップグレードポリシーを有効にすることを強くお勧めします。延長サポートは標準サポート期間の終了後に有効にできます。ただし、クラスターで自動アップグレードが既に始まっていると、変更が有効になる保証はありません。
> クラスターが延長サポート期間に入った後は、延長サポートを無効にすることはできません。延長サポートは、標準サポートで Kubernetes バージョンを実行しているクラスターでのみ無効にできます。

延長サポートを有効にしていない場合は、標準サポート期間の終了後にクラスターが自動的にアップグレードされます。

EKS 延長サポートを有効にする (AWS コンソール)

1. AWS コンソールで EKS クラスターに移動します。[クラスター情報] ページの [概要] タブを選択します。
2. [Kubernetes バージョン設定] セクションで、[管理] を選択します。
3. [延長サポート] を選択し、[変更を保存] します。

EKS 延長サポートを有効にする (AWS CLI)

1. AWS CLI がインストールされ、ログインしていることを確認します。AWS CLI を更新およびインストールする方法について説明します。
2. EKS クラスターの名前を決定します。
3. 次のコマンドを実行します。

```
aws eks update-cluster-config \
--name <cluster-name> \
--upgrade-policy supportType=EXTENDED
```

EKS 延長サポートを無効にすることにより、クラスターのコストが増えないようにする

このトピックでは、EKS クラスターのアップグレードポリシーを設定して延長サポートを無効にする方法について説明します。EKS クラスターのアップグレードポリシーは、クラスターが標準サポート期間の終了に達したときにどうなるのかを決定します。クラスターアップグレードポリシーで延長サポートが無効になっている場合、自動的に次の Kubernetes バージョンにアップグレードされます。

アップグレードポリシーの詳細については、「クラスターアップグレードポリシー」を参照してください。

> **Important**
>
> クラスターが延長サポート期間に入った後は、延長サポートを無効にすることはできません。クラスターの延長サポートは、標準サポートでのみ無効にできます。
> AWS では、クラスターを標準サポート期間内のバージョンにアップグレードすることをお勧めしています。

EKS 延長サポートを無効にする (AWS コンソール)

1. AWS コンソールで EKS クラスターに移動します。[クラスター情報] ページの [概要] タブを選択します。
2. [Kubernetes バージョン設定] セクションで、[管理] を選択します。
3. [標準サポート] を選択し、[変更を保存] します。

EKS 延長サポートを無効にする (AWS CLI)

1. AWS CLI がインストールされ、ログインしていることを確認します。AWS CLI を更新およびインストールする方法について説明します。
2. EKS クラスターの名前を決定します。
3. 次のコマンドを実行します。

```
aws eks update-cluster-config \
--name <cluster-name> \
--upgrade-policy supportType=STANDARD
```

各 Kubernetes バージョンの Amazon EKS プラットフォームバージョンを表示する

Amazon EKS プラットフォームのバージョンは、Amazon EKS クラスターコントロールプレーンの各機能 (有効な Kubernetes API サーバーのフラグや、現在の Kubernetes パッチバージョンなど) を表しています。各 Kubernetes マイナーバージョンには、Amazon EKS プラットフォームのバージョンが 1 つ以上関連付けられています。別の Kubernetes マイナーバージョンのプラットフォームバージョンは独立しています。クラスターの現在のプラットフォームバージョンを取得する には AWS CLI または AWS マネジメントコンソール を使用します。AWS Outposts にローカルクラスターがあ

る場合、このトピックの代わりに「the section called “EKS プラットフォームのバージョン”」を参照してください。

新しい Kubernetes マイナーバージョン (1.37 など) が Amazon EKS で使用可能になると、その Kubernetes マイナーバージョンでの Amazon EKS プラットフォームのバージョン付けが `eks.1` から開始されます。また、Amazon EKS では、新しいプラットフォームバージョンを定期的にリリースすることで、新しい Kubernetes コントロールプレーン設定を有効化し、セキュリティ修正プログラムを提供します。

Amazon EKS プラットフォームのバージョンで、新しいマイナーバージョンが使用可能になった場合は、以下が行われます。

- Amazon EKS プラットフォームのバージョン番号がインクリメントされます (`eks.<n+1>`)。
- Amazon EKS は、既存のすべてのクラスターを、対応する Kubernetes マイナーバージョン用の最新の Amazon EKS プラットフォームバージョンに自動的にアップグレードします。既存の Amazon EKS プラットフォームバージョンの自動アップグレードは、段階的にロールアウトされます。ロールアウトプロセスには時間がかかる場合があります。最新の Amazon EKS プラットフォームバージョンの機能がすぐに必要な場合は新しい Amazon EKS クラスターを作成する必要があります。

  クラスターが現在のプラットフォームバージョンより 2 バージョン以上遅れている場合、Amazon EKS がクラスターを自動的に更新できなかった可能性があります。この原因の詳細については、「the section called “Amazon EKS プラットフォームのバージョンは現在のプラットフォーム バージョンより 2 つ以上遅れています”」を参照してください。
- Amazon EKS は対応するパッチバージョンを適用して新しいノード AMI を発行する可能性があります。ただし、すべてのパッチバージョンは、特定の Kubernetes マイナーバージョンにおいて、EKS コントロールプレーンとノード AMI との間で互換性を持ちます。

新しい Amazon EKS プラットフォームバージョンにより、重大な変更が発生したり、サービスが中断されたりすることはありません。

クラスターは常に、指定された Kubernetes バージョン用に入手可能な、最新の Amazon EKS プラットフォームバージョン (`eks.<n>`) で作成されます。クラスターを新しい Kubernetes マイナーバージョンに更新した場合、そのクラスターは、更新した Kubernetes マイナーバージョンでの、現在の Amazon EKS プラットフォームバージョンを受け取ります。

最新および間近にリリースされた、Amazon EKS プラットフォームのバージョンを以下の表に示します。

> **Note**
>
> AWS は2024 年 6 月に公開された一部のプラットフォームバージョンを最近無効にしました。該当のプラットフォームバージョンには安定性の問題がありました。アクションは不要です。

この特定のドキュメントページへのすべてのソースファイルの変更通知を受け取るには、RSS リーダーを使用して次の URL をサブスクライブできます。

```
https://github.com/awsdocs/amazon-eks-user-guide/commits/mainline/latest/ug/versioning/platform-versions.adoc.atom
```

Kubernetes バージョン 1.37

次の受付コントローラーはすべての 1.37 プラットフォームバージョンで有効です: NodeRestriction、ExtendedResourceToleration、NamespaceLifecycle、LimitRanger、Serv

Kubernetes バージョン	EKS プラットフォームのバージョン	リリースノート	リリース日
1.37.0	eks.4	EKS 用 Kubernetes バージョン 1.37 の初回リリース。詳細については、「the section called “Kubernetes 1.37”」を参照してください。	2026 年 10 月 1 日

Kubernetes バージョン 1.36

次の受付コントローラーはすべての 1.36 プラットフォームバージョンで有効です: NodeRestriction、ExtendedResourceToleration、NamespaceLifecycle、LimitRanger、Serv

Kubernetes バージョン	EKS プラットフォームのバージョン	リリースノート	リリース日
1.36.4	eks.13	セキュリティの修正および機能強化が行われた新しいプラットフォームバージョン。	2026 年 9 月 16 日
1.36.3	eks.12	セキュリティの修正および機能強化が行われた新しいプラットフォームバージョン。	2026 年 9 月 8 日
1.36.3	eks.11	セキュリティの修正および機能強化が行われた新しいプラットフォームバージョン。	2026 年 9 月 8 日
1.36.2	eks.10	セキュリティの修正および機能強化が行われた新しいプラットフォームバージョン。	2026 年 8 月 17 日
1.36.2	eks.9	セキュリティの修正および機能強化が行われた新しいプラットフォームバージョン。	2026 年 8 月 7 日
1.36.2	eks.8	セキュリティの修正および機能強化が行われた新しいプラッ	2026 年 7 月 23 日

Kubernetes バージョン	EKS プラットフォームのバージョン	リリースノート	リリース日
		トフォームバージョン。	
1.36.2	eks.7	セキュリティの修正および機能強化が行われた新しいプラットフォームバージョン。	2026 年 7 月 14 日
1.36.2	eks.6	セキュリティの修正と機能強化を含む新しいプラットフォームバージョン。1.36 eks.5 は内部で破棄され、今後リリースされることはありません。	2026 年 6 月 22 日
1.36.1	eks.4	セキュリティの修正および機能強化が行われた新しいプラットフォームバージョン。	2026 年 6 月 3 日
1.36.1	eks.3	EKS 用 Kubernetes バージョン 1.36 の初回リリース。詳細については、「the section called “Kubernetes 1.36”」を参照してください。	2026 年 6 月 2 日

Kubernetes バージョン 1.35

次の受付コントローラーはすべての 1.35 プラットフォームバージョンで有効です:
NodeRestriction、ExtendedResourceToleration、NamespaceLifecycle、LimitRanger、Serv

Kubernetes バージョン	EKS プラットフォームのバージョン	リリースノート	リリース日
1.35.8	eks.23	セキュリティの修正および機能強化が行われた新しいプラットフォームバージョン。	2026 年 9 月 16 日
1.35.7	eks.22	セキュリティの修正および機能強化が行われた新しいプラットフォームバージョン。	2026 年 9 月 8 日
1.35.6	eks.21	セキュリティの修正および機能強化が行われた新しいプラットフォームバージョン。	2026 年 8 月 17 日
1.35.6	eks.20	セキュリティの修正および機能強化が行われた新しいプラットフォームバージョン。	2026 年 8 月 7 日
1.35.6	eks.19	セキュリティの修正および機能強化が行われた新しいプラットフォームバージョン。	2026 年 7 月 23 日

Kubernetes バージョン	EKS プラットフォームのバージョン	リリースノート	リリース日
1.35.6	eks.18	セキュリティの修正および機能強化が行われた新しいプラットフォームバージョン。	2026 年 7 月 14 日
1.35.6	eks.17	セキュリティの修正と機能強化を含む新しいプラットフォームバージョン。1.35 eks.16 は内部で破棄され、今後リリースされることはありません。	2026 年 6 月 22 日
1.35.5	eks.15	セキュリティの修正および機能強化が行われた新しいプラットフォームバージョン。	2026 年 6 月 3 日
1.35.5	eks.14	セキュリティの修正および機能強化が行われた新しいプラットフォームバージョン。	2026 年 5 月 26 日

Kubernetes バージョン	EKS プラットフォームのバージョン	リリースノート	リリース日
1.35.4	eks.13	セキュリティの修正と機能強化を含む新しいプラットフォームバージョン。1.35 eks.12 は内部で破棄され、今後リリースされることはありません。	2026 年 5 月 19 日
1.35.4	eks.11	セキュリティの修正と機能強化を含む新しいプラットフォームバージョン。1.35 eks.10 は内部で破棄され、今後リリースされることはありません。	2026 年 5 月 7 日
1.35.2	eks.9	セキュリティの修正および機能強化が行われた新しいプラットフォームバージョン。	2026 年 4 月 3 日
1.35.2	eks.8	セキュリティの修正と機能強化を含む新しいプラットフォームバージョン。 1.35 eks.6 と 1.35 eks.7 は内部で破棄され、リリースされることはありません。	2026 年 3 月 18 日

Kubernetes バージョン	EKS プラットフォームのバージョン	リリースノート	リリース日
1.35.0	eks.5	セキュリティの修正および機能強化が行われた新しいプラットフォームバージョン。	2026 年 3 月 5 日
1.35.0	eks.4	セキュリティの修正および機能強化が行われた新しいプラットフォームバージョン。	2026 年 2 月 24 日
1.35.0	eks.3	EKS 用 Kubernetes バージョン 1.35 の初回リリース。詳細については、the section called “Kubernetes 1.35” を参照してください。	2026 年 1 月 27 日

Kubernetes バージョン 1.34

次の受付コントローラーはすべての 1.34 プラットフォームバージョンで有効です: NodeRestriction、ExtendedResourceToleration、NamespaceLifecycle、LimitRanger、Serv

Kubernetes バージョン	EKS プラットフォームのバージョン	リリースノート	リリース日
1.34.11	eks.33	セキュリティの修正および機能強化が行われた新しいプラットフォームバージョン。	2026 年 9 月 16 日

Kubernetes バージョン	EKS プラットフォームのバージョン	リリースノート	リリース日
1.34.10	eks.32	セキュリティの修正および機能強化が行われた新しいプラットフォームバージョン。	2026 年 9 月 8 日
1.34.9	eks.31	セキュリティの修正および機能強化が行われた新しいプラットフォームバージョン。	2026 年 8 月 17 日
1.34.9	eks.30	セキュリティの修正および機能強化が行われた新しいプラットフォームバージョン。	2026 年 8 月 7 日
1.34.9	eks.29	セキュリティの修正および機能強化が行われた新しいプラットフォームバージョン。	2026 年 7 月 23 日
1.34.9	eks.28	セキュリティの修正および機能強化が行われた新しいプラットフォームバージョン。	2026 年 7 月 14 日

Kubernetes バージョン	EKS プラットフォームのバージョン	リリースノート	リリース日
1.34.9	eks.27	セキュリティの修正と機能強化を含む新しいプラットフォームバージョン。1.34 eks.26 は内部で破棄され、今後リリースされることはありません。	2026 年 6 月 22 日
1.34.8	eks.25	セキュリティの修正および機能強化が行われた新しいプラットフォームバージョン。	2026 年 6 月 3 日
1.34.8	eks.24	セキュリティの修正および機能強化が行われた新しいプラットフォームバージョン。	2026 年 5 月 26 日
1.34.7	eks.23	セキュリティの修正と機能強化を含む新しいプラットフォームバージョン。1.34 eks.22 は内部で破棄され、今後リリースされることはありません。	2026 年 5 月 19 日

Kubernetes バージョン	EKS プラットフォームのバージョン	リリースノート	リリース日
1.34.7	eks.21	セキュリティの修正と機能強化を含む新しいプラットフォームバージョン。1.34 eks.20 は内部で破棄され、今後リリースされることはありません。	2026 年 5 月 7 日
1.34.4	eks.19	セキュリティの修正および機能強化が行われた新しいプラットフォームバージョン。	2026 年 4 月 3 日
1.34.4	eks.18	セキュリティの修正と機能強化を含む新しいプラットフォームバージョン。 1.34 eks.16 と 1.34 eks.17 は内部で破棄され、リリースされることはありません。	2026 年 3 月 18 日
1.34.3	eks.15	セキュリティの修正および機能強化が行われた新しいプラットフォームバージョン。	2026 年 3 月 5 日

Kubernetes バージョン	EKS プラットフォームのバージョン	リリースノート	リリース日
1.34.3	eks.14	セキュリティの修正および機能強化が行われた新しいプラットフォームバージョン。	2026 年 2 月 24 日
1.34.3	eks.13	セキュリティの修正および機能強化が行われた新しいプラットフォームバージョン。	2026 年 1 月 28 日
1.34.3	eks.12	セキュリティの修正と機能強化を含む新しいプラットフォームバージョン。1.34 eks.11 は内部で破棄され、今後リリースされることはありません。	2026 年 1 月 16 日
1.34.2	eks.10	セキュリティの修正および機能強化が行われた新しいプラットフォームバージョン。	2025 年 12 月 19 日
1.34.1	eks.9	セキュリティの修正および機能強化が行われた新しいプラットフォームバージョン。	2025 年 11 月 18 日

Kubernetes バージョン	EKS プラットフォームのバージョン	リリースノート	リリース日
1.34.1	eks.8	セキュリティの修正および機能強化が行われた新しいプラットフォームバージョン。	2025 年 11 月 17 日
1.34.1	eks.7	セキュリティの修正および機能強化が行われた新しいプラットフォームバージョン。	2025 年 11 月 6 日
1.34.1	eks.6	セキュリティの修正および機能強化が行われた新しいプラットフォームバージョン。	2025 年 10 月 30 日
1.34.1	eks.5	セキュリティの修正および機能強化が行われた新しいプラットフォームバージョン。	2025 年 10 月 15 日
1.34.1	eks.4	EKS 用 Kubernetes バージョン 1.34 の初回リリース。詳細については、the section called “Kubernetes 1.34” を参照してください。	2025 年 10 月 2 日

Kubernetes バージョン **1.33**

次の受付コントローラーはすべての 1.33 プラットフォームバージョンで有効です:
NodeRestriction、ExtendedResourceToleration、NamespaceLifecycle、LimitRanger、Serv

Kubernetes バージョン	EKS プラットフォームのバージョン	リリースノート	リリース日
1.33.13	eks.47	セキュリティの修正および機能強化が行われた新しいプラットフォームバージョン。	2026 年 9 月 16 日
1.33.13	eks.46	セキュリティの修正および機能強化が行われた新しいプラットフォームバージョン。	2026 年 9 月 8 日
1.33.13	eks.45	セキュリティの修正および機能強化が行われた新しいプラットフォームバージョン。	2026 年 8 月 17 日
1.33.13	eks.44	セキュリティの修正および機能強化が行われた新しいプラットフォームバージョン。	2026 年 8 月 7 日
1.33.13	eks.43	セキュリティの修正および機能強化が行われた新しいプラットフォームバージョン。	2026 年 7 月 23 日

Kubernetes バージョン	EKS プラットフォームのバージョン	リリースノート	リリース日
1.33.13	eks.42	セキュリティの修正および機能強化が行われた新しいプラットフォームバージョン。	2026 年 7 月 14 日
1.33.13	eks.41	セキュリティの修正と機能強化を含む新しいプラットフォームバージョン。1.33 eks.40 は内部で破棄され、今後リリースされることはありません。	2026 年 6 月 22 日
1.33.12	eks.39	セキュリティの修正および機能強化が行われた新しいプラットフォームバージョン。	2026 年 6 月 3 日
1.33.12	eks.38	セキュリティの修正および機能強化が行われた新しいプラットフォームバージョン。	2026 年 5 月 26 日

Kubernetes バージョン	EKS プラットフォームのバージョン	リリースノート	リリース日
1.33.11	eks.37	セキュリティの修正と機能強化を含む新しいプラットフォームバージョン。1.33 eks.36 は内部で破棄され、今後リリースされることはありません。	2026 年 5 月 19 日
1.33.11	eks.35	セキュリティの修正と機能強化を含む新しいプラットフォームバージョン。1.33 eks.34 は内部で破棄され、今後リリースされることはありません。	2026 年 5 月 7 日
1.33.8	eks.33	セキュリティの修正および機能強化が行われた新しいプラットフォームバージョン。	2026 年 4 月 3 日
1.33.8	eks.32	セキュリティの修正と機能強化を含む新しいプラットフォームバージョン。 1.33 eks.30 と 1.33 eks.31 は内部で破棄され、リリースされることはありません。	2026 年 3 月 18 日

Kubernetes バージョン	EKS プラットフォームのバージョン	リリースノート	リリース日
1.33.7	eks.29	セキュリティの修正および機能強化が行われた新しいプラットフォームバージョン。	2026 年 3 月 5 日
1.33.7	eks.28	セキュリティの修正および機能強化が行われた新しいプラットフォームバージョン。	2026 年 2 月 24 日
1.33.7	eks.27	セキュリティの修正および機能強化が行われた新しいプラットフォームバージョン。	2026 年 1 月 28 日
1.33.7	eks.26	セキュリティの修正と機能強化を含む新しいプラットフォームバージョン。1.33 eks.25 は内部で破棄され、今後リリースされることはありません。	2026 年 1 月 16 日
1.33.6	eks.24	セキュリティの修正および機能強化が行われた新しいプラットフォームバージョン。	2025 年 12 月 19 日

Kubernetes バージョン	EKS プラットフォームのバージョン	リリースノート	リリース日
1.33.5	eks.23	セキュリティの修正および機能強化が行われた新しいプラットフォームバージョン。	2025 年 11 月 18 日
1.33.5	eks.22	セキュリティの修正および機能強化が行われた新しいプラットフォームバージョン。	2025 年 11 月 17 日
1.33.5	eks.21	セキュリティの修正および機能強化が行われた新しいプラットフォームバージョン。	2025 年 11 月 6 日
1.33.5	eks.20	セキュリティの修正および機能強化が行われた新しいプラットフォームバージョン。	2025 年 10 月 30 日
1.33.5	eks.19	セキュリティの修正および機能強化が行われた新しいプラットフォームバージョン。	2025 年 10 月 28 日
1.33.5	eks.18	セキュリティの修正および機能強化が行われた新しいプラットフォームバージョン。	2025 年 10 月 15 日

Kubernetes バージョン	EKS プラットフォームのバージョン	リリースノート	リリース日
1.33.5	eks.17	セキュリティの修正および機能強化が行われた新しいプラットフォームバージョン。	2025 年 10 月 15 日
1.33.5	eks.16	セキュリティの修正および機能強化が行われた新しいプラットフォームバージョン。	2025 年 10 月 2 日
1.33.4	eks.15	セキュリティの修正および機能強化が行われた新しいプラットフォームバージョン。	2025 年 9 月 29 日
1.33.4	eks.14	セキュリティの修正および機能強化が行われた新しいプラットフォームバージョン。	2025 年 9 月 29 日
1.33.4	eks.13	セキュリティの修正および機能強化が行われた新しいプラットフォームバージョン。	2025 年 9 月 16 日
1.33.4	eks.12	セキュリティの修正および機能強化が行われた新しいプラットフォームバージョン。	2025 年 9 月 5 日

Kubernetes バージョン	EKS プラットフォームのバージョン	リリースノート	リリース日
1.33.3	eks.11	セキュリティの修正および機能強化が行われた新しいプラットフォームバージョン。	2025年08月25日
1.33.3	eks.10	セキュリティの修正および機能強化が行われた新しいプラットフォームバージョン。	2025 年 8 月 17 日
1.33.2	eks.9	セキュリティの修正と機能強化を含む新しいプラットフォームバージョン。 1.33 eks.7 と 1.33 eks.8 は内部で破棄され、リリースされることはありません。	2025 年 7 月 30 日
1.33.1	eks.6	セキュリティの修正および機能強化が行われた新しいプラットフォームバージョン。	2025 年 6 月 26 日
1.33.1	eks.5	セキュリティの修正および機能強化が行われた新しいプラットフォームバージョン。	2025 年 6 月 11 日

Kubernetes バージョン	EKS プラットフォームのバージョン	リリースノート	リリース日
1.33.1	eks.4	EKS 用 Kubernetes バージョン 1.33 の初回リリース。詳細については、the section called “Kubernetes 1.33” を参照してください。	2025 年 5 月 28 日

Kubernetes バージョン 1.32

次の受付コントローラーはすべての 1.32 プラットフォームバージョンで有効です: NodeRestriction、ExtendedResourceToleration、NamespaceLifecycle、LimitRanger、Serv

Kubernetes バージョン	EKS プラットフォームのバージョン	リリースノート	リリース日
1.32.13	eks.54	セキュリティの修正および機能強化が行われた新しいプラットフォームバージョン。	2026 年 9 月 16 日
1.32.13	eks.53	セキュリティの修正および機能強化が行われた新しいプラットフォームバージョン。	2026 年 9 月 8 日
1.32.13	eks.52	セキュリティの修正および機能強化が行われた新しいプラットフォームバージョン。	2026 年 8 月 17 日

Kubernetes バージョン	EKS プラットフォームのバージョン	リリースノート	リリース日
1.32.13	eks.51	セキュリティの修正および機能強化が行われた新しいプラットフォームバージョン。	2026 年 8 月 7 日
1.32.13	eks.50	セキュリティの修正および機能強化が行われた新しいプラットフォームバージョン。	2026 年 7 月 23 日
1.32.13	eks.49	セキュリティの修正および機能強化が行われた新しいプラットフォームバージョン。	2026 年 7 月 14 日
1.32.13	eks.48	セキュリティの修正と機能強化を含む新しいプラットフォームバージョン。1.32 eks.47 は内部で破棄され、今後リリースされることはありません。	2026 年 6 月 22 日
1.32.13	eks.46	セキュリティの修正および機能強化が行われた新しいプラットフォームバージョン。	2026 年 6 月 3 日

Kubernetes バージョン	EKS プラットフォームのバージョン	リリースノート	リリース日
1.32.13	eks.45	セキュリティの修正および機能強化が行われた新しいプラットフォームバージョン。	2026 年 5 月 26 日
1.32.13	eks.44	セキュリティの修正と機能強化を含む新しいプラットフォームバージョン。1.32 eks.43 は内部で破棄され、今後リリースされることはありません。	2026 年 5 月 19 日
1.32.13	eks.42	セキュリティの修正と機能強化を含む新しいプラットフォームバージョン。1.32 eks.41 は内部で破棄され、今後リリースされることはありません。	2026 年 5 月 7 日
1.32.12	eks.40	セキュリティの修正および機能強化が行われた新しいプラットフォームバージョン。	2026 年 4 月 3 日

Kubernetes バージョン	EKS プラットフォームのバージョン	リリースノート	リリース日
1.32.12	eks.39	セキュリティの修正と機能強化を含む新しいプラットフォームバージョン。 1.32 eks.37 と 1.32 eks.38 は内部で破棄され、リリースされることはありません。	2026 年 3 月 18 日
1.32.11	eks.36	セキュリティの修正および機能強化が行われた新しいプラットフォームバージョン。	2026 年 3 月 5 日
1.32.11	eks.35	セキュリティの修正および機能強化が行われた新しいプラットフォームバージョン。	2026 年 2 月 24 日
1.32.11	eks.34	セキュリティの修正および機能強化が行われた新しいプラットフォームバージョン。	2026 年 1 月 28 日

Kubernetes バージョン	EKS プラットフォームのバージョン	リリースノート	リリース日
1.32.11	eks.33	セキュリティの修正と機能強化を含む新しいプラットフォームバージョン。1.32 eks.32 は内部で破棄され、今後リリースされることはありません。	2026 年 1 月 16 日
1.32.10	eks.31	セキュリティの修正および機能強化が行われた新しいプラットフォームバージョン。	2025 年 12 月 19 日
1.32.9	eks.30	セキュリティの修正および機能強化が行われた新しいプラットフォームバージョン。	2025 年 11 月 18 日
1.32.9	eks.29	セキュリティの修正および機能強化が行われた新しいプラットフォームバージョン。	2025 年 11 月 17 日
1.32.9	eks.28	セキュリティの修正および機能強化が行われた新しいプラットフォームバージョン。	2025 年 11 月 6 日

Kubernetes バージョン	EKS プラットフォームのバージョン	リリースノート	リリース日
1.32.9	eks.27	セキュリティの修正および機能強化が行われた新しいプラットフォームバージョン。	2025 年 10 月 30 日
1.32.9	eks.26	セキュリティの修正および機能強化が行われた新しいプラットフォームバージョン。	2025 年 10 月 28 日
1.32.9	eks.25	セキュリティの修正および機能強化が行われた新しいプラットフォームバージョン。	2025 年 10 月 15 日
1.32.9	eks.24	セキュリティの修正および機能強化が行われた新しいプラットフォームバージョン。	2025 年 10 月 15 日
1.32.9	eks.23	セキュリティの修正および機能強化が行われた新しいプラットフォームバージョン。	2025 年 10 月 2 日
1.32.8	eks.22	セキュリティの修正および機能強化が行われた新しいプラットフォームバージョン。	2025 年 9 月 29 日

Kubernetes バージョン	EKS プラットフォームのバージョン	リリースノート	リリース日
1.32.8	eks.21	セキュリティの修正および機能強化が行われた新しいプラットフォームバージョン。	2025 年 9 月 29 日
1.32.8	eks.20	セキュリティの修正および機能強化が行われた新しいプラットフォームバージョン。	2025 年 9 月 16 日
1.32.8	eks.19	セキュリティの修正および機能強化が行われた新しいプラットフォームバージョン。	2025 年 9 月 5 日
1.32.7	eks.18	セキュリティの修正および機能強化が行われた新しいプラットフォームバージョン。	2025年08月25日
1.32.7	eks.17	セキュリティの修正および機能強化が行われた新しいプラットフォームバージョン。	2025 年 8 月 17 日

Kubernetes バージョン	EKS プラットフォームのバージョン	リリースノート	リリース日
1.32.6	eks.16	セキュリティの修正と機能強化を含む新しいプラットフォームバージョン。 1.32 eks.14 と 1.32 eks.15 は内部で破棄され、リリースされることはありません。	2025 年 7 月 30 日
1.32.5	eks.13	セキュリティの修正および機能強化が行われた新しいプラットフォームバージョン。	2025 年 6 月 26 日
1.32.5	eks.12	セキュリティの修正および機能強化が行われた新しいプラットフォームバージョン。	2025 年 6 月 11 日
1.32.5	eks.11	セキュリティの修正および機能強化が行われた新しいプラットフォームバージョン。	2025 年 5 月 30 日
1.32.3	eks.10	セキュリティの修正および機能強化が行われた新しいプラットフォームバージョン。	2025 年 5 月 16 日

Kubernetes バージョン	EKS プラットフォームのバージョン	リリースノート	リリース日
1.32.3	eks.9	セキュリティの修正および機能強化が行われた新しいプラットフォームバージョン。	2025 年 4 月 29 日
1.32.3	eks.8	セキュリティの修正および機能強化が行われた新しいプラットフォームバージョン。	2025 年 4 月 18 日
1.32.3	eks.7	セキュリティの修正および機能強化が行われた新しいプラットフォームバージョン。	2025 年 4 月 18 日
1.32.3	eks.6	セキュリティの修正および機能強化が行われた新しいプラットフォームバージョン。	2025 年 4 月 2 日
1.32.2	eks.5	セキュリティの修正および機能強化が行われた新しいプラットフォームバージョン。	2025 年 3 月 17 日
1.32.2	eks.4	セキュリティの修正および機能強化が行われた新しいプラットフォームバージョン。	2025 年 3 月 4 日

Kubernetes バージョン	EKS プラットフォームのバージョン	リリースノート	リリース日
1.32.1	eks.3	セキュリティの修正および機能強化が行われた新しいプラットフォームバージョン。	2025 年 2 月 24 日
1.32.0	eks.2	EKS 用 Kubernetes バージョン 1.32 の初回リリース。詳細については、the section called “Kubernetes 1.32” を参照してください。	2025 年 1 月

Kubernetes バージョン 1.31

次の受付コントローラーはすべての 1.31 プラットフォームバージョンで有効です: NodeRestriction、ExtendedResourceToleration、NamespaceLifecycle、LimitRanger、Serv

Kubernetes バージョン	EKS プラットフォームのバージョン	リリースノート	リリース日
1.31.14	eks.70	セキュリティの修正および機能強化が行われた新しいプラットフォームバージョン。	2026 年 9 月 16 日
1.31.14	eks.69	セキュリティの修正および機能強化が行われた新しいプラットフォームバージョン。	2026 年 9 月 8 日

Kubernetes バージョン	EKS プラットフォームのバージョン	リリースノート	リリース日
1.31.14	eks.68	セキュリティの修正および機能強化が行われた新しいプラットフォームバージョン。	2026 年 8 月 17 日
1.31.14	eks.67	セキュリティの修正および機能強化が行われた新しいプラットフォームバージョン。	2026 年 8 月 7 日
1.31.14	eks.66	セキュリティの修正および機能強化が行われた新しいプラットフォームバージョン。	2026 年 7 月 23 日
1.31.14	eks.65	セキュリティの修正および機能強化が行われた新しいプラットフォームバージョン。	2026 年 7 月 14 日
1.31.14	eks.64	セキュリティの修正と機能強化を含む新しいプラットフォームバージョン。1.31 eks.63 は内部で破棄され、今後リリースされることはありません。	2026 年 6 月 22 日

Kubernetes バージョン	EKS プラットフォームのバージョン	リリースノート	リリース日
1.31.14	eks.62	セキュリティの修正および機能強化が行われた新しいプラットフォームバージョン。	2026 年 6 月 3 日
1.31.14	eks.61	セキュリティの修正および機能強化が行われた新しいプラットフォームバージョン。	2026 年 5 月 26 日
1.31.14	eks.60	セキュリティの修正と機能強化を含む新しいプラットフォームバージョン。1.31 eks.59 は内部で破棄され、今後リリースされることはありません。	2026 年 5 月 19 日
1.31.14	eks.58	セキュリティの修正と機能強化を含む新しいプラットフォームバージョン。1.31 eks.57 は内部で破棄され、今後リリースされることはありません。	2026 年 5 月 7 日

Kubernetes バージョン	EKS プラットフォームのバージョン	リリースノート	リリース日
1.31.14	eks.56	セキュリティの修正および機能強化が行われた新しいプラットフォームバージョン。	2026 年 4 月 3 日
1.31.14	eks.55	セキュリティの修正と機能強化を含む新しいプラットフォームバージョン。 1.31 eks.53 と 1.31 eks.54 は内部で破棄され、リリースされることはありません。	2026 年 3 月 18 日
1.31.14	eks.52	セキュリティの修正および機能強化が行われた新しいプラットフォームバージョン。	2026 年 3 月 5 日
1.31.14	eks.51	セキュリティの修正および機能強化が行われた新しいプラットフォームバージョン。	2026 年 2 月 24 日
1.31.14	eks.50	セキュリティの修正および機能強化が行われた新しいプラットフォームバージョン。	2026 年 1 月 28 日

Kubernetes バージョン	EKS プラットフォームのバージョン	リリースノート	リリース日
1.31.14	eks.49	セキュリティの修正と機能強化を含む新しいプラットフォームバージョン。1.31 eks.48 は内部で破棄され、今後リリースされることはありません。	2026 年 1 月 16 日
1.31.14	eks.47	セキュリティの修正および機能強化が行われた新しいプラットフォームバージョン。	2025 年 12 月 19 日
1.31.13	eks.46	セキュリティの修正および機能強化が行われた新しいプラットフォームバージョン。	2025 年 11 月 18 日
1.31.13	eks.45	セキュリティの修正および機能強化が行われた新しいプラットフォームバージョン。	2025 年 11 月 17 日
1.31.13	eks.44	セキュリティの修正および機能強化が行われた新しいプラットフォームバージョン。	2025 年 11 月 6 日

Kubernetes バージョン	EKS プラットフォームのバージョン	リリースノート	リリース日
1.31.13	eks.43	セキュリティの修正および機能強化が行われた新しいプラットフォームバージョン。	2025 年 10 月 30 日
1.31.13	eks.42	セキュリティの修正および機能強化が行われた新しいプラットフォームバージョン。	2025 年 10 月 28 日
1.31.13	eks.41	セキュリティの修正および機能強化が行われた新しいプラットフォームバージョン。	2025 年 10 月 15 日
1.31.13	eks.40	セキュリティの修正および機能強化が行われた新しいプラットフォームバージョン。	2025 年 10 月 15 日
1.31.13	eks.39	セキュリティの修正および機能強化が行われた新しいプラットフォームバージョン。	2025 年 10 月 2 日
1.31.12	eks.38	セキュリティの修正および機能強化が行われた新しいプラットフォームバージョン。	2025 年 9 月 29 日

Kubernetes バージョン	EKS プラットフォームのバージョン	リリースノート	リリース日
1.31.12	eks.37	セキュリティの修正および機能強化が行われた新しいプラットフォームバージョン。	2025 年 9 月 29 日
1.31.12	eks.36	セキュリティの修正および機能強化が行われた新しいプラットフォームバージョン。	2025 年 9 月 16 日
1.31.12	eks.35	セキュリティの修正および機能強化が行われた新しいプラットフォームバージョン。	2025 年 9 月 5 日
1.31.11	eks.34	セキュリティの修正および機能強化が行われた新しいプラットフォームバージョン。	2025年08月25日
1.31.11	eks.33	セキュリティの修正および機能強化が行われた新しいプラットフォームバージョン。	2025 年 8 月 17 日

Kubernetes バージョン	EKS プラットフォームのバージョン	リリースノート	リリース日
1.31.10	eks.32	セキュリティの修正と機能強化を含む新しいプラットフォームバージョン。 1.31 eks.30 と 1.31 eks.31 は内部で破棄され、リリースされることはありません。	2025 年 7 月 30 日
1.31.9	eks.29	セキュリティの修正および機能強化が行われた新しいプラットフォームバージョン。	2025 年 6 月 26 日
1.31.9	eks.28	セキュリティの修正および機能強化が行われた新しいプラットフォームバージョン。	2025 年 6 月 11 日
1.31.9	eks.27	セキュリティの修正および機能強化が行われた新しいプラットフォームバージョン。	2025 年 5 月 30 日
1.31.7	eks.26	セキュリティの修正および機能強化が行われた新しいプラットフォームバージョン。	2025 年 5 月 16 日

Kubernetes バージョン	EKS プラットフォームのバージョン	リリースノート	リリース日
1.31.7	eks.25	セキュリティの修正および機能強化が行われた新しいプラットフォームバージョン。	2025 年 4 月 29 日
1.31.7	eks.24	セキュリティの修正および機能強化が行われた新しいプラットフォームバージョン。	2025 年 4 月 18 日
1.31.7	eks.23	セキュリティの修正および機能強化が行われた新しいプラットフォームバージョン。	2025 年 4 月 18 日
1.31.7	eks.22	セキュリティの修正および機能強化が行われた新しいプラットフォームバージョン。	2025 年 4 月 2 日
1.31.6	eks.21	セキュリティの修正および機能強化が行われた新しいプラットフォームバージョン。	2025 年 3 月 17 日
1.31.6	eks.20	セキュリティの修正および機能強化が行われた新しいプラットフォームバージョン。	2025 年 3 月 4 日

Kubernetes バージョン	EKS プラットフォームのバージョン	リリースノート	リリース日
1.31.5	eks.19	セキュリティの修正および機能強化が行われた新しいプラットフォームバージョン。	2025 年 2 月 24 日
1.31.5	eks.18	セキュリティの修正および機能強化が行われた新しいプラットフォームバージョン。	2025 年 2 月 24 日
1.31.4	eks.17	セキュリティの修正および機能強化が行われた新しいプラットフォームバージョン。	2025年1月17日
1.31.2	eks.12	Amazon EKS Hybrid Nodes のサポートとコントロールプレーンのオブザーバビリティの強化機能を備えた新しいプラットフォームバージョン。「the section called “ハイブリッドノード”」と「Amazon EKS enhances performance observability」をそれぞれ参照してください。	2024 年 11 月 15 日

Kubernetes バージョン	EKS プラットフォームのバージョン	リリースノート	リリース日
1.31.1	eks.6	セキュリティの修正および機能強化が行われた新しいプラットフォームバージョン。	2024 年 10 月 21 日
1.31.0	eks.4	EKS 用 Kubernetes バージョン 1.31 の初回リリース。詳細については、the section called “Kubernetes 1.31” を参照してください。	2024 年 9 月 26 日

最新のプラットフォームバージョンを取得する

最新のプラットフォームバージョンを取得する (AWS コンソール)

1. Amazon EKS コンソールを開きます。
2. ナビゲーションペインで [クラスター] を選択してください。
3. クラスターのリストで、プラットフォームバージョンを確認する [クラスター名] を選択してください。
4. [概要] タブを選択してください。
5. [プラットフォームバージョン] は [詳細] セクションで参照できます。

最新のプラットフォームバージョンを取得する (AWS CLI)

1. プラットフォームバージョンを確認するクラスターの名前を特定します。
2. 次のコマンドを実行してください。

```
aws eks describe-cluster --name my-cluster --query cluster.platformVersion
```

出力例は次のとおりです。

```
"eks.10"
```

プラットフォームバージョンを変更する

EKS クラスターのプラットフォームバージョンを変更することはできません。ある Kubernetes バージョンのために新しい Amazon EKS プラットフォームバージョンが使用可能になると、EKS は既存のすべてのクラスターを、対応する Kubernetes バージョンの最新の Amazon EKS プラットフォームバージョンに自動的にアップグレードします。既存の Amazon EKS プラットフォームバージョンの自動アップグレードは段階的にロールアウトされます。AWS コンソールまたは CLI を使用してプラットフォームバージョンを変更することはできません。

Kubernetes バージョンをアップグレードすると、クラスターはその Kubernetes バージョンの最新のプラットフォームバージョンに移行します。

Amazon EKS クラスターとノードに関する問題をトラブルシューティングする

この章では Amazon EKS の使用中に表示される一般的なエラーとその回避方法について説明します。特定の Amazon EKS 領域のトラブルシューティングが必要な場合、別の the section called “トラブルシューティング”、the section called “EKS Connector をトラブルシューティングする”、および「EKS アドオンを使用した ADOT のトラブルシューティング」を参照してください。

その他のトラブルシューティング情報については、AWS re:Post の「Amazon Elastic Kubernetes Service に関するナレッジセンターのコンテンツ」を参照してください。

容量不足

Amazon EKS クラスターを作成しようとするときに次のエラーが表示された場合、指定したいずれかのアベイラビリティーゾーンに、クラスターをサポートするための十分な容量がありません。

```
Cannot create cluster 'example-cluster' because region-1d, the targeted Availability Zone, does not currently have sufficient capacity to support the cluster. Retry and choose from these Availability Zones: region-1a, region-1b, region-1c
```

このエラーメッセージによって返されるアベイラビリティーゾーンでホストされているクラスター VPC 内のサブネットを使用して、クラスターを再作成してください。

クラスターを配置できないアベイラビリティーゾーンがあります。サブネットが属するアベイラビリティーゾーンを、サブネットの要件と考慮事項内のアベイラビリティーゾーンのリストと比較してください。

ノードがクラスターへの参加に失敗します

ノードがクラスターに参加できない一般的な理由はいくつかあります。

- ノードがマネージド型の場合、Amazon EKS はノードグループの作成時に、aws-auth ConfigMap にエントリを追加します。エントリが削除または変更された場合は再度追加する必要があります。詳細についてはターミナルに `eksctl create iamidentitymapping --help` を入力してください。現在の aws-auth ConfigMap エントリを確認するには次のコマンドの *my-cluster* をクラスターの名前に置き換えて、変更したコマンド `eksctl get iamidentitymapping --cluster` *my-cluster* を実行してください。指定するロールの

ARN には / 以外のパスを含めることはできません。例えば、ロールの名前が development/apps/my-role の場合、ロールの ARN を指定するときに my-role に変更する必要があります。ノード IAM ロール ARN (インスタンスプロファイルの ARN ではない) を指定していることを確認してください。

ノードがセルフマネージド型で、ノードの IAM ロールの ARN のアクセスエントリを作成していない場合はマネージド型ノード用にリストされているのと同じコマンドを実行してください。ノード IAM ロールの ARN のアクセスエントリを作成している場合、そのエントリがアクセスエントリで正しく設定されていない可能性があります。aws-auth ConfigMap エントリまたはアクセスエントリのプリンシパル ARN として、ノード IAM ロール ARN (インスタンスプロファイルの ARN ではない) が指定されていることを確認してください。アクセスエントリの詳細については、「the section called “アクセスエントリ”」を参照してください。

- ノードの AWS CloudFormation テンプレートの クラスター名 が、ノードを参加させるクラスターの名前と正確に一致しません。このフィールドに正しくない値を渡すと、ノードの /var/lib/kubelet/kubeconfig ファイルの設定が正しくなくなるため、ノードはクラスターに参加できません。
- ノードはクラスターによって所有されているためタグ付けされていません。ノードには次のタグが適用されている必要があります。*my-cluster* はクラスターの名前に置き換えられます。

キー	値
kubernetes.io/cluster/*my-cluster*	owned

- ノードはパブリック IP アドレスを使用してクラスターにアクセスできない場合があります。パブリックサブネットにデプロイされたノードにパブリック IP アドレスが割り当てられていることを確認してください。割り当てられていない場合は起動後にノードに Elastic IP アドレスを関連付けることができます。詳細については、「Elastic IP アドレスをインスタンスまたはネットワークインターフェイスに関連付ける」を参照してください。デプロイされたインスタンスにパブリック IP アドレスを自動的に割り当てるようにパブリックサブネットが設定されていない場合はその設定を有効にすることをお勧めします。詳細については、「サブネットの IPv4 アドレス指定属性の変更」を参照してください。ノードがプライベートサブネットにデプロイされている場合、サブネットにはパブリック IP アドレスが割り当てられた NAT ゲートウェイへのルートが必要です。
- ノードのデプロイ先 AWS リージョンの AWS STS エンドポイントはアカウントに対して有効になっていません。リージョンを有効にするには「AWS リージョンでの AWS STS のアクティブ化と非アクティブ化」を参照してください。

- ノードにはプライベート DNS エントリがないため、node "" not found エラーを含む kubelet ログが記録されます。ノードが作成される VPC に、DHCP options set の Options として domain-name と domain-name-servers の値のセットがあることを確認してください。デフォルト値は domain-name:<region>.compute.internal および domain-name-servers:AmazonProvidedDNS です。詳細については、Amazon VPC ユーザーガイドの「DHCP オプションセット」を参照してください。
- マネージドノードグループ内のノードが 15 分以内にクラスターに接続しない場合、「NodeCreationFailure」という正常性の問題が出力され、コンソールのステータスが Create failed に設定されます。起動時間が遅い Windows AMI の場合、この問題は高速起動を使用して解決できます。

ワーカーノードがクラスターに参加できない一般的な原因を特定してトラブルシューティングするには、AWSSupport-TroubleshootEKSWorkerNode ランブックを使用できます。詳細については、AWS Systems Manager Automation ランブックリファレンスの「 AWSSupport-TroubleshootEKSWorkerNode 」を参照してください。

許可されていないか、アクセスが拒否されました (kubectl)

kubectl コマンドの実行中に次のいずれかのエラーが発生した場合は、kubectl が Amazon EKS に対して適切に設定されていないか、使用している IAM プリンシパルの認証情報 (ロールまたはユーザー) が、Amazon EKS クラスターで Kubernetes オブジェクトに対して十分なアクセス許可を持つ Kubernetes ユーザー名にマッピングされていません。

- could not get token: AccessDenied: Access denied
- error: You must be logged in to the server (Unauthorized)
- error: the server doesn’t have a resource type "svc"

この原因としては次のいずれかが考えられます。

- クラスターはある IAM プリンシパルの認証情報を使用して作成されており、kubectl は別の IAM プリンシパルの認証情報を使用するよう設定されています。この問題を解決するにはクラスターを作成した認証情報を使用するように kube config ファイルを更新してください。詳細については、「the section called “kubectl を使用してクラスターにアクセスする”」を参照してください。
- クラスターが EKS アクセスエントリを使用して Kubernetes へのアクセスを IAM ユーザーに許可するの前提条件セクションの最小プラットフォーム要件を満たしている場合、IAM プリンシパル

にはアクセスエントリは存在しません。存在する場合、必要な Kubernetes グループ名が定義されていないか、適切なアクセスポリシーが関連付けられていません。詳細については、「the section called “アクセスエントリ”」を参照してください。

- クラスターが EKS アクセス エントリを使用して IAM ユーザーに Kubernetes へのアクセスを許可する の最小プラットフォーム要件を満たしていない場合、aws-auth ConfigMap に IAM プリンシパルのエントリが存在しません。存在しても、Kubernetes Role または ClusterRole にバインドされている、必要なアクセス許可を持つ Kubernetes グループ名にはマップされません。Kubernetes ロールベースの承認 (RBAC) オブジェクトの詳細については、Kubernetes ドキュメントの「RBAC 認可を使用する」を参照してください。現在の aws-auth ConfigMap エントリを確認するには次のコマンドの *my-cluster* をクラスターの名前に置き換えて、変更したコマンド `eksctl get iamidentitymapping --cluster` *my-cluster* を実行してください。IAM プリンシパルの ARN を含むエントリが ConfigMap にない場合はターミナルに `eksctl create iamidentitymapping --help` を入力して作成方法を確認してください。

AWS CLI をインストールして設定する場合は使用する IAM 認証情報を設定できます。詳細については、AWS コマンドラインインターフェイスユーザーガイドの「AWS CLI の設定」を参照してください。クラスター上の Kubernetes オブジェクトにアクセスする IAM ロールを引き受ける場合は、IAM ロールを使用するように kubectl を設定することもできます。詳細については、「the section called “kubectl を使用してクラスターにアクセスする”」を参照してください。

hostname doesn’t match

システムの Python のバージョンは 2.7.9 以降であることが必要です。それ以外の場合は AWS CLI で Amazon EKS を呼び出すと `hostname doesn’t match` エラーが表示されます。詳細については、「Python Requests のよくある質問」の「What are "hostname doesn't match" errors?」(ホスト名の不一致」エラーとは) を参照してください。

getsockopt: no route to host

Docker は Amazon EKS クラスターの 172.17.0.0/16 CIDR 範囲で実行されます。クラスターの VPC サブネットがこの範囲と重ならないようにすることをお勧めします。重なっている場合は以下のエラーが発生します。

```
Error: : error upgrading connection: error dialing backend: dial tcp
 172.17.<nn>.<nn>:10250: getsockopt: no route to host
```

Instances failed to join the Kubernetes cluster

AWS マネジメントコンソール で `Instances failed to join the Kubernetes cluster` エラーが表示された場合、クラスターのプライベートエンドポイントアクセスが有効になっているか、パブリックエンドポイントアクセス用に CIDR ブロックが正しく設定されていることを確認します。詳細については、「the section called “クラスターエンドポイントのアクセス”」を参照してください。

マネージド型ノードグループのエラー

マネージド型ノードグループでハードウェアのヘルスに問題が発生した場合は Amazon EKS によって問題の診断に役立つエラーメッセージが返されます。これらのヘルスチェックは Amazon EC2 ヘルスチェックに基づいているため、ソフトウェアの問題は検出されません。次のリストはエラーコードについての説明です。

アクセス拒否

Amazon EKS または 1 つ以上のマネージド型ノードが、Kubernetes クラスター API サーバーでの認証または認可に失敗しています。共通のレスポンスヘッダーの詳細については、「the section called “マネージド型ノードグループの `AccessDenied` エラーの一般的な原因を修正する”」を参照してください。プライベート Windows AMI では、`Not authorized for images` エラーメッセージと共にこのエラーコードが表示されることもあります。詳細については、「the section called “`Not authorized for images`”」を参照してください。

AmiIdNotFound

起動テンプレートに関連付けられている AMI ID が見つかりませんでした。AMI が存在し、アカウントと共有されていることを確認してください。

AutoScalingGroupNotFound

マネージド型ノードグループに関連付けられている 自動スケーリング グループが見つかりませんでした。同じ設定で 自動スケーリング グループを再作成して復旧できる場合があります。

ClusterUnreachable

Amazon EKS または 1 つ以上のマネージド型ノードが Kubernetes クラスター API サーバーと通信できません。このエラーはネットワークが中断したり、API サーバーがリクエストの処理をタイムアウトしている場合に発生する可能性があります。

Ec2SecurityGroupNotFound

クラスターのクラスターセキュリティグループが見つかりませんでした。クラスターを再作成する必要があります。

Ec2SecurityGroupDeletionFailure

マネージド型ノードグループのリモートアクセスセキュリティグループを削除できませんでした。セキュリティグループから依存関係を削除します。

Ec2LaunchTemplateNotFound

マネージド型ノードグループの Amazon EC2 起動テンプレートが見つかりませんでした。復旧にはノードグループを再作成する必要があります。

Ec2LaunchTemplateVersionMismatch

マネージド型ノードグループの Amazon EC2 起動テンプレートのバージョンが、Amazon EKS が作成したバージョンと一致しません。Amazon EKS が作成したバージョンに戻すことで復旧できる場合があります。

IamInstanceProfileNotFound

マネージド型ノードグループの IAM インスタンスプロファイルが見つかりませんでした。同じ設定でインスタンスプロファイルを再作成して復旧できる場合があります。

IamNodeRoleNotFound

マネージド型ノードグループの IAM ロールが見つかりませんでした。同じ設定で IAM ロールを再作成して復旧できる場合があります。

AsgInstanceLaunchFailures

インスタンスの起動中に 自動スケーリング グループに障害が発生しています。

NodeCreationFailure

起動したインスタンスを Amazon EKS クラスターに登録できません。この障害の一般的な原因はノード IAM ロールのアクセス許可が不十分か、ノードのアウトバウンドインターネットアクセスがないことです。ノードは以下の要件のいずれかを満たしている必要があります。

- パブリック IP アドレスを使用してインターネットにアクセス可能です。ノードが存在するサブネットに関連付けられているセキュリティグループが、通信を許可する必要があります。詳細については、「the section called “サブネットの要件と考慮事項”」および「the section called “セキュリティグループの要件”」を参照してください。

- ノードと VPC は「インターネットアクセスが制限されたプライベートクラスターをデプロイする」に記載される要件を満たしている必要があります。

InstanceLimitExceeded

AWS アカウントが、指定されたインスタンスタイプのインスタンスをこれ以上起動できません。Amazon EC2 のインスタンス制限の引き上げをリクエストして復旧できる場合があります。

InsufficientFreeAddresses

マネージド型ノードグループに関連付けられている 1 つ以上のサブネットに、新しいノードに使用できる十分な IP アドレスがありません。

InternalFailure

これらのエラーは通常、Amazon EKS サーバー側の問題が原因で発生します。

マネージド型ノードグループの **AccessDenied** エラーの一般的な原因を修正する

マネージド型ノードグループで操作を実行する際に AccessDenied エラーが発生する最も一般的な原因は eks:node-manager、ClusterRole または ClusterRoleBinding がないことです。Amazon EKS はマネージド型ノードグループでのオンボーディングの一環として、クラスター内にこれらのリソースを設定します。これらのリソースはノードグループの管理に必要です。

ClusterRole は時間の経過とともに変化する可能性がありますが、次の例のようになります。

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: eks:node-manager
rules:
- apiGroups:
  - ''
  resources:
  - pods
  verbs:
  - get
  - list
  - watch
  - delete
- apiGroups:
  - ''
```

```
  resources:
  - nodes
  verbs:
  - get
  - list
  - watch
  - patch
- apiGroups:
  - ''
  resources:
  - pods/eviction
  verbs:
  - create
```

ClusterRoleBinding は時間の経過とともに変化する可能性がありますが、次の例のようになります。

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: eks:node-manager
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: eks:node-manager
subjects:
- apiGroup: rbac.authorization.k8s.io
  kind: User
  name: eks:node-manager
```

eks:node-manager ClusterRole が存在することを確認します。

```
kubectl describe clusterrole eks:node-manager
```

存在する場合は出力を以前の ClusterRole の例と比較します。

eks:node-manager ClusterRoleBinding が存在することを確認します。

```
kubectl describe clusterrolebinding eks:node-manager
```

存在する場合は出力を以前の ClusterRoleBinding の例と比較します。

マネージド型ノードグループの操作のリクエスト中に AccessDenied エラーの原因として ClusterRole や ClusterRoleBinding が紛失または破損していることを発見した場合はそれらを復元できます。*eks-node-manager-role.yaml* という名前のファイルに、以下の内容を保存します。

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: eks:node-manager
rules:
- apiGroups:
  - ''
  resources:
  - pods
  verbs:
  - get
  - list
  - watch
  - delete
- apiGroups:
  - ''
  resources:
  - nodes
  verbs:
  - get
  - list
  - watch
  - patch
- apiGroups:
  - ''
  resources:
  - pods/eviction
  verbs:
  - create
---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: eks:node-manager
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: eks:node-manager
```

```
subjects:
- apiGroup: rbac.authorization.k8s.io
  kind: User
  name: eks:node-manager
```

ファイルを適用します。

```
kubectl apply -f eks-node-manager-role.yaml
```

ノードグループの操作を再試行して、問題が解決したかどうかを確認します。

Not authorized for images

`Not authorized for images` エラーメッセージの考えられる原因の 1 つは、プライベート Amazon EKS Windows AMI を使用して Windows マネージドノードグループを起動することです。新しい Windows AMI をリリースすると、AWS は 4 か月以上前の AMI を非公開にし、アクセス不可にします。マネージドノードグループがプライベート Windows AMI を使用している場合は、Windows マネージドノードグループを更新することを検討してください。非公開になった AMI へのアクセスを可能にできるかは保証できませんが、AWS サポートにチケットを提出することでアクセスをリクエストできます。詳細については、Amazon EC2 ユーザーガイドの「パッチ」を参照してください。

ノードが NotReady 状態です

ノードが `NotReady` 状態になった場合は、そのノードに異常があり、新しい Pod をスケジュールできない可能性があります。これはノードに CPU、メモリ、利用可能なディスクスペースの十分なリソースがないなど、さまざまな理由で発生します。

Amazon EKS 最適化 Windows AMI の場合、`kubelet` 設定のデフォルトではコンピューティングリソースの予約は指定されていません。リソースの問題を防ぐために、`kubelet` に kube-reserved および/または system-reserved の設定値を指定することで、システムプロセスのコンピューティングリソースを予約できます。これを行うにはブートストラップスクリプトの `-KubeletExtraArgs` コマンドラインパラメータを使用します。詳細については、Kubernetes ドキュメントの「Reserve Compute Resources for System Daemons」、およびこのユーザーガイドの「the section called “ブートストラップスクリプトの設定パラメータ”」を参照してください。

EKS ログコレクター

Amazon EKS ノードの問題をトラブルシューティングするために、ノード上の `/etc/eks/log-collector-script/eks-log-collector.sh` に構築済みのスクリプトが用意されています。このスクリプトを使用して、サポートケースおよび一般的なトラブルシューティングの診断ログを収集できます。

ノードでスクリプトを実行するには次のコマンドを使用します。

```
sudo bash /etc/eks/log-collector-script/eks-log-collector.sh
```

> **Note**
>
> スクリプトがその場所にない場合は、以下のコマンドを使用してスクリプトを手動でダウンロードして実行できます。
>
> ```
> curl -O https://amazon-eks.s3.amazonaws.com/support/log-collector-script/linux/eks-log-collector.sh
> sudo bash eks-log-collector.sh
> ```

このスクリプトは次の診断情報を収集します。

```
$ sudo bash /etc/eks/log-collector-script/eks-log-collector.sh

      This is version 0.7.8. New versions can be found at https://github.com/awslabs/amazon-eks-ami/blob/main/log-collector-script/

Trying to collect common operating system logs...
Trying to collect kernel logs...
Trying to collect mount points and volume information...
...
...

 Done... your bundled logs are located in /var/log/eks_i-EXAMPLE_2025-03-25_0000-UTC_0.7.8.tar.gz
```

診断情報が収集され、以下に保存されます。

```
/var/log/eks_i-EXAMPLE_2025-03-25_0000-UTC_0.7.8.tar.gz
```

Bottlerocket ノードのログバンドルを取得する方法の詳細については、「Bottlerocket Logs」を参照してください。

コンテナランタイムネットワークの準備ができていません

以下のような Container runtime network not ready エラーと承認エラーが表示される場合があります。

```
4191 kubelet.go:2130] Container runtime network not ready: NetworkReady=false
 reason:NetworkPluginNotReady message:docker: network plugin is not ready: cni config
 uninitialized
4191 reflector.go:205] k8s.io/kubernetes/pkg/kubelet/kubelet.go:452: Failed to list
 *v1.Service: Unauthorized
4191 kubelet_node_status.go:106] Unable to register node
 "ip-10-40-175-122.ec2.internal" with API server: Unauthorized
4191 reflector.go:205] k8s.io/kubernetes/pkg/kubelet/kubelet.go:452: Failed to list
 *v1.Service: Unauthorized
```

これは次のいずれかの理由で発生します。

1. クラスターに aws-auth ConfigMap がないか、ノードに設定した IAM ロールのエントリが含まれていないかのどちらかです。

   この問題を解決するには次のコマンドの *my-cluster* をクラスターの名前に置き換えて、変更したコマンド eksctl get iamidentitymapping --cluster *my-cluster* を実行することで、ConfigMap 内の現在のエントリを確認します。コマンドからエラーメッセージが表示される場合はクラスターに aws-auth ConfigMap がないことが原因である可能性があります。次のコマンドは ConfigMap にエントリを追加します。ConfigMap が存在しない場合はコマンドによって作成されます。*111122223333* を IAM ロールの AWS ID に、*myAmazonEKSNodeRole* をノードのロールの名前に置き換えます。

   ```
   eksctl create iamidentitymapping --cluster my-cluster \
       --arn arn:aws:iam::111122223333:role/myAmazonEKSNodeRole --group
    system:bootstrappers,system:nodes \
       --username system:node:{{EC2PrivateDNSName}}
   ```

指定するロールの ARN には / 以外のパスを含めることはできません。例えば、ロールの名前が development/apps/my-role の場合、ロールの ARN を指定するときに my-role に変更する必要があります。ノード IAM ロール ARN (インスタンスプロファイルの ARN ではない) を指定していることを確認してください。

2. セルフマネージド型ノードが、「EKS アクセスエントリを使用して Kubernetes へのアクセスを IAM ユーザーに許可する」のトピックの前提条件に記載されている最小バージョンのプラットフォームバージョンのクラスターにあるものの、ノードの IAM ロールのエントリが aws-auth ConfigMap (前の項目を参照) に記載されていないか、ロールのアクセスエントリが存在しません。この問題を解決するには次のコマンドの *my-cluster* をクラスターの名前に置き換えて、変更したコマンド aws eks list-access-entries --cluster-name *my-cluster* を実行することで、現在のアクセスエントリを確認します。次のコマンドはノードの IAM ロールのアクセスエントリを追加します。*111122223333* を IAM ロールの AWS ID に、*myAmazonEKSNodeRole* をノードのロールの名前に置き換えます。Windows ノードを使用している場合は*EC2_LINUX* を EC2_Windows に置き換えてください。ノード IAM ロール ARN (インスタンスプロファイルの ARN ではない) を指定していることを確認してください。

```
aws eks create-access-entry --cluster-name my-cluster --principal-arn
 arn:aws:iam::111122223333:role/myAmazonEKSNodeRole --type EC2_LINUX
```

TLS ハンドシェイクタイムアウト

ノードがパブリック API サーバーエンドポイントへの接続を確立できない場合、次のようなエラーが発生する可能性があります。

```
server.go:233] failed to run Kubelet: could not init cloud provider "aws": error
 finding instance i-1111f2222f333e44c: "error listing AWS instances: \"RequestError:
 send request failed\\ncaused by: Post  net/http: TLS handshake timeout\""
```

kubelet プロセスはAPI サーバーエンドポイントを継続的に再生成およびテストします。このエラーはコントロールプレーンでクラスターのローリング更新を行う手順 (設定の変更やバージョンの更新など) で一時的に発生することもあります。

この問題を解決するにはルートテーブルとセキュリティグループを確認して、ノードからのトラフィックがパブリックエンドポイントに到達できることを確認します。

無効なクライアントトークンID

中国 AWS リージョンでクラスターにデプロイされている Pod または DaemonSet 用サービスアカウントの IAM ロールを使用していて、仕様に AWS_DEFAULT_REGION 環境変数を設定していない場合、Pod または DaemonSet に次のエラーが表示されることがあります。

```
An error occurred (InvalidClientTokenId) when calling the GetCallerIdentity operation:
 The security token included in the request is invalid
```

この問題を解決するには、次の Pod の仕様例に示すように、AWS_DEFAULT_REGION 環境変数を Pod または DaemonSet の仕様に追加する必要があります。

```
apiVersion: v1
kind: Pod
metadata:
  name: envar-demo
  labels:
    purpose: demonstrate-envars
spec:
  containers:
  - name: envar-demo-container
    image: gcr.io/google-samples/node-hello:1.0
    env:
    - name: AWS_DEFAULT_REGION
      value: "region-code"
```

コントロールプレーンをアップグレードする前に、ノードグループが Kubernetes バージョンと一致する必要があります

コントロールプレーンを新しい Kubernetes バージョンにアップグレードする前に、クラスター内のマネージドノードと Fargate ノードのマイナーバージョンが、コントロールプレーンの現在のバージョンと同じである必要があります。Amazon EKS update-cluster-version API はすべての EKS マネージド型ノードが現在のクラスターバージョンにアップグレードされるまで、リクエストを拒否します。Amazon EKS はマネージド型ノードをアップグレードするための API を提供します。マネージドノードグループの Kubernetes バージョンのアップグレードについては、「the section called “更新”」を参照してください。Fargate ノードのバージョンをアップグレードするには、ノードによって表されるポッドを削除し、コントロールプレーンをアップグレードした後に

ポッドを再デプロイします。詳細については、「the section called “Kubernetes バージョンを更新する”」を参照してください。

多数のノードを起動すると、Too Many Requests エラーが発生します。

多数のノードを同時に起動すると、Amazon EC2 ユーザーデータの実行ログに「Too Many Requests」というエラーメッセージが表示される場合があります。これはコントロールプレーンが describeCluster 呼び出しで過負荷になっているために発生する可能性があります。過負荷が原因で、スロットリングが発生し、ノードがブートストラップスクリプトを実行できなくなり、ノードが完全にクラスターに参加できなくなります。

--apiserver-endpoint、--b64-cluster-ca、および --dns-cluster-ip 引数がノードのブートストラップスクリプトに渡されていることを確認してください。これらの引数を含める場合、ブートストラップスクリプトが describeCluster 呼び出しを行う必要がなくなり、コントロールプレーンが過負荷になるのを防ぐのに役立ちます。詳細については、「the section called “AMI を指定する”」を参照してください。

Kubernetes API サーバーリクエストに対する HTTP 401 Unauthorized エラーレスポンス

これらのエラーは、Pod のサービスアカウントトークンがクラスターで期限切れになった場合に表示されます。

Amazon EKS クラスターの Kubernetes API サーバーでは、90 日を超えるトークンによるリクエストが拒否されます。以前の Kubernetes バージョンでは、トークンに有効期限はありませんでした。つまり、これらのトークンに依存しているクライアントは、1 時間以内にトークンを更新する必要があります。無効なトークンが原因で Kubernetes API サーバーによりリクエストが拒否されないようにするには、ワークロードで使用されている Kubernetes クライアント SDK バージョンを、次のバージョンと同じか、それ以降にする必要があります。

- Go – バージョン 0.15.7 以降
- Python バージョン 12.0.0 以降
- Java バージョン 9.0.0 以降
- JavaScript バージョン 0.10.3 以降
- Ruby master ブランチ

- Haskell バージョン`0.3.0.0`
- C# バージョン `7.0.5` 以降

古いトークンを使用しているクラスター内の既存の Pod をすべて特定できます。詳細については、「the section called “サービスアカウントトークン”」を参照してください。

Amazon EKS プラットフォームのバージョンは現在のプラットフォーム バージョンより 2 つ以上遅れています

これは、Amazon EKS がクラスターの platform-version を自動的に更新できない場合に発生する可能性があります。これには多くの原因がありますが、一般的な原因のいくつかが続きます。これらの問題のいずれかがクラスターに当てはまる場合、それはまだ機能する可能性がありますが、そのプラットフォームバージョンは Amazon EKS によって更新されません。

問題

クラスターIAMロール が削除されました – このロールはクラスターの作成時に指定されました。次のコマンドで、どのロールが指定されたかを確認できます。*my-cluster* の部分は自分のクラスター名に置き換えます。

```
aws eks describe-cluster --name my-cluster --query cluster.roleArn --output text | cut
 -d / -f 2
```

出力例は次のとおりです。

```
eksClusterRole
```

ソリューション

同じ名前で新しい クラスター IAM ロール を作成します。

問題

クラスターの作成中に指定されたサブネットが削除されました – クラスターで使用するサブネットはクラスターの作成中に指定されました。次のコマンドで、指定されたサブネットを確認できます。*my-cluster* の部分は自分のクラスター名に置き換えます。

```
aws eks describe-cluster --name my-cluster --query cluster.resourcesVpcConfig.subnetIds
```

出力例は次のとおりです。

```
[
"subnet-EXAMPLE1",
"subnet-EXAMPLE2"
]
```

ソリューション

アカウントにサブネット ID が存在するかどうかを確認します。

```
vpc_id=$(aws eks describe-cluster --name my-cluster --query
 cluster.resourcesVpcConfig.vpcId --output text)
aws ec2 describe-subnets --filters "Name=vpc-id,Values=$vpc_id" --query
 "Subnets[*].SubnetId"
```

出力例は次のとおりです。

```
[
"subnet-EXAMPLE3",
"subnet-EXAMPLE4"
]
```

出力で返されたサブネット ID が、クラスターの作成時に指定されたサブネット ID と一致しない場合、Amazon EKS でクラスターを更新するにはクラスターで使用されるサブネットを変更する必要があります。これはクラスターの作成時に 2 つ以上のサブネットを指定した場合、Amazon EKS は指定したサブネットをランダムに選択して新しいエラスティックネットワークインターフェイスを作成するためです。これらのネットワーク インターフェイスにより、コントロールプレーンはノードと通信できます。Amazon EKS は選択したサブネットが存在しない場合、クラスターを更新しません。Amazon EKS が新しいネットワーク インターフェイスを作成するために選択する、クラスターの作成時に指定したサブネットをコントロールすることはできません。

クラスターの Kubernetes バージョンの更新を開始すると、同じ理由で更新が失敗する可能性があります。

問題

クラスターの作成中に指定されたセキュリティグループが削除されました – クラスターの作成中にセキュリティグループを指定した場合は次のコマンドでそれらの ID を確認できます。*my-cluster* の部分は自分のクラスター名に置き換えます。

```
aws eks describe-cluster --name my-cluster --query
 cluster.resourcesVpcConfig.securityGroupIds
```

出力例は次のとおりです。

```
[
    "sg-EXAMPLE1"
]
```

[] が返された場合、クラスターの作成時にセキュリティグループが指定されておらず、セキュリティグループの欠落は問題ではありません。セキュリティグループが返された場合はセキュリティグループがアカウントに存在することを確認します。

ソリューション

これらのセキュリティグループがアカウントに存在するかどうかを確認します。

```
vpc_id=$(aws eks describe-cluster --name my-cluster --query
 cluster.resourcesVpcConfig.vpcId --output text)
aws ec2 describe-security-groups --filters "Name=vpc-id,Values=$vpc_id" --query
 "SecurityGroups[*].GroupId"
```

出力例は次のとおりです。

```
[
"sg-EXAMPLE2"
]
```

出力で返されたセキュリティグループ ID が、クラスターの作成時に指定されたセキュリティグループ ID と一致しない場合、Amazon EKS でクラスターを更新するにはクラスターで使用されるセキュリティグループを変更する必要があります。クラスターの作成時に指定されたセキュリティグループ ID が存在しない場合、Amazon EKS はクラスターを更新しません。

クラスターの Kubernetes バージョンの更新を開始すると、同じ理由で更新が失敗する可能性があります。

- クラスターの作成時に指定した各サブネットで、少なくとも 6 つ (ただし、16 をお勧めします) の使用可能な IP アドレスがありません。サブネットに十分な使用可能な IP アドレスがない場合はサブネット内の IP アドレスを解放するか、十分な使用可能な IP アドレスを持つサブネットを使用するように、クラスターで使用されるサブネットを変更する必要があります。

- クラスターの作成時にシークレット暗号化を有効にしましたが、指定した AWS KMS キーは削除されました。Amazon EKS でクラスターを更新する場合は新しいクラスターを作成する必要があります

クラスターの正常性に関するよくある質問およびエラーコードと解決パス

Amazon EKS は EKS クラスターとクラスターインフラストラクチャの問題を検出し、それらを EKS クラスターリソースのヘルスオブジェクトに保存します。クラスターヘルスの情報を利用することで、クラスターの問題をより迅速に検出、トラブルシューティング、対処できます。これにより、より安全で最新のアプリケーション環境を構築できます。さらに、必要なインフラストラクチャまたはクラスター設定に問題があるため、Kubernetes の新しいバージョンにアップグレードしたり、Amazon EKS が劣化したクラスターにセキュリティ更新をインストールしたりできない場合があります。Amazon EKS は問題を検出したり、問題が解決されたことを検出したりするまでに 3 時間かかる場合があります。

Amazon EKS クラスターの状態は Amazon EKS とそのユーザー間で共有される責任です。IAM ロールや Amazon VPC サブネットの前提となるインフラストラクチャ、および事前に提供する必要があるその他の必要なインフラストラクチャはお客様の責任となります。Amazon EKS はこのインフラストラクチャとクラスターの設定の変更を検出します。

Amazon EKS コンソールからクラスターのヘルスにアクセスするには、Amazon EKS クラスターの詳細ページからオブザーバビリティのダッシュボードにアクセスし、[クラスターヘルスの問題] タブの中から [ヘルスの問題] と書かれたテーブルを探します。このデータは EKS API の `DescribeCluster` アクションを、例えば AWS コマンドラインインターフェイス内から呼び出すことでも使用できます。

この機能を使用する理由は何ですか?

> Amazon EKS クラスターの状態を把握しやすくなり、問題を迅速に診断して修正できます。デバッグや AWS サポートケースの作成に時間を費やす必要はありません。例えば、Amazon EKS クラスターのサブネットを誤って削除した場合、Amazon EKS はクロスアカウントのネットワークインターフェイスや `kubectl exec` や `kubectl logs` などの Kubernetes AWS CLI コマンドを作成できなくなります。この場合、次のエラーとともに処理が失敗します: `Error from server: error dialing backend: remote error: tls: internal error.` そして、次のような Amazon EKS ヘルスの問題が表示されます: `subnet-da60e280 was deleted: could not create network interface`。

この機能は他の AWS サービスとどのように関連し、連携しますか?

IAM ロールと Amazon VPC サブネットはクラスターヘルスが問題を検出するための前提条件となるインフラストラクチャの 2 つの例です。この機能はこれらのリソースが正しく設定されていない場合に詳細情報を返します。

ヘルスに問題があるクラスターには料金がかかりますか?

はい。Amazon EKS クラスターはすべて標準の Amazon EKS 料金で請求されます。クラスターのヘルス機能は追加料金なしで利用できます。

この機能は AWS アウトポスト の Amazon EKS クラスターで動作しますか?

はい。クラスターの問題は、AWS Outposts 上の拡張クラスターや AWS Outposts 上のローカルクラスターを含む AWS クラウド内の EKS クラスターで検出されます。クラスターヘルスでは、Amazon EKS Anywhere や Amazon EKS Distro (EKS-D) の問題は検出されません。

新しい問題が検出されたときに通知を受けることはできますか?

はい。AWS は新しいクラスターのヘルスの問題が検出されると、E メールと Personal Health Dashboard の通知を送信します。

コンソールにはヘルスの問題に関する警告が表示されますか?

はい。ヘルスに問題があるクラスターにはコンソールの上部にバナーが表示されます。

最初の 2 つの列は API レスポンス値に必要なものです。Health ClusterIssue オブジェクトの 3 番目のフィールドは resourceIds で、返される値は課題タイプによって異なります。

コード	メッセージ	ResourceIds	クラスターは回復可能ですか?
SUBNET_NOT_FOUND	クラスターに現在関連付けられている 1 つ以上のサブネットが見つかりませんでした。Amazon EKS update-cluster-config API を呼び出して、サブネットを更新します。	サブネット ID	はい

コード	メッセージ	ResourceIds	クラスターは回復可能ですか?
SECURITY_GROUP_NOT_FOUND	クラスターに現在関連付けられている 1 つ以上のセキュリティグループが見つかりませんでした。セキュリティグループを更新するには、Amazon EKS update-cluster-config API を呼び出します。	セキュリティグループ ID	はい
IP_NOT_AVAILABLE	クラスターに関連付けられている 1 つ以上のサブネットに、Amazon EKS がクラスター管理操作を実行するための十分な IP アドレスがありません。サブネット内のアドレスを解放するか、Amazon EKS update-cluster-config API を使用して別のサブネットをクラスターに関連付けます。	サブネット ID	はい
VPC_NOT_FOUND	クラスターに関連付けられている VPC が見つかりませんでした。クラスターを削除して再作成する必要があります。	VPC ID	いいえ

コード	メッセージ	ResourceIds	クラスターは回復可能ですか?
ASSUME_ROLE_ACCESS_DENIED	クラスターは Amazon EKS サービスにリンクされたロールを使用していません。クラスターに関連するロールを引き受けて、必要な Amazon EKS 管理オペレーションを実行することができませんでした。ロールが存在し、必要な信頼ポリシーが適用されていることを確認してください。	クラスター IAM ロール	はい

コード	メッセージ	ResourceIds	クラスターは回復可能ですか?
PERMISSION_ACCESS_DENIED	クラスターは Amazon EKS サービスにリンクされたロールを使用していません。クラスターに関連付けられているロールでは、Amazon EKS に必要な管理操作を実行するための十分なアクセス許可が付与されていません。クラスターロールにアタッチされているポリシーを確認し、別の拒否ポリシーが適用されているかどうかを確認してください。	クラスター IAM ロール	はい
ASSUME_ROLE_ACCESS_DENIED_USING_SLR	Amazon EKS クラスター管理サービスにリンクされたロールを引き受けることができませんでした。ロールが存在し、必要な信頼ポリシーが適用されていることを確認してください。	Amazon EKS サービスにリンクされたロール	はい

コード	メッセージ	ResourceIds	クラスターは回復可能ですか?
PERMISSION_ACCESS_DENIED_USING_SLR	Amazon EKS クラスター管理サービスにリンクされたロールは Amazon EKS に必要な管理操作を実行するための十分なアクセス許可を付与しません。クラスターロールにアタッチされているポリシーを確認し、別の拒否ポリシーが適用されているかどうかを確認してください。	Amazon EKS サービスにリンクされたロール	はい
OPT_IN_REQUIRED	アカウントに Amazon EC2 サービスのサブスクリプションがありません。アカウント設定ページでアカウントサブスクリプションを更新してください。	該当なし	はい
STS_REGIONAL_ENDPOINT_DISABLED	STS リージョナルエンドポイントは無効になっています。Amazon EKS のエンドポイントで必要なクラスター管理操作を実行できるようにします。	該当なし	はい

コード	メッセージ	ResourceIds	クラスターは回復可能ですか?
KMS_KEY_DISABLED	クラスターに関連付けられている AWS KMS キーは無効になっています。クラスターを復元するにはキーを再度有効にします。	KMS Key ARN	はい
KMS_KEY_NOT_FOUND	クラスターに関連付けられている AWS KMS キーが見つかりませんでした。クラスターを削除して再作成する必要があります。	KMS Key ARN	いいえ
KMS_GRANT_REVOKED	クラスターに関連付けられた AWS KMS キーの付与は取り消されます。クラスターを削除して再作成する必要があります。	KMS Key ARN	いいえ

コード	メッセージ	ResourceIds	クラスターは回復可能ですか?
ETCD_DB_SIZE_EXCEEDED	Amazon EKS クラスターが etcd データベースのサイズ制限を超えました。etcd は、Kubernetes コントロールプレーンで実行され、クラスターの設定と状態を維持するキーと値のデータストアです。クラスターがパフォーマンス低下状態にならないようにするには、不要な Kubernetes オブジェクトを削除して etcd データベースのサイズを減らしてください。データベースサイズに寄与するオブジェクトの特定とクリーンアップに関するガイダンスについては、「Amazon EKS クラスターでの etcd データベースサイズの管理」を参照してください。クリーンアップ後も問題が解決しない場合は、AWS サポートにお問い合わせください。	クラスター ARN	はい

オープンソースプロジェクトで Amazon EKS の機能を拡張する

これらのオープンソースプロジェクトは、Amazon EKS によって管理されているクラスターを含め、AWS 上や外部で実行されている Kubernetes クラスターの機能を拡張します。

EKS にデプロイされたソフトウェアのサポート

Amazon EKS のドキュメントを確認すると、手順や例を使用した、さまざまなオープンソースのツールやソフトウェアへの参照があります。これらのツールには、Kubernetes メトリクスサーバーや Cert Manager も含まれています。

デプロイするサードパーティーソフトウェアまたはオープンソースソフトウェアは、AWS サポート契約の範囲外であることに注意してください。Kubernetes を使用する利点は、アクティブなオープンソースコミュニティであるということです。関連するオープンソースコミュニティやプロジェクトの管理者と直接連携し、そのようなコンポーネントに適切なサポートチャネルを確立することをお勧めします。詳細については、Cloud Native Computing Foundation (CNCF) に関連付けられた Graduated Projects および Incubating Projects を参照してください。

Kubernetes エコシステムには、さまざまなレベルのコミュニティサポート、応答時間、および目的のユースケースを備えたプロジェクトやコンポーネントが多数含まれています。これらのテクノロジーを EKS と一緒に実装する場合は、各コンポーネントに対するサポートマトリックスについて確実に理解しておいてください。

AWS では、EKS コントロールプレーンに統合するオープンソースコンポーネントを管理しています。これには、配布するすべてのコンテナイメージとバイナリのビルド検証、脆弱性スキャン、検証テスト、パッチ管理を網羅する包括的なセキュリティのパイプラインが含まれます。例えば、AWS は Kubernetes API サーバーに対する責任を負います。したがって、Kubernetes API サーバーは Amazon EKS サービスレベルアグリーメントの対象となります。Amazon Web Services サポートプランを使用して、Kubernetes API サーバーの問題を解決したり、一般的なガイダンスを取得したりできます。

さまざまな Amazon EKS アドオンで提供されているサポートを慎重に確認する必要があります。AWS のアドオンは、AWS で完全にサポートされている Amazon EKS アドオンの唯一のタイプです。AWSマーケットプレイスのアドオンは、主に AWS パートナーによってサポートされています。コミュニティのアドオンは、AWS から基本的なライフサイクルサポートを受けます。詳細については、「アドオンのサポート」を参照してください。

マーケットプレイスのアドオンなど、アドオンのタイプに関係なくすべての EKS アドオンが EKS から基本的なライフサイクルサポートを受けます。基本的なライフサイクルサポートには、アドオンのインストールとアンインストールが含まれます。使用可能な Amazon EKS アドオンのタイプと関連するサポートレベルの詳細については、「Amazon EKS アドオンのサポート範囲」を参照してください。AWS で完全にサポートされているアドオンを表示するには、「Amazon Web Services のアドオン」を参照してください。

- セキュリティプラクティスとサポート境界の詳細については、「Amazon EKS のセキュリティ」を参照してください。
- Amazon EKS アドオンで利用可能なコミュニティのアドオンと AWS のマーケットプレイスのアドオンの詳細については、「EKS アドオンのサポート」を参照してください。

管理ツール

Amazon EKS および Kubernetes クラスターに関連する管理ツール。

eksctl

`eksctl` は、Amazon EKS 上にクラスターを作成するためのシンプルな CLI ツールです。

- プロジェクト URL
- プロジェクトドキュメント
- AWS オープンソースブログ: 「eksctl: 1 つのコマンドによる Amazon EKS クラスター」

AWS Controllers for Kubernetes

Kubernetes 用 AWS コントローラーを使用すると、Kubernetes クラスターから、直接 AWS リソースを作成および管理できます。

EKS 機能で利用可能です。

- プロジェクト URL
- AWS オープンソースブログ: AWS Service Operator for Kubernetes Now Available

kro (Kube Resource Orchestrator)

kro を使用すると、複数のリソースを高レベルの抽象化に構成するカスタム Kubernetes API を作成できます。プラットフォームチームはガードレール付きの再利用可能なパターンを定義できますが、アプリケーションチームはシンプルで高レベルな API を使用してリソースをプロビジョニングおよび管理します。

EKS 機能で利用可能です。

- プロジェクト URL
- プロジェクトドキュメント

Argo CD

Argo CD は、Kubernetes 向けの宣言型 GitOps 継続的デリバリーツールです。これは、Git リポジトリを継続的にモニタリングし、変更をクラスターに自動的に同期します。

EKS 機能で利用可能です。

- プロジェクト URL
- プロジェクトドキュメント

Flux CD

Flux は Git を使用して、クラスター設定の管理に使用できるツールです。クラスター内のオペレータを使用して、Kubernetes 内のデプロイをトリガーします。オペレーターの詳細については、GitHub の「OperatorHub.io」を参照してください。

- プロジェクト URL
- プロジェクトドキュメント

Kubernetes 用 CDK

Kubernetes 用 CDK (cdk8s) を使用すると、使い慣れたプログラミング言語を使用して Kubernetes アプリとコンポーネントを定義できます。cdk8s アプリは標準の Kubernetes マニフェストに合成され、どの Kubernetes クラスターにも適用できます。

- プロジェクト URL

- プロジェクトドキュメント
- AWS コンテナのブログ: 「cdk8s+ の紹介: Kubernetes オブジェクト用のインテント駆動型 API」

ネットワーク

Amazon EKS および Kubernetes クラスターに関連するネットワークプロジェクト。

Amazon VPC CNI Plugin for Kubernetes

Amazon EKS は、Amazon VPC CNI Plugin for Kubernetes を使用した、従来の VPC ネットワークをサポートしています。プラグインは、VPC から各 Pod に IP アドレスを割り当てます。

- プロジェクト URL
- プロジェクトドキュメント

AWS Load Balancer Controller for Kubernetes

AWS Load Balancer Controller は、Kubernetes クラスターの AWS Elastic Load Balancers の管理に役立ちます。AWS Application Load Balancers をプロビジョニングすることで、Kubernetes Ingress リソースを満たします。AWS Network Load Balancers をプロビジョニングすることで、Kubernetes Service リソースを満たします。

- プロジェクト URL
- プロジェクトドキュメント

ExternalDNS

ExternalDNS は、公開された Kubernetes サービスとそのイングレスを、Amazon Route 53 や AWS Service Discovery を含む DNS プロバイダーと同期させます。

- プロジェクト URL
- プロジェクトドキュメント

機械学習

Amazon EKS および Kubernetes クラスターに関連する機械学習プロジェクト。

Kubeflow

Kubernetes 用の機械学習ツールキット。

- プロジェクト URL
- プロジェクトドキュメント
- AWS オープンソースブログ: 「Amazon EKS での Kubeflow」

Auto Scaling

Amazon EKS および Kubernetes クラスターに関連する自動スケーリングプロジェクト。

Cluster Autoscaler

Cluster Autoscaler は、CPU とメモリーの負荷に基づいて Kubernetes クラスターのサイズを自動的に調整するツールです。

- プロジェクト URL
- プロジェクトドキュメント
- Amazon EKS ワークショップ: Cluster Autoscaler

Karpenter

Karpenter は、柔軟性、パフォーマンス、シンプルさを実現するために構築された Kubernetes Node Autoscaler です。

- プロジェクト URL
- プロジェクトドキュメント
- Amazon EKS ワークショップ: Karpenter

Escalator

Escalator は、Kubernetes 用のバッチまたはジョブ最適化水平オートスケーラーです。

- プロジェクト URL
- プロジェクトドキュメント

モニタリング

Amazon EKS および Kubernetes クラスターに関連するモニタリングプロジェクト。

Prometheus

Prometheus はオープンソースのシステムモニタリングおよび警告ツールキットです。

- プロジェクト URL
- プロジェクトドキュメント
- Amazon EKS ワークショップ: https://eksworkshop.com/intermediate/240_monitoring/

継続的インテグレーション/継続的デプロイメント

Amazon EKS および Kubernetes クラスターに関連する命令型 CI/CD プロジェクト。

Jenkins X

Amazon EKS および Kubernetes クラスター上の最新のクラウドアプリケーション用 CI/CD ソリューション。

- プロジェクト URL
- プロジェクトドキュメント

Amazon EKS 新機能とロードマップの詳細

Amazon EKS の新機能については、AWS の最新情報ページの新機能フィードをスクロールし、確認してください。また、GitHub のroadmap (ロードマップ)を参照してください。そうすることで今後の機能と優先度について知り、今後の Amazon EKS の使用方法についての計画を立てることができます。ロードマップの優先度について、直接フィードバックをお寄せください。

ドキュメント履歴

次の表は、Amazon EKS ユーザーガイドの主な更新と新機能を示しています。このテーブルが新しいエントリを取得したときに通知を受け取るには、RSS リーダーを使用して次の URL をサブスクライブできます。

```
https://docs.aws.amazon.com/eks/latest/userguide/doc-history.rss
```

変更	説明	日付
Kubernetes バージョン 1.37	Amazon EKS に Kubernetes バージョン 1.37 を追加しました。	2026 年 10 月 1 日
EKS 最適化 Windows Server 2025 AMI での CimFS スナップショッターのサポート	Amazon EKS 最適化 Windows Server 2025 AMI は、オプトインの Composite Image File System (CimFS) containerd スナップショッターをサポートするようになりました。この機能により、大きな Windows コンテナイメージのコールドイメージプル時間を短縮できます。CimFS は Kubernetes バージョン 1.35 以降で使用でき、デフォルトではオフになっています。	2026 年 9 月 21 日
EKS 最適化 AL2023 NVIDIA AMI に 2 つの NVIDIA ドライバーバージョンを同梱	EKS 最適化 AL2023 NVIDIA AMI では、2 つの NVIDIA ドライバーバージョンが 1 つの AMI にバンドルされるようになりました。各ノードは起動時に GPU に適したドライバーを自動的に選択するため、カスタム AMI を構築せずに g7	2026 年 9 月 15 日

	インスタンスタイプをサポートできます。	
AWS マネージドポリシーの更新	Amazon EKS Auto Mode コントローラーが Application Load Balancer の関連付けを調整する際に AWS WAF Web ACL を一覧表示できるようにするため、AmazonEKSLoadBalancingPolicy に wafv2:ListWebACLs アクセス許可を追加しました。AmazonEKSLoadBalancingPolicy のリソースタグ付け許可リストに gateway.eks.amazonaws.com.alb/stack 、gateway.eks.amazonaws.com.alb/resource 、gateway.eks.amazonaws.com.nlb/stack 、gateway.eks.amazonaws.com.nlb/resource のタグキーを追加しました。これにより、Amazon EKS Auto Mode コントローラーが Kubernetes Gateway API (ALB および NLB) ワークロードの AWS リソースを管理できるようになります。	2026 年 9 月 2 日

決定論的な Argo CD 機能エンドポイント	EKS Capability for Argo CD エンドポイントの形式を文書化しました。エンドポイントのホスト名は決定論的であるため、機能名、クラスター名、AWS アカウント ID、AWS リージョンからコンストラクトすることも、`describe-capability` の `serverUrl` フィールドから取得することもできます。	2026 年 8 月 24 日
クラスター OIDC エンドポイントの AWS PrivateLink	Amazon EKS は、クラスター OIDC 検出/JWKS エンドポイント (`com.amazonaws.region-code.oidc-eks`) の AWS PrivateLink をサポートするようになりました。これにより、インターネットエグレスなしで VPC からプライベートにサービスアカウント (IRSA) の IAM ロールによって使用されるエンドポイントに到達できます。	2026 年 7 月 28 日

EKS Auto Mode での静的キャパシティの EFA サポート	EKS Auto Mode NodeClass リソースでの EFA ネットワークインターフェイスの設定のサポートが追加されました。高性能な分散トレーニングおよび推論ワークロードのために、インスタンスの起動時にネットワークインターフェイスを静的に定義できるようになりました。	2026 年 7 月 21 日
サービスにリンクされたロールの更新	サービスにリンクされたロールがクラスターアクセスエントリを関連付けおよび関連付け解除できる AmazonEKSServiceRolePolicy のアクセスポリシーのリストに AmazonEKSPodIdentityPolicy が追加されました。この追加により、Amazon EKS は EKS クラスターで実行されているワークロードの Pod Identity 認証情報を管理できます。	2026 年 6 月 26 日
コントロールプレーンのエグレスルーティング	顧客ルーティングのコントロールプレーンエグレス (controlPlaneEgressMode=CUSTOMER_ROUTED) のサポートが追加され、クラスターコントロールプレーンから VPC 内のリソースへのネットワークパスを管理できるようになりました。	2026 年 6 月 18 日

サービスにリンクされたロールの更新	Amazon EKS Auto Mode Controller が ALB の指定されたリスナールールの優先順位を設定できるようにするために、elasticloadbalancing:SetRulePriorities アクセス許可が AmazonEKSLoadBalancingPolicy に追加されました。ipam プールを記述するために、ec2:DescribeIpamPools アクセス許可が AmazonEKSLoadBalancingPolicy に追加されました	2026 年 6 月 15 日
Kubernetes バージョン 1.36	Amazon EKS に Kubernetes バージョン 1.36 を追加しました。	2026 年 6 月 2 日

AWS マネージドポリシーの更新	既存の eks:cluster-name タグと ebs.csi.aws.com/cluster-name タグに加えて、eks:eks-cluster-name タグでタグ付けされたインスタンスで ec2:AttachVolume と ec2:DetachVolume を承認するように AmazonEBSCSIDriverEKSClusterScopedPolicy を更新しました。これにより、オープンソースの Karpenter や Amazon EKS Auto Mode など、EKS コントローラーのタグ付け標準に準拠したコントローラーとの互換性が追加されました。	2026 年 5 月 28 日
AWS マネージドポリシーの更新	Amazon EKS Auto Mode がアカウントの中断可能なキャパシティ予約からインスタンスを起動できるようにするために、AmazonEKSComputePolicy にアクセス許可が追加されました。	2026 年 5 月 18 日
AWS マネージドポリシーの更新	新しい AWS 管理ポリシーが追加されました: AmazonEKSBlockStoragePolicyV2	2026 年 5 月 12 日

AWS マネージドポリシーの更新	EBS CSI ドライバーがボリューム制限/カード情報を実行時に動的に取得できるように、AmazonEBSCSIDriverPolicy に新しい API アクション ec2:DescribeInstanceTypes が追加されました。	2026 年 5 月 12 日
サービスにリンクされたロールの更新	Amazon EKS オートモードコントローラーで LoadBalancer を記述できるように、AmazonEKSLoadBalancingPolicy に elasticloadbalancing:DescribeLoadBalancers アクセス許可が追加されました。	2026 年 4 月 27 日
Kubernetes バージョン 1.32 が AWS Outposts のローカルクラスターで利用可能に	AWS Outposts のローカルの Amazon EKS クラスター向けに、Kubernetes バージョン 1.32 のサポートが追加されました。	2026 年 4 月 23 日
サービスにリンクされたロールの更新	Amazon EKS オートモードコントローラーで LoadBalancer キャパシティユニットを記述できるように、AmazonEKSLoadBalancingPolicy に elasticloadbalancing:DescribeCapacityReservation アクセス許可が追加されました。	2026 年 4 月 20 日

サービスにリンクされたロールの更新	AmazonEKSServiceRolePolicy に ec2:DescribeInstanceStatus および iam:ListInstanceProfiles アクセス許可が追加されました。これらのアクセス許可により、Amazon EKS オートモードでマネージドインスタンスの EC2 ヘルスイベントを検出し、マネージドインスタンスプロファイルをクリーンアップできます。	2026 年 4 月 20 日
AWS マネージドポリシーの更新	AmazonEKSComputePolicy にアクセス許可が追加され、Amazon EKS Auto Mode でアカウントの EC2 プレイスメントグループにインスタンスを起動できるようになりました。	2026 年 4 月 17 日

新しい AWS マネージドポリシー	AmazonEBSCSIDriverPolicyV2 および AmazonEBSCSIDriverEKSClusterScopedPolicy のより制限の厳しい代替手段として、AmazonEBSCSIDriverPolicy が導入されました。AmazonEBSCSIDriverPolicyV2 は、EBS CSI ドライバーを true に設定された ebs.csi.aws.com/cluster でタグ付けされたボリュームとスナップショットにスコープします。AmazonEBSCSIDriverEKSClusterScopedPolicy は、ebs.csi.aws.com/cluster-name タグを使用して、特定の EKS クラスターに属するボリュームとスナップショットにドライバーをスコープします。	2026 年 4 月 16 日
AWS マネージドポリシーの更新	AmazonEKSServiceRolePolicy にアクセス許可が追加され、Amazon EKS Auto Mode でアカウントの EC2 プレイスメントグループにインスタンスを起動できるようになりました。	2026 年 4 月 15 日

AWS マネージドポリシーの更新	AmazonEKSLoadBalancingPolicy の shield:CreateProtection および shield:DeleteProtection アクセス許可を更新しました。これにより、Amazon EKS Auto Mode コントローラーはシールド保護を追加できます。	2026 年 4 月 14 日
マネージド型ノードグループのウォームプールサポート	Amazon EKS マネージド型ノードグループが、Amazon EC2 Auto Scaling ウォームプールをサポートするようになりました。ウォームプールは、スケールアウト時にクラスターにすばやく参加できる初期化済みの EC2 インスタンスのプールを維持し、ブート時間が長いアプリケーションのレイテンシーを短縮します。	2026 年 4 月 7 日

AWS マネージドポリシーの更新	AmazonEKSLoadBalancingPolicy でアクセス許可を elasticloadbalancing:ModifyIpPools 追加および elasticloadbalancing:RegisterTargets 更新しました。これにより、Amazon EKS 自動モードコントローラーは ALB の IPAM プールの設定をサポートし、マルチクラスターとカスタムのターゲットグループ割り当てをそれぞれサポートできます。	2026 年 3 月 20 日
AWS マネージドポリシーの更新	AWSServiceRoleForAmazonEKSNodegroup ポリシーに autoscaling:PutWarmPool 、autoscaling:DeleteWarmPool 、および autoscaling:DescribeWarmPool のアクセス許可を追加しました。これにより、Amazon EKS Managed Nodegroup はノードグループのライフサイクル全体で基盤となる ASG ウォームプールリソースを管理できます。	2026 年 2 月 17 日

AWS マネージドポリシーの更新	AmazonEKSNetworkingPolicy に ec2:ModifyNetworkInterfaceAttribute アクセス許可を追加しました。これにより、Amazon EKS 自動モードコントローラーは EC2 インスタンスに関連するネットワークインターフェイス属性を変更できます。	2026 年 2 月 3 日
AWS マネージドポリシーの更新	AmazonEKSServiceRolePolicy 内の、iam:GetInstanceProfile アクセス許可のターゲットインスタンスプロファイル名に「eks」プレフィックスを付ける要件を削除しました。これにより、Amazon EKS Auto Mode は「eks」という命名プレフィックスを必要とせずに NodeClasses のカスタムインスタンスプロファイルを検証して使用できます。	2026 年 2 月 2 日
Kubernetes バージョン 1.35	Kubernetes バージョン 1.35 の新しいクラスターとバージョンアップグレードのサポートが追加されました。	2026 年 1 月 27 日
AWS マネージドポリシーの更新	EBS CSI ドライバーが EBS スナップショットを直接ロックできるように、AmazonEBSCSIDriverPolicy に ec2:LockSnapshot アクセス許可を追加しました。	2026 年 1 月 15 日

Amazon EKS の機能	Amazon EKS は、EKS 機能を使用したフルマネージド型のクラスター機能を提供し、Argo CD のサポートによる宣言型の継続的デプロイのサポート、AWS Controllers for Kubernetes のサポートによる AWS リソース管理、Kube Resource Orchestrator のサポートによる Kubernetes リソースのコンポジションとオーケストレーションをサポートするようになりました。	2025 年 11 月 30 日
新しい AWS マネージドポリシー	Amazon EKS は、オブザーバビリティとトラブルシューティングのために、Amazon EKS MCP サーバーで読み取り専用ツールを有効にする新しいマネージドポリシー `AmazonEKSMCPReadOnlyAccess` をリリースしました。詳細については、「Amazon EKS による AWS 管理ポリシーの更新」を参照してください。	2025 年 11 月 21 日
ネットワークオブザーバビリティ	Amazon EKS では、パフォーマンスモニタリングとワークロードトラフィックの可視性により、コンテナネットワークのオブザーバビリティが向上しました。	2025 年 11 月 19 日

AWS マネージドポリシーの更新	EBS CSI ドライバーが EBS ボリュームを直接コピーできるように、AmazonEBSCSIDriverPolicy に ec2:CopyVolumes アクセス許可を追加しました。	2025 年 11 月 17 日
AWS マネージドポリシーの更新	AmazonEKSServiceRolePolicy に ec2:DescribeRouteTables および ec2:DescribeNetworkAcls アクセス許可を追加しました。これにより、Amazon EKS はクラスターインサイトの一部として、ハイブリッドノードの VPC ルートテーブルとネットワーク ACL の設定問題を検出できます。	2025 年 10 月 22 日
サービスにリンクされたロールの更新	AmazonEKSConnectorServiceRolePolicy に ssmmessages:OpenDataChannel アクセス許可を追加しました。	2025 年 10 月 15 日
Kubernetes バージョン 1.34	Kubernetes バージョン 1.34 の新しいクラスターとバージョンアップグレードのサポートが追加されました。	2025 年 10 月 2 日
設定可能なノードの自動修復	Amazon EKS では、ノードの自動修復動作をより詳細に制御できるようになりました。	2025 年 9 月 22 日

クラスターインサイトを更新する	クラスターインサイトは、現時点では手動で更新できません。	2025 年 8 月 27 日
AWS マネージドポリシーの更新	AmazonEKSServiceRolePolicy にアクセス許可が追加されました。このロールは、新しいアクセスポリシー AmazonEKSEventPolicy をアタッチできます。ec2:DeleteLaunchTemplate と ec2:TerminateInstances の制限されたアクセス許可。	2025 年 8 月 26 日
サービス間の混乱した代理の防止	サービス間の混乱した代理の防止に適用できる信頼ポリシーの例と共に、その内容のトピックを追加しました。Amazon EKS は、EKS クラスターロールの信頼ポリシーで aws:SourceArn 条件と aws:SourceAccount 条件を受け入れます。	2025 年 8 月 19 日
Amazon EKS プラットフォームバージョンの更新	これはセキュリティの修正および機能強化が行われた新しいプラットフォームバージョンです。これには、Kubernetes 1.33.2、1.32.6、1.31.10、1.30.14 の新しいパッチバージョンが含まれます。	2025 年 7 月 30 日

マルチホームポッドの VPC CNI マルチ NIC 機能	Amazon EKS は、VPC CNI にマルチホームポッドを追加しました。これにより、各ポッドに対し、ワークロードと、EC2 インスタンス上のすべての NIC から VPC CNI が割り当てた IP アドレスを設定できるようになりました。アプリケーションは、各 NIC の帯域幅を使用する同時接続を行うことができます。すべてのネットワークインターフェイスは、ノードと同じサブネットとセキュリティグループで設定されます。以前は、マルチホームポッドを作成するためには、Multus CNI を使用して他の複数の CNI を実行する必要がありました。これを行うためのドキュメントと手順は https://docs.aws.amazon.com/eks/latest/userguide/pod-multus.html に移動しました。	2025 年 7 月 15 日
VPC CNI トラブルシューティングコンテンツの更新	VPC CNI の Kubernetes ネットワークポリシー のトラブルシューティングページを拡大しました。CRD および RBAC のアクセス許可に加え、12 の既知の問題が追加されました。	2025 年 6 月 30 日

Amazon EKS プラットフォームバージョンの更新	これはセキュリティの修正および機能強化が行われた新しいプラットフォームバージョンです。これらのプラットフォームバージョンには、Kubernetes の新しいパッチのバージョンはありません。	2025 年 6 月 26 日
AWS マネージドポリシーの更新	AmazonEKSLocalOutpostServiceRolePolicy に ssmmessages:OpenDataChannel アクセス許可が追加されました。	2025 年 6 月 26 日
AWS マネージドポリシーの更新	アカウントで EC2 オンデマンドキャパシティ予約を使用して Amazon EKS Auto Mode がインスタンスを起動できるようにするため、AmazonEKSServiceRolePolicy と AmazonEKSComputePolicy にアクセス許可が追加されました。また、Amazon EKS がユーザーに代わって EC2 スポットサービスにリンクされたロールを作成するため、AmazonEKSComputePolicy にアクセス許可が追加されました。	2025 年 6 月 20 日
マネージドポリシーの更新	AmazonEKSDashboardConsoleReadOnly ポリシーを追加しました。	2025 年 6 月 19 日

Amazon EKS Auto Mode の NodeClass への更新	Auto Mode ノードの NodeClass テンプレートに、個別のポッドサブネットの設定が追加されました。これにより、オプションのキー podSubnetSelectorTerms と podSecurityGroupSelectorTerms が追加され、ポッドのサブネットとセキュリティグループが設定されます。	2025 年 6 月 13 日
EKS Pod Identity を使用してセカンダリロールとクロスアカウントロールをターゲットにする	Amazon EKS は、ターゲット IAM ロール を EKS Pod Identity に追加して、ロールの連鎖を自動化します。これを使用して、別のアカウントのロールを自動的に引き受けると、EKS Pod Identity は一時的な認証情報をローテーションします。各 Pod Identity 関連付けには、最初に引き受ける同じアカウントに IAM ロールが必要です。次に、そのロールを使用してターゲットロールを引き受けます。	2025 年 6 月 11 日
Amazon EKS プラットフォームバージョンの更新	これはセキュリティの修正および機能強化が行われた新しいプラットフォームバージョンです。これらのプラットフォームバージョンには、Kubernetes の新しいパッチのバージョンはありません。	2025 年 6 月 11 日

Amazon EKS AWS リージョンの拡張	Amazon EKS がアジアパシフィック (台北) (ap-east-2) AWS リージョンで利用可能になりました。	2025 年 6 月 6 日
新しい IPv6 クラスターのデュアルスタックパブリックエンドポイントの IPv6 アクセスコントロール	Amazon EKS は、IPv6 CIDR ブロックを追加して、新しい IPv6 クラスターのパブリッククラスターエンドポイントへのアクセスを制御します。以前は、デュアルスタックのクラスターエンドポイントであっても、パブリッククラスターエンドポイントへのトラフィックを許可する IPv4 CIDR ブロックのみを追加できました。	2025 年 6 月 5 日
Amazon EKS プラットフォームバージョンの更新	これはセキュリティの修正および機能強化が行われた新しいプラットフォームバージョンです。これには、Kubernetes 1.32.5、1.31.9、1.30.13 の新しいパッチバージョンが含まれます。	2025 年 5 月 30 日
EKS Hybrid Nodes の新しいクラスターインサイト	Amazon EKS は、ハイブリッドノードの設定を確認する新しいクラスターインサイトを追加します。これらのインサイトチェックでは、オンプレミスノード、ポッド、およびクラスターのリモートネットワーク設定に関する問題について警告します。	2025 年 5 月 29 日

Kubernetes バージョン 1.33	Kubernetes バージョン 1.33 の新しいクラスターとバージョンアップグレードのサポートが追加されました。	2025 年 5 月 29 日
Amazon FSx CSI ドライバーのアドオンサポート	AWS マネジメントコンソール、AWS CLI、API を使用して Amazon FSx CSI ドライバーを管理できるようになりました。	2025 年 5 月 23 日
コンソールで Prometheus スクレイパーを編集する	Amazon EKS コンソールで Amazon Managed Service for Prometheus スクレイパーを編集できるようになりました。	2025 年 5 月 22 日
マネージドポリシーの更新	`AmazonEKSDashboardServiceRolePolicy` ポリシーを追加しました。	2025 年 5 月 21 日
Amazon EKS プラットフォームバージョンの更新	これはセキュリティの修正および機能強化が行われた新しいプラットフォームバージョンです。これらのプラットフォームバージョンには、Kubernetes の新しいパッチのバージョンはありません。	2025 年 5 月 16 日
Amazon FSx for Lustre のパフォーマンスの新規ページ	Amazon FSx for Lustre のパフォーマンスの最適化に関する詳細を記載した、新しいトピックが追加されました。	2025 年 5 月 2 日

Amazon EKS Auto Mode の NodeClass への更新	Auto Mode ノードの NodeClass テンプレートに、フォワードネットワークプロキシの設定が追加されました。これにより、HTTPS プロキシを設定するためのオプションキー advancedNetworking が追加されます。	2025 年 4 月 30 日
ハイブリッドノードの Bottlerocket	Bottlerocket が EKS Hybrid Nodes で利用できるようになりました。	2025 年 4 月 29 日
Amazon EKS プラットフォームバージョンの更新	これはセキュリティの修正および機能強化が行われた新しいプラットフォームバージョンです。これらのプラットフォームバージョンには、Kubernetes の新しいパッチのバージョンはありません。	2025 年 4 月 29 日
ハイブリッドネットワークの新しいコンセプトページ	EKS Hybrid Nodes の概念に関するページを追加しました。オンプレミスとクラウドのネットワークについて、図を使いながら詳細に解説しています。	2025 年 4 月 18 日

AWS マネージドポリシーの更新	`AmazonEKSClusterPolicy` にアクセス許可が追加され、VPC CNI によって作成された Elastic Network Interface を Amazon EKS でクリーンアップできるようになりました。こうした権限付与が必要なのは、VPC CNI が予期せず終了した場合に、残った Elastic Network Interface を EKS によってクリーンアップするためです。	2025 年 4 月 16 日
AWS マネージドポリシーの更新	`AmazonEKSServiceRolePolicy` にアクセス権限を追加して、EKS AI/ML のお客様がデフォルトの EKS クラスターセキュリティグループに Egress ルールを追加できるようにしました。	2025 年 4 月 14 日
Kubernetes バージョン 1.31 が AWS Outposts のローカルクラスターで利用可能に	Kubernetes バージョン 1.31 を使用して AWS Outposts に Amazon EKS ローカルクラスターを作成できるようになりました。	2025 年 4 月 8 日
EKS Hybrid Nodes のノードのヘルス	`1.2.0-eksbuild.1` バージョン以降は、`eks-node-monitoring-agent` をハイブリッドノードで使用できます。`eks-node-monitoring-agent` を Amazon EKS アドオンとして実行することで、健全性に関する問題を検出し表示できます。	2025 年 3 月 31 日

既存のクラスターの EKS Hybrid Nodes	既存のクラスターのハイブリッドノード設定を追加、変更、削除できるようになりました。以前は、クラスターを作成する際にしか、ハイブリッドノード設定を新しいクラスターに追加できませんでした。Amazon EKS Hybrid Nodes ではオンプレミスおよびエッジインフラストラクチャを Amazon EKS クラスターのノードとして使用できます。AWS は Amazon EKS クラスターの AWS ホスト型 Kubernetes コントロールプレーンを管理し、オンプレミス環境またはエッジ環境で実行されるハイブリッドノードを管理します。	2025 年 3 月 31 日
ロールバック: クラスターインサイトによる偶発的なアップグレードの防止	Amazon EKS では、特定のクラスターインサイトの問題が発生した際に `--force` フラグでのクラスターアップグレードが必要となる機能が一時的にロールバックされています。詳細については、GitHub の「Temporary rollback of enforcing upgrade insights on update cluster version」を参照してください。	2025 年 3 月 28 日
Bottlerocket FIPS AMI	Bottlerocket FIPS AMI が、標準のマネージド型ノードグループで利用可能になりました。	2025 年 3 月 27 日

AWS マネージドポリシーの更新	Amazon EKS が Auto Mode で作成された EC2 インスタンスを終了できるよう、AmazonEKSServiceRolePolicy へのアクセス許可を追加しました。	2025 年 2 月 28 日
マネージド型ノードグループの戦略の更新	更新戦略を使用して、マネージド型ノードグループのバージョン更新プロセスを設定できるようになりました。新しいノードを作成する前にノードを終了する 最小限 の更新戦略は、容量に制約のある環境で役立ちます。デフォルトの更新戦略では、既存の動作が継続されます。	2025 年 1 月 27 日
Kubernetes バージョン 1.32	Kubernetes バージョン 1.32 の新しいクラスターとバージョンアップグレードのサポートが追加されました。	2025 年 1 月 23 日
Amazon EKS AWS リージョンの拡張	Amazon EKS が、アジアパシフィック (タイ) リージョン (ap-southeast-7) およびメキシコ (中部) (mx-central-1) AWSリージョンで利用可能になりました。EKS API の EKS Auto Mode と VPC エンドポイントはどちらのリージョンでも利用できません。	2025 年 1 月 14 日

AWS マネージドポリシーの更新	Amazon EBS CSI ドライバーがすべてのスナップショットを復元AmazonEBSCSIDriverPolicy し、EBS ボリュームで高速スナップショット復元 (FSR) 有効にし、ボリュームのタグを変更できるようにする複数のアクセス許可を に追加しました。	2025 年 1 月 13 日
AWS マネージドポリシーの更新	許可を AmazonEKSLoadBalancingPolicy に追加しました。	2024 年 12 月 26 日
クラスターインサイトの更新	Amazon EKS のアップグレードインサイトはクラスターの健全性とバージョンの互換性に関するより多くの問題について警告するようになります。kubelet、kube-proxy 、Amazon EKS アドオンなど、さまざまな Kubernetes と Amazon EKS コンポーネント間の問題を検出できます。	2024 年 12 月 20 日
ノードモニタリングエージェントと自動修復	新しい eks-node-monitoring-agent を Amazon EKS アドオンとして使用して、健全性に関する問題を検出および表示できます。ノードの自動修復を有効にして、問題が検出されたときノードを自動的に置き換えることもできます。	2024 年 12 月 16 日

Amazon EKS Hybrid Nodes	Amazon EKS クラスターに接続されたノードをオンプレミスで実行できるようになりました。Amazon EKS Hybrid Nodes ではオンプレミスおよびエッジインフラストラクチャを Amazon EKS クラスターのノードとして使用できます。AWS は Amazon EKS クラスターの AWS ホスト型 Kubernetes コントロールプレーンを管理し、オンプレミス環境またはエッジ環境で実行されるハイブリッドノードを管理します。	2024 年 12 月 1 日
Amazon EKS Auto Mode	Amazon EKS Auto Mode は、AWS でのコンピューティング、ストレージ、ネットワーキングのための Kubernetes クラスターインフラストラクチャ管理を完全に自動化します。インフラストラクチャの自動プロビジョニング、最適なコンピューティングインスタンスの選択、リソースの動的なスケーリング、継続的なコストの最適化、オペレーティングシステムへのパッチ適用、AWS セキュリティサービスとの統合により、Kubernetes の管理を簡素化します。	2024 年 12 月 1 日

Amazon EKS プラットフォームバージョンの更新	これはセキュリティの修正および機能強化が行われた新しいプラットフォームバージョンです。これには、Kubernetes 1.31.2、1.30.6、1.29.10、1.28.15 の新しいパッチバージョンが含まれます。	2024 年 11 月 22 日
AWS マネージドポリシーの更新	AWSServiceRoleForAmazonEKSNodegroup が中国リージョンとの互換性のために更新されました。	2024 年 11 月 22 日
Kubernetes バージョン 1.30 が AWS Outposts のローカルクラスターで利用可能に	Kubernetes バージョン 1.30 を使用して AWS Outposts に Amazon EKS ローカルクラスターを作成できるようになりました。	2024 年 11 月 21 日
AWS マネージドポリシーの更新	EKS が AWS マネージドポリシー AmazonEKSLocalOutpostClusterPolicy を更新しました。クラスターコントロールプレーンでの AWS Cloud Controller Manager が各ノードが存在するアベイラビリティーゾーンを識別できるように、ec2:DescribeAvailabilityZones 許可を追加しました。	2024 年 11 月 21 日

FIPS 140-3 を使用する Bottlerocket AMI	Bottlerocket AMI は FIPS 140-3 検証済み暗号化モジュールを使用するように事前設定されています。これには Amazon Linux 2023 Kernel Crypto API 暗号化モジュールや AWS-LC 暗号化モジュールなどがあります。	2024 年 11 月 20 日
AWS マネージドポリシーの更新	Amazon EKS マネージドノードグループによって作成されたインスタンスのために `ec2:RebootInstances` を許可するように `AWSServiceRoleForAmazonEKSNodegroup` ポリシーを更新しました。Amazon EC2 リソースについての `ec2:CreateTags` 許可を制限しました。	2024 年 11 月 20 日
オブザーバビリティダッシュボード	オブザーバビリティダッシュボードは問題を迅速に検出、トラブルシューティング、是正するのに役立ちます。また、`AWS/EKS` 名前空間では新しい CloudWatch 出力メトリクスを使用することもできます。	2024 年 11 月 18 日

AWS マネージドポリシーの更新	EKS が AWS マネージドポリシー AmazonEKSServiceRolePolicy を更新しました。EKS アクセスポリシー、ロードバランサーの管理、クラスターリソースの自動クリーンアップの許可を追加しました。	2024 年 11 月 16 日
EKS Pod Identity をサポートするアドオンのための、コンソールでの新しいロールの作成	コンソールを使用して EKS Pod Identity をサポートするアドオンを作成または更新する場合、新しいステップがあります。このステップではアドオンのために適切な名前、ロールポリシー、信頼ポリシーを使用して IAM ロールを自動的に生成できます。	2024 年 11 月 15 日
AWS ローカルゾーン のマネージドノードグループ	マネージドノードグループを AWS ローカルゾーンで作成できるようになりました。	2024 年 11 月 15 日
新しいメトリクスが使用可能に	API グループ metrics.eks.amazonaws.com の下に使用可能な新しいメトリクスがあります。	2024 年 11 月 11 日
AWS マネージドポリシーの更新	EKS が AWS マネージドポリシー AmazonEKSComputePolicy を更新しました。iam:AddRoleToInstanceProfile アクションのリソース許可を更新しました。	2024 年 11 月 7 日

AWS マネージドポリシーの更新	EKS に、新しい AWS マネージドポリシーが追加されました: AmazonEKSComputePolicy	2024 年 11 月 1 日
AWS マネージドポリシーの更新	許可を AmazonEKSClusterPolicy に追加しました。Amazon EKS がトポロジ情報をラベルとしてノードにアタッチできるようにする ec2:DescribeInstanceTopology 許可を追加しました。	2024 年 11 月 1 日
AWS マネージドポリシーの更新	EKS に、新しい AWS マネージドポリシーが追加されました: AmazonEKSBlockStoragePolicy	2024 年 10 月 30 日
AWS マネージドポリシーの更新	EKS に、新しい AWS マネージドポリシーが追加されました: AmazonEKSLoadBalancingPolicy	2024 年 10 月 30 日
AWS マネージドポリシーの更新	Amazon EKS が Amazon CloudWatch にメトリクスを発行できるようにする cloudwatch:PutMetricData 許可を AmazonEKSServiceRolePolicy に追加しました。	2024 年 10 月 29 日
AWS マネージドポリシーの更新	EKS に、新しい AWS マネージドポリシーが追加されました: AmazonEKSNetworkingPolicy	2024 年 10 月 28 日

新しい IPv6 クラスターのデュアルスタックエンドポイント	デュアルスタックの eks-cluster. *region*.api.aws エンドポイントを使用して新しい IPv6 クラスターに接続します。このエンドポイントはこれらのクラスターを記述するときに返されます。IPv4、IPv6、またはデュアルスタック環境の kubectl およびその他の Kubernetes API クライアントはパブリッククラスターまたはプライベートクラスターのこれらのエンドポイントを解決して接続できます。	2024 年 10 月 21 日
AWS マネージドポリシーの更新	autoscaling:ResumeProcesses 、autoscaling:SuspendProcesses および関連するアクセス許可を中国リージョンの AWSServiceRoleForAmazonEKSNodegroup に追加し、EKS 用の Amazon Application Recovery Controller と統合しました。他のリージョンに変更はありません。	2024 年 10 月 21 日
AL2023 高速 AMI	AL2023 に基づいて高速 NVIDIA と AWS Neuron インスタンスを AMI に使用できるようになりました。	2024 年 10 月 11 日

新しいソース形式	レイアウトが変更された新しいソース形式に切り替えました。対処中の一時的で軽微なフォーマットの問題があります。	2024 年 10 月 10 日
AWS マネージドポリシーの更新	AmazonEKSServicePolicy および AmazonEKSServiceRolePolicy にアクセス許可を追加しました。EKS がセキュリティグループ情報を読み取り、関連するタグを更新できるように、ec2:GetSecurityGroupsForVpc および関連するタグの許可を追加しました。	2024 年 10 月 10 日
AWS マネージドポリシーの更新 – 新しいポリシー	EKS に、新しい AWS マネージドポリシーが追加されました。	2024 年 10 月 3 日
Kubernetes バージョン 1.31	Kubernetes バージョン 1.31 の新しいクラスターとバージョンアップグレードのサポートが追加されました。	2024 年 9 月 24 日
AWSマネージドポリシーの更新 – 既存のポリシーへの更新	Amazon EKS において、既存の AWS マネージドポリシーの更新を行いました。	2024 年 8 月 21 日
Kubernetes バージョン 1.29 が AWS Outposts のローカルクラスターで利用可能に	Kubernetes バージョン 1.29 を使用して AWS Outposts に Amazon EKS ローカルクラスターを作成できるようになりました。	2024 年 8 月 20 日

AWS GovCloud (米国) での EKS Pod Identity	Amazon EKS Pod Identity は IAM ロールを Kubernetes サービスアカウントに関連付けます。この機能を使用すると、ノード IAM ロールに拡張されたアクセス許可を提供する必要がなくなります。このようにして、そのノード上の Pod は AWS API を呼び出すことができます。サービスアカウントの IAM ロールとは異なり、EKS Pod Identity は完全に EKS 内部にあるため、OIDC ID プロバイダーは必要ありません。	2024 年 8 月 14 日
シナリオ駆動型コンテンツの更新	ガイド全体でよりシナリオ駆動型になるようにトピックの名前を変更および更新しました。	2024 年 8 月 9 日
Amazon EKS のデュアルスタック VPC インターフェイスエンドポイント	Amazon EKS のデュアルスタック VPC インターフェイスエンドポイントを、IPv4 および IPv6 IP アドレスと DNS 名の両方で作成できるようになりました。	2024 年 8 月 7 日

IPv6 アドレスを持つ Amazon EKS API の新しいデュアルスタックエンドポイント	クラスターを作成および管理するための EKS API、およびクラスターの OIDC 発行者 URL には新しいデュアルスタックエンドポイントがあります。Amazon EKS API の新しい DNS 名は `eks.region.api.aws` であり、これは IPv4 アドレスと IPv6 アドレスに解決されます。新しいクラスターには新しいデュアルスタック OIDC 発行者 URL (`oidc-eks. region.api.aws`) があります。	2024 年 8 月 1 日
マネージド型ノードグループのキャパシティブロック	マネージド型ノードグループにキャパシティブロックを使用できるようになりました。	2024 年 7 月 1 日
デフォルトで Auto Scaling グループのメトリクス収集を有効化	Amazon EKS マネージドノードグループでは追加料金なしで Amazon EC2 Auto Scaling グループのメトリクスがデフォルトで有効になりました。以前はこの機能を有効にするためにいくつかのステップを実行する必要がありました。	2024 年 6 月 28 日
AWS マネージドポリシーの更新 – 既存のポリシーへの更新	Amazon EKS において、既存の AWS マネージドポリシーの更新を行いました。	2024 年 6 月 27 日
Kubernetes バージョン 1.26	Kubernetes バージョン 1.26 が延長サポート対象になりました。	2024 年 6 月 12 日

AMI 情報リファレンスの改善	AMI 情報リファレンス、特に Bottlerocket が改善されました。	2024 年 6 月 12 日
Kubernetes バージョン 1.30	Kubernetes バージョン 1.30 の新しいクラスターとバージョンアップグレードのサポートが追加されました。	2024 年 5 月 23 日
CoreDNS の自動スケーリング	CoreDNS オートスケーラーはノードと CPU コアの数に基づいて、EKS クラスター内の CoreDNS デプロイのレプリカの数を動的に調整します。この機能は CoreDNS v1.9 および EKS リリースバージョン 1.25 以降の最新プラットフォームバージョンで機能します。	2024 年 5 月 14 日
Amazon EKS プラットフォームバージョンの更新	これはセキュリティの修正および機能強化が行われた新しいプラットフォームバージョンです。これには、Kubernetes 1.29.4、1.28.9、1.27.13 の新しいパッチバージョンが含まれます。	2024 年 5 月 14 日
Windows での CloudWatch Container Insights のサポート	Amazon CloudWatch Observability Operator アドオンはクラスター内の Windows ワーカーノードでも Container Insights を許可するようになりました。	2024 年 4 月 10 日

Kubernetes の概念	Kubernetes の概念のトピックが新たに追加されました。	2024 年 4 月 5 日
アクセスと IAM コンテンツを再構築する	認証設定マップ、アクセスエントリ、ポッド ID、IRSA など、アクセスと IAM トピックに関連する既存のページが新しいセクションに移動します。概要の内容を改訂しました。	2024 年 4 月 2 日
Amazon S3 CSI ドライバーの Bottlerocket OS サポート	Mountpoint for Amazon S3 CSI ドライバーが Bottlerocket と互換性を持つようになりました。	2024 年 3 月 13 日
AWS マネージドポリシーの更新 – 既存のポリシーへの更新	Amazon EKS において、既存の AWS マネージドポリシーの更新を行いました。	2024 年 3 月 4 日
Amazon Linux 2023	Amazon Linux 2023 (AL2023) はクラウドアプリケーションに安全かつ安定した高パフォーマンス環境を提供するように設計された、新しい Linux ベースのオペレーティングシステムです。	2024 年 2 月 29 日

Kubernetes 1.29 で EKS Pod Identity と IRSA がサイドカーをサポート	Kubernetes 1.29 ではサイドカーコンテナは Amazon EKS クラスターで使用できます。サイドカーコンテナはサービスアカウントまたは EKS Pod Identity の IAM ロールでサポートされています。サイドカーの詳細については、Kubernetes ドキュメントの「サイドカーコンテナ」を参照してください。	2024 年 2 月 26 日
Kubernetes バージョン 1.29	Kubernetes バージョン 1.29 の新しいクラスターとバージョンアップグレードのサポートが追加されました。	2024 年 1 月 23 日
フルリリース: Kubernetes バージョンに対する Amazon EKS の延長サポート	Kubernetes バージョンに対する延長サポートにより、特定の Kubernetes バージョンを 14 か月よりも長く使用することができます。	2024 年 1 月 16 日

AWS クラウドでの Amazon EKS クラスターヘルスの検出	Amazon EKS はクラスターヘルス内の、Amazon EKS クラスターとクラスターの前提条件のインフラストラクチャの問題を検出します。EKS API ではAWS マネジメントコンソール 内およびクラスター の `health` 内の EKS クラスターの問題が確認できます。これらの問題はコンソールによって検出され表示される問題に追加されます。以前はクラスターヘルスは AWS Outposts 上のローカルクラスターのみで利用可能でした。	2023 年 12 月 28 日
クラスターのインサイト	定期的なチェックに基づいて、クラスターに関する推奨事項を取得できるようになりました。	2023 年 12 月 20 日
Amazon EKS AWS リージョンの拡張	Amazon EKS がカナダ西部 (カルガリー) (`ca-west-1`) AWS リージョンで利用可能になりました。	2023 年 12 月 20 日

IAM ロールとユーザーに、アクセスエントリを使用してクラスターへのアクセス権を付与できるようになりました	アクセスエントリを導入する前は aws-auth ConfigMap にエントリを追加して IAM ロールとユーザーにクラスターへのアクセス権を付与していました。これで、各クラスターにアクセスモードが用意され、スケジュールに従ってアクセスエントリを使用するように切り替えることができるようになりました。モードを切り替えたら、AWS CLI、AWS CloudFormation、AWS SDK にアクセスエントリを追加してユーザーを追加できます。	2023 年 12 月 18 日
Amazon EKS プラットフォームバージョンの更新	これはセキュリティの修正および機能強化が行われた新しいプラットフォームバージョンです。これにはKubernetes 1.28.4、1.27.8、1.26.11、1.25.16 の新しいパッチバージョンが含まれます。	2023 年 12 月 12 日
Mountpoint for Amazon S3 CSI ドライバー	Amazon EKS クラスターで Mountpoint for Amazon S3 CSI ドライバーをインストールできるようになりました。	2023 年 11 月 27 日
クラスターの作成時に Prometheus メトリクスを有効にする	AWS マネジメントコンソール ではクラスターの作成時に Prometheus メトリクスをオンにできるようになりました。Prometheus スクレイパーの詳細は [オブザーバビリティ] タブでも表示できます。	2023 年 11 月 26 日

Amazon EKS Pod Identity	Amazon EKS Pod Identity は IAM ロールを Kubernetes サービスアカウントに関連付けます。この機能を使用すると、ノード IAM ロールに拡張されたアクセス許可を提供する必要がなくなります。このようにして、そのノード上の Pod は AWS API を呼び出すことができます。サービスアカウントの IAM ロールとは異なり、EKS Pod Identity は完全に EKS 内部にあるため、OIDC ID プロバイダーは必要ありません。	2023 年 11 月 26 日
AWSマネージドポリシーの更新 – 既存のポリシーへの更新	Amazon EKS において、既存の AWS マネージドポリシーの更新を行いました。	2023 年 11 月 26 日
CSI スナップショットコントローラー	CSI スナップショットコントローラーをインストールして、Amazon EBS CSI ドライバーなどの互換性のある CSI ドライバーで使用できるようになりました。	2023 年 11 月 17 日
ADOT オペレータートピックリライト	ADOT Operator セクションの Amazon EKS アドオンサポートは AWS Distro for OpenTelemetry のドキュメントと重複していました。古い情報や一貫性のない情報を減らすために、残りの重要な情報をそのリソースに移行しました。	2023 年 11 月 14 日

Prometheus メトリクスの CoreDNS EKS アドオンサポート	CoreDNS 用の EKS アドオンの v1.10.1-eksbuild.5 、v1.9.3-eksbuild.9 、v1.8.7-eksbuild.8 バージョンでは、CoreDNS がメトリクスを公開したポートが kube-dns サービス内で公開されます。これにより、モニタリングシステムに CoreDNS メトリクスを組み込むことが容易になります。	2023 年 11 月 10 日
Amazon EKS CloudWatch オブザーバビリティオペレーターアドオン	Amazon EKS CloudWatch オブザーバビリティオペレーターページを追加しました。	2023 年 11 月 6 日
米国東部 (オハイオ) のセルフマネージド型 P5 インスタンスのキャパシティブロック	米国東部 (オハイオ) では米国東部 (オハイオ) のセルフマネージド型 P5 インスタンスにキャパシティブロックを使用できるようになりました。	2023 年 10 月 31 日

クラスターでのサブネットおよびセキュリティグループの変更のサポート	クラスターを更新して、クラスターが使用するサブネットとセキュリティグループを変更できます。AWS マネジメントコンソール、AWS CLI の最新バージョン、AWS CloudFormation、および `eksctl` バージョン `v0.164.0-rc.0` 以降から更新できます。クラスターバージョンを正常にアップグレードするにはこれを実行して、サブネットに利用可能な IP アドレスを増やす必要がある場合があります。	2023 年 10 月 24 日
クラスターロールおよびマネージドノードグループロールでのカスタマーマネージド AWS アイデンティティとアクセス管理 ポリシーのサポート	クラスターロールでは AmazonEKSClusterPolicy AWS マネージドポリシーの代わりにカスタム IAM ポリシーを使用できます。また、マネージドノードグループのノードロールでは AmazonEKSWorkerNodePolicy AWS マネージドポリシーの代わりにカスタム IAM ポリシーを使用できます。これにより、厳格なコンプライアンス要件を満たす最小特権でポリシーを作成できます。	2023 年 10 月 23 日
eksctl インストールへのリンクの修正	ページが移動された後の eksctl のインストールリンクを修正しました。	2023 年 10 月 6 日

プレビューリリース: Kubernetes バージョンに対する Amazon EKS の延長サポート	Kubernetes バージョンに対する延長サポートにより、特定の Kubernetes バージョンを 14 か月よりも長く使用することができます。	2023 年 10 月 4 日
AWS App Mesh 統合へのリファレンスを削除	Amazon EKS と AWS App Mesh の統合は引き続き App Mesh の既存のお客様のみを対象としています。	2023 年 9 月 29 日
Kubernetes バージョン 1.28	Kubernetes バージョン 1.28 の新しいクラスターとバージョンアップグレードのサポートが追加されました。	2023 年 9 月 26 日
既存のクラスターが、Amazon VPC CNI plugin for Kubernetes で Kubernetes ネットワークポリシーの適用をサポート	サードパーティーのソリューションの代わりに、既存のクラスターにある Kubernetes のネットワークポリシーと Amazon VPC CNI plugin for Kubernetes を使用できます。	2023 年 9 月 15 日
CoreDNS Amazon EKS アドオンが PDB の変更をサポート	バージョン v1.9.3-eksbuild.7 以降および v1.10.1-eksbuild.4 以降では CoreDNS の EKS アドオンの PodDisruptionBudget を変更できます。	2023 年 9 月 15 日
Amazon EKS での共有サブネットのサポート	共有サブネット内に Amazon EKS クラスターを作成するための、新しい「共有サブネットの要件と考慮事項」。	2023 年 9 月 7 日

「Amazon EKS とは」の更新	新規トピック「一般的なユースケース」と「アーキテクチャ」の追加。その他に複数のトピックの更新。	2023 年 9 月 6 日
Amazon VPC CNI plugin for Kubernetes での Kubernetes ネットワークポリシーの適用	サードパーティーのソリューションの代わりに、Kubernetes のネットワークポリシーと Amazon VPC CNI plugin for Kubernetes を使用できます。	2023 年 8 月 29 日
Amazon EKS AWS リージョンの拡張	Amazon EKS がイスラエル (テルアビブ) (`il-central-1`) AWS リージョンで利用可能になりました。	2023 年 8 月 1 日
設定可能な Fargate 用エフェメラルストレージ	Amazon EKS Fargate で実行される各 Pod 用エフェメラルストレージの総量を増やすことができます。	2023 年 7 月 31 日
Amazon EFS CSI ドライバーのアドオンサポート	AWS マネジメントコンソール、AWS CLI、API を使用して Amazon EFS CSI ドライバーを管理できるようになりました。	2023 年 7 月 26 日
AWS マネージドポリシーの更新 – 新しいポリシー	Amazon EKS に、新しい AWS マネージドポリシーが追加されました。	2023 年 7 月 26 日
1.27、1.26、1.25、1.24 向けの Kubernetes バージョンアップデートが、AWS Outposts のローカルクラスターで利用可能に	1.27.3、1.26.6、1.25.11、1.24.15 向けの Kubernetes バージョンアップデートが、AWS アウトポストのローカルクラスターで利用できるようになりました。	2023 年 7 月 20 日

Windows ノードの IP プレフィックスサポート	ノードに IP プレフィックスを割り当てると、個別のセカンダリ IP アドレスをノードに割り当てる場合よりもはるかに多くの数の Pod をノードでホストできます。	2023 年 7 月 6 日
Amazon FSx for OpenZFS CSI ドライバー	Amazon EKS クラスターで Amazon FSx for OpenZFS CSI ドライバーをインストールできるようになりました。	2023 年 6 月 30 日
IPv4 クラスター内の Linux ノード上の Pod が、IPv6 エンドポイントと通信できるようになりました。	IPv6 アドレスがノードに割り当てられると、Pod の IPv4 アドレスは実行されているノードの IPv6 アドレスにネットワークアドレス変換されます。	2023 年 6 月 19 日
AWS GovCloud (米国) リージョンの Windows マネージド型ノードグループ	AWS GovCloud (US) リージョンでは、Amazon EKS マネージドノード型グループが Windows コンテナを実行できるようになりました。	2023 年 5 月 30 日
Kubernetes バージョン 1.27	Kubernetes バージョン 1.27 の新しいクラスターとバージョンアップグレードのサポートが追加されました。	2023 年 5 月 24 日
Kubernetes バージョン 1.26	Kubernetes バージョン 1.26 の新しいクラスターとバージョンアップグレードのサポートが追加されました。	2023 年 4 月 11 日
ドメインレス gMSA	Windows Pod でドメインレス gMSA を使用できるようになりました。	2023 年 3 月 27 日

Amazon EKS AWS リージョンの拡張	Amazon EKS がアジアパシフィック (メルボルン) (ap-southeast-4) AWS リージョンで利用可能になりました。	2023 年 3 月 10 日
Amazon File Cache CSI ドライバー	Amazon EKS クラスターに Amazon File Cache CSI ドライバーをインストールできるようになりました。	2023 年 3 月 3 日
Kubernetes バージョン 1.25 が AWS Outposts のローカルクラスターで利用可能に	Kubernetes バージョン 1.22 ~ 1.25 を使用して、Outpost に Amazon EKS ローカルクラスターを作成できるようになりました。	2023 年 3 月 1 日
Kubernetes バージョン 1.25	Kubernetes バージョン 1.25 の新しいクラスターとバージョンアップグレードのサポートが追加されました。	2023 年 2 月 22 日
AWSマネージドポリシーの更新 – 既存のポリシーへの更新	Amazon EKS において、既存の AWS マネージドポリシーの更新を行いました。	2023 年 2 月 7 日
Amazon EKS AWS リージョンの拡張	Amazon EKS は現在、アジアパシフィック (ハイデラバード) (ap-south-2)、欧州 (チューリッヒ) (eu-central-2)、欧州 (スペイン) (eu-south-2) のAWS リージョンで利用可能になりました。	2023 年 2 月 6 日

AWS Outposts 上のローカルクラスターで Kubernetes バージョン 1.21 ~ 1.24 が利用可能に。	Kubernetes バージョン 1.21 ~ 1.24 を使用して、Outpost に Amazon EKS ローカルクラスターを作成できるようになりました。以前は 1.21 バージョンのみが使用可能でした。	2023 年 1 月 17 日
Amazon EKS が AWS プライベートリンク をサポート	AWS プライベートリンク を使用して、VPC と Amazon EKS 間にプライベート接続を作成できます。	2022 年 12 月 16 日
マネージド型ノードグループの Windows サポート	Amazon EKS マネージド型ノードグループに、Windows を使用できるようになりました。	2022 年 12 月 15 日
AWS マーケットプレイス で独立系ソフトウェアベンダーの Amazon EKS アドオンの提供を開始	AWS マーケットプレイス を通じて、独立系ソフトウェアベンダーの Amazon EKS アドオンを閲覧し、サブスクライブできるようになりました。	2022 年 11 月 28 日
AWSマネージドポリシーの更新 – 既存のポリシーへの更新	Amazon EKS において、既存の AWS マネージドポリシーの更新を行いました。	2022 年 11 月 17 日
Kubernetes バージョン 1.24	Kubernetes バージョン 1.24 の新しいクラスターとバージョンアップグレードのサポートが追加されました。	2022 年 11 月 15 日
Amazon EKS AWS リージョンの拡張	Amazon EKS が中東 (UAE) (me-central-1) AWS リージョンで利用可能になりました。	2022 年 11 月 3 日

AWSマネージドポリシーの更新 – 既存のポリシーへの更新	Amazon EKS において、既存の AWS マネージドポリシーの更新を行いました。	2022 年 10 月 24 日
AWSマネージドポリシーの更新 – 既存のポリシーへの更新	Amazon EKS において、既存の AWS マネージドポリシーの更新を行いました。	2022 年 10 月 20 日
AWS Outposts のローカルクラスターが利用可能	アウトポスト に Amazon EKS ローカルクラスターを作成できるようになりました。	2022 年 9 月 19 日
Fargate vCPU ベースのクォータ	Fargate は、Pod ベースのクォータから vCPU ベースのクォータに移行しています。	2022 年 9 月 8 日
AWSマネージドポリシーの更新 – 既存のポリシーへの更新	Amazon EKS において、既存の AWS マネージドポリシーの更新を行いました。	2022 年 8 月 31 日
コストモニタリング	Amazon EKS が Kubecost をサポートするようになりました。これにより、Pod、ノード、名前空間、ラベルなどの Kubernetes リソースごとにコストを分類して監視することができます。	2022 年 8 月 24 日
AWS マネージドポリシーの更新 – 新しいポリシー	Amazon EKS に、新しい AWS マネージドポリシーが追加されました。	2022 年 8 月 24 日
AWS マネージドポリシーの更新 – 新しいポリシー	Amazon EKS に、新しい AWS マネージドポリシーが追加されました。	2022 年 8 月 23 日

請求用のタグリソース	すべてのクラスターで、aws:eks:cluster-name 生成のコスト配分タグがサポートされるようになりました。	2022 年 8 月 16 日
Fargate プロファイルのワイルドカード	名前空間、ラベルキー、およびラベル値のセレクター基準で、Fargate プロファイルのワイルドカードがサポートされるようになりました。	2022 年 8 月 16 日
Kubernetes バージョン 1.23	Kubernetes バージョン 1.23 の新しいクラスターとバージョンアップグレードのサポートが追加されました。	2022 年 8 月 11 日
に Kubernetes リソースを表示するAWS マネジメントコンソール	AWS マネジメントコンソールを使用してクラスターにデプロイされた Kubernetes リソースの情報を表示できるようになりました。	2022 年 5 月 3 日
Amazon EKS AWS リージョンの拡張	Amazon EKS がアジアパシフィック (ジャカルタ) (ap-southeast-3) AWS リージョンで利用可能になりました。	2022 年 5 月 2 日
オブザーバビリティページと ADOT アドオンのサポート	オブザーバビリティページと、AWS Distro for OpenTelemetry (ADOT) を追加。	2022 年 4 月 21 日
Kubernetes バージョン 1.22	Kubernetes バージョン 1.22 の新しいクラスターとバージョンアップグレードのサポートが追加されました。	2022 年 4 月 4 日

AWS マネージドポリシーの更新 – 新しいポリシー	Amazon EKS に、新しい AWS マネージドポリシーが追加されました。	2022 年 4 月 4 日
Fargate Pod のパッチ適用の詳細を追加	Fargate Pod をアップグレードする場合、Amazon EKS はまず Pod の中断予算に基づいて Pod を削除しようとします。Pod が削除される前に、失敗したエビクションに対応するイベントルールを作成できます。	2022 年 4 月 1 日
フルリリース: Amazon EBS CSI ドライバーのアドオンサポート	AWS マネジメントコンソール、AWS CLI、API を使用して Amazon EBS CSI ドライバーを管理できるようになりました。	2022 年 3 月 31 日
AWS Outposts コンテンツの更新	AWS アウトポスト で Amazon EKS クラスターをデプロイする手順。	2022 年 3 月 22 日
AWSマネージドポリシーの更新 – 既存のポリシーへの更新	Amazon EKS において、既存の AWS マネージドポリシーの更新を行いました。	2022 年 3 月 21 日
Windows `containerd` サポート	Windows ノードの `containerd` ランタイムを選択できるようになりました。	2022 年 3 月 14 日
Amazon EKS コネクトor の考慮事項をセキュリティドキュメントに追加	接続されたクラスターに関連する責任共有モデルについて説明しています。	2022 年 2 月 25 日

Pod とサービスへの IPv6 アドレスの割り当て	Pod とサービスに IPv6 アドレスを割り当てる 1.21 以降のクラスターを作成できるようになりました。	2022 年 1 月 6 日
AWSマネージドポリシーの更新 – 既存のポリシーへの更新	Amazon EKS において、既存の AWS マネージドポリシーの更新を行いました。	2021 年 12 月 13 日
プレビューリリース: Amazon EBS CSI ドライバーのアドオンサポート	AWS マネジメントコンソール、AWS CLI、および API を使用して Amazon EBS CSI ドライバーを管理できるようになりました。	2021 年 12 月 9 日
Karpenter オートスケーラーのサポート	Karpenter オープンソースプロジェクトを使用してノードを自動スケールできるようになりました。	2021 年 11 月 29 日
Fargate ログ記録での Fluent Bit Kubernetes フィルターのサポート	Fargate ログ記録で Fluent Bit Kubernetes フィルターを使用できるようになりました。	2021 年 11 月 10 日
コントロールプレーンで利用可能な Windows サポート	Windows サポートがコントロールプレーンで利用可能になりました。データプレーンで有効にする必要はなくなりました。	2021 年 11 月 9 日

マネージド型ノードグループの AMI タイプとして追加された Bottlerocket	以前は、Bottlerocket はセルフマネージド型のノードオプションとしてのみ利用可能でした。マネージドノードグループとして設定できるようになったため、ノードのコンプライアンス要件を満たすために必要な労力が軽減されます。	2021 年 10 月 28 日
DL1 ドライバーのサポート	カスタム Amazon Linux AMI が、Amazon Linux 2 の深層学習ワークロードをサポートするようになりました。この有効化により、一般的なオンプレミスまたはクラウドのベースライン設定が可能になります。	2021 年 10 月 25 日
VT1 ビデオのサポート	カスタム Amazon Linux AMI では一部のディストリビューションで VT1 がサポートされるようになりました。この有効化により、Amazon EKS クラスター上の Xilinx U30 デバイスがアドバタイズされます。	2021 年 9 月 13 日
Amazon EKS コネクトor が利用可能	Amazon EKS Connector を使用して、準拠する Kubernetes クラスターを AWS に登録および接続し、Amazon EKS コンソールで視覚化できます。	2021 年 9 月 8 日

Amazon EKS Anywhere が利用可能	Amazon EKS Anywhere はオンプレミスの Kubernetes クラスターの作成と運用に使用できる Amazon EKS の新しいデプロイオプションです。	2021 年 9 月 8 日
Amazon FSx for NetApp ONTAP CSI ドライバ	Amazon FSx for NetApp ONTAP CSI ドライバーの概要と、その他のリファレンスへのリンクを提供するトピックを追加しました。	2021 年 9 月 2 日
マネージド型ノードグループは、ノードの Amazon EKS 推奨最大 Pod を自動計算するようになりました	マネージド型ノードグループは、起動テンプレートなしで、または AMI ID を指定していない起動テンプレートを使用して、デプロイするノードの Amazon EKS 最大 Pod を自動的に計算するようになりました。	2021 年 8 月 30 日
Amazon EKS アドオンソフトウェアを削除せずに、アドオン設定の Amazon EKS 管理を削除してください	クラスターからアドオンソフトウェアを削除せずに Amazon EKS アドオンを削除できるようになりました。	2021 年 8 月 20 日
Multus を使用したマルチホーム Pod の作成	ポッドに対し、Multus を使用しながら複数のネットワークインターフェースを追加できるようになりました。	2021 年 8 月 2 日
Linux Amazon EC2 ノードに IP アドレスを追加する	Linux Amazon EC2 ノードに大幅に多くの IP アドレスを追加できるようになりました。つまり、各ノードでより高い密度の Pod を実行できるということです。	2021 年 7 月 27 日

containerd ランタイムブートストラップ	Amazon EKS に最適化されアクセラレートされた Amazon Linux Amazon マシンイメージ (AMI) に、Amazon EKS に最適化された Bottlerocket AMI で containerd ランタイムの有効化に使用できる、ブートストラップフラグが含まれるようになりました。このフラグは、Kubernetes でサポートされているすべての AMI のバージョンで使用できます。	2021 年 7 月 19 日
Kubernetes バージョン 1.21	Kubernetes バージョン 1.21 のサポートを追加しました。	2021 年 7 月 19 日
管理ポリシーのトピックを追加	Amazon EKS IAM 管理ポリシーと、2021 年 6 月 17 日以降に行われたすべての変更のリスト。	2021 年 6 月 17 日
Fargate での Pod のセキュリティグループの使用	Amazon EC2 ノードに加えて、Fargate でも Pod のセキュリティグループを使用できるようになりました。	2021 年 6 月 1 日
CoreDNS と kube-proxy Amazon EKS アドオンの追加	Amazon EKS は、クラスターの CoreDNS および kube-proxy Amazon EKS アドオンの管理もお手伝いできるようになりました。	2021 年 5 月 19 日
Kubernetes バージョン 1.20	Kubernetes バージョン 1.20 の新しいクラスターとバージョンアップグレードのサポートが追加されました。	2021 年 5 月 18 日

AWS Load Balancer Controller 2.2.0 のリリース	AWS ロードバランサーコントローラーを使用して Elastic Load Balancer を作成できるようなりました。インスタンスまたは IP ターゲットを使用します。	2021 年 5 月 14 日
マネージド型ノードグループのノードテイント	Amazon EKS では、マネージド型ノードグループへのノードテイントの追加がサポートされるようになりました。	2021 年 5 月 11 日
既存クラスターのシークレット暗号化	Amazon EKS では既存クラスターへのシークレット暗号化の追加がサポートされるようになりました。	2021 年 2 月 26 日
Kubernetes バージョン 1.19	Kubernetes バージョン 1.19 の新しいクラスターとバージョンアップグレードのサポートが追加されました。	2021 年 2 月 16 日
Amazon EKS では、バージョン 1.16 以降のクラスターに対してユーザーを認証する方法として OpenID Connect (OIDC) ID プロバイダーがサポートされるようになりました。	OIDC ID プロバイダーは AWS アイデンティティとアクセス管理 (IAM) と併用、または代替として使用できます。	2021 年 2 月 12 日
でのノードとワークロードのリソースの表示AWS マネジメントコンソール	AWS マネジメントコンソールで、マネージド型ノード、セルフマネージド型ノード、Fargate ノード、およびデプロイされた Kubernetes ワークロードに関する詳細を表示できるようになりました。	2020 年 12 月 1 日

マネージド型ノードグループへのスポットインスタンスタイプのデプロイ	複数のスポットまたはオンデマンドインスタンスタイプを、マネージド型ノードグループにデプロイできるようになりました。	2020 年 12 月 1 日
Amazon EKS がクラスターの特定のアドオンを管理できるようになりました	アドオンをお客様自身で管理することもできますが、Amazon EKS API を介してアドオンの起動とバージョンを Amazon EKS に制御させることもできます。	2020 年 12 月 1 日
複数の Ingress 間で ALB を共有する	AWS Application Load Balancer (ALB) を複数の Kubernetes Ingress 間で共有できるようになりました。以前は、各 Ingress に個別の ALB をデプロイする必要がありました。	2020 年 10 月 23 日
NLB IP ターゲットのサポート	IP ターゲットを使用して、Network Load Balancer (NLB) をデプロイできるようになりました。これにより、NLB を使用して、Fargate Pod へのネットワークトラフィック、および Amazon EC2 ノードで実行されている Pod への直接のネットワークトラフィックの負荷分散ができます。	2020 年 10 月 23 日
Kubernetes バージョン 1.18	Kubernetes バージョン 1.18 の新しいクラスターとバージョンアップグレードのサポートが追加されました。	2020 年 10 月 13 日

Kubernetes サービス IP アドレス割り当て用のカスタム CIDR ブロックを指定します。	Kubernetes がサービス IP アドレスの割り当てるカスタム CIDR ブロックを指定できるようになりました。	2020 年 9 月 29 日
セキュリティグループを個別の Pod に割り当てる	多くの Amazon EC2 インスタンスタイプで実行されている個別の Pod の一部に、異なるセキュリティグループを関連付けることができるようになりました。	2020 年 9 月 9 日
ノードに Bottlerocketをデプロイする	Bottlerocket を実行中のノードをデプロイできるようになりました。	2020 年 8 月 31 日
Arm ノードを起動する機能は一般的に利用可能です	マネージド型ノードグループ、およびセルフマネージド型ノードグループで、Arm ノードを起動できるようになりました。	2020 年 8 月 17 日
マネージド型ノードグループの起動テンプレートとカスタム AMI	Amazon EC2 起動テンプレートを使用して、マネージド型ノードグループをデプロイできるようになりました。起動テンプレートで、カスタム AMI を指定できます。	2020 年 8 月 17 日
AWS Fargate の EFS サポート	AWS Fargate で Amazon EFS を使用できるようになりました。	2020 年 8 月 17 日

Amazon EKS プラットフォームバージョンの更新	これはセキュリティの修正および機能強化が行われた新しいプラットフォームバージョンです。これにはバージョン 1.15 以降で Network Load Balancer を使用する場合、LoadBalancer タイプのサービスに対する UDP サポートが含まれます。詳細については、GitHub で「Allow UDP for AWS Network Load Balancer」(Network Load Balancer に UDP を許可する) という問題を参照してください。	2020 年 8 月 12 日
Amazon EKS AWS リージョンの拡張	Amazon EKS がアフリカ (ケープタウン) (af-south-1) および欧州 (ミラノ) (eu-south-1) AWS リージョンで利用可能になりました。	2020 年 8 月 6 日
Fargate 使用状況メトリクス	AWS Fargate は、Fargate オンデマンドリソースのアカウント使用状況を可視化する CloudWatch 使用状況メトリクスを提供します。	2020 年 8 月 3 日
Kubernetes バージョン 1.17	Kubernetes バージョン 1.17 の新しいクラスターとバージョンアップグレードのサポートが追加されました。	2020 年 7 月 10 日

Kubernetes 用 App Mesh コントローラーを使用して、Kubernetes 内から App Mesh リソースを作成し管理する	Kubernetes 内から App Mesh リソースを作成し管理できます。また、コントローラーは Envoy プロキシと init コンテナを、デプロイする Pod に自動的に挿入します。	2020 年 6 月 18 日
Amazon EKS が Amazon EC2 Inf1 ノードをサポートするようになりました	Amazon EC2 Inf1 ノードをクラスターに追加できます。	2020 年 6 月 4 日
Amazon EKS AWS リージョンの拡張	Amazon EKS が AWS GovCloud (米国東部) (`us-gov-east-1`) および AWS GovCloud (米国西部) (`us-gov-west-1`) AWS リージョンで利用可能になりました。	2020 年 5 月 13 日
Kubernetes 1.12 は Amazon EKS でサポートされなくなりました	Kubernetes バージョン 1.12 は Amazon EKS でサポートされなくなりました。サービスの中断を防ぐため、すべての 1.12 クラスターをバージョン 1.13 以降に更新してください。	2020 年 5 月 12 日
Kubernetes バージョン 1.16	Kubernetes バージョン 1.16 の新しいクラスターとバージョンアップグレードのサポートが追加されました。	2020 年 4 月 30 日
サービスにリンクされたロール AWSServiceRoleForAmazonEKS を追加	AWSServiceRoleForAmazonEKS サービスリンクロールを追加しました。	2020 年 4 月 16 日

Kubernetes バージョン 1.15	Kubernetes バージョン 1.15 の新しいクラスターとバージョンアップグレードのサポートが追加されました。	2020 年 3 月 10 日
Amazon EKS AWS リージョンの拡張	Amazon EKS が北京 (cn-north-1) および寧夏 (cn-northwest-1) AWS リージョンで利用可能になりました。	2020 年 2 月 26 日
FSx for Lustre CSI ドライバー	Kubernetes 1.14 Amazon EKS クラスターに FSx for Lustre CSI ドライバーをインストールするトピックを追加しました。	2019 年 12 月 23 日
クラスターのパブリックアクセスエンドポイントへのネットワークアクセスを制限	今回の更新では、Amazon EKS を使用して、Kubernetes API サーバーのパブリックアクセスエンドポイントと通信できる CIDR 範囲を制限できるようになりました。	2019 年 12 月 20 日
VPC 外部からクラスターのプライベートアクセスエンドポイントアドレスを解決	今回の更新では、Amazon EKS を使用して、VPC 外から Kubernetes API サーバーのプライベートアクセスエンドポイントを解決できるようになりました。	2019 年 12 月 13 日
(ベータ) Amazon EC2 A1 Amazon EC2 インスタンスノード	Amazon EKS クラスターに登録する Amazon EC2 A1 Amazon EC2 インスタンスノードを起動します。	2019 年 12 月 4 日

AWS Outposts でのクラスターの作成	Amazon EKS ではAWS アウトポスト でのクラスターの作成がサポートされるようになりました。	2019 年 12 月 3 日
Amazon EKS の AWS Fargate	Amazon EKS Kubernetes クラスターで、Fargate での Pod の実行がサポートされるようになりました。	2019 年 12 月 3 日
Amazon EKS AWS リージョンの拡張	Amazon EKS がカナダ (中部) (`ca-central-1`) AWS リージョンで使用可能になりました。	2019 年 11 月 21 日
マネージドノードグループ	Amazon EKS マネージド型ノードグループは、Amazon EKS Kubernetes クラスターのノード (Amazon EC2 インスタンス) のプロビジョニングとライフサイクル管理を自動化します。	2019 年 11 月 18 日
Amazon EKS プラットフォームバージョンの更新	CVE-2019-11253に対応する新しいプラットフォームバージョン	2019 年 11 月 6 日
Kubernetes `1.11` は Amazon EKS でサポートされなくなりました	Kubernetes バージョン `1.11` は Amazon EKS でサポートされなくなりました。サービスの中断を防ぐため、すべての `1.11` クラスターをバージョン `1.12` 以降に更新してください。	2019 年 11 月 4 日

Amazon EKS AWS リージョンの拡張	Amazon EKS が南米 (サンパウロ) (sa-east-1) AWS リージョンで利用可能になりました。	2019 年 10 月 16 日
Windows サポート	Kubernetes バージョン 1.14 を実行する Amazon EKS クラスターが Windows ワークロードをサポートするようになりました。	2019 年 10 月 7 日
自動スケーリング	Amazon EKS クラスターでサポートされる異なるタイプの Kubernetes 自動スケーリングに関する章を追加しました。	2019 年 9 月 30 日
Kubernetes ダッシュボードの更新	ベータ 2.0 バージョンを使用するため、Amazon EKS クラスターに Kubernetes ダッシュボードをインストールするトピックを更新しました。	2019 年 9 月 28 日
Amazon EFS CSI ドライバー	Kubernetes 1.14 Amazon EKS クラスターに Amazon EFS CSI ドライバーをインストールするトピックを追加しました。	2019 年 9 月 19 日
Amazon EKS 最適化された AMI ID の Amazon EC2 Systems Manager パラメータ	Amazon EC2 Systems Manager パラメータを使用して、Amazon EKS 最適化された AMI ID を取得するトピックを追加しました。このパラメータを使用すると、AMI ID を検索する必要がなくなります。	2019 年 9 月 18 日

Amazon EKS リソースのタグ付け	Amazon EKS クラスターのタグ付けを管理できます。	2019 年 9 月 16 日
Amazon EBS CSI ドライバー	Kubernetes 1.14 Amazon EKS クラスターに Amazon EBS CSI ドライバーをインストールするトピックを追加しました。	2019 年 9 月 9 日
新しい Amazon EKS 最適化 AMI に CVE-2019-9512 および CVE-2019-9514 用のパッチを適用	Amazon EKS は CVE-2019-9512 および CVE-2019-9514 に対応するため、Amazon EKS 最適化された AMI を更新しました。	2019 年 9 月 6 日
Amazon EKS の Kubernetes 1.11 の廃止を発表	Amazon EKS は 2019 年 11 月 4 日に Kubernetes バージョン 1.11 のサポートを終了しました。	2019 年 9 月 4 日
Kubernetes バージョン 1.14	Kubernetes バージョン 1.14 の新しいクラスターとバージョンアップグレードのサポートが追加されました。	2019 年 9 月 3 日
サービスアカウントの IAM ロール	Amazon EKS クラスターのサービスアカウントの IAM ロールを使用すると、IAM ロールを Kubernetes サービスアカウントに関連付けることができます。この機能を使用すると、ノード IAM ロールに拡張されたアクセス許可を提供する必要がなくなります。このようにして、そのノード上の Pod は AWS API を呼び出すことができます。	2019 年 9 月 3 日

Amazon EKS AWS リージョンの拡張	Amazon EKS が中東 (バーレーン) (me-south-1) AWS リージョンで利用可能になりました。	2019 年 8 月 29 日
Amazon EKS プラットフォームバージョンの更新	CVE-2019-9512 および CVE-2019-9514 に対応する新しいプラットフォームバージョン。	2019 年 8 月 28 日
Amazon EKS プラットフォームバージョンの更新	CVE-2019-11247 および CVE-2019-11249に対応する新しいプラットフォームバージョン。	2019 年 8 月 5 日
Amazon EKS リージョンの拡張	Amazon EKS がアジアパシフィック (香港) (ap-east-1) AWS リージョンで利用可能になりました。	2019 年 7 月 31 日
Kubernetes 1.10 は Amazon EKS でサポートされなくなりました	Kubernetes バージョン 1.10 は Amazon EKS でサポートされなくなりました。サービスの中断を防ぐため、すべての 1.10 クラスターをバージョン 1.11 以降に更新してください。	2019 年 7 月 30 日
ALB イングレスコントローラー に関するトピックの追加	AWS ALB Ingress Controller for Kubernetes はイングレスリソースが作成されたときに ALB を作成するコントローラーです。	2019 年 7 月 11 日
新しい Amazon EKS 最適化された AMI	不要な kubectl バイナリを AMI から削除します。	2019 年 7 月 3 日

Kubernetes バージョン 1.13	Kubernetes バージョン 1.13 の新しいクラスターとバージョンアップグレードのサポートが追加されました。	2019 年 6 月 18 日
新しい Amazon EKS 最適化 AMI に AWS-2019-005 用のパッチを適用	Amazon EKS は Amazon EKS 最適化 AMI を更新して、link:security/security-bulletins/AWS-2019-005/[AWS-2019-005、type="marketing"] で説明されている脆弱性に対処しました：	2019 年 6 月 17 日
Amazon EKS での Kubernetes 1.10 のサポートの停止を発表	Amazon EKS は、2019 年 7 月 22 日に Kubernetes バージョン 1.10 のサポートを停止しました。	2019 年 5 月 21 日
Amazon EKS プラットフォームバージョンの更新	kubelet 証明書のカスタム DNS 名をサポートし、etcd パフォーマンスを向上させる Kubernetes 1.11 および 1.10 クラスター用の新しいプラットフォームバージョン。	2019 年 5 月 21 日
の開始方法eksctl	この入門ガイドは、eksctl を使用して Amazon EKS を開始するために必要な、すべてのリソースの作成に役立ちます。これは、Amazon EKS で Kubernetes クラスターを作成および管理するためのシンプルなコマンドラインユーティリティです。	2019 年 5 月 10 日

AWS CLI の get-token コマンド	aws eks get-token コマンドが AWS CLI に追加されました。クラスター API サーバー通信用のクライアントセキュリティトークンを作成するために、AWS IAM Authenticator for Kubernetes をインストールする必要がなくなりました。この新しい機能を使用するにはインストール済みの AWS CLI を最新バージョンにアップグレードしてください。詳細については、AWS コマンドラインインターフェイスユーザーガイドの「AWS コマンドラインインターフェイスのインストール」を参照してください。	2019 年 5 月 10 日
Amazon EKS プラットフォームバージョンの更新	kubelet 証明書のカスタム DNS 名をサポートし、etcd のパフォーマンスを向上させる Kubernetes 1.12 クラスター用の新しいプラットフォームバージョン。これにより、ノードの kubelet デーモンによって、数秒ごとに新しい証明書が要求されるバグが修正されました。	2019 年 5 月 8 日
Prometheus チュートリアル	Amazon EKS クラスターへの Prometheus のデプロイのトピックが追加されました。	2019 年 4 月 5 日

Amazon EKS コントロールプレーンのログ記録	今回の更新により、監査および診断ログを Amazon EKS コントロールプレーンから直接取得できるようになりました。これらの CloudWatch ログはクラスターをセキュリティで保護および実行するための参照としてアカウントで使用できます。	2019 年 4 月 4 日
Kubernetes バージョン 1.12	Kubernetes バージョン 1.12 の新しいクラスターとバージョンアップグレードのサポートが追加されました。	2019 年 3 月 28 日
App Mesh 入門ガイドを追加しました	App Mesh と Kubernetes の入門用ドキュメントが追加されました。	2019 年 3 月 27 日
Amazon EKS API サーバーエンドポイントのプライベートアクセス	Amazon EKS クラスターの Kubernetes API サーバーエンドポイントのパブリックアクセスを無効にする方法に関するドキュメントが追加されました。	2019 年 3 月 19 日
Kubernetes メトリクスサーバーのインストールに関するトピックの追加	Kubernetes メトリクスサーバーは、クラスター内のリソース使用状況データを集約します。	2019 年 3 月 18 日
関連するオープンソースプロジェクトのリストの追加	これらのオープンソースプロジェクトは、Amazon EKS によって管理されているクラスターを含め、AWS で実行されている Kubernetes クラスターの機能を拡張します。	2019 年 3 月 15 日

Helm のローカルインストールに関するトピックの追加	Kubernetes 用の `helm` パッケージマネージャーを使用すると、Kubernetes クラスターにアプリケーションをインストールし、管理できます。このトピックでは `helm` および `tiller` バイナリをローカルにインストールして実行する方法について説明します。これにより、ローカルシステムで Helm CLI を使用してチャートをインストールおよび管理することができます。	2019 年 3 月 11 日
Amazon EKS プラットフォームバージョンの更新	CVE-2019-1002100 に対応するために Amazon EKS Kubernetes `1.11` クラスターをパッチレベル `1.11.8` に更新する新しいプラットフォームバージョン。	2019 年 3 月 8 日
クラスターの上限の引き上げ	Amazon EKS ではAWS リージョン内で作成できるクラスターの数が 3 から 50 に増えました。	2019 年 2 月 13 日
Amazon EKS AWS リージョンの拡張	Amazon EKS が欧州 (ロンドン) (`eu-west-2`)、欧州 (パリ) (`eu-west-3`)、アジアパシフィック (ムンバイ) (`ap-south-1`) AWS リージョンで利用可能になりました。	2019 年 2 月 13 日

新しい Amazon EKS 最適化 AMI に ALAS-2019-1156 用のパッチを適用	Amazon EKS は ALAS-2019-1156 で説明されている脆弱性に対応するため、Amazon EKS 最適化 AMI を更新しました。	2019 年 2 月 11 日
新しい Amazon EKS 最適化 AMI に ALAS2-2019-1141 用のパッチを適用	Amazon EKSは ALAS2-2019-1141 で参照されている CVE に対応するため、Amazon EKS 最適化された AMI を更新しました。	2019 年 1 月 9 日
Amazon EKS AWS リージョンの拡張	Amazon EKS がアジアパシフィック (ソウル) (ap-northeast-2) AWS リージョンで利用可能になりました。	2019 年 1 月 9 日
Amazon EKS リージョンの拡張	Amazon EKS が、欧州 (フランクフルト) (eu-central-1)、アジアパシフィック (東京) (ap-northeast-1)、アジアパシフィック (シンガポール) (ap-southeast-1)、およびアジアパシフィック (シドニー) (ap-southeast-2) の AWS リージョンでも新たに利用可能になりました：	2018 年 12 月 19 日
Amazon EKS クラスターの更新	Amazon EKS クラスターの Kubernetes バージョンの更新およびノード置換に関するドキュメントが追加されました。	2018 年 12 月 12 日

Amazon EKS AWS リージョンの拡張	Amazon EKS が欧州 (ストックホルム) (eu-north-1) AWS リージョンで利用可能になりました。	2018 年 12 月 11 日
Amazon EKS プラットフォームバージョンの更新	link:security/security-bulletins/AWS-2018-020/[CVE-2018-1002105,type="marketing"] に対応するために、新しいプラットフォームバージョンをパッチレベル 1.10.11 に更新しました。	2018 年 12 月 4 日
ALB イングレスコントローラー のバージョン 1.0.0 のサポートを追加	ALB イングレスコントローラー で、AWS の公式サポートを備えたバージョン 1.0.0 がリリースされました。	2018 年 11 月 20 日
CNI ネットワーク設定のサポートを追加	Kubernetes バージョン 1.2.1 用の Amazon VPC CNI プラグインが、セカンダリ Pod ネットワークインターフェイスのカスタムネットワーク設定をサポートするようになりました。	2018 年 10 月 10 日
MutatingAdmissionWebhook と ValidatingAdmissionWebhook のサポートを追加	Amazon EKS プラットフォームバージョン 1.10-eks.2 が MutatingAdmissionWebhook および ValidatingAdmissionWebhook アドミッションコントローラーをサポートするようになりました。	2018 年 10 月 10 日

パートナー AMI 情報の追加	Canonical は Amazon EKS と提携し、クラスターで使用できるノード AMI を作成しました。	2018 年 10 月 3 日
AWS CLI の update-kubeconfig コマンドの手順を追加	Amazon EKS で update-kubeconfig が AWS CLI に追加され、クラスターにアクセスするための kubeconfig ファイルの作成プロセスが簡単になりました。	2018 年 9 月 21 日
新しい Amazon EKS 最適化された AMI	Amazon EKS はさまざまなセキュリティ修正と AMI 最適化を提供する Amazon EKS 最適化された AMI (GPU サポートあり/なし) 更新しました。	2018 年 9 月 13 日
Amazon EKS AWS リージョンの拡張	Amazon EKS が欧州 (アイルランド) (eu-west-1) リージョンで利用可能になりました。	2018 年 9 月 5 日
Amazon EKS プラットフォームバージョンの更新	Kubernetes アグリゲーションレイヤと 水平ポッドオートスケーラー (HPA) をサポートする新しいプラットフォームバージョン。	2018 年 8 月 31 日

新しい Amazon EKS 最適化された AMI と GPU サポート	Amazon EKS は新しい AWS CloudFormation ノードテンプレートとブートストラップスクリプトを使用するため、Amazon EKS 最適化 AMI を更新しました。さらに、新しい GPU サポートを備えた Amazon EKS 最適化 AMI が利用可能になりました。	2018 年 8 月 22 日
新しい Amazon EKS 最適化 AMI に ALAS2-2018-1058 用のパッチを適用	Amazon EKS は ALAS2-2018-1058 で参照されている CVE に対応するため、Amazon EKS 最適化 AMI を更新しました。	2018 年 8 月 14 日
Amazon EKS 最適化された AMI のビルドスクリプト	Amazon EKS で、Amazon EKS 最適化された AMI のビルドに使用されているビルドスクリプトをオープンソース化されました。これらのビルドスクリプトは、現在 GitHub で入手できます。	2018 年 7 月 10 日
Amazon EKS 初回リリース	サービス開始時の初版ドキュメント	2018 年 5 月 6 日

EKS ユーザーガイドに貢献する

AWS は「EKS ユーザーガイド」の貢献エクスペリエンスを改善しています。

EKS ユーザーガイドのソースを GitHub 上で直接編集できるようになりました。

更新されたエクスペリエンスではマークダウンと同様の強力なオーサリング言語である AsciiDoc が使用されます。AsciiDoc はシンプルな構文と、高度な書式設定、相互参照、セキュリティコントロールなどのエンタープライズドキュメント機能を組み合わせます。

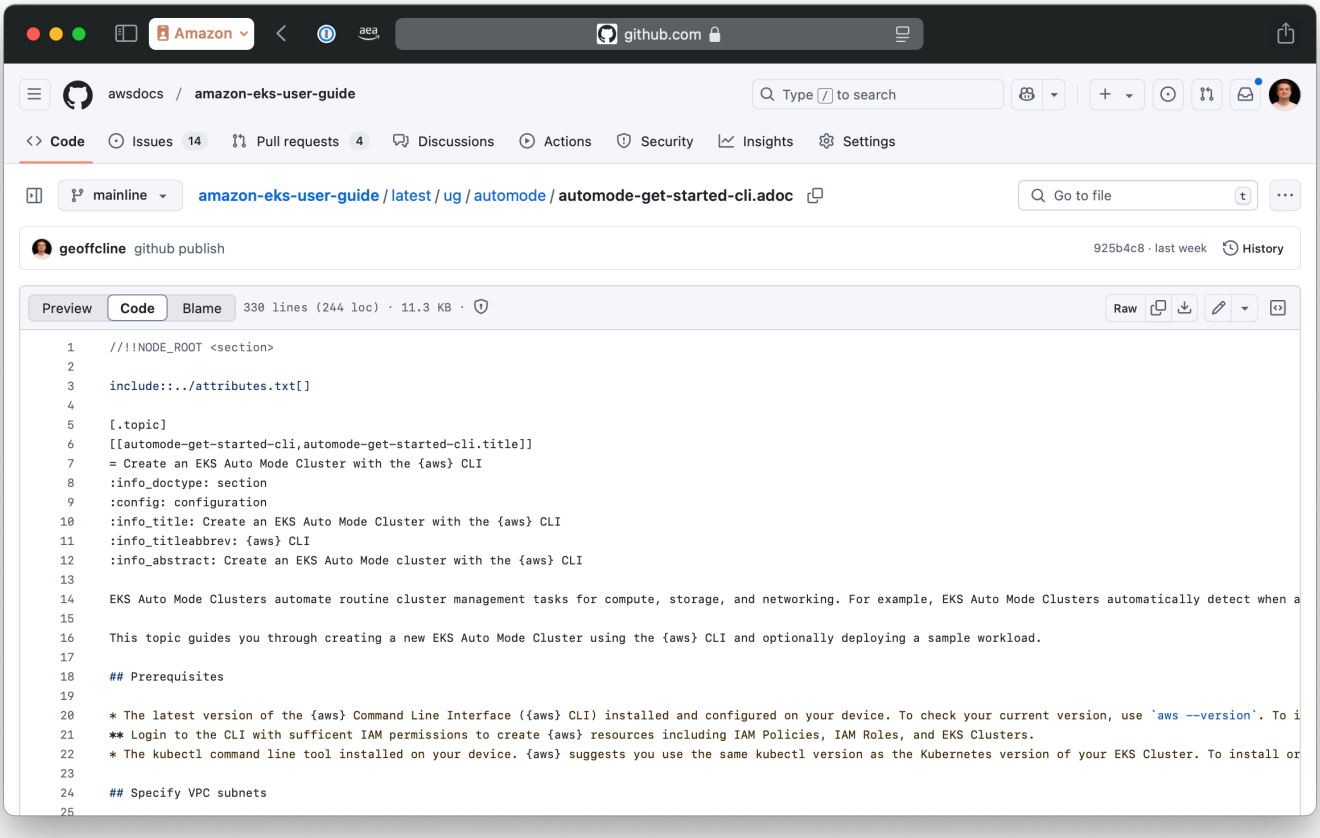
EKS Docs を GitHub 上で直接編集できるようになりました。合理化されたプロセスには次が含まれます:

- プルリクエストの処理の高速化
- 手動ステップの削減
- 自動コンテンツ品質チェック

お客様の貢献をお待ちしています。

ウェブブラウザから単一ページを編集する

EKS ユーザーガイドの 1 ページはウェブブラウザから直接簡単に編集できます。



ウェブブラウザから複数のページを編集する場合は「」を参照してくださいthe section called “GitHub でファイルを編集する”。

前提条件

- 変更するドキュメント ページが Web ブラウザで開かれていること。
- GitHub にサインインしていること。

手順

1. Amazon EKS User Guide で編集したいページに移動します。
2. 右側のペインで、[GitHub でこのページを編集する] を選択します。
3. GitHub で、次のいずれかの方法でエディタを開きます:
   - キーボードの e キーを押します。
   - 鉛筆アイコンをクリックし、ドロップダウンメニューから[インプレース編集] を選択します。

- 編集するオプションがない場合は、GitHub にログインする必要があります。GitHub アカウントでは変更を提案するための特別なアクセス許可は必要ありません。ただし、内部 Amazon 寄稿者は GitHub プロファイルをリンクする必要があります。

4. GitHub エディタでコンテンツに必要な変更を加えます。
   - エディタは構文の強調表示とプレビュー機能を提供します。
   - AsciiDoc マークアップを使用して変更をフォーマットすることができます。
   - `ctrl-f` を使用すると、find/replaceインターフェースを開くことができます。
5. (オプション) 変更内容をプレビューします。
   - `preview` タブを使用して、豊富な書式で変更内容をプレビューします。
   - 変更されたセクションを強調表示するには、`show diff` オプションを使用します。削除されたセクションの左余白には赤色のインジケータがあります。新しいセクションでは左余白に緑色のインジケータが表示されます。
6. 編集が完了したら、エディターの上部にあるCommit changes...ボタンをクリックします。
7. コミットダイアログで、次の操作を行います:
   - メールアドレスが正しいことを確認します。
   - 変更点を説明する簡潔でわかりやすいコミットメッセージを追加します。
   - 必要に応じて、より長い説明を追加します。
   - 新しいブランチとプルリクエストを作成することを選択します。

提案された変更を含むプル リクエストを作成しました。

プルリクエストの概要

PR を作成する場合:

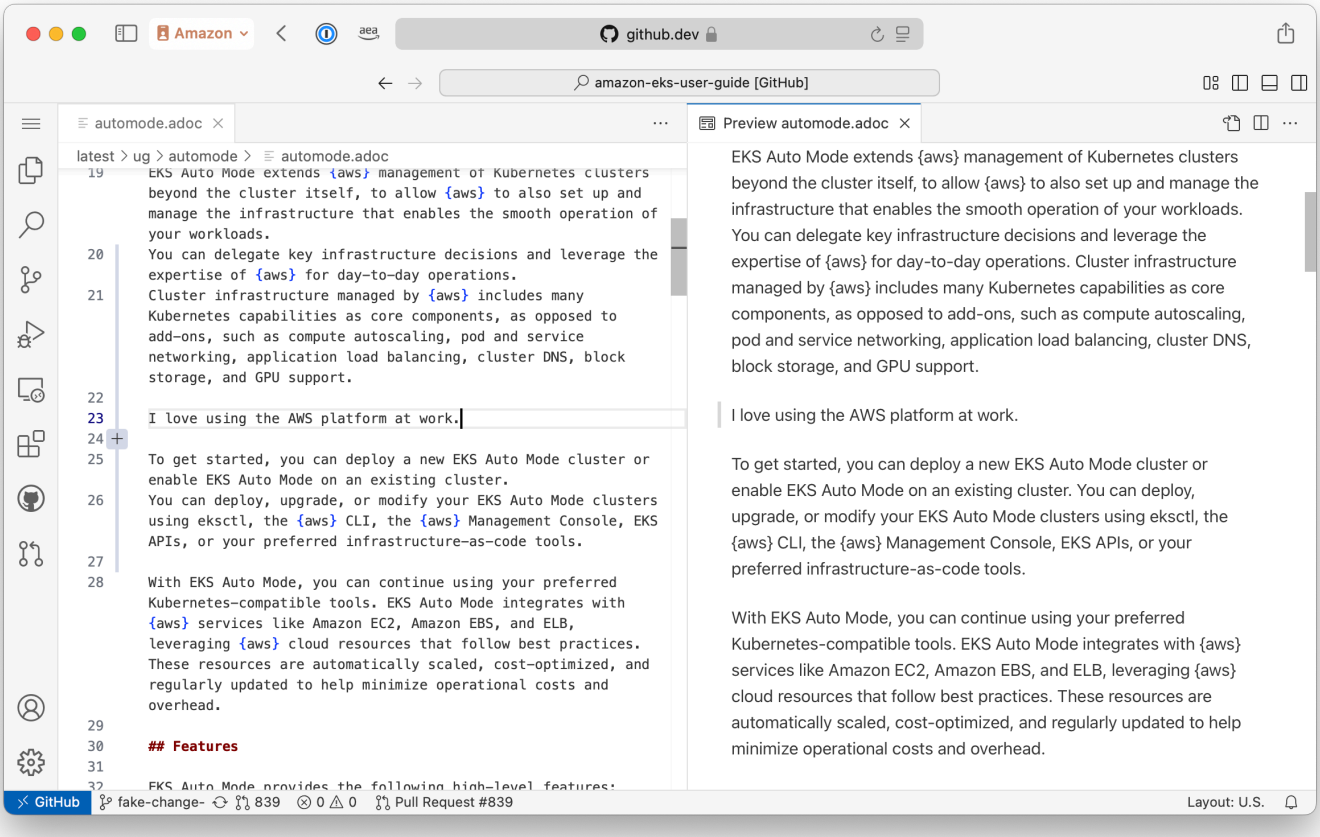
- 変更内容はリポジトリ管理者によるレビューのために送信されます。
- レビュー担当者は変更内容にコメントしたり、修正をリクエストしたりできます。
- 変更を検証するために自動テストが実行される場合があります。
- 承認されると、変更はメインリポジトリにマージされます。

プルリクエストは品質を確保し、変更が統合される前に変更について話し合う方法を提供します。

GitHub Docs でプル リクエストがどのようにレビューされ、承認されるかを学びます。

GitHub Web Editorを使用してWebブラウザから複数のファイルを編集する

複数のページへの変更を提案したり、新しいドキュメントページを作成したりする場合は、GitHub.dev ウェブエディタを使用します。このウェブエディタは一般的な Visual Studio Code テキストエディタに基づいています。



前提条件

- GitHubにログインしました
- Visual Studio Code エディターに精通していること
- Git の知識

手順

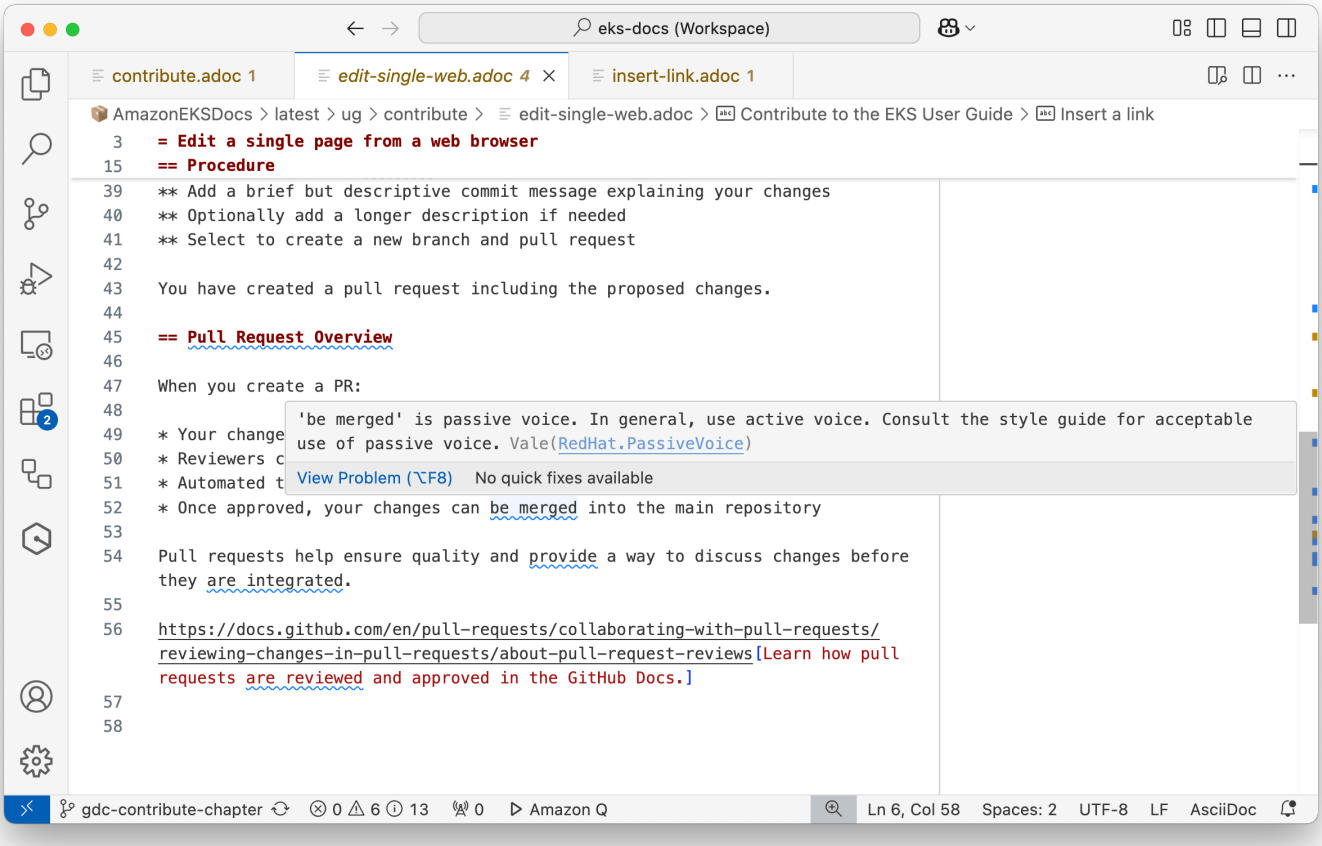
> **Note**
>
> EKS Docs チームはテキストラップや AsciiDoc 構文の強調表示など、エディタの推奨設定を含むワークスペースファイルを作成しました。このワークスペースファイルをロードすることをお勧めします。

1. GitHub.dev でワークスペースを開く
   - URL をブックマークできます `https://github.dev/awsdocs/amazon-eks-user-guide/blob/mainline/eks-docs.code-workspace?workspace=true`
2. (初回セットアップのみ) 自分の GitHub アカウントでリポジトリのフォークを作成するように求められる場合があります。このプロンプトを受け入れます。詳細については「GitHub Docs」の「Fork a repo」(レポジトリのフォーク) 参照してください。
3. (初回セットアップ時のみ) 右下のプロンプトを受け入れて、AsciiDoc 拡張機能をインストールします。
4. `latest/ug` のドキュメントコンテンツに移動します。
   - ドキュメントファイルは最上位のセクション別に整理されています。たとえば、「セキュリティ」章のページには「セキュリティ/」ディレクトリの下にソースファイルがあります。
5. ドキュメントページのプレビューを表示するには右上の「サイドへのプレビューを開く」ボタンを使用します。アイコンには小さな拡大鏡が含まれています。
6. 左側のSource Controlタブを使用して、変更をコミットします。詳細についてはVisual Studio Code のドキュメントを参照してください：
   - Commit changes
   - プルリクエストの作成

プル リクエストを作成すると、ドキュメント チームによってレビューされます。

Vale をローカルにインストールして、入力時にスタイルフィードバックを表示する

入力するとスタイルフィードバックが表示されます。これにより、ぎこちない書き方やタイプミスを特定するのに役立ちます。



概要:

- Vale CLI はスタイル ガイドを読み込み、ソース ファイルに対して実行します。
- EKS Docs リポジトリにはスタイルガイドとローカルルールをロードする Vale 設定ファイルが含まれています。
- Visual Studio (VS) Code の Vale 拡張機能はエディタ内に Vale フィードバックを表示します。

Vale をインストールする

Vale CLI ドキュメントの指示に従って、パッケージマネージャーで Vale をインストールします。

VS Code Vale 拡張機能をインストールする

1. VS Code を開きます。
2. アクティビティバーの拡張機能アイコンをクリックします (または Ctrl+Shift+X キーを押します)。

3. [Vale] を検索します。
4. Chris Chinchilla による「Vale VSCode」拡張機能のインストールをクリックします。
5. プロンプトが表示されたら VS Code を再ロードします。

Vale を同期する

Vale はプロジェクトルートの `.vale.ini` 設定ファイルを使用して、適用するスタイルルールを決定します。

1. VS Code を開きます。
2. [表示] > [ターミナル]をクリックします (またはCtrl+ を押します)。
3. 必要に応じてプロジェクトのルートディレクトリに移動します。
4. コマンドを実行します。

```
vale sync
```

5. Vale がスタイルルールのダウンロードと同期を完了するのを待ちます。

VS Code でスタイルフィードバックを表示する

1. VS Code で Markdown または AsciiDoc ファイルを開きます。
2. Vale 拡張機能はスタイルルールと照合してテキストを自動的にチェックします。
3. スタイルの問題は工ディターに下線付きで表示されます。
4. 下線付きのテキストにマウスを合わせると、特定のスタイルの提案が表示されます。
5. 提案に従うか、スタイルガイドを参照して問題を修正します。

新しいページを作成する

新しいドキュメントページを作成する方法について説明します。このトピックでは、最初のページのメタデータを作成してガイドの目次にページを追加する手順について説明します。

ページを作成

1. 章ディレクトリに移動します。例えば、「セキュリティ」セクションに新しいページを作成する場合は `latest/ug/security` ディレクトリに移動します。

2. ページ ID を決定します。慣例により、ページ ID はすべて小文字で、- で区切られます。このページの ID は create-page です。
3. ページ ID と adoc 拡張子を持つ新しいファイルを作成します。例えば、create-page.adoc。
4. このテンプレートを使用してページメタデータを挿入します。

```
include::../attributes.txt[]

[.topic]
[#unique-page-id]
= Page title
:info_titleabbrev: Short title

[abstract]
--
Brief summary of the page's content.
--

Introduction paragraph goes here.
```

ナビゲーションにページを追加

1. 親ページに移動します。最上位セクションの親ページは ですbook.adoc。
2. 親ページの下部に、子ページを含めます。

   ```
   include::${filename}[leveloffset=+1]
   ```

   例：

   ```
   include::create-page.adoc[leveloffset=+1]
   ```

リンクを挿入する

AsciiDoc は複数の種類のリンクをサポートしています。さまざまな環境でリンクが適切に機能するように、適切なリンク タイプを使用することが重要です。

EKS User Guideのページまたはセクションへのリンク

「EKS ユーザーガイド」など、同じドキュメントサイト内のページまたはセクション間をリンクするには、クロスリファレンス (xref) を使用します。対象セクションが移動または名前変更されると、自動的に更新されます。

ページタイトルをリンクテキストとして使用する

このユーザーガイド内で別の ID にリンクする場合、ほとんどのケースでは、必要に応じてリンクテキストが最新のタイトルに自動更新されるように、以下の手法を使用します。

```
For more information, see <<page-or-section-id>>.
```

カスタムリンクテキストを定義する

カスタムリンクテキストが必要な場合、次の形式を使用します。

```
Here's an example of a <<page-or-section-id,link with custom text>>.
```

AWS ドキュメント内の別のガイドへのリンク

1. AWS ドキュメントページへのリンクを見つけます。
2. `https://docs.aws.amazon.com/` プレフィックスを削除し、パスのみを残します。パスは文字で始まる必要があります。
3. 以下のようにリンクを作成します。

```
link:AmazonS3/latest/userguide/create-bucket-overview.html[Create a bucket,
 type="documentation"]
```

外部ウェブページへのリンク

この形式では、Amazon がホストしていないページへの標準リンクが作成されます。例えば、これを GitHub リンクに使用します。

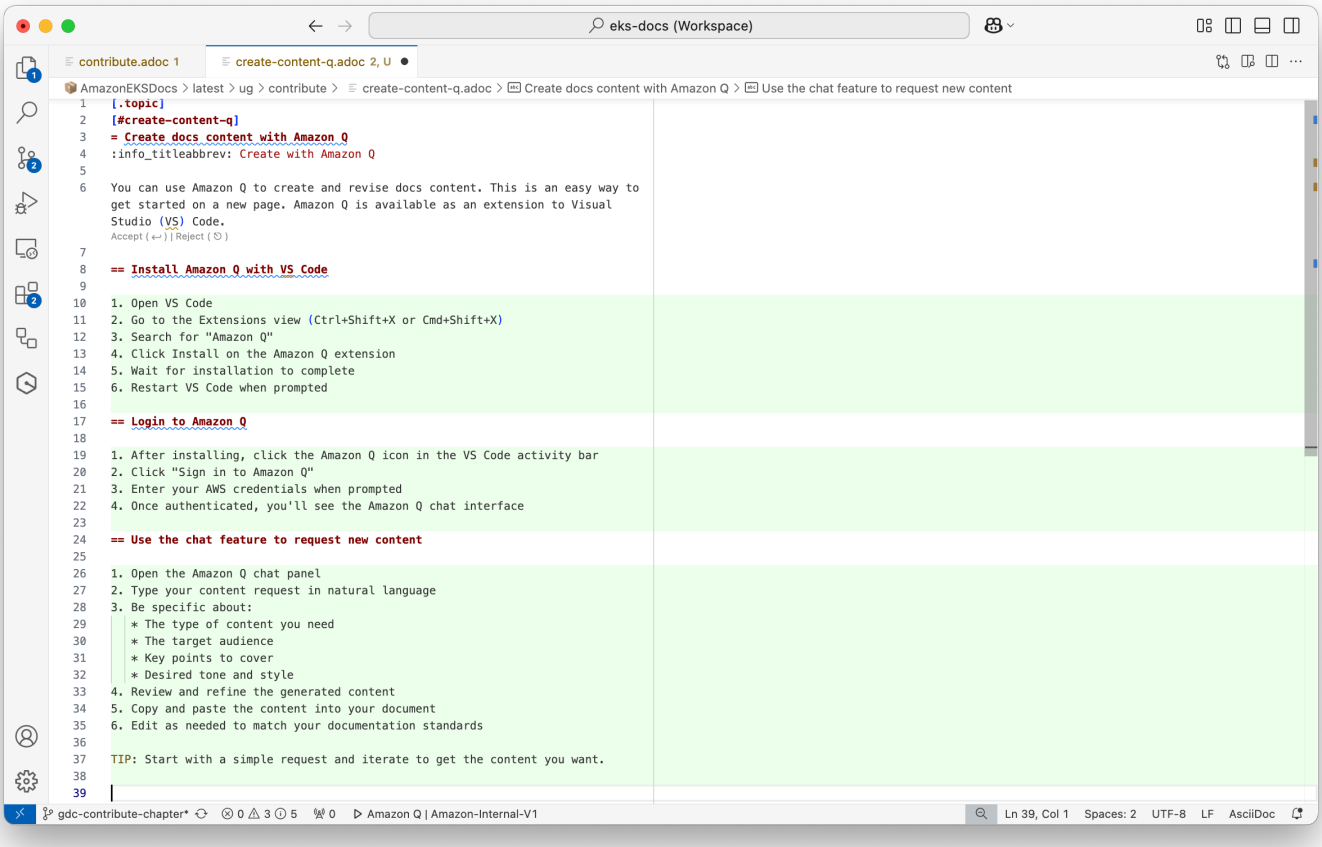
```
https://example.com[Link text]
```

> **Note**
>
> 外部ドメインの許可リストがあります。許可リストは `vale/styles/EksDocs/ExternalDomains.yml`にあります

Amazon Q でドキュメントコンテンツを作成する

Amazon Q を使用して、ドキュメントコンテンツを作成および修正できます。これは新しいページを簡単に開始する方法です。Amazon Q は Visual Studio Code および JetBrains の拡張機能として利用できます。

次の画像では Amazon Q が緑色でマークされた行を生成しました。



VS Code で Amazon Q をインストールする

1. VS Code を開きます。

2. 拡張機能ビューに移動します (Ctrl+Shift+X または Cmd+Shift+X)。
3. 「Amazon Q」を検索します。
4. Amazon Q 拡張機能で [インストール] を選択します。
5. インストールが完了するまで待ちます。
6. プロンプトが表示されたら VS Code を再起動します。

Amazon Q にログインする

1. インストール後、VS Code アクティビティバーの Amazon Q アイコンを選択します。
2. [Amazon Q にサインイン] を選択します。
3. プロンプトが表示されたら、AWS 認証情報を入力します。
4. 認証が完了すると、Amazon Q チャットインターフェイスが表示されます。

Amazon Q を使用してコンテンツを作成する

1. VS Code で編集したいファイルを開きます。
2. 修正するテキストまたは新しいコンテンツの場所を選択します。
3. [Ctrl+I] または [Cmd+I] を押します。
4. プロンプトでは以下について具体的に説明します。
   - 必要なコンテンツの種類。
   - 対象者。
   - カバーする重要なポイント。
   - 望ましいトーンとスタイル。
5. 生成されたコンテンツをインラインプレビューで確認します。
6. 変更を承認するには Enter キーを押し、変更を拒否するには Esc キーを押します。
7. 必要に応じてさらに編集します。

ヒント

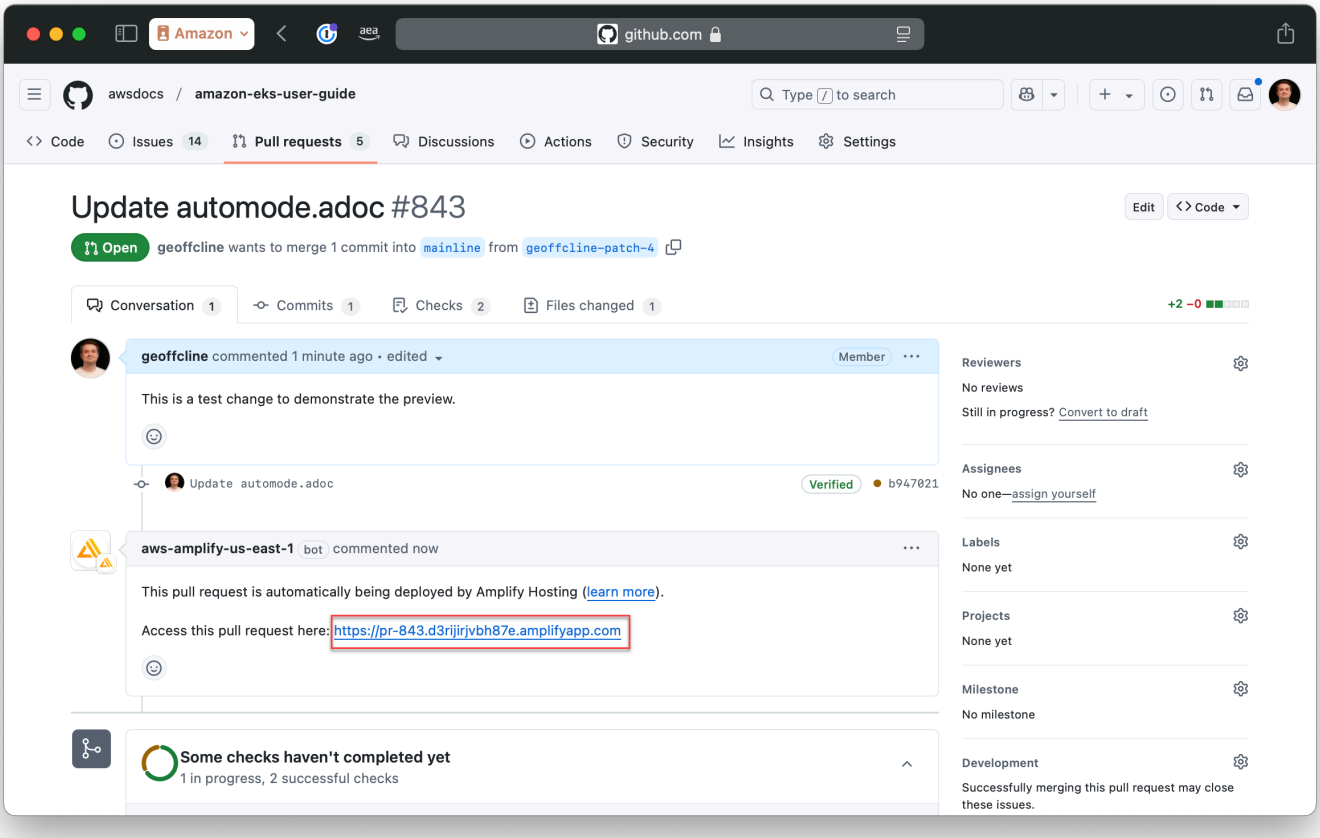
- 簡単なリクエストから始めて、繰り返し実行して必要なコンテンツを取得します。
- ページの見出しの最初のドラフトを作成し、質問して埋めていきます。

- Amazon Q は Markdown を出力する場合があります。これは問題ありません。AsciiDoc ツールではほとんどのマークダウン構文を理解できます。

Amazon Q Developer の詳細については、「IDE での Amazon Q Developer の使用」を参照してください。

プルリクエストのコンテンツのプレビューを表示する

Amazon EKS ユーザーガイド GitHub は、ドキュメントサイトのプレビューを構築して生成するように設定されています。このプレビューには完全な AWS テーマは含まれていませんが、コンテンツが適切にビルドされ、リンクが機能することがチェックされます。



このプレビューは、AWS Amplify によって一時的な URL でホストされます。

プレビューを見る

プルリクエストを送信すると、AWS Amplify はコンテンツのプレビューを構築してデプロイしようとします。

ビルドが正常に完了すると、aws-amplify-us-east-1 はプレビューへのリンクが張られたコメントをプルリクエストに追加します。「Access this pull request here」 (スクリーンショットの赤の枠線で囲まれた部分) の右側にあるリンクを選択します。

ビルドが失敗した場合、リポジトリ管理者はログを確認してフィードバックを提供できます。

> **Note**
>
> 以前に寄稿したことがない場合はプロジェクト保守者がビルドの実行を承認する必要がある場合があります。

プレビューの制限事項

プレビューは、単一の大きな HTML ファイルとして構築されます。公開すると複数のページとして表示されます。

何が効果的か:

- 相互参照 (`xref`)
- インターネットへのリンク
- イメージ
- `samples/` からホストされているコンテンツ

機能しないもの:

- `type="documentation"` を使用した、他の AWS コンテンツへのリンク。というのも、プレビュー環境にはこのコンテンツが存在しないためです。
- 属性 `{aws}` は正しく表示されません。その値は環境によって変化します。

AsciiDoc 構文のリファレンス

AsciiDoc は AWS ドキュメントに推奨されるマークアップ言語です。ツールには Markdown 構文 (見出しやリストなど) が部分的にサポートされていますが、一貫性および適切なレンダリングを確保するため、すべてのコンテンツに AsciiDoc 構文を使用することをお勧めします。

このガイドでは、Amazon EKS ドキュメントに貢献する際に必要な一般的な構文要素について説明します。より高度な構文および詳細情報については、AsciiDoc のドキュメントを参照してください。

新しいページ

すべての新しい AsciiDoc ドキュメントは、the section called “ページを作成” で定義された構造で始まる必要があります。

ヘッダー

見出しを使用してコンテンツを階層的に整理してください。

```
= Page/topic title (level 1)
== Section title (level 2)
=== Level 3 heading
==== Level 4 heading
===== Level 5 heading
====== Level 6 heading
```

注: AWS ドキュメントの見出しには、必ずセンテンスケース (文頭のみ大文字) を使用してください。

テキストのフォーマット

テキストをフォーマットして重要な情報を強調します。

```
Use *bold text* for UI labels.
Use _italic text_ for introducing terms or light emphasis.
Use `monospace text` for code, file names, and commands.
```

Lists

順序付けられていないリスト

特定の順序がない項目には箇条書きリストを作成します。

```
* First item
* Second item
** Nested item
** Another nested item
* Third item
```

順序付けられたリスト

順序付けられたステップや優先順位の高い項目には、番号付きリストを作成します。

```
. First step
. Second step
.. Substep 1
.. Substep 2
. Third step
```

リンク

リンクを適切にフォーマットする方法の詳細については、「the section called “リンクを挿入”」を参照してください。Markdown 形式のリンクはサポートされていません。

コードの例

インラインコード

インラインコードにバックティックを使用します。

```
Use the `kubectl get pods` command to list all pods.
```

コードブロック

構文強調表示機能と属性サポート機能 (エンティティに類似) を持つコードブロックを作成します。

```
 [source,python,subs="verbatim,attributes"]
```

```
----
def hello_world():
    print("Hello, World!")
----
```

イメージ

アクセシビリティのために代替テキストがある画像を挿入します。

```
image::images/image-file.png[Alt text description]
```

テーブル

テーブルを作成して情報を整理します。

```
[%header,cols="2"]
|===
|Header 1
|Header 2

|Cell 1,1
|Cell 1,2

|Cell 2,1
|Cell 2,2
|===
```

より複雑なテーブルについては、AsciiDoc のテーブルに関するドキュメントを参照してください。

コールアウト

コールアウトを使用して重要な情報および勧告を強調表示します。

```
NOTE: This is a note callout for general information.

TIP: This is a tip callout for helpful advice.

IMPORTANT: This is an important callout for critical information.
```

プレビュー:

> **Note**
>
> これはメモのコールアウトです。

他のファイルを含める

他のファイルのコンテンツを含めます。

```
include::filename.adoc[]
```

属性 (エンティティに類似)

事前定義された属性を使用して整合性を維持します。特に、AWS と `arn:aws:` には属性を使用する必要があります。

```
{aws} provides Amazon EKS as a managed Kubernetes service.
```

```
[source,bash,subs="verbatim,attributes"]
----
aws iam attach-role-policy \
    --role-name AmazonEKSAutoClusterRole \
    --policy-arn {arn-aws}iam::aws:policy/AmazonEKSClusterPolicy
----
```

属性のリストについては、「`../attributes.txt`」ファイルを参照してください。

手順

ステップバイステップの手順には以下の書式を使用します。

```
To create an Amazon EKS cluster, do the following steps.

. Sign in to the {aws} Management Console.
. Open the Amazon EKS console.
. Choose *Create cluster*.

...
```