

ユーザーガイド

アマゾン EKS



アマゾン EKS: ユーザーガイド

Copyright © 2025 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

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

Table of Contents

.....	xxii
.....	xxiii
Amazon EKS とは	1
Amazon EKS: Kubernetes 管理の簡素化	1
Amazon EKS の機能	2
関連サービス	4
Amazon EKS の料金	4
一般的なユースケース	5
アーキテクチャ	6
コントロールプレーン	7
コンピューティング	7
Kubernetes の概念	9
Kubernetes が選ばれる理由	10
クラスター	14
ワークロード	19
次のステップ	25
デプロイオプション	25
Amazon EKS デプロイオプションを理解する	25
クラウド内の Amazon EKS	26
データセンターまたはエッジ環境の Amazon EKS	27
エアギャップ環境用の Amazon EKS Anywhere	28
Amazon EKS ツール	28
セットアップする	30
次のステップ	30
AWS CLI の設定	30
アクセスキーを作成するには	31
AWS CLI を構成するには	31
セキュリティトークンを取得するには	32
ユーザー ID を確認するには	32
次のステップ	33
kubectl と eksctl のセットアップ	33
kubectl をインストールまたは更新する	33
ステップ 1: kubectl がインストールされているかどうかを確認する	33
ステップ 2: kubectl をインストールまたは更新する	34

eksctl をインストールする	48
次のステップ	48
クイックスタート	49
このチュートリアルでは、次の作業を行いました。	49
前提条件	50
クラスターを設定する	50
IngressClass を作成する	51
2048 ゲームサンプルアプリケーションをデプロイする	51
Amazon EKS 自動モードを使用してデータを永続化する	55
クラスターとノードを削除する	57
Amazon EKS の詳細	58
概要	58
Amazon EKS ワークショップ	58
Amazon EKS の実践的なクラスターセットアップチュートリアル	60
Amazon EKS のサンプル	61
AWS のチュートリアル	61
開発者ワークショップ	62
Terraform ワークショップ	62
AWS Amazon EKS トレーニング	62
はじめに	64
クラスターを作成する (EKS Auto Mode)	64
クラスターを作成する (eksctl)	65
前提条件	66
ステップ 1: Amazon EKS クラスターとノードを作成する	66
ステップ 2: Kubernetes リソースを表示する	67
ステップ 3: クラスターとノードを削除する	69
次のステップ	70
クラスターを作成する (コンソールと CLI)	70
前提条件	71
ステップ 1: Amazon EKS クラスターを作成する	72
ステップ 2: 自分のコンピュータでクラスターとの通信を設定する	74
ステップ 3: ノードを作成する	75
ステップ 4: リソースを表示する	77
ステップ 5: リソースを削除する	78
次のステップ	79
EKS Auto Mode	80

機能	80
自動コンポーネント	81
設定	83
クラスターを作成する	83
eksctl CLI	84
AWS CLI	86
マネジメントコンソール	94
既存のクラスターを有効にする	96
移行のリファレンス	97
EBS ボリュームの移行	98
ロードバランサーの移行	99
クラスターで有効にする	99
Karpenter からの移行	104
MNG から移行する	107
Fargate からの移行	108
ワークロードを実行する	113
拡張ワークロードのデプロイ	114
ロードバランサーをデプロイする	117
ステートフルワークロードをデプロイする	123
構成する	128
ノードクラスの作成	131
ノードプールの作成	138
ingress class の作成	144
サービスの作成	151
StorageClass を作成する	157
EKS 自動モードI を無効にする	166
Kubernetes バージョンを更新する	168
組み込みノードプールを確認する	169
デプロイを制御する	171
重要なアドオンを実行する	172
ネットワークポリシーの使用	174
サブネットにタグを付ける	176
加速ワークロードのデプロイ	178
レポートを生成する	185
ルートボリュームの暗号化	188
仕組み	192

管理されたインスタンス	192
ID とアクセス	196
ネットワーク	202
トラブルシューティング	205
ノードモニタリングエージェント	206
AWS EC2 CLI を使用して EC2 マネージドインスタンスからコンソール出力を取得する ...	207
デバッグコンテナと kubectl CLI を使用してノードログを取得する	207
AWS コンソールで EKS 自動モードに関連付けられたリソースを表示する	209
AWS アカウントの IAM エラーを表示します	209
Auto Mode ノードにスケジュールできない Pod のトラブルシューティング	209
クラスターに参加しないノードのトラブルシューティング	210
Pod 間でボリュームを共有する	212
Auto Mode に含まれているコントローラーのトラブルシューティング	213
リリースノート	213
2025 年 6 月 13 日	213
2025 年 4 月 30 日	213
2025 年 4 月 18 日	214
2025 年 4 月 11 日	214
2025 年 4 月 4 日	214
2025 年 3 月 31 日	214
2025 年 3 月 21 日	214
2025 年 3 月 14 日	215
クラスター	216
自動クラスターの作成	217
前提条件	50
クラスターの作成 - AWS コンソール	219
クラスターの作成 - AWS CLI	224
次のステップ	94
クラスターを作成する	230
前提条件	50
ステップ 1: クラスター IAM ロールを作成する	231
ステップ 2: クラスターを作成する	232
ステップ 3: kubeconfig を更新する	242
ステップ 4: クラスターのセットアップ	242
次のステップ	94
クラスターのインサイト	244

クラスターインサイトを使用した Kubernetes バージョンアップグレードの準備	245
クラスターインサイトを使用した設定ミスのトラブルシューティング	246
クラスターインサイトを表示する	246
Kubernetes バージョンを更新する	252
Amazon EKS Auto Mode の考慮事項	253
概要	253
ステップ 1: アップグレードの準備をする	254
ステップ 2: アップグレードに関する考慮事項を確認する	255
ステップ 3: クラスターコントロールプレーンを更新する	256
ステップ 4: クラスターコンポーネントを更新する	259
Amazon EKS クラスターの Kubernetes バージョンをダウングレードする	260
クラスターを削除	260
考慮事項	99
クラスターの削除 (eksctl)	261
クラスターを削除する (AWS コンソール)	262
クラスターを削除する (AWS CLI)	263
クラスターエンドポイントのアクセス	265
IPv6 クラスターエンドポイント形式	265
IPv4 クラスターエンドポイント形式	266
クラスタープライベートエンドポイント	267
クラスターエンドポイントのアクセスの変更	267
プライベート専用 API サーバーへのアクセス	270
エンドポイントアクセスの設定	271
Windows サポートを有効にする	275
考慮事項	99
前提条件	50
Windows サポートを有効にする	277
Windows ポッドをデプロイする	279
Windows ノードでより高い Pod 密度をサポートする	279
Windows サポートを無効にする	280
プライベートクラスター	280
Kubernetes バージョン	284
標準サポートで入手可能なバージョン	285
延長サポートで利用可能なバージョン	285
Amazon EKS Kubernetes リリースカレンダー	286
AWS CLI を使用してバージョン情報を取得する	287

Amazon EKS のバージョンのよくある質問	288
Amazon EKS 延長サポートに関するよくある質問	290
標準サポートバージョン	293
延長サポートバージョン	299
サポート期間を表示する	303
アップグレードポリシーを表示する	303
延長サポートを有効にする	306
延長サポートを無効にする	307
プラットフォームのバージョン	308
Kubernetes バージョン 1.33	310
Kubernetes バージョン 1.32	310
Kubernetes バージョン 1.31	312
Kubernetes バージョン 1.30	316
Kubernetes バージョン 1.29	320
Kubernetes バージョン 1.28	326
Kubernetes バージョン 1.27	334
Kubernetes バージョン 1.26	342
最新のプラットフォームバージョンを取得する	350
プラットフォームバージョンを変更する	350
自動スケーリング	350
EKS Auto Mode	351
その他のソリューション	351
ゾーンシフトの詳細	352
ポッド間の東西ネットワークトラフィックフローを理解する	352
EKS ゾーンシフトの要件	357
よくある質問	365
その他のリソース	367
ゾーンシフトを有効にする	368
Amazon Application Recovery Controller とは	369
ゾーンシフトとは	369
ゾーンオートシフトとは	369
オートシフト中の EKS の動作	369
EKS クラスターを Amazon Application Recovery Controller (ARC) に登録する (AWS コン ソール)	370
次のステップ	94
アクセスを管理する	371

一般的なタスク	371
背景	176
EKS Auto Mode に関する考慮事項	261
Kubernetes API アクセス	373
IAM アイデンティティと Kubernetes のアクセス許可を関連付ける	374
クラスター認証モードを設定する	375
許可を付与する	376
aws-auth ConfigMap	417
OIDC プロバイダーをリンクする	427
OIDC プロバイダーのリンクを解除する	432
クラスターリソースにアクセスする	432
必要なアクセス許可	433
kubectl を使用してクラスターにアクセスする	439
kubeconfig ファイルを自動で作成する	440
AWS へのワークロードアクセス	441
サービスアカウントトークン	442
クラスターアドオン	443
Amazon Elastic Kubernetes Service クラスター上のワークロードへの AWS Identity and Access Management 許可の付与	444
IRSA による認証情報	448
Pod Identity	472
コンピューティングを管理する	506
コンピューティングオプションの比較	506
マネージド型ノードグループ	512
マネージド型ノードグループの概念	513
マネージド型ノードグループのキャパシティータイプ	516
作成	519
更新	530
動作の詳細を更新する	535
起動テンプレート	540
削除	557
セルフマネージド型ノード	559
Amazon Linux	560
Bottlerocket	570
Windows	574
Ubuntu Linux	583

更新方式	586
AWS Fargate	599
AWS Fargate 考慮事項	600
Fargate 比較テーブル	602
はじめに	605
プロファイルを定義する	610
プロファイルを削除する	616
ポッド設定の詳細	617
OS パッチ適用イベント	620
メトリクスを収集する	623
ログ記録	625
Amazon EC2 インスタンスタイプ	637
各 Amazon EC2 インスタンスタイプの Amazon EKS の推奨最大ポッド数	639
EKS Auto Mode に関する考慮事項	261
事前構築済みの最適化された AMI	641
AL2 AMI の廃止	641
Amazon Linux	647
Bottlerocket	656
Ubuntu Linux	662
Windows	663
ノードのヘルス	753
ノードモニタリングエージェント	754
ノードの自動修復	754
ノードのヘルスの問題	755
ノードのヘルスを表示する	767
ノードログの取得	769
ハイブリッドノード	772
機能	773
一般的な考慮事項	774
前提条件	774
ハイブリッドノードを実行する	840
構成する	862
仕組み	901
ハイブリッドノード用 nodeadm	945
トラブルシューティング	961
アプリデータストレージ	983

Amazon EBS	983
考慮事項	984
前提条件	984
ステップ 1: IAM ロールを作成する	985
ステップ 2: Amazon EBS CSI ドライバーを入手する	993
ステップ 3: サンプルアプリケーションをデプロイする	993
Amazon EFS	993
考慮事項	993
前提条件	994
ステップ 1: IAM ロールを作成する	995
ステップ 2: Amazon EFS CSI ドライバーを取得する	1001
ステップ 3: Amazon EFS ファイルシステムを作成する	1001
ステップ 4: サンプルアプリケーションをデプロイする	1002
Amazon FSx for Lustre	1002
ドライバをデプロイする	1002
最適化 (EFA)	1010
最適化 (EFA 以外)	1027
Amazon FSx for NetApp ONTAP	1034
Amazon FSx for OpenZFS	1035
Amazon File Cache	1035
Mountpoint for Amazon S3	1036
考慮事項	1036
ドライバをデプロイする	1037
ドライバを削除する	1047
CSI スナップショットコントローラー	1048
ネットワークを設定する	1050
管理コンソールから既存の VPC サブネットを Amazon EKS クラスターに追加する	1050
VPC およびサブネットの要件	1050
VPC の要件と考慮事項	1051
サブネットの要件と考慮事項	1053
共有サブネットの要件と考慮事項	1059
「VPC を作成する」	1060
前提条件	50
パブリックサブネットおよびプライベートサブネット	1061
パブリックサブネットのみ	1064
プライベートサブネットのみ	1065

セキュリティグループの要件	1067
デフォルトのクラスターセキュリティグループ	1067
クラスタートラフィックの制限	1069
共有セキュリティグループ	1070
ネットワーキングアドオンの管理	1071
組み込みアドオン	1071
オプションの AWS ネットワークアドオン	1072
Amazon VPC CNI	1072
代替 CNI プラグイン	1179
AWS ロードバランサーコントローラー	1181
CoreDNS	1199
kube-proxy	1218
ワークロード	1224
サンプルデプロイ (Linux)	1224
前提条件	50
名前空間を作成します。	1225
Kubernetes デプロイメントを作成する	1225
サービスを作成する	1227
作成されたリソースを確認する	1228
ポッドでシェ尔を実行する	1231
次のステップ	1232
サンプルデプロイ (Windows)	1232
前提条件	50
名前空間を作成します。	1225
Kubernetes デプロイメントを作成する	1225
サービスを作成する	1227
作成されたリソースを確認する	1235
ポッドでシェ尔を実行する	1231
次のステップ	1239
Vertical Pod Autoscaler	1239
Vertical Pod Autoscaler をデプロイする	1240
Vertical Pod Autoscaler のインストールをテストします	1241
Horizontal Pod Autoscaler	1245
Horizontal Pod Autoscaler テストアプリケーションを実行する	1246
ネットワーク負荷分散	1249
前提条件	50

考慮事項	99
ネットワークロードバランサーを作成する	1253
(オプション) サンプルアプリケーションをデプロイする	1255
アプリケーション負荷分散	1259
前提条件	50
インGRESグループで ALB を再利用する	1262
(オプション) サンプルアプリケーションをデプロイする	1264
サービスの外部 IP を制限する	1267
リポジトリにイメージをコピーする	1269
Amazon イメージレジストリを表示する	1272
Amazon EKS アドオン	1276
考慮事項	1277
Amazon EKS 自動モードI の考慮事項	1278
サポート	1279
AWSアドオン	1281
コミュニティアドオン	1297
マーケットプレイス アドオン	1302
アドオンを作成する	1318
アドオンを更新する	1331
互換性を確認する	1338
アドオンを削除する	1340
IAM ロール	1344
カスタマイズ可能なフィールド	1350
コンテナイメージを検証する	1354
クラスターの管理	1355
コストモニタリング	1355
Pod ごとにコストを表示する	1356
Kubecost のインストール	1357
Kubecost ダッシュボードにアクセスする	1358
Kubecost の詳細	1359
メトリクスサーバー	1368
考慮事項	99
Amazon EKS アドオンを使用してコミュニティアドオンとしてデプロイする	1369
マニフェストを使用してデプロイする	1370
Helm を使用してアプリケーションをデプロイする	1371
リソースのタグ付け	1373

タグの基本	1373
リソースのタグ付け	1374
タグの制限	1375
請求用のリソースにタグを付ける	1375
コンソールでのタグの処理	1376
CLI、API または eksctl でのタグの操作	1377
Service Quotas	1379
AWS Management Console で EKS Service Quotas を表示する	1379
AWS CLI で EKS サービスクォータを表示する	1380
Amazon EKS Service Quotas	1381
AWS Fargate サービスクォータ	1381
セキュリティ	1383
ベストプラクティス	1384
脆弱性の分析	1384
Amazon EKS 用 Center for Internet Security (CIS) ベンチマーク	1385
Amazon EKS のプラットフォームバージョン	1385
オペレーティングシステムの脆弱性リスト	1386
Amazon Inspector によるノード検出	1386
Amazon GuardDuty によるクラスターとノードの検出	1386
コンプライアンスの検証	1386
EKSに関する考慮事項	1387
インフラストラクチャセキュリティ	1388
レジリエンス	1392
Kubernetes に関する考慮事項	1393
証明書への署名	1393
デフォルトの ロールとユーザー	1396
シークレット暗号化を有効にする	1401
AWS 秘密マネジャー	1406
すべての Kubernetes API データを対象とするデフォルトのエンベロープ暗号化	1407
EKS Auto に関する考慮事項	1415
API セキュリティと認証	1415
ネットワークセキュリティ	1415
EC2 マネージドインスタンスのセキュリティ	1416
自動リソース管理	1417
セキュリティに関するベストプラクティス	1418
IAM リファレンス	1418

対象者	1418
アイデンティティを使用した認証	1419
ポリシーを使用したアクセス権の管理	1422
IAM および Amazon EKS	1425
アイデンティティベースのポリシー	1430
サービスにリンクされた役割	1437
ポッド実行IAMロール	1454
コネクタ IAM ロール	1459
AWS 管理ポリシー	1463
トラブルシューティング	1488
クラスター IAM ロール	1490
ノード の IAM ロール	1494
Amazon EKS 自動モードI クラスター IAM ロール	1500
Auto Mode ノードの IAM ロール	1503
クラスターをモニタリングする	1507
Amazon EKS でのモニタリングとログ記録	1507
Amazon EKS のモニタリングとログ記録のツール	1509
オブザーバビリティダッシュボード	1513
概要	1513
クラスターヘルス	1513
コントロールプレーンのモニタリング	1514
クラスターのインサイト	1516
ノードのヘルスの問題	1516
Prometheus メトリクス	1517
ステップ 1: Prometheus メトリクスを有効にする	1518
ステップ 2: Prometheus メトリクスを使用する	1520
ステップ 3: Prometheus スクレイパーを管理する	1520
Helm を使用してデプロイする	1520
コントロールプレーン	1523
Amazon CloudWatch	1531
Amazon CloudWatch の基本メトリクス	1531
Amazon CloudWatch オブザーバビリティオペレータ	1540
コントロールプレーンのログ	1541
コントロールプレーンログを有効または無効にする	1542
クラスターのコントロールプレーンログを表示する	1545
AWS CloudTrail	1546

リファレンス	1547
ログファイルエントリ	1547
Auto Scaling グループメトリクス	1550
ADOT Operator	1551
Amazon EKS ダッシュボード	1551
Amazon EKS ダッシュボードとは	1551
ダッシュボードは AWS Organizations をどのように使用しますか?	1552
AWS コンソールを使用して EKS ダッシュボードを有効にする	1553
EKS ダッシュボードを表示する	1554
ダッシュボードを設定する	1554
AWS コンソールを使用して EKS ダッシュボードを無効にする	1555
EKS ダッシュボードのトラブルシューティング	1556
組織を設定する	1557
他の サービスでの使用	1563
AWS CloudFormation	1563
Amazon EKS と AWS CloudFormation のテンプレート	1563
AWS CloudFormation の詳細はこちら	1564
Amazon Detective	1564
Amazon EKS との連携で Amazon Detective を使用する	1565
Amazon GuardDuty	1565
AWS Resilience Hub	1567
Amazon Security Lake	1567
Security Lake を Amazon EKS で使用する利点	1567
Amazon EKS の Security Lake の有効化	1568
Security Lake の EKS ログの分析	1568
Amazon VPC Lattice	1568
AWS Local Zones	1569
トラブルシューティング	1570
容量不足	1570
ノードをクラスターに結合できません	1570
許可されていないか、アクセスが拒否されました (kubectl)	1572
hostname doesn't match	1573
getsockopt: no route to host	1573
Instances failed to join the Kubernetes cluster	1574
マネージド型ノードグループのエラー	1574
Not authorized for images	1579

ノードが NotReady 状態です	1579
EKS ログコレクター	1580
コンテナランタイムネットワークの準備ができていません	1581
TLS ハンドシェイクタイムアウト	1582
無効なクライアントトークンID	1583
コントロールプレーンをアップグレードする前に、ノードグループが Kubernetes バージョン と一致する必要があります	1583
多数のノードを起動すると、Too Many Requests エラーが発生します。	1584
Kubernetes API サーバリクエストに対する HTTP 401 Unauthorized エラーレスポンス	1584
Amazon EKS プラットフォームのバージョンは現在のプラットフォーム バージョンより 2 つ 以上遅れています	1585
クラスターの正常性に関するよくある質問およびエラーコードと解決パス	1588
Amazon EKS Connector	1595
Amazon EKS Connector の考慮事項	1595
Amazon EKS コネクターの必要な IAM ロール	1596
クラスターに接続する	1596
考慮事項	1597
前提条件	1597
ステップ 1: クラスターの登録	1597
ステップ 2: eks-connector エージェントのインストール	1600
次のステップ	1602
クラスターに対するアクセスを許可する	1602
前提条件	1602
クラスターの登録解除	1604
Kubernetes クラスターの登録を解除する	1605
Kubernetes クラスターのリソースをクリーンアップする	1606
EKS Connector をトラブルシューティングする	1606
基本的なトラブルシューティング	1606
Helm の問題: 403 Forbidden	1608
コンソールのエラー: クラスターが Pending 状態でスタックしています	1609
コンソールのエラー: ユーザーシステム:serviceaccount:eks-connector:eks-connector can't impersonate resource users in API group at cluster scope (eks-connector はクラスタース コープで API グループのリソースユーザーを偽装できません)	1609
コンソールのエラー: [...] is forbidden: User [...] cannot list resource [...] in API group at the cluster scope ([...] は禁止されています: ユーザー [...] が、クラスタースコープで API グ ループのリソース [...] をリストできません)	1610

コンソールのエラー: Amazon EKS can't communicate with your Kubernetes cluster API server. 正常に接続するには、クラスターが ACTIVE 状態である必要があります。数分後にもう一度お試しください。	1610
Amazon EKS Connector Pod がクラッシュループしている	1611
eks-connector の起動の失敗: InvalidActivation	1611
クラスターノードにアウトバウンド接続がありません	1612
Amazon EKS Connector Pod が ImagePullBackOff 状態になっている	1613
よくある質問	1613
セキュリティに関する考慮事項	1614
AWS の責任	1615
お客様の責任	1615
Amazon EKS on AWS Outposts	1616
各デプロイオプションを使用する時	1616
デプロイオプションの比較	1617
ローカルクラスターを実行する	1620
サポート対象の AWS リージョン	1620
ローカルクラスターをデプロイする	1620
EKS プラットフォームのバージョン	1631
VPC とサブネットを作成する	1647
切断の準備をする	1651
容量に関する考慮事項	1656
クラスターのトラブルシューティングを行う	1658
ノード	1670
eksctl	1672
AWS Management Console	1674
EKS での機械学習	1680
AI/ML のために EKS を選択すべき理由	1680
主なユースケース	1681
ケーススタディ	1682
EKS での機械学習の使用を開始する	1683
ML の使用を開始する	1683
ML on EKS の使用を続ける	1684
ML のために準備する	1684
Linux GPU AMI を実行する	1685
Windows GPU AMI を実行する	1687
MNG の GPU を予約する	1694

SMN の GPU を予約する	1697
GPU ノードにテイントを適用する	1700
EFA を使用したクラスターのトレーニング	1702
インフェクエンティア クラスターを準備する	1713
ML on EKS のチュートリアルを試す	1720
EKS で生成 AI プラットフォームを構築する	1720
EKS で専門的な生成 AI フレームワークを実行する	1720
ML on EKS の NVIDIA GPU パフォーマンスを最大化する	1720
EKS で動画エンコーディングワークロードを実行する	1721
推論ワークロードのイメージロードを高速化する	1721
ML ワークロードのモニタリング	1721
ML on EKS に関するお知らせ	1721
Amazon EKS に関連するプロジェクト	1722
EKS にデプロイされたソフトウェアのサポート	1722
管理ツール	1723
eksctl	1723
Kubernetes 用の AWS コントローラー	1723
Flux CD	1724
Kubernetes 用 CDK	1724
ネットワーク	1724
Amazon VPC CNI Plugin for Kubernetes	1724
AWS Load Balancer Controller for Kubernetes	1724
ExternalDNS	1725
機械学習	1725
Kubeflow	1725
Auto Scaling	1725
Cluster Autoscaler	1725
Karpenter	1726
Escalator	1726
モニタリング	1726
Prometheus	1726
継続的インテグレーション/継続的デプロイメント	1726
Jenkins X	1726
新機能とロードマップ	1728
ドキュメント履歴	1729
貢献する	1784

単一ページを編集	1784
前提条件	50
手順	102
プルリクエストの概要	1786
GitHub でファイルを編集する	1787
前提条件	50
手順	102
スタイルのフィードバックを表示する	1788
Vale をインストールする	1789
VS Code Vale 拡張機能をインストールする	1789
Vale を同期する	1790
VS Code でスタイルフィードバックを表示する	1790
ページを作成	1790
ページを作成	1790
ナビゲーションにページを追加	1791
リンクを挿入	1792
EKS User Guideのページまたはセクションへのリンク	1792
AWS ドキュメント内の別のページへのリンク	1792
インターネットにリンクする	1792
Amazon Q で作成する	1793
VS Code で Amazon Q をインストールする	1793
Amazon Q にログインする	1794
Amazon Q を使用してコンテンツを作成する	1794
ヒント	1794
PR プレビューを見る	1795
プレビューを見る	1796
プレビューの制限事項	1796
AsciiDoc 構文	1796
基本的なフォーマット	1797
ヘッダー	1797
Lists	1797
リンク	1798
イメージ	1798
コードブロック	1798
テーブル	1798
警告	1798

含まれるもの 1799

このページの改善にご協力ください

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

Amazon EKS とは

Amazon EKS: Kubernetes 管理の簡素化

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

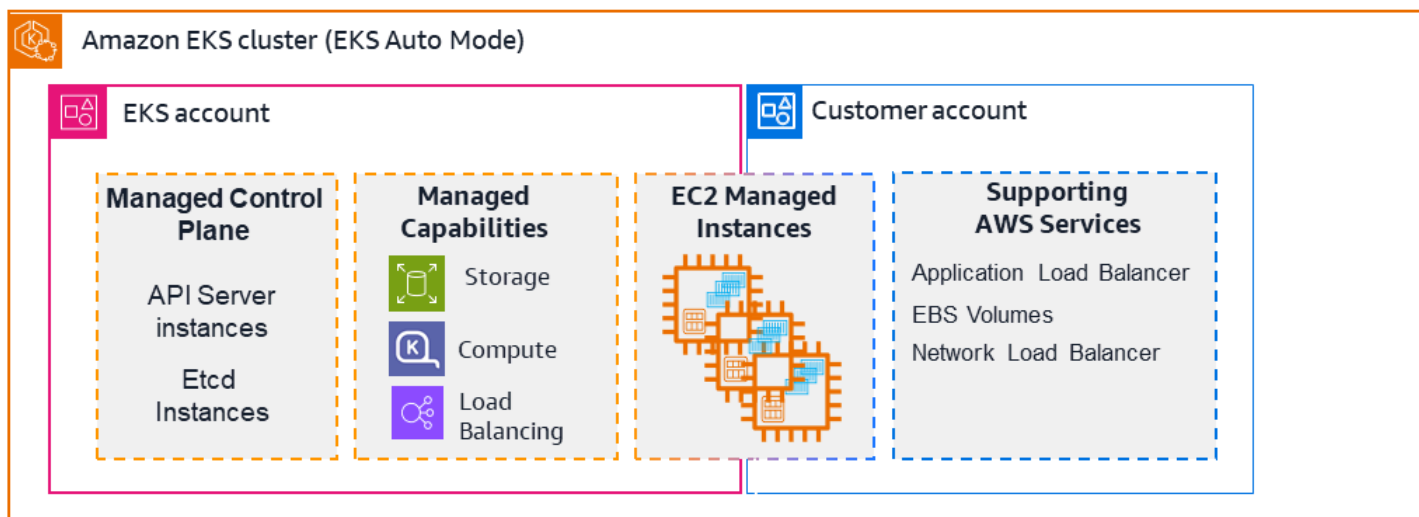
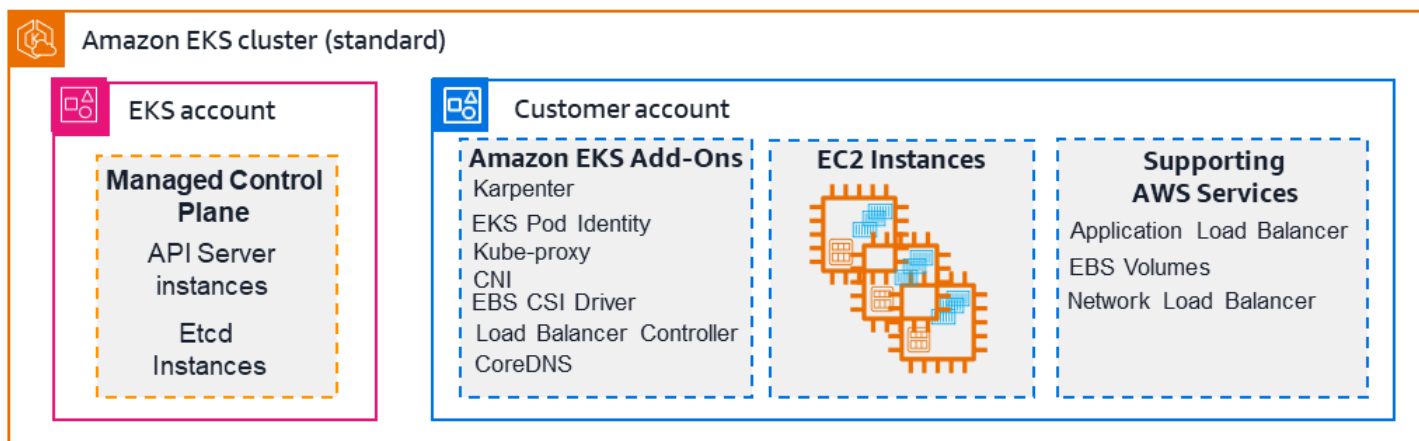
- 運用オーバーヘッドを減らしてアプリケーションをすばやくデプロイできる
- 絶えず変化するワークロードの需要に合わせてシームレスにスケールできる
- AWS 統合と自動更新によりセキュリティが向上する
- 標準 EKS または完全に自動化された EKS 自動モードから選択

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

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

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

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



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

Amazon EKS の機能

Amazon EKS には次の高度な機能があります。

管理インターフェイス

EKS には、AWS Management Console、Amazon EKS API/SDK、CDK、AWS CLI、eksctl CLI、AWS CloudFormation、Terraform など、クラスターをプロビジョニング、管理、保守するためのインターフェイスがいくつか用意されています。詳細については、「[はじめに](#)」および「[クラスター](#)」を参照してください。

アクセス制御ツール

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

コンピューティングリソース

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

[ストレージ]

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

セキュリティ

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

モニタリングツール

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

Kubernetes の互換性とサポート

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

関連サービス

Amazon EKS で利用できるサービス

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

Amazon EC2

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

Amazon EBS

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

Amazon ECR

[Amazon ECR](#) では、コンテナイメージを安全に保存できます。

Amazon CloudWatch

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

Amazon Prometheus

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

エラスティックロードバランシング

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

Amazon GuardDuty

[Amazon GuardDuty](#) で EKS クラスターへの脅威を検出します。

AWS Resilience Hub

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

Amazon EKS の料金

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

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

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

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

Amazon EKS の一般的なユースケース

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

高可用性アプリケーションのデプロイ

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

マイクロサービスアーキテクチャの構築

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

ソフトウェアリリースプロセスの自動化

自動化されたアプリケーションの構築、テスト、デプロイのプロセスを簡素化する、継続的インテグレーションおよび継続的デプロイ (CI/CD) パイプラインを管理します。

サーバーレスアプリケーションの実行

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

機械学習ワークロードの実行

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

オンプレミスとクラウドで一貫性のあるデプロイ

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

費用対効果の高いバッチ処理とビッグデータワークロードの実行

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

アプリケーションの保護とコンプライアンスの確保

[AWS Identity and Access Management](#) (IAM)、[Amazon Virtual Private Cloud](#) (Amazon VPC)、[AWS Key Management Service](#) (AWS KMS) などの AWS セキュリティサービスと統合されている Amazon EKS を使用して、強力なセキュリティ対策を実施し、コンプライアンスを維持します。これにより、業界標準に準拠したデータのプライバシーと保護が保証されます。

Amazon EKS アーキテクチャ

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

コントロールプレーン

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

分散コンポーネント

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

最適なパフォーマンス

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

耐障害性

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

一貫したアップタイム

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

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

コンピューティング

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

EKS Auto Mode

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

AWS Fargate

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

Karpenter

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

マネージドノードグループ

[マネージド型ノードグループ](#) は、Amazon EKS クラスター内の Amazon EC2 インスタンスのコレクションを管理するための自動化とカスタマイズを組み合わせたものです。AWS がノードのパッチ適用、更新、スケーリングなどのタスクを処理することで、運用面を容易にします。それと同時に、カスタム kubelet 引数がサポートされていることで、高度な CPU およびメモリ管理ポリシーの可能性が広がります。さらに、サービスアカウントの AWS Identity and Access Management (IAM) ロールによってセキュリティを強化しながら、クラスターごとに個別のアクセス許可を設定する必要性を抑えます。

セルフマネージド型ノード

[セルフマネージド型ノード](#) では、Amazon EKS クラスター内の Amazon EC2 インスタンスを完全に制御できます。ユーザーは、ノードの管理、スケーリング、メンテナンスを担当すること

で、基盤となるインフラストラクチャを完全に制御できます。このオプションは、ノードをきめ細かく制御およびカスタマイズする必要があり、インフラストラクチャの管理と保守に時間を費やす準備ができていないユーザーに適しています。

Amazon EKS Hybrid Nodes

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

Kubernetes の概念

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

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

トピック

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

このコンテンツに含まれているリンクをクリックすると、Amazon EKS と Kubernetes の両方のドキュメントに記載されている Kubernetes の概念について解説したページが開き、ここで取り上げた

トピックをさらに深く掘り下げることができます。Amazon EKS がどのように Kubernetes コントロールプレーンとコンピューティング機能を実装するかの詳細については「[the section called “アーキテクチャ”](#)」を参照してください。

Kubernetes が選ばれる理由

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

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

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

Kubernetes の属性

上記の目標を達成するため、Kubernetes は次の属性を使用します。

- コンテナ化 - Kubernetes はコンテナオーケストレーションツールです。Kubernetes を使用するには、まずアプリケーションをコンテナ化する必要があります。アプリケーションの種類に応じて、これはマイクロサービスのセット、バッチジョブ、その他形式のいずれかになります。それにより、アプリケーションで、ツールの巨大なエコシステムを含む Kubernetes ワークフローを利用することができます。そこではコンテナを[イメージとしてコンテナレジストリ](#)に保存したり、Kubernetes [クラスター](#)にデプロイしたり、利用可能な[ノード](#)上で実行したりすることができます。個々のコンテナを Kubernetes クラスターにデプロイする前に、Docker または別の[コンテナランタイム](#)を使用してローカルのコンピュータ上で構築しテストすることができます。

- スケーラブル - アプリケーションを実行しているインスタンスのキャパシティを超えてアプリケーションの需要が拡大した場合は、Kubernetes をスケールアップすることができます。必要に応じて、Kubernetes はアプリケーションに追加の CPU やメモリが必要かどうかを判断し、自動的に利用可能な容量を拡張するか、既存の容量を追加で利用します。アプリケーションでより多くのインスタンスを実行するための十分なコンピューティングがある場合はポッドレベルでスケールリングを実行することができます ([水平ポッド自動スケールリング](#))。あるいは容量の増加に対応するためにより多くのノードを起動する必要がある場合はノードレベルでスケールリングを実行することができます ([クラスターオートスケーラー](#) または [Karpenter](#))。容量が不要になると、これらのサービスは不要なポッドを削除して、不要なノードをシャットダウンすることができます。
- 使用可能 - アプリケーションまたはノードが異常な状態になった場合や使用できなくなった場合、Kubernetes は実行中のワークロードを使用可能な別のノードに移動できます。この問題はワークロードを実行しているワークロードまたはノードの実行中のインスタンスを削除すれば、強制的に解決できます。ここでのポイントはワークロードを現在の場所で実行できなくなったときは他の場所で起動できるという点です。
- 宣言型 - Kubernetes は、アクティブな照合を使用して、クラスターに宣言した状態が実際の状態と一致していることを常にチェックします。[Kubernetes オブジェクト](#) をクラスターに適用すると、通常は YAML 形式の設定ファイルを通じて、例えばクラスター上で実行するワークロードの起動をリクエストすることができます。後で設定に変更を加えることができます。例えば、新しいバージョンのコンテナを使用する、メモリをさらに割り当てるといったことができます。Kubernetes は、目的の状態を確立するためにやるべきことをやります。例えば、ノードの起動や停止、ワークロードの停止および再起動、更新したコンテナの取得などを実行してください。
- コンポーザブル - アプリケーションは通常、複数のコンポーネントで構成されているため、これら一連のコンポーネント (通常は複数のコンテナで表されます) をまとめて管理できるようにしておくことをお勧めします。これを、Docker Compose では Docker を使用して直接実行するのに対して、Kubernetes では [Kompose](#) コマンドを利用して実行します。その方法の例については「[Translate a Compose File to Kubernetes Resources](#)」を参照してください。
- 拡張可能 - 独自のソフトウェアとは異なり、オープンソースの Kubernetes プロジェクトでは二ーズに合わせて Kubernetes を自由に拡張できるようになっています。API と設定ファイルは直接変更することができます。サードパーティーは、インフラストラクチャとエンドユーザー Kubernetes 機能の両方を拡張する場合、独自の[コントローラー](#)を作成するよう奨励されています。[Webhook](#) を使用すると、クラスターを設定して、ポリシーを適用し、状況の変化に対応することができます。Kubernetes クラスターを拡張する方法の詳細については、「[Kubernetes を拡張する](#)」を参照してください。
- ポータブル - 多くの組織では、アプリケーションの二ーズをすべて同じ方法で管理できることから、自社での Kubernetes の運用を標準化しています。デベロッパーはコンテナ化されたアプリ

ケーションを同じパイプラインを使って構築し、保存することができます。その後、オンプレミス、クラウド、レストランの POS 端末、企業のリモートサイトに分散する IOT デバイスなどで稼働している Kubernetes クラスターにこうしたアプリケーションをデプロイできます。オープンソースであるため、このような特殊な Kubernetes ディストリビューションをその管理に必要なツールと併せて開発することができます。

Kubernetes の管理

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

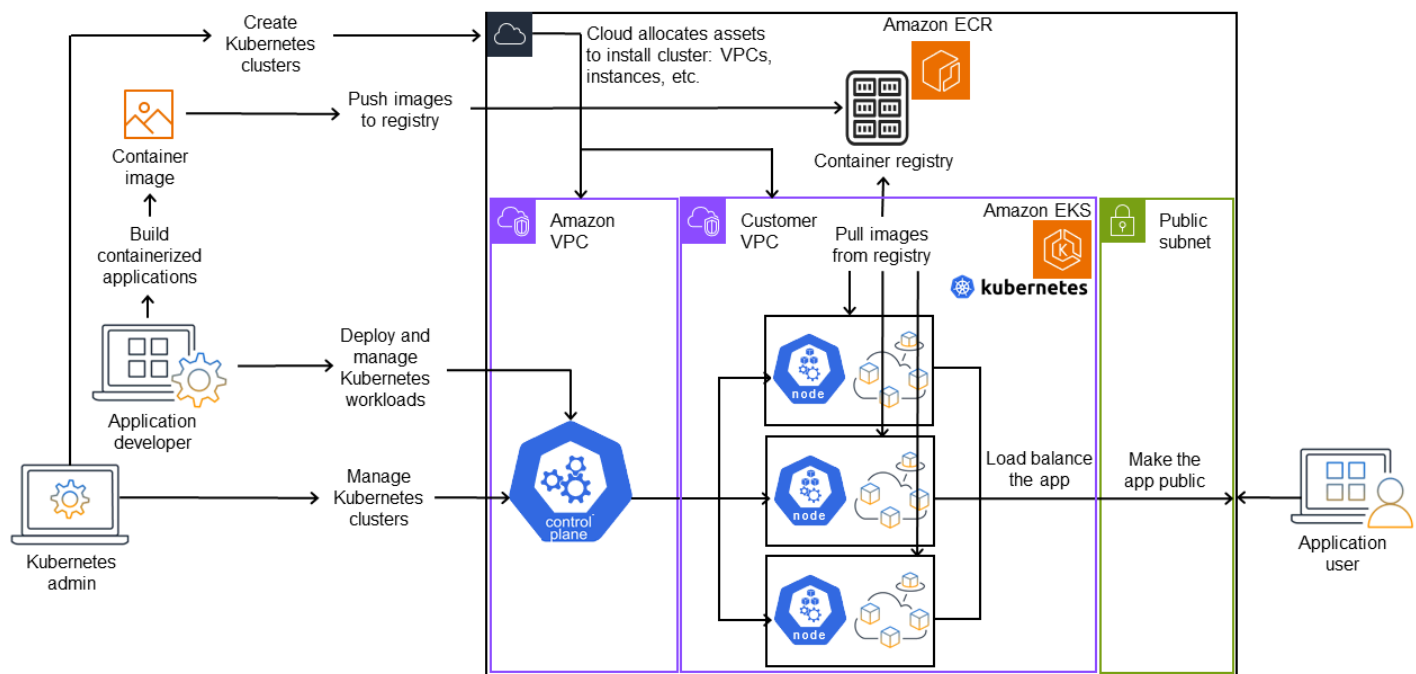
- **ハードウェア** - 要件に応じて Kubernetes を実行できるハードウェアがない場合、AWS Amazon EKS などのクラウドプロバイダーを利用すると初期費用を節約できます。Amazon EKS ではコンピュートインスタンス (Amazon Elastic Compute Cloud)、独自のプライベート環境 (Amazon VPC)、アイデンティティとアクセス許可の一元管理 (IAM)、ストレージ (Amazon EBS) など、AWS で提供されている最善のクラウドリソースを利用できます。AWS はコンピュータ、ネットワーク、データセンター、および Kubernetes の実行に必要な他のすべての物理コンポーネントを管理します。同様に、需要が最も高い日に最大容量を処理できるようにデータセンターを計画しておく必要はありません。Amazon EKS Anywhere やその他のオンプレミスの Kubernetes クラスターの場合、Kubernetes デプロイに使用されるインフラストラクチャの管理は利用する側の責任となりますが、引き続き AWS を利用して Kubernetes を最新の状態に保つことができます。
- **コントロールプレーン管理** - Amazon EKS は AWS がホストしている Kubernetes コントロールプレーンのセキュリティと可用性を管理します。コンテナのスケジュール、アプリケーションの可用性の管理、その他の重要なタスクはこのコントロールプレーンが行うため、ユーザーはアプリケーションのワークロードに専念できます。クラスターが故障した場合、AWS には実行状態に復元するための方法がいくつかあります。Amazon EKS Anywhere の場合、コントロールプレーンを自分で管理することになります。
- **テスト済みのアップグレード** - クラスターをアップグレードするときは、Amazon EKS または Amazon EKS Anywhere を利用して、Kubernetes ディストリビューションのテスト済みバージョンを用意できます。

- アドオン - 拡張して Kubernetes と連携するように構築された数百ものプロジェクトがあり、クラスターのインフラストラクチャに追加したり、ワークロードの実行を支援するために使用したりできます。これらのアドオンを自分で構築して管理する代わりに、AWS にはクラスターで利用できる [the section called “Amazon EKS アドオン”](#) Amazon EKS アドオンが用意されています。Amazon EKS Anywhere には多くの一般的なオープンソースプロジェクトのビルドを含む、[キュレーションパッケージ](#)があります。そのため、ユーザーはソフトウェアを自分で構築したり、重要なセキュリティパッチ、バグ修正、アップグレードを管理したりする必要はありません。同様に、デフォルトの設定がニーズを満たしていれば、通常、それらのアドオンの設定はほとんど必要ありません。アドオンを使用してクラスターを拡張する方法については「[the section called “拡張クラスター”](#)」を参照してください。

Kubernetes の動作

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

A Kubernetes cluster in action



Kubernetes 管理者は、クラスターが構築されるプロバイダーのタイプに固有のツールを使用して、Kubernetes クラスターを作成します。この例では、Amazon EKS という Kubernetes マネージ

ドサービスを提供する AWS クラウドをプロバイダーとして使用します。マネージドサービスは、クラスター用に仮想プライベートクラウド (Amazon VPC) を新たに 2 つ作成する、ネットワークを設定する、クラウドアセット管理用に新しい VPC に対して Kubernetes アクセス許可を直接マッピングするなど、クラスターの作成に必要なリソースを自動的に割り当てます。マネージドサービスは、コントロールプレーンサービスの実行先を確認し、ワークロードを実行するための Kubernetes ノードとして 0 個以上の Amazon EC2 インスタンスを割り当てます。AWS はコントロールプレーン用に 1 つの Amazon VPC 自体を管理し、もう 1 つの Amazon VPC にはワークロードを実行するカスタマーノードが含まれます。

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

このクラスターにワークロードをデプロイする場合、アプリケーションデベロッパーはいくつかのタスクを実行することができます。デベロッパーはアプリケーションを 1 つ以上のコンテナイメージに構築し、そのイメージを Kubernetes クラスターにアクセスできるコンテナレジストリにプッシュする必要があります。AWS には、それを行うための Amazon Elastic Container Registry (Amazon ECR) というレジストリがあります。

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

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

クラスター

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

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

クラスターの作成および管理ツール

Kubernetes プロジェクトには、Kubernetes クラスターを手動で作成するためのツールが用意されています。したがって、Kubernetes を単一のマシンにインストールしたり、マシン上でコントロールプレーンを実行したり、ノードを手動で追加したりする場合は、Kubernetes [インストールツール](#)に一覧表示されている [kind](#)、[minikube](#)、[kubeadm](#) などの CLI ツールを使用できます。クラスターの作成と管理のライフサイクル全体を簡素化して自動化するには、Amazon EKS や Amazon EKS Anywhere といった定評ある Kubernetes プロバイダーがサポートしているツールを使用してください。作業がはるかに簡単になります。

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

- マネージドコントロールプレーン - AWS はコントロールプレーンをユーザーに代わって管理し、AWS アベイラビリティーゾーン全体で利用可能にするため、Amazon EKS クラスターが利用可能かつスケーラブルになります。
- ノード管理 - ノードを手動で追加する代わりに、[the section called “マネージド型ノードグループ”](#)マネージドノードグループまたは [Karpenter](#) を使用して、必要に応じて Amazon EKS でノードを自動的に作成できます。マネージドノードグループは、Kubernetes [クラスター自動スケーリング](#)と統合されています。ノード管理ツールを使用すると、[スポットインスタンス](#)やノード統合などによりコストを節約したり、ワークロードのデプロイ方法やノードの選択方法を設定する[スケジューリング機能](#)を使用して可用性を実現したりすることができます。
- クラスターネットワーク - `eksctl` は、クラウドフォーメーション テンプレートを使用して、Kubernetes クラスター内のコントロールプレーンとデータプレーン (ノード) コンポーネントとの間にネットワークを設定します。また、内部と外部の通信を行うためのエンドポイントも設定します。詳細については「[ツールのインストール Amazon EKS ワーカーノードのクラスターネットワークキングの謎を解く](#)」を参照してください。Amazon EKS 内のポッド間の通信は [the section](#)

called “Pod Identity” Amazon EKS Pod Identity を使用して行われます。これによりポッドは認証情報やアクセス許可を管理する AWS クラウドの方法を利用できるようになります。

- アドオン - Amazon EKS を使用すると、Kubernetes クラスターをサポートするためによく使用されるソフトウェアコンポーネントを構築して追加する必要がなくなります。例えば、AWS Management Console から Amazon EKS クラスターを作成すると、Amazon EKS kube-proxy ([the section called “kube-proxy”](#))、Amazon VPC CNI plugin for Kubernetes ([the section called “Amazon VPC CNI”](#))、CoreDNS ([the section called “CoreDNS”](#)) アドオンが自動的に追加されます。利用可能なアドオンも含む、これらのアドオンの詳細については「[the section called “Amazon EKS アドオン”](#)」を参照してください。

クラスターをオンプレミスのコンピュータとネットワーク上で実行できるように、Amazon は [Amazon EKS Anywhere](#) を提供しています。AWS クラウドをプロバイダーにする代わりに、Amazon EKS Anywhere を [VMware vSphere](#)、[ベアメタル \(ティンカーベル プロバイダー\)](#)、[Snow](#)、[CloudStack](#)、[Nutanix](#) の各プラットフォームで、独自の機器を使用して実行することを選択できます。

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

クラスターコンポーネント

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

コントロールプレーン

コントロールプレーンはクラスターを管理する一連のサービスで構成されています。これらのサービスはすべて 1 台のコンピュータで実行される場合もあれば、複数のコンピュータに分散される場合もあります。内部的にはこれらはコントロールプレーンインスタンス (CPI) と呼ばれます。CPI の実

行方法はクラスターの規模と、高可用性の要件に応じて異なります。クラスターの需要が増えると、コントロールプレーンサービスはスケールしてそのサービスのインスタンスを増やし、リクエストはインスタンス間で負荷分散されます。

Kubernetes コントロールプレーンのコンポーネントが実行するタスクには、以下のものがあります。

- クラスターコンポーネント (API サーバー) との通信 - API サーバー ([kube-apiserver](#)) はクラスターへのリクエストをクラスターの内部と外部の両方から行えるようにするために Kubernetes API を公開します。つまり、クラスターのオブジェクト (ポッド、サービス、ノードなど) を追加または変更するリクエストはポッドを実行する `kubectl` のリクエストなどの外部コマンドから送信することができます。同様に、ポッドのステータスを `kubelet` サービスにクエリするときのように、API サーバーからクラスター内のコンポーネントに対してリクエストを実行することができます。
- クラスターに関するデータの保存 (`etcd` キーバリューストア)* - サービスはクラスターの現在の状態を追跡するという重要な役割を担っています。etcdetcd サービスにアクセスできなくなると、ワークロードはしばらく実行し続けますが、クラスターの状態を更新したりクエリしたりすることはできなくなります。そのため、重要なクラスターは通常、複数の負荷分散された etcd サービスのインスタンスを同時に実行し、データの損失や破損に備えて、etcd キーバリューストアのバックアップを定期的に行っています。Amazon EKS では、これはすべてデフォルトで自動的に処理されます。Amazon EKS Anywhere は [etcd のバックアップと復元](#) の手順を公開しています。によるデータ管理の方法についてはhttps://etcd.io/docs/v3.5/learning/data_model/ Data Model etcd を参照してください。
- ノードへの Pod のスケジューリング (スケジューラー) - Kubernetes での Pod の起動または停止のリクエストは、[Kubernetes スケジューラー \(kube-scheduler\)](#) に送信されます。クラスターにはポッドを実行できるノードが複数含まれている場合があるため、どのノード (レプリカの場合は複数のノード) でポッドを実行するかはスケジューラーが決定します。リクエストされたポッドを既存のノードで実行するための十分な容量がない場合、他のポッドをプロビジョニングしていなければそのリクエストは失敗します。プロビジョニングには新しいノードを自動的に起動してワークロードを処理する[the section called “マネージド型ノードグループ”](#) マネージド型ノードグループや [Karpenter](#) などのサービスを有効化する作業が含まれます。
- コンポーネントを目的の状態に維持する (コントローラーマネージャー) - Kubernetes コントローラーマネージャーは、クラスターの状態をモニタリングしたり、クラスターに変更を加えて期待される状態に戻したりするデーモンプロセス ([kube-controller-manager](#)) として実行されます。特に、statefulset-controller、endpoint-controller、cronjob-controller、node-controller など、さまざまな Kubernetes オブジェクトを監視するコントローラーがいくつかあります。

- クラウドリソースの管理 (クラウドコントローラマネージャー) - Kubernetes とその基盤となるデータセンターリソースのリクエストを実行するクラウドプロバイダーとのやり取りは、[クラウドコントローラマネージャー \(cloud-controller-manager\)](#) が処理します。クラウド コントローラマネージャー が管理するコントローラーにはルートコントローラー (クラウドネットワークルートの設定用)、サービスコントローラー (クラウドのロードバランシングサービスを使用するためのもの)、ノードライフサイクルコントローラー (クラウド API を使用してライフサイクル全体でノードを Kubernetes と同期させるためのもの) があります。

ワーカーノード (データプレーン)

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

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

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

拡張クラスター

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

クラスターに追加できるアドオンにはさまざまな種類があります。クラスターを健全な状態に保つにはログ記録、監査、メトリクスなどの作業を実行できる、[クラスターをモニタリングする](#) オブザーバビリティ機能を追加します。この情報があれば、多くの場合、同じオブザーバビリティインターフェイスを使用して、発生した問題をトラブルシューティングすることができます。こうしたタイプのサービスには、[Amazon GuardDuty](#)、CloudWatch ([the section called “Amazon CloudWatch”](#) を参照)、[AWS Distro for OpenTelemetry](#)、Amazon VPC CNI plugin for Kubernetes ([the section called “Amazon VPC CNI”](#) を参照)、[Grafana Kubernetes Monitoring](#) などがあります。[アプリデータストレージ](#) ストレージでは Amazon EKS のアドオンには [the section called “Amazon EBS”](#) Amazon エラスティック・ブロック・ストア CSI ドライバー (ブロックストレージデバイスの追加用)、[the section called “Amazon EFS”](#) Amazon エラスティック・ファイル・システム CSI ドライバー (ファイルシステムストレージの追加用)、サードパーティー製ストレージのアドオン ([the section called “Amazon FSx for NetApp ONTAP”](#) Amazon FSx for NetApp ONTAP CSI ドライバーなど) などがあります。

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

ワークロード

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

コンテナ

Kubernetes にデプロイして管理するアプリケーションワークロードの最も基本的な要素が [Pod](#) です。ポッドはアプリケーションのコンポーネントを格納する方法であるだけでなく、ポッドの属性

を記述する仕様を定義する方法でもあります。RPM や Deb パッケージは Linux システム用のソフトウェアをまとめてパッケージ化しますが、それ自体はエンティティとしては動作しません。

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

ポッドの仕様 ([PodSpec](#)) はポッドの望ましい状態を定義します。ワークロードリソースを使用して[ポッドテンプレート](#)を管理することで、個別のポッドまたは複数のポッドをデプロイできます。ワークロードリソースには[展開](#) (複数のポッドレプリカを管理する)、[StatefulSets](#) (データベースポッドなどの一意にする必要があるポッドをデプロイする)、[DaemonSets](#) (ポッドをすべてのノードで継続的に実行する必要がある場合) が含まれます。詳細は後ほど説明します。

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

コンテナの構築

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

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

- ベースイメージ - ベースのコンテナイメージは通常はオペレーティングシステムのファイルシステム ([レッドハット・エンタープライズ Linux](#) や [Ubuntu](#) など) の最小バージョン、または特定の種類のアプリケーション ([nodejs](#) や [python](#) アプリケーションなど) を実行するソフトウェアを提供するように拡張された最小システムのいずれかから構築されるコンテナです。

- アプリケーションソフトウェア - Linux システムに追加するときとはほぼ同じ方法で、アプリケーションソフトウェアをコンテナに追加できます。例えば、Dockerfile では `npm` や `yarn` を実行して Java アプリケーションをインストールするか、`yum` や `dnf` を実行して RPM パッケージをインストールすることができます。つまり、Dockerfile で `RUN` コマンドを使用すると、ベースイメージのファイルシステムで使用可能な任意のコマンドを実行して、生成されたコンテナイメージの内部でソフトウェアをインストールしたり、設定したりできます。
- インストラクション - [Dockerfile 参照](#) には Dockerfile の設定時に追加できるインストラクションが記載されています。これにはコンテナ自体の中にあるもの (ローカルシステムの `ADD` または `COPY` ファイル) の構築、コンテナを実行するときに実行するコマンドの特定 (`CMD` または `ENTRYPOINT`)、コンテナを実行するシステムへのコンテナの接続 (実行する `USER`、マウントするローカル `VOLUME`、`EXPOSE` するポート) などに使用する指示が含まれています。

`docker` コマンドとサービスは従来、コンテナ (`docker build`) の構築に使用されてきましたが、コンテナイメージの構築に使用できるその他ツールには [podman](#) や [nerdctl](#) などがあります。コンテナの構築方法については「[より良いコンテナ・イメージの構築](#)」または「[Dockerビルドの概要](#)」参照してください。

コンテナの保存

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

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

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

コンテナの実行

コンテナは標準フォーマットで構築されているため、コンテナランタイム (Docker など) を実行でき、コンテンツがローカルマシンのアーキテクチャ (x86_64 や arm など) に一致するマシンであれ

ば、どのマシンでもコンテナを実行できます。コンテナをテストしたり、ローカルのデスクトップで実行したりするには `docker run` または `podman run` コマンドを使用して、ローカルホストでコンテナを起動します。ただし Kubernetes では、各ワーカーノードにコンテナランタイムがデプロイされており、そのノードにコンテナの実行をリクエストするかどうかは Kubernetes が決定します。

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

ポッド

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

ポッドの設定

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

- ストレージ - 実行中のコンテナを停止して削除すると、より永続的なストレージを設定しない限り、そのコンテナ内のデータストレージは削除されます。Kubernetes は、多種多様なストレージタイプをサポートしており、[Volumes](#) の制御の下でタイプを抽出します。ストレージタイプには [CephFS](#)、[NFS](#)、[iSCSI](#) などがあります。ローカルのコンピュータから [ローカルブロックデバイス](#) を使用することもできます。クラスターからこれらのストレージタイプのいずれかを使用すれば、ストレージボリュームをコンテナのファイルシステム内の選択したマウントポイントにマウントできます。[永続ボリューム](#) はポッドが削除された後も残り、[エフェメラルボリューム](#) はポッドが削除されると、削除されます。クラスター管理者がクラスター用に異なる [ストレージクラス](#) を作成した場合、使用後にボリュームを削除するか、再利用するか、スペースがさらに必要になった場合に拡張するか、また、特定のパフォーマンス要件を満たすようにするかなど、使用するストレージの属性を選択できることがあります。
- シークレット - ポッド仕様のコンテナで [シークレット](#) を使用できるようにすると、ファイルシステム、データベース、その他保護されているアセットにアクセスするために必要な許可をそれらのコンテナに付与できます。キー、パスワード、トークンはシークレットとして保存できるアイテムです。シークレットを使用すると、この情報をコンテナイメージに保存する必要はなく、実行中の

コンテナでシークレットを利用できるようにすることができます。シークレットに似ているのが [ConfigMaps](#) です。ConfigMap には一般に、サービスを設定するためのキーと値のペアなど、重要度が低い情報が格納されます。

- コンテナリソース - コンテナを詳細に設定するためのオブジェクトはリソース設定の形式をとることができます。コンテナごとに、そのコンテナが使用できるメモリと CPU の量をリクエストできるほか、コンテナが使用できるリソースの合計量の上限を設定できます。例については「[Resource Management for Pods and Containers](#)」を参照してください。
- 中断 - ポッドは意図せずに (ノードがダウンしたとき)、または意図的に (アップグレードが必要なとき) 中断することがあります。 [ポッド中断の予算](#) を設定することで、中断が起きた場合のアプリケーションの可用性をある程度制御することができます。アプリケーションの例については「[中断予算の指定](#)」を参照してください。
- 名前空間 - Kubernetes には、Kubernetes コンポーネントとワークロードを互いに分離するさまざまな方法が用意されています。特定のアプリケーションのすべてのポッドを、同じ [名前空間](#) で実行することはそれらのポッドをまとめて保護し管理する一般的な方法です。独自の名前空間を作成して使用することも、名前空間を指定しない (Kubernetes が default の名前空間を使用する) ように選択することもできます。Kubernetes コントロールプレーンコンポーネントは通常、[kube-system](#) 名前空間で動作します。

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

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

ポッドのデプロイ

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

- ステートレスアプリケーション - ステートレスアプリケーションはクライアントのセッションデータを保存しないため、その次のセッションは前のセッションで起きたことを参照する必要がありません。これにより、ポッドが異常な状態になった場合は新しいポッドと交換するか、状態を保存せずにポッドを移動させるだけで済みます。 [ステートレスアプリケーション \(ウェブサーバーなど\)](#) を実行している場合は [Deployment](#) を使用して [Pods](#) と [ReplicaSets](#) をデプロイできま

す。ReplicaSet は同時に実行したいポッドのインスタンス数を定義します。ReplicaSet は直接実行することもできますが、1 回に実行するポッドのレプリカ数を定義するため、デプロイ内でレプリカを直接実行するのが一般的です。

- ステートフルアプリケーション - ステートフルアプリケーションとはポッドの ID とポッドの起動順序が重要であるアプリケーションのことです。これらのアプリケーションには安定した永続的ストレージが必要であり、一貫した方法でデプロイしスケールインする必要があります。Kubernetes にステートフルアプリケーションをデプロイするには、[StatefulSets](#) を使用します。通常、StatefulSet として実行されるアプリケーションの一例が、データベースです。StatefulSet 内ではレプリカ、ポッドとそのコンテナ、マウントするストレージボリューム、データを保存するコンテナ内の場所を定義できます。ReplicaSet としてデプロイされるデータベースの例については「[レプリケートされたステートフル アプリ](#)」を参照してください。
- ノードあたりのアプリケーション - アプリケーションを Kubernetes クラスター内のすべてのノードで実行しなければならない場合があります。例えば、データセンターで、すべてのコンピュータでモニタリングアプリケーションまたは特定のリモートアクセスサービスを実行する必要がある場合です。Kubernetes では、[DaemonSet](#) を使用することで、選択したアプリケーションがクラスター内のすべてのノードで実行されるようにすることができます。
- 完了まで実行するアプリケーション - 特定のタスクを完了するために、複数のアプリケーションを実行する必要がある場合があります。例えば、毎月のステータスレポートを実行したり、古いデータを消去したりする場合です。[Job](#) オブジェクトを使用すると、アプリケーションを起動して実行し、タスクが完了したら、終了するように設定することができます。[CronJob](#) オブジェクトを使用すると、Linux の [crontab](#) 形式で定義された構造を使用して、特定の時間、分、日、月、曜日に、アプリケーションを実行するように設定できます。

ネットワークからアプリケーションにアクセスできるようにする

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

- サービス - ポッドは異なるノードやアドレスに移動できるため、最初のポッドと通信する必要のある別のポッドがそのポッドの場所を特定することが困難になる場合があります。この問題を解決するため、Kubernetes ではアプリケーションを [Service](#) として表すことができます。サービスを使用すると、特定の名前を持つポッドまたはポッドのセットを識別し、そのアプリケーションのサー

ピスをポッドから公開するポートと、他のアプリケーションがそのサービスにアクセスする際に使用できるポートを指定することができます。クラスター内の別の Pod は名前で Service をリクエストでき、Kubernetes はそのリクエストをその Service を実行している Pod のインスタンスの適切なポートに転送します。

- イングレス – [イングレス](#)は、Kubernetes Service によって表されたアプリケーションをクラスター外のクライアントから利用できるようにします。イングレス の基本機能にはロードバランサー (イングレス が管理)、イングレス コントローラー、コントローラーから サービス にリクエストを送信するルールなどがあります。Kubernetes で選択できる [イングレスコントローラー](#)がいくつかあります。

次のステップ

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

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

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

Amazon EKS デプロイオプションを理解する

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

クラウドでは、Amazon EKS は Kubernetes コントロールプレーンとノードの Kubernetes クラスターインフラストラクチャ管理を自動化します。これは、コンテナのスケジューリング、アプリケーションの可用性の管理、リソースの動的なスケーリング、コンピューティングの最適化、クラスターデータの保存、その他の重要な機能の実行に不可欠です。Amazon EKS を使用すると、AWS インフラストラクチャの堅牢なパフォーマンス、スケーラビリティ、信頼性、可用性とともに、AWS の

ネットワーキング、セキュリティ、ストレージ、オブザーバビリティサービスとのネイティブな統合を実現できます。

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

クラウド内の Amazon EKS

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

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

	AWS リージョンの Amazon EKS	Local/Wavelength Zones の Amazon EKS
	• AWS Fargate	
Kubernetes データプレーンの場所	AWS リージョン	AWS Local Zones または Wavelength Zones

データセンターまたはエッジ環境の Amazon EKS

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

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

エアギャップ環境用の Amazon EKS Anywhere

[Amazon EKS Anywhere](#) は、オンプレミス環境とエッジ環境でのインフラストラクチャのセットアップや Kubernetes クラスターのライフサイクルオペレーションなど、差別化につながらない手間のかかる作業を自動化することで、Kubernetes クラスターの管理を簡素化します。Amazon EKS とは異なり、Amazon EKS Anywhere はカスタマーマネージド製品であり、お客様は Amazon EKS Anywhere クラスターのライフサイクルオペレーションとメンテナンスを担当します。Amazon EKS Anywhere は Kubernetes サブプロジェクトクラスター API (CAPI) 上に構築されており、VMware vSphere、ベアメタル、Nutanix、Apache CloudStack、AWS Snow などのさまざまなインフラストラクチャをサポートしています。Amazon EKS Anywhere はエアギャップ環境で実行でき、オブザーバビリティと ID 管理のためのリージョナル AWS サービスとの統合を選択的に利用できます。Amazon EKS Anywhere のサポートと AWS が提供する Kubernetes アドオンへのアクセスを受けるには、[Amazon EKS Anywhere エンタープライズサブスクリプション](#)を購入できます。

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

Amazon EKS ツール

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

[Amazon EKS Distro](#) は、すべての Amazon EKS サービスを強化する基盤となる Kubernetes コンポーネントの AWS ディストリビューションです。これには、Kubernetes コントロールプレーンコンポーネント (etcd、kube-apiserver、kube-scheduler、kube-controller-manager) やネットワークコンポーネント (CoreDNS、kube-proxy、CNI プラグイン) など、機能する Kubernetes クラスターに

必要なコアコンポーネントが含まれています。Amazon EKS Distro を使用して、選択したツールで Kubernetes クラスターを自己管理できます。Amazon EKS Distro デプロイは、AWS サポートプランの対象外です。

Amazon EKS を使用するようにセットアップする

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

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

次のステップ

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

AWS CLI の設定

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

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

アクセスキーを作成するには

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

AWS CLI を構成するには

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

1. ターミナルウィンドウで、以下のコマンドを入力します。

```
aws configure
```

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

2. AWS 認証情報を入力します。例：

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

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

セキュリティトークンを取得するには

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

デフォルトでは、トークンは 15 分間有効です。デフォルトのセッションタイムアウトを変更するには、`--duration-seconds` フラグを渡します。例：

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

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

```
{
  "Credentials": {
    "AccessKeyId": "ASIA5FTRU3LOEXAMPLE",
    "SecretAccessKey": "JnKgvmfQUD9mNsPoi9IbxAYEXAMPLE",
    "SessionToken": "VERYLONGSESSIONTOKENSTRING",
    "Expiration": "2023-02-17T03:14:24+00:00"
  }
}
```

ユーザー ID を確認するには

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

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

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

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

次のステップ

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

kubectl および eksctl のセットアップ

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

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

kubectl をインストールまたは更新する

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

Note

Amazon EKS クラスターコントロールプレーンとのマイナーバージョンの相違が 1 つ以内である kubectl バージョンを使用する必要があります。例えば、1.31 kubectl クライアントは Kubernetes 1.30、1.31、および 1.32 クラスターで動作します。

ステップ 1: kubectl がインストールされているかどうかを確認する

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

```
kubectl version --client
```

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

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

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

ステップ 2: kubectl をインストールまたは更新する

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

- [the section called “macOS”](#)
- [the section called “Linux \(amd64\)”](#)
- [the section called “Linux \(arm64\)”](#)
- [the section called “Windows”](#)

Note

このセクションで使用されている AWS リージョンからご使用の AWS リージョンへのダウンロードに時間がかかる場合は、コンテンツを前面に表示するように CloudFront を設定することを検討してください。詳細については、「[Get started with a basic CloudFront distribution](#)」を参照してください。

macOS

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

- 必要な Kubernetes バージョンのバイナリを選択してダウンロードします。
- 必要に応じて、バイナリのチェックサムを確認します。
- バイナリのアクセス許可に実行を追加します。

- バイナリを PATH のフォルダにコピーします。
- 必要に応じて、バイナリのディレクトリを PATH に追加します。

手順:

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

- Kubernetes 1.33

```
curl -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.33.0/2025-05-01/bin/darwin/amd64/kubectl
```

- Kubernetes 1.32

```
curl -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.32.3/2025-04-17/bin/darwin/amd64/kubectl
```

- Kubernetes 1.31

```
curl -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.31.7/2025-04-17/bin/darwin/amd64/kubectl
```

- Kubernetes 1.30

```
curl -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.30.11/2025-04-17/bin/darwin/amd64/kubectl
```

- Kubernetes 1.29

```
curl -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.29.15/2025-04-17/bin/darwin/amd64/kubectl
```

- Kubernetes 1.28

```
curl -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.28.15/2025-04-17/bin/darwin/amd64/kubectl
```

- Kubernetes 1.27

```
curl -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.27.16/2025-01-10/bin/darwin/amd64/kubectl
```

- Kubernetes 1.26

```
curl -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.26.15/2024-12-12/bin/darwin/amd64/kubectl
```

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

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

- Kubernetes 1.33

```
curl -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.33.0/2025-05-01/bin/darwin/amd64/kubectl.sha256
```

- Kubernetes 1.32

```
curl -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.32.3/2025-04-17/bin/darwin/amd64/kubectl.sha256
```

- Kubernetes 1.31

```
curl -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.31.7/2025-04-17/bin/darwin/amd64/kubectl.sha256
```

- Kubernetes 1.30

```
curl -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.30.11/2025-04-17/bin/darwin/amd64/kubectl.sha256
```

- Kubernetes 1.29

```
curl -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.29.15/2025-04-17/bin/darwin/amd64/kubectl.sha256
```

- Kubernetes 1.28

```
curl -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.28.15/2025-04-17/bin/darwin/amd64/kubectl.sha256
```

- Kubernetes 1.27


```
curl -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.27.16/2025-01-10/bin/darwin/amd64/kubectl.sha256
```

- Kubernetes 1.26

```
curl -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.26.15/2024-12-12/bin/darwin/amd64/kubectl.sha256
```

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

```
openssl sha1 -sha256 kubectl
```

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

3. バイナリへの実行アクセス許可を適用します。

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

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

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

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

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

Linux (amd64)

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

- 必要な Kubernetes バージョンのバイナリを選択してダウンロードします。
- 必要に応じて、バイナリのチェックサムを確認します。
- バイナリのアクセス許可に実行を追加します。

- バイナリを PATH のフォルダにコピーします。
- 必要に応じて、バイナリのディレクトリを PATH に追加します。

手順:

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

- Kubernetes 1.33

```
curl -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.33.0/2025-05-01/bin/linux/amd64/kubectl
```

- Kubernetes 1.32

```
curl -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.32.3/2025-04-17/bin/linux/amd64/kubectl
```

- Kubernetes 1.31

```
curl -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.31.7/2025-04-17/bin/linux/amd64/kubectl
```

- Kubernetes 1.30

```
curl -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.30.11/2025-04-17/bin/linux/amd64/kubectl
```

- Kubernetes 1.29

```
curl -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.29.15/2025-04-17/bin/linux/amd64/kubectl
```

- Kubernetes 1.28

```
curl -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.28.15/2025-04-17/bin/linux/amd64/kubectl
```

- Kubernetes 1.27

```
curl -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.27.16/2024-12-12/bin/linux/amd64/kubectl
```

- Kubernetes 1.26

```
curl -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.26.15/2024-12-12/bin/linux/amd64/kubectl
```

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

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

- Kubernetes 1.33

```
curl -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.33.0/2025-05-01/bin/linux/amd64/kubectl.sha256
```

- Kubernetes 1.32

```
curl -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.32.3/2025-04-17/bin/linux/amd64/kubectl.sha256
```

- Kubernetes 1.31

```
curl -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.31.7/2025-04-17/bin/linux/amd64/kubectl.sha256
```

- Kubernetes 1.30

```
curl -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.30.11/2025-04-17/bin/linux/amd64/kubectl.sha256
```

- Kubernetes 1.29

```
curl -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.29.15/2025-04-17/bin/linux/amd64/kubectl.sha256
```

- Kubernetes 1.28

```
curl -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.28.15/2025-04-17/bin/  
linux/amd64/kubectl.sha256
```

- Kubernetes 1.27

```
curl -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.27.16/2024-12-12/bin/  
linux/amd64/kubectl.sha256
```

- Kubernetes 1.26

```
curl -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.26.15/2024-12-12/bin/  
linux/amd64/kubectl.sha256
```

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

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

or

```
openssl sha1 -sha256 kubectl
```

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

3. バイナリへの実行アクセス許可を適用します。

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

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

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

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

 Note

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

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

Linux (arm64)

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

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

手順:

1. クラスターの Kubernetes バージョンの `kubectl` バイナリを Amazon S3 からダウンロードします。
2. クラスターの Kubernetes バージョンの `kubectl` バイナリを Amazon S3 からダウンロードします。
 - Kubernetes 1.33

```
curl -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.33.0/2025-05-01/bin/linux/arm64/kubectl
```

- Kubernetes 1.32

```
curl -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.32.3/2025-04-17/bin/linux/arm64/kubectl
```

- Kubernetes 1.31

```
curl -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.31.7/2025-04-17/bin/linux/arm64/kubectl
```

- Kubernetes 1.30

```
curl -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.30.11/2025-04-17/bin/linux/arm64/kubectl
```

- Kubernetes 1.29

```
curl -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.29.15/2025-04-17/bin/linux/arm64/kubectl
```

- Kubernetes 1.28

```
curl -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.28.15/2025-04-17/bin/linux/arm64/kubectl
```

- Kubernetes 1.27

```
curl -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.27.16/2024-12-12/bin/linux/arm64/kubectl
```

- Kubernetes 1.26

```
curl -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.26.15/2024-12-12/bin/linux/arm64/kubectl
```

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

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

- Kubernetes 1.33

```
curl -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.33.0/2025-05-01/bin/linux/arm64/kubectl.sha256
```

- Kubernetes 1.32

```
curl -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.32.3/2025-04-17/bin/
linux/arm64/kubectl.sha256
```

- Kubernetes 1.31

```
curl -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.31.7/2025-04-17/bin/
linux/arm64/kubectl.sha256
```

- Kubernetes 1.30

```
curl -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.30.11/2025-04-17/bin/
linux/arm64/kubectl.sha256
```

- Kubernetes 1.29

```
curl -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.29.15/2025-04-17/bin/
linux/arm64/kubectl.sha256
```

- Kubernetes 1.28

```
curl -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.28.15/2025-04-17/bin/
linux/arm64/kubectl.sha256
```

- Kubernetes 1.27

```
curl -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.27.16/2024-12-12/bin/
linux/arm64/kubectl.sha256
```

- Kubernetes 1.26

```
curl -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.26.15/2024-12-12/bin/
linux/arm64/kubectl.sha256
```

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

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

or

```
openssl sha1 -sha256 kubectl
```

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

4. バイナリへの実行アクセス許可を適用します。

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

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

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

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

 Note

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

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

Windows

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

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

手順:

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

- Kubernetes 1.33

```
curl.exe -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.33.0/2025-05-01/bin/windows/amd64/kubectl.exe
```

- Kubernetes 1.32

```
curl.exe -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.32.3/2025-04-17/bin/windows/amd64/kubectl.exe
```

- Kubernetes 1.31

```
curl.exe -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.31.7/2025-04-17/bin/windows/amd64/kubectl.exe
```

- Kubernetes 1.30

```
curl.exe -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.30.11/2025-04-17/bin/windows/amd64/kubectl.exe
```

- Kubernetes 1.29

```
curl.exe -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.29.15/2025-04-17/bin/windows/amd64/kubectl.exe
```

- Kubernetes 1.28

```
curl.exe -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.28.15/2025-04-17/bin/windows/amd64/kubectl.exe
```

- Kubernetes 1.27

```
curl.exe -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.27.16/2024-12-12/bin/windows/amd64/kubectl.exe
```

- Kubernetes 1.26

```
curl.exe -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.26.15/2024-12-12/bin/windows/amd64/kubectl.exe
```

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

- a. クラスターの Kubernetes バージョンに応じて、SHA-256 チェックサム (Windows 版) をダウンロードします。

- Kubernetes 1.33

```
curl.exe -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.33.0/2025-05-01/bin/windows/amd64/kubectl.exe.sha256
```

- Kubernetes 1.32

```
curl.exe -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.32.3/2025-04-17/bin/windows/amd64/kubectl.exe.sha256
```

- Kubernetes 1.31

```
curl.exe -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.31.7/2025-04-17/bin/windows/amd64/kubectl.exe.sha256
```

- Kubernetes 1.30

```
curl.exe -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.30.11/2025-04-17/bin/windows/amd64/kubectl.exe.sha256
```

- Kubernetes 1.29

```
curl.exe -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.29.15/2025-04-17/bin/windows/amd64/kubectl.exe.sha256
```

- Kubernetes 1.28

```
curl.exe -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.28.15/2025-04-17/bin/windows/amd64/kubectl.exe.sha256
```

- Kubernetes 1.27

```
curl.exe -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.27.16/2024-12-12/bin/windows/amd64/kubectl.exe.sha256
```

- Kubernetes 1.26

```
curl.exe -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.26.15/2024-12-12/bin/windows/amd64/kubectl.exe.sha256
```

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

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

- c. 出力で生成されたチェックサムが、ダウンロードした `kubectl.sha256` ファイルのチェックサムと一致することを確認してください。PowerShell の出力は、大文字の同じ文字列である必要があります。
4. バイナリを PATH のフォルダにコピーします。コマンドラインユーティリティで使用する PATH 内に既存のディレクトリがある場合、そのディレクトリにバイナリをコピーします。それ以外の場合は、以下の手順を完了します。
- a. コマンドラインのバイナリ用に新しいディレクトリ (`C:\bin` など) を作成します。
 - b. `kubectl.exe` バイナリを新しいディレクトリにコピーします。
 - c. ユーザーまたはシステムの PATH 環境変数を編集し、新しいディレクトリを PATH に追加します。
 - d. 新しい PATH 変数を取得するには、PowerShell ターミナルを閉じ、新しいターミナルを開きます。
5. `kubectl` をインストールしたら、そのバージョンを確認できます。

```
kubectl version --client
```

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

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

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

eksctl をインストールする

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

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

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

次のステップ

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

クイックスタート: ウェブアプリケーションをデプロイしてデータを保存する

ゲームアプリケーションをデプロイし、そのデータを Amazon EKS に保持する

このクイックスタートチュートリアルでは、[eksctl](#) を使用して 2048 ゲームサンプルアプリケーションをデプロイし、そのデータを Amazon EKS Auto Mode クラスターに保持する手順を示します。Amazon EKS Auto Mode は、クラスターブロックストレージ、ネットワーク、ロードバランシング、コンピューティングの自動スケーリングのルーチンタスクを自動化します。

先に進んだら、クラスターのセットアッププロセスについて説明します。Amazon EKS Auto Mode は、EC2 マネージドインスタンスを使用したノードの作成、Application Load Balancer の作成、EBS ボリュームの作成のタスクを自動化します。

全体としては、AWS サービスとの完全統合に必要なカスタム注釈を含むサンプルワークロードをデプロイします。

このチュートリアルでは、次の作業を行いました。

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

VPC 設定 以下の eksctl クラスターテンプレートを使用すると、eksctl はクラスターの IPv4 仮想プライベートクラウド (VPC) を自動的に作成します。デフォルトでは、eksctl はパブリックエンドポイントとプライベートエンドポイントの両方を作成するだけでなく、すべてのネットワーク要件に対応する VPC を設定します。

インスタンス管理 EKS Auto Mode は、Kubernetes アプリケーションの需要に基づいて、EKS クラスター内のノードを動的に追加または削除します。

データ永続性 EKS Auto Mode のブロックストレージ機能を使用して、ポッドの再起動や障害が発生するシナリオでも、アプリケーションデータの永続性を確保します。

外部アプリケーションアクセス EKS Auto Mode の負荷分散機能を使用して、Application Load Balancer (ALB) を動的にプロビジョニングします。

前提条件

開始する前に、Amazon EKS を使用するように以下の前提条件が設定されていることを確認してください:

- AWS CLI のセットアップと認証情報の設定
- eksctl をインストールする
- kubectl をインストールする

詳細については、「[セットアップする](#)」を参照してください。

クラスターを設定する

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

cluster-config.yaml ファイルを作成して、次の内容を貼り付けます。region-code を、us-east-1 などの有効なリージョンに置き換えます。

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

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

autoModeConfig:
  enabled: true
```

これで、クラスターを作成する準備が整いました。

Amazon EKS クラスターを作成する:

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

⚠ Important

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

IngressClass を作成する

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

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

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

IngressClass をクラスターに適用します。

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

2048 ゲームサンプルアプリケーションをデプロイする

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

Note

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

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

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

以下のようなレスポンス出力が表示されます。

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

2. [2048 ゲームサンプルアプリケーション](#)をデプロイします。

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

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

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

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

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

以下のようなレスポンス出力が表示されます。

NAME	CLASS	HOSTS	ADDRESS
		PORTS	AGE


```
ingress-2048    alb      *      k8s-game2048-ingress2-eb379a0f83-378466616.region-  
code.elb.amazonaws.com    80      31s
```

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

4. ウェブブラウザを開き、前のステップの ADDRESS を入力してウェブアプリケーションにアクセスします。例：

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

ブラウザに 2048 ゲームが表示されるはずです。プレイしてください!

2048

SCORE

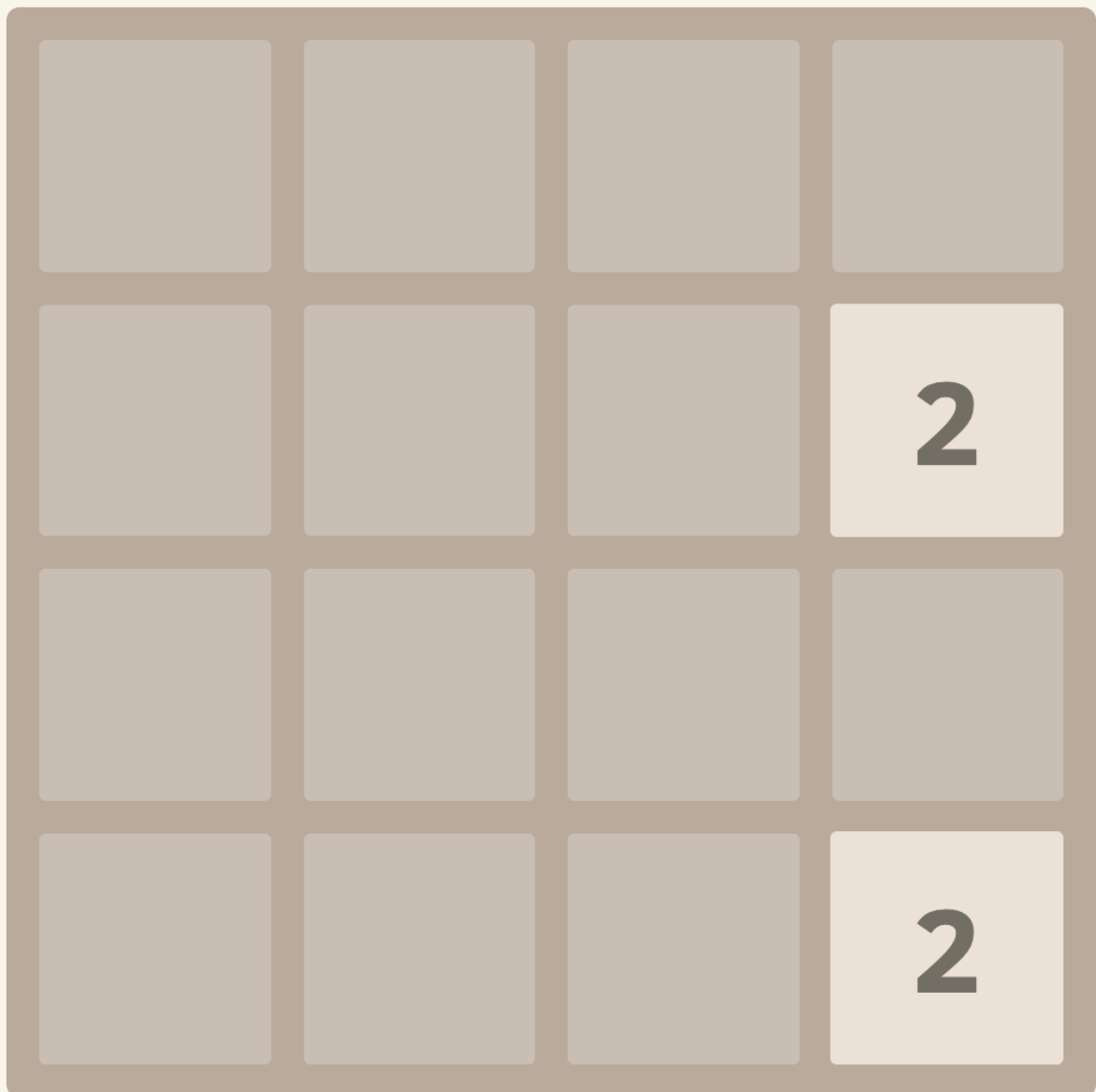
0

BEST

48

Join the numbers and get to the **2048** tile!

New Game



Amazon EKS 自動モードを使用してデータを永続化する

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

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

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

2. `StorageClass` を適用します。

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

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

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

4. `PVC` をクラスターに適用します。

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

以下のようなレスポンス出力が表示されます。

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

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

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

6. 更新されたデプロイを適用します。

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

以下のようなレスポンス出力が表示されます。

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

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

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

クラスターとノードを削除する

このチュートリアルのために作成したクラスターの使用が終了したら、クリーンアップのために、次のコマンドを使用してそれらのクラスターを削除する必要があります。クリーンアップする前にこのクラスターでやる必要がある場合は、「次のステップ」を参照してください。

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

EKS は、クラスターが削除されたときにプロビジョニングされたノードを自動的にクリーンアップします。

Amazon EKS を例で学ぶ

概要

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

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

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

Amazon EKS ワークショップ

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

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

Amazon EKS の実践的なクラスターセットアップチュートリアル

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

クラスターの構築

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

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

クラスターの強化

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

- [EFS CSI によるストレージソリューションを提供する](#)
- [EBS CSI による動的データベースストレージを提供する](#)
- [AWS Load Balancer Controller を使用して IPv4 クラスターでアプリケーションを公開する](#)
- [AWS Load Balancer Controller を使用して IPv6 クラスターでアプリケーションを公開する](#)

AWS サービスの最適化

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

- [ExternalDNS を使用してマイクロサービスの DNS レコードを管理する](#)
- [CloudWatch でアプリケーションをモニタリングする](#)

- [SQS および EFS ストレージを使用して非同期タスクを管理する](#)
- [ワークロードから AWS Secrets Manager シークレットを使用する](#)
- [Fargate、NGINX、および ACM PCA で mTLS をセットアップする](#)

Amazon EKS のサンプル

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

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

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

AWS のチュートリアル

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

- 「[Deploy a Container Web App on Amazon EKS](#)」

- 「[Run Kubernetes clusters for less \(Amazon EKS and Spot instances\)](#)」
- 「[How to cost optimize Jenkins jobs on Kubernetes](#)」

開発者ワークショップ

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

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

Terraform ワークショップ

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

AWS Amazon EKS トレーニング

AWS では、Amazon EKS について学習するための正式なトレーニングを提供しています。

「[Running Containers on Amazon Elastic Kubernetes Service](#)」という 3 日間のトレーニングコースでは、次のことを学習します。

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

- コスト、効率、回復性を改善するためのテクニック

Amazon EKS の使用を開始する

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

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

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

また、以下のリファレンスも提供しています。

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

Amazon EKS – EKS Auto Mode の使用開始

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

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

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

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

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

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

Amazon EKS – eksctl の使用を開始する

Note

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

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

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

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

前提条件

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

ステップ 1: Amazon EKS クラスターとノードを作成する

Important

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

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

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

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

Example

Fargate - Linux

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

Managed nodes - Linux

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

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

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

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

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

ステップ 2: Kubernetes リソースを表示する

1. クラスターノードを表示します。

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

出力例は次のとおりです。

Example

Fargate - Linux

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

Managed nodes - Linux

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

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

2. クラスターで実行されているワークロードを表示します。

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

出力例は次のとおりです。

Example

Fargate - Linux

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

Managed nodes - Linux

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

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

ステップ 3: クラスターとノードを削除する

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

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

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

次のステップ

以下のトピックはクラスターの機能を拡張するのに役立ちます。

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

Amazon EKS の使用を開始する – AWS Management Console と AWS CLI

Note

このトピックではEKS 自動モードを使用せずに開始する方法について説明します。管理されたノードグループを使用してノードをデプロイします。

EKS 自動モードはクラスターコンピューティング、ストレージ、ネットワーキングのルーチンタスクを自動化します。[Amazon EKS 自動モードの使用を開始する方法について説明します。](#)ノードをデプロイするにはEKS 自動モードが推奨されます。

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

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

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

前提条件

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

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

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

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

ステップ 1: Amazon EKS クラスターを作成する

⚠ Important

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

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

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

💡 Tip

前のコマンドが作成したすべてのリソースの一覧については AWS コンソール の <https://console.aws.amazon.com/cloudformation> 開きます：*my-eks-vpc-stack* スタックを選択し、[Resources] (リソース タブ) を選択してください。

2. クラスターの IAM ロールを作成して、そのロールに必要な Amazon EKS IAM 管理ポリシーをアタッチします。Amazon EKS によって管理される Kubernetes クラスターは、そのサービスで使用するリソースを管理するために、ユーザーに代わって他の AWS のサービスを呼び出します。
 - a. 次の内容を *eks-cluster-role-trust-policy.json* という名前のファイルにコピーします。

```
{  
  "Version": "2012-10-17",  
  "Statement": [  
    {  
      "Effect": "Allow",  
      "Action": "sts:AssumeRole",  
      "Resource": "arn:aws:iam::123456789012:role/eks-cluster-role"    }  
  ]  
}
```

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

- b. ロールを作成します。

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

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

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

3. <https://console.aws.amazon.com/eks/home#/clusters> で Amazon EKS コンソールを開きます。

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

4. [クラスターを作成] を選択してください。このオプションが表示されない場合はまず左のナビゲーションペインの [クラスター] を選択してください。
5. [クラスターの設定] ページで、次の手順を実行してください：
- a. カスタム設定を選択し、Use EKS Auto Modeを無効にします。(EKS 自動モードクラスターを使用する場合は代わりに「」を参照してください) [the section called “マネジメントコンソール”](#)。
 - b. [名前] に、クラスターの名前 (など) #####。この名前には英数字 (大文字と小文字が区別されます) とハイフンのみを使用できます。先頭の文字は英数字である必要があります。また、100 文字より長くすることはできません。名前はクラスターを作成する AWS リージョンおよび AWS アカウント内で一意である必要があります。
 - c. [クラスターサービス役割] で、「*myAmazonEKSClusterRole*」を選択してください。
 - d. その他の設定はデフォルト値のままにし、[次へ] をクリックします。

6. [ネットワーキングを指定] ページで、以下の作業を行います：

- a. 前のステップで [VPC] ドロップダウンリストから作成した VPC の ID を選択してください。* `/ my-eks-vpc-stack-VPC` に類似した ID です。
- b. 前のステップで作成したサブネットを[サブネット] のドロップダウンリストの中から選択します。このサブネットは * `/ my-eks-vpc-stack-*` のように表示されます。
- c. 前のステップで作成したセキュリティグループを [追加のセキュリティグループ] のドロップダウンリストの中から選択します。このグループは * `/ my-eks-vpc-stack-ControlPlaneSecurityGroup-*` のように表示されます。
- d. その他の設定はデフォルト値のままにし、[次へ] をクリックします。

7. [オブザーバビリティの設定] ページで、[次へ] を選択してください。

8. [アドオンの選択] ページで、[次へ] を選択してください。

アドオンの詳細については「[the section called “Amazon EKS アドオン”](#)」を参照してください。

9. [選択したアドオンセッティングの設定] ページで、[次へ] を選択してください。

10. [確認して作成] ページで、[作成] をクリックします。

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

Note

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

ステップ 2: 自分のコンピュータでクラスターとの通信を設定する

このセクションではクラスター用の kubeconfig ファイルを作成します。このファイルの設定により、kubectl CLI からクラスターへの通信ができるようになります。

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

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

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

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

2. 設定をテストします。

```
kubectl get svc
```

Note

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

出力例は次のとおりです。

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

ステップ 3: ノードを作成する

Important

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

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

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

EC2 Linux マネージド型ノードグループを作成するには

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

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "ec2.amazonaws.com"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
```

- b. ノードの IAM ロールを作成します。

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

- c. ロールに、必要なマネージド IAM ポリシーをアタッチします。

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



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

- d. <https://console.aws.amazon.com/eks/home#/clusters> で Amazon EKS コンソールを開きます。
 - e. [ステップ 1: Amazon EKS クラスターを作成する](#) で作成したクラスターの名前 (##### など) 選択してください：
 - f. [#####] ページで、次の操作を行います：
 - g. [コンピューティング] タブを開きます。
 - h. [ノードグループを追加] をクリックします。
2. [ノードグループの設定] ページで以下を実行してください：
- a. [名前] に、マネージド型ノードグループの一意の名前 (##### など) 入力してください。ノードグループ名は 63 文字以下である必要があります。先頭は文字または数字でなければなりません。残りの文字にはハイフンおよびアンダースコアを含めることもできます。
 - b. [ノード IAM ロール名] で、前のステップで作成した *myAmazonEKSNodeRole* ロールを選択してください。各ノードグループには独自の一意の IAM ロールを使用することをお勧めします。
 - c. [次へ] を選択してください。
3. [コンピューティングとスケーリングの設定] ページではデフォルトの値を受け入れ、[次へ] をクリックします。
4. [ネットワーキングの指定] ページではデフォルトの値を受け入れ、[次へ] をクリックします。
5. [確認と作成] ページで、マネージド型ノードグループの設定を確認し、[作成] を選択してください。
6. 数分後、[ノードグループの設定] セクションにある [ステータス] の表示が、[作成中] から [アクティブ] に変わります。ステータスが [アクティブ] になるまで、次のステップに進まないでください。

ステップ 4: リソースを表示する

ノードと Kubernetes のワークロードを表示することができます。

1. 左のナビゲーションペインで [クラスター] を選択してください。[クラスター] のリストで ##### ## など、作成したクラスターの名前を選択してください。

2. [#####] ページで、次の項目を選択してください：

- a. [コンピューティング] タブ – クラスターにデプロイした [ノード] が一覧表示されます。ノードの名前を選択すると、そのノードに関するより詳細な情報が表示されます。
- b. [リソース] タブ – Amazon EKS クラスターにデフォルトでデプロイされたすべての Kubernetes リソースが表示されます。コンソールでリソースタイプを選択すると、その詳細を確認できます。

ステップ 5: リソースを削除する

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

1. ここで作成した、いずれかのノードグループもしくは Fargate プロファイルを削除してください。
 - a. <https://console.aws.amazon.com/eks/home#/clusters> で Amazon EKS コンソールを開きます。
 - b. 左のナビゲーションペインで [クラスター] を選択してください。クラスターのリストから ##### を選択してください。
 - c. [コンピューティング] タブを開きます。
 - d. ノードグループを作成している場合はノードグループ「#####」を選択した上で、[削除] をクリックしてください。##### と入力し、[削除] を選択してください。
 - e. ノードグループプロファイルが削除されるまで続行しないでください。
2. クラスターを削除します。
 - a. 左のナビゲーションペインで [クラスター] を選択してください。クラスターのリストから ##### を選択してください。
 - b. [クラスターの削除] を選択してください。
 - c. ##### と入力し、[削除] を選択してください。クラスターが削除されるまで続行しないでください。
3. 作成した VPC AWS CloudFormation スタックを削除します。
 - a. クラウドフォーメーション コンソールの <https://console.aws.amazon.com/cloudformation/> 開きます：
 - b. 「*my-eks-vpc-stack*」スタックを選択した上で、[削除] をクリックしてください。
 - c. [*my-eks-vpc-stack* の削除] 確認ダイアログボックスで、[スタックを削除] を選択してください。

4. 作成した IAM ロールを削除します。

- a. IAM コンソール の <https://console.aws.amazon.com/iam/> 開きます :
- b. 左のナビゲーションペインで、[ロール] を選択してください。
- c. リスト (**[myAmazonEKSClusterRole]** または **[myAmazonEKSNodeRole]** から、作成した各ロールを選択してください。[削除] を選択し、要求された確認テキストを入力した後で、[削除] をクリックします。

次のステップ

以下のトピックはクラスターの機能を拡張するのに役立ちます。

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

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

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

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

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

機能

EKS Auto Mode には次の高度な機能があります。

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

アプリケーションの可用性: EKS Auto Mode は、Kubernetes アプリケーションの需要に基づいて EKS クラスター内のノードを動的に追加または削除します。この結果、手動キャパシティプランニングの必要性が最小限に抑えられ、アプリケーションの可用性が確保されます。

効率: EKS Auto Mode は、NodePool とワークロードの要件で定義された柔軟性を維持しながら、計算コストを最適化するように設計されています。また、未使用のインスタンスを終了し、ワークロードを他のノードに統合して、コスト効率を向上させます。

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

自動アップグレード: EKS Auto Mode は、設定されたポッド中断の予算 (PDB) や NodePool 中断の予算 (NDB) を維持しながら、Kubernetes クラスター、ノード、関連コンポーネントを最新のパッチにより最新版に保ちます。PDB やその他の設定をブロックすることで更新が妨げられる場合、最大 21 日間の最大存続期間まで介入が必要になる場合があります。

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

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

自動コンポーネント

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

以下は、自動化されているデータプレーンコンポーネントのリストです。

- **コンピューティング:** 多くのワークロードでは、EKS Auto Mode を使用すると、EKS クラスターのコンピューティングの多くの側面を気にする必要がなくなります。具体的には次のとおりです。
 - **ノード:** EKS Auto Mode ノードは、アプライアンスのように扱われるように設計されています。EKS Auto Mode は以下を実行します。
 - ワークロードを介入なしで実行するのに必要な多くのサービスを備えて設定された適切な AMI を選択します。

- SELinux 強制モードと読み取り専用のルートファイルシステムを使用して、AMI 上のファイルへのアクセスをロックダウンします。
- SSH または SSM アクセスを禁止することで、ノードへの直接アクセスを防止します。
- GPU サポートがあり、NVIDIA GPU および Neuron GPU 用のカーネルドライバとプラグインが分離されているため、高性能ワークロードが実現されます。
- 自動スケーリング: EKS Auto Mode は [Karpenter](#) 自動スケーリングに依存して、スケジュール不可能なポッドをモニタリングし、新しいノードをデプロイしてそのポッドを実行できるようにします。ワークロードが終了すると、不要になったノードは EKS Auto Mode によって動的に中断および終了され、リソースの使用状況が最適化されます。
- アップグレード: ノードを制御することで、EKS Auto Mode が必要に応じてセキュリティパッチやオペレーティングシステム、コンポーネントのアップグレードを提供する機能が効率化されます。これらのアップグレードは、ワークロードの中断を最小限に抑えるように設計されています。EKS Auto Mode は、ソフトウェアと API を最新にするために、ノードの最大存続期間を強制的に 21 日に設定します。
- ロードバランシング: EKS Auto Mode は、Amazon の Elastic Load Balancing サービスと統合することでロードバランシングを合理化し、Kubernetes サービスおよびインGRESSリソースに対するロードバランサーのプロビジョニングと設定を自動化します。また、Application Load Balancer と Network Load Balancer の両方の高度な機能をサポートし、それらのライフサイクルを管理し、それらをクラスターの需要に合わせてスケールします。この統合により、AWS ベストプラクティスに準拠した本番環境対応のロードバランシングソリューションが提供されるため、インフラストラクチャ管理ではなくアプリケーションに集中できます。
- ストレージ: EKS Auto Mode は、ノード終了時にボリュームタイプ、ボリュームサイズ、暗号化ポリシー、削除ポリシーを設定することで、エフェメラルストレージを設定します。
- ネットワーキング: EKS Auto Mode は、ポッドとサービスの接続に関する重要なネットワーキングタスクを自動化します。これには、IPv4/IPv6 サポートと、IP アドレススペースを拡張するためのセカンダリ CIDR ブロックの使用が含まれます。
- Identity and Access Management: EKS Auto Mode クラスターに EKS Pod Identity Agent をインストールする必要はありません。

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

設定

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

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

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

Amazon EKS 自動モードI クラスターを作成する

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

Note

EKS 自動モード には Kubernetes バージョン 1.29 以降が必要です。EKS Auto Mode は ap-southeast-7 または mx-central-1 の AWS リージョンでは使用できません。

ニーズに応じて任意のツールを選択する: AWS Management Console には、EKS Auto Mode の機能について学習し、個々のクラスターを作成するのに最適なビジュアルインターフェイスが用意されています。AWS CLI は特にクラスター作成を既存のワークフローまたは CI/CD パイプラインに統合する場合、スクリプト作成および自動化タスクに最適です。eksctl CLI は Kubernetes ネイティブのエクスペリエンスを提供し、実用的なデフォルトにより簡素化されたコマンドラインオペレーションを求める、Kubernetes ツールに精通しているユーザーにお勧めです。

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

AWS CLI、AWS Management Console、または eksctl CLI を使用して、Amazon EKS Auto Mode でクラスターを作成できます。

トピック

- [eksctl CLI を使用して EKS Auto Mode クラスターを作成する](#)
- [AWS CLI を使用して EKS Auto Mode クラスターを作成する](#)
- [AWS Management Console を使用して EKS Auto Mode クラスターを作成する](#)

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

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

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

Note

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

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

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

次のコマンドを実行して、新しい EKS Auto Mode クラスターを作成します

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

前提条件

- AWS コマンドラインインターフェイス (AWS CLI) の最新バージョンがデバイスにインストールおよび設定されていること。現在のバージョンを確認するには「`aws --version`」を参照してください。最新バージョンをインストールするには、「AWS コマンドラインインターフェイスユー

ザーガイド」の「[インストール](#)」および「[aws configure を使用したクイック設定](#)」を参照してください。

- 十分な IAM アクセス許可を使用して CLI にログインし、IAM ポリシー、IAM ロール、EKS クラスターなどの AWS リソースを作成します。
- kubectl コマンドラインツールがデバイスにインストールされている。AWS では、EKS クラスターの Kubernetes バージョンと同じ kubectl バージョンを使用することをお勧めします。kubectl をインストールまたはアップグレードするには、「[the section called “kubectl と eksctl のセットアップ”](#)」を参照してください。

VPC サブネットを指定する

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

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

デフォルト VPC のサブネット ID を検出するには

AWS CLI の使用:

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

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

2. 出力を保存し、サブネット ID をメモします。

サンプル出力:

```
-----
```

```

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

```

EKS Auto Mode クラスターの IAM ロール

クラスター IAM ロール

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

- [AmazonEKSCheckpointPolicy](#)
- [アマゾンEKSブロックストレージポリシー](#)
- [アマゾンEKSロードバランシングポリシー](#)
- [アマゾンEKSネットワークングポリシー](#)
- [アマゾンEKSクラスターポリシー](#)

ノード IAM ロール

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

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

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

- [AmazonEKSECSWorkerNodeMinimalPolicy](#)
- [AmazonEC2ContainerRegistryPullOnly](#)

サービスにリンクされたロール

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

EKS Auto Mode クラスター IAM ロールを作成する

ステップ 1: 信頼ポリシーを作成する

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

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

ステップ 2: IAM ロールを作成する

信頼ポリシーを使用して、クラスター IAM ロールを作成します。

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

ステップ 3: ロール ARN をメモする

以降のステップで使用する新しいロールの ARN を取得して保存します。

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

ステップ 4: 必要なポリシーをアタッチする

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

AmazonEKSClusterPolicy:

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

アマゾンEKSコンピュートポリシー:

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

アマゾンEKSブロックストレージポリシー:

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

アマゾンEKSロードバランシングポリシー:

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

AmazonEKSNetworkingPolicy:

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

EKS Auto Mode ノード IAM ロールを作成する

ステップ 1: 信頼ポリシーを作成する

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

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "ec2.amazonaws.com"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
```

ステップ 2: ノード IAM ロールを作成する

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

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

ステップ 3: ロール ARN をメモする

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

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

ステップ 4: 必要なポリシーをアタッチする

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

AmazonEKSWorkerNodeMinimalPolicy:

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

AmazonEC2ContainerRegistryPullOnly:

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

EKS Auto Mode クラスターを作成する

概要

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

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

デフォルトのクラスター設定

クラスターを作成する前に、以下のデフォルト値および機能を確認してください。

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

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

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

コマンドを実行する

次のコマンドを使用してクラスターを作成します。

```
aws eks create-cluster \  
  --region ${AWS_REGION} \  
  --cli-input-json \  
  "{  
    \"name\": \"${CLUSTER_NAME}\",  
    \"version\": \"${K8S_VERSION}\",  
    \"roleArn\": \"${CLUSTER_ROLE_ARN}\",  
    \"resourcesVpcConfig\": {  
      \"subnetIds\": ${SUBNETS_JSON},  
      \"endpointPublicAccess\": true,  
      \"endpointPrivateAccess\": true  
    },  
    \"computeConfig\": {  
      \"enabled\": true,  
      \"nodeRoleArn\": \"${NODE_ROLE_ARN}\",  
      \"nodePools\": [\"general-purpose\", \"system\"]  
    },  
    \"kubernetesNetworkConfig\": {  
      \"elasticLoadBalancing\": {  
        \"enabled\": true  
      }  
    },  
    \"storageConfig\": {  
      \"blockStorage\": {  
        \"enabled\": true  
      }  
    },  
    \"accessConfig\": {  
      \"authenticationMode\": \"API\"  
    }  
  }  
}
```

クラスターのステータスを確認する

ステップ 1: クラスターの作成を確認する

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

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

ステップ 2: kubeconfig を更新する

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

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

ステップ 3: ノードプールを検証する

次のコマンドを使用して、クラスター内のノードプールを一覧表示します。

```
kubectl get nodepools
```

次のステップ

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

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

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

コンソールでクラスターを作成するオプションは 2 つあります。

- クイック設定 (EKS Auto Mode を使用)
- カスタム設定

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

クイック設定オプションを使用して EKS Auto Mode を作成する

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

1. EKS コンソールに移動します
2. [クラスターを作成] をクリックします
3. [クイック設定] オプションが選択されていることを確認します
4. 次の値を決定するか、テストクラスター用のデフォルトを使用します。
 - クラスター名
 - Kubernetes バージョン
5. クラスター IAM ロールを選択します。EKS Auto Mode クラスターを初めて作成する場合は、[推奨ロールを作成] オプションを使用します。
 - 必要に応じて、すべての EKS Auto Mode クラスターに対して、AWS アカウントで 1 つのクラスター IAM ロールを再利用できます。
 - クラスター IAM ロールには、EC2 インスタンス、EBS ボリューム、EC2 ロードバランサーなどのリソースを管理するのに必要な EKS Auto Mode のアクセス許可が含まれています。
 - [推奨ロールを作成] オプションでは、すべてのフィールドに推奨値が事前入力されています。[次へ]、[作成] の順に選択します。ロールは推奨される AmazonEKSAutoClusterRole という名前を使用します。
 - 最近新しいロールを作成した場合は、[更新] アイコンを使用してロール選択ドロップダウンを再ロードします。
6. ノード IAM ロールを選択します。EKS Auto Mode クラスターを初めて作成する場合は、[推奨ロールを作成] オプションを使用します。
 - 必要に応じて、すべての EKS Auto Mode クラスターに対して、AWS アカウントで 1 つのノード IAM ロールを再利用できます。
 - ノード IAM ロールには、Auto Mode ノードがクラスターに接続するのに必要なアクセス許可が含まれています。ノード IAM ロールには、コンテナの ECR イメージを取得するためのアクセス許可が付与されている必要があります。
 - [推奨ロールを作成] オプションでは、すべてのフィールドに推奨値が事前入力されています。[次へ]、[作成] の順に選択します。ロールは推奨される AmazonEKSAutoNodeRole という名前を使用します。

- 最近新しいロールを作成した場合は、[更新] アイコンを使用してロール選択ドロップダウンを再ロードします。
7. EKS Auto Mode クラスターの VPC を選択します。[VPC を作成] を選択して EKS 用の新しい VPC を作成するか、以前に EKS 用に作成した VPC を選択します。
 - VPC コンソールを使用して新しい VPC を作成する場合は、AWS ではアベイラビリティゾーンごとに少なくとも 1 つの NAT ゲートウェイを作成することをお勧めします。それ以外の場合は、他のすべてのデフォルトを使用できます。
 - さらなる情報と IPv6 クラスターの要件の詳細については、「[the section called “「VPC を作成する」”](#)」を参照してください。
 8. (任意) EKS Auto Mode は、選択した VPC のプライベートサブネットを自動的に入力します。不要なサブネットは削除できます。
 - EKS では、ベストプラクティスに従って VPC からプライベートサブネットが自動的に選択されます。必要に応じて、パブリックサブネットなど、VPC から追加のサブネットを選択できます。
 9. (任意) [クイック設定のデフォルトを表示] を選択して、新しいクラスターのすべての設定値を確認します。この表は、クラスターの作成後には一部の値を編集できないことを示しています。
 - 10[Create cluster (クラスターの作成)] を選択します。クラスターの作成が完了するまでに 15 分かかる場合があります。

次のステップ

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

既存の EKS クラスターで EKS 自動モードを有効にする

既存の EKS クラスターで EKS 自動モード を有効にできます。

Note

EKS 自動モード には Kubernetes バージョン 1.29 以降が必要です。EKS 自動モードは ap-southeast-7 または mx-central-1 AWS リージョンでは使用できません。

AWS では次の移行がサポートされています。

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

AWS では以下の移行はサポートされていません。

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

Amazon EKS 自動モード クラスターに AWS ロードバランサー コントローラー をインストールできます。IngressClass または loadBalancerClass のオプションを使用して、サービスリソースとイングレスリソースを ロードバランサー コントローラー または EKS 自動モード のいずれかに関連付けます。
- 代替 CNI またはその他のサポートされていないネットワーク設定を使用した EKS クラスターの移行

移行のリファレンス

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

機能	[リソース]	フィールド	セルフマネージド型	EKS 自動モード
ブロッkstレージ	StorageClass	provisioner	ebs.csi.aws.com	ebs.csi.eks.amazonaws.com
負荷分散	Service	loadBalancerClass	service.k8s.aws/nlb	eks.amazonaws.com/nlb
負荷分散	IngressClass	controller	ingress.k8s.aws/alb	eks.amazonaws.com/alb
負荷分散	IngressClassParams	apiversion	elbv2.k8s.aws/v1beta1	eks.amazonaws.com/v1
負荷分散	TargetGroupBinding	apiversion	elbv2.k8s.aws/v1beta1	eks.amazonaws.com/v1
コンピューティング	NodeClass	apiVersion	karpenter.sh/v1alpha5	eks.amazonaws.com/v1

EBS ボリュームの移行

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

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

永続ボリュームを移行するには、次の手順を実行します。

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

考慮事項

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

ロードバランサーの移行

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

サービスの中断を最小限に抑えるには DNS ベースのトラフィックシフトアプローチをお勧めします。まず、既存の設定を稼働させたまま、EKS 自動モードを使用して新しいロードバランサーを作成します。次に、DNS ルーティング (Route 53 など) を使用して、古いロードバランサーから新しいロードバランサーにトラフィックを徐々に移行します。トラフィックが正常に移行され、新しい設定を確認したら、古いロードバランサーとセルフマネージドコントローラーを廃止できます。

既存のクラスターで EKS Auto Mode を有効にする

このトピックでは、既存の Amazon EKS クラスターで Amazon EKS Auto Mode を有効にする方法について説明します。既存のクラスターで Auto Mode を有効にするには、IAM アクセス許可を更新し、コア EKS Auto Mode 設定を構成する必要があります。有効にすると、既存のコンピューティン

グワークロードの移行を開始して、Auto Mode のシンプルなオペレーションと自動化されたインフラストラクチャ管理を活用できます。

Important

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

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

AWS Management Console

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

Note

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

クラスター IAM ロールを更新する

1. AWS Management Consoleでクラスターの概要ページを開きます。
2. [クラスター IAM ロール ARN] で、[IAM で表示] を選択します。
3. [アクセス許可を追加] ドロップダウンから [ポリシーをアタッチ] を選択します。
4. [検索] ボックスより、次のポリシーを検索して選択します。
 - AmazonEKSCoordinatePolicy
 - AmazonEKSBloKStoragePolicy
 - AmazonEKSLoadBalancingPolicy
 - AmazonEKSNetworkingPolicy

- AmazonEKSClusterPolicy
5. [アクセス許可を追加] を選択します。
 6. [信頼関係] タブで、[信頼ポリシーを編集] を選択します。
 7. 次のクラスター IAM ロール信頼ポリシーを挿入し、[ポリシーを更新] を選択します。

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

EKS Auto Mode を有効にする

1. AWS Management Consoleでクラスターの概要ページを開きます。
2. [EKS Auto Mode] で [管理] を選択します
3. EKS Auto Mode をオンに切り替えます。
4. [EKS ノードプール] ドロップダウンから、作成するデフォルトのノードプールを選択します。
 - EKS Auto Mode のノードプールの詳細を確認してください。詳細については、「[the section called “ノードプールの作成”](#)」を参照してください。
5. この AWS アカウントで EKS Auto Mode Node IAM ロールを以前に作成している場合は、[ノード IAM ロール] ドロップダウンでそのロールを選択します。以前にこのロールを作成したことがない場合は、[推奨ロールを作成] を選択し、手順に従います。

AWS CLI

前提条件

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

手順

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

Note

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

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

必要なアドオンバージョン

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

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

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

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

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

次のステップ

- ノードグループワークロードの管理を移行するには、「[the section called “MNG から移行する”](#)」を参照してください。
- セルフマネージド Karpenter から移行するには、「[the section called “Karpenter からの移行”](#)」を参照してください。

kubectl を使用して Karpenter から EKS 自動モードIに移行する

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

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

前提条件

移行を開始する前に、以下のことを確認してください：

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

このトピックではKarpenter と ノードプールs に精通していることを前提としています。詳細については[Karpenter ドキュメント](#)を参照してください。

ステップ 1: クラスターで EKS 自動モードI を有効にする

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

Note

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

ステップ 2: テイントが適用された EKS 自動モードノードプールを作成する

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

テイントが適用されたノードプールの例:

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

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

ステップ 3: 移行するワークロードを更新する

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

```
apiVersion: apps/v1
kind: Deployment
spec:
  template:
    spec:
```

```
tolerations:
- key: "eks-auto-mode"
  effect: "NoSchedule"
nodeSelector:
  eks.amazonaws.com/compute-type: auto
```

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

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

ステップ 4: ワークロードを段階的に移行する

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

ステップ 5: 元の Karpenter ノードプール を削除する

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

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

ステップ 6: EKS 自動モードI ノードプール からテイントを削除する (任意)

EKS 自動モードI を新しいワークロードのデフォルトにする場合は EKS 自動モードI ノードプール からテイントを削除できます：

```
apiVersion: karpenter.sh/v1
kind: NodePool
metadata:
  name: eks-auto-mode
spec:
  template:
    spec:
      nodeClassRef:
        group: eks.amazonaws.com
        kind: NodeClass
        name: default
      # Remove the taints section
```

ステップ 7: ワークロードからノードセクターを削除する (任意)

EKS 自動モードノードプール からティントを削除した場合はEKS 自動モードがデフォルトになったため、必要に応じてワークロードからノードセクターを削除できます：

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

ステップ 8: クラスターから Karpenter をアンインストールする

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

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

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

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

前提条件

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

手順

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

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

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

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

EKS Fargate から EKS Auto Mode に移行する

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

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

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

AWS Fargate を使用する Amazon EKS は、EKS を実行するお客様に引き続きオプションとしてご利用いただけますが、今後のアプローチとしては Amazon EKS Auto Mode がお勧めです。EKS Auto Mode は、Kubernetes に完全に準拠しており、Fargate では対応していない Istio などのアップストリーム Kubernetes プリミティブとプラットフォームツールをすべてサポートしています。EKS Auto Mode は、GPU インスタンスやスポットインスタンスを含め、すべての EC2 ランタイム購入オプションにも完全に対応しているため、EC2 の割引の交渉といった仕組みを活用して節約を行います。Fargate で EKS を使用する場合は、そのような機能が用意されていません。

さらに、EKS Auto Mode なら、Fargate と同じ分離モデルを実現できます。これを行うには、標準の Kubernetes スケジューリング機能を使用して、各 EC2 インスタンスで 1 つのアプリケーションコンテナを実行します。Amazon EKS Auto Mode を導入すると、Kubernetes を AWS で実行するメリットを最大限に活用できます。AWS は、Kubernetes に完全準拠したプラットフォームであ

り、EC2 と購入の広範なオプションをすべて利用できます。また、Fargate が実現するインフラストラクチャ管理の利便性や抽象化も維持されています。

前提条件

移行を開始する前に、以下を確認します。

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

ステップ 1: Fargate クラスターを確認する

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

```
kubectl get node
```

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

2. 実行中のポッドを確認します。

```
kubectl get pod -A
```

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

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

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
  labels:
    app: nginx
spec:
  replicas: 3
```

```

selector:
  matchLabels:
    app: nginx
template:
  metadata:
    labels:
      app: nginx
    annotations:
      eks.amazonaws.com/compute-type: fargate
  spec:
    containers:
    - name: nginx
      image: nginx
      ports:
      - containerPort: 80

```

4. デプロイを適用します：

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

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

5. ポッドとデプロイを確認します。

```
kubectl get pod,deploy
```

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

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

6. ノードを確認します。

```
kubectl get node -owide
```

NAME	STATUS	ROLES	AGE	VERSION
INTERNAL-IP	EXTERNAL-IP	OS-IMAGE	KERNEL-VERSION	
CONTAINER-RUNTIME				

```
fargate-ip-192-168-111-43.ec2.internal Ready <none> 31s v1.30.8-eks-2d5f260
192.168.111.43 <none> Amazon Linux 2 5.10.234-225.910.amzn2.x86_64
containerd://1.7.25
fargate-ip-192-168-117-130.ec2.internal Ready <none> 36s v1.30.8-eks-2d5f260
192.168.117.130 <none> Amazon Linux 2 5.10.234-225.910.amzn2.x86_64
containerd://1.7.25
fargate-ip-192-168-74-140.ec2.internal Ready <none> 36s v1.30.8-eks-2d5f260
192.168.74.140 <none> Amazon Linux 2 5.10.234-225.910.amzn2.x86_64
containerd://1.7.25
```

ステップ 2: クラスターで EKS Auto Mode を有効にする

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

```
kubectl get nodepool
```

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

ステップ 3: 移行するワークロードを更新する

EKS 自動モード I に移行するワークロードを特定して更新します。

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

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

```
apiVersion: apps/v1
kind: Deployment
metadata:
```

```

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

```

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

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

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

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

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

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

```
kubectl get node -owide
```

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

ステップ 4: ワークロードを段階的に移行する

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

ステップ 5: 元の fargate プロファイルを削除する

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

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

ステップ 6: CoreDNS をスケールダウンする

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

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

EKS 自動モードI クラスターでワークロードを実行する

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

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

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

[the section called “拡張ワークロードのデプロイ”](#)

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

[the section called “ロードバランサーをデプロイする”](#)

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

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

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

[the section called “デプロイを制御する”](#)

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

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

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

前提条件

- コンピューティング機能が有効になっている Amazon EKS Auto Mode クラスター。クラスターの名前と AWS リージョンをメモします。

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

ステップ 1: 既存のコンピューティングリソースを確認する (任意)

まず、kubectl を使用してクラスターのノードプールを一覧表示します。

```
kubectl get nodepools
```

サンプル出力:

```
general-purpose
```

このチュートリアルでは、general-purpose ノードプールを使用するように設定されたワークロードをデプロイします。このノードプールは EKS Auto Mode に組み込まれ、マイクロサービスやウェブアプリケーションなどの一般的なワークロードに適したデフォルトが含まれます。独自のノードプールを作成できます。詳細については、「[the section called “ノードプールの作成”](#)」を参照してください。

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

```
kubectl get nodes
```

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

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

ステップ 2: サンプルアプリケーションをクラスターにデプロイする

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

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

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

デプロイをクラスターに適用します。

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

ステップ 3: Kubernetes イベントを監視する

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


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

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

```
kubectl get nodes
```

ステップ 4: AWS コンソールでノードとインスタンスを表示する

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

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

ステップ 5: デプロイを削除する

kubectl を使用してサンプルデプロイを削除します

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

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

デフォルト設定では、EKS Auto Mode は空のノードを 30 秒間検出し、そのノードを終了します。

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

サンプルロードバランサーワークロードを EKS 自動モードI にデプロイする

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

前提条件

- EKS 自動モードI クラスター
- クラスターとやり取りするように設定された kubectl

- ALB リソースを作成するための適切な IAM アクセス許可

ステップ 1: 名前空間を作成する

まず、2048 ゲームアプリケーション専用の名前空間を作成します。

01-namespace.yaml という名前のファイルを作成します：

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

この名前空間設定を適用します：

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

ステップ 2: アプリケーションをデプロイする

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

02-deployment.yaml という名前のファイルを作成します：

```
apiVersion: apps/v1
kind: Deployment
metadata:
  namespace: game-2048
  name: deployment-2048
spec:
  selector:
    matchLabels:
      app.kubernetes.io/name: app-2048
  replicas: 5
  template:
    metadata:
      labels:
        app.kubernetes.io/name: app-2048
    spec:
      containers:
        - image: public.ecr.aws/l6m2t8p7/docker-2048:latest
          imagePullPolicy: Always
          name: app-2048
```

```
ports:
  - containerPort: 80
resources:
  requests:
    cpu: "0.5"
```

Note

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

主要コンポーネント:

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

デプロイを適用します :

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

ステップ 3: サービスを作成する

サービスによりデプロイがクラスターネットワークに公開されます。

`03-service.yaml` という名前のファイルを作成します :

```
apiVersion: v1
kind: Service
metadata:
  namespace: game-2048
  name: service-2048
spec:
  ports:
```

```
- port: 80
  targetPort: 80
  protocol: TCP
type: NodePort
selector:
  app.kubernetes.io/name: app-2048
```

主要コンポーネント:

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

サービスを適用します :

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

ステップ 4: ロードバランシングを設定する

アプリケーションがインターネットに公開されるようにイングレスを設定します。

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

```
apiVersion: networking.k8s.io/v1
kind: IngressClass
metadata:
  labels:
    app.kubernetes.io/name: LoadBalancerController
  name: alb
spec:
  controller: eks.amazonaws.com/alb
```

Note

EKS Auto Mode では、パブリックサブネットとプライベートサブネットを識別するためにサブネットタグが必要です。

eksctl を使用してクラスターを作成した場合は、これらのタグは既にあります。

[「the section called “サブネットにタグを付ける”」](#)ではその方法を説明しています。

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

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

主要コンポーネント:

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

イングレス設定を適用します :

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

ステップ 5: デプロイを検証する

1. すべてのポッドが実行されていることを確認します :

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

2. サービスが作成されていることを確認します：

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

3. ALB エンドポイントを取得します：

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

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

ステップ 6: ゲームにアクセスする

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

ステップ 7: クリーンアップ

このチュートリアルで作成したリソースをすべて削除するには：

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

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

シーンの背後で起きていること

1. デプロイでは2048 ゲームを実行するポッドが 5 つ作成されます
2. このサービスではこれらのポッドに安定したネットワークアクセスを提供します
3. EKS 自動モード:
 - AWS で Application Load Balancer を作成します
 - ポッドのターゲットグループを設定します
 - サービスにトラフィックを誘導するルーティングルールを設定します

トラブルシューティング

ゲームがロードされない場合:

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

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

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

前提条件

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

ステップ 1: 環境を設定する

1. 環境変数を設定する:

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

2. kubeconfig を更新します:

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

ステップ 2: ストレージクラスを作成する

StorageClassは、EBS ボリュームを EKS Auto Mode でプロビジョニングする方法を定義します。

EKS Auto Mode では、お客様用の StorageClass は作成されません。EKS Auto Mode のストレージ機能を使用するには、`ebs.csi.eks.amazonaws.com` を参照する StorageClass を作成する必要があります。

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

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

2. StorageClass を適用します。

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

主要コンポーネント：

- `provisioner: ebs.csi.eks.amazonaws.com` - EKS Auto Mode を使用する
- `volumeBindingMode: WaitForFirstConsumer` - ポッドが必要とするまでボリュームの作成を遅らせる
- `type: gp3` - EBS ボリュームタイプを指定する
- `encrypted: "true"` - EBS はデフォルトの `aws/ebs` キーを使用して、このクラスで作成されたボリュームを暗号化します。これはオプションですが推奨されます。
- `storageclass.kubernetes.io/is-default-class: "true"` - 永続ボリュームのクレームで別のボリュームクラスを指定しない限り、Kubernetes はデフォルトでこのストレージクラスを使用します。別のストレージコントローラーから移行する場合は、この値を設定するときに注意してください。(オプション)

ステップ3: 永続的なボリューム要求を作成する

PVC はStorageClassからのストレージをリクエストします。

1. pvc.yaml という名前のファイルを作成します：

```
apiVersion: v1
```



```
kind: PersistentVolumeClaim
metadata:
  name: auto-ebs-claim
spec:
  accessModes:
    - ReadWriteOnce
  storageClassName: auto-ebs-sc
  resources:
    requests:
      storage: 8Gi
```

2. SCP を適用します:

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

主要コンポーネント:

- accessModes: ReadWriteOnce - ボリュームは一度に 1 つのノードからマウントすることができます
- storage: 8Gi - 8 GiB ボリュームをリクエストします
- storageClassName: auto-ebs-sc - 作成したStorageClassを参照します

ステップ 4: アプリケーションのデプロイ

Deployment は、永続的ボリュームにタイムスタンプを書き込むコンテナを実行します。

1. deployment.yaml という名前のファイルを作成します :

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: inflate-stateful
spec:
  replicas: 1
  selector:
    matchLabels:
      app: inflate-stateful
  template:
    metadata:
      labels:
```

```
    app: inflate-stateful
  spec:
    terminationGracePeriodSeconds: 0
    nodeSelector:
      eks.amazonaws.com/compute-type: auto
    containers:
      - name: bash
        image: public.ecr.aws/docker/library/bash:4.4
        command: ["/usr/local/bin/bash"]
        args: ["-c", "while true; do echo $(date -u) >> /data/out.txt; sleep 60;
done"]
    resources:
      requests:
        cpu: "1"
      volumeMounts:
        - name: persistent-storage
          mountPath: /data
    volumes:
      - name: persistent-storage
        persistentVolumeClaim:
          claimName: auto-ebs-claim
```

2. デプロイを適用します。

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

主要コンポーネント:

- ファイルにタイムスタンプを書き込むシンプルな bash コンテナ
- PVC を /data にマウントします
- 1 CPU コアをリクエストします
- EKS マネージドノードのノードセクタを使用します

ステップ 5: セットアップを検証する

1. ポッドが稼働していることを確認します:

```
kubectl get pods -l app=inflate-stateful
```

2. PVC がバインドされていることを確認します:

```
kubectl get pvc auto-ebs-claim
```

3. EBS ボリュームを確認します:

```
# Get the PV name
PV_NAME=$(kubectl get pvc auto-ebs-claim -o jsonpath='{.spec.volumeName}')
# Describe the EBS volume
aws ec2 describe-volumes \
  --filters Name=tag:CSIVolumeName,Values=${PV_NAME}
```

4. データが書き込まれていることを確認します:

```
kubectl exec "$(kubectl get pods -l app=inflate-stateful \
  -o=jsonpath='{.items[0].metadata.name}')" -- \
  cat /data/out.txt
```

ステップ 6: クリーンアップ

次のコマンドを実行して、このチュートリアルで作成したすべてのリソースを削除します。

```
# Delete all resources in one command
kubectl delete deployment/inflate-stateful pvc/auto-ebs-claim storageclass/auto-ebs-sc
```

シーンの背後で起きていること

1. PVC は StorageClass からのストレージをリクエストします
2. ポッドがスケジュールされている場合:
 - a. EKS Auto Mode が EBS ボリュームをプロビジョニングします
 - b. PersistentVolume を作成します
 - c. ボリュームをノードにアタッチします。
3. ポッドはボリュームをマウントし、タイムスタンプの書き込みを開始します

Snapshot Controller

EKS Auto Mode は、スナップショットコントローラーとも呼ばれる Kubernetes CSI Snapshotter と互換性があります。ただし、EKS Auto Mode にはスナップショットコントローラーがありません。

スナップショットコントローラーのインストールと設定はお客様の責任となります。詳細については、「[the section called “CSI スナップショットコントローラー”](#)」を参照してください。

EKS Auto Mode のストレージ機能を参照する以下の VolumeSnapshotClass を確認してください。

```
apiVersion: snapshot.storage.k8s.io/v1
kind: VolumeSnapshotClass
metadata:
  name: auto-ebs-vsclass
driver: ebs.csi.eks.amazonaws.com
deletionPolicy: Delete
```

[Kubernetes CSI Snapshotter の詳細をご覧ください。](#)

EKS 自動モード設定を構成する

この章ではAmazon エラスティックKubernetesサービス (EKS) 自動モード クラスターの特定の機能を設定する方法について説明します。EKS 自動モード はほとんどのインフラストラクチャコンポーネントを自動的に管理しますが、ワークロードの要件を満たすように特定の機能をカスタマイズできます。

このトピックで説明されている設定オプションを使用して、自動インフラストラクチャ管理の利点を維持しながら、ネットワーク設定、コンピューティングリソース、ロードバランシングの動作を変更できます。設定を変更する前に、以下のセクションで使用可能なオプションを確認して、ニーズに最適なアプローチを決定してください。

どの機能を設定したいですか？	設定オプション
ノードネットワークとストレージ	the section called “ノードクラスの作成”
- パブリックサブネットとプライベートサブネットにまたがるノードの配置を設定する - ノードアクセスコントロールのカスタムセキュリティグループを定義する - ネットワークアドレス変換 (SNAT) ポリシーをカスタマイズする	

どの機能を設定したいですか？	設定オプション
• Kubernetes ネットワークポリシー (詳細なネットワークポリシーのログ記録とモニタリング) を有効にする • エフェメラルストレージパラメータ (サイズ、IOPS、スループット) を設定する • カスタム KMS キーを使用して暗号化されたエフェメラルストレージを設定する • ポッドトラフィックをノードから別々のサブネットに分離する	
ノードコンピューティングリソース • 特定の EC2 インスタンスタイプとファミリーを選択する • CPU アーキテクチャを定義する (x86_64、ARM64) • キャパシティタイプを設定する (オンデマンド、スポット) • アベイラビリティーゾーンを指定する • ノードのテイントとラベルを設定する • 最小ノード数と最大ノード数を設定する	the section called “ノードプールの作成”

どの機能を設定したいですか？	設定オプション
Application Load Balancer の設定 • 内部ロードバランサーまたはインターネット向けロードバランサーをデプロイする • クロスゾーン負荷分散の設定 • アイドルタイムアウト期間を設定する • HTTP/2 および WebSocket サポートを有効にする • ヘルスチェックパラメータを設定する • TLS 証明書設定を指定する • ターゲットグループの属性を定義する • IP アドレスタイプを設定する (IPv4、デュアルスタック)	the section called “ingress class の作成”
Network Load Balancer の設定 • 直接ポッド IP ルーティングを設定する • クロスゾーン負荷分散の有効化 • 接続のアイドルタイムアウトを設定する • ヘルスチェックパラメータを設定する • サブネットの配置を指定する • IP アドレスタイプを設定する (IPv4、デュアルスタック) • クライアントソース IP の保存を設定する • ターゲットグループの属性を定義する	the section called “サービスの作成”

どの機能を設定したいですか？	設定オプション
ストレージクラスの設定	the section called “StorageClass を作成する”
• EBS ボリュームタイプ (gp3、io1、io2 など) を定義する • ボリューム暗号化と KMS キーの使用を設定する • IOPS とスループットパラメータを設定する • デフォルトのストレージクラスとして設定する • プロビジョニングされたボリュームのカスタムタグを定義する	

Amazon EKS のノードクラスを作成する

Amazon EKS ノードクラスは、EKS Auto Mode マネージドノードの設定をきめ細かく制御できるテンプレートです。ノードクラスは、ネットワーク設定、ストレージ設定、リソースタグ付けなど、EKS クラスター内のノードのグループに適用されるインフラストラクチャレベルの設定を定義します。このトピックでは、特定の運用要件を満たすようにノードクラスを作成および設定する方法について説明します。

EKS Auto Mode がデフォルト設定を超えて EC2 インスタンスをプロビジョニングおよび設定する方法をカスタマイズする必要がある場合は、ノードクラスを作成すると、重要なインフラストラクチャパラメータを正確に制御できます。例えば、セキュリティを強化するためにプライベートサブネットの配置を指定したり、パフォーマンス重視のワークロード用にインスタンスエフェメラルストレージを設定したり、コスト配分のためにカスタムタグ付けを適用したりできます。

ノードクラスを作成する

NodeClass を作成するには、次の手順に従います。

1. 自分のノードクラス設定で YAML ファイル (nodeclass.yaml など) を作成します
2. kubectl を使用してこの設定をクラスターに適用します
3. ノードプール設定でノードクラスを参照します。詳細については、「[the section called “ノードプールの作成”](#)」を参照してください。

kubectl がインストールおよび設定されている必要があります。詳細については、「[セットアップする](#)」を参照してください。

基本的なノードクラスの例

ノードクラスの例を次に示します。

```
apiVersion: eks.amazonaws.com/v1
kind: NodeClass
metadata:
  name: private-compute
spec:
  ephemeralStorage:
    size: "160Gi"
```

この NodeClass により、ノード上のエフェメラルストレージの量が増加します。

次を使用してこの設定を適用します。

```
kubectl apply -f nodeclass.yaml
```

次に、ノードプール設定でノードクラスを参照します。詳細については、「[the section called “ノードプールの作成”](#)」を参照してください。

ノードクラスのアクセスエントリを作成する

カスタムノードクラスを作成する場合は、ノードがクラスターに参加できるように EKS アクセスエントリを作成する必要があります。組み込みのノードクラスとノードプールを使用すると、EKS によってアクセスエントリが自動的に作成されます。

アクセスエントリがどのように機能するかについては、「[the section called “許可を付与する”](#)」を参照してください。

EKS Auto Mode ノードクラスのアクセスエントリを作成するときは、EC2 のアクセスエントリタイプを使用する必要があります。

CLI でアクセスエントリを作成する

EC2 ノードのアクセスエントリを作成して EKS Auto Node ポリシーを関連付けるには:

次の CLI コマンドを、使用するクラスター名とノードロール ARN で更新します。ノードロール ARN は、ノードクラス YAML で指定されます。


```
# Create the access entry for EC2 nodes
aws eks create-access-entry \
  --cluster-name <cluster-name> \
  --principal-arn <node-role-arn> \
  --type EC2

# Associate the auto node policy
aws eks associate-access-policy \
  --cluster-name <cluster-name> \
  --principal-arn <node-role-arn> \
  --policy-arn arn:aws:eks::aws:cluster-access-policy/AmazonEKSAutoNodePolicy \
  --access-scope type=cluster
```

CloudFormation でアクセスエントリを作成する

EC2 ノードのアクセスエントリを作成して EKS Auto Node ポリシーを関連付けるには:

次の CloudFormation を、使用するクラスター名とノードロール ARN で更新します。ノードロール ARN は、ノードクラス YAML で指定されます。

```
EKSAutoNodeRoleAccessEntry:
  Type: AWS::EKS::AccessEntry
  Properties:
    ClusterName: <cluster-name>
    PrincipalArn: <node-role-arn>
    Type: "EC2"
    AccessPolicies:
      - AccessScope:
          Type: cluster
          PolicyArn: arn:aws:eks::aws:cluster-access-policy/AmazonEKSAutoNodePolicy
    DependsOn: [ <cluster-name> ] # previously defined in CloudFormation
```

CloudFormation スタックのデプロイの詳細については、「[Getting started with CloudFormation](#)」を参照してください

ノードクラスの指定

```
apiVersion: eks.amazonaws.com/v1
kind: NodeClass
metadata:
  name: my-node-class
spec:
```

```
# Required fields
role: MyNodeRole # IAM role for EC2 instances

subnetSelectorTerms:
  - tags:
      Name: "private-subnet"
      kubernetes.io/role/internal-elb: "1"
  # Alternative using direct subnet ID
  # - id: "subnet-0123456789abcdef0"

securityGroupSelectorTerms:
  - tags:
      Name: "eks-cluster-sg"
  # Alternative approaches:
  # - id: "sg-0123456789abcdef0"
  # - name: "eks-cluster-security-group"

# Optional: Pod subnet selector for advanced networking
podSubnetSelectorTerms:
  - tags:
      Name: "pod-subnet"
      kubernetes.io/role/pod: "1"
  # Alternative using direct subnet ID
  # - id: "subnet-0987654321fedcba0"

# Optional fields
snatPolicy: Random # or Disabled

networkPolicy: DefaultAllow # or DefaultDeny
networkPolicyEventLogs: Disabled # or Enabled

ephemeralStorage:
  size: "80Gi" # Range: 1-59000Gi or 1-64000G or 1-58Ti or 1-64T
  iops: 3000 # Range: 3000-16000
  throughput: 125 # Range: 125-1000
  # Optional KMS key for encryption
  kmsKeyID: "arn:aws:kms:region:account:key/key-id"
  # Accepted formats:
  # KMS Key ID
  # KMS Key ARN
  # Key Alias Name
  # Key Alias ARN

# Optional: Forward proxy, commonly requires certificateBundles as well
```

```
#for EC2, see https://repost.aws/knowledge-center/eks-http-proxy-containerd-
automation
advancedNetworking:
  httpsProxy: http://192.0.2.4:3128 #commonly port 3128 (Squid) or 8080 (NGINX) #Max
255 characters
  #httpsProxy: http://[2001:db8::4]:3128 # IPv6 address with port, use []
  noProxy: #Max 50 entries
    - localhost #Max 255 characters each
    - 127.0.0.1
    #- ::1 # IPv6 localhost
    #- 0:0:0:0:0:0:0:1 # IPv6 localhost
    - 169.254.169.254 # EC2 Instance Metadata Service
    #- [fd00:ec2::254] # IPv6 EC2 Instance Metadata Service
    # Domains to exclude, put all VPC endpoints here
    - .internal
    - .eks.amazonaws.com

# Optional: Custom certificate bundles.
certificateBundles:
  - name: "custom-cert"
    data: "base64-encoded-cert-data"

# Optional: Additional EC2 tags (with restrictions)
tags:
  Environment: "production"
  Team: "platform"
  # Note: Cannot use restricted tags like:
  # - kubernetes.io/cluster/*
  # - karpenter.sh/provisioner-name
  # - karpenter.sh/nodepool
  # - karpenter.sh/nodeclaim
  # - karpenter.sh/managed-by
  # - eks.amazonaws.com/nodeclass
```

考慮事項

- ノード IAM ロールの置換 - NodeClass に関連付けられているノード IAM ロールを変更する場合は、新しいアクセスエントリを作成する必要があります。EKS は、クラスターの作成時にノード IAM ロールのアクセスエントリを自動的に作成します。ノード IAM ロールには AmazonEKSAutoNodePolicy EKS アクセスポリシーが必要です。詳細については、「[the section called “許可を付与する”](#)」を参照してください。

- 最大ポッド密度 - EKS では、ノード上のポッドの最大数が 110 に制限されます。この制限は、既存の最大ポッド計算の後に適用されます。詳細については、「[the section called “Amazon EC2 インスタンスタイプ”](#)」を参照してください。
- タグ - Kubernetes から EC2 にタグを伝播させる場合は、追加の IAM アクセス許可を設定する必要があります。詳細については、「[the section called “ID とアクセス”](#)」を参照してください。
- デフォルトのノードクラス - カスタムノードクラスに default という名前を付けないでください。これは、EKS Auto Mode には、少なくとも 1 つの組み込み NodePool を有効にすると自動的にプロビジョニングされる default と呼ばれる NodeClass が含まれるためです。組み込み NodePools を有効にする方法については、「[the section called “組み込みノードプールを確認する”](#)」を参照してください。
- 複数のサブネットを使用した **subnetSelectorTerms** 動作 - subnetSelectorTerms 条件に一致するサブネットまたは ID で指定したサブネットが複数ある場合、EKS Auto Mode はサブネット全体に分散されたノードを作成します。
- サブネットが異なるアベイラビリティゾーン (AZ) にある場合、[ポッドトポロジスプレッド制約](#)や [Topology Aware Routing](#) などの Kubernetes 機能を使用して、ポッドとトラフィックをそれぞれゾーン間で分散できます。
- 同じ AZ 内に subnetSelectorTerms に一致するサブネットが複数ある場合、EKS Auto Mode はその AZ のサブネット全体に分散された各ノードにポッドを作成します。EKS Auto Mode は、同じ AZ 内の他のサブネットの各ノードにセカンダリネットワークインターフェイスを作成します。各サブネットで使用可能な IP アドレスの数に基づいて選択し、サブネットをより効率的に使用します。ただし、EKS Auto Mode が各ポッドに使用するサブネットを指定することはできません。特定のサブネットでポッドを実行する必要がある場合は、代わりに [the section called “ポッドのサブネット選択”](#) を使用します。

ポッドのサブネット選択

podSubnetSelectorTerms と podSecurityGroupSelectorTerms フィールドは、ポッドをノードとは異なるサブネットで実行できるようにすることで、高度なネットワーク設定を可能にします。この分離により、ネットワークトラフィックのルーティングとセキュリティポリシーの制御が強化されます。podSecurityGroupSelectorTerms には podSubnetSelectorTerms が必要であることに注意してください。

ユースケース

次の場合に podSubnetSelectorTerms を使用します:

- インフラストラクチャトラフィック (ノード間の通信) とアプリケーショントラフィック (ポッド間の通信) を分離する
- ポッドサブネットとは異なるネットワーク設定をノードサブネットに適用する。
- ノードとポッドに異なるセキュリティポリシーまたはルーティングルールを実装する。
- ポッドトラフィックに影響を与えずに、ノードトラフィック専用のリバースプロキシまたはネットワークフィルタリングを設定する。advancedNetworking と certificateBundles を使用して、リバースプロキシと、プロキシの自己署名証明書またはプライベート証明書を定義する。

設定例

```
apiVersion: eks.amazonaws.com/v1
kind: NodeClass
metadata:
  name: advanced-networking
spec:
  role: MyNodeRole

  # Subnets for EC2 instances (nodes)
  subnetSelectorTerms:
    - tags:
        Name: "node-subnet"
        kubernetes.io/role/internal-elb: "1"

  securityGroupSelectorTerms:
    - tags:
        Name: "eks-cluster-sg"

  # Separate subnets for pods
  podSubnetSelectorTerms:
    - tags:
        Name: "pod-subnet"
        kubernetes.io/role/pod: "1"

  podSecurityGroupSelectorTerms:
    - tags:
        Name: "eks-pod-sg"
```

ポッドのサブネットセレクトタに関する考慮事項

- ポッド密度の低下: podSubnetSelectorTerms を使用する場合、ノードのプライマリネットワークインターフェイスがノードサブネットにあり、ポッドサブネット内のポッドに使用できないため、各ノードで実行できるポッドが少なくなります。
- サブネットセレクトタの制限: 標準 subnetSelectorTerms および securityGroupSelectorTerms 設定は、ポッドサブネットの選択には適用されません。
- ネットワーク計画: ワークロード要件をサポートするために、ノードサブネットとポッドサブネットの両方に十分な IP アドレススペースを確保します。
- ルーティング設定: ポッドサブネットのルートテーブルとネットワークアクセスコントロールリスト (ACL) が、ノードとポッドサブネット間の通信用に正しく設定されていることを確認します。

EKS 自動モードI 用のノードプールを作成する

Amazon EKS ノードプールにより、Kubernetes クラスター内のコンピューティングリソースを柔軟に管理できます。このトピックでは、クラスターのスケーリングとリソース使用率を最適化するノードプロビジョニングツールである Karpenter を使用してノードプールを作成および設定する方法を示します。Karpenter の ノードプール リソースを使用すると、インスタンスタイプ、アベイラビリティゾーン、アーキテクチャ、キャパシティタイプなど、コンピューティングリソースの特定の要件を定義できます。

組み込みの system および general-purpose ノードプールは変更できません。これらは、有効または無効にすることのみができます。詳細については、「[the section called “組み込みノードプールを確認する”](#)」を参照してください。

ノードプールの指定により、サポートされているさまざまなラベルと要件を通じて、EKS クラスターのコンピューティングリソースをきめ細かく制御できます。これには EC2 インスタンスカテゴリ、CPU 設定、アベイラビリティゾーン、アーキテクチャ (ARM64/AMD64)、キャパシティタイプ (スポットまたはオンデマンド) を指定するためのオプションが含まれます。CPU とメモリの使用量のリソース制限を設定して、クラスターが必要な運用境界内に収まるようにすることもできます。

EKS 自動モードI は広く知られている Kubernetes ラベルを活用して、ノードの特性を識別する一貫性のある標準化された方法を提供します。アベイラビリティゾーンの topology.kubernetes.io/zone や CPU アーキテクチャの kubernetes.io/arch といったこのラベルは確立された Kubernetes 規則に従います。さらに、EKS 固有のラベル (プレフィックスが eks.amazonaws.com/) により、インスタンスタイプ、CPU メーカー、GPU 機能、ネットワーク仕様などの AWS 固有の属性でこの機能が拡張されます。この標準化されたラベル付けシステムによ

り、既存の Kubernetes ツールとシームレスに統合できると同時に、緊密な AWS インフラストラクチャ統合が提供されます。

ノードプール を作成する

Amazon EKS クラスターの ノードプール を作成するには次のステップに従います：

1. 必要な NodePool 設定で `nodepool.yaml` という名前の YAML ファイルを作成します。以下のサンプル設定を使用できます。
2. ノードプール をクラスターに適用します：

```
kubectl apply -f nodepool.yaml
```

3. ノードプール が正常に作成されたことを確認します：

```
kubectl get nodepools
```

4. (任意 ノードプール のステータスをモニタリングします：

```
kubectl describe nodepool default
```

ノードプール がクラスターに存在する有効な ノードクラス を参照していることを確認します。ノードクラス はコンピューティングリソースの AWS 固有の設定を定義します。詳細については「[the section called “ノードクラスの作成”](#)」を参照してください。

サンプル ノードプール

```
apiVersion: karpenter.sh/v1
kind: NodePool
metadata:
  name: my-node-pool
spec:
  template:
    metadata:
      labels:
        billing-team: my-team
    spec:
      nodeClassRef:
        group: eks.amazonaws.com
        kind: NodeClass
```

```

name: default

requirements:
  - key: "eks.amazonaws.com/instance-category"
    operator: In
    values: ["c", "m", "r"]
  - key: "eks.amazonaws.com/instance-cpu"
    operator: In
    values: ["4", "8", "16", "32"]
  - key: "topology.kubernetes.io/zone"
    operator: In
    values: ["us-west-2a", "us-west-2b"]
  - key: "kubernetes.io/arch"
    operator: In
    values: ["arm64", "amd64"]

limits:
  cpu: "1000"
  memory: 1000Gi

```

EKS Auto Mode でサポートされているラベル

EKS 自動モードは以下の広く知られてるラベルをサポートしています。

ラベル	例	説明
topology.kubernetes.io/zone	us-east-2a	AWS リージョン
node.kubernetes.io/インスタス 型	g4dn.8xlarge	AWS インスタスタイプ
kubernetes.io/arch	amd64	アーキテクチャはインスタンスの GOARCH 値 によって定義されます
karpenter.sh/capacity-type	spot	キャパシティータイプには spot、on-demand などがあります
eks.amazonaws.com/instance-hypervisor	nitro	特定のハイパーバイザーを使用するインスタスタイプ

ラベル	例	説明
eks.amazonaws.com/compute-type	自動	EKS 自動モードI マネージドノードを識別します
eks.amazonaws.com/instance-encryption-in-transit-supported	真	転送中の暗号化をサポートする (またはサポートしない) インスタンスタイプ
eks.amazonaws.com/instance-category	g	同じカテゴリのインスタンスタイプ、通常は生成番号の前の文字列
eks.amazonaws.com/instance-generation	4	インスタンスカテゴリ内のインスタンスタイプ生成番号
eks.amazonaws.com/instance-family	g4dn	プロパティが類似していてリソース量が異なるインスタンスタイプ
eks.amazonaws.com/instance-size	8xlarge	リソース量が類似していてプロパティが異なるインスタンスタイプ
eks.amazonaws.com/instance-cpu	32	インスタンスにおける CPU の数
eks.amazonaws.com/instance-cpu-manufacturer	aws	CPU メーカーの名前
eks.amazonaws.com/instance-memory	131072	インスタンスのメモリのメビバイト数
eks.amazonaws.com/instance-ebs-bandwidth	9500	インスタンスで利用できる EBS の 最大メガビット 数
eks.amazonaws.com/instance-network-bandwidth	131072	インスタンスで使用可能な ベースラインメガビット 数
eks.amazonaws.com/instance-gpu-name	t4	インスタンス上の GPU の名前 (使用可能な場合)
eks.amazonaws.com/instance-gpu-manufacturer	nvidia	GPU メーカーの名前

ラベル	例	説明
eks.amazonaws.com/instance-gpu-count	1	インスタンスにおける GPU の数
eks.amazonaws.com/instance-gpu-memory	16384	GPU のメモリのメビバイト数
eks.amazonaws.com/instance-local-nvme	900	インスタンス上のローカル nvme ストレージのギビバイト数

Note

EKS Auto Mode は特定のインスタンスのみをサポートし、最小サイズ要件があります。詳細については、「[the section called “EKS Auto Mode でサポートされているインスタンスリファレンス”](#)」を参照してください。

EKS Auto Mode でサポートされていないラベル

EKS 自動モードは次のラベルをサポートしていません。

- EKS 自動モードは Linux のみをサポートしています
 - node.kubernetes.io/windows-build
 - kubernetes.io/os

組み込みノードプールを無効にする

カスタムノードプールを作成する場合は、組み込みノードプールを無効にすることができます。詳細については、「[the section called “組み込みノードプールを確認する”](#)」を参照してください。

組み込みノードプールがないクラスター

組み込みノードプールがないクラスターを作成できます。これは、組織がカスタマイズしたノードプールを作成している場合に役立ちます。

Note

組み込みノードプールなしでクラスターを作成すると、default NodeClass は自動的にプロビジョニングされません。カスタム NodeClass を作成する必要があります。詳細については、「[the section called “ノードクラスの作成”](#)」を参照してください。

概要 :

1. nodePools と nodeRoleArn の両方の値を空にして、EKS クラスターを作成します。

- eksctl autoModeConfig の例:

```
autoModeConfig:
  enabled: true
  nodePools: []
  # Do not set a nodeRoleARN
```

詳細については、[the section called “eksctl CLI”](#)を参照してください。

2. ノードロール ARN を使用してカスタムノードクラスを作成します

- 詳細については、[the section called “ノードクラスの作成”](#)を参照してください。

3. カスタムノードクラスのアクセスエントリを作成します

- 詳細については、[the section called “ノードクラスのアクセスエントリを作成する”](#)を参照してください。

4. 前述のように、カスタムノードプールを作成します。

中断

NodePool を介してノードを中断するように、EKS Auto Mode を複数の方法で設定できます。spec.disruption.consolidationPolicy、spec.disruption.consolidateAfter、または spec.template.spec.expireAfter を使用できます。NodePool の spec.disruption.budgets を使用して、EKS Auto Mode の中断をレート制限することもできます。中断される時間枠と同時ノード数を制御することもできます。この動作を設定する手順については、Karpenter ドキュメントの「[Disruption](#)」を参照してください。

ノードプールの中断は、次の目的で設定できます。

- インスタンスの使用率が低いタイミングを特定し、ワークロードを統合する。

- ノードプールの中断予算を作成して、ドリフト、空、統合が理由でのノードの終了をレート制限する。

デフォルトでは、EKS Auto Mode では以下が行われます。

- 使用率が低いインスタンスを統合します。
- 336 時間後にインスタンスを終了します。
- ノードの 10% の単一の中断予算を設定します。
- 新しい Auto Mode AMI がリリースされたときに、ドリフトを理由にノードを置き換えることを許可します。これは、およそ週に 1 回行われます。

終了猶予期間

EKS Auto NodePool で `terminationGracePeriod` を明示的に定義していない場合、関連付けた `NodeClaim` に 24 時間 (デフォルト) の終了猶予期間が自動的に適用されます。EKS Auto のカスタム NodePool 設定では `terminationGracePeriod` がデフォルトであることを確認できませんが、`NodeClaim` には、このデフォルト値が表示されます。猶予期間を NodePool で明示的に設定済みか、`NodeClaim` でデフォルト設定を行ったかに関係なく、この機能には一貫性があり、予測可能なノード終了動作がクラスター全体で維持されます。

IngressClass を作成して Application Load Balancer を設定する

EKS Auto Mode は、インターネットへのクラスターアプリケーションの公開など、ロードバランシングのルーチンタスクを自動化します。

AWS では、Application Load Balancer (ALB) を使用して HTTP および HTTPS トラフィックを処理することを推奨しています。Application Load Balancer は、リクエストの内容に基づいてリクエストをルーティングできます。Application Load Balancer の詳細については、「[What is Elastic Load Balancing?](#)」を参照してください。

EKS Auto Mode は、Application Load Balancer (ALB) の作成と設定を行います。例えば、Ingress Kubernetes オブジェクトを作成すると EKS Auto Mode によってロードバランサーが作成され、クラスターワークロードにトラフィックをルーティングするように設定されます。

概要

1. インターネットに公開するワークロードを作成します。

2. SSL/TLS や VPC サブネットに使用する証明書などの AWS 固有の設定値を指定して、IngressClassParams リソースを作成します。
3. IngressClass リソースを作成し、EKS Auto Mode がリソースのコントローラーになるように指定します。
4. HTTP パスおよびポートをクラスターワークロードに関連付ける Ingress リソースを作成します。

EKS Auto Mode は、IngressClassParams リソースで指定されたロードバランサー設定を使用して、Ingress リソースで指定されたワークロードをポイントする Application Load Balancer を作成します。

前提条件

- EKS Auto Mode が Amazon EKS クラスターで有効になっている
- kubectl がクラスターに接続するように設定されている
 - `kubectl apply -f <filename>` を使用して、以下のサンプル設定 YAML ファイルをクラスターに適用できます。

Note

EKS Auto Mode では、パブリックサブネットとプライベートサブネットを識別するためにサブネットタグが必要です。

eksctl を使用してクラスターを作成した場合は、これらのタグは既にあります。

[「the section called “サブネットにタグを付ける”」](#)ではその方法を説明しています。

ステップ 1: ワークロードを作成する

インターネットに公開するワークロードを作成します。これは、デプロイやサービスなど、HTTP トラフィックを処理する任意の Kubernetes リソースです。この例では、ポート 80 で HTTP トラフィックを処理する `service-2048` という名前の単純な HTTP サービスを使用します。

ステップ 2: IngressClassParams を作成する

IngressClassParams オブジェクトを作成して、Application Load Balancer に対する AWS 固有の設定オプションを指定します。この例では、`alb` という名前の IngressClassParams リソー

ス (次のステップで使用する) を作成し、ロードバランサースキームを `internet-facing` として `alb-ingressclassparams.yaml` というファイルで指定します。

```
apiVersion: eks.amazonaws.com/v1
kind: IngressClassParams
metadata:
  name: alb
spec:
  scheme: internet-facing
```

設定をクラスターに適用します。

```
kubectl apply -f alb-ingressclassparams.yaml
```

ステップ 3: IngressClass を作成する

`alb-ingressclass.yaml` という名前のファイル内に、`IngressClassParams` リソースで設定された AWS 固有の設定値を参照する `IngressClass` を作成します。`IngressClass` の名前をメモしてください。この例では、`IngressClass` と `IngressClassParams` の両方が `alb` という名前になっています。

`is-default-class` 注釈を使用して、`Ingress` リソースがこのクラスをデフォルトで使用するかどうかを制御します。

```
apiVersion: networking.k8s.io/v1
kind: IngressClass
metadata:
  name: alb
  annotations:
    # Use this annotation to set an IngressClass as Default
    # If an Ingress doesn't specify a class, it will use the Default
    ingressclass.kubernetes.io/is-default-class: "true"
spec:
  # Configures the IngressClass to use EKS Auto Mode
  controller: eks.amazonaws.com/alb
  parameters:
    apiGroup: eks.amazonaws.com
    kind: IngressClassParams
    # Use the name of the IngressClassParams set in the previous step
    name: alb
```

設定オプションの詳細については、「[the section called “IngressClassParams リファレンス”](#)」を参照してください。

設定をクラスターに適用します。

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

ステップ 4: Ingress を作成する

alb-ingress.yaml という名前のファイルに Ingress リソースを作成します。このリソースの目的は、Application Load Balancer のパスとポートをクラスター内のワークロードに関連付けることです。この例では、ポート 80 で service-2048 という名前のサービスにトラフィックをルーティングする 2048-ingress という名前の Ingress リソースを作成します。

このリソースの構成方法の詳細については、Kubernetes ドキュメントの「[Ingress](#)」を参照してください。

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: 2048-ingress
spec:
  # this matches the name of IngressClass.
  # this can be omitted if you have a default ingressClass in cluster: the one with
  ingressclass.kubernetes.io/is-default-class: "true"  annotation
  ingressClassName: alb
  rules:
    - http:
        paths:
          - path: /*
            pathType: ImplementationSpecific
            backend:
              service:
                name: service-2048
                port:
                  number: 80
```

設定をクラスターに適用します。

```
kubectl apply -f alb-ingress.yaml
```

ステップ 5: ステータスを確認する

kubectl を使用して、Ingress のステータスを取得します。ロードバランサーが利用できるようになるまで数分かかる場合があります。

前のステップで設定した Ingress リソースの名前を使用します。例:

```
kubectl get ingress 2048-ingress
```

リソースの準備ができたなら、ロードバランサーのドメイン名を取得します。

```
kubectl get ingress 2048-ingress -o  
jsonpath='{.status.loadBalancer.ingress[0].hostname}'
```

ウェブブラウザでサービスを表示するには、Ingress レスキューで指定したポートとパスを確認します。

ステップ 6: クリーンアップ

ロードバランサーをクリーンアップするには、次のコマンドを使用します。

```
kubectl delete ingress 2048-ingress  
kubectl delete ingressclass alb  
kubectl delete ingressclassparams alb
```

EKS Auto Mode が、AWS アカウント内の関連するロードバランサーを自動的に削除します。

IngressClassParams リファレンス

次の表は、一般的に使用される設定オプションのクイックリファレンスです。

フィールド	説明	値の例
scheme	ALB が内部向けかインターネット向けかを定義します	internet-facing
namespaceSelector	この IngressClass を使用できる名前空間を制限します	environment: prod

フィールド	説明	値の例
group.name	1 つの ALB を共有する複数の Ingress をグループ化します	retail-apps
ipAddressType	ALB の IP アドレスタイプを設定します	dualstack
subnets.ids	ALB デプロイにおけるサブネット ID のリスト	subnet-xxxx, subnet-yyyy
subnets.tags	ALB のサブネットを選択するタグフィルター	Environment: prod
certificateARNs	使用する SSL 証明書の ARN	arn:aws:acm:region:account:certificate/id
tags	AWS リソースのカスタムタグ	Environment: prod, Team: platform
loadBalancerAttributes	ロードバランサー固有の属性	idle_timeout.timeout_seconds: 60

考慮事項

- EKS Auto Mode でロードバランサーを設定するのに、IngressClass で注釈を使用することはできません。
- クラスター IAM ロールを更新して、Kubernetes から AWS ロードバランサーリソースへのタグの伝播を有効にする必要があります。詳細については、「[the section called “EKS Auto リソースのカスタム AWS タグ”](#)」を参照してください。
- リソースを EKS Auto Mode またはセルフマネージド AWS Load Balancer Controller に関連付ける方法については、「[the section called “移行のリファレンス”](#)」を参照してください。
- ロードバランサーの問題の修正方法については、「[the section called “トラブルシューティング”](#)」を参照してください。
- EKS Auto Mode のロードバランシング機能の使用に関する考慮事項については、「[the section called “負荷分散”](#)」を参照してください。

次の表は、EKS Auto Mode の IngressClassParams、Ingress 注釈、および TargetGroupBinding 設定の変更の詳細な比較を示しています。これらの表は、API バージョンの変更、廃止された機能、更新されたパラメータ名など、EKS Auto Mode のロードバランシング機能とオープンソースのロードバランサーコントローラーの主な違いを強調しています。

IngressClassParams

旧	新	説明
<code>elbv2.k8s.aws/v1beta1</code>	<code>eks.amazonaws.com/v1</code>	API バージョンの変更
<code>spec.certificateArn</code>	<code>spec.certificateARNs</code>	複数の証明書 ARN のサポート
<code>spec.subnets.tags</code>	<code>spec.subnets.matchTags</code>	サブネット一致スキーマの変更
<code>spec.listeners.listenerAttributes</code>	<code>spec.listeners.attributes</code>	属性の命名を簡素化

Ingress 注釈

旧	新	説明
<code>kubernetes.io/ingress.class</code>	サポートされていません	Ingress オブジェクトで <code>spec.ingressClassName</code> を使用
<code>alb.ingress.kubernetes.io/group.name</code>	サポートされていません	IngressClass でのみグループを指定
<code>alb.ingress.kubernetes.io/waf-acl-id</code>	サポートされていません	代わりに WAF v2 を使用
<code>alb.ingress.kubernetes.io/web-acl-id</code>	サポートされていません	代わりに WAF v2 を使用

旧	新	説明
<code>alb.ingress.kubernetes.io/shield-advanced-protection</code>	サポートされていません	Shield 統合が無効
<code>alb.ingress.kubernetes.io/auth-type: oidc</code>	サポートされていません	現時点で OIDC 認証タイプはサポートされていません

TargetGroupBinding

旧	新	説明
<code>elbv2.k8s.aws/v1beta1</code>	<code>eks.amazonaws.com/v1</code>	API バージョンの変更
<code>spec.targetType</code> が任意	<code>spec.targetType</code> が必須	明示的なターゲットタイプの指定
<code>spec.networking.ingress.from</code>	サポートされていません	セキュリティグループなしでは NLB がサポートされなくなりました

サービス注釈を使用して Network Load Balancer を設定する

Kubernetes サービス注釈を使用して Amazon EKS で Network Load Balancer (NLB) を設定する方法について説明します。このトピックでは、インターネットアクセシビリティ、ヘルスチェック、SSL/TLS ターミネーション、IP ターゲティングモードなど、NLB の動作をカスタマイズするための EKS Auto Mode でサポートされている注釈について説明します。

EKS Auto Mode でタイプ `LoadBalancer` の Kubernetes サービスを作成すると、EKS は指定した注釈に基づいて AWS Network Load Balancer を自動的にプロビジョニングして設定します。この宣言型アプローチにより、Kubernetes マニフェストを通じてロードバランサーの設定を直接管理し、Infrastructure as Code プラクティスを維持できます。

EKS Auto Mode は、デフォルトでタイプ LoadBalancer のサービスすべてに対する Network Load Balancer プロビジョニングを処理します。追加のコントローラーのインストールや設定は必要ありません。loadBalancerClass: eks.amazonaws.com/nlb の仕様は、クラスターのデフォルトとして自動的に設定され、既存の Kubernetes ワークロードとの互換性を維持しながらデプロイプロセスが合理化されます。

Note

EKS Auto Mode では、パブリックサブネットとプライベートサブネットを識別するためにサブネットタグが必要です。

eksctl を使用してクラスターを作成した場合は、これらのタグは既にあります。

「[the section called “サブネットにタグを付ける”](#)」ではその方法を説明しています。

サンプルサービス

Kubernetes Service リソースの詳細については、「[Kubernetes ドキュメント](#)」を参照してください。

以下のサンプル Service リソースを確認してください。

```
apiVersion: v1
kind: Service
metadata:
  name: echoserver
  annotations:
    # Specify the load balancer scheme as internet-facing to create a public-facing
    # Network Load Balancer (NLB)
    service.beta.kubernetes.io/aws-load-balancer-scheme: internet-facing
spec:
  selector:
    app: echoserver
  ports:
    - port: 80
      targetPort: 8080
      protocol: TCP
  type: LoadBalancer
  # Specify the new load balancer class for NLB as part of EKS Auto Mode feature
  # For clusters with Auto Mode enabled, this field can be omitted as it's the default
  loadBalancerClass: eks.amazonaws.com/nlb
```

一般的に使用される注釈

次の表に、EKS Auto Mode でサポートされている、一般的に使用される注釈を示します。EKS Auto Mode は、すべての注釈をサポートしているわけではないことに注意してください。

Tip

以下の注釈はすべて、`service.beta.kubernetes.io/` というプレフィックスを付ける必要があります

フィールド	説明	例
<code>aws-load-balancer-type</code>	ロードバランサーのタイプを指定します。新しいデプロイには <code>external</code> を使用します。	<code>external</code>
<code>aws-load-balancer-nlb-target-type</code>	トラフィックをノードインスタンスにルーティングするか、ポッド IP に直接ルーティングするかを指定します。標準デプロイでは <code>instance</code> を、直接ポッドルーティングでは <code>ip</code> を使用します。	<code>instance</code>
<code>aws-load-balancer-scheme</code>	ロードバランサーが内部向けかインターネット向けかを指定します。	<code>internet-facing</code>
<code>aws-load-balancer-healthcheck-protocol</code>	ターゲットグループのヘルスチェックプロトコル。一般的なオプションは TCP (デフォルト) または HTTP です。	<code>HTTP</code>
<code>aws-load-balancer-healthcheck-path</code>	HTTP/HTTPS プロトコルを使用する場合のヘルスチェックの HTTP パス。	<code>/healthz</code>

フィールド	説明	例
aws-load-balancer-healthcheck-port	ヘルスチェックに使用されるポート。特定のポート番号または <code>traffic-port</code> を指定できます。	<code>traffic-port</code>
aws-load-balancer-subnets	ロードバランサーを作成するサブネットを指定します。サブネット ID またはサブネット名を使用できます。	<code>subnet-xxxx</code> , <code>subnet-yyyy</code>
aws-load-balancer-ssl-cert	HTTPS/TLS 用の AWS Certificate Manager からの SSL 証明書の ARN。	<code>arn:aws:acm:region:account:certificate/cert-id</code>
aws-load-balancer-ssl-ports	SSL/TLS を使用するポートを指定します。	<code>443</code> , <code>8443</code>
load-balancer-source-ranges	ロードバランサーへのアクセスが許可されている CIDR 範囲。	<code>10.0.0.0/24</code> , <code>192.168.1.0/24</code>
aws-load-balancer-additional-resource-tags	ロードバランサーおよび関連リソースに適用する追加の AWS タグ。	<code>Environment=prod</code> , <code>Team=platform</code>
aws-load-balancer-ip-address-type	ロードバランサーが使用するのが IPv4 かデュアルスタック (IPv4 + IPv6) を指定します。	<code>ipv4</code> 、または <code>dualstack</code>

考慮事項

- クラスター IAM ロールを更新して、Kubernetes から AWS ロードバランサーリソースへのタグの伝播を有効にする必要があります。詳細については「[the section called “EKS Auto リソースのカスタム AWS タグ”](#)」を参照してください。

- リソースを EKS Auto Mode またはセルフマネージド AWS Load Balancer Controller に関連付ける方法については、「[the section called “移行のリファレンス”](#)」を参照してください。
- ロードバランサーの問題の修正方法については、「[the section called “トラブルシューティング”](#)」を参照してください。
- EKS Auto Mode のロードバランシング機能の使用に関する考慮事項については、「[the section called “負荷分散”](#)」を参照してください。

ロードバランシングのために EKS Auto Mode に移行する場合は、サービス注釈とリソース設定にいくつかの変更が必要です。次の表は、サポートされていないオプションや推奨される代替案など、新旧の実装の主な違いをまとめたものです。

サービス注釈

旧	新	説明
<code>service.beta.kubernetes.io/load-balancer-source-ranges</code>	サポートされていません	Service で <code>spec.loadBalancerSourceRanges</code> を使用
<code>service.beta.kubernetes.io/aws-load-balancer-type</code>	サポートされていません	Service で <code>spec.loadBalancerClass</code> を使用
<code>service.beta.kubernetes.io/aws-load-balancer-internal</code>	サポートされていません	<code>service.beta.kubernetes.io/aws-load-balancer-scheme</code> を使用する
各種ロードバランサー属性	サポートされていません	<code>service.beta.kubernetes.io/aws-load-balancer-attributes</code> を使用する
<code>service.beta.kubernetes.io/aws-load-balancer-proxy-protocol</code>	サポートされていません	代わりに <code>service.beta.kubernetes.io/aws-load-balancer-attributes</code> を使用

旧	新	説明
<code>service.beta.kubernetes.io/aws-load-balancer-access-log-enabled</code>	サポートされていません	代わりに <code>service.beta.kubernetes.io/aws-load-balancer-attributes</code> を使用
<code>service.beta.kubernetes.io/aws-load-balancer-access-log-s3-bucket-name</code>	サポートされていません	代わりに <code>service.beta.kubernetes.io/aws-load-balancer-attributes</code> を使用
<code>service.beta.kubernetes.io/aws-load-balancer-access-log-s3-bucket-prefix</code>	サポートされていません	代わりに <code>service.beta.kubernetes.io/aws-load-balancer-attributes</code> を使用
<code>service.beta.kubernetes.io/aws-load-balancer-cross-zone-load-balancing-enabled</code>	サポートされていません	代わりに <code>service.beta.kubernetes.io/aws-load-balancer-attributes</code> を使用

非推奨のロードバランサー属性注釈から移行するには、これらの設定を

`service.beta.kubernetes.io/aws-load-balancer-attributes` 注釈に統合します。この注釈は、各種ロードバランサー属性のキーと値のペアのカンマ区切りリストを受け入れます。例えば、プロキシプロトコル、アクセスログ記録、クロスゾーンロードバランシングを指定するには、次の形式を使用します。

```
service.beta.kubernetes.io/aws-load-balancer-attributes: |
  access_logs.s3.enabled=true
  access_logs.s3.bucket=my-bucket
  access_logs.s3.prefix=my-prefix
  load_balancing.cross_zone.enabled=true
```

```
service.beta.kubernetes.io/aws-load-balancer-target-group-attributes: |
  proxy_protocol_v2.enabled=true
```


この統合形式により、より優れた一貫性と柔軟性で、必要な個々の注釈の数を減らしつつロードバランサー属性を設定することができます。既存の Service 設定を確認し、この統合形式を使用するように更新してください。

TargetGroupBinding

旧	新	説明
<code>elbv2.k8s.aws/v1beta1</code>	<code>eks.amazonaws.com/v1</code>	API バージョンの変更
<code>spec.targetType</code> が任意	<code>spec.targetType</code> が必須	明示的なターゲットタイプの指定
<code>spec.networking.ingress.from</code>	サポートされていません	セキュリティグループなしでは NLB がサポートされなくなりました

ストレージクラスを作成する

Amazon EKS Auto Mode の `StorageClass` は、アプリケーションが永続的ストレージをリクエストしたときの Amazon EBS ボリュームを自動的にプロビジョニングする方法を定義します。このページでは、Amazon EKS Auto Mode で動作する `StorageClass` を作成して設定し、EBS ボリュームをプロビジョニングする方法について説明します。

`StorageClass` を設定することで、ボリュームタイプ、暗号化、IOPS、その他のストレージパラメータなど、EBS ボリュームのデフォルト設定を指定できます。暗号化管理に AWS KMS キーを使用するように `StorageClass` を設定することもできます。

EKS Auto Mode では、お客様用の `StorageClass` は作成されません。EKS Auto Mode のストレージ機能を使用するには、`ebs.csi.eks.amazonaws.com` を参照する `StorageClass` を作成する必要があります。

まず、`storage-class.yaml` という名前のファイルを作成します。

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
```

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

次に、ストレージクラスをクラスターに適用します。

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

主要コンポーネント:

- `provisioner: ebs.csi.eks.amazonaws.com` - EKS Auto Mode を使用する
- `volumeBindingMode: WaitForFirstConsumer` - ポッドが必要とするまでボリュームの作成を遅らせる
- `type: gp3` - EBS ボリュームタイプを指定する
- `encrypted: "true"` - EBS は、StorageClass を使用して作成されたすべてのボリュームを暗号化します。EBS はデフォルトの `aws/ebs` キーエイリアスを使用します。詳細については、「Amazon EBS ユーザーガイド」の「[How Amazon EBS encryption works](#)」を参照してください。この値は省略可能ですが、推奨です。
- `storageclass.kubernetes.io/is-default-class: "true"` - 永続ボリュームのクレームで別のボリュームクラスを指定しない限り、Kubernetes はデフォルトでこのストレージクラスを使用します。この値はオプションです。別のストレージコントローラーから移行する場合は、この値を設定するときに注意してください。

セルフマネージド KMS キーを使用して EBS ボリュームを暗号化する

セルフマネージド KMS キーを使用して EKS Auto Mode によって自動化される EBS ボリュームを暗号化するには、以下を実行する必要があります。

1. セルフマネージド KMS キーを作成します。
 - 詳細については、「KMS ユーザーガイド」の「[Create a symmetric encryption KMS key](#)」または「[How Amazon Elastic Block Store \(Amazon EBS\) uses KMS](#)」を参照してください。
2. KMS キーへのアクセスを許可する新しいポリシーを作成します。

- 以下のサンプル IAM ポリシーを使用してポリシーを作成します。新しいセルフマネージド KMS キーの ARN を挿入します。詳細については、「AWS IAM ユーザーガイド」の「[ロールの作成とポリシーのアタッチ \(コンソール\)](#)」を参照してください。
3. ポリシーを EKS クラスターロールにアタッチします。
 - AWS コンソールを使用して、EKS クラスターロールの ARN を見つけます。ロール情報は [概要] セクションに表示されます。詳細については、「[the section called “クラスター IAM ロール”](#)」を参照してください。
 4. `parameters.kmsKeyId` フィールドの KMS キー ID を参照するように `StorageClass` を更新します。

サンプルセルフマネージド KMS IAM ポリシー

以下のポリシーで次の値を更新します。

- `<account-id>` – AWS アカウント ID (111122223333 など)
- `<aws-region>` – クラスターの AWS リージョン (us-west-2 など)

```
{
  "Version": "2012-10-17",
  "Id": "key-auto-policy-3",
  "Statement": [
    {
      "Sid": "Enable IAM User Permissions",
      "Effect": "Allow",
      "Principal": {
        "AWS": "arn:aws:iam::<account-id>:root"
      },
      "Action": "kms:*",
      "Resource": "*"
    },
    {
      "Sid": "Allow access through EBS for all principals in the account that are authorized to use EBS",
      "Effect": "Allow",
      "Principal": {
        "AWS": "*"
      },
      "Action": [
        "kms:Encrypt",
```

```

        "kms:Decrypt",
        "kms:ReEncrypt*",
        "kms:GenerateDataKey*",
        "kms:CreateGrant",
        "kms:DescribeKey"
    ],
    "Resource": "*",
    "Condition": {
        "StringEquals": {
            "kms:CallerAccount": "<account-id>",
            "kms:ViaService": "ec2.<aws-region>.amazonaws.com"
        }
    }
}
]
}

```

サンプルセルフマネージド KMS **StorageClass**

```

parameters:
  type: gp3
  encrypted: "true"
  kmsKeyId: <custom-key-arn>

```

StorageClass パラメータリファレンス

Kubernetes StorageClass リソースの一般的な情報については、Kubernetes ドキュメントの「[ストレージクラス](#)」を参照してください。

StorageClass リソースの parameters セクションは AWS に固有です。次の表を使用して使用可能なオプションを確認してください。

パラメータ	値	デフォルト	説明
「csi.storage.k8s.io/fstype」	xfs、ext2、ext3、ext4	ext4	ボリュームの作成時にフォーマットされるファイルシステムタイプ。このパラメータは大文字と小文字が区別されます。

パラメータ	値	デフォルト	説明
「type」	io1、io2、gp2、gp3、sc1、st1、standard、sbp1、sbg1	gp3	EBS ボリュームタイプ。
「iopsPerGB」			GiB ごとの 1 秒あたりの I/O オペレーション回数秒。IO1、IO2、GP3 ボリュームで指定できます。
「allowAutoIOPSPerGBIncrease」	true、false	false	"true" である と、iopsPerGB * <volume size> が小さすぎて AWS でサポートされている IOPS 範囲に収まらない場合に、CSI ドライバーによりボリュームの IOPS が増加します。この結果、ユーザーが指定する PVC キャパシティや iopsPerGB 値が小さすぎる場合でも、動的プロビジョニングが常に成功します。一方、このようなボリュームの IOPS は iopsPerGB でリクエストされた値よりも高くなるため、追加コストが発生する可能性があります。

パラメータ	値	デフォルト	説明
「iops」			1 秒あたりの I/O オペレーション回数。 IO1、IO2、GP3 ボリュームで指定できます。
「throughput」		125	単位 MiB/秒でのスループット。gp3 ボリュームタイプが指定されている場合にのみ有効です。
「encrypted」	true、false	false	ボリュームを暗号化するかどうか。有効な値は「true」または「false」です。
「blockExpress」	true、false	false	io2 Block Express ボリュームの作成を有効にします。
「kmsKeyId」			ボリュームの暗号化時に使用するキーの完全な ARN。指定されていない場合、AWS ではボリュームが存在するリージョンのデフォルトの KMS キーが使用されます。これは、変更されていない場合、/aws/ebs という名前の自動生成キーになります。

パラメータ	値	デフォルト	説明
「blockSize」			基盤となるファイルシステムをフォーマットするときに使用するブロックサイズ。Linux ノードと fstype ext2、ext3、ext4 または xfs でのみサポートされます。
「inodeSize」			基盤となるファイルシステムをフォーマットするときに使用する inode サイズ。Linux ノードと fstype ext2、ext3、ext4 または xfs でのみサポートされます。
「bytesPerInode」			基盤となるファイルシステムをフォーマットするときに使用する bytes-per-inode 。Linux ノードと fstype ext2、ext3、ext4 でのみサポートされます。

パラメータ	値	デフォルト	説明
「numberOfInodes」			基盤となるファイルシステムをフォーマットするときに使用する number-of-inodes 。Linux ノードと fstype ext2、ext3、ext4 でのみサポートされます。
「ext4BigAlloc」	true、false	false	bigalloc フォーマットオプションを有効にして、クラスター化されたブロック割り当てを使用するように ext4 ファイルシステムを変更します。警告: ノードの Linux カーネルでは bigalloc が完全にはサポートされない場合があります。
「ext4ClusterSize」			bigalloc 機能が有効になっている場合に、ext4 ファイルシステムをフォーマットするときに使用するクラスターサイズ。注: ext4BigAlloc パラメータを true に設定する必要があります。

詳細については、GitHub の「[AWS EBS CSI Driver](#)」を参照してください。

考慮事項

EKS Auto Mode のブロックストレージ機能は、EBS CSI ドライバーとは異なります。

- 静的プロビジョニング
 - EKS Auto Mode で外部作成の EBS ボリュームを使用する場合は、キー `eks:eks-cluster-name` とクラスター名の値を含む AWS タグを手動で追加する必要があります。
- ノード起動テイント
 - ノード起動テイント機能を使用して、ストレージ機能の準備が整う前にポッドのスケジューリングを防ぐことはできません
- 動的にプロビジョニングされたボリュームのカスタムタグ
 - 追加タグ CLI フラグを使用して、動的にプロビジョニングされた EBS ボリュームにカスタムタグを設定することはできません
 - StorageClass タグ付けを使用してカスタムタグを追加できます。EKS Auto Mode は、関連付けられた AWS リソースにタグを追加します。カスタムタグのクラスター IAM ロールを更新する必要があります。詳細については、「[the section called “EKS Auto リソースのカスタム AWS タグ”](#)」を参照してください。
- EBS 詳細パフォーマンスメトリクス
 - EBS 詳細パフォーマンスの Prometheus メトリクスにアクセスできません

CSI スナップショットコントローラーアドオンをインストールする

EKS Auto Mode は、CSI スナップショットコントローラー Amazon EKS アドオンと互換性があります。

AWS では、組み込み system ノードプールで実行するようにこのアドオンを設定することをお勧めします。

詳細については、以下を参照してください。

- [the section called “重要なアドオンを実行する”](#)
- [the section called “組み込みノードプールを確認する”](#)
- [the section called “CSI スナップショットコントローラー”](#)

スナップショットコントローラーをシステムノードプールにインストールするには

1. AWS コンソールで EKS クラスターを開きます
2. [アドオン] タブから、[アドオンをさらに追加] を選択します
3. [CSI スナップショットコントローラー]、[次へ] の順に選択します
4. [選択したアドオン設定を構成する] ページで、[オプションの設定] を選択して [アドオン設定スキーマ] を表示します
 - a. 次の yaml を挿入して、スナップショットコントローラーを system ノードプールに関連付けます。スナップショットコントローラーには、CriticalAddonsOnly テイントの許容が含まれています。

```
{
  "nodeSelector": {
    "karpenter.sh/nodepool": "system"
  }
}
```

- b. [次へ] を選択します
5. アドオン設定を確認し、[作成] を選択します

EKS 自動モードI を無効にする

既存の EKS クラスターで EKS 自動モードI を無効にすることができます。これは破壊的な操作です。

- EKS はEKS 自動モードI によって運用されているすべての EC2 インスタンスを終了します。
- EKS はEKS 自動モードI によって運用されているすべてのロードバランサーを削除してください。
- EKS はEKS 自動モードI によってプロビジョニングされた EBS ボリュームを削除しません。

EKS Auto Mode は作成するリソースを完全に管理するように設計されています。手動で操作すると、EKS Auto Mode が無効になっているときにそれらのリソースを完全にクリーンアップできなくなる可能性があります。例えば、外部のセキュリティグループルールからマネージドセキュリティグループを参照し、クラスターの EKS Auto Mode を無効にする前にその参照を削除するのを忘れた場合、マネージドセキュリティグループはリークします (削除されません)。以下のステップではリークされたセキュリティグループが発生した場合に削除する方法について説明します。

EKS 自動モードI を無効にする (AWS コンソール)

1. AWS Management Consoleでクラスターの概要ページを開きます。
2. [EKS 自動モードI] で [管理] を選択してください
3. EKS 自動モードI を off に切り替えます。

このプロセスの最後にマネージドセキュリティグループが削除されない場合は、「[セキュリティグループを削除する](#)」の説明を参照して手動で削除できます。

EKS 自動モードI を無効にする (AWS CLI)

既存のクラスターで EKS 自動モードI を無効にするには次のコマンドを使用します。

aws CLI をインストールし、EKS クラスターを管理するのに十分なアクセス許可が付与された状態でログインする必要があります。詳細については「[セットアップする](#)」を参照してください。

Note

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

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

EKS Auto Mode を無効にした後、リークした EKS Auto Mode セキュリティグループの削除に失敗したかどうかは次のように確認できます。

```
aws ec2 describe-security-groups \
  --filters Name=tag:eks:eks-cluster-name,Values=<cluster-Name> Name=tag-
key,Values=ingress.eks.amazonaws.com/resource,service.eks.amazonaws.com/resource --
query "SecurityGroups[*].[GroupName]"
```

セキュリティグループを削除するには：

```
aws ec2 delete-security-group --group-name=<sg-name>
```

EKS Auto Mode クラスターの Kubernetes バージョンを更新する

このトピックでは、Auto Mode クラスターの Kubernetes バージョンを更新する方法について説明します。Auto Mode は、ポッド中断の予算を通じてワークロードの可用性を維持しながら、ノード交換によるコントロールプレーンの更新を調整することで、バージョン更新プロセスを簡素化します。

Auto Mode クラスターのアップグレード時に、従来手動更新が必要だった多くのコンポーネントがサービスの一部として管理されるようになりました。アップグレードプロセスの自動化された側面とユーザー担当部分を理解することで、クラスターのバージョンをスムーズに移行できます。

EKS Auto Mode による更新についての説明

コントロールプレーンのアップグレードを開始すると、EKS Auto Mode はクラスター内のノードの置き換えを開始します。新しいノードには、対応する新しい Kubernetes バージョンがあります。EKS Auto Mode は、ノードのアップグレード時にポッド中断の予算を監視します。

さらに、次のようなコンポーネントは更新する必要がありません。

- Amazon VPC CNI
- AWS ロードバランサーコントローラー
- CoreDNS
- kube-proxy
- Karpenter
- AWS EBS CSI ドライバー

EKS Auto Mode は、これらのコンポーネントをサービス機能に置き換えます。

以下の更新は引き続きユーザーが行います。

- クラスターにデプロイされたアプリケーションとワークロード
- セルフマネージドアドオンおよびコントローラー
- Amazon EKS アドオン
 - 「[the section called “アドオンを更新する”](#)」ではその方法を説明しています。

「[クラスターのアップグレードのベストプラクティス](#)」を参照してください

クラスターの更新を開始する

クラスターの更新を開始するには、「[the section called “Kubernetes バージョンを更新する”](#)」を参照してください。

組み込み NodePool を有効または無効にする

EKS Auto Mode には 2 つの組み込み NodePool があります。これらの NodePool を有効または無効にするには、AWS コンソール、CLI、または API を使用します。

組み込み NodePool リファレンス

- `system`
 - この NodePool には `CriticalAddonsOnly` ティントがあります。CoreDNS など多くの EKS アドオンで、このティントが許容されています。このシステムノードプールを使用して、クラスターに不可欠なアプリケーションを分離します。
 - amd64 と arm64 アーキテクチャの両方をサポートします。
- `general-purpose`
 - この NodePool は、クラスター内の汎用ワークロード用のノードの起動をサポートします。
 - amd64 アーキテクチャのみを使用します。

両方の組み込み NodePool:

- デフォルトの EKS NodeClass を使用する
- オンデマンド EC2 容量のみを使用する
- C、M、および R EC2 インスタンスファミリーを使用する
- 世代 5 以降の EC2 インスタンスが必要になる

Note

EKS が「デフォルト」の NodeClass をプロビジョニングするには、少なくとも 1 つの組み込み NodePool を有効にする必要があります。すべての組み込み NodePool を無効にする場合は、カスタム NodeClass を作成し、それを使用するように NodePool を設定する必要があります。NodeClass の詳細については、「[the section called “ノードクラスの実装”](#)」を参照してください。

手順

前提条件

- AWS コマンドラインインターフェイス (AWS CLI) の最新バージョンがデバイスにインストールおよび設定されていること。現在のバージョンを確認するには「aws --version」を参照してください。最新バージョンをインストールするには、「AWS コマンドラインインターフェイスユーザーガイド」の「[インストール](#)」および「[aws configure を使用したクイック設定](#)」を参照してください。
- 十分な IAM アクセス許可を使用して CLI にログインし、IAM ポリシー、IAM ロール、EKS クラスターなどの AWS リソースを作成します。

AWS CLI で有効にする

次のコマンドを使用して、両方の組み込み NodePool を有効にします。

```
aws eks update-cluster-config \
  --name <cluster-name> \
  --compute-config '{
    "nodeRoleArn": "<node-role-arn>",
    "nodePools": ["general-purpose", "system"],
    "enabled": true
  }' \
  --kubernetes-network-config '{
    "elasticLoadBalancing":{"enabled": true}
  }' \
  --storage-config '{
    "blockStorage":{"enabled": true}
  }'
```

コマンドを変更して、NodePool を選択的に有効にすることができます。

AWS CLI で無効にする

次のコマンドを使用して、両方の組み込み NodePool を無効にします。

```
aws eks update-cluster-config \
  --name <cluster-name> \
  --compute-config '{
    "enabled": true,
```

```
"nodePools": []
}' \
--kubernetes-network-config '{
"elasticLoadBalancing":{"enabled": true}}' \
--storage-config '{
"blockStorage":{"enabled": true}
}'
```

ワークロードが EKS Auto Mode ノードにデプロイされるかどうかを制御する

EKS Auto Mode を使用して EKS クラスターでワークロードを実行する場合、特定のワークロードが EKS Auto Mode ノードで実行されるか、他のコンピューティングタイプで実行されるかを制御することが必要になる場合があります。このトピックでは、ノードセクターとアフィニティルールを使用して、ワークロードを目的のコンピューティングインフラストラクチャでスケジュールする方法について説明します。

このトピックの例では、`eks.amazonaws.com/compute-type` ラベルを使用して EKS Auto Mode ノードへのワークロードのデプロイを要求または禁止する方法を示します。これは、EKS Auto Mode と、セルフマネージド Karpenter プロビジョナーや EKS マネージドノードグループなどの他のコンピューティングタイプの両方を実行している混合モードクラスターで特に便利です。

EKS Auto Mode ノードでは、ラベル `eks.amazonaws.com/compute-type` の値が `auto` に設定されています。このラベルを使用して、ワークロードが EKS Auto Mode によって管理されるノードにデプロイされるかどうかを制御できます。

ワークロードを EKS Auto Mode ノードにデプロイするように要求する

Note

この `nodeSelector` 値は EKS Auto Mode では必須ではありません。この `nodeSelector` 値は、クラスターを混合モードで実行している場合にのみ関連し、ノードタイプは EKS Auto Mode で管理されません。例えば、EKS マネージドノードグループを使用してクラスターに静的コンピューティング容量をデプロイし、EKS Auto Mode で動的計算能力を管理しているとします。

この `nodeSelector` をデプロイまたは他のワークロードに追加して、Kubernetes が EKS Auto Mode ノードにスケジュールするように要求できます。

```
apiVersion: apps/v1
kind: Deployment
spec:
  template:
    nodeSelector:
      eks.amazonaws.com/compute-type: auto
```

ワークロードを EKS Auto Mode ノードにデプロイしないように要求する

この `nodeAffinity` をデプロイまたは他のワークロードに追加して、Kubernetes が EKS Auto Mode ノードにスケジュールしないように要求できます。

```
affinity:
  nodeAffinity:
    requiredDuringSchedulingIgnoredDuringExecution:
      nodeSelectorTerms:
        - matchExpressions:
            - key: eks.amazonaws.com/compute-type
              operator: NotIn
              values:
                - auto
```

ハードウェア専有インスタンスで重要なアドオンを実行する

このトピックでは、EKS Auto Mode が system ノードプールにワークロードをスケジュールできるように、`CriticalAddonsOnly` の容認を持つワークロードをデプロイする方法について説明します。

EKS Auto Mode の組み込みの system ノードプールは、専用インスタンス上で重要なアドオンを実行するために設計されています。この分離により、重要なコンポーネントに専用のリソースが備わり、一般的なワークロードから分離されることが保証され、クラスター全体の安定性とパフォーマンスが向上します。

このガイドでは、`CriticalAddonsOnly` の容認と適切なノードセクターを利用して、system ノードプールにアドオンをデプロイする方法を説明します。これらのステップに従うことで、EKS Auto Mode の特殊なノードプール構造が提供する分離とリソース割り当ての利点を活用して、重要なアプリケーションを専用の system ノードにスケジュールできるようになります。

EKS Auto Mode には、general-purpose と system の 2 つのノードプールが組み込まれています。詳細については、「[the section called “組み込みノードプールを確認する”](#)」を参照してください。

system ノードプールの目的は、重要なアドオンを異なるノードに分離することです。system ノードプールによってプロビジョニングされたノードには、CriticalAddonsOnly という Kubernetes のテイントがあります。Kubernetes は、対応する容認がある場合のみ、これらのノードにポッドをスケジュールします。詳細については、Kubernetes ドキュメントの「[テイントと容認](#)」を参照してください。

前提条件

- EKS Auto Mode クラスターで組み込みの system ノードプールが有効になっていること。詳細については、「[the section called “組み込みノードプールを確認する”](#)」を参照してください。
- kubectl がインストールされ、設定済みであること。詳細については、「[セットアップする](#)」を参照してください。

手順

以下の yaml の例を確認してください。以下の設定に留意してください。

- nodeSelector — これにより、ワークロードが組み込みの system ノードプールに関連付けられます。このノードプールは AWS API で有効にする必要があります。詳細については、「[the section called “組み込みノードプールを確認する”](#)」を参照してください。
- tolerations — この容認は、system ノードプール内のノードの CriticalAddonsOnly テイントを克服します。

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: sample-app
spec:
  replicas: 3
  selector:
    matchLabels:
      app: sample-app
  template:
    metadata:
```

```
labels:
  app: sample-app
spec:
  nodeSelector:
    karpenter.sh/nodepool: system
  tolerations:
  - key: "CriticalAddonsOnly"
    operator: "Exists"
  containers:
  - name: app
    image: nginx:latest
    resources:
      requests:
        cpu: "500m"
        memory: "512Mi"
```

system ノードプールで実行するようにワークロードを更新するには、以下の操作を行う必要があります。

1. 既存のワークロードを更新して、上記で説明した以下の設定を追加します。
 - nodeSelector
 - tolerations
2. `kubectl apply` を使用して、更新されたワークロードをクラスターにデプロイする

ワークロードを更新すると、専用ノードで実行されるようになります。

EKS Auto Mode でネットワークポリシーを使用する

ネットワークポリシーを使用すると、Amazon EKS クラスター内の IP アドレスまたはポートレベルでトラフィックフローを制御できます。このトピックでは、EKS Auto Mode でネットワークポリシーを有効にして使用方法について説明します。

前提条件

- EKS Auto Mode が有効になっている Amazon EKS クラスター
- `kubectl` がクラスターに接続するように設定されている

ステップ 1: ネットワークポリシーコントローラーを有効にする

EKS Auto Mode でネットワークポリシーを使用するには、まずクラスターに ConfigMap を適用してネットワークポリシーコントローラーを有効にする必要があります。

1. 以下の内容で `enable-network-policy.yaml` という名前のファイルを作成します。

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: amazon-vpc-cni
  namespace: kube-system
data:
  enable-network-policy-controller: "true"
```

2. ConfigMap をクラスターに適用します。

```
kubectl apply -f enable-network-policy.yaml
```

ステップ 2: ノードクラスでネットワークポリシーを有効にする

ネットワークポリシーを使用するには、ノードクラスがそれらをサポートするように設定する必要があります。以下の手順に従ってください。

1. 次の内容でノードクラス YAML ファイル (例: `nodeclass-network-policy.yaml`) を作成または編集します。

```
apiVersion: eks.amazonaws.com/v1
kind: NodeClass
metadata:
  name: network-policy-enabled
spec:
  # Enables network policy support
  networkPolicy: DefaultAllow
  # Optional: Enables logging for network policy events
  networkPolicyEventLogs: Enabled
  # Include other Node Class configurations as needed
```

2. ノードクラス設定をクラスターに適用します。

```
kubectl apply -f nodeclass-network-policy.yaml
```

3. ノードクラスが作成されていることを確認します。

```
kubectl get nodeclass network-policy-enabled
```

4. このノードクラスを使用するようにノードプールを更新します。詳細については、「[the section called “ノードプールの作成”](#)」を参照してください。

ノードがこのノードクラスを使用していると、ネットワークポリシーを適用できるようになります。これで、クラスター内のトラフィックを制御するネットワークポリシーの作成と適用に進むことができます。すべてのノードクラス設定オプションについては、「[the section called “ノードクラスの作成”](#)」を参照してください。

ステップ 3: ネットワークポリシーを作成してテストする

これで、EKS Auto Mode クラスターが Kubernetes ネットワークポリシーをサポートするように設定されました。これは [the section called “Stars Policy デモ”](#) でテストできます。

EKS Auto Mode のサブネットにタグを付ける

EKS Auto Mode の負荷分散機能を使用する場合は、VPC サブネットに AWS タグを追加する必要があります。

背景

これらのタグは、サブネットがクラスターに関連付けられているものと特定し、さらに重要な点として、サブネットがパブリックかプライベートかを識別します。

パブリックサブネットは、インターネットゲートウェイを介してインターネットに直接アクセスできます。ロードバランサーなど、パブリックにアクセスできるようにする必要があるリソースに使用されます。

プライベートサブネットには直接インターネットにアクセスできる機能がなく、アウトバウンドトラフィックに NAT ゲートウェイを使用します。プライベートサブネットは、パブリック IP を必要としない EKS ノードなどの内部リソースに使用されます。

NAT ゲートウェイとインターネットゲートウェイの詳細については、「Amazon Virtual Private Cloud (VPC) ユーザーガイド」の「[VPC を他のネットワークに接続する](#)」を参照してください。

要件

現時点で、EKS Auto Mode で負荷分散に使用するサブネットには、次のいずれかのタグが必要です。

パブリックサブネット

パブリックサブネットは、インターネット向けロードバランサーに使用されます。これらのサブネットには、次のタグが必要です。

キー	値
kubernetes.io/role/elb	1 または ``

プライベートサブネット

プライベートサブネットは、内部ロードバランサーに使用されます。これらのサブネットには、次のタグが必要です。

キー	値
kubernetes.io/role/internal-elb	1 または ``

手順

開始する前に、パブリックサブネット (インターネットゲートウェイアクセスを使用) とプライベートサブネット (NAT ゲートウェイを使用) を特定します。VPC リソースを変更するには、アクセス許可が必要です。

AWS Management Console

1. Amazon VPC コンソールを開き、[サブネット] に移動します。
2. タグ付けするサブネットを選択します。
3. [タグ] タブを選択してから、[タグを追加] を選択します。
4. 適切なタグを追加します。
 - パブリックサブネットの場合: Key=kubernetes.io/role/elb

- プライベートサブネットの場合: Key=kubernetes.io/role/internal-elb

5. [値] を 1 に設定するか、空のままにします。

6. 保存して、残りのサブネットについても繰り返します。

AWS CLI

パブリックサブネットの場合:

```
aws ec2 create-tags \  
  --resources subnet-ID \  
  --tags Key=kubernetes.io/role/elb,Value=1
```

プライベートサブネットの場合:

```
aws ec2 create-tags \  
  --resources subnet-ID \  
  --tags Key=kubernetes.io/role/internal-elb,Value=1
```

subnet-ID を実際のサブネット ID に置き換えます。

加速ワークロードをデプロイする

このチュートリアルでは、Amazon EKS Auto Mode が加速ワークロードの開始をどのように簡素化しているかを説明します。Amazon EKS Auto Mode は、コンピューティング、ネットワーキング、ロードバランシング、ストレージ、Identity Access and Management の各機能を追加の設定なしで実現する主要なインフラストラクチャコンポーネントを自動化することで、クラスター自体にとどまらずさまざまな運用を簡素化します。

Amazon EKS Auto Mode には、NVIDIA ドライバーや AWS Neuron ドライバーなど、特定のインスタンスタイプに必要なドライバーとデバイスプラグインが含まれています。これらのコンポーネントをお客様側でインストールおよび更新する必要はありません。

EKS Auto Mode が次のアクセラレーター用のドライバーを自動的に管理します。

- [AWS Trainium](#)
- [AWS Inferentia](#)
- [NVIDIA GPUs on Amazon EC2 加速インスタンス](#)

Note

EKS Auto Mode には、Kubernetes 用の NVIDIA デバイスプラグインが含まれています。このプラグインは、自動的に動作し、クラスターにデーモンセットとして表示されません。

ネットワーキングに関するその他のサポート:

- [Elastic Fabric Adapter \(EFA\)](#)

Amazon EKS Auto Mode により、アクセラレーター用のドライバーとデバイスプラグインを管理するという手間が不要になります。

また、クラスターがゼロにスケーリングされるので、コスト削減というメリットも得られます。実行中のワークロードがなくなったらインスタンスを終了するように、EKS Auto Mode を設定できます。これはバッチベースの推論ワークロードに便利です。

以下に、Amazon EKS Auto Mode で加速ワークロードを開始する方法の例を示します。

前提条件

- Amazon EKS Auto Mode で Kubernetes クラスターが設定されていること。
- `general-purpose` または `system` のいずれかのマネージドノードプールが有効であれば、`default` EKS ノードクラスが作成されること。

ステップ 1: GPU ワークロードをデプロイする

この例では、45 GB の GPU メモリを必要とする NVIDIA ベースのワークロード用に NodePool を作成します。EKS Auto Mode では、Kubernetes スケジューリング制約を使用して、インスタンス要件を定義します。

Amazon EKS Auto Mode NodePool とサンプルの workload をデプロイするには、NodePool とポッドに関する以下の定義を確認し、`nodepool-gpu.yaml` および `pod.yaml` として保存します。

`nodepool-gpu.yaml`

```
apiVersion: karpenter.sh/v1
kind: NodePool
metadata:
```

```
name: gpu
spec:
  disruption:
    budgets:
      - nodes: 10%
    consolidateAfter: 1h
    consolidationPolicy: WhenEmpty
  template:
    metadata: {}
    spec:
      nodeClassRef:
        group: eks.amazonaws.com
        kind: NodeClass
        name: default
      requirements:
        - key: "karpenter.sh/capacity-type"
          operator: In
          values: ["on-demand"]
        - key: "kubernetes.io/arch"
          operator: In
          values: ["amd64"]
        - key: "eks.amazonaws.com/instance-family"
          operator: In
          values:
            - g6e
            - g6
      taints:
        - key: nvidia.com/gpu
          effect: NoSchedule
      terminationGracePeriod: 24h0m0s
```

pod.yaml

```
apiVersion: v1
kind: Pod
metadata:
  name: nvidia-smi
spec:
  nodeSelector:
    eks.amazonaws.com/instance-gpu-name: 140s
    eks.amazonaws.com/compute-type: auto
  restartPolicy: OnFailure
  containers:
```



```
- name: nvidia-smi
  image: public.ecr.aws/amazonlinux/amazonlinux:2023-minimal
  args:
  - "nvidia-smi"
  resources:
    requests:
      memory: "30Gi"
      cpu: "3500m"
      nvidia.com/gpu: 1
    limits:
      memory: "30Gi"
      nvidia.com/gpu: 1
  tolerations:
  - key: nvidia.com/gpu
    effect: NoSchedule
    operator: Exists
```

eks.amazonaws.com/compute-type: auto セレクターでは、ワークロードを Amazon EKS Auto Mode ノードにデプロイする必要があることに注意してください。また、NodePool は Nvidia GPU に対する許容範囲が設定されたポッドに限りスケジュールできるというテイントを設定します。

NodePool とワークロードをクラスターに適用します。

```
kubectl apply -f nodepool-gpu.yaml
kubectl apply -f pod.yaml
```

以下の出力が表示されます。

```
nodepool.karpenter.sh/gpu configured created
pod/nvidia-smi created
```

数秒待ってからクラスター内のノードを確認します。これで、新しいノードが Amazon EKS Auto Mode クラスターにプロビジョニングされるはずです。

```
> kubectl get nodes
```

NAME	TYPE	CAPACITY	ZONE	NODE	READY	AGE
gpu-dnknr	g6e.2xlarge	on-demand	us-west-2b	i-02315c7d7643cdee6	True	76s

ステップ 2: 検証する

Amazon EKS Auto Mode が以下の Kubernetes スケジューリング制約に従って、l40s GPU 搭載のインスタンスを必要とするワークロードとして g6e.2xlarge ではなく g6.2xlarge を開始したことを確認できます。

```
...
nodeSelector:
  eks.amazonaws.com/instance-gpu-name: l40s
...
requests:
  memory: "30Gi"
  cpu: "3500m"
  nvidia.com/gpu: 1
limits:
  memory: "30Gi"
  nvidia.com/gpu: 1
```

次のコマンドを実行して、コンテナのログを参照します。

```
kubectl logs nvidia-smi
```

サンプル出力:

```
+-----+
+
| NVIDIA-SMI 535.230.02                Driver Version: 535.230.02   CUDA Version: 12.2
|
|-----+-----+
+-----+
| GPU   Name                               Persistence-M | Bus-Id        Disp.A | Volatile Uncorr. ECC
|
| Fan   Temp   Perf           Pwr:Usage/Cap |      Memory-Usage | GPU-Util  Compute M.
|
|                               |                      |              MIG M.
|=====+=====+
+=====+
|    0   NVIDIA L40S                     On      | 00000000:30:00.0 Off  |
|
| N/A    27C    P8              23W / 350W |      0MiB / 46068MiB |      0%      Default
|
```

```

| | | N/A
|
+-----+-----+
+-----+

+-----+
+
| Processes:
|
| GPU    GI    CI          PID    Type    Process name                  GPU Memory
|
|          ID    ID
|
|
|=====|
| No running processes found
|
+-----+
+

```

出力を見ると、コンテナは NVIDIA GPU 搭載のインスタンスで実行されていること、デバイスドライバは Amazon EKS Auto Mode によって管理されるためインストールする必要がなかったことを確認できます。

ステップ 3: クリーンアップする

作成されたすべてのオブジェクトを削除するには、`kubectl` を使用してサンプルのデプロイと NodePool を削除します。これでノードが終了します。

```

kubectl delete -f nodepool-gpu.yaml
kubectl delete -f pod.yaml

```

NodePool 参照の例

NVIDIA NodePool を作成する

NodePool で定義する内容は次のとおりです。

- g6e と g6 ファミリーのインスタンスを開始するのみ
- 1 時間が過ぎても空のままであればノードを統合する

- `consolidateAfter` が 1 時間というのは、ワークロードの急増に対応し、ノードのチャーンを抑制するのに十分な値です。`consolidateAfter` はワークロード要件に基づいて調整できます。

NodePool を GPU インスタンスファミリーおよび統合で使用する例

```
apiVersion: karpenter.sh/v1
kind: NodePool
metadata:
  name: gpu
spec:
  disruption:
    budgets:
      - nodes: 10%
    consolidateAfter: 1h
    consolidationPolicy: WhenEmpty
  template:
    metadata: {}
    spec:
      nodeClassRef:
        group: eks.amazonaws.com
        kind: NodeClass
        name: default
      requirements:
        - key: "karpenter.sh/capacity-type"
          operator: In
          values: ["on-demand"]
        - key: "kubernetes.io/arch"
          operator: In
          values: ["amd64"]
        - key: "eks.amazonaws.com/instance-family"
          operator: In
          values:
            - g6e
            - g6
    terminationGracePeriod: 24h0m0s
```

`eks.amazonaws.com/instance-gpu-name` を設定するのではなく、`eks.amazonaws.com/instance-family` を使用してインスタンスファミリーを指定することもできます。この他にスケジュールのレビューに影響を与えるラベルとしてよく知られたものについては、「[the section called “EKS Auto Mode でサポートされているラベル”](#)」を参照してください。

固有のストレージ要件がある場合は、NodePool で参照する [NodeClass](#) を独自に作成することで、ノードのエフェメラルストレージの iops、size、throughput を調整できます。[設定可能な NodeClass オプション](#) について確認してください。

NodeClass 用にストレージを設定する例

```
apiVersion: eks.amazonaws.com/v1
kind: NodeClass
metadata:
  name: gpu
spec:
  ephemeralStorage:
    iops: 3000
    size: 80Gi
    throughput: 125
```

AWS Trainium と AWS Inferentia の NodePool を定義する

以下の NodePool では、Inferentia と Trainium ファミリーのインスタンスを開始するだけという `eks.amazonaws.com/instance-category` 設定を定義しています。

```
- key: "eks.amazonaws.com/instance-category"
  operator: In
  values:
    - inf
    - trn
```

kubectl debug を使用して Kubernetes ノードの CIS コンプライアンスレポートを生成する

このトピックでは、`kubectl debug` コマンドを使用して Amazon EKS ノードの CIS (Center for Internet Security) コンプライアンスレポートを生成する方法について説明します。このコマンドを使用すると、Kubernetes ノードにデバッグコンテナを一時的に作成し、`apiclient` ツールを使用して CIS コンプライアンスチェックを実行できます。この `apiclient` ツールは、Bottlerocket OS (EKS Auto Mode ノードで使用する OS) に組み込まれています。

前提条件

開始する前に、以下を確認してください。

- `kubectl` が設定された Amazon EKS クラスターへのアクセス (バージョンは v1.32.0 以上である必要があります。確認するには `kubectl version` と入力します)。
- ノードをデバッグするための適切な IAM アクセス権限。
- デバッグオペレーションが許可される有効なプロファイル (例: `sysadmin`)。

`kubectl` とデバッグプロファイルの併用方法については、Kubernetes ドキュメントの「[Debugging a Pod or Node while applying a profile](#)」で詳しく確認できます。

手順

1. レポートを実行するノードの AWS インスタンス ID を決定します。クラスター内のノードを一覧表示するには、以下のコマンドを使用します。インスタンス ID は名前の列にあり、`i-` で始まります。

```
kubectl get nodes
```

NAME	STATUS	ROLES	AGE	VERSION
i-0ea0ba0f8ef9ad609	Ready	<none>	62s	v1.30.10-eks-1a9dacd

2. `<instance-id>` を、クエリするノードのインスタンス ID に置き換えて、次のコマンドを実行します。

```
kubectl debug node/<instance-id> -it --profile=sysadmin --image=public.ecr.aws/amazonlinux/amazonlinux:2023 -- bash -c "yum install -q -y util-linux-core; nsenter -t 1 -m apiclient report cis --level 1 --format text"
```

このコマンドのコンポーネントは次のとおりです。

- `kubectl debug node/<instance-id>` — 指定した EC2 インスタンス ID でデバッグセッションを開始します。
- `-it` — TTY (コマンドラインシェル) を割り当てて、`stdin` を引いたままにし、インタラクティブに使用できるようにします。
- `--profile=sysadmin` — 指定した `kubectl` プロファイル (適切なアクセス権限が付与されている) を使用します。
- `--image=public.ecr.aws/amazonlinux/amazonlinux:2023` — `amazonlinux:2023` を、デバッグ用のコンテナイメージに使用します。
- `bash -c "..."` — `bash` シェルで次のコマンドを実行します。

- `yum install -q -y util-linux-core` — 必要なユーティリティパッケージをサイレントインストールします。
- `nsenter -t 1 -m` — `nsenter` を実行して、ホストプロセス (PID 1) の名前空間に入ります。
- `apiclient report cis --level 1 --format text` — テキスト出力を使用して、レベル 1 で CIS コンプライアンスレポートを実行します。

3. レポートのテキスト出力を確認します。

出力の解釈

このコマンドにより、テキストベースのレポートを生成して、さまざまな CIS コントロールのコンプライアンスステータスを確認できます。以下のような情報が出力されます。

- 個々の CIS コントロール ID
- 各コントロールの説明
- 各チェックの合格、失敗、スキップのステータス
- コンプライアンス上の問題の詳細説明

Bottlerocket インスタンスで実行したレポートの出力例を次に示します。

```
Benchmark name:  CIS Bottlerocket Benchmark
Version:         v1.0.0
Reference:       https://www.cisecurity.org/benchmark/bottlerocket
Benchmark level: 1
Start time:      2025-04-11T01:40:39.055623436Z

[SKIP] 1.2.1      Ensure software update repositories are configured (Manual)
[PASS] 1.3.1      Ensure dm-verity is configured (Automatic)[PASS] 1.4.1      Ensure
setuid programs do not create core dumps (Automatic)
[PASS] 1.4.2      Ensure address space layout randomization (ASLR) is enabled
(Automatic)
[PASS] 1.4.3      Ensure unprivileged eBPF is disabled (Automatic)
[PASS] 1.5.1      Ensure SELinux is configured (Automatic)
[SKIP] 1.6        Ensure updates, patches, and additional security software are
installed (Manual)
[PASS] 2.1.1.1    Ensure chrony is configured (Automatic)
[PASS] 3.2.5      Ensure broadcast ICMP requests are ignored (Automatic)
[PASS] 3.2.6      Ensure bogus ICMP responses are ignored (Automatic)
```

```
[PASS] 3.2.7      Ensure TCP SYN Cookies is enabled (Automatic)
[SKIP] 3.4.1.3    Ensure IPv4 outbound and established connections are configured
                (Manual)
[SKIP] 3.4.2.3    Ensure IPv6 outbound and established connections are configured
                (Manual)
[PASS] 4.1.1.1    Ensure journald is configured to write logs to persistent disk
                (Automatic)
[PASS] 4.1.2      Ensure permissions on journal files are configured (Automatic)

Passed:          11
Failed:           0
Skipped:          4
Total checks:    15
```

ベンチマークの詳細については、Center for Internet Security (CIS) の「[Kubernetes Benchmark](#)」を参照してください。

関連リソース

- 「[Bottlerocket CIS Benchmark](#)」(Bottlerocket OS ドキュメント)
- 「[実行中のポッドのデバッグ](#)」(Kubernetes ドキュメント)
- 「[Kubernetes Benchmark](#)」(Center for Internet Security (CIS))

EKS オートモードでカスターマネージド KMS キー使用した EBS ボリュームの暗号化を有効にする

カスターマネージド KMS キーを使用すると、EKS オートモードのインスタンスのエフェメラルルートボリュームを暗号化することができます。

Amazon EKS オートモードは、Kubernetes クラスターの暗号化された EBS ボリュームを管理するときに、サービスリンクロールを使用して他の AWS サービスにアクセス許可を委任します。このトピックでは、EKS オートモードの Amazon EBS 暗号化のカスターマネージドキーを指定するときに必要になる、キーポリシーの設定方法について解説します。

考慮事項:

- EKS オートモードでは、デフォルトの AWS マネージドキーを使用してアカウントにある暗号化されたボリュームを保護する場合、追加の承認は不要です。

- このトピックでは、EC2 インスタンスのルートボリュームである、エフェメラルボリュームを暗号化する方法について取り上げます。ワークロードに使用するデータボリュームの暗号化の方法については、「[the section called “StorageClass を作成する”](#)」を参照してください。

概要

以下の AWS KMS キーは、EKS オートモードでインスタンスを起動するときに、Amazon EBS ルートボリュームの暗号化に使用できます。

- AWS マネージドキー – Amazon EBS が作成、所有、管理するアカウント内の暗号化キーです。これは、新しいアカウントのデフォルトの暗号化キーです。
- カスタマーマネージド型キー – お客様が作成、所有、および管理するカスタム暗号化キー。

Note

KMS キーは対称である必要があります。Amazon EBS は非対称カスタマー管理キーをサポートしていません。

ステップ 1: キーポリシーを設定する

KMS キーには、カスタマーマネージドキーで暗号化された Amazon EBS ボリュームを使用してインスタンスを起動することを EKS オートモードに許可する、キーポリシーが必要です。

キーポリシーは次の構造で設定します。

Note

このポリシーには EKS オートモードのアクセス許可しか含まれていません。他の ID でキーを使用したり許可を管理したりする必要がある場合、このキーポリシーに追加のアクセス許可が必要になる可能性があります。

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

```

```
{
  "Sid": "Allow use of the key",
  "Effect": "Allow",
  "Principal": {
    "AWS": [
      "arn:aws:iam::<account-id>:role/ClusterServiceRole"
    ]
  },
  "Action": [
    "kms:Encrypt",
    "kms:Decrypt",
    "kms:ReEncrypt*",
    "kms:GenerateDataKey*",
    "kms:DescribeKey"
  ],
  "Resource": "*"
},
{
  "Sid": "Allow attachment of persistent resources",
  "Effect": "Allow",
  "Principal": {
    "AWS": [
      "arn:aws:iam::<account-id>:role/ClusterServiceRole"
    ]
  },
  "Action": [
    "kms:CreateGrant",
    "kms:ListGrants",
    "kms:RevokeGrant"
  ],
  "Resource": "*",
  "Condition": {
    "Bool": {
      "kms:GrantIsForAWSResource": "true"
    }
  }
}
]
```

<account-id> を、現在お使いの AWS アカウント ID に置き換えます。

キーポリシーを設定する場合:

- ClusterServiceRole には、暗号化のオペレーションに KMS キーを使用するために必要な、IAM アクセス許可が含まれている必要があります。
- kms:GrantIsForAWSResource 条件によって、AWS サービスに対してのみ許可を作成することができます。

ステップ 2: カスタマーマネージドキーを使用して NodeClass を設定する

キーポリシーを設定したら、EKS オートモード NodeClass 設定で KMS キーを参照します。

```
apiVersion: eks.amazonaws.com/v1
kind: NodeClass
metadata:
  name: my-node-class
spec:
  # Insert existing configuration

  ephemeralStorage:
    size: "80Gi" # Range: 1-59000Gi or 1-64000G or 1-58Ti or 1-64T
    iops: 3000 # Range: 3000-16000
    throughput: 125 # Range: 125-1000

    # KMS key for encryption
    kmsKeyID: "arn:aws:kms:<region>:<account-id>:key/<key-id>"
```

プレースホルダーの値を実際の値にそれぞれ置き換えます。

- <region> は現在の AWS リージョンに
- <account-id> はご自身の AWS アカウント ID に
- <key-id> はご自身の KMS キー ID に

KMS キーは以下のいずれかの形式を使用して指定することができます。

- KMS キー ID: 1a2b3c4d-5e6f-1a2b-3c4d-5e6f1a2b3c4d
- KMS キー ARN: arn:aws:kms:us-west-2:111122223333:key/1a2b3c4d-5e6f-1a2b-3c4d-5e6f1a2b3c4d
- キーエイリアス名: alias/eks-auto-mode-key
- キーエイリアス ARN: arn:aws:kms:us-west-2:111122223333:alias/eks-auto-mode-key

kubectl を使用して NodeClass 設定を適用します。

```
kubectl apply -f nodeclass.yaml
```

関連リソース

- [Amazon EKS のノードクラスを作成する](#)
- 詳細については、「AWS Key Management Service デベロッパーガイド」を参照してください。
 - [キーポリシーにおける AWS サービスのアクセス許可](#)
 - [キーポリシーを変更する](#)
 - [Grants in AWS KMS](#)

EKS Auto Mode の仕組みの説明

この章では、Amazon EKS Auto Mode クラスターのコンポーネントの仕組みについて説明します。

トピック

- [Amazon EKS 自動モードI マネージドインスタンスについての説明](#)
- [EKS Auto Mode での ID とアクセスについての説明](#)
- [EKS 自動モードI での VPC ネットワーキングとロードバランシングの説明](#)

Amazon EKS 自動モードI マネージドインスタンスについての説明

このトピックではAmazon EKS 自動モードI が EKS クラスター内の Amazon EC2 インスタンスを管理する方法について説明します。EKS 自動モードI を有効にすると、クラスターのコンピューティングリソースが EKS によって自動的にプロビジョニングおよび管理され、クラスター内のノードとして機能する EC2 インスタンスの操作方法が変わります。

Amazon EKS 自動モードI がインスタンスを管理する方法を理解することはワークロードのデプロイ戦略と運用手順を計画するのに不可欠です。従来の EC2 インスタンスやマネージドノードグループとは異なり、これらのインスタンスは特定のタイプのアクセスとカスタマイズを制限しながら、EKS が多くの運用面に対して責任を負う別のライフサイクルモデルに従います。

Amazon EKS 自動モードI は新しい EC2 インスタンスを作成するためのルーチンタスクを自動化し、それらをノードとして EKS クラスターにアタッチします。EKS 自動モードI はワークロードが既存のノードに収まらない状況を検出し、新しい EC2 インスタンスを作成します。

Amazon EKS 自動モードI はEC2 インスタンスの作成、削除、パッチ適用を担います。インスタンスにデプロイされたコンテナとポッドについてはユーザーの責任となります。

EKS 自動モードI によって作成された EC2 インスタンスは他の EC2 インスタンスとは異なり、マネージドインスタンスです。このマネージドインスタンスは EKS によって所有され、制限が厳しくなっています。EKS 自動モードI によって管理されているインスタンスに直接アクセスしたりソフトウェアをインストールしたりすることはできません。

AWS ではEKS 自動モードI またはセルフマネージド Karpenter の実行をお勧めします。移行中でも高度な設定においてでもインストールが可能です。両方がインストールされている場合はワークロードが Karpenter または EKS 自動モードI に関連付けられるようにノードプールを設定します。

詳細については「Amazon EC2 ユーザーガイド」の「[Amazon EC2 管理されたインスタンス](#)」を参照してください。

比較表

スタンダード EC2 インスタンス	EKS 自動モードI マネージドインスタンス
インスタンスのパッチ適用と更新はユーザーの責任となります。	AWS で自動的にインスタンスのパッチ適用と更新が行われます。
EKS はインスタンス上のソフトウェアに関与しません。	EKS はkubelet、コンテナランタイム、オペレーティングシステムなど、インスタンス上の特定のソフトウェアに関与します。
EC2 API を使用して EC2 インスタンスを削除できます。	EKS によりアカウントにデプロイされたインスタンスの数が決定されます。ワークロードを削除すると、EKS によりアカウント内のインスタンスの数が減ります。
SSH を使用して EC2 インスタンスにアクセスできます。	ポッドとコンテナをマネージドインスタンスにデプロイできます。
ユーザーがオペレーティングシステムとイメージ (AMI) を決定します。	AWS によりオペレーティングシステムとイメージが決定されます。
Windows または Ubuntu の機能に依存するワークロードをデプロイできます。	Linux に基づいてコンテナをデプロイできますが、特定の OS 依存関係はありません。

スタンダード EC2 インスタンス	EKS 自動モードI マネージドインスタンス
ユーザーが起動するインスタンスタイプとファミリーを決定します。	AWS により起動するインスタンスタイプとファミリーが決定されます。ノードプールを使用して、EKS 自動モードI が選択するインスタンスタイプを制限できます。

次の機能はマネージドインスタンスとスタンダード EC2 インスタンスの両方で機能します：

- インスタンスは AWS コンソールで表示できます。
- インスタンスストレージはワークロードのエフェメラルストレージとして使用できます。

AMI サポート

EKS Auto Mode では、AWS がコンピューティングノードに使用されるイメージ (AMI) を決定します。AWS は新しい EKS Auto Mode AMI バージョンのロールアウトをモニタリングします。AMI バージョンに関連するワークロードの問題が発生した場合は、サポートケースを作成します。詳細については、「AWS サポートユーザーガイド」の「[Creating support cases and case management](#)」を参照してください。

EKS の場合、通常、CVE とセキュリティ修正を含む新しい AMI が毎週リリースされます。

EKS Auto Mode でサポートされているインスタンスリファレンス

EKS Auto Mode は、サポートされているタイプの、最小サイズ要件を満たしたインスタンスのみを作成します。

EKS 自動モードI では次のインスタンスタイプがサポートされています：

ファミリー	インスタンスタイプ
コンピューティングの最適化 (C)	c8g、c7a、c7g、c7gn、c7gd、c7i、c7i-flex、c6a、c6g、c6i、c6gn、c6id、c6in、c6gd、c5、c5a、c5d、c5ad、c5n、c4
汎用 (M)	m8g、m7i、m7a、m7g、m7gd、m7i-flex、m6a、m6i、m6in、m6g、m6idn、m6id、m6gd、m5、m5a、m5ad、m5n、m5dn、m5d、m5zn、m4

ファミリー	インスタンスタイプ
メモリ最適化 (R)	r8g、r7a、r7iz、r7gd、r7i、r7g、r6a、r6i、r6id、r6in、r6idn、r6g、r6gd、r5、r5n、r5a、r5dn、r5b、r5ad、r5d、r4
バースト可能 (T)	t4g、t3、t3a、t2
ハイメモリ (Z/X)	z1d、x8g、x2gd
ストレージ最適化 (I/D)	i4g、i4i、i3、i3en、is4gen、d3、d3en、im4gn
高速コンピューティング (P/G/Inf/Trn)	p5、p4d、p3、p3dn、gr6、g6、g6e、g5g、g5、g4dn、inf2、inf1、trn1、trn1n
ハイパフォーマンスコンピューティング (X2)	x2iezn、x2iedn、x2idn

さらに、EKS Auto Mode は、次の要件を満たす EC2 インスタンスのみを作成します。

- 複数の CPU
- インスタンスサイズがナノ、マイクロ、スモールではない

詳細については、「[Amazon EC2 インスタンスタイプの命名規則](#)」を参照してください。

インスタンスメタデータサービス

- EKS Auto Mode では、デフォルトでホップ制限が 1 の IMDSv2 を適用し、AWS セキュリティのベストプラクティスに準拠します
- このデフォルト設定は Auto Mode では変更できません
- 通常 IMDS アクセスを必要とするアドオンの場合
 - IMDS ルックアップを回避するために、インストール中に AWS リージョンなどのパラメータを指定します
 - 詳細については、[the section called “カスタマイズ可能なフィールド”](#)を参照してください。
- Auto Mode で実行されるポッドに IMDS へのアクセスが必須の場合

- ポッドの実行設定で、`hostNetwork: true` を指定する必要があります
- これにより、ポッドは、インスタンスメタデータサービスに直接アクセスできます
- インスタンスメタデータへのアクセスをポッドに許可した場合、セキュリティ上どのような影響があるかを考察します

EC2 インスタンスメタデータサービス (IMDS) の詳細については、「EC2 ユーザーガイド」の「[Configure the Instance Metadata Service options](#)」を参照してください。

考慮事項

- EKS 自動モードはサポートされているインスタンスタイプで NVMe ローカルストレージを自動的にフォーマットして設定します。複数の NVMe ドライブを持つノードの場合、EKS で RAID 0 アレイが設定されます。この自動化により、EKS クラスター内のローカル NVMe ストレージの手動フォーマットと RAID 設定が不要になります。
- Amazon EKS 自動モードは AWS Fault Injection Service をサポートしていません。詳細については「AWS レジリエンスハブ ユーザーガイド」の「[フォールトインジェクションサービスの実験管理](#)」を参照してください。
- EKS 自動モードノードに Neuron Device Plugin をインストールする必要はありません。
 - クラスターに他のタイプのノードがある場合は自動モードノードで実行されないように ニューロンデバイス プラグインを設定する必要があります。詳細については「[the section called “デブロイを制御する”](#)」を参照してください。

EKS Auto Mode での ID とアクセスについての説明

このトピックでは、EKS Auto Mode を使用するのに必要な Identity and Access Management (IAM) ロールとアクセス許可について説明します。EKS Auto Mode は、クラスター IAM ロールとノード IAM ロールという 2 つのプライマリ IAM ロールを使用します。これらのロールは、EKS Pod Identity および EKS アクセスエントリと連携して動作し、EKS クラスターの包括的なアクセス管理を提供します。

EKS Auto Mode を設定するときには、AWS サービスがクラスターリソースとやり取りできるようにする特定のアクセス許可を持つこれらの IAM ロールを設定する必要があります。これには、コンピューティングリソース、ストレージボリューム、ロードバランサー、ネットワーキングコンポーネントを管理するためのアクセス許可が含まれます。これらのロール設定を理解することは、適切なクラスターオペレーションおよびセキュリティに不可欠です。

EKS Auto Mode では、AWS IAM ロールは EKS アクセスエントリを介して Kubernetes アクセス許可に自動的にマッピングされるため、aws-auth ConfigMaps またはカスタムバインディングを手動で設定する必要はありません。新しい Auto Mode クラスターを作成すると、EKS はアクセスエントリを使用して対応する Kubernetes アクセス許可を自動的に作成し、AWS サービスとクラスターコンポーネントが AWS と Kubernetes の両方の認証システムで適切なアクセスレベルになるようにします。この自動統合により、設定の複雑さが軽減され、EKS クラスターの管理時によく発生するアクセス許可関連の問題を防ぐことができます。

クラスター IAM ロール

クラスター IAM ロールは、Kubernetes クラスターのアクセス許可を管理するために Amazon EKS が使用する AWS Identity and Access Management (IAM) ロールです。このロールは、クラスターに代わって他の AWS サービスとやり取りするのに必要なアクセス許可を Amazon EKS に付与し、EKS アクセスエントリにより Kubernetes アクセス許可が付与されて自動的に設定されます。

- AWS IAM ポリシーをこのロールにアタッチする必要があります。
- EKS Auto Mode は、EKS アクセスエントリを使用して、このロールに Kubernetes アクセス許可を自動的にアタッチします。
- EKS Auto Mode を使用する場合、AWS では AWS アカウントごとに 1 つのクラスター IAM ロールを作成することをお勧めします。
- AWS では、このロールの名前を AmazonEKSAutoClusterRole とすることをお勧めします。
- このロールには、EBS ボリューム、Elastic Load Balancer、EC2 インスタンスなどのリソースを管理するために、複数の AWS サービスに対するアクセス許可が必要です。
- このロールの推奨設定には、EKS Auto Mode のさまざまな機能に関連する複数の AWS マネージド IAM ポリシーが含まれます。
 - AmazonEKSCoordinatePolicy
 - AmazonEKSBLOCKStoragePolicy
 - AmazonEKSLoadBalancingPolicy
 - AmazonEKSNetworkingPolicy
 - AmazonEKSClusterPolicy

クラスター IAM ロールと AWS マネージド IAM ポリシーの詳細については、次を参照してください。

- [the section called “AWS 管理ポリシー”](#)

- [the section called “クラスター IAM ロール”](#)

Kubernetes アクセスの詳細については、次を参照してください。

- [the section called “アクセスポリシーを確認する”](#)

ノード の IAM ロール

ノード IAM ロールは、Kubernetes クラスター内のワーカーノードのアクセス許可を管理するために Amazon EKS が使用する AWS Identity and Access Management (IAM) ロールです。このロールは、Kubernetes ノードとして実行されている EC2 インスタンスに、AWS サービスおよびリソースを操作するのに必要なアクセス許可を付与し、EKS アクセスエントリにより Kubernetes RBAC アクセス許可が付与されて自動的に設定されます。

- AWS IAM ポリシーをこのロールにアタッチする必要があります。
- EKS Auto Mode は、EKS アクセスエントリを使用して、このロールに Kubernetes RBAC アクセス許可を自動的にアタッチします。
- AWS では、このロールの名前を AmazonEKSAutoNodeRole とすることをお勧めします。
- EKS Auto Mode を使用する場合、AWS では AWS アカウントごとに 1 つのノード IAM ロールを作成することをお勧めします。
- このロールには制限されたアクセス許可があります。その主なアクセス許可には、Pod Identity ロールの引き受け、ECR からのイメージのプルなどがあります。
- AWS では、以下の AWS マネージド IAM ポリシーをお勧めします。
 - AmazonEKSWorkerNodeMinimalPolicy
 - AmazonEC2ContainerRegistryPullOnly

クラスター IAM ロールと AWS マネージド IAM ポリシーの詳細については、次を参照してください。

- [the section called “AWS 管理ポリシー”](#)
- [the section called “ノード の IAM ロール”](#)

Kubernetes アクセスの詳細については、次を参照してください。

- [the section called “アクセスポリシーを確認する”](#)

サービスリンクロール

Amazon EKS は、特定のオペレーションにサービスリンクロール (SLR) を使用します。サービスにリンクされた役割はアマゾン EKS に直接リンクされた一意のタイプの IAM 役割です。サービスにリンクされた役割はアマゾン EKS で事前定義されています。この役割にはサービスがユーザーに代わって他の AWS のサービス呼び出すために必要な、すべてのアクセス許可が付与されています。

AWS では SLR が自動的に作成および設定されます。先に関連リソースを削除した後にのみ、SLR を削除できます。これにより、リソースへのアクセス許可を不用意に削除することが防止され、アマゾン EKS リソースが保護されます。

SLR ポリシーは、EC2 リソース (インスタンス、ネットワークインターフェイス、セキュリティグループ)、ELB リソース (ロードバランサー、ターゲットグループ)、CloudWatch 機能 (ログ記録とメトリクス)、「eks」というプレフィックスが付いた IAM ロールなどのコアインフラストラクチャコンポーネントを監視および削除する Amazon EKS アクセス許可を付与します。また、VPC/ホストゾーンの関連付けによるプライベートエンドポイントネットワーキングを有効にし、EKS タグ付きリソースの EventBridge モニタリングとクリーンアップのためのアクセス許可を備えています。

詳細については、以下を参照してください。

- [the section called “AWS 管理ポリシー: AmazonEKSServiceRolePolicy”](#)
- [the section called “アマゾン EKS でのサービスにリンクされた役割のアクセス許可”](#)

EKS Auto リソースのカスタム AWS タグ

デフォルトでは、EKS Auto Mode に関連するマネージドポリシーでは、ユーザー定義タグを Auto Mode でプロビジョニングされた AWS リソースに適用することはできません。ユーザー定義タグを AWS リソースに適用する場合は、AWS リソースのタグを作成および変更するのに十分なアクセス許可を持つクラスター IAM ロールに追加のアクセス許可をアタッチする必要があります。以下は、無制限のタグ付けアクセスを許可するポリシーの例です。

カスタムタグポリシーの例を表示する

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "Compute",
      "Effect": "Allow",
      "Action": [
```

```

        "ec2:CreateFleet",
        "ec2:RunInstances",
        "ec2:CreateLaunchTemplate"
    ],
    "Resource": "*",
    "Condition": {
        "StringEquals": {
            "aws:RequestTag/eks:eks-cluster-name": "${aws:PrincipalTag/eks:eks-
cluster-name}"
        },
        "StringLike": {
            "aws:RequestTag/eks:kubernetes-node-class-name": "*",
            "aws:RequestTag/eks:kubernetes-node-pool-name": "*"
        }
    }
},
{
    "Sid": "Storage",
    "Effect": "Allow",
    "Action": [
        "ec2:CreateVolume",
        "ec2:CreateSnapshot"
    ],
    "Resource": [
        "arn:aws:ec2:*:*:volume/*",
        "arn:aws:ec2:*:*:snapshot/*"
    ],
    "Condition": {
        "StringEquals": {
            "aws:RequestTag/eks:eks-cluster-name": "${aws:PrincipalTag/eks:eks-
cluster-name}"
        }
    }
},
{
    "Sid": "Networking",
    "Effect": "Allow",
    "Action": "ec2:CreateNetworkInterface",
    "Resource": "*",
    "Condition": {
        "StringEquals": {
            "aws:RequestTag/eks:eks-cluster-name": "${aws:PrincipalTag/eks:eks-
cluster-name}"
        }
    }
},

```

```

        "StringLike": {
            "aws:RequestTag/eks:kubernetes-cni-node-name": "*"
        }
    },
    {
        "Sid": "LoadBalancer",
        "Effect": "Allow",
        "Action": [
            "elasticloadbalancing:CreateLoadBalancer",
            "elasticloadbalancing:CreateTargetGroup",
            "elasticloadbalancing:CreateListener",
            "elasticloadbalancing:CreateRule",
            "ec2:CreateSecurityGroup"
        ],
        "Resource": "*",
        "Condition": {
            "StringEquals": {
                "aws:RequestTag/eks:eks-cluster-name": "${aws:PrincipalTag/eks:eks-cluster-name}"
            }
        }
    },
    {
        "Sid": "ShieldProtection",
        "Effect": "Allow",
        "Action": [
            "shield:CreateProtection"
        ],
        "Resource": "*",
        "Condition": {
            "StringEquals": {
                "aws:RequestTag/eks:eks-cluster-name": "${aws:PrincipalTag/eks:eks-cluster-name}"
            }
        }
    },
    {
        "Sid": "ShieldTagResource",
        "Effect": "Allow",
        "Action": [
            "shield:TagResource"
        ],
        "Resource": "arn:aws:shield:*:*:protection/*",
    }
}

```

```
        "Condition": {
            "StringEquals": {
                "aws:RequestTag/eks:eks-cluster-name": "${aws:PrincipalTag/eks:eks-cluster-name}"
            }
        }
    }
}
```

アクセスポリシーのリファレンス

EKS Auto Mode で使用される Kubernetes アクセス許可の詳細については、「[the section called “アクセスポリシーを確認する”](#)」を参照してください。

EKS 自動モードIでの VPC ネットワーキングとロードバランシングの説明

このトピックではEKS 自動モードIで Virtual Private クラウド (VPC ネットワーキングおよびロードバランシング機能を設定する方法について説明します。EKS Auto Mode はほとんどのネットワークコンポーネントを自動的に管理しますが、NodeClass リソースとロードバランサーの注釈を使用してクラスターのネットワーキング設定の特定の側面をカスタマイズできます。

EKS 自動モードIを使用すると、AWS でクラスターの VPC Container Network Interface (CNI 設定とロードバランサーのプロビジョニングが管理されます。NodeClass オブジェクトを定義し、サービスリソースとインGRESSリソースに特定の注釈を適用することで、EKS Auto Mode が提供する自動運用モデルを維持しながら、ネットワーキング動作に参与することができます。

ネットワーク機能

EKS Auto Mode には、ノードとポッドのネットワークを処理する新しいネットワーク機能があります。この機能は、NodeClass Kubernetes オブジェクトを作成することで設定できます。

以前の AWS VPC CNI の設定オプションは EKS Auto Mode には適用されません。

NodeClass を使用したネットワークの設定

EKS Auto Mode の NodeClass リソースを使用すると、ネットワーク機能の特定の側面をカスタマイズできます。NodeClass を使用すると、セキュリティグループ選択の指定、VPC サブネット間のノード配置の制御、SNAT ポリシーの設定、ネットワークポリシーの設定、ネットワークイベントのログ記録の有効化を行うことができます。このアプローチはEKS 自動モードIの自動運用モデルを維持しながら、ネットワークのカスタマイズで柔軟性が得られます。

NodeClass を使用して以下のことができます。

- ノードのセキュリティグループを選択する
- VPC サブネットにノードを配置する方法を制御する
- ノード SNAT ポリシーを random または disabled に設定する
- 以下を含む Kubernetes ネットワークポリシーを有効にする。
 - ネットワークポリシーを「デフォルトで拒否」または「デフォルトで許可」に設定する
 - ファイルへのネットワークイベントのログ記録を有効にする。
- 異なるサブネットにポッドをアタッチして、ノードトラフィックからポッドトラフィックを分離する。

[Amazon EKS ノードクラス の作成](#)方法を参照してください。

考慮事項

EKS 自動モードは以下をサポートしています：

- EKS ネットワークポリシー。
- Kubernetes ポッドの HostPort および HostNetwork オプション。
- パブリックサブネットまたはプライベートサブネットのノードとポッド。
- ノードでの DNS クエリキャッシュ。

EKS 自動モードは以下をサポートしていません：

- ポッドあたりのセキュリティグループ (SGPP)。
- ENIConfig のカスタムネットワーキング。ポッドを複数のサブネットに配置することも、[the section called “ポッドのサブネット選択”](#) を使用してノードトラフィックから排他的に分離することもできます。
- ウォーム IP、ウォームプレフィックス、ウォーム ENI 設定。
- 最小 IP ターゲット設定。
- プレフィックス委任の有効化または無効化。
- オープンソース AWS VPC CNI でサポートされているその他の設定。
- conntrack タイマーのカスタマイズなどのネットワークポリシー設定 (デフォルトは 300 秒)。
- CloudWatch へのネットワークイベントログのエクスポート。

ネットワークリソース管理

EKS Auto Mode では、NodeClass リソースのネットワーク設定をモニタリングすることで、プレフィックス、IP アドレス指定、ネットワークインターフェイス管理を処理します。このサービスでは、次のいくつかの主要なオペレーションが自動的に実行されます。

プレフィックスの委任

EKS Auto Mode では、ノードのプライマリネットワークインターフェイスに /28 IPv4 プレフィックスをプロビジョニングして、スケジュール済みポッドの数を基に、事前に定義したリソースウォームプールを維持します。また、必要に応じて、ノードのサブネット内のプライマリインターフェイスと同じセキュリティグループを持つセカンダリネットワークインターフェイスをプロビジョニングします。サブネットでプレフィックスが使用できなくなると、セカンダリ IPv4 アドレスにフォールバックします。

クールダウン管理

このサービスでは、使用されなくなったプレフィックスまたはセカンダリ IPv4 アドレスのクールダウンプールを実装します。クールダウン期間が終了すると、これらのリソースは VPC に解放されます。ただし、こうしたリソースは、クールダウン期間にポッドで再利用されると、クールダウンプールから復元されます。

IPv6 のサポート

IPv6 クラスターの EKS Auto Mode では、プライマリネットワークインターフェイスのノードごとに /80 IPv6 プレフィックスをプロビジョニングします。

また、すべてのネットワークインターフェイスの適切な管理とガベージコレクションを確実に行います。

負荷分散

EKS 自動モードI でプロビジョニングされた AWS 弾性ロードバランサー はサービスリソースとイングレスリソースの注釈を使用して設定します。

詳細については「[the section called “ingress class の作成”](#)」または「[the section called “サービスの作成”](#)」を参照してください。

EKS 自動モードI によるロードバランシングに関する考慮事項

- デフォルトのターゲットモードは IP モードであり、インスタンスモードではありません。

- EKS 自動モードI はネットワークロードバランサー のセキュリティグループモードのみをサポートします。
- AWS はセルフマネージド AWS ロードバランサーコントローラー から EKS 自動モードI による管理へのロードバランサーの移行をサポートしていません。
- TargetGroupBinding 仕様の `networking.ingress.ipBlock` フィールドはサポートされていません。
- ワーカーノードがカスタムセキュリティグループ (`eks-cluster-sg-` 命名パターンではない) を使用している場合、クラスターロールには追加の IAM アクセス許可が必要です。デフォルトの EKS マネージドポリシーではEKS による `eks-cluster-sg-` という名前のセキュリティグループの変更のみを許可します。カスタムセキュリティグループを変更する許可がないと、EKS は ALB/NLB トラフィックがポッドに到達できるようにするのに必要な進入ルールを追加できません。

EKS 自動モードI のトラブルシューティング

EKS 自動モードI ではAWS が AWS アカウント内の EC2 インスタンスに対してより多くの責任を負います。EKS はノード上のコンテナランタイム、ノード上のオペレーティングシステム、および特定のコントローラーに対して責任を負います。これにはブロックストレージコントローラー、ロードバランシングコントローラー、コンピューティングコントローラーなどがあります。

ノードのトラブルシューティングにはAWS API および Kubernetes API を使用する必要があります。次のようにできます：

- Kubernetes NodeDiagnostic リソースを使用し、[the section called “ノードモニタリングエージェント”](#)を使用することでノードログを取得します。詳細な手順については「[the section called “ノードログの取得”](#)」を参照してください。
- AWS EC2 CLI コマンド `get-console-output` を使用して、ノードからコンソール出力を取得します。詳細な手順については「[the section called “AWS EC2 CLI を使用して EC2 マネージドインスタンスからコンソール出力を取得する”](#)」を参照してください。
- Kubernetes デバッグコンテナを使用してノードログを取得します。詳細な手順については「[the section called “デバッグコンテナと kubectl CLI を使用してノードログを取得する”](#)」を参照してください。

Note

EKS 自動モードI は EC2 マネージドインスタンスを使用します。SSH による場合を含め、EC2 マネージドインスタンスに直接アクセスすることはできません。

EKS Auto Mode コンポーネントに固有の解決策がある、次のような問題が発生する可能性があります。

- Auto Mode ノードにスケジュールされていない Pod が Pending 状態でスタックする。解決策については、「[the section called “Auto Mode ノードにスケジュールできない Pod のトラブルシューティング”](#)」を参照してください。
- EC2 マネージドインスタンスが Kubernetes ノードとしてクラスターに参加しない。解決策については、「[the section called “クラスターに参加しないノードのトラブルシューティング”](#)」を参照してください。
- EKS Auto Mode に含まれているコントローラーを使用する NodePools、PersistentVolumes、Services に関するエラーと問題。解決策については、「[the section called “Auto Mode に含まれているコントローラーのトラブルシューティング”](#)」を参照してください。
- 強化された Pod のセキュリティにより、Pod 間でボリュームを共有できない。解決策については、「[the section called “Pod 間でボリュームを共有する”](#)」を参照してください。

EKS Auto Mode コンポーネントのトラブルシューティングには、次の方法を使用できます。

- [the section called “AWS EC2 CLI を使用して EC2 マネージドインスタンスからコンソール出力を取得する”](#)
- [the section called “デバッグコンテナと kubectl CLI を使用してノードログを取得する”](#)
- [the section called “AWS コンソールで EKS 自動モードI に関連付けられたリソースを表示する”](#)
- [the section called “AWS アカウントの IAM エラーを表示します”](#)
- [the section called “VPC Reachability Analyzer でノード接続の問題を検出する”](#)

ノードモニタリングエージェント

EKS 自動モードI には Amazon EKS ノードモニタリングエージェントが備わっています。このエージェントを使用して、ノードに関するトラブルシューティングとデバッグの情報を表示できます。

ノードモニタリングエージェントはKubernetes events とノードの conditions を発行します。詳細については「[the section called “ノードのヘルス”](#)」を参照してください。

AWS EC2 CLI を使用して EC2 マネージドインスタンスからコンソール出力を取得する

この手順はブートタイムまたはカーネルレベルの問題のトラブルシューティングに役立ちます。

まず、ワークロードに関連付けられたインスタンスの EC2 インスタンス ID を決定する必要があります。次に、AWS CLI を使用してコンソール出力を取得します。

1. kubectl がインストールされ、クラスターに接続されていることを確認します
2. (任意 Kubernetes デプロイの名前を使用して、関連するポッドを一覧表示します。

```
kubectl get pods -l app=<deployment-name>
```

3. Kubernetes ポッドの名前を使用して、関連付けられたノードの EC2 インスタンス ID を決定します。

```
kubectl get pod <pod-name> -o wide
```

4. EC2 インスタンス ID を使用してコンソール出力を取得します。

```
aws ec2 get-console-output --instance-id <instance id> --latest --output text
```

デバッグコンテナと kubectl CLI を使用してノードログを取得する

EKS Auto Mode ノードからログを取得するために推奨される方法は、NodeDiagnostic リソースを使用することです。手順については、「[the section called “ノードログの取得”](#)」を参照してください。

ただし、kubectl debug node コマンドを使用してインスタンスからログをライブストリーミングすることができます。このコマンドにより、デバッグするノードで新しい Pod が起動されて、インタラクティブに使用することができます。

1. デバッグコンテナを起動します。次のコマンドは、ノードのインスタンス ID に i-01234567890123456 を使用し、インタラクティブな使用のために -it で tty を割り当てて stdin をアタッチし、kubeconfig ファイルの sysadmin プロファイルを使用します。

```
kubectl debug node/i-01234567890123456 -it --profile=sysadmin --image=public.ecr.aws/amazonlinux/amazonlinux:2023
```

出力例は次のとおりです。

```
Creating debugging pod node-debugger-i-01234567890123456-nxb9c with container
debugger on node i-01234567890123456.
If you don't see a command prompt, try pressing enter.
bash-5.2#
```

2. シェルから、`nsenter` コマンドを提供する `util-linux-core` をインストールできるようになりました。`nsenter` を使用してホスト上の PID 1 (init) のマウント名前空間を入力し、`journalctl` コマンドを実行して `kubelet` からログをストリーミングします。

```
yum install -y util-linux-core
nsenter -t 1 -m journalctl -f -u kubelet
```

セキュリティのため、Amazon Linux コンテナイメージはデフォルトでは多くのバイナリをインストールしません。`yum whatprovides` コマンドを使用して、特定のバイナリを提供するためにインストールする必要があるパッケージを識別できます。

```
yum whatprovides ps
```

```
Last metadata expiration check: 0:03:36 ago on Thu Jan 16 14:49:17 2025.
procps-ng-3.3.17-1.amzn2023.0.2.x86_64 : System and process monitoring utilities
Repo          : @System
Matched from:
Filename      : /usr/bin/ps
Provide       : /bin/ps

procps-ng-3.3.17-1.amzn2023.0.2.x86_64 : System and process monitoring utilities
Repo          : amazonlinux
Matched from:
Filename      : /usr/bin/ps
Provide       : /bin/ps
```

AWS コンソールで EKS 自動モードI に関連付けられたリソースを表示する

AWS コンソールを使用して、EKS 自動モードI クラスターに関連付けられているリソースのステータスを表示できます。

- [EBS ボリューム](#)

- タグキー eks:eks-cluster-name を検索して EKS 自動モードI ボリュームを表示します

- [ロードバランサー](#)

- タグキー eks:eks-cluster-name を検索して EKS 自動モードI ロードバランサーを表示します

- [EC2 インスタンス](#)

- タグキー eks:eks-cluster-name を検索して EKS 自動モードI インスタンスを表示します

AWS アカウントの IAM エラーを表示します

1. クラウドTrail コンソールに移動します
2. 左側のナビゲーションペインで [イベント履歴] を選択します
3. 次のエラーコードフィルターを適用します：
 - アクセス拒否
 - 不正操作
 - 無効なクライアントトークンID

EKS クラスターに関連するエラーを探します。エラーメッセージを使用して、EKS アクセスエントリ、クラスター IAM ロール、またはノード IAM ロールを更新します。EKS Auto Mode に対するアクセス許可が付与された状態で、これらのロールを新しいポリシーにアタッチすることが必要になる場合があります。

Auto Mode ノードにスケジュールできない Pod のトラブルシューティング

ポッドが Pending 状態のままになり、Auto Mode ノードにスケジュールされない場合は、ポッドまたはデプロイマニフェストに nodeSelector が含まれているかどうかを確認します。nodeSelector が存在する場合は、EKS Auto Mode によって作成されたノードでスケジュールされる eks.amazonaws.com/compute-type: auto が使用されていることを確認します。EKS Auto Mode で使用されるノードラベルの詳細については、「[the section called “デプロイを制御する”](#)」を参照してください。

クラスターに参加しないノードのトラブルシューティング

EKS Auto Mode は、クラスターエンドポイントやクラスター証明機関 (CA) などの、クラスターに参加するための正しい情報を使用して新しい EC2 インスタンスを自動的に設定します。しかし、それでもこれらのインスタンスがノードとして EKS クラスターに参加できない場合があります。次のコマンドを実行して、クラスターに参加しなかったインスタンスを特定します。

1. `kubectl get nodeclaim` を実行して、NodeClaims が `Ready = False` であるかどうかを確認します。

```
kubectl get nodeclaim
```

2. `kubectl describe nodeclaim <node_claim>` を実行して、[ステータス] でノードがクラスターに参加できない問題を見つけます。

```
kubectl describe nodeclaim <node_claim>
```

一般的なエラーメッセージ：

Error getting launch template configs

デフォルトのクラスター IAM ロールのアクセス許可を使用して NodeClass でカスタムタグを設定する場合、このエラーが表示されることがあります。「[the section called “ID とアクセス”](#)」を参照してください。

Error creating fleet

EC2 API から `RunInstances` コールを呼び出すと、認可の問題が発生する可能性があります。AWS CloudTrail でエラーを確認し、必要な IAM アクセス許可については「[the section called “Amazon EKS 自動モードI クラスター IAM ロール”](#)」を参照してください。

VPC Reachability Analyzer でノード接続の問題を検出する

Note

VPC Reachability Analyzer を実行する分析ごとに課金されます。料金の詳細については、「[Amazon VPC の料金](#)」を参照してください。

インスタンスがクラスターに参加しない理由の 1 つは、API サーバーに到達できないネットワーク接続の問題です。この問題を診断するために、[VPC Reachability Analyzer](#) を使用して、クラスターに参加できないノードと API サーバー間の接続の分析を実行することができます。次の 2 つの情報が必要になります。

- クラスターに参加できないノードのインスタンス ID
- Kubernetes API サーバーエンドポイントの IP アドレス

インスタンス ID を取得するには、クラスターにワークロードを作成して、EKS Auto Mode で EC2 インスタンスを起動する必要があります。これにより、インスタンス ID を持つ NodeClaim オブジェクトもクラスター内に作成されます。kubectl get nodeclaim -o yaml を実行して、クラスター内のすべての NodeClaims を出力します。各 NodeClaim にはインスタンス ID がフィールドとして含まれ、providerID にも再度含まれています。

```
kubectl get nodeclaim -o yaml
```

出力例は次のとおりです。

```
nodeName: i-01234567890123456
providerID: aws:///us-west-2a/i-01234567890123456
```

Kubernetes API サーバーエンドポイントは、kubectl get endpoint kubernetes -o yaml を実行して判断できます。このアドレスは addresses フィールドにあります。

```
kubectl get endpoints kubernetes -o yaml
```

出力例は次のとおりです。

```
apiVersion: v1
kind: Endpoints
metadata:
  name: kubernetes
  namespace: default
subsets:
- addresses:
  - ip: 10.0.143.233
  - ip: 10.0.152.17
  ports:
  - name: https
```

```
port: 443
protocol: TCP
```

これら 2 つの情報を使用して分析を実行できます。まず、AWS Management Console で VPC Reachability Analyzer に移動します。

1. [パスを作成および分析] をクリックします
2. 分析の名前を指定します (例: 「Node Join Failure」)
3. [ソースタイプ] には [インスタンス] を選択します
4. 障害が発生したノードのインスタンス ID を [ソース] として入力します
5. [パスの送信先] に [IP アドレス] を選択します
6. API サーバーの IP アドレスの 1 つを [送信先アドレス] として入力します
7. [追加のパケットヘッダーの設定セクション] を展開します
8. [送信先ポート] に 443 と入力します
9. まだ選択されていない場合は、[プロトコル] を [TCP] として選択します
10. [パスを作成および分析] をクリックします
11. 分析の完了には数分かかることがあります。分析結果で到達可能性の失敗が示された場合は、ネットワークパスのどこで障害が生じたかが示され、問題を解決することができます。

Pod 間でボリュームを共有する

EKS Auto Mode ノードは、SELinux の強制モードで設定され、同じノードで実行されている Pod 間の分離を強化します。SELinux を有効にすると、特権のないほとんどのポッドに、独自のマルチカテゴリセキュリティ (MCS) ラベルが自動的に適用されます。この MCS ラベルは Pod ごとに一意であり、1 つの Pod のプロセスで他の Pod またはホストのプロセスを操作できないように設計されています。ラベル付き Pod がルートとして実行され、ホストファイルシステムにアクセスできる場合でも、ファイルの操作、ホストでの機密システムの呼び出し、コンテナランタイムへのアクセス、kubelet のシークレットキーマテリアルの取得を行うことはできません。

このため、Pod 間でデータを共有しようとする問題が発生する可能性があります。例えば、アクセスモードが `ReadWriteOnce` の `PersistentVolumeClaim` でも、複数の Pod がボリュームに同時にアクセスすることはできません。

Pod 間でこの共有を有効にするには、Pod の `seLinuxOptions` を使用して、これらの Pod に同じ MCS ラベルを設定します。この例では、Pod に 3 つのカテゴリ (`c123`, `c456`, `c789`) を割り当てま

す。2つのカテゴリが割り当てられるだけであるため、これによりノード上の Pod に自動的に割り当てられたカテゴリとの競合が発生することはありません。

```
securityContext:
  seLinuxOptions:
    level: "s0:c123,c456,c789"
```

Auto Mode に含まれているコントローラーのトラブルシューティング

コントローラーに問題がある場合は以下について調べる必要があります：

- そのコントローラーに関連付けられているリソースが適切にフォーマットされ、有効であるかどうか。
- AWS IAM リソースおよび Kubernetes RBAC リソースがクラスター用に適切に設定されているかどうか。詳細については、「[the section called “ID とアクセス”](#)」を参照してください。

EKS Auto Mode のリリースノートを確認する

このページでは、Amazon EKS Auto Mode の更新について説明します。ここでは、機能、バグ修正、既知の問題、廃止された機能に関するお知らせを定期的に確認できます。

この特定のドキュメントページへのすべてのソースファイルの変更の通知を受け取るには、RSS リーダーを使用して次の URL をサブスクライブできます。

```
https://github.com/awsdocs/amazon-eks-user-guide/commits/mainline/latest/ug/automode/
auto-change.adoc.atom
```

2025 年 6 月 13 日

機能: NodeClass での個別のポッドサブネットのサポート。これにより、オプションのキー `podSubnetSelectorTerms` と `podSecurityGroupSelectorTerms` が追加され、ポッドのサブネットとセキュリティグループが設定されます。詳細については、「[the section called “ポッドのサブネット選択”](#)」を参照してください。

2025 年 4 月 30 日

機能: NodeClass でのフォワードネットワークプロキシのサポート。これにより、HTTPS プロキシを設定するためのオプションキー `advancedNetworking` が追加されます。詳細については、「[the section called “ノードクラスの指定”](#)」を参照してください。

2025 年 4 月 18 日

機能: ユニキャスト DNS を介したローカルドメイン (通常はマルチキャスト DNS 用に予約) の解決をサポートします。

2025 年 4 月 11 日

機能: `certificateBundles` と `ephemeralStorage.kmsKeyID` を `NodeClass` に追加しました。詳細については、「[the section called “ノードクラスの指定”](#)」を参照してください。

機能: イメージのプル速度が向上しました。特に、高速なイメージ解凍を活用できるローカルインスタンスストレージを持つインスタンスタイプで、この効果を得られます。

バグ修正: `FailedCreatePodSandBox` のエラー原因である競合状態を解決しました。エラー詳細: `Error while dialing: dial tcp 127.0.0.1:50051: connect: connection refused to sometimes occur for Pods scheduling to a Node immediately at startup`

2025 年 4 月 4 日

機能: `registryPullQPS` を 5 から 25、`registryBurst` を 10 から 50 に増やして、クライアントで強制されるイメージプルスロットリングを減らします (`Failed to pull image xyz: pull QPS exceeded`)

2025 年 3 月 31 日

バグ修正: CoreDNS ポッドを Auto Mode ノードで実行している場合に、ノードのポッドからの DNS クエリがノードローカル DNS サーバーではなくその CoreDNS ポッドにヒットする問題を修正しました。Auto Mode ノードのポッドからの DNS クエリは、常にノードローカル DNS に送信されます。

2025 年 3 月 21 日

バグ修正: Auto Mode ノードは、クラスターに `kube-dns` サービスがインストールされていない場合にも `kube-dns.kube-system.svc.cluster.local` が正しく解決されるようになりました。GitHub の問題 [#2546](#) に対処します。

2025 年 3 月 14 日

機能: IPv6 クラスターで IPv4 egress が有効になりました。IPv6 Auto Mode クラスターから送信される IPv4 トラフィックは、ノードプライマリ ENI の v4 アドレスに自動的に変換されるようになりました。

Amazon EKS クラスターを使用してワークロードを整理する

Amazon EKS クラスターは、主要な 2 つのコンポーネントで構成されています。

- Amazon EKS コントロールプレーン
- コントロールプレーンに登録された Amazon EKS ノード

Amazon EKS コントロールプレーンは、Kubernetes ソフトウェア (etcd や Kubernetes API サーバーなど) を実行するコントロールプレーンノードで構成されています。コントロールプレーンは AWS によって管理されるアカウントで実行されます。また、Kubernetes API はクラスターに関連付けられている Amazon EKS エンドポイント経由で公開されています。各 Amazon EKS クラスターのコントロールプレーンは、一意かつシングルテナントであり、固有の Amazon EC2 インスタンスセット上で実行されます。

etcd ノードおよび関連する Amazon EBS ボリュームによって格納されているすべてのデータに対しては、AWS KMS を使用した暗号化が行われます。クラスターコントロールプレーンは、複数のアベイラビリティゾーンに渡ってプロビジョニングされ、Elastic Load Balancing Network Load Balancer によって前面に置かれます。また、Amazon EKS は、VPC サブネット内に Elastic Network Interface もプロビジョニングします。これにより、コントロールプレーンインスタンスからノードへの接続が可能になります (例えば、`kubectl exec logs proxy` のデータフローをサポートします)。

Important

Amazon EKS 環境では、[アップストリーム](#)ガイダンスのとおり、etcd ストレージは 8 GiB に制限されます。次のコマンドを実行して、現在のデータベースサイズのメトリックを監視できます。クラスターの Kubernetes バージョンが 1.28 未満の場合、`apiserver_storage_size_bytes` を `apiserver_storage_db_total_size_in_bytes` に置き換えます。

```
kubectl get --raw=/metrics | grep "apiserver_storage_size_bytes"
```

Amazon EKS ノードは、ユーザーの AWS アカウントで実行されます。クラスターのコントロールプレーンへの接続は、API サーバーエンドポイントや、クラスターのために作成された証明書ファイル経由で行われます。

Note

- 「[Amazon EKS クラスターのネットワーキングを設定する](#)」で、Amazon EKS のさまざまなコンポーネントがどのように機能するかについて、ご確認いただけます。
- 接続されたクラスターについては、[Amazon EKS Connector](#)をご覧ください。

トピック

- [Amazon EKS Auto Mode クラスターを作成する](#)
- [Amazon EKS クラスターを作成します。](#)
- [Kubernetes バージョンアップグレードの準備およびクラスターインサイトでの設定ミスのトラブルシューティング](#)
- [既存のクラスターを新しい Kubernetes バージョンに更新する](#)
- [クラスターを削除](#)
- [クラスター API サーバーエンドポイント](#)
- [EKS クラスターに WiWindows ノードをデプロイする](#)
- [Windows サポートを無効にする](#)
- [インターネットアクセスが制限されたプライベートクラスターをデプロイする](#)
- [EKS の Kubernetes バージョンライフサイクルを理解する](#)
- [各 Kubernetes バージョンの Amazon EKS プラットフォームバージョンを表示する](#)
- [Karpenter と Cluster Autoscaler を使用したクラスターコンピューティングのスケールリング](#)
- [Amazon EKS での Amazon Application Recovery Controller \(ARC\) のゾーンシフトの詳細](#)
- [EKS ゾーンシフトを有効にしてアベイラビリティゾーンの障害を回避する](#)

Amazon EKS Auto Mode クラスターを作成する

このトピックでは、高度な設定オプションを使用して Amazon EKS Auto Mode クラスターを作成する詳細な手順について説明します。前提条件、ネットワーキングオプション、アドオン設定を取り

上げています。このプロセスには、IAM ロールの設定、クラスター設定の構成、ネットワーキングパラメータの指定、アドオンの選択が含まれます。ユーザーは、AWS Management ConsoleまたはAWS CLI のいずれかを使用してクラスターを作成でき、どちらの方法でもステップバイステップのガイダンスが付いています。

より複雑でないセットアッププロセスを求める場合は、クラスター作成手順の簡略化について以下を参照してください。

- [the section called “eksctl CLI”](#)
- [the section called “AWS CLI”](#)
- [the section called “マネジメントコンソール”](#)

この高度な設定ガイドは、EKS Auto Mode クラスターのセットアップをより細かく制御する必要があり、Amazon EKS の概念と要件に精通しているユーザーを対象としています。高度な設定に進む前に、すべての前提条件を満たし、EKS Auto Mode クラスターのネットワーキングおよび IAM 要件を十分に理解してください。

EKS Auto Mode には、追加の IAM アクセス許可が必要です。詳細については、以下を参照してください。

- [the section called “EKS Auto Mode クラスターの IAM ロール”](#)
- [the section called “ID とアクセス”](#)

Note

EKS Auto Mode を使用せずにクラスターを作成する場合は、「[the section called “クラスターを作成する”](#)」を参照してください。

このトピックでは、高度な設定を取り上げています。EKS Auto Mode を使用して開始する場合は、「[the section called “クラスターを作成する”](#)」を参照してください。

前提条件

- [アマゾン EKS の要件](#) を満たす既存の VPC とサブネット。本番用にクラスターをデプロイする前に、VPC とサブネットの要件を十分に理解しておくことをお勧めします。VPC とサブネットがない場合は[アマゾン EKS に用意されている AWS クラウドフォーメーション テンプレート](#)を使用して作成できます。

- デバイスまたは AWS クラウドシェル に、kubectl コマンドラインツールがインストールされていること。バージョンはご使用のクラスターの Kubernetes バージョンと同じか、1 つ前のマイナーバージョン以前、あるいはそれより新しいバージョンが使用できます。例えば、クラスターのバージョンが 1.29 である場合、kubectl のバージョン 1.28、1.29、または 1.30 が使用できます。kubectl をインストールまたはアップグレードする方法については「[the section called “kubectl と eksctl のセットアップ”](#)」を参照してください。
- ご使用のデバイスまたは AWS クラウドシェル で、バージョン 2.12.3 以降、または AWS コマンドラインインターフェイス (AWS CLI のバージョン 1.27.160 以降がインストールおよび設定されていること。現在のバージョンを確認するには「aws --version」を参照してください。最新バージョンをインストールするには「AWS コマンドラインインターフェイスユーザーガイド」の「[インストール](#)」および「[aws configure を使用したクイック設定](#)」を参照してください。
- EKS および IAM リソースを作成および変更するアクセス許可を持つ [IAM プリンシパル](#)。

クラスターの作成 - AWS コンソール

1. [アマゾン EKS コンソール](#)を開きます。
2. [クラスターを追加]、[作成] の順にクリックします。
3. [設定オプション] で [カスタム設定] を選択します。
 - このトピックでは、カスタム設定を取り上げています。クイック設定の詳細については、「[the section called “マネジメントコンソール”](#)」を参照してください。
4. [EKS Auto Mode を使用する] が有効になっていることを確認します。
 - このトピックでは、EKS Auto Mode でのクラスターの作成を取り上げています。EKS Auto Mode を使用せずにクラスターを作成する方法の詳細については、「[the section called “クラスターを作成する”](#)」を参照してください。
5. [クラスターの設定] ページで、次のフィールドに入力します：
 - [名前] - クラスターの名前。この名前には英数字 (大文字と小文字が区別されます)、ハイフン、下線のみを使用できます。先頭の文字は英数字である必要があります。また、100 文字より長くすることはできません。名前はクラスターを作成する AWS リージョンおよび AWS アカウント内で一意である必要があります。
 - [クラスター IAM ロール] - ユーザーに代わって AWS リソースを管理することを Kubernetes コントロールプレーンに許可するために作成した Amazon EKS クラスター IAM ロールを選択します。EKS Auto Mode 用のクラスター IAM ロールをまだ作成していない場合は、IAM コンソールで [推奨ロールを作成] ボタンを選択して、必要なアクセス許可を持つロールを作成します。

- [Kubernetes バージョン] – クラスターで使用する Kubernetes のバージョン。以前のバージョンが必要でない限り、最新バージョンを選択することをお勧めします。
 - [アップグレードポリシー] – クラスターに対して設定する Kubernetes バージョンポリシー。クラスターを標準サポートバージョンでのみ実行したい場合は、[標準] を選択できます。バージョンの標準サポート終了時にクラスターの延長サポートに入りたい場合は、[延長] を選択できます。現在延長サポートを利用中の Kubernetes バージョンを選択した場合、オプションとして標準サポートを選択することはできません。
6. [クラスターを設定] ページの [Auto Mode コンピューティング] セクションで、次のフィールドに入力します。
- ノードプール – ノードプールでビルドを使用するかどうかを決定します。詳細については、[「the section called “組み込みノードプールを確認する”」](#) を参照してください。
 - ノード IAM ロール – 組み込みノードプールのいずれかを有効にする場合は、ノード IAM ロールを選択する必要があります。EKS Auto Mode は、このロールを新しいノードに割り当てます。クラスターの作成後はこの値を変更することはできません。EKS Auto Mode 用のノード IAM ロールをまだ作成していない場合は、[推奨ロールを作成] ボタンを選択して、必要なアクセス許可を持つロールを作成します。このロールの詳細については、[「the section called “ID とアクセス”」](#) を参照してください。
7. [クラスターの設定] ページの [クラスターアクセス] セクションで、次のフィールドに入力します。
- [クラスター管理者アクセスをブートストラップ] – クラスター作成者は自動的に Kubernetes 管理者になります。これを無効にする場合は、[クラスター管理者アクセスを拒否] を選択します。
 - [クラスター認証モード] – EKS Auto Mode には、EKS アクセスエントリ、EKS API 認証モードが必要です。必要に応じて、[EKS API と ConfigMap] を選択して ConfigMap 認証モードを有効にできます。
8. 「クラスターの設定」ページの残りのフィールドに入力します。
- [Secret encryption] (シークレット暗号化) – (オプション) KMS キーを使用して Kubernetes シークレットのシークレット暗号化を有効にするよう選択します。クラスターを作成した後で、これを有効にすることもできます。この機能を有効にする前に、[the section called “シークレット暗号化を有効にする”](#) の情報をよく理解していることを確認してください。
 - [ARC ゾーンシフト] – EKS Auto Mode は ARC ゾーンシフトをサポートしていません。
 - [タグ] – (オプション) クラスターに任意のタグを追加します。詳細については、[「the section called “リソースのタグ付け”」](#) を参照してください。

このページを読み終えたら、[次へ] を選択してください。

9. [ネットワーキングの指定] ページで、次のフィールドの値を選択してください：

- [VPC] – [アマゾン EKS VPC 要件](#)を満たす既存の VPC を選択し、そこでクラスターを作成します。VPC を選択する前に、「[the section called “VPC およびサブネットの要件”](#)」の要件と考慮事項をすべて理解しておくことをお勧めします。クラスターの作成後は使用する VPC を変更できません。VPC が表示されていない場合はまず作成する必要があります。詳細については「[the section called “VPC を作成する”](#)」を参照してください。
- [サブネット] - デフォルトで、前のフィールドで指定した VPC 内の利用可能なすべてのサブネットがあらかじめ選択されています。少なくとも 2 つ選択する必要があります。

選択するサブネットは [アマゾン EKS サブネットの要件](#)を満たす必要があります。サブネットを選択する前に、[アマゾン EKS VPC およびサブネットの要件と考慮事項](#)をすべて理解しておくことをお勧めします。

[セキュリティグループ] - (オプション アマゾン EKS が作成するネットワークインターフェイスに関連付ける 1 つまたは複数のセキュリティグループを指定します。

セキュリティグループを選択するかどうかにかかわらず、アマゾン EKS はクラスターと VPC 間の通信を可能にするセキュリティグループを作成します。アマゾン EKS はこのセキュリティグループおよびユーザーが選択したセキュリティグループを、作成するネットワークインターフェイスに関連付けます。アマゾン EKS が作成するクラスターセキュリティグループの詳細については「[the section called “セキュリティグループの要件”](#)」を参照してください。アマゾン EKS が作成するクラスターセキュリティグループのルールを変更できます。

- クラスターの IP アドレスファミリーの選択 – IPv4 と IPv6 のどちらかを選択できます。

デフォルトで、Kubernetes は IPv4 アドレスを Pod とサービスに割り当てます。IPv6 ファミリーの使用を決定する前に、[VPC の要件と考慮事項](#)、[the section called “サブネットの要件と考慮事項”](#)、[the section called “セキュリティグループの要件”](#)、および [the section called “IPv6”](#) のトピックの考慮事項と要件をすべて理解していることを確認します。IPv6 ファミリーを選択すると、Kubernetes が IPv6 サービスアドレスを割り当てる範囲を IPv4 ファミリーで指定できるようには指定できません。Kubernetes は、一意のローカルアドレス範囲 (fc00::/7) からサービスにアドレスを割り当てます。

- (オプション) [Kubernetes サービス IP アドレスの範囲を設定する] を選択し、[サービスの IPv4 範囲] を指定します。

独自の範囲を指定すると、Kubernetes サービスと VPC にピアリングまたは接続されたその他のネットワークとの間の競合を防ぐことができます。CIDR 表記で範囲を入力します。例: 10.2.0.0/16。

この CIDR ブロックでは以下の要件を満たす必要があります：

- 10.0.0.0/8、172.16.0.0/12、または 192.168.0.0/16 のいずれかの範囲内にある。
- 最小サイズが /24、最大サイズが /12。
- アマゾン EKS リソースの VPC の範囲と重複しない。

このオプションを指定できるのは IPv4 アドレスファミリーを使用してクラスターを作成するときのみです。これを指定しない場合、Kubernetes は、10.100.0.0/16 または 172.20.0.0/16 のいずれかの CIDR ブロックからサービス IP アドレスを割り当てます。

- [クラスターエンドポイントのアクセス] で、オプションを選択してください。クラスターを作成した後で、このオプションを変更できます。デフォルト以外のオプションを選択する前に、オプションとその意味を理解しておいてください。詳細については「[the section called “クラスターエンドポイントのアクセス”](#)」を参照してください。

このページを読み終えたら、[次へ] を選択してください。

10(オプション) [オブザーバビリティの設定] ページで、有効にする [メトリクス] と [コントロールプレーンのロギング] オプションを選択してください。デフォルトではそれぞれのログタイプは無効化されています。

- Prometheus メトリクスの詳細については、「[the section called “ステップ 1: Prometheus メトリクスを有効にする”](#)」を参照してください。
- [トラフィック設定] のオプションの詳細については「[the section called “コントロールプレーンのログ”](#)」を参照してください。
- このページを読み終えたら、[次へ] を選択してください。

11[アドオンの選択] ページで、クラスターに追加するアドオンを選択してください。[アマゾン EKS アドオン] と [AWS マーケットプレイス アドオン] は必要な数だけ選択できます。インストールする AWS マーケットプレイス アドオンがリストされていない場合はページ番号をクリックして追加のページ結果を表示するか、または検索ボックスにテキストを入力して使用可能な AWS マーケットプレイス アドオンを検索できます。[カテゴリ]、[ベンダー]、または [料金モデル] でフィルタリングして、検索結果からアドオンを選択することもできます。クラスターを作成する際に、「[the section called “Pod Identity”](#)」で詳述されているように、EKS ポッドアイデンティティをサポートするアドオンを表示、選択、インストールできます。

- EKS Auto Mode は、特定のアドオンの機能を自動化します。EKS Auto Mode クラスターに EKS マネージドノードグループをデプロイする場合は、[追加の Amazon EKS アドオン] を選択し、オプションを確認します。CoreDNS や kube-proxy などのアドオンをインストールすることが必要になる場合があります。EKS は、このセクションのアドオンをセルフマネージドノードとノードグループにのみインストールします。
- このページを読み終えたら、[次へ] を選択してください。

12[選択したアドオン設定の構成] ページで、インストールするバージョンを選択し、[次へ] を選択してください。クラスターを作成した後はいつでも新しいバージョンに更新できます。

EKS ポッドアイデンティティをサポートするアドオンの場合、コンソールを使用して、アドオン専用に入力された名前、AWS マネージドポリシー、信頼ポリシーを使用してロールを自動的に生成できます。既存のロールを再利用したり、サポートされているアドオン用の新しいロールを作成したりできます。コンソールを使用して EKS ポッドアイデンティティをサポートするアドオンのロールを作成するためのステップについては「[the section called “アドオンの作成 \(AWS コンソール\)”](#)」を参照してください。アドオンが EKS Pod Identity をサポートしていない場合、クラスターの作成後にウィザードを使用してサービスアカウント用の IAM ロール (IRSA 作成する手順を示すメッセージが表示されます。

クラスターの作成後に、各アドオンの設定を更新できます。アドオンの設定の詳細については「[the section called “アドオンを更新する”](#)」を参照してください。このページを読み終えたら、[次へ] を選択してください。

13[確認と作成] ページで、前のページで入力または選択した情報を確認します。変更する必要がある場合は[編集] を選択してください。そのままであれば、[作成] を選択してください。クラスターがプロビジョニングされている間、[ステータス] フィールドに [作成中] と表示されます。

Note

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

クラスターのプロビジョニングには数分かかります。

クラスターの作成 - AWS CLI

次の CLI の手順では、IAM リソースの作成とクラスターの作成について説明します。

EKS Auto Mode クラスター IAM ロールを作成する

ステップ 1: 信頼ポリシーを作成する

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

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

ステップ 2: IAM ロールを作成する

信頼ポリシーを使用して、クラスター IAM ロールを作成します。

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

ステップ 3: ロール ARN をメモする

以降のステップで使用する新しいロールの ARN を取得して保存します。

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

ステップ 4: 必要なポリシーをアタッチする

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

AmazonEKSClusterPolicy:

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

アマゾンEKSコンピュートポリシー:

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

アマゾンEKSブロックストレージポリシー:

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

アマゾンEKSロードバランシングポリシー:

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

AmazonEKSNetworkingPolicy:

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

EKS Auto Mode ノード IAM ロールを作成する

ステップ 1: 信頼ポリシーを作成する

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

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "ec2.amazonaws.com"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
```

ステップ 2: ノード IAM ロールを作成する

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

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

ステップ 3: ロール ARN をメモする

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

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

ステップ 4: 必要なポリシーをアタッチする

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

AmazonEKSWorkerNodeMinimalPolicy:

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

AmazonEC2ContainerRegistryPullOnly:

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

クラスターを作成する

1. 下記のコマンドを使用して、クラスターを作成します。コマンドを実行する前に、次の置き換えを行います：

- ##### はクラスターを作成する AWS リージョンに置き換えます。
- ##### の部分は自分のクラスター名に置き換えます。この名前には英数字 (大文字と小文字が区別されます)、ハイフン、下線のみを使用できます。先頭の文字は英数字である必要があります。また、100 文字より長くすることはできません。名前はクラスターを作成する AWS リージョンおよび AWS アカウント内で一意である必要があります。
- 1.30 を [Amazon EKS がサポートする任意のバージョン](#) に置き換えます。
- 111122223333 は、自分のアカウント ID に置き換えます。
- クラスターロールとノードロールに異なる名前の IAM ロールを作成した場合は、ARN を置き換えます。
- subnetIds の値を独自の値に置き換えます。さらに ID を追加することもできます。少なくとも 2 つのサブネット ID を指定する必要があります。

選択するサブネットは [アマゾン EKS サブネットの要件](#)を満たす必要があります。サブネットを選択する前に、[アマゾン EKS VPC およびサブネットの要件と考慮事項](#)をすべて理解しておくことをお勧めします。

- セキュリティグループ ID を指定しない場合はコマンドから ,securityGroupIds=sg-
<ExampleID1> を削除します。1 つまたは複数のセキュリティグループ ID を指定する場合は securityGroupIds の値を独自の値に置き換えます。さらに ID を追加することもできます。

セキュリティグループを選択するかどうかにかかわらず、アマゾン EKS はクラスターと VPC 間の通信を可能にするセキュリティグループを作成します。アマゾン EKS はこのセキュリティグループおよびユーザーが選択したセキュリティグループを、作成するネットワークインターフェイスに関連付けます。アマゾン EKS が作成するクラスターセキュリティグループの詳細については「[the section called “セキュリティグループの要件”](#)」を参照してください。アマゾン EKS が作成するクラスターセキュリティグループのルールを変更できます。


```
aws eks create-cluster \
  --region region-code \
  --name my-cluster \
  --kubernetes-version 1.30 \
  --role-arn arn:aws:iam::111122223333:role/AmazonEKSAutoClusterRole \
  --resources-vpc-config '{"subnetIds": ["subnet-ExampleID1", "subnet-ExampleID2"],
"securityGroupIds": ["sg-ExampleID1"], "endpointPublicAccess": true,
"endpointPrivateAccess": true}' \
  --compute-config '{"enabled": true, "nodeRoleArn":
"arn:aws:iam::111122223333:role/AmazonEKSAutoNodeRole", "nodePools": ["general-
purpose", "system"]}' \
  --kubernetes-network-config '{"elasticLoadBalancing": {"enabled": true}}' \
  --storage-config '{"blockStorage": {"enabled": true}}' \
  --access-config '{"authenticationMode": "API"}'
```

Note

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

必要に応じて前のコマンドに追加する必要があるオプションの設定を次に示します。これらのオプションは、クラスターの作成時にのみ有効にすることができ、作成後は有効にできません。

- どの IPv4 Classless Inter-domain Routing (CIDR) ブロックから Kubernetes がサービス IP アドレスを割り当てるかを指定する場合、`--kubernetes-network-config serviceIpv4Cidr=<cidr-block>` を次のコマンドに追加することによって指定する必要があります。

独自の範囲を指定すると、Kubernetes サービスと VPC にピアリングまたは接続されたその他のネットワークとの間の競合を防ぐことができます。CIDR 表記で範囲を入力します。例: `10.2.0.0/16`。

この CIDR ブロックでは以下の要件を満たす必要があります：

- 10.0.0.0/8、172.16.0.0/12、または 192.168.0.0/16 のいずれかの範囲内にある。
- 最小サイズが /24、最大サイズが /12。
- アマゾン EKS リソースの VPC の範囲と重複しない。

このオプションを指定できるのは IPv4 アドレスファミリーを使用してクラスターを作成するときのみです。これを指定しない場合、Kubernetes は、10.100.0.0/16 または 172.20.0.0/16 のいずれかの CIDR ブロックからサービス IP アドレスを割り当てます。

- クラスターを作成していて、そのクラスターで IPv4 アドレスではなく IPv6 アドレスを Pod とサービスに割り当てようとする場合は、次のコマンドに `--kubernetes-network-config ipFamily=ipv6` を追加します。

デフォルトで、Kubernetes は IPv4 アドレスを Pod とサービスに割り当てます。IPv6 ファミリーの使用を決定する前に、[VPC の要件と考慮事項](#)、[the section called “サブネットの要件と考慮事項”](#)、[the section called “セキュリティグループの要件”](#)、および [the section called “IPv6”](#) のトピックの考慮事項と要件をすべて理解していることを確認します。IPv6 ファミリーを選択すると、Kubernetes が IPv6 サービスアドレスを割り当てられる範囲を IPv4 ファミリーで指定できるようには指定できません。Kubernetes は、一意のローカルアドレス範囲 (fc00::/7) からサービスにアドレスを割り当てます。

2. クラスターがプロビジョニングされるまでに数分かかります。クラスターのステータスのクエリを実行するには次のコマンドを使用します。

```
aws eks describe-cluster --region region-code --name my-cluster --query  
"cluster.status"
```

次のステップ

- [the section called “kubectl を使用してクラスターにアクセスする”](#)
- [the section called “許可を付与する”](#)
- [クラスターのシークレット暗号化を有効にする。](#)
- [クラスターのログ記録を設定する。](#)
- [クラスターにノードを追加する。](#)

Amazon EKS クラスターを作成します。

Note

このトピックではEKS 自動モードI を使用しない EKS クラスターの作成を取り上げています。

EKS 自動モードI クラスターの詳細な作成手順については「[the section called “自動クラスターの作成”](#)」を参照してください。

EKS 自動モードI を使用して開始するには「[the section called “クラスターを作成する \(EKS Auto Mode\)”](#)」を参照してください。

このトピックでは使用可能なオプションの概要と、Amazon EKS クラスターの作成時に考慮すべき点を説明します。ノードのコンピューティングとしてオンプレミスインフラストラクチャを使用してクラスターを作成する必要がある場合は「[the section called “クラスターを作成する”](#)」を参照してください。Amazon EKS クラスターを初めて作成する場合は[はじめに](#) のガイドのいずれかに従うことをお勧めします。これらのガイドは使用可能なすべてのオプションを展開することなく、シンプルでデフォルトのクラスターを作成するのに役立ちます。

前提条件

- [Amazon EKS の要件](#) を満たす既存の VPC とサブネット。本番用にクラスターをデプロイする前に、VPC とサブネットの要件を十分に理解しておくことをお勧めします。VPC とサブネットがない場合は[Amazon EKS に用意されている AWS クラウドフォーメーション テンプレート](#) を使用して作成できます。
- デバイスまたは AWS クラウドシェル に、kubectl コマンドラインツールがインストールされていること。バージョンはご使用のクラスターの Kubernetes バージョンと同じか、1 つ前のマイナーバージョン以前、あるいはそれより新しいバージョンが使用できます。kubectl をインストールまたはアップグレードする方法については「[the section called “kubectl と eksctl のセットアップ”](#)」を参照してください。
- ご使用のデバイスまたは AWS クラウドシェル で、バージョン 2.12.3 以降、または AWS コマンドラインインターフェイス (AWS CLI のバージョン 1.27.160 以降がインストールおよび設定されていること。現在のバージョンを確認するには「`aws --version | cut -d / -f2 | cut -d ' ' -f1`」を参照してください。yum、apt-get、macOS 用の Homebrew などのパッケージマネージャーは、多くの場合 AWS CLI の最新バージョンより数バージョン古くなっています。最新バージョンをインストールするには「AWS コマンドラインインターフェイスユーザーガイド」の「[インストール](#)」および「[aws configure を使用したクイック設定](#)」を参照してくださ

い。AWS クラウドシェル にインストールされている AWS CLI バージョンも最新バージョンより数バージョン遅れることがあります。更新するには「AWS クラウドシェル ユーザーガイド」の「[ホームディレクトリへの AWS CLI のインストール](#)」を参照してください。

- Amazon EKS クラスターを create および describe するための許可を持つ [IAM プリンシパル](#)。詳細については[the section called “Outpost にローカル Kubernetes クラスターを作成する”](#)および[the section called “すべてのクラスターの一覧表示または説明”](#)を参照してください。

ステップ 1: クラスター IAM ロールを作成する

1. 既にクラスター IAM ロールがある場合、または eksctl を使用してクラスターを作成する場合はこのステップはスキップできます。デフォルトではeksctl により、ロールが自動的に作成されます。
2. IAM 信頼ポリシー用の JSON ファイルを作成するには次のコマンドを実行してください。

```
cat >eks-cluster-role-trust-policy.json <<EOF
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "eks.amazonaws.com"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
EOF
```

3. Amazon EKS クラスターの IAM ロールを作成します。必要であれば、前のステップでファイルを書き込んだコンピュータ上のパスを *eks-cluster-role-trust-policy.json* の前につけます。このコマンドは前のステップで作成した信頼ポリシーをロールに関連付けます。IAM ロールを作成するにはロールを作成する [IAM プリンシパル](#)に iam:CreateRole アクション (許可) を割り当てる必要があります。

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

4. Amazon EKS 管理のポリシーを割り当てるか、独自のカスタムポリシーを作成できます。カスタムポリシーで使用する必要がある最小限の許可については「[the section called “クラスター IAM ロール”](#)」を参照してください。

[AmazonEKSClusterPolicy](#) という名前の Amazon EKS マネージド型ポリシーをロールにアタッチします。IAM ポリシーを [IAM プリンシパル](#) にアタッチするにはポリシーのアタッチを行っているプリンシパルに、次のいずれかの IAM アクション (許可) を割り当てる必要があります: `iam:AttachUserPolicy` または `iam:AttachRolePolicy`。

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

サービスにリンクされたロール

Amazon EKS では、サービスにリンクされたロールが自動的に作成されます `AWSServiceRoleForAmazonEKS`。

これはクラスター IAM ロールに追加されます。サービスにリンクされたロールは Amazon EKS に直接リンクされた一意のタイプの IAM ロールです。このロールにより、Amazon EKS はアカウント内のクラスターを管理できます。詳細については「[the section called “クラスターの ロール”](#)」を参照してください。

EKS クラスターの作成に使用する IAM アイデンティティには、サービスにリンクされたロールを作成するアクセス許可が必要です。これには `iam:CreateServiceLinkedRole` 権限が含まれます。

サービスにリンクされたロールがまだ存在せず、現在の IAM ロールに作成するための十分なアクセス許可がない場合、クラスター作成オペレーションは失敗します。

ステップ 2: クラスターを作成する

以下を使用してクラスターを作成できます：

- [eksctl](#)
- [の AWS Management Console](#)
- [AWS CLI](#)

クラスターを作成します - eksctl

1. デバイスまたは AWS クラウドシェル にインストールされている eksctl コマンドラインツールのバージョン 0.210.0 以降が必要です。eksctl をインストールまたはアップグレードするには eksctl ドキュメントの「[インストール](#)」を参照してください。
2. デフォルトの AWS リージョンに、Amazon EKS のデフォルト Kubernetes バージョンを使用して、Amazon EKS IPv4 クラスターを作成します。コマンドを実行する前に、次の置き換えを行います：
3. ##### はクラスターを作成する AWS リージョンに置き換えます。
4. ##### の部分は自分のクラスター名に置き換えます。この名前には英数字 (大文字と小文字が区別されます) とハイフンのみを使用できます。先頭の文字は英数字である必要があります。また、100 文字より長くすることはできません。名前はクラスターを作成する AWS リージョンおよび AWS アカウント内で一意である必要があります。
5. 1.32 を [Amazon EKS がサポートする任意のバージョン](#) に置き換えます。
6. 要件を満たすように vpc-private-subnets の値を変更します。さらに ID を追加することもできます。少なくとも 2 つのサブネット ID を指定する必要があります。パブリックサブネットを指定する場合は --vpc-private-subnets を --vpc-public-subnets に変更できます。パブリックサブネットにはインターネットゲートウェイへのルートに関連付けられたルートテーブルがありますが、プライベートサブネットには関連付けられたルートテーブルがありません。可能な限り、プライベートサブネットを使用することをお勧めします。

選択するサブネットは [Amazon EKS サブネットの要件](#) を満たす必要があります。サブネットを選択する前に、[Amazon EKS VPC およびサブネットの要件と考慮事項](#) をすべて理解しておくことをお勧めします。

7. 次のコマンドを実行してください：

```
eksctl create cluster --name my-cluster --region region-code --version 1.32 --vpc-private-subnets subnet-ExampleID1,subnet-ExampleID2 --without-nodegroup
```

クラスターのプロビジョニングには数分かかります。クラスターの作成中は数行の出力が表示されます。出力の最後の行は次のサンプル行のようになります。

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

8. 「[the section called “ステップ 3: kubeconfig を更新する”](#)」に進みます

オプション設定

`eksctl` を使用してクラスターを作成するときに指定できるほとんどのオプションを表示するには `eksctl create cluster --help` コマンドを使用します。使用可能なオプションをすべて表示するには `config` ファイルを使用します。詳細については「[eksctl ドキュメント](#)」の「[Using config files](#)」(設定ファイルの使用) と「[設定ファイルのスキーマ](#)」を参照してください。[設定ファイルの例](#)は、GitHub で見つけることができます。

必要に応じて前のコマンドに追加する必要があるオプションの設定を次に示します。これらのオプションはクラスターの作成時にのみ有効にすることができ、作成後は有効にできません。これらのオプションを指定する必要がある場合は前のコマンドを使用するのではなく、[eksctl config ファイル](#)を使用してクラスターを作成し、設定を指定する必要があります。

- Amazon EKS が作成するネットワークインターフェイスに割り当てる 1 つまたは複数のセキュリティグループを指定する場合は[securityGroup](#) オプションを指定します。

セキュリティグループを選択するかどうかにかかわらず、Amazon EKS はクラスターと VPC 間の通信を可能にするセキュリティグループを作成します。Amazon EKS はこのセキュリティグループおよびユーザーが選択したセキュリティグループを、作成するネットワークインターフェイスに関連付けます。Amazon EKS が作成するクラスターセキュリティグループの詳細については「[the section called “セキュリティグループの要件”](#)」を参照してください。Amazon EKS が作成するクラスターセキュリティグループのルールを変更できます。

- どの IPv4 Classless Inter-domain Routing (CIDR) ブロックから Kubernetes がサービス IP アドレスを割り当てるかを指定する場合、[serviceIPv4CIDR](#) オプションを指定します。

独自の範囲を指定すると、Kubernetes サービスと VPC にピアリングまたは接続されたその他のネットワークとの間の競合を防ぐことができます。CIDR 表記で範囲を入力します。例: `10.2.0.0/16`。

この CIDR ブロックでは以下の要件を満たす必要があります：

- `10.0.0.0/8`、`172.16.0.0/12`、または `192.168.0.0/16` のいずれかの範囲内にある。
- 最小サイズが `/24`、最大サイズが `/12`。
- Amazon EKS リソースの VPC の範囲と重複しない。

このオプションを指定できるのは IPv4 アドレスファミリーを使用してクラスターを作成するときのみです。これを指定しない場合、Kubernetes は、`10.100.0.0/16` または `172.20.0.0/16` のいずれかの CIDR ブロックからサービス IP アドレスを割り当てます。

- クラスターを作成していて、そのクラスターで IPv4 アドレスではなく IPv6 アドレスを Pod とサービスに割り当てるようにする場合は、[ipFamily](#) オプションを指定します。

デフォルトで、Kubernetes は IPv4 アドレスを Pod とサービスに割り当てます。IPv6 ファミリーの使用を決定する前に、「[the section called “VPC の要件と考慮事項”](#)」、「[the section called “サブネットの要件と考慮事項”](#)」、「[the section called “セキュリティグループの要件”](#)」、および「[the section called “IPv6”](#)」のトピックの考慮事項と要件をすべて理解していることを確認します。IPv6 ファミリーを選択すると、Kubernetes が IPv6 サービスアドレスを割り当てる範囲を IPv4 ファミリーで指定できるようには指定できません。Kubernetes は、一意のローカルアドレス範囲 (fc00::/7) からサービスにアドレスを割り当てます。

クラスターの作成 - AWS コンソール

1. [Amazon EKS コンソール](#)を開きます。
2. [クラスターを追加]、[作成] の順にクリックします。
3. [設定オプション] で [カスタム設定] を選択してください。
 - EKS 自動モードI でクラスターをすばやく作成する方法については「[the section called “マネジメントコンソール”](#)」を参照してください。
4. [EKS 自動モード] で、[EKS 自動モードI を使用する] をオフに切り替えます。
 - カスタム設定で EKS 自動モードI クラスターを作成する方法については「[the section called “自動クラスターの作成”](#)」を参照してください。
5. [クラスターの設定] ページで、次のフィールドに入力します：
 - [名前] - クラスターの名前。この名前には英数字 (大文字と小文字が区別されます)、ハイフン、下線のみを使用できます。先頭の文字は英数字である必要があります。また、100 文字より長くすることはできません。名前はクラスターを作成する AWS リージョンおよび AWS アカウント内で一意である必要があります。
 - [クラスター IAM ロール] - ユーザーに代わって AWS リソースを管理することを Kubernetes コントロールプレーンに許可するために作成した Amazon EKS クラスター IAM ロールを選択します。
 - [Kubernetes バージョン] - クラスターで使用する Kubernetes のバージョン。以前のバージョンが必要でない限り、最新バージョンを選択することをお勧めします。
 - [サポートタイプ] - クラスターに設定する Kubernetes バージョンポリシー。クラスターを標準サポートバージョンでのみ実行する場合は[標準サポート]を選択できます。バージョンの標準サポート終了時にクラスターで延長サポートに入る場合は[延長サポート]を選択できます。

現在延長サポートを利用中の Kubernetes バージョンを選択した場合、オプションとして標準サポートを選択することはできません。

- [Secret encryption] (シークレット暗号化) – (オプション) KMS キーを使用して Kubernetes シークレットのシークレット暗号化を有効にするよう選択します。クラスターを作成した後で、これを有効にすることもできます。この機能を有効にする前に、[the section called “シークレット暗号化を有効にする”](#) の情報をよく理解していることを確認してください。
- [タグ] – (オプション) クラスターにタグを追加します。詳細については「[the section called “リソースのタグ付け”](#)」を参照してください。
- [ARC ゾーンシフト] – (任意 Route53 アプリケーションリカバリコントローラー を使用して、アベイラビリティゾーンの障害を軽減できます。詳細については「[the section called “ゾーンシフトの詳細”](#)」を参照してください。

6. [クラスターの設定] ページの [クラスターアクセス] セクションで、次のフィールドに入力します：

- [クラスター管理者アクセスをブートストラップ] – クラスター作成者は自動的に Kubernetes 管理者になります。これを無効にする場合は [クラスター管理者アクセスを拒否] を選択してください。
- [クラスター認証モード] – IAM ユーザーおよびロールに Kubernetes API へのアクセスを許可する方法を決定します。詳細については「[the section called “クラスター認証モードを設定する”](#)」を参照してください。

このページを読み終えたら、[次へ] を選択してください。

7. [ネットワーキングの指定] ページで、次のフィールドの値を選択してください：

- [VPC] – [Amazon EKS VPC 要件](#)を満たす既存の VPC を選択し、そこでクラスターを作成します。VPC を選択する前に、「[the section called “VPC およびサブネットの要件”](#)」の要件と考慮事項をすべて理解しておくことをお勧めします。クラスターの作成後は使用する VPC を変更できません。VPC が表示されていない場合はまず作成する必要があります。詳細については「[the section called “VPC を作成する”](#)」を参照してください。
- [サブネット] – デフォルトで、前のフィールドで指定した VPC 内の利用可能なすべてのサブネットがあらかじめ選択されています。少なくとも 2 つ選択する必要があります。

選択するサブネットは [Amazon EKS サブネットの要件](#)を満たす必要があります。サブネットを選択する前に、[Amazon EKS VPC およびサブネットの要件と考慮事項](#)をすべて理解しておくことをお勧めします。

[セキュリティグループ] – (オプション Amazon EKS が作成するネットワークインターフェイスに関連付ける 1 つまたは複数のセキュリティグループを指定します。

セキュリティグループを選択するかどうかにかかわらず、Amazon EKS はクラスターと VPC 間の通信を可能にするセキュリティグループを作成します。Amazon EKS はこのセキュリティグループおよびユーザーが選択したセキュリティグループを、作成するネットワークインターフェイスに関連付けます。Amazon EKS が作成するクラスターセキュリティグループの詳細については「[the section called “セキュリティグループの要件”](#)」を参照してください。Amazon EKS が作成するクラスターセキュリティグループのルールを変更できます。

- クラスターの IP アドレスファミリーの選択 – IPv4 と IPv6 のどちらかを選択できます。

デフォルトで、Kubernetes は IPv4 アドレスを Pod とサービスに割り当てます。IPv6 ファミリーの使用を決定する前に、[VPC の要件と考慮事項](#)、[the section called “サブネットの要件と考慮事項”](#)、[the section called “セキュリティグループの要件”](#)、および [the section called “IPv6”](#) のトピックの考慮事項と要件をすべて理解していることを確認します。IPv6 ファミリーを選択すると、Kubernetes が IPv6 サービスアドレスを割り当てる範囲を IPv4 ファミリーで指定できるようには指定できません。Kubernetes は、一意のローカルアドレス範囲 (fc00::/7) からサービスにアドレスを割り当てます。

- (オプション) [Kubernetes サービス IP アドレスの範囲を設定する] を選択し、[サービスの **IPv4** 範囲] を指定します。

独自の範囲を指定すると、Kubernetes サービスと VPC にピアリングまたは接続されたその他のネットワークとの間の競合を防ぐことができます。CIDR 表記で範囲を入力します。例: 10.2.0.0/16。

この CIDR ブロックでは以下の要件を満たす必要があります：

- 10.0.0.0/8、172.16.0.0/12、または 192.168.0.0/16 のいずれかの範囲内にある。
- 最小サイズが /24、最大サイズが /12。
- Amazon EKS リソースの VPC の範囲と重複しない。

このオプションを指定できるのは IPv4 アドレスファミリーを使用してクラスターを作成するときのみです。これを指定しない場合、Kubernetes は、10.100.0.0/16 または 172.20.0.0/16 のいずれかの CIDR ブロックからサービス IP アドレスを割り当てます。

- [クラスターエンドポイントのアクセス] で、オプションを選択してください。クラスターを作成した後で、このオプションを変更できます。デフォルト以外のオプションを選択する前に、オプションとその意味を理解しておいてください。詳細については「[the section called “クラスターエンドポイントのアクセス”](#)」を参照してください。

このページを読み終えたら、[次へ] を選択してください。

8. (オプション) [オブザーバビリティの設定] ページで、有効にする [メトリクス] と [コントロールプレーンのロギング] オプションを選択してください。デフォルトではそれぞれのログタイプは無効化されています。

- Prometheus メトリクスの詳細については、「[the section called “ステップ 1: Prometheus メトリクスを有効にする”](#)」を参照してください。
- [トラフィック設定] のオプションの詳細については「[the section called “コントロールプレーンのログ”](#)」を参照してください。

このページを読み終えたら、[次へ] を選択してください。

9. [アドオンの選択] ページで、クラスターに追加するアドオンを選択してください。特定のアドオンが事前に選択されています。[Amazon EKS アドオン] と [AWS マーケットプレイス アドオン] は必要な数だけ選択できます。インストールする AWS マーケットプレイス アドオンがリストされていない場合はページ番号をクリックして追加のページ結果を表示するか、または検索ボックスにテキストを入力して使用可能な AWS マーケットプレイス アドオンを検索できます。[カテゴリ]、[ベンダー]、または [料金モデル] でフィルタリングして、検索結果からアドオンを選択することもできます。クラスターを作成する際に、「[the section called “Pod Identity”](#)」で詳述されているように、EKS ポッドアイデンティティをサポートするアドオンを表示、選択、インストールできます。

このページを読み終えたら、[次へ] を選択してください。

Amazon VPC CNI、CoreDNS、kube-proxy などの一部のアドオンはデフォルトでインストールされます。デフォルトのアドオンのいずれかを無効にすると、Kubernetes アプリケーションを実行する機能に影響する場合があります。

10[選択したアドオン設定の構成] ページで、インストールするバージョンを選択し、[次へ] を選択してください。クラスターを作成した後はいつでも新しいバージョンに更新できます。

EKS ポッドアイデンティティをサポートするアドオンの場合、コンソールを使用して、アドオン専用に入力された名前、AWS マネージドポリシー、信頼ポリシーを使用してロールを自動的に生成できます。既存のロールを再利用したり、サポートされているアドオン用の新しいロールを作成したりできます。コンソールを使用して EKS ポッドアイデンティティをサポートするアドオンのロールを作成するためのステップについては「[the section called “アドオンの作成 \(AWS コンソール\)”](#)」を参照してください。アドオンが EKS Pod Identity をサポートしていない場合、クラスターの作成後にウィザードを使用してサービスアカウント用の IAM ロール (IRSA 作成する手順を示すメッセージが表示されます。

クラスターの作成後に、各アドオンの設定を更新できます。アドオンの設定の詳細については「[the section called “アドオンを更新する”](#)」を参照してください。このページを読み終えたら、[次へ] を選択してください。

- 11[確認と作成] ページで、前のページで入力または選択した情報を確認します。変更する必要がある場合は[編集] を選択してください。そのままであれば、[作成] を選択してください。クラスターがプロビジョニングされている間、[ステータス] フィールドに [作成中] と表示されます。

 Note

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

クラスターのプロビジョニングには数分かかります。

12. 「[the section called “ステップ 3: kubeconfig を更新する”](#)」に進みます

クラスターの作成 - AWS CLI

- 下記のコマンドを使用して、クラスターを作成します。コマンドを実行する前に、次の置き換えを行います：
 - #####** はクラスターを作成する AWS リージョンに置き換えます。
 - #####** の部分は自分のクラスター名に置き換えます。この名前には英数字 (大文字と小文字が区別されます)、ハイフン、下線のみを使用できます。先頭の文字は英数字である必要があります。また、100 文字より長くすることはできません。名前はクラスターを作成する AWS リージョンおよび AWS アカウント内で一意である必要があります。
 - 1.32** を [Amazon EKS がサポートする任意のバージョン](#) に置き換えます。
 - 111122223333** をアカウント ID に、**myAmazonEKSClusterRole** をクラスター IAM ロールの名前に置き換えます。
 - subnetIds の値を独自の値に置き換えます。さらに ID を追加することもできます。少なくとも 2 つのサブネット ID を指定する必要があります。

選択するサブネットは [Amazon EKS サブネットの要件](#)を満たす必要があります。サブネットを選択する前に、[Amazon EKS VPC およびサブネットの要件と考慮事項](#)をすべて理解しておくことをお勧めします。

- セキュリティグループ ID を指定しない場合はコマンドから ,securityGroupIds=sg-
<ExampleID1> を削除します。1 つまたは複数のセキュリティグループ ID を指定する場合はsecurityGroupIds の値を独自の値に置き換えます。さらに ID を追加することもできます。

セキュリティグループを選択するかどうかにかかわらず、Amazon EKS はクラスターと VPC 間の通信を可能にするセキュリティグループを作成します。Amazon EKS はこのセキュリティグループおよびユーザーが選択したセキュリティグループを、作成するネットワークインターフェイスに関連付けます。Amazon EKS が作成するクラスターセキュリティグループの詳細については「[the section called “セキュリティグループの要件”](#)」を参照してください。Amazon EKS が作成するクラスターセキュリティグループのルールを変更できます。

```
aws eks create-cluster --region region-code --name my-cluster --kubernetes-version  
1.32 \  
--role-arn arn:aws:iam::111122223333:role/myAmazonEKSClusterRole \  
--resources-vpc-config subnetIds=subnet-ExampleID1,subnet-  
ExampleID2,securityGroupIds=sg-ExampleID1
```

Note

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

必要に応じて前のコマンドに追加する必要があるオプションの設定を次に示します。これらのオプションはクラスターの作成時にのみ有効にすることができ、作成後は有効にできません。

- デフォルトではEKS はクラスターの作成時に複数のネットワーキングアドオンをインストールします。これには Amazon VPC CNI、CoreDNS、kube-proxy が含まれます。

これらのデフォルトのネットワーキングアドオンのインストールを無効にする場合は以下のパラメータを使用します。これはCiliumなどの代替 CNI に使用できます。詳細については「[EKS API リファレンス](#)」を参照してください。

```
aws eks create-cluster --bootstrapSelfManagedAddons false
```

- どの IPv4 Classless Inter-domain Routing (CIDR) ブロックから Kubernetes がサービス IP アドレスを割り当てるかを指定する場合、`--kubernetes-network-config serviceIpv4Cidr=<cidr-block>` を次のコマンドに追加することによって指定する必要があります。

独自の範囲を指定すると、Kubernetes サービスと VPC にピアリングまたは接続されたその他のネットワークとの間の競合を防ぐことができます。CIDR 表記で範囲を入力します。例: 10.2.0.0/16。

この CIDR ブロックでは以下の要件を満たす必要があります：

- 10.0.0.0/8、172.16.0.0/12、または 192.168.0.0/16 のいずれかの範囲内にある。
- 最小サイズが /24、最大サイズが /12。
- Amazon EKS リソースの VPC の範囲と重複しない。

このオプションを指定できるのは IPv4 アドレスファミリーを使用してクラスターを作成するときのみです。これを指定しない場合、Kubernetes は、10.100.0.0/16 または 172.20.0.0/16 のいずれかの CIDR ブロックからサービス IP アドレスを割り当てます。

- クラスターを作成していて、そのクラスターで IPv4 アドレスではなく IPv6 アドレスを Pod とサービスに割り当てるようにする場合は、次のコマンドに `--kubernetes-network-config ipFamily=ipv6` を追加します。

デフォルトで、Kubernetes は IPv4 アドレスを Pod とサービスに割り当てます。IPv6 ファミリーの使用を決定する前に、[VPC の要件と考慮事項](#)、[the section called “サブネットの要件と考慮事項”](#)、[the section called “セキュリティグループの要件”](#)、および [the section called “IPv6”](#) のトピックの考慮事項と要件をすべて理解していることを確認します。IPv6 ファミリーを選択すると、Kubernetes が IPv6 サービスアドレスを割り当てる範囲を IPv4 ファミリーで指定できるようには指定できません。Kubernetes は、一意のローカルアドレス範囲 (fc00::/7) からサービスにアドレスを割り当てます。

2. クラスターがプロビジョニングされるまでに数分かかります。クラスターのステータスのクエリを実行するには次のコマンドを使用します。

```
aws eks describe-cluster --region region-code --name my-cluster --query  
"cluster.status"
```

出力が「ACTIVE」を返すまで、次のステップに進まないでください。

3. 「[the section called “ステップ 3: kubeconfig を更新する”](#)」に進みます

ステップ 3: kubeconfig を更新する

1. eksctl を使用してクラスターを作成した場合、このステップはスキップできます。eksctl によってこのステップはすでに完了しているからです。新しいコンテキストを kubectl config ファイルに追加して、kubectl がクラスターと通信できるようにします。ファイルを作成および更新する方法の詳細については「[the section called “kubectl を使用してクラスターにアクセスする”](#)」を参照してください。

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

出力例は次のとおりです。

```
Added new context arn:aws:eks:region-code:111122223333:cluster/my-cluster to /home/  
username/.kube/config
```

2. 次のコマンドを実行して、クラスターとの通信を確認します。

```
kubectl get svc
```

出力例は次のとおりです。

NAME	TYPE	CLUSTER-IP	EXTERNAL-IP	PORT(S)	AGE
kubernetes	ClusterIP	10.100.0.1	<none>	443/TCP	28h

ステップ 4: クラスターのセットアップ

1. (推奨) Amazon EKS アドオンを使用するか、固有の AWS Identity and Access Management (IAM) アクセス許可を個々の Kubernetes ワークロードに付与できるようにするには、クラスター用に [IAM OpenID Connect \(OIDC\) プロバイダーを作成](#) します。クラスター用に IAM OIDC プロバイ

ダーを作成する必要があるのは 1 回だけです。Amazon EKS アドオンの詳細については「[the section called “Amazon EKS アドオン”](#)」を参照してください。ワークロードに特定の IAM アクセス許可を割り当てる方法については「[the section called “IRSA による認証情報”](#)」を参照してください。

2. (推奨) Amazon EC2 ノードをクラスターにデプロイする前に Amazon VPC CNI plugin for Kubernetes プラグイン用にクラスターを設定します。デフォルトではプラグインはクラスターとともにインストールされています。Amazon EC2 ノードをクラスターに追加すると、プラグインは追加する各 Amazon EC2 ノードに自動的にデプロイされます。プラグインでは次の IAM ポリシーのいずれかを IAM ロールにアタッチする必要があります。クラスターが IPv4 ファミリーを使用する場合は[AmazonEKS_CNI_Policy](#) マネージド IAM ポリシーを使用します。クラスターが IPv6 ファミリーを使用している場合、[自ら作成した IAM ポリシー](#)を使用します。

ポリシーをアタッチする IAM ロールはノード IAM ロール、またはプラグインにのみ使用される専用ロールです。このロールにポリシーをアタッチすることをお勧めします。ロールの作成の詳細については「[the section called “IRSA 用の設定”](#)」または「[Amazon EKS ノードの IAM ロール](#)」を参照してください。

3. AWS Management Console を使用してクラスターをデプロイした場合、このステップはスキップできます。AWS Management Consoleでは、デフォルトで、Amazon VPC CNI plugin for Kubernetes、CoreDNS、および kube-proxy Amazon EKS アドオンがデプロイされます。

eksctl または AWS CLI のいずれかを使用してクラスターをデプロイする場合、Amazon VPC CNI plugin for Kubernetes、CoreDNS、および kube-proxy セルフマネージド型アドオンがデプロイされます。クラスターと共に Amazon EKS アドオンにデプロイされる、Amazon VPC CNI plugin for Kubernetes、CoreDNS、および kube-proxy セルフマネージド型アドオンを移行できます。詳細については、「[the section called “Amazon EKS アドオン”](#)」を参照してください。

4. (オプション) まだ作成していない場合は、クラスターの Prometheus メトリクスを有効にできます。詳細については「Amazon Managed Service for Prometheus ユーザーガイド」の「[スクレイパーの作成](#)」を参照してください。
5. Amazon EBS ボリュームを使用するクラスターにワークロードをデプロイする予定の場合、ワークロードをデプロイする前に、[Amazon EBS CSI](#) をクラスターにインストールする必要があります。

次のステップ

- クラスターを作成した [IAM プリンシパル](#)はクラスターにアクセスできる唯一のプリンシパルです。[他の IAM プリンシパルに許可を付与](#)して、クラスターにアクセスできるようにします。

- クラスターを作成した IAM プリンシパルが、前提条件で参照されている最低限の IAM 許可しか持たない場合、そのプリンシパルに Amazon EKS 許可を追加することができます。Amazon EKS 許可を IAM プリンシパルに付与する方法については「[the section called “IAM リファレンス”](#)」を参照してください。
- クラスターを作成した IAM プリンシパル、またはその他のプリンシパルに Amazon EKS コンソールで Kubernetes リソースを表示させる場合は、[必要なアクセス許可](#)をエンティティに付与します。
- ノードと IAM プリンシパルが VPC 内からクラスターにアクセスできるようにする場合、クラスターのプライベートエンドポイントを有効にします。デフォルトではパブリックエンドポイントは有効です。プライベートエンドポイントを有効にした後、必要に応じてパブリックエンドポイントを無効にできます。詳細については「[the section called “クラスターエンドポイントのアクセス”](#)」を参照してください。
- [クラスターのシークレット暗号化を有効にする](#)。
- [クラスターのログ記録を設定する](#)。
- [クラスターにノードを追加する](#)。

Kubernetes バージョンアップグレードの準備およびクラスターインサイトでの設定ミスのトラブルシューティング

Amazon EKS クラスターインサイトでは、問題の検出と解決のための推奨事項が提供され、クラスターの管理に役立ちます。すべての Amazon EKS クラスターは、Amazon EKS がキュレートしたインサイトのリストと照合して、自動的かつ定期的にチェックされます。これらのインサイトチェックは Amazon EKS によって完全に管理され、どのような検出結果に対処するための推奨事項を提示します。

Amazon EKS は、設定インサイトとアップグレードインサイトの 2 種類のインサイトを提供します。設定インサイトは、クラスターまたはワークロードの機能を損なう可能性のある EKS Hybrid Nodes の設定ミスを特定します。アップグレードインサイトは、Kubernetes の新しいバージョンにアップグレードする機能に影響を与える可能性のある問題を特定します。

実行されたインサイトチェックのリストと、Amazon EKS によって特定された関連する問題を確認するには、AWS Management Console、AWS CLI、AWS SDK、および Amazon EKS ListInsights API オペレーションを呼び出すことができます。この手順については、「[the section called “クラスターインサイトを表示する”](#)」を参照してください。

頻度 Amazon EKS では、24 時間ごとにクラスターインサイトが更新されます。クラスターインサイトは、手動では更新できません。クラスターの問題を修正すると、クラスターインサイトが更新されるまでしばらく時間がかかります。修正が完了したかどうかを判断するには、変更を実施した時刻とインサイトチェックの「最終更新時刻」を比較します。

アクセス許可 Amazon EKS は、すべての EKS クラスターでクラスターインサイトのクラスターアクセスエントリを自動的に作成します。このエントリは、クラスターに関する情報を表示するアクセス許可を EKS に付与します。EKS はこの情報を使用してインサイトを生成します。詳細については、「[the section called “AmazonEKSClusterInsightsPolicy”](#)」を参照してください。

クラスターインサイトを使用した Kubernetes バージョンアップグレードの準備

Important

Amazon EKS では、特定のクラスターインサイトの問題が発生した際に `--force` フラグでのクラスターアップグレードが必要となる機能が一時的にロールバックされています。詳細については、GitHub の「[Temporary rollback of enforcing upgrade insights on update cluster version](#)」を参照してください。

クラスターの更新に関する詳細については、「[the section called “ステップ 3: クラスターコントロールプレーンを更新する”](#)」を参照してください。

アップグレードインサイトは、クラスターインサイト内の特定のタイプのインサイトチェックです。これらのチェックは Kubernetes バージョンアップグレードの準備状況に関するインサイトを返します。Amazon EKS は、すべての EKS クラスターでアップグレードインサイトチェックを実行します。

- クラスターの Kubernetes バージョンを更新する前に、[Amazon EKS コンソール](#)のオブザーバビリティダッシュボードの [クラスターインサイト] タブを確認してください。
- クラスターに問題が見つかった場合は、確認して適切な修正を行います。問題には、Amazon EKS および Kubernetes のドキュメントへのリンクも含まれます。
- 問題を修正したら、クラスターインサイトが更新されるまで待機します。すべての問題が解決したら、[クラスターを更新](#)します。

Amazon EKS は Kubernetes バージョンアップグレードの準備状況に関するインサイトを返します。アップグレードインサイトは、Kubernetes クラスターのアップグレードに影響する可能性のある問

題を特定します。これにより、管理者がアップグレードの準備に費やす労力が最小限に抑えられ、新しい Kubernetes バージョンでのアプリケーションの信頼性が高まります。クラスターは、Amazon EKS によって Kubernetes バージョンアップグレードに影響する可能性のある問題のリストと照合して自動的にスキャンされます。Amazon EKS は、各 Kubernetes バージョンリリースで行われた変更のレビューに基づいて、インサイトチェックのリストを頻繁に更新します。

Amazon EKS のアップグレードインサイトにより、新しいバージョンのテストと検証のプロセスがスピードアップします。また、クラスター管理者やアプリケーション開発者は、懸念事項を浮き彫りにし、修正に関するアドバイスを提供することで、最新の Kubernetes 機能を活用できます。

クラスターインサイトを使用した設定ミスのトラブルシューティング

EKS クラスターインサイトは、ハイブリッドノードを使用して Amazon EKS クラスターを自動的にスキャンし、Kubernetes コントロールプレーンからウェブフックへの通信、exec やログなどの kubectl コマンドなどに障害をもたらす設定の問題を特定します。設定インサイトは問題を明らかにし、修復に関する推奨事項を提供し、完全に機能するハイブリッドノードのセットアップまでの時間を短縮します。

クラスターインサイトを表示する

Amazon EKS は、設定インサイトとアップグレードインサイトの 2 種類のインサイトを提供します。設定インサイトは、クラスターまたはワークロードの機能を損なう可能性のある EKS Hybrid Nodes の設定ミスを特定します。アップグレードインサイトは、Kubernetes の新しいバージョンにアップグレードする機能に影響を与える可能性のある問題を特定します。

実行されたインサイトチェックのリストと、Amazon EKS によって特定された関連する問題を確認するには、AWS Management Console、AWS CLI、AWS SDK、および Amazon EKS ListInsights API オペレーションを呼び出すことができます。

設定インサイトを表示する (コンソール)

1. [Amazon EKS コンソール](#)を開きます。
2. クラスターのリストから、インサイトを確認する Amazon EKS クラスターの名前を選択します。
3. [クラスターを監視する] を選択します。
4. [クラスターヘルス] タブを選択します。
5. [設定インサイト] テーブルには、次の列が表示されます。
 - 名前 — Amazon EKS がクラスターに対して実行したチェック。

- インサイトステータス – ステータスが Error のインサイトは、クラスター機能に影響を与える可能性のある設定ミスがあることを意味します。ステータスが Warning のインサイトは、設定が文書化されたアプローチと一致しないことを意味しますが、クラスター機能を意図的に設定すると機能する可能性があります。ステータスが Passing のインサイトは、Amazon EKS でこのインサイトチェックに関連する問題がクラスター内で検出されなかったことを意味します。
- 説明 — アラートと修復のための推奨アクションを含むインサイトチェックの情報。
- 最終更新時刻 – このクラスターについてのインサイトのステータスが最後に更新された時刻。

アップグレードインサイトを表示する (コンソール)

1. [Amazon EKS コンソール](#)を開きます。
2. クラスターのリストから、インサイトを確認する Amazon EKS クラスターの名前を選択します。
3. [ダッシュボードを表示] を選択します。
4. [クラスターインサイト] タブを選択します。
5. [アップグレードインサイト] テーブルには、次の列が表示されます。
 - 名前 — Amazon EKS がクラスターに対して実行したチェック。
 - インサイトステータス — ステータスが「エラー」のインサイトは、通常、影響を受ける Kubernetes バージョンが現在のクラスターバージョンの N+1 であることを意味し、ステータスが「警告」とは、インサイトが将来の Kubernetes バージョン N+2 以上に適用されることを意味します。ステータスが「合格」のインサイトは、Amazon EKS がこのインサイトチェックに関連する問題をクラスター内で発見しなかったことを意味します。インサイトステータスが「不明」の場合は、Amazon EKS はクラスターがこのインサイトチェックの影響を受けているかどうかを判断できないことを意味します。
 - バージョン — インサイトにより問題の可能性がチェックされた Kubernetes バージョン。
 - 最終更新時刻 – このクラスターについてのインサイトのステータスが最後に更新された時刻。
 - 前回の移行時刻 – このインサイトのステータスが最後に変更された時刻。
 - 説明 — アラートと修復のための推奨アクションを含むインサイトチェックの情報。

クラスターインサイトを表示する (AWS CLI)

1. インサイトを確認したいクラスターを決定します。次のコマンドは、指定されたクラスターのインサイトをリスト表示しています。必要に応じてコマンドに次の変更を加え、変更したコマンドを実行してください：

- *region-code* を、AWS リージョンのコードに置き換えます。
- ##### の部分は自分のクラスター名に置き換えます。

```
aws eks list-insights --region region-code --cluster-name my-cluster
```

出力例は次のとおりです。

```
{
  "insights":
    [
      {
        "id": "a1b2c3d4-5678-90ab-cdef-EXAMPLE11111",
        "name": "Deprecated APIs removed in Kubernetes vX.XX",
        "category": "UPGRADE_READINESS",
        "kubernetesVersion": "X.XX",
        "lastRefreshTime": 1734557315.000,
        "lastTransitionTime": 1734557309.000,
        "description": "Checks for usage of deprecated APIs that are scheduled
for removal in Kubernetes vX.XX. Upgrading your cluster before migrating to the
updated APIs supported by vX.XX could cause application impact.",
        "insightStatus":
          {
            "status": "PASSING",
            "reason": "No deprecated API usage detected within the last 30
days.",
          },
      },
      {
        "id": "a1b2c3d4-5678-90ab-cdef-EXAMPLE22222",
        "name": "Kubelet version skew",
        "category": "UPGRADE_READINESS",
        "kubernetesVersion": "X.XX",
        "lastRefreshTime": 1734557309.000,
        "lastTransitionTime": 1734557309.000,
        "description": "Checks for kubelet versions of worker nodes in the
cluster to see if upgrade would cause non compliance with supported Kubernetes
kubelet version skew policy.",
        "insightStatus":
          {
            "status": "UNKNOWN",
            "reason": "Unable to determine status of node kubelet versions.",
          }
      }
    ]
  }
```

```

    },
  },
  {
    "id": "a1b2c3d4-5678-90ab-cdef-EXAMPLE33333",
    "name": "Deprecated APIs removed in Kubernetes vX.XX",
    "category": "UPGRADE_READINESS",
    "kubernetesVersion": "X.XX",
    "lastRefreshTime": 1734557315.000,
    "lastTransitionTime": 1734557309.000,
    "description": "Checks for usage of deprecated APIs that are scheduled
for removal in Kubernetes vX.XX. Upgrading your cluster before migrating to the
updated APIs supported by vX.XX could cause application impact.",
    "insightStatus":
    {
      "status": "PASSING",
      "reason": "No deprecated API usage detected within the last 30
days.",
    },
  },
  {
    "id": "a1b2c3d4-5678-90ab-cdef-EXAMPLEaaaaa",
    "name": "Cluster health issues",
    "category": "UPGRADE_READINESS",
    "kubernetesVersion": "X.XX",
    "lastRefreshTime": 1734557314.000,
    "lastTransitionTime": 1734557309.000,
    "description": "Checks for any cluster health issues that prevent
successful upgrade to the next Kubernetes version on EKS.",
    "insightStatus":
    {
      "status": "PASSING",
      "reason": "No cluster health issues detected.",
    },
  },
  {
    "id": "a1b2c3d4-5678-90ab-cdef-EXAMPLEbbbbb",
    "name": "EKS add-on version compatibility",
    "category": "UPGRADE_READINESS",
    "kubernetesVersion": "X.XX",
    "lastRefreshTime": 1734557314.000,
    "lastTransitionTime": 1734557309.000,
    "description": "Checks version of installed EKS add-ons to ensure they
are compatible with the next version of Kubernetes. ",

```

```

    "insightStatus": { "status": "PASSING", "reason": "All installed EKS
add-on versions are compatible with next Kubernetes version."},
  },
  {
    "id": "a1b2c3d4-5678-90ab-cdef-EXAMPLEccccc",
    "name": "kube-proxy version skew",
    "category": "UPGRADE_READINESS",
    "kubernetesVersion": "X.XX",
    "lastRefreshTime": 1734557314.000,
    "lastTransitionTime": 1734557309.000,
    "description": "Checks version of kube-proxy in cluster to see if
upgrade would cause non compliance with supported Kubernetes kube-proxy version
skew policy.",
    "insightStatus":
      {
        "status": "PASSING",
        "reason": "kube-proxy versions match the cluster control plane
version.",
      },
  },
  {
    "id": "a1b2c3d4-5678-90ab-cdef-EXAMPLEddddd",
    "name": "Deprecated APIs removed in Kubernetes vX.XX",
    "category": "UPGRADE_READINESS",
    "kubernetesVersion": "X.XX",
    "lastRefreshTime": 1734557315.000,
    "lastTransitionTime": 1734557309.000,
    "description": "Checks for usage of deprecated APIs that are scheduled
for removal in Kubernetes vX.XX. Upgrading your cluster before migrating to the
updated APIs supported by vX.XX could cause application impact.",
    "insightStatus":
      {
        "status": "PASSING",
        "reason": "No deprecated API usage detected within the last 30
days.",
      },
  },
],
"nextToken": null,
}

```

- インサイトの詳細を表示するには、次のコマンドを実行します。必要に応じてコマンドに次の変更を加え、変更したコマンドを実行してください：

- `region-code` を、AWS リージョンのコードに置き換えます。
- `a1b2c3d4-5678-90ab-cdef-EXAMPLE22222` を、クラスターインサイトのリストから取得したインサイト ID に置き換えます。
- `#####` の部分は自分のクラスター名に置き換えます。

```
aws eks describe-insight --region region-code --id a1b2c3d4-5678-90ab-cdef-EXAMPLE22222 --cluster-name my-cluster
```

出力例は次のとおりです。

```
{
  "insight": {
    "id": "a1b2c3d4-5678-90ab-cdef-EXAMPLE22222",
    "name": "Kubelet version skew",
    "category": "UPGRADE_READINESS",
    "kubernetesVersion": "1.27",
    "lastRefreshTime": 1734557309.000,
    "lastTransitionTime": 1734557309.000,
    "description": "Checks for kubelet versions of worker nodes in the cluster to see if upgrade would cause non compliance with supported Kubernetes kubelet version skew policy.",
    "insightStatus": {
      "status": "UNKNOWN",
      "reason": "Unable to determine status of node kubelet versions.",
    },
    "recommendation": "Upgrade your worker nodes to match the Kubernetes version of your cluster control plane.",
    "additionalInfo": {
      "Kubelet version skew policy": "https://kubernetes.io/releases/version-skew-policy/#kubelet",
      "Updating a managed node group": "https://docs.aws.amazon.com/eks/latest/userguide/update-managed-node-group.html",
    },
    "resources": [],
    "categorySpecificSummary": {
      "deprecationDetails": [],
      "addonCompatibilityDetails": []
    }
  }
}
```

既存のクラスターを新しい Kubernetes バージョンに更新する

Amazon EKS で利用可能な新しい Kubernetes バージョンがある場合には、Amazon EKS クラスターを最新バージョンに更新できます。

Important

クラスターをアップグレードすると、以前のバージョンにダウングレードすることはできません。新しい Kubernetes バージョンに更新する前に、「[the section called “Kubernetes バージョン”](#)」およびこのトピック内の更新手順を参照することをお勧めします。

新しい Kubernetes バージョンでは、大幅な変更が加えられている場合があります。このため、本稼働用クラスターで更新を行う前に、新しいバージョンの Kubernetes に対するアプリケーションの動作をテストしておくことをお勧めします。この際は、継続的な統合ワークフローを構築し、新しい Kubernetes バージョンに移行する前にアプリケーションの動作をテストします。

更新プロセスに含まれている Amazon EKS が、更新された Kubernetes バージョンを使用しながら新しい API サーバーノードを起動することで、既存のバージョンを置き換えます。Amazon EKS は、これらの新しいノードで、ネットワークトラフィックの標準インフラストラクチャと準備状況に関するヘルスチェックを実行し、想定どおりに動作していることを確認します。ただし、クラスターのアップグレードを開始すると、一時停止または停止することはできません。これらのヘルスチェックのいずれかが失敗すると、Amazon EKS はインフラストラクチャのデプロイを元に戻します。クラスターは前の Kubernetes バージョンのままになります。この際も、実行中のアプリケーションは影響を受けません。また、クラスターが不確定または回復不可能な状態のままになることもあります。Amazon EKS は定期的にすべてのマネージド型クラスターをバックアップします。さらに、必要に応じてクラスターを復元するメカニズムも存在します。Kubernetes インフラストラクチャの管理プロセスは、継続的に評価、改善されています。

クラスターをアップグレードする際、Amazon EKS には、クラスターの作成時に指定したサブネット内に、最大で 5 つの使用可能な IP アドレスが必要となります。Amazon EKS は、指定したサブネットのいずれかに、新しいクラスターの Elastic Network Interface (ネットワークインターフェイス) を作成します。新しいネットワークインターフェイスは、既存のネットワークインターフェイスがあるのとは b サブネット内に作成される場合があります。ですので、クラスター作成時に指定したサブネットのいずれでも [必要なクラスターとの通信](#) が許可されるよう、セキュリティグループルールを設定します。クラスターの作成時に指定したサブネットのいずれかが存在しない、使用できる十分な IP アドレスがない、または必要なクラスターとの通信を許可するセキュリティグループルールがない場合、更新が失敗する可能性があります。

クラスターの API サーバーエンドポイントに常にアクセスできるように、Amazon EKS では高可用性を備えた Kubernetes コントロールプレーンを提供しており、アップデート実行中に API サーバーインスタンスのローリングアップデートを行います。Kubernetes API サーバーエンドポイントをサポートする API サーバーインスタンスの IP アドレスの変更を考慮するために、API サーバークライアントが再接続を適切に管理しているか確認する必要があります。kubectl の最新バージョンと公式にサポートされている Kubernetes クライアント [ライブラリ](#) は、この再接続プロセスを透過的に実行します。

Note

クラスターの更新の詳細については、「EKS Best Practices Guide」の「[Best Practices for Cluster Upgrades](#)」を参照してください。このリソースは、アップグレードを計画し、クラスターのアップグレード戦略を理解するのに役立ちます。

Amazon EKS Auto Mode の考慮事項

- Amazon EKS Auto Mode のコンピューティング機能は、Kubernetes バージョンのノードを制御します。コントロールプレーンをアップグレードすると、EKS Auto Mode はマネージドノードの段階的な更新を開始します。EKS Auto Mode は、ポッドの停止状態の予算を尊重します。
- コンピューティングの自動スケーリング、ブロックストレージ、ロードバランシング機能など、Amazon EKS Auto Mode の機能を手動でアップグレードする必要はありません。

概要

Amazon EKS クラスターのアップグレードプロセスの概要は以下のとおりです。

1. クラスターがアップグレードをサポートする状態であることを確認します。これには、クラスターにデプロイされたリソースで使用されている Kubernetes API をチェックし、クラスターに正常性の問題がないことを確認することが含まれます。クラスターのアップグレードの準備状況を評価する際には、Amazon EKS のアップグレードインサイトを使用する必要があります。
2. コントロールプレーンを次のマイナーバージョンにアップグレードします (1.31 から 1.32 など)。
3. データプレーン内のノードをコントロールプレーンのノードと一致するようにアップグレードします。
4. その他にクラスターで実行されているアプリケーション (cluster-autoscaler など) がある場合は、そのアプリケーションをアップグレードします。

5. Amazon EKS が提供するアドオン (デフォルトで含まれるアドオンなど) をアップグレードします。
 - [Amazon VPC CNI の推奨バージョン](#)
 - [CoreDNS の推奨バージョン](#)
 - [kube-proxy の推奨バージョン](#)
6. クラスターと通信しているクライアントがある場合は、そのクライアントをアップグレードします (kubectl など)。

ステップ 1: アップグレードの準備をする

クラスターコントロールプレーンの Kubernetes バージョンと、ノードの Kubernetes バージョンを比較します。

- クラスターコントロールプレーンの Kubernetes バージョンを取得します。

```
kubectl version
```

- ノードの Kubernetes バージョンを取得します。このコマンドでは、セルフマネージド型およびマネージド型の Amazon EC2、Fargate、およびハイブリッドノードがすべて返されます。各 Fargate Pod は、独自のノードとしてリストされます。

```
kubectl get nodes
```

コントロールプレーンを新しい Kubernetes バージョンに更新する前に、クラスター内のマネージド型ノードと Fargate ノードの双方の Kubernetes マイナーバージョンが、コントロールプレーンのバージョンと同じであることを確認してください。例えば、コントロールプレーンがバージョン 1.29 を実行し、かつノードの 1 つがバージョン 1.28 を実行している場合、コントロールプレーンを 1.30 に更新する前に、ノードをバージョン 1.29 に更新する必要があります。また、コントロールプレーンを更新する前に、セルフマネージド型ノードとハイブリッドノードをコントロールプレーンと同じバージョンに更新することをお勧めします。詳細については[the section called “更新”](#)、[the section called “更新方式”](#)、および[the section called “ハイブリッドノードをアップグレードする”](#)を参照してください。Fargate ノードのマイナーバージョンがコントロールプレーンのバージョンよりも古い場合、まずノードの示す Pod を削除します。次に、コントロールプレーンを更新します。残りの Pod は、再デプロイ後に新しいバージョンに更新されます。

ステップ 2: アップグレードに関する考慮事項を確認する

Amazon EKS クラスターインサイトは、非推奨の Kubernetes API の使用など、Kubernetes バージョンのアップグレードに影響を与える可能性のある問題のリストに照らして、クラスターを自動的にスキャンします。Amazon EKS は、Kubernetes プロジェクトの変更の評価に基づいて、実行するインサイトチェックのリストを定期的に更新します。また、Amazon EKS は、新しいバージョンに伴って Amazon EKS サービスに変更が導入された際にも、インサイトチェックのリストを更新します。詳細については、「[the section called “クラスターのインサイト”](#)」を参照してください。

Kubernetes ドキュメントの「[Deprecated API Migration Guide](#)」を確認してください。

クラスターインサイトを確認する

Amazon EKS クラスターインサイトを使用して問題を特定する

1. [Amazon EKS コンソール](#)を開きます。
2. クラスターのリストから、インサイトを確認する Amazon EKS クラスターの名前を選択します。
3. [ダッシュボードを表示] を選択します。
4. [クラスターインサイト] タブを選択します。
5. [アップグレードインサイト] テーブルには、次の列が表示されます。
 - 名前 — Amazon EKS がクラスターに対して実行したチェック。
 - インサイトステータス — ステータスが「エラー」のインサイトは、通常、影響を受ける Kubernetes バージョンが現在のクラスターバージョンの N+1 であることを意味し、ステータスが「警告」とは、インサイトが将来の Kubernetes バージョン N+2 以上に適用されることを意味します。ステータスが「合格」のインサイトは、Amazon EKS がこのインサイトチェックに関連する問題をクラスター内で発見しなかったことを意味します。インサイトステータスが「不明」の場合は、Amazon EKS はクラスターがこのインサイトチェックの影響を受けているかどうかを判断できないことを意味します。
 - バージョン — インサイトにより問題の可能性がチェックされた Kubernetes バージョン。
 - 最終更新時刻 — このクラスターについてのインサイトのステータスが最後に更新された時刻。
 - 前回の移行時刻 — このインサイトのステータスが最後に変更された時刻。
 - 説明 — アラートと修復のための推奨アクションを含むインサイトチェックの情報。

詳細な考慮事項

- Amazon EKS は可用性の高いコントロールプレーンを実行しているため、一回に更新できるのはマイナーバージョン 1 つのみです。この要件の詳細については、「[Kubernetes Version and Version Skew Support Policy](#)」(Kubernetes のバージョンおよびバージョンスキューのサポートポリシー) を参照してください。現在のクラスターバージョンが 1.28 であり、1.30 に更新することを仮定します。最初にバージョン 1.28 クラスターをバージョン 1.29 に更新し、次にバージョン 1.29 クラスターをバージョン 1.30 に更新する必要があります。
- ノード上の Kubernetes kube-apiserver と kubelet 間のバージョンスキューを確認してください。
- Kubernetes バージョン 1.28 以降、kubelet は kube-apiserver より最大 3 マイナーバージョン古い場合があります。「[Kubernetes アップストリームバージョンのスキューポリシー](#)」を参照してください。
- マネージドノードと Fargate ノードの kubelet が Kubernetes バージョン 1.25 以降の場合は、kubelet バージョンを更新せずに最大 3 バージョン先までクラスターを更新できます。例えば、kubelet がバージョン 1.25 である場合、kubelet のバージョンを 1.25 のままにしても、Amazon EKS クラスターのバージョンは 1.25 から 1.26、1.27、1.28 に更新できます。
- 更新を開始する前のベストプラクティスとして、ノード上の kubelet がコントロールプレーンと同じ Kubernetes バージョンであることを確認してください。
- クラスターが 1.8.0 より前のバージョンの Amazon VPC CNI Plugin for Kubernetes で設定されている場合は、クラスターを更新する前に、プラグインを最新バージョンに更新することをお勧めします。プラグインのアップデートについては、「[the section called “Amazon VPC CNI”](#)」を参照してください。

ステップ 3: クラスターコントロールプレーンを更新する

Important

Amazon EKS では、特定のクラスターインサイトの問題が発生した際に `--force` フラグでのクラスターアップグレードが必要となる機能が一時的にロールバックされています。詳細については、GitHub の「[Temporary rollback of enforcing upgrade insights on update cluster version](#)」を参照してください。

Amazon EKS は、「最終更新時間」から 24 時間後にクラスターインサイトを更新します。問題に対処した時刻とクラスターインサイトの「最終更新時間」を比較できます。

さらに、非推奨 API の使用に対処した後、インサイトのステータスが更新されるまでに最大 30 日かかる場合があります。アップグレードインサイトは、常に、過去 30 日の期間中の非推奨 API の使用を確認しています。

EKS コントロールプレーンバージョンのアップグレードリクエストは、以下を使用して送信できます。

- [eksctl](#)
- [AWS コンソール](#)
- [AWS CLI](#)

クラスターの更新 - eksctl

この手順には、eksctl バージョン 0.210.0 以降が必要です。お使いのバージョンは、以下のコマンドを使用して確認できます。

```
eksctl version
```

eksctl のインストールと更新の手順については、eksctl ドキュメントの「[インストール](#)」を参照してください。

Amazon EKS コントロールプレーンの Kubernetes バージョンを更新します。<cluster-name> をクラスター名に置き換えます。<version-number> は、Amazon EKS がサポートする、クラスターの更新先のバージョン番号に置き換えてください。サポートされているバージョン番号のリストについては、「[the section called “Kubernetes バージョン”](#)」を参照してください。

```
eksctl upgrade cluster --name <cluster-name> --version <version-number> --approve
```

この更新は完了までに数分かかることがあります。

「[the section called “ステップ 4: クラスターコンポーネントを更新する”](#)」に進みます。

クラスターの更新 - AWS コンソール

1. [Amazon EKS コンソール](#)を開きます。
2. アップグレードするクラスターに対して [今すぐアップグレード] を選択します。
3. クラスターの更新先のバージョンを選択し、[アップグレード] を選択します。

4. この更新は完了までに数分かかることがあります。「[the section called “ステップ 4: クラスターコンポーネントを更新する”](#)」に進みます。

クラスターの更新 - AWS CLI

1. AWS CLI がインストールされ、ログインしていることを確認します。詳細については、「[AWS CLI の最新バージョンのインストールまたは更新](#)」を参照してください。
2. 次の AWS CLI コマンドを使用して、Amazon EKS クラスターを更新します。アップグレードするクラスターの <cluster-name> と <region-code> を置き換えます。<version-number> は、Amazon EKS がサポートする、クラスターの更新先のバージョン番号に置き換えてください。サポートされているバージョン番号のリストについては、「[the section called “Kubernetes バージョン”](#)」を参照してください。

```
aws eks update-cluster-version --name <cluster-name> \
    --kubernetes-version <version-number> --region <region-code>
```

出力例は次のとおりです。

```
{
  "update": {
    "id": "<update-id>",
    "status": "InProgress",
    "type": "VersionUpdate",
    "params": [
      {
        "type": "Version",
        "value": "<version-number>"
      },
      {
        "type": "PlatformVersion",
        "value": "eks.1"
      }
    ],
    [...]
  },
  "errors": []
}
```

3. この更新は完了までに数分かかることがあります。クラスター更新のステータスをモニタリングするには、次のコマンドを使用します。同じ <cluster-name> と <region-code> を使用することに加えて、前のコマンドで返された <update-id> を使用します。

```
aws eks describe-update --name <cluster-name> \
  --region <region-code> --update-id <update-id>
```

Successful ステータスが表示されると、更新は完了です。

4. 「[the section called “ステップ 4: クラスターコンポーネントを更新する”](#)」に進みます。

ステップ 4: クラスターコンポーネントを更新する

1. クラスターの更新が完了したら、更新したクラスターでの Kubernetes と同じマイナーバージョンに、ノードを更新する必要があります。詳細については[the section called “更新方式”](#)、[the section called “更新”](#)、および[the section called “ハイブリッドノードをアップグレードする”](#)を参照してください。Fargate で起動される新しい Pod であれば、クラスターのバージョンと一致する kubelet バージョンを持っています。それまでに存在していた Fargate Pod は変更されていません。
2. (オプション) クラスターを更新する前に、そのクラスターに Kubernetes Cluster Autoscaler をデプロイしてある場合は、更新後の Kubernetes のメジャーバージョンとマイナーバージョンに一致するように、Cluster Autoscaler を最新バージョンに更新します。
 - a. ウェブブラウザで Cluster Autoscaler の[リリース](#)ページを開き、クラスターの Kubernetes メジャーバージョンとマイナーバージョンに一致する最新の Cluster Autoscaler バージョンを見つけます。例えば、クラスターの Kubernetes バージョンが 1.30 である場合、1.30 で始まる最新の Cluster Autoscaler リリースを見つけます。次のステップで使用するため、そのリリースのセマンティックバージョン番号 (例: 1.30.n) を書き留めておきます。
 - b. 次のコマンドを使用して、Cluster Autoscaler イメージタグを、前のステップで書き留めたバージョンに設定します。必要に応じて、X.XX.X を独自の値に置き換えます。

```
kubectl -n kube-system set image deployment.apps/cluster-autoscaler cluster-autoscaler=registry.k8s.io/autoscaling/cluster-autoscaler:vX.XX.X
```

3. (GPU ノードを含むクラスターのみ) クラスターに GPU 対応のノードグループ (p3.2xlarge など) がある場合は、クラスターの [NVIDIA Device Plugin for Kubernetes](#) DaemonSet を更新する必要があります。次のコマンドを実行する前に、<vX.X.X> を必要となる [NVIDIA/k8s-device-plugin](#) バージョンに置き換えます。

```
kubectl apply -f https://raw.githubusercontent.com/NVIDIA/k8s-device-plugin/<vX.X.X>/deployments/static/nvidia-device-plugin.yml
```


4. Amazon VPC CNI Plugin for Kubernetes、CoreDNS、および kube-proxy アドオンを更新します。[サービスアカウントトークン](#)にリストされている最小バージョンに対してアドオンを更新することをお勧めします。
 - Amazon EKS アドオンを使用している場合は、Amazon EKS コンソールで [クラスター] をクリックし、左のナビゲーションペインで更新したクラスター名を選択します。通知がコンソールに表示されます。更新可能なアドオンごとに、新しいバージョンが利用可能であることが通知されます。アドオンを更新するには、[アドオン] タブを選択します。更新があるアドオンが表示されているボックスで [今すぐ更新] を選択し、使用可能なバージョンを選択してから、[更新] を選択します。
 - 別の方法として、AWS CLI または eksctl を使用してアドオンを更新することもできます。詳細については、「[the section called “アドオンを更新する”](#)」を参照してください。
5. 必要に応じて、kubectl のバージョンを更新します。Amazon EKS クラスターコントロールプレーンとのマイナーバージョンの相違が 1 つ以内である kubectl バージョンを使用する必要があります。

Amazon EKS クラスターの Kubernetes バージョンをダウングレードする

Amazon EKS クラスターの Kubernetes をダウングレードすることはできません。代わりに、以前の Amazon EKS バージョンで新しいクラスターを作成し、ワークロードを移行します。

クラスターを削除

Amazon EKS クラスターの使用が終了したら、関連付けられたリソースを削除して、不要なコストが発生しないようにする必要があります。

eksctl、AWS Management Console、またはAWS CLI を使用してクラスターを削除できます。

考慮事項

- クラスター内にロードバランサーに関連付けられているアクティブなサービスがある場合は、クラスターを削除する前にこれらのサービスを削除して、ロードバランサーが正しく削除されるようにする必要があります。この操作を行わないと、VPC 内のリソースが孤立し、VPC を削除できなくなる可能性があります。
- クラスター作成者が削除されたためにエラーが発生した場合は、[この記事](#)を参照して解決してください。

- Amazon Managed Service for Prometheus リソースはクラスターのライフサイクル外にあるため、クラスターとは別途維持する必要があります。クラスターを削除するときは、対象コストを抑えるために、該当するスクレイパーも削除してください。詳細については、Amazon Managed Service for Prometheus ユーザーガイドの「[スクレイパーの検出と作成](#)」を参照してください。
- 接続されたクラスターを削除するには、「[the section called “クラスターの登録解除”](#)」を参照してください。

EKS Auto Mode に関する考慮事項

- EC2 マネージドインスタンスを含む、すべての EKS Auto Mode ノードが削除されます
- すべてのロードバランサーが削除されます

詳細については、「[the section called “EKS 自動モードI を無効にする”](#)」を参照してください。

クラスターの削除 (eksctl)

この手順には、eksctl バージョン 0.210.0 以降が必要です。お使いのバージョンは、以下のコマンドを使用して確認できます。

```
eksctl version
```

eksctl のインストールまたはアップグレードの手順については、eksctl ドキュメントの「[インストール](#)」を参照してください。

1. クラスターで実行されているすべてのサービスを一覧表示します。

```
kubectl get svc --all-namespaces
```

- a. 関連付けられた EXTERNAL-IP 値のあるサービスすべてを削除します。これらのサービスの前面には Elastic Load Balancing のロードバランサーが置かれているため、そのロードバランサーや関連するリソースを適切に解放するためには、これらのサービスを Kubernetes から削除する必要があります。**service-name** を説明に従って各サービスの名前に置き換えます。

```
kubectl delete svc service-name
```

2. 以下のコマンドを使用して、クラスターとそれに関連するノードを削除します。この際、**prod** は実際のクラスター名に置き換えます。

```
eksctl delete cluster --name prod
```

出力:

```
[#] using region region-code
[#] deleting EKS cluster "prod"
[#] will delete stack "eksctl-prod-nodegroup-standard-nodes"
[#] waiting for stack "eksctl-prod-nodegroup-standard-nodes" to get deleted
[#] will delete stack "eksctl-prod-cluster"
[#] the following EKS cluster resource(s) for "prod" will be deleted: cluster. If in
doubt, check CloudFormation console
```

クラスターを削除する (AWS コンソール)

1. クラスターで実行されているすべてのサービスを一覧表示します。

```
kubectl get svc --all-namespaces
```

2. 関連付けられた EXTERNAL-IP 値のあるサービスすべてを削除します。これらのサービスの前面には Elastic Load Balancing のロードバランサーが置かれているため、そのロードバランサーや関連するリソースを適切に解放するためには、これらのサービスを Kubernetes から削除する必要があります。*service-name* を説明に従って各サービスの名前に置き換えます。

```
kubectl delete svc service-name
```

3. すべてのノードグループと Fargate プロファイルを削除します。
 - a. [Amazon EKS コンソール](#)を開きます。
 - b. 左のナビゲーションペインで Amazon EKS [クラスター] を選択し、クラスターのタブ付きリストから、削除するクラスター名を選択します。
 - c. [コンピューティング] タブを選択し、削除するノードグループを選択します。[削除] を選択し、ノードグループの名前を入力し、[削除] を選択します。クラスター内のすべてのノードグループを削除します。

Note

ここでリストに表示されるノードグループは、[マネージド型ノードグループ](#)のみです。

- d. 削除する [Fargate プロファイル] を選択し、[削除] を選択し、プロファイルの名前を入力した上で [削除] を選択します。クラスター内のすべてのFargate プロファイルを削除します。
4. すべてのセルフマネージド型ノードの AWS CloudFormation スタックを削除します。
 - a. [AWS CloudFormation コンソール](#)を開きます。
 - b. 削除するノードスタックを選択し、[削除] を選択します。
 - c. [スタックの削除] 確認ダイアログボックスで、[スタックを削除] をクリックします。クラスター内のすべてのセルフマネージド型ノードスタックを削除します。
5. クラスターを削除します。
 - a. [Amazon EKS コンソール](#)を開きます。
 - b. 削除するクラスターを選択し、[削除] を選択します。
 - c. クラスターの削除確認画面で、[削除] を選択します。
6. (オプション) VPC AWS CloudFormation スタックを削除します。
 - a. [AWS CloudFormation コンソール](#)を開きます。
 - b. 削除する VPC スタックを選択し、[削除] をクリックします。
 - c. [スタックの削除] 確認ダイアログボックスで、[スタックを削除] をクリックします。

クラスターを削除する (AWS CLI)

1. クラスターで実行されているすべてのサービスを一覧表示します。

```
kubectl get svc --all-namespaces
```

2. 関連付けられた EXTERNAL-IP 値のあるサービスすべてを削除します。これらのサービスの前面には Elastic Load Balancing のロードバランサーが置かれているため、そのロードバランサーや関連するリソースを適切に解放するためには、これらのサービスを Kubernetes から削除する必要があります。*service-name* を説明に従って各サービスの名前に置き換えます。

```
kubectl delete svc service-name
```

3. すべてのノードグループと Fargate プロファイルを削除します。
 - a. 次のコマンドを使用して、クラスター内のノードグループを一覧表示します。

```
aws eks list-nodegroups --cluster-name my-cluster
```

 Note

ここでリストに表示されるノードグループは、マネージド型ノードグループのみです。

- b. 次のコマンドを使用して、各ノードグループを削除します。クラスター内のすべてのノードグループを削除します。

```
aws eks delete-nodegroup --nodegroup-name my-nodegroup --cluster-name my-cluster
```

- c. 次のコマンドを使用して、クラスターの Fargate プロファイルを一覧表示します。

```
aws eks list-fargate-profiles --cluster-name my-cluster
```

- d. 次のコマンドを使用して、各 Fargate プロファイルを削除します。クラスター内のすべての Fargate プロファイルを削除します。

```
aws eks delete-fargate-profile --fargate-profile-name my-fargate-profile --cluster-name my-cluster
```

4. すべてのセルフマネージド型ノードの AWS CloudFormation スタックを削除します。

- a. 次のコマンドを使用して、使用可能な AWS CloudFormation スタックを一覧表示します。出力結果から、ノードテンプレート名を見つけます。

```
aws cloudformation list-stacks --query "StackSummaries[].StackName"
```

- b. 次のコマンドを使用して、各ノードスタックを削除します。この際、*node-stack* の部分は実際のノードスタック名に置き換えます。クラスター内のすべてのセルフマネージド型ノードスタックを削除します。

```
aws cloudformation delete-stack --stack-name node-stack
```

5. 以下のコマンドを使用して、クラスターを削除します。*my-cluster* は実際のクラスター名に置き換えてください。

```
aws eks delete-cluster --name my-cluster
```

6. (オプション) VPC AWS CloudFormation スタックを削除します。

- a. 次のコマンドを使用して、使用可能な AWS CloudFormation スタックを一覧表示します。結果の出力で VPC テンプレート名を見つけます。

```
aws cloudformation list-stacks --query "StackSummaries[].StackName"
```

- b. 以下のコマンドを使用して、VPC スタックを削除します。*my-vpc-stack* は実際の VPC スタック名に置き換えてください。

```
aws cloudformation delete-stack --stack-name my-vpc-stack
```

クラスター API サーバーエンドポイント

このトピックでは、Amazon EKS クラスターの Kubernetes API サーバーエンドポイントや上限へのプライベートアクセスを有効化する方法、あるいは、インターネットからのパブリックアクセスを完全に無効化する方法について説明します。

新しいクラスターを作成すると、Amazon EKS によって、マネージド型 Kubernetes API サーバー用にエンドポイントが作成されます。このエンドポイントは、ユーザーが (kubectl などの Kubernetes 管理ツールを使用ながら) クラスターとの通信に使用します。デフォルトでは、この API サーバーエンドポイントはインターネットに公開されます。API サーバーへのアクセスの保護には、AWS Identity and Access Management (IAM) と、ネイティブの Kubernetes [ロールベースアクセスコントロール](#) (RBAC) が組み合わせて使用されます。このエンドポイントはクラスターパブリックエンドポイントと呼ばれます。また、クラスタープライベートエンドポイントもあります。クラスタープライベートエンドポイントの詳細についてはセクション [the section called “クラスタープライベートエンドポイント”](#) を参照してください。

IPv6 クラスターエンドポイント形式

EKS は2024 年 10 月以降に作成された新しい IPv6 クラスターに対して、次の形式で一意的なデュアルスタックエンドポイントを作成します。IPv6 クラスターはクラスターの IP ファミリー (ipFamily) 設定で IPv6 を選択するクラスターです。

Example

AWS

EKS クラスターパブリック / プライベートエンドポイント: eks-cluster.*region*.api.aws

AWS GovCloud (US)

EKS クラスターパブリック / プライベートエンドポイント: eks-cluster.*region*.api.aws

Amazon Web Services in China

EKS クラスターパブリック / プライベートエンドポイント: eks-cluster.**region**.api.amazonwebservices.com.cn

Note

デュアルスタッククラスターエンドポイントは2024 年 10 月に導入されました。IPv6 クラスターの詳細については「[the section called “IPv6”](#)」を参照してください。2024 年 10 月より前に作成されたクラスターは代わりに次のエンドポイント形式を使用します。

IPv4 クラスターエンドポイント形式

EKS はクラスターの IP ファミリー (ipFamily 設定で IPv4 を選択したクラスターごとに、次の形式で一意的なエンドポイントを作成します：

Example

AWS

EKS クラスターパブリック / プライベートエンドポイント eks-cluster.**region**.eks.amazonaws.com

AWS GovCloud (US)

EKS クラスターパブリック / プライベートエンドポイント eks-cluster.**region**.eks.amazonaws.com

Amazon Web Services in China

EKS クラスターパブリック / プライベートエンドポイント eks-cluster.**region**.amazonwebservices.com.cn

Note

2024 年 10 月以前、IPv6 クラスターはこのエンドポイント形式も使用していました。これらのクラスターではパブリックエンドポイントとプライベートエンドポイントの両方において、このエンドポイントから解決できるのは IPv4 アドレスのみです。

クラスタープライベートエンドポイント

Kubernetes API サーバーへのプライベートアクセスを有効にすると、ノードと API サーバー間のすべての通信が VPC 内で行われるようになります。インターネットから API サーバーにアクセスできる IP アドレスを制限したり、API サーバーへのインターネットアクセスを完全に無効にしたりできます。

Note

このエンドポイントは Kubernetes API サーバー用であり、AWS API と通信するための従来の AWS PrivateLink エンドポイントではないため、Amazon VPC コンソール上にはエンドポイントとして表示されません。

クラスターでエンドポイントへのプライベートアクセスを有効にすると、Amazon EKS によって自動的に ルート 53 のプライベートホストゾーンが作成され、クラスターの VPC に関連付けられます。このプライベートホストゾーンは Amazon EKS によって管理され、アカウントの ルート 53 リソースには表示されません。プライベートホストゾーンが API サーバーに正しくトラフィックをルーティングするためには VPC で `enableDnsHostnames` と `enableDnsSupport` が `true` に設定され、VPC 用に設定された DHCP オプションで、ドメイン名サーバーリストに `AmazonProvidedDNS` が含まれている必要があります。詳細については [Amazon VPC ユーザーガイド](#) の「VPC の DNS サポートを表示および更新する」を参照してください。

API サーバーエンドポイントのアクセス要件は新しいクラスターを作成するときに定義できます。また、クラスターの API サーバーエンドポイントのアクセスは随時更新できます。

クラスターエンドポイントのアクセスの変更

既存クラスターのエンドポイントのアクセスを変更するにはこのセクションの手順に従ってください。次の表はサポートされている API サーバーエンドポイントのアクセスの組み合わせとそれらに関連付けられている動作を示しています。

エンドポイントのパブリックアクセス	エンドポイントのプライベートアクセス	行動
有効	無効	• これは新しい Amazon EKS クラスターのデフォルトの動作です。

エンドポイント のパブリックア クセス	エンドポイント のプライベート アクセス	行動
		• クラスターの VPC 内から送信される Kubernetes API リクエスト (ノードからコントロールプレーンへの通信など) は、VPC の外部に伝達されますが、Amazon のネットワークからは離れません。 • クラスター API サーバーにはインターネットからアクセスできます。必要に応じて、パブリックエンドポイントにアクセスできる CIDR ブロックを制限できます。特定の CIDR ブロックへのアクセスを制限する場合は、同時にプライベートエンドポイントも有効にすることをお勧めします。あるいは、ノードと (存在する場合は) Fargate Pod がパブリックエンドポイントにアクセスするためのアドレスを、対象の CIDR ブロックに含めるようにします。
有効	有効	• クラスターの VPC 内の Kubernetes API リクエスト (ノードとコントロールプレーン間の通信など) では、プライベート VPC エンドポイントが使用されます。 • クラスター API サーバーにはインターネットからアクセスできます。必要に応じて、パブリックエンドポイントにアクセスできる CIDR ブロックを制限できます。 • Amazon EKS クラスターでハイブリッドノードを使用している場合はパブリックとプライベートの両方のクラスターエンドポイントアクセスを有効にすることはお勧めしません。ハイブリッドノードは VPC の外部で実行されているため、クラスターエンドポイントはパブリック IP アドレスに解決されます。ハイブリッドノードを持つクラスターにはパブリックまたはプライベートのいずれかのクラスターエンドポイントアクセスを使用することをお勧めします。

エンドポイント のパブリックア クセス	エンドポイント のプライベート アクセス	行動
無効	有効	- クラスター API サーバーへのすべてのトラフィックはクラスターの VPC または 接続されたネットワーク 内から送信する必要があります。 - インターネットから API サーバーへのパブリックアクセスは存在しません。kubectl コマンドはすべて、VPC または接続されたネットワーク内から実行する必要があります。接続オプションについては the section called “プライベート専用 API サーバーへのアクセス” を参照してください。 - クラスターの API サーバーエンドポイントはパブリック DNS サーバーによって VPC のプライベート IP アドレスに解決されます。これまではエンドポイントは VPC 内からしか解決できませんでした。 エンドポイントが既存のクラスターの VPC 内のプライベート IP アドレスに解決されない場合は次の操作を実行できます： - パブリックアクセスを有効にし、再度無効にします。この操作はクラスターに対して 1 回行うだけで済みます。それ以降、エンドポイントはプライベート IP アドレスに解決されます。 - クラスターを 更新 します。

パブリックエンドポイントの CIDR ブロック (IPv6 クラスター)

パブリックエンドポイントはデュアルスタックであるため、IPv6 クラスターのパブリックエンドポイントに IPv6 および IPv4 CIDR ブロックを追加できます。これは、2024 年 10 月以降に作成した ipFamily が IPv6 に設定されている新しいクラスターにのみ適用されます。これらのクラスターは、新しいエンドポイントドメイン名 `api.aws` で識別できます。

パブリックエンドポイントの CIDR ブロック (IPv4 クラスター)

IPv4 クラスターのパブリックエンドポイントに IPv4 CIDR ブロックを追加できます。IPv4 クラスターのパブリックエンドポイントに IPv6 CIDR ブロックを追加することはできません。試みると、EKS は次のエラーメッセージを返します: The following CIDRs are invalid in publicAccessCidrs

パブリックエンドポイントの CIDR ブロック (2024 年 10 月以前に作成された **IPv6** クラスター)

2024 年 10 月以前に作成した古い IPv6 クラスターのパブリックエンドポイントに IPv4 CIDR ブロックを追加できます。これらのクラスターは eks.amazonaws.com エンドポイントで識別できます。2024 年 10 月以前に作成したこれらの古い IPv6 クラスターのパブリックエンドポイントに IPv6 CIDR ブロックを追加することはできません。試みると、EKS は次のエラーメッセージを返します: The following CIDRs are invalid in publicAccessCidrs

プライベート専用 API サーバーへのアクセス

クラスターの Kubernetes API サーバーエンドポイントに対するパブリックアクセスを無効にした場合は、VPC または [接続されたネットワーク](#) 内からのみ API サーバーにアクセスできます。Kubernetes API サーバーエンドポイントにアクセスする方法はいくつかあります。

接続されたネットワーク

[AWS トランジットゲートウェイ](#) またはその他の [接続](#) オプションを使用してネットワークを VPC に接続し、接続されたネットワークのコンピュータを使用します。接続されたネットワークからのポート 443 でのイングレストラフィックを許可するためのルールが、Amazon EKS コントロールプレーンセキュリティグループに含まれていることを確認する必要があります。

Amazon EC2 踏み台ホスト

Amazon EC2 インスタンスをクラスターの VPC のパブリックサブネットで起動し、SSH 経由でそのインスタンスにログインして kubectl コマンドが実行できます。詳細については [AWS での Linux 踏み台ホスト](#) を参照してください。踏み台ホストからのポート 443 でのイングレストラフィックを許可するためのルールが、Amazon EKS コントロールプレーンセキュリティグループに含まれていることを確認する必要があります。詳細については「[the section called “セキュリティグループの要件”](#)」を参照してください。

踏み台ホスト用に kubectl を設定するときにはクラスターの RBAC 設定に既にマッピングされている AWS 認証情報を使用するか、踏み台が使用する [IAM プリンシパル](#) を RBAC 設定に追加してから、エンドポイントのパブリックアクセスを削除します。詳細については [the section called “Kubernetes API アクセス”](#) および [the section called “許可されていないが、アクセスが拒否されました \(kubectl\)”](#) を参照してください。

AWS クラウド9 IDE

AWS クラウド9 はブラウザだけでコードを記述、実行、およびデバッグできるクラウドベースの統合開発環境 (IDE) です。クラスターの VPC に AWS クラウド9 IDE を作成し、その IDE を使用してクラスターと通信できます。詳細については「[AWS クラウド9 で環境を作成する](#)」を参照してください。Amazon EKS コントロールプレーンセキュリティグループに、IDE セキュリティグループからのポート 443 でのイングレストラフィックを許可するためのルールが、含まれていることを確認する必要があります。詳細については「[the section called “セキュリティグループの要件”](#)」を参照してください。

AWS クラウド9 IDE 用に `kubectl` を設定するときにはクラスターの RBAC 設定に既にマッピングされている AWS 認証情報を使用するか、IDE が使用する IAM プリンシパルを RBAC 設定に追加してから、エンドポイントのパブリックアクセスを削除してください。詳細については、「[the section called “Kubernetes API アクセス”](#)」および「[the section called “許可されていないが、アクセスが拒否されました \(kubectl\)”](#)」を参照してください。

[GitHub でこのページを編集する](#)

クラスター API サーバーエンドポイントへのネットワークアクセスを設定する

次のセクションでは、AWS Management Console または AWS CLI を使用して、クラスター API サーバーのエンドポイントアクセスを変更できます。

エンドポイントアクセスの設定 - AWS コンソール

1. [Amazon EKS コンソール](#)を開きます。
2. クラスター名を選択すると、そのクラスターの情報を表示されます。
3. [ネットワーキング] タブを選択し、[エンドポイントアクセスを管理] を選択します。
4. [プライベートアクセス] の場合はクラスターの Kubernetes API サーバーエンドポイントに対するプライベートアクセスを有効にするか無効にするかを選択してください。プライベートアクセスを有効にした場合、クラスターの VPC 内から送信される Kubernetes API リクエストはプライベート VPC エンドポイントを使用します。パブリックアクセスを無効にするにはプライベートアクセスを有効にする必要があります。
5. [パブリックアクセス] の場合はクラスターの Kubernetes API サーバーエンドポイントに対するパブリックアクセスを有効にするか無効にするかを選択してください。パブリックアクセスを無効

にすると、クラスターの Kubernetes API サーバーはクラスター VPC 内からのみリクエストを受信できます。

6. (オプション) [パブリックアクセス] で有効化を行うと、インターネットからパブリックエンドポイントと通信するためのアドレスを指定できるようになります。[詳細設定] を選択してください。「[203.0.113.5/32](#)」などの CIDR ブロックを入力します。ブロックに[予約済みアドレス](#)を含めることはできません。[ソースの追加] を選択すると、追加のブロックを入力できます。指定できる CIDR ブロックには最大数があります。詳細については、「[the section called "Service Quotas"](#)」を参照してください。ブロックをまったく指定しない場合、パブリック API サーバーエンドポイントは、IPv4 (0.0.0.0/0) と、さらにデュアルスタック IPv6 クラスターの IPv6 (:::/0) の両方のすべての IP アドレスからのリクエストを受信します。CIDR ブロックを使用してパブリックエンドポイントへのアクセスを制限する場合は、同時にプライベートエンドポイントアクセスも有効化することをお勧めします。これにより、ノードと (使用している場合は) Fargate Pods がクラスターと通信できるようになります。プライベートエンドポイントが有効になっていない場合はパブリックアクセスエンドポイントの CIDR ソースに、VPC からの出力ソースを含める必要があります。例えば、プライベートサブネットに NAT ゲートウェイを介してインターネットと通信するノードがある場合、パブリックエンドポイントで許可された CIDR ブロックの一部として、NAT ゲートウェイのアウトバウンド IP アドレスを追加する必要があります。
7. [更新] を選択して終了します。

エンドポイントアクセスの設定 - AWS CLI

AWS CLI バージョン 1.27.160 以降を使用して、次のステップを実行してください。現在のバージョンは `aws --version` で確認できます。AWS CLI をインストールまたはアップグレードするには「[AWS CLI のインストール](#)」を参照してください。

1. 次の AWS CLI コマンドを使用してクラスター API サーバーエンドポイントのアクセスを更新します。クラスター名と必要なエンドポイントアクセス値を置き換えま
す。endpointPublicAccess=true を設定した場合は(オプションで) 1 つの CIDR ブロック、または publicAccessCidrs の CIDR ブロックのカンマ区切りリストを入力できます。ブロックに[予約済みアドレス](#)を含めることはできません。CIDR ブロックを指定すると、パブリック API サーバーエンドポイントはリストされたブロックからのリクエストのみを受信します。指定できる CIDR ブロックには最大数があります。詳細については「[the section called "Service Quotas"](#)」を参照してください。CIDR ブロックを使用してパブリックエンドポイントへのアクセスを制限する場合は同時にプライベートエンドポイントアクセスも有効化することをお勧めします。これにより、ノードと (存在している場合は Fargate Pods がクラスターと通信できるようになります。プライベートエンドポイントが有効になっていない場合はパブリックアクセスエンドポイン

トの CIDR ソースに、VPC からの出力ソースを含める必要があります。例えば、プライベートサブネットに NAT ゲートウェイを介してインターネットと通信するノードがある場合、パブリックエンドポイントで許可された CIDR ブロックの一部として、NAT ゲートウェイのアウトバウンド IP アドレスを追加する必要があります。CIDR ブロックをまったく指定しない場合、パブリック API サーバーエンドポイントはすべての (0.0.0.0/0) IP アドレスからのリクエストと、さらにデュアルスタック IPv6 クラスターの IPv6 (:::/0) も受信します。

 Note

次のコマンドは API サーバーエンドポイントの 1 つの IP アドレスからのプライベートアクセスとパブリックアクセスを有効にします。**203.0.113.5/32** は単一の CIDR ブロック、またはネットワークアクセスを制限する CIDR ブロックのカンマ区切りリストに置き換えます。

```
aws eks update-cluster-config \
  --region region-code \
  --name my-cluster \
  --resources-vpc-config
endpointPublicAccess=true,publicAccessCidrs="203.0.113.5/32",endpointPrivateAccess=true
```

出力例は次のとおりです。

```
{
  "update": {
    "id": "e6f0905f-a5d4-4a2a-8c49-EXAMPLE000000",
    "status": "InProgress",
    "type": "EndpointAccessUpdate",
    "params": [
      {
        "type": "EndpointPublicAccess",
        "value": "true"
      },
      {
        "type": "EndpointPrivateAccess",
        "value": "true"
      },
      {
        "type": "publicAccessCidrs",
        "value": "[\"203.0.113.5/32\"]"
      }
    ]
  }
}
```

```
    }
  ],
  "createdAt": 1576874258.137,
  "errors": []
}
}
```

2. 次のコマンドでエンドポイントアクセス更新のステータスをモニタリングします。この際、以前の
コマンドで返ったクラスター名と更新 ID を使用します。ステータスが `Successful` と表示さ
れたら、更新は完了です。

```
aws eks describe-update \
  --region region-code \
  --name my-cluster \
  --update-id e6f0905f-a5d4-4a2a-8c49-EXAMPLE000000
```

出力例は次のとおりです。

```
{
  "update": {
    "id": "e6f0905f-a5d4-4a2a-8c49-EXAMPLE000000",
    "status": "Successful",
    "type": "EndpointAccessUpdate",
    "params": [
      {
        "type": "EndpointPublicAccess",
        "value": "true"
      },
      {
        "type": "EndpointPrivateAccess",
        "value": "true"
      },
      {
        "type": "publicAccessCidrs",
        "value": "[\"203.0.113.5/32\"]"
      }
    ],
    "createdAt": 1576874258.137,
    "errors": []
  }
}
```

[GitHub でこのページを編集する](#)

EKS クラスターに WiWindows ノードをデプロイする

Amazon EKS クラスターの Windows サポートを有効にして管理し、Linux コンテナと一緒に Windows コンテナを実行する方法について説明します。

考慮事項

Windows ノードをデプロイする前に、以下の考慮事項を確認してください。

- EKS Auto Mode は Windows ノードをサポートしていません
- HostProcess ポッドを使用して Windows ノードでホスト ネットワークを使用できます。詳細については、Kubernetes ドキュメントの「[Create a Windows HostProcessPod](#)」を参照してください。
- CoreDNS など、Linux でのみ動作するコアシステム Pod を実行するには、Amazon EKS クラスターに 1 つ以上の Linux ノードまたは Fargate ノードが含まれている必要があります。
- kubelet および kube-proxy イベントログは EKS Windows Event Log にリダイレクトされ、200 MB の制限に設定されます。
- Windows ノードで実行されている Pod で、[\[セキュリティグループを個別のポッドに割り当てる\]](#)を使用することはできません。
- Windows ノードで[カスタムネットワーク](#)を使用することはできません。
- IPv6 を Windows ノードで使用することはできません。
- Windows ノードは、ノードごとに 1 つの Elastic Network Interface をサポートします。デフォルトでは、Windows ノードごとに実行できる Pod の数は、ノードのインスタンスタイプの Elastic Network Interface ごとに使用できる IP アドレスの数から 1 を引いた数に等しくなります。詳細については、Amazon EC2 ユーザーガイドの「[各インスタンスタイプのネットワークインターフェイスごとの IP アドレス](#)」を参照してください。
- Amazon EKS クラスターでは、ロードバランサーを持つ単一のサービスが、最大 1,024 個のバックエンド Pod をサポートできます。各ポッドには固有の IP アドレスがあります。[OS ビルド 17763.2746](#) で始まる [Windows Server 更新プログラム](#)以降、以前の 64 個の Pod という上限はなくなりました。
- Windows コンテナは Fargate の Amazon EKS Pod ではサポートされていません。
- Windows を搭載した Amazon EKS Hybrid Nodes をホストのオペレーティングシステムとして使用することはできません。

- vpc-resource-controller ポッドからはログを取得できません。コントローラーをデータプレーンにデプロイしていたときには取得できていました。
- IPv4 アドレスが新しいポッドに割り当てられるまでにはクールダウン期間があります。これは、古い kube-proxy ルールが原因で、同じ IPv4 アドレスを持つ古いポッドにトラフィックが流れるのを防ぎます。
- コントローラーのソースは GitHub で管理されます。コントローラーに関する投稿や、問題の登録を行うには、GitHub の [project](#) にアクセスしてください。
- Windows マネージド型ノードグループのカスタム AMI ID を指定するときは、AWS IAM オーセンテイクーター設定マップに eks:kube-proxy-windows を追加します。詳細については、「[the section called “AMI ID を指定する場合の制限と条件”](#)」を参照してください。
- サブネットで使用可能な IPv4 アドレスを保持することが重要な場合は、「[EKS Best Practices Guide - Windows Networking IP Address Management](#)」のガイダンスを参照してください。
- EKS アクセスエントリに関する考慮事項
 - Windows インスタンスに対して別のノード IAM ロールを使用する場合、EKS は必要な Windows アクセスエントリを自動的に作成します。
 - Windows ノードで使用するアクセスエントリには、タイプとして EC2_WINDOWS を指定する必要があります。詳細については、「[the section called “アクセスエントリを作成する”](#)」を参照してください。

Windows ノード用のアクセスエントリを作成するには、次のように指定します。

```
aws eks create-access-entry --cluster-name my-cluster --principal-arn
arn:aws:iam::111122223333:role/<role-name> --type EC2_Windows
```

前提条件

- 既存のクラスター。
- CoreDNS を実行するには、クラスターに少なくとも 1 つ (推奨値は少なくとも 2 つ) の Linux ノードまたは Fargate Pod が必要です。レガシー Windows サポートを有効にする場合は、CoreDNS の実行に Linux ノードを使用する必要があります (Fargate Pod は使用できません)。
- 既存の [Amazon EKS クラスター IAM ロール](#)。

Windows サポートを有効にする

1. クラスターに Amazon Linux ノードがなく、Pod のセキュリティグループを使用する場合は、次のステップに進みます。それ以外の場合は、[クラスターのロール](#)に AmazonEKSVPCResourceController マネージドポリシーがアタッチされていることを確認します。**eksClusterRole** は、自分のクラスターのロール名に置き換えてください。

```
aws iam list-attached-role-policies --role-name eksClusterRole
```

出力例は次のとおりです。

```
{
  "AttachedPolicies": [
    {
      "PolicyName": "AmazonEKSClusterPolicy",
      "PolicyArn": "arn:aws:iam::aws:policy/AmazonEKSClusterPolicy"
    },
    {
      "PolicyName": "AmazonEKSVPCResourceController",
      "PolicyArn": "arn:aws:iam::aws:policy/AmazonEKSVPCResourceController"
    }
  ]
}
```

前の出力のようにポリシーがアタッチされている場合は、次のステップをスキップします。

2. [AmazonEKSVPCResourceController](#) マネージド型ポリシーを [Amazon EKS クラスターの IAM ロール](#) にアタッチします。**eksClusterRole** は、自分のクラスターのロール名に置き換えてください。

```
aws iam attach-role-policy \
  --role-name eksClusterRole \
  --policy-arn arn:aws:iam::aws:policy/AmazonEKSVPCResourceController
```

3. VPC CNI ConfigMap を更新して Windows IPAM を有効にします。
 - a. 次の内容で、**vpc-resource-controller-configmap.yaml** という名前のファイルを作成します。

```
apiVersion: v1
kind: ConfigMap
```

```
metadata:
  name: amazon-vpc-cni
  namespace: kube-system
data:
  enable-windows-ipam: "true"
```

b. ConfigMap をクラスターに適用する

```
kubectl apply -f vpc-resource-controller-configmap.yaml
```

4. クラスターの認証モードが aws-auth configmap を有効にするように設定されている場合は、次の手順を実行します。

- aws-auth ConfigMap に、Windows ノードのインスタンスロールに eks:kube-proxy-windows RBAC 権限グループを含めるマッピングが含まれていることを確認します。次のコマンドを使用してインストールします。

```
kubectl get configmap aws-auth -n kube-system -o yaml
```

出力例は次のとおりです。

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: aws-auth
  namespace: kube-system
data:
  mapRoles: |
    - groups:
      - system:bootstrappers
      - system:nodes
      - eks:kube-proxy-windows # This group is required for Windows DNS resolution
        to work
      rolearn: arn:aws:iam::111122223333:role/eksNodeRole
      username: system:node:{{EC2PrivateDNSName}}
[...]
```

グループの下に eks:kube-proxy-windows がリストされているはずですが、グループが指定されていない場合は、必要なグループを含むように ConfigMap を更新するか、作成する必要があります。aws-auth ConfigMapの詳細については、「[the section called “aws-auth ConfigMap をクラスターに適用する”](#)」を参照してください。

5. クラスターの認証モードが `aws-auth configmap` を無効にするように設定されている場合は、EKS アクセスエントリを使用できます。Windows インスタンスで使用する新しいノードロールを作成すると、EKS によって `EC2_WINDOWS` タイプのアクセスエントリが自動的に作成されます。

Windows ポッドをデプロイする

クラスターにポッドをデプロイするときに、さまざまなノードタイプを混在させて実行する場合に使用するオペレーティングシステムを指定する必要があります。

Linux Pod の場合は、マニフェストで以下のノードセレクターテキストを使用します。

```
nodeSelector:
  kubernetes.io/os: linux
  kubernetes.io/arch: amd64
```

Windows Pod の場合は、マニフェストで以下のノードセレクターテキストを使用します。

```
nodeSelector:
  kubernetes.io/os: windows
  kubernetes.io/arch: amd64
```

[サンプルアプリケーション](#)をデプロイして、使用されているノードセレクタを確認できます。

Windows ノードでより高い Pod 密度をサポートする

Amazon EKS では、各 Pod に VPC の IPv4 アドレスが割り当てられます。このため、ノード上でさらに多くの Pod を実行するのに十分なリソースがある場合でも、ノードにデプロイできる Pod の数の上限は、使用可能な IP アドレスによって制限されます。Windows ノードでサポートされる Elastic Network Interface は 1 つだけなので、デフォルトでは Windows ノードで利用できる IP アドレスの最大数は次のようになります。

```
Number of private IPv4 addresses for each interface on the node - 1
```

1 つの IP アドレスがネットワークインターフェイスのプライマリ IP アドレスとして使用されるため、Pod に割り当てることはできません。

IP プレフィックスの委任を有効にすると、Windows ノードで Pod 密度をより高めることができます。この機能により、セカンダリ IPv4 アドレスを割り当てる代わりに、プライマリネットワークイ

インターフェイスに /28IPv4 プレフィックスを割り当てることができます。IP プレフィックスを割り当てると、ノードで利用できる IPv4 アドレスの最大数が次のように増加します。

```
(Number of private IPv4 addresses assigned to the interface attached to the node - 1) * 16
```

使用可能な IP アドレス数が大幅に増加したため、使用可能な IP アドレスによってノード上の Pod の数をスケーリングできる機能が制限されることはありません。詳細については、「[the section called “IP アドレスを増やす”](#)」を参照してください。

Windows サポートを無効にする

1. クラスターに Amazon Linux ノードが含まれていて、[ポッドのセキュリティグループ](#)をそれらとともに使用する場合は、このステップをスキップしてください。

[クラスターのロール](#)から AmazonVPCResourceController マネージド IAM ポリシーを削除します。**eksClusterRole** は、実際に使用するクラスターのロールに置き換えてください。

```
aws iam detach-role-policy \
  --role-name eksClusterRole \
  --policy-arn arn:aws:iam::aws:policy/AmazonEKSVPCResourceController
```

2. amazon-vpc-cni ConfigMap で Windows IPAM を無効にします。

```
kubectl patch configmap/amazon-vpc-cni \
  -n kube-system \
  --type merge \
  -p '{"data":{"enable-windows-ipam":"false"}}'
```

インターネットアクセスが制限されたプライベートクラスターをデプロイする

このトピックでは、AWS Cloud にデプロイされているが、アウトバウンドインターネットアクセスがない Amazon EKS クラスターをデプロイする方法について説明します。AWS Outposts にローカルクラスターがある場合、このトピックの代わりに「[the section called “ノード”](#)」を参照してください。

Amazon EKS でのネットワークに詳しくない場合は、「[De-mystifying cluster networking for Amazon EKS worker nodes \(Amazon EKS ワーカーノードのクラスターネットワークを解明する\)](#)」を参照してください。クラスターにアウトバウンドインターネットアクセスがない場合、次の要件を満たす必要があります。

- クラスターは VPC 内のコンテナレジストリからイメージを取得する必要があります。VPC 内に Amazon Elastic Container Registry を作成し、そこにコンテナイメージをコピーしてノードの取得元にすることができます。詳細については、「[the section called “リポジトリにイメージをコピーする”](#)」を参照してください。
- クラスターでは、エンドポイントのプライベートアクセスが有効になる必要があります。これは、ノードをクラスターエンドポイントに登録するために必要です。エンドポイントのパブリックアクセスはオプションです。詳細については、「[the section called “クラスターエンドポイントのアクセス”](#)」を参照してください。
- セルフマネージド型 Linux ノードおよび Windows ノードには、起動する前に次のブートストラップ引数を含める必要があります。これらの引数は Amazon EKS のイントロスペクションをバイパスするため、VPC 内からの Amazon EKS API へのアクセスは不要です。
 - a. 次のコマンドを使用して、クラスターのエンドポイントの値を確認します。##### の部分は自分のクラスター名に置き換えます。

```
aws eks describe-cluster --name my-cluster --query cluster.endpoint --output text
```

出力例は次のとおりです。

```
https://EXAMPLE108C897D9B2F1B21D5EXAMPLE.sk1.region-code.eks.amazonaws.com
```

- b. 次のコマンドを使用して、クラスターの認証機関の値を確認します。##### の部分は自分のクラスター名に置き換えます。

```
aws eks describe-cluster --name my-cluster --query cluster.certificateAuthority --output text
```

返された出力は長い文字です。

- c. 次のコマンドの *cluster-endpoint* および *certificate-authority* を前のコマンドで返された出力の値に置き換えます。セルフマネージド型ノードを起動する際にブートストラップ引数を指定する方法の詳細については、「[the section called “Amazon Linux”](#)」および「[the section called “Windows”](#)」を参照してください。

- Linux ノードの場合:

```
--apiserver-endpoint cluster-endpoint --b64-cluster-ca certificate-authority
```

その他の引数については、GitHub の「[ブートストラップのスクリプト](#)」を参照してください。

- Windows ノードの場合:

 Note

カスタムサービス CIDR を使用している場合は `-ServiceCIDR` パラメータを使用して指定する必要があります。そうしないと、クラスター内の Pod の DNS 解決が失敗します。

```
-APIServerEndpoint cluster-endpoint -Base64ClusterCA certificate-authority
```

その他の引数については「[the section called “ブートストラップスクリプトの設定パラメータ”](#)」を参照してください。

- クラスターの `aws-auth ConfigMap` は VPC 内から作成する必要があります。エントリの作成と `aws-auth ConfigMap` への追加の詳細については、ターミナルで `eksctl create iamidentitymapping --help` と入力してください。サーバーに `ConfigMap` が存在しない場合は、このコマンドを使用して ID マッピングを追加したときに `eksctl` によって作成されます。
- [サービスアカウントの IAM ロール](#) で設定された Pod は、AWS Security Token Service (AWS STS) API コールから認証情報を取得します。アウトバウンドインターネットアクセスがない場合は、VPC 内で AWS STS VPC エンドポイントを作成して使用する必要があります。ほとんどの AWS v1 SDK は、AWS STS VPC エンドポイントを使用しないグローバル AWS STS エンドポイント (`sts.amazonaws.com`) をデフォルトで使用します。AWS STS VPC エンドポイントを使用するには、リージョンの AWS STS エンドポイント (`sts.region-code.amazonaws.com`) を使用するように SDK を構成する必要がある場合があります。詳細については、「[the section called “STS エンドポイント”](#)」を参照してください。
- Pod がアクセスする必要のあるすべての AWS サービスについて、クラスターの VPC サブネットに VPC インターフェイスエンドポイントが必要です。詳細については、「[インターフェイス VPC エンドポイントを使用して AWS サービスにアクセスする](#)」を参照してください。下表には、一般的に使用されるサービスとエンドポイントがリスト表示されています。エンドポイントの詳細なり

ストについては、[AWS PrivateLink ガイド](#)の「[AWS PrivateLink と連携する AWS サービス](#)」を参照してください。

VPC エンドポイントのために [プライベート DNS 名を有効にする](#) ことをお勧めします。これにより、ワークロードはパブリック AWS サービスエンドポイントを問題なく引き続き使用できます。

サービス	エンドポイント
Amazon EC2	com.amazonaws. <i>region-code</i> .ec2
Amazon Elastic Container Registry (コンテナイメージの取得用)	com.amazonaws. <i>region-code</i> .ecr.api、 com.amazonaws. <i>region-code</i> .ecr.dkr、および com.amazonaws. <i>region-code</i> .s3
Application Load Balancerおよび Network Load Balancer	com.amazonaws. <i>region-code</i> .elasticloadbalancing
AWS X-Ray	com.amazonaws. <i>region-code</i> .xray
Amazon CloudWatch Logs	com.amazonaws. <i>region-code</i> .logs
AWS Security Token Service (サービスアカウントに IAM ロールを使用している場合に必要)	com.amazonaws. <i>region-code</i> .sts
Amazon EKS Auth (Pod Identity 関連付けを使用する場合には必須)	com.amazonaws. <i>region-code</i> .eks-auth
Amazon EKS	com.amazonaws. <i>region-code</i> .eks

- セルフマネージド型ノードはすべて、必要な VPC インターフェイスエンドポイントを持つサブネットにデプロイする必要があります。マネージド型ノードグループを作成する場合には、VPC インターフェイスエンドポイントのセキュリティグループでサブネットの CIDR を許可するか、ノードのセキュリティグループを作成し VPC インターフェイスエンドポイントのセキュリティグループに追加する必要があります。
- Pod で Amazon EFS ボリュームを使用する場合、「[Store an elastic file system with Amazon EFS](#)」をデプロイする前に、Amazon EKS クラスターと同じ AWS リージョンを使用するように、ドライバーの [kustomization.yaml](#) ファイルを変更して、コンテナイメージを設定する必要があります。

- Route 53 は AWS PrivateLink をサポートしていません。プライベート Amazon EKS クラスターから Route 53 DNS レコードを管理することはできません。これは Kubernetes [external-dns](#) に影響します。
- EKS 最適化 AMI を使用した場合は、上記の表に挙げた ec2 エンドポイントを有効にする必要があります。あるいは、ノードの DNS 名を手動で設定することもできます。最適化 AMI では、EC2 API を使用して、ノードの DNS 名を自動的に設定します。
- [AWS Load Balancer Controller](#) を使用して、AWS Application Load Balancer (ALB) および Network Load Balancer をプライベートクラスターにデプロイできます。デプロイするときは、[コマンドラインフラグ](#)を使用して、enable-shield、enable-waf、および enable-wafv2 を false に設定する必要があります。Ingress オブジェクトのホスト名を使用した[証明書の検出](#)は、サポートされていません。これは、コントローラーが VPC インターフェイスエンドポイントを持たない AWS Certificate Manager に到達する必要があるからです。

このコントローラーでは、Fargate で必要な IP ターゲットを持つネットワークロードバランサをサポートします。詳細については、[the section called “アプリケーション負荷分散”](#)および[the section called “ネットワークロードバランサーを作成する”](#)を参照してください。

- [Cluster Autoscaler](#) がサポートされています。クラスター自動スケーラー Pod をデプロイする場合は、コマンドラインに --aws-use-static-instance-list=true が含まれていることを確認してください。詳細については、GitHub で「[静的インスタンス・リストを使用](#)」を参照してください。ワーカーノード VPC には、AWS STS VPC エンドポイントと自動スケーリング VPC エンドポイントも含める必要があります。
- 一部のコンテナソフトウェア製品では、AWS Marketplace Metering Service にアクセスする API コールを使用して、使用状況をモニタリングします。プライベートクラスターではこれらの呼び出しが許可されないため、これらのコンテナタイプはプライベートクラスターには使用できません。

EKS の Kubernetes バージョンライフサイクルを理解する

Kubernetes は、新しい機能、設計の更新、バグの修正と共に急速に進化しています。コミュニティでは、平均して 4 か月に一度、新しい Kubernetes のマイナーバージョン (1.32 など) をリリースしています。Amazon EKS はマイナーバージョンのアップストリームリリースと非推奨サイクルに従います。新しい Kubernetes バージョンが Amazon EKS で利用可能になったら、利用可能な最新のバージョンが使用できるように、クラスターをタイムリーに更新することをお勧めします。

マイナーバージョンはリリース後最初の 14 か月間は Amazon EKS の標準サポート対象となります。標準サポート終了日を過ぎたバージョンは次の 12 か月間の延長サポートに入ります。延長サポートでは、クラスター時間あたりの追加料金を支払えば、特定の Kubernetes バージョンを延長し

て使用することができます。延長サポート期間が終了する前にクラスターを更新しなかった場合、クラスターは現在サポートされている最も古い延長バージョンに自動的にアップグレードされます。

延長サポートはデフォルトで有効になっています。 [the section called “延長サポートを無効にする”](#)

Amazon EKS でサポートされている利用可能な最新の Kubernetes バージョンを使用してクラスターを作成することをお勧めします。アプリケーションに特定のバージョンの Kubernetes が必要な場合は、古いバージョンを選択できます。標準サポートまたは延長サポートが提供されているどのバージョンでも、新しい Amazon EKS クラスターを作成できます。

標準サポートで入手可能なバージョン

Amazon EKS 標準サポートでは、現在以下の Kubernetes バージョンを利用できます。

- 1.33
- 1.32
- 1.31
- 1.30

標準サポートの各バージョンで注意すべき重要な変更点については「[the section called “標準サポートバージョン”](#)」を参照してください。

延長サポートで利用可能なバージョン

Amazon EKS 延長サポートでは、現在以下の Kubernetes バージョンを利用できます。

- 1.29
- 1.28
- 1.27
- 1.26

延長サポートにおいて各バージョンで注意すべき重要な変更点については「[the section called “延長サポートバージョン”](#)」を参照してください。

Amazon EKS Kubernetes リリースカレンダー

次の表は、Kubernetes バージョンごとに検討すべき重要なリリース日とサポート日を示しています。延長サポートの請求はバージョンが標準サポートの終了日に達した日の初めに開始されます。

Note

月と年のみの日付はおおよその日付であり、確定後に正確な日付で更新されます。

この特定のドキュメントページへのすべてのソースファイルの変更の通知を受け取るには、RSS リーダーを使用して次の URL をサブスクライブできます。

<https://github.com/awsdocs/amazon-eks-user-guide/commits/mainline/latest/ug/clusters/kubernetes-versions.adoc.atom>

Kubernetes バージョン	アップストリームのリリース	Amazon EKS リリース	標準サポート終了日	延長サポートの終了日
1.33	2025 年 4 月 23 日	2025 年 5 月 29 日	2026 年 7 月 29 日	2027 年 7 月 29 日
1.32	2024 年 12 月 11 日	2025 年 1 月 23 日	2026 年 3 月 23 日	2027 年 3 月 23 日
1.31	2024 年 8 月 13 日	2024 年 9 月 26 日	2025 年 11 月 26 日	2026 年 11 月 26 日
1.30	2024 年 4 月 17 日	2024 年 5 月 23 日	2025 年 7 月 23 日	2026 年 7 月 23 日
1.29	2023 年 12 月 13 日	2024 年 1 月 23 日	2025 年 3 月 23 日	2026 年 3 月 23 日
1.28	2023 年 8 月 15 日	2023 年 9 月 26 日	2024 年 11 月 26 日	2025 年 11 月 26 日
1.27	2023 年 4 月 11 日	2023 年 5 月 24 日	2024 年 7 月 24 日	2025 年 7 月 24 日

Kubernetes バージョン	アップストリームのリリース	Amazon EKS リリース	標準サポート終了日	延長サポートの終了日
1.26	2022 年 12 月 9 日	2023 年 4 月 11 日	2024 年 6 月 11 日	2025 年 6 月 11 日

AWS CLI を使用してバージョン情報を取得する

AWS CLI を使用すると、標準サポートの終了日など、EKS で利用可能な Kubernetes バージョンに関する情報を取得できます。

AWS CLI を使用して EKS で利用可能な Kubernetes バージョンに関する情報を取得するには

1. ターミナルを開きます。
2. AWS CLI がインストールされ、設定されていることを確認します。詳細については「[CLI の最新バージョンのインストールまたはアップデート](#)」を参照してください。
3. 次のコマンドを実行してください。

```
aws eks describe-cluster-versions
```

4. このコマンドは使用可能なクラスターバージョンに関する詳細を含む JSON 出力を返します。以下に出力の例を示します。

```
{
  "clusterVersions": [
    {
      "clusterVersion": "1.31",
      "clusterType": "eks",
      "defaultPlatformVersion": "eks.21",
      "defaultVersion": true,
      "releaseDate": "2024-09-25T17:00:00-07:00",
      "endOfStandardSupportDate": "2025-11-25T16:00:00-08:00",
      "endOfExtendedSupportDate": "2026-11-25T16:00:00-08:00",
      "status": "STANDARD_SUPPORT",
      "kubernetesPatchVersion": "1.31.3"
    }
  ]
}
```

```
}
```

この出力にはクラスターバージョンごとに次の情報が表示されます。

- `clusterVersion`: EKS クラスターの Kubernetes バージョン
- `clusterType`: クラスターのタイプ (「eks」など)
- `defaultPlatformVersion`: デフォルトの EKS プラットフォームバージョン
- `defaultVersion`: これがデフォルトのバージョンであるかどうか
- `releaseDate`: このバージョンがリリースされた日付
- `endOfStandardSupportDate`: 標準サポートが終了する日付
- `endOfExtendedSupportDate`: 延長サポートが終了する日付
- `status`: STANDARD_SUPPORT や EXTENDED_SUPPORT などの、そのバージョンの現在のサポートステータス
- `kubernetesPatchVersion`: 特定の Kubernetes パッチバージョン

Amazon EKS のバージョンのよくある質問

標準サポートで利用できる Kubernetes バージョンはいくつありますか？

Kubernetes のバージョンに対する Kubernetes コミュニティのサポートに沿って、Amazon EKS では Kubernetes の 3 つのバージョンのサポートを提供するように努めています。特定の Kubernetes マイナーバージョンの標準サポートを終了する場合は、終了日の少なくとも 60 日前に発表します。Kubernetes の新しいバージョンの Amazon EKS における認定およびリリースプロセスにより、Amazon EKS での Kubernetes バージョンの標準サポート終了日は、Kubernetes プロジェクトがそのバージョンでアップストリームを停止した日より後になります。

Kubernetes は、Amazon EKS による標準サポートをどのくらいの期間受けられますか？

Kubernetes バージョンは、Amazon EKS で最初に利用可能になってから 14 か月間標準サポートを受けていました。これは、Amazon EKS で利用可能なバージョンが Kubernetes アップストリームでサポートされなくなった場合にも当てはまります。Amazon は、Amazon EKS でサポートされている Kubernetes の各バージョンに適用されるセキュリティパッチをバックポートしています。

Kubernetes バージョンのサポートが Amazon EKS で終了する際には通知が届きますか？

はい。アカウント内のいずれかのクラスターでサポートの終了が近い Kubernetes バージョンを実行している場合、そのバージョンが Amazon EKS でリリースされてから約 12 か月後に、Amazon EKS から AWS Health Dashboard を介して通知が送信されます。通知にはサポート終了日が含まれます。これは通知の日から 60 日以上後です。

Amazon EKS では、どの Kubernetes 機能がサポートされていますか？

Amazon EKS では、Kubernetes API の一般提供されている (GA) すべての機能がサポートされています。新しいベータ版 API はデフォルトではクラスターで有効になっていません。デフォルトでは既存のベータ API と既存のベータ API の新しいバージョンは引き続き有効になっています。アルファ機能はサポートされません。

Amazon EKS のマネージド型ノードグループはクラスターのコントロールプレーンのバージョンとともに自動的に更新されますか？

いいえ。マネージドノード型グループではアカウントに Amazon EC2 インスタンスを作成します。これらのインスタンスはお客様または Amazon EKS がコントロールプレーンを更新しても、自動的にアップグレードされません。詳細については、「[the section called “更新”](#)」を参照してください。コントロールプレーンとノードでは、Kubernetes バージョンを同じに維持することをお勧めします。

セルフマネージド型ノードグループはクラスターのコントロールプレーンのバージョンとともに自動的に更新されますか？

いいえ。セルフマネージド型ノードグループにはアカウント内の Amazon EC2 インスタンスが含まれています。これらのインスタンスはお客様または代わりに Amazon EKS がコントロールプレーンのバージョンを更新しても、自動的にアップグレードされません。セルフマネージド型ノードグループは更新が必要であることをコンソールに表示しません。更新が必要なノードを確認するにはクラスターの [概要] タブにある [ノード] のリストからノードを選択して、そこにインストールされている kubelet バージョンを表示します。ノードは手動で更新する必要があります。詳細については、「[the section called “更新方式”](#)」を参照してください。

Kubernetes では、最大 3 つ前のマイナーバージョンに対して、コントロールプレーンとノード間の互換性がテストされています。例えば、1.29 ノードは 1.32 コントロールプレーンによってオーケストレーションされた場合も動作し続けます。しかし、コントロールプレーンの背後で 3 つのマイナーバージョンのノードを使用し続けながら、クラスターを実行することはお勧めしません。詳細については、Kubernetes ドキュメントの「[Kubernetes のバージョンおよびバージョンスキューのサポートポリシー](#)」を参照してください。コントロールプレーンとノードでは、同一の Kubernetes バージョンを維持することをお勧めします。

クラスターコントロールプレーンのバージョンに対する自動アップグレードによって、Fargate で実行中の Pod も自動的にアップグレードされますか？

いいえ。Fargate Pod は Kubernetes デプロイなどのレプリケーションコントローラーの一部として実行することを強くお勧めします。その後、すべての Fargate Pod のローリング再起動を実行してください。Fargate Pod の新しいバージョンは、更新されたクラスターのコントロールプレーンのバージョンと同じ kubelet バージョンでデプロイされます。詳細については、Kubernetes ドキュメントの「[Deployments](#)」を参照してください。

 Important

コントロールプレーンを更新する場合は引き続き Fargate ノードを自分で更新する必要があります。Fargate ノードを更新するには、ノードと対応した Fargate Pod を削除した上で Pod を再デプロイします。新しい Pod は、クラスターと同じ kubelet バージョンでデプロイされます。

ハイブリッドノードではどの Kubernetes バージョンがサポートされていますか？

Amazon EKS Hybrid Nodes は標準および拡張 Kubernetes バージョンのサポートなど、他のノードコンピューティングタイプを持つ Amazon EKS クラスターと同じ Kubernetes バージョンをサポートします。コントロールプレーンのバージョンをアップグレードしても、ハイブリッドノードは自動的にアップグレードされません。また、ハイブリッドノードのアップグレードはお客様の責任となります。詳細については「[the section called “ハイブリッドノードをアップグレードする”](#)」を参照してください。

Amazon EKS 延長サポートに関するよくある質問

標準サポートと延長サポートという用語を初めて知りました。これらの用語はどのような意味ですか？

Amazon EKS での Kubernetes バージョンに対する標準サポートは、Amazon EKS で Kubernetes バージョンがリリースされた時点から始まり、リリース日の 14 か月後に終了します。Kubernetes バージョンの延長サポートは、標準サポートの終了後すぐに始まり、その 12 か月後に終了します。たとえば、バージョン 1.23 向けの Amazon EKS 標準サポートは 2023 年 10 月 11 日に終了します。バージョン 1.23 の延長サポートは 2023 年 10 月 12 日に開始され、2024 年 10 月 11 日に終了しました。

Amazon EKS クラスターの延長サポートを受けるには何をする必要がありますか？

クラスターのアップグレードポリシーを EXTENDED に変更して、クラスターの拡張サポート (「[the section called “延長サポートを有効にする”](#)」を参照) を有効にする必要があります。デフォルトではすべての新規および既存のクラスターについて、特に指定がない限り、アップグレードポリシーは EXTENDED に設定されています。クラスターのアップグレードポリシーを表示するには「[the section called “アップグレードポリシーを表示する”](#)」を参照してください。標準サポートは、Amazon EKS で Kubernetes バージョンがリリースされた時点で始まり、リリース日の 14 か月後に終了します。Kubernetes バージョンの延長サポートは、標準サポートの終了後すぐに始まり、その 12 か月後に終了します。

どの Kubernetes バージョンで延長サポートを受けることができますか？

クラスターはそのバージョンの標準サポートが終了してから最大 12 か月間、どのバージョンでも実行できます。これは Amazon EKS では各バージョンが 26 か月間サポートされることを意味します (14 か月間標準サポートと 12 か月間延長サポート)。

延長サポートを利用したくない場合はどうすればいいですか？

延長サポートに自動的に登録されないようにする場合は、クラスターを Amazon EKS 標準サポートの対象となる Kubernetes バージョンにアップグレードします。延長サポートを無効にする方法については「[the section called “延長サポートを無効にする”](#)」を参照してください。注意: 延長サポートを無効にすると、クラスターは標準サポートの終了時に自動的にアップグレードされます。

12 か月間の延長サポートが終了するとどうなりますか？

26 か月のライフサイクル (14 か月の標準サポートと 12 か月の延長サポート) が終了した Kubernetes バージョンで実行されているクラスターは、次のバージョンに自動的にアップグレードされます。自動アップグレードは Kubernetes コントロールプレーンのみが対象です。EKS 自動モード ノードがある場合、ノードは自動的に更新される場合があります。セルフマネージド型ノードと EKS マネージド型ノードグループは以前のバージョンのままになります。

延長サポート終了日には終了の対象となっているバージョンで新しい Amazon EKS クラスターを作成できなくなります。サポート終了日を過ぎると、既存のコントロールプレーンは後の段階的なデプロイプロセスを通じて Amazon EKS によりサポートされている最も初期のバージョンに自動的に更新されます。コントロールプレーンの自動更新後はクラスターアドオンと Amazon EC2 ノードを手動で更新してください。詳細については「[the section called “Kubernetes バージョンを更新する”](#)」を参照してください。

延長サポート終了後にコントロールプレーンが自動的に更新されるのは正確にはいつですか？

Amazon EKS での具体的なスケジュールは決まっています。自動更新は延長サポート終了日以降に任意のタイミングで実行される可能性があります。更新前には通知は届きません。Amazon EKS の自動更新プロセスに頼ることなく、事前にコントロールプレーンを更新することをお勧めします。詳細については、「[the section called “Kubernetes バージョンを更新する”](#)」を参照してください。

コントロールプレーンを 1 つの Kubernetes バージョンに恒久的に維持することはできますか？

いいえ。AWS ではクラウドのセキュリティを最優先事項ととらえています。特定の時点 (通常は 1 年) を過ぎると、Kubernetes コミュニティは共通脆弱性識別子 (CVE) パッチのリリースを停止し、サポートされていないバージョンの CVE 提出を非推奨とします。つまり、Kubernetes の古いバージョンに固有の脆弱性がある場合、報告すらされない可能性があります。クラスターは脆弱性の発生時にも通知および修復オプションなしで公開され続けることになります。このため、Amazon EKS では延長サポートが終了したバージョンでコントロールプレーンを維持することはできません。

延長サポートを受けるには追加料金がかかりますか？

はい。延長サポートで稼働している Amazon EKS クラスターに対して追加料金が発生します。料金の詳細については AWS ブログの「[Kubernetes バージョンの Amazon EKS 延長サポートの料金](#)」または「[料金ページ](#)」を参照してください。

延長サポートには何が含まれますか？

延長サポートの Amazon EKS クラスターは、Kubernetes コントロールプレーンのセキュリティパッチを継続的に受け取ります。また、Amazon EKS は延長サポート対象バージョンに対する Amazon VPC CNI、kube-proxy、CoreDNS アドオンのパッチをリリースします。また、Amazon EKS は Amazon Linux、Bottlerocket、および Windows 向けに AWS が公開した Amazon EKS に最適化された AMI のパッチと、それらのバージョンの Amazon EKS Fargate ノードのパッチもリリースします。延長サポートに該当するすべてのクラスターは引き続き AWS からテクニカルサポートを受けることができます。

延長サポートでは Kubernetes 以外のコンポーネントに適用できるパッチに制限はありますか？

延長サポートは AWS から Kubernetes 固有のすべてのコンポーネントを対象としていますが、常にサポートされるのは Amazon Linux、Bottlerocket、および Windows 向けに AWS が公開した Amazon EKS の最適化された AMI のみです。つまり、延長サポートを利用している間は Amazon EKS に最適化された AMI にはより新しいコンポーネント (OS やカーネルなど) が搭載される場合があります。たとえば、Amazon Linux 2 の[ライフサイクルが 2025 年に終了](#)すると、Amazon EKS に最適化された Amazon Linux AMI はより新しい Amazon Linux OS を使用して構築され

るようになります。Amazon EKS は、このようなサポートライフサイクルの重要な相違点を Kubernetes バージョンごとに発表し、文書化します。

延長サポートのバージョンを使用して新しいクラスターを作成できますか？

はい。

標準サポートの Kubernetes バージョンのリリースノートを確認する

このトピックでは、標準サポートで注意すべき重要な変更点を Kubernetes バージョンごとに説明します。アップグレードするときはクラスターの古いバージョンと新しいバージョン間で発生した変更を注意深く確認してください。

Kubernetes 1.33

Amazon EKS で Kubernetes 1.33 が利用可能になりました。Kubernetes1.33 の詳細については、[公式リリースのお知らせ](#)を参照してください。

Important

- 動的リソース割り当てで Kubernetes の beta API が有効になっています。
 - この beta API によって、GPU などのリソースを必要とするワークロードのスケジューリングとモニタリングのエクスペリエンスが向上します。
 - beta API は Kubernetes コミュニティによって定義され、今後の Kubernetes バージョンで変更される可能性があります。
 - Kubernetes ドキュメントの「[機能ステージ](#)」を注意深く確認し、beta API の使用による影響を理解してください。
- AWS では、EKS で最適化された Amazon Linux 2 AMI for Kubernetes 1.33 をリリースしていません。
 - AWS では、Amazon Linux 2023 への移行を推奨しています。「[the section called “AL2023 にアップグレードする”](#)」ではその方法を説明しています。
 - 詳細については、「[the section called “Amazon Linux 2 AMI の廃止”](#)」を参照してください。

- インプレースポッドリソースのサイズ変更 (ベータ): インプレースリソースのサイズ変更がベータに昇格し、再起動をせずに既存のポッドの CPU とメモリリソースを動的に更新できるようになり

ました。これにより、ダウンタイムなしでステートフルワークロードを垂直スケーリングし、トラフィックパターンに基づいてリソースをシームレスに調整できます。

- サイドカーコンテナの安定化: サイドカーコンテナは、アプリケーションコンテナの前に開始され、ポッドのライフサイクルを通して実行され、動作状態シグナリングのプロブをサポートする、restartPolicy: Always を使用する特別な init コンテナとしてサイドカーを実装することで、徐々に安定してきています。
- 詳細については、Kubernetes ドキュメントの「[サイドカーコンテナ](#)」を参照してください。
- エンドポイント API の廃止: エンドポイント API が正式に廃止され、アクセス時に警告が返されるようになりました。代わりに EndpointSlices API を使用するようにワークロードとスクリプトを移行してください。これにより、デュアルスタックネットワークなどの最新機能がサポートされ、サービスごとに複数の EndpointSlices が処理されます。
- 詳細については、Kubernetes ブログの「[Kubernetes v1.33: Continuing the transition from Endpoints to EndpointSlice](#)」を参照してください。
- Elastic Fabric Adapter のサポート: Amazon EKS クラスターのデフォルトのセキュリティグループで、Elastic Fabric Adapter (EFA) トラフィックをサポートするようになりました。デフォルトのセキュリティグループには、同じセキュリティグループの宛先に対して EFA トラフィックを許可する新しいアウトバウンドルールがあります。これにより、クラスター内での EFA トラフィックが可能になります。
- 詳細については、「Amazon Elastic Compute Cloud ユーザーガイド」の「[Amazon EC2 の AI/ML および HPC ワークロード用の Elastic Fabric Adapter](#)」を参照してください。

Kubernetes 1.33 の変更ログの詳細については、「<https://github.com/kubernetes/kubernetes/blob/master/CHANGELOG/CHANGELOG-1.33.md>」を参照してください。

Kubernetes 1.32

Amazon EKS で Kubernetes 1.32 が利用可能になりました。Kubernetes 1.32 の詳細については、[公式リリースのお知らせ](#)を参照してください。

Important

- FlowSchema および PriorityLevelConfiguration の flowcontrol.apiserver.k8s.io/v1beta3 API バージョンは、バージョン 1.32 で削除されました。これらの APIs を使用している場合はアップグレードする前に、サポートされている最新バージョンを使用するように設定を更新する必要があります。

- `ServiceAccount.metadata.annotations[kubernetes.io/enforce-mountable-secrets]` は、バージョン 1.32 で廃止され、今後の Kubernetes マイナーバージョンリリースで削除されます。マウントされたシークレットへのアクセスを分離するには、別の名前空間を使用することをお勧めします。
 - Kubernetes バージョン 1.32 は、Amazon EKS が Amazon Linux 2 (AL2) AMI をリリースする最後のバージョンです。バージョン 1.33 以降、Amazon EKS は引き続き Amazon Linux 2023 (AL2023) および Bottlerocket ベースの AMI をリリースします。
-
- Memory Manager 機能は、Kubernetes バージョン 1.32 で一般提供 (GA) ステータスに移行しました。この機能強化により、コンテナ化されたアプリケーションに対してより効率的で予測可能なメモリ割り当てが可能になり、特定のメモリ要件を持つワークロードに特に役立ちます。
 - StatefulSets によって作成された PersistentVolumeClaims (PVC) に自動クリーンアップ機能が含まれるようになりました。PVCs が不要になった場合は StatefulSet 更新およびノードメンテナンスオペレーション中にデータの永続性を維持しながら、自動的に削除されます。この機能により、ストレージ管理が簡素化され、クラスター内で孤立した PVC が発生するのを防ぐことができます。
 - カスタム リソース フィールド セレクター機能が導入され、開発者はカスタム リソースにフィールド セレクターを追加できるようになりました。この機能は、組み込みの Kubernetes オブジェクトで利用できるものと同じフィルタリング機能をカスタム リソースに提供し、より正確で効率的なリソース フィルタリングを可能にし、より優れた API 設計プラクティスを促進します。

Kubernetes 1.32 の変更ログの詳細については、「<https://github.com/kubernetes/kubernetes/blob/master/CHANGELOG/CHANGELOG-1.32.md>」を参照してください。

匿名認証の変更

Amazon EKS 以降 1.32、匿名認証は次の API サーバーのヘルスチェックエンドポイントに制限されています。

- `/healthz`
- `/livez`
- `/readyz`

system:unauthenticated ユーザーを使用した他のエンドポイントへのリクエストは HTTP 401 Unauthorized レスポンスを受け取ります。このセキュリティ強化は RBAC ポリシーの設定ミスが原因で発生する可能性のある意図しないクラスターアクセスを防ぐのに役立ちます。

 Note

public-info-viewer RBAC 役割は上記のヘルスチェックエンドポイントに引き続き適用されます。

Amazon Linux 2 AMI の廃止

Kubernetes バージョン 1.33 以降ではEKS は事前に構築された最適化された Amazon Linux 2 (AL2 Amazon マシンイメージ (AMIs) を提供しません。

AWS ではEKS 自動モード を採用するか、Amazon Linux 2023 (AL2023) や Bottlerocket などの最新のオペレーティングシステムに移行することを提案しています。

- [the section called “MNG から移行する”](#)
- [the section called “AL2023 にアップグレードする”](#)
- [the section called “Bottlerocket”](#)

 Note

この更新はEKS 最適化 AL2 AMIs に適用されます。オペレーティングシステム自体の詳細については [「Amazon Linux 2 のFAQs」](#) を参照してください。

Kubernetes 1.31

Amazon EKS で Kubernetes 1.31 が利用可能になりました。Kubernetes1.31 の詳細については、[公式リリースのお知らせ](#)を参照してください。

 Important

- 2017 年以降廃止された kubelet フラグ `--keep-terminated-pod-volumes` は、バージョン 1.31 リリースの一部として削除されました。この変更は終了したポッドボリュームが kubelet によって処理される方法に影響します。ノード設定でこのフラグを使用している場合はアップグレードする前にブートストラップスクリプトと起動テンプレートを更新してこのフラグを削除する必要があります。

- ベータ VolumeAttributesClass 機能ゲートと API リソースは、Amazon EKS バージョン 1.31 で有効になっています。この機能を使用すると、クラスターオペレーターは Amazon EBS CSI ドライバーなど、互換性のある CSI ドライバーによって管理される永続ボリューム (PVs) のミュータブルプロパティを変更できます。この機能を活用するにはお使いの CSI ドライバーが VolumeAttributesClass 機能をサポートしていることを確認します (Amazon EBS CSI ドライバーの場合はバージョン 1.35.0 以降にアップグレードして自動的に機能を有効にします)。VolumeAttributesClass オブジェクトを作成して、ボリュームタイプやスループットなどの必要なボリューム属性を定義し、永続的ボリュームクレーム (PVC) に関連付けることができます。詳細については[Kubernetes の公式ドキュメント](#) とご使用の CSI ドライバーのドキュメントを参照してください。
- Amazon EBS CSI ドライバーの詳細については「[the section called “Amazon EBS”](#)」を参照してください。
- [AppArmor](#) の Kubernetes サポートは安定するように段階的に拡張され、一般公開されました。この機能を使用すると、コンテナの securityContext で appArmorProfile.type フィールドを設定することで、AppArmor を使用してコンテナを保護できます。バージョン 1.30 より前の Kubernetes では、AppArmor は注釈によって制御されていました。バージョン 1.30 以降では、フィールドを使用して制御されます。この機能を活用するには注釈から移行し、appArmorProfile.type フィールドを使用してワークロードに互換性があることを確認することをお勧めします。
- PersistentVolume の最終フェーズ移行時間機能は、安定するように段階的に拡張され、Kubernetes バージョン 1.31 で一般公開されました。この機能では PersistentVolumeStatus に新しいフィールド .status.lastTransitionTime が導入され、PersistentVolume が最後に別のフェーズに移行したときのタイムスタンプが提供されます。この機能強化により、特にボリュームのライフサイクルを理解することが重要なシナリオでは PersistentVolumes の追跡と管理が向上します。

Kubernetes 1.31 の変更ログの詳細については、「<https://github.com/kubernetes/kubernetes/blob/master/CHANGELOG/CHANGELOG-1.31.md>」を参照してください。

Kubernetes 1.30

Amazon EKS で Kubernetes 1.30 が利用可能になりました。Kubernetes 1.30 の詳細については、[公式リリースのお知らせ](#)を参照してください。

⚠ Important

- Amazon EKS バージョン 1.30 以降では新しく作成されたマネージド型ノードグループはノードオペレーティングシステムとして自動的に Amazon Linux 2023 (AL2023) を使用するようデフォルトで設定されます。以前は新しいノードグループはデフォルトで Amazon Linux 2 (AL2) を使用するよう設定されていました。新しいノードグループを作成するときに AL2 を AMI タイプとして選択すれば、AL2 を引き続き使用できます。
 - AL2 から AL2023 への移行の詳細については、「[the section called “AL2023 にアップグレードする”](#)」を参照してください。
 - Amazon Linux の詳細については「Amazon Linux ユーザーガイド」の「[Comparing AL2 and AL2023](#)」を参照してください。
 - マネージドノードグループのオペレーティングシステムの指定の詳細については「[the section called “作成”](#)」を参照してください。
-
- Amazon EKS 1.30 では `topology.k8s.aws/zone-id` ラベルがワーカーノードに追加されます。アベイラビリティゾーン ID (AZ ID) を使用すると、アカウント間でリソースの場所を区別できます。詳細については「AWS RAM ユーザーガイド」の「[AWS リソースのアベイラビリティゾーン ID](#)」を参照してください。
 - 1.30 以降では新しく作成されたクラスターに適用された gp2 StorageClass リソースの default 注釈が Amazon EKS に含まれなくなりました。このストレージクラスを名前で参照する場合、これは影響しません。クラスターにデフォルトの StorageClass を設定する場合はアクションを実行する必要があります。gp2 という名前で StorageClass を参照する必要があります。バージョン 1.31.0 または `aws-ebs-csi-driver` add-on のインストール時に `defaultStorageClass.enabled` パラメータを `true` に設定することで、Amazon EBS が推奨するデフォルトのストレージクラスをデプロイすることもできます。
 - Amazon EKS クラスター IAM 役割に必要な最小 IAM ポリシーが変更されました。アクション `ec2:DescribeAvailabilityZones` は必須です。詳細については、「[the section called “クラスター IAM ロール”](#)」を参照してください。

Kubernetes 1.30 の変更ログの詳細については、「<https://github.com/kubernetes/kubernetes/blob/master/CHANGELOG/CHANGELOG-1.30.md>」を参照してください。

延長サポートの Kubernetes バージョンに関するリリースノートを確認する

Amazon EKS は、アップストリームでサポートされているよりも長く Kubernetes バージョンをサポートしており、Amazon EKS でリリースされてから 14 か月間は Kubernetes マイナーバージョンの標準サポートと、さらに 12 か月間のサポート (バージョンごとに合計 26 か月) は Kubernetes マイナーバージョンの拡張サポートを提供します。

このトピックでは延長サポートにおいて各 Kubernetes バージョンで注意すべき重要な変更点を説明します。アップグレードするときはクラスターの古いバージョンと新しいバージョン間で発生した変更を注意深く確認してください。

Kubernetes 1.29

Amazon EKS で Kubernetes 1.29 が利用可能になりました。Kubernetes 1.29 の詳細については、[公式リリースのお知らせ](#)を参照してください。

Important

- FlowSchema と PriorityLevelConfiguration の非推奨の `flowcontrol.apiserver.k8s.io/v1beta2` API バージョンは、Kubernetes バージョン 1.29 では提供されなくなりました。非推奨のベータ API グループを使用するマニフェストやクライアントソフトウェアがある場合は、バージョン 1.29 にアップグレードする前にこれらを変更する必要があります。
- ノードオブジェクトの `.status.kubeProxyVersion` フィールドは非推奨となり、Kubernetes プロジェクトでは今後のリリースでそのフィールドを削除することが提案されています。非推奨のフィールドは正確ではなく、従来は kubelet によって管理されていましたが、実際には kube-proxy バージョンを認識できず、kube-proxy が実行されているかどうかも判別できませんでした。クライアントソフトウェアでこのフィールドを使用している場合は停止してください。かかるフィールドの情報に信頼性はなく、現在は非推奨です。
- Kubernetes 1.29 で潜在的なアタックサーフェスを減らすため、LegacyServiceAccountTokenCleanUp 機能は以前に自動生成されたものの長期間 (デフォルトでは 1 年) 使用されていないシークレットベースのトークンに無効のラベルを付け、そのラベルが付いてからさらに長期間使用が試みられなかった場合は自動的に削除します (デフォルトではその後 1 年)。このようなトークンを識別するには以下を実行してください。

```
kubectl get cm kube-apiserver-legacy-service-account-token-tracking -n kube-system
```

Kubernetes 1.29 の変更ログの詳細については、「<https://github.com/kubernetes/kubernetes/blob/master/CHANGELOG/CHANGELOG-1.29.md#changelog-since-v1280>」を参照してください。

Kubernetes 1.28

Amazon EKS で Kubernetes 1.28 が利用可能になりました。Kubernetes 1.28 の詳細については、[公式リリースのお知らせ](#)を参照してください。

- Kubernetes v1.28 により、n-2 から n-3 へ、1 つのマイナーバージョンごとのコアノードとコントロールプレーンコンポーネントの間でサポートされるスキューが拡張されました。これにより、サポートされている最も古いマイナーバージョンのノードコンポーネント (kubelet および kube-proxy) が、サポートされている最新のマイナーバージョンのコントロールプレーンコンポーネント (kube-apiserver、kube-scheduler、kube-controller-manager、cloud-controller-manager) と連携できるようになりました。
- Pod GC Controller にあるメトリクス `force_delete_pods_total` および `force_delete_pod_errors_total` はポッドの強制削除をすべて考慮するよう拡張されました。ポッドが強制的に削除される理由が、ポッドが終了しているのか、孤立しているのか、サービス外ティントで終了しているのか、あるいは終了してスケジュールされていないのかを示す理由がメトリックに追加されます。
- PersistentVolume (PV) コントローラーは `storageClassName` が設定されておらずバインドされていない PersistentVolumeClaim にデフォルトの StorageClass を自動的に割り当てるように変更されました。さらに、API サーバー内の PersistentVolumeClaim アドミッション検証メカニズムが調整され、値が未設定の状態から実際の StorageClass 名に変更できるようになりました。

Kubernetes 1.28 の変更ログの詳細については、「<https://github.com/kubernetes/kubernetes/blob/master/CHANGELOG/CHANGELOG-1.28.md#changelog-since-v1270>」を参照してください。

Kubernetes 1.27

Amazon EKS で Kubernetes 1.27 が利用可能になりました。Kubernetes 1.27 の詳細については、[公式リリースのお知らせ](#)を参照してください。

⚠ Important

- アルファ `seccomp` アノテーション `seccomp.security.alpha.kubernetes.io/pod` と `container.seccomp.security.alpha.kubernetes.io` アノテーションのサポートは削除されました。アルファ `seccomp` アノテーションは 1.19 で非推奨となり、1.27 で削除されたことで、`seccomp` アノテーションを持つ Pods に `seccomp` フィールドが自動的に入力されなくなります。代わりに、Pods またはコンテナの `securityContext.seccompProfile` フィールドを使用して `seccomp` プロファイルを構成してください。クラスターで非推奨のアルファ `seccomp` アノテーションを使用しているかどうかを確認するには次のコマンドを実行してください。

```
kubectl get pods --all-namespaces -o json | grep  
-E 'seccomp.security.alpha.kubernetes.io/pod|  
container.seccomp.security.alpha.kubernetes.io'
```

- `kubelet` の `--container-runtime` コマンドライン引数が削除されました。1.24 以降、Amazon EKS のデフォルトのコンテナランタイムが `containerd` となっており、これによりコンテナランタイムを指定する必要がなくなりました。1.27 以降、Amazon EKS はブートストラップスクリプトに渡された `--container-runtime` 引数を無視します。ノードのブートストラッププロセス中のエラーを防ぐために、この引数を `--kubelet-extra-args` に渡さないことが重要です。すべてのノード作成ワークフローとビルドスクリプトから `--container-runtime` 引数を削除する必要があります。
- Kubernetes 1.27 の `kubelet` は、デフォルトの `kubeAPIQPS` を 50 に、`kubeAPIBurst` を 100 に増やしました。これらの機能強化により、`kubelet` がより大量の API クエリを処理できるようになり、応答時間とパフォーマンスが向上します。スケーリング要件により、Pods に対する需要が増加した場合、修正されたデフォルト値により、`kubelet` は増加したワークロードを効率的に管理できるようになります。その結果、Pod の起動が速くなり、クラスター操作がより効率的になります。
- よりきめ細かい Pod トポロジを使用して、`minDomain` などのポリシーを分散できます。このパラメータにより、Pods を分散させる必要のあるドメインの最小数を指定できます。`nodeAffinityPolicy` および `nodeTaintPolicy` を使用することで、Pod の分散を管理する際の粒度をさらに細かくすることができます。これはノードのアフィニティ、テイント、Pod の仕様の `topologySpreadConstraints` の `matchLabelKeys` フィールドによって異なります。これにより、ローリングアップグレード後の分散計算のために Pods を選択できます。

- Kubernetes StatefulSets では、その PersistentVolumeClaims (PVCs) の存続期間を制御する 1.27 の新しいポリシーメカニズムがベータ版に昇格しました。新しい PVC リテンションポリシーでは StatefulSet が削除されたとき、または StatefulSet 内のレプリカがスケールダウンされたときに、StatefulSet スペックテンプレートから生成された PVCs を自動的に削除するか、保持するかを指定できます。
- Kubernetes API サーバーの [goaway-chance](#) オプションを使用すると、接続がランダムに閉じることで、HTTP/2 クライアント接続が単一の API サーバーインスタンスにスタックされることを防ぐことができます。接続が閉じると、クライアントは再接続を試み、負荷分散の結果として別の API サーバーにアクセスする可能性があります。Amazon EKS バージョン 1.27 では goaway-chance フラグが有効になっています。Amazon EKS クラスターで実行されているワークロードで [HTTP GOAWAY](#) と互換性のないクライアントが使用されている場合は接続終了時に再接続することで GOAWAY を処理できるようにクライアントを更新することをお勧めします。

Kubernetes 1.27 の変更ログの詳細については、「<https://github.com/kubernetes/kubernetes/blob/master/CHANGELOG/CHANGELOG-1.27.md#changelog-since-v1260>」を参照してください。

Kubernetes 1.26

Amazon EKS で Kubernetes 1.26 が利用可能になりました。Kubernetes 1.26 の詳細については、[公式リリースのお知らせ](#)を参照してください。

Important

Kubernetes 1.26 では、CRI v1alpha2 をサポートしなくなりました。この結果、コンテナランタイムが CRI v1 をサポートしていない場合、kubelet はノードを登録しなくなります。つまり、Kubernetes 1.26 が containerd のマイナーバージョン 1.5 以前のバージョンをサポートしていないということでもあります。containerd を使用している場合は、ノードを Kubernetes 1.26 にアップグレードする前に containerd をバージョン 1.6.0 以降にアップグレードする必要があります。また、v1alpha2 のみをサポートする他のコンテナランタイムもアップグレードする必要があります。詳細についてはコンテナランタイムベンダーにお問い合わせください。Amazon Linux と Bottlerocket の AMI にデフォルトで含まれる containerd のバージョンは 1.6.6 です。

- Kubernetes 1.26 にアップグレードする前に、Amazon VPC CNI plugin for Kubernetes をバージョン 1.12 以降にアップグレードしてください。Amazon VPC CNI plugin for Kubernetes をバージョン 1.12 以降にアップグレードしないと、Amazon VPC CNI plugin for Kubernetes はク

ラッシュします。詳細については、「[the section called “Amazon VPC CNI”](#)」を参照してください。

- Kubernetes API サーバーの [goaway-chance](#) オプションを使用すると、接続がランダムに閉じることで、HTTP/2 クライアント接続が単一の API サーバーインスタンスにスタックされることを防ぐことができます。接続が閉じると、クライアントは再接続を試み、負荷分散の結果として別の API サーバーにアクセスする可能性があります。Amazon EKS バージョン 1.26 では goaway-chance フラグが有効になっています。Amazon EKS クラスターで実行されているワークロードで [HTTP GOAWAY](#) と互換性のないクライアントが使用されている場合は接続終了時に再接続することで GOAWAY を処理できるようにクライアントを更新することをお勧めします。

Kubernetes 1.26 の変更ログの詳細については、「<https://github.com/kubernetes/kubernetes/blob/master/CHANGELOG/CHANGELOG-1.26.md#changelog-since-v1250>」を参照してください。

現在のクラスターサポート期間を表示する

AWS コンソールの [クラスターサポート期間] セクションには、クラスターが現在、標準サポートであるか、または延長サポートであるかが表示されます。クラスターサポート期間が [延長サポート] である場合、EKS 延長サポートについて課金されています。

標準サポートおよび延長サポートの詳細については、「[the section called “Kubernetes バージョン”](#)」を参照してください。

1. AWS コンソールの EKS セクションの [クラスター] ページに移動します。コンソールが、確認するクラスターと同じ AWS リージョンに設定されていることを確認します。
2. [サポート期間] 列を確認します。値が [...まで標準サポート] である場合、現在、延長サポートについて課金されていません。標準サポート期間内です。値が [延長サポート...] である場合、このクラスターは現在、延長サポートについて課金されています。

Note

[サポート期間] は AWS API または CLI では取得できません。

現在のクラスターアップグレードポリシーを表示する

[クラスターのアップグレードポリシー] は、標準サポート期間を過ぎたクラスターがどうなるのかを決定します。アップグレードポリシーが EXTENDED である場合、クラスターは自動的にアップグ

レードされず、延長サポートに入ります。アップグレードポリシーが STANDARD である場合、自動的にアップグレードされます。

Kubernetes バージョンポリシーの Amazon EKS コントロールでは、EKS クラスターの標準サポート終了時の動作を選択できます。これらのコントロールを使用すると、Kubernetes バージョンの標準サポートの終了時に、どのクラスターが延長サポートに入るか、およびどのクラスターを自動的にアップグレードするかを決定できます。

マイナーバージョンはリリース後最初の 14 か月間は アマゾン EKS の標準サポート対象となります。標準サポート終了日を過ぎたバージョンは次の 12 か月間の延長サポートに入ります。延長サポートでは、クラスター時間あたりの追加料金を支払えば、特定の Kubernetes バージョンを延長して使用することができます。EKS クラスターの延長サポートを有効または無効にできます。延長サポートを無効にすると、AWS は標準サポートの終了時にクラスターを次のバージョンに自動的にアップグレードします。延長サポートを有効にすると、一定期間、追加料金で最新バージョンを維持できます。延長サポートを使用している場合でも、Kubernetes クラスターを定期的にアップグレードする計画を立ててください。

supportType プロパティを使用して、新規クラスターと既存のクラスターの両方のバージョンポリシーを設定できます。バージョンサポートポリシーの設定に使用できるオプションは 2 つあります：

- **STANDARD** — 標準サポートの終了時に自動アップグレードの対象となる EKS クラスター。この設定では延長サポート料金は発生しませんが、EKS クラスターは標準サポートで次にサポートされている Kubernetes バージョンに自動的にアップグレードされます。
- **EXTENDED** — Kubernetes バージョンが標準サポートの終了に達すると、EKS クラスターは延長サポートに入ります。この設定では、延長サポート料金が発生します。クラスターをサポートされている標準 Kubernetes バージョンにアップグレードして、延長サポート料金の発生を停止できます。延長サポートで実行されているクラスターは、延長サポートの終了時に自動アップグレードの対象となります。

延長サポートは、新しいクラスターと既存のクラスターのためにデフォルトで有効になっています。AWS Management Console で、または AWS CLI を使用して、クラスターのために延長サポートが有効になっているかどうかを確認できます。

⚠ Important

クラスターを現在の Kubernetes バージョンのままにして延長サポート期間を利用するには、標準サポート期間の終了前に延長サポートアップグレードポリシーを有効にする必要があります。

クラスターのバージョンサポートポリシーは、標準サポートの Kubernetes バージョンで実行している間に設定できます。バージョンが延長サポートに入ると、標準サポートのバージョンで実行されるまで、この設定を変更することはできなくなります。

例えば、バージョンサポートポリシーを standard として設定した場合、クラスターで実行されている Kubernetes バージョンが標準サポートの終了に達した後は、この設定を変更することはできなくなります。バージョンサポートポリシーを extended として設定した場合、クラスターで実行されている Kubernetes バージョンが標準サポートの終了に達した後は、この設定を変更することはできなくなります。バージョンサポートポリシーの設定を変更するには、クラスターがサポートされている標準の Kubernetes バージョンで実行されている必要があります。

クラスターアップグレードポリシーを表示する (AWS コンソール)

1. AWS コンソールの EKS セクションの [クラスター] ページに移動します。コンソールが、確認するクラスターと同じ AWS リージョンに設定されていることを確認します。
2. [アップグレードポリシー] 列を確認します。値が [標準サポート] である場合、クラスターは延長サポートに入りません。値が [延長サポート] である場合、クラスターは延長サポートに入ります。

クラスターアップグレードポリシーを表示する (AWS CLI)

1. AWS CLI がインストールされ、ログインしていることを確認します。 [AWS CLI を更新およびインストールする方法について説明します。](#)
2. EKS クラスターの名前を決定します。CLI を EKS クラスターと同じ AWS リージョンに設定します。
3. 次のコマンドを実行してください。

```
aws eks describe-cluster \
  --name <cluster-name> \
  --query "cluster.upgradePolicy.supportType"
```

4. 値が [STANDARD] である場合、クラスターは延長サポートに入りません。値が [EXTENDED] である場合、クラスターは延長サポートに入ります。

EKS 延長サポートを有効にして、Kubernetes バージョンアップグレードの計画にさらなる柔軟性をもたらす

このトピックは、EKS クラスターのアップグレードポリシーを設定して延長サポートを有効にする方法について説明します。EKS クラスターのアップグレードポリシーは、クラスターが標準サポート期間の終了に達したときにどうなるのかを決定します。クラスターアップグレードポリシーで延長サポートが有効になっている場合、標準サポート期間が終了するのと同時に、延長サポート期間に入ります。クラスターは、標準サポート期間の終了時に自動的にアップグレードされません。

実際に延長サポート期間中のクラスターでは、より高額なコストが発生します。延長サポートを有効にするように設定されたアップグレードポリシーがクラスターにあるが、標準サポート期間中であれば、標準コストが発生します。

AWS コンソールでクラスターを作成すると、延長サポートを無効にするようにアップグレードポリシーが設定されます。別の方法でクラスターを作成すると、延長サポートを有効にするようにアップグレードポリシーが設定されます。例えば、AWS API で作成されたクラスターでは、拡張サポートが有効になっています。

アップグレードポリシーの詳細については、「[the section called “アップグレードポリシーを表示する”](#)」を参照してください。

Important

クラスターを現在の Kubernetes バージョンのままにして延長サポート期間を利用するには、標準サポート期間の終了前に延長サポートアップグレードポリシーを有効にする必要があります。

延長サポートを有効にしない場合、クラスターは自動的にアップグレードされます。

EKS 延長サポートを有効にする (AWS コンソール)

1. AWS コンソールで EKS クラスターに移動します。[クラスター情報] ページの [概要] タブを選択します。
2. [Kubernetes バージョン設定] セクションで、[管理] を選択します。

3. [延長サポート] を選択し、[変更を保存] します。

EKS 延長サポートを有効にする (AWS CLI)

1. AWS CLI がインストールされ、ログインしていることを確認します。 [AWS CLI を更新およびインストールする方法について説明します。](#)
2. EKS クラスターの名前を決定します。
3. 次のコマンドを実行してください。

```
aws eks update-cluster-config \
--name <cluster-name> \
--upgrade-policy supportType=EXTENDED
```

EKS 延長サポートを無効にすることにより、クラスターのコストが増えないようにする

このトピックでは、EKS クラスターのアップグレードポリシーを設定して延長サポートを無効にする方法について説明します。EKS クラスターのアップグレードポリシーは、クラスターが標準サポート期間の終了に達したときにどうなるのかを決定します。クラスターアップグレードポリシーで延長サポートが無効になっている場合、自動的に次の Kubernetes バージョンにアップグレードされます。

アップグレードポリシーの詳細については、「[the section called “アップグレードポリシーを表示する”](#)」を参照してください。

Important

クラスターが延長サポート期間に入った後は、延長サポートを無効にすることはできません。クラスターの延長サポートは、標準サポートでのみ無効にできます。

AWS では、クラスターを標準サポート期間内のバージョンにアップグレードすることをお勧めしています。

EKS 延長サポートを無効にする (AWS コンソール)

1. AWS コンソールで EKS クラスターに移動します。[クラスター情報] ページの [概要] タブを選択します。

2. [Kubernetes バージョン設定] セクションで、[管理] を選択します。
3. [標準サポート] を選択し、[変更を保存] します。

EKS 延長サポートを無効にする (AWS CLI)

1. AWS CLI がインストールされ、ログインしていることを確認します。 [AWS CLI を更新およびインストールする方法について説明します。](#)
2. EKS クラスターの名前を決定します。
3. 次のコマンドを実行してください。

```
aws eks update-cluster-config \
  --name <cluster-name> \
  --upgrade-policy supportType=STANDARD
```

各 Kubernetes バージョンの Amazon EKS プラットフォームバージョンを表示する

Amazon EKS プラットフォームのバージョンは、Amazon EKS クラスターコントロールプレーンの各機能 (有効な Kubernetes API サーバーのフラグや、現在の Kubernetes パッチバージョンなど) を表しています。各 Kubernetes マイナーバージョンには、Amazon EKS プラットフォームのバージョンが 1 つ以上関連付けられています。別の Kubernetes マイナーバージョンのプラットフォームバージョンは独立しています。 [クラスターの現在のプラットフォームバージョンを取得する](#) には AWS CLI または AWS Management Console を使用します。AWS アウトポスト にローカルクラスターがある場合、このトピックの代わりに「[the section called “EKS プラットフォームのバージョン”](#)」を参照してください。

新しい Kubernetes マイナーバージョン (1.32 など) が Amazon EKS で使用可能になると、その Kubernetes マイナーバージョンでの Amazon EKS プラットフォームのバージョンが `eks.1` から始まります。また、Amazon EKS では、新しいプラットフォームバージョンを定期的にリリースすることで、新しい Kubernetes コントロールプレーン設定を有効化し、セキュリティ修正プログラムを提供します。

Amazon EKS プラットフォームのバージョンで、新しいマイナーバージョンが使用可能になった場合は、以下が行われます。

- Amazon EKS プラットフォームのバージョン番号がインクリメントされます (`eks.<n+1>`)。

- Amazon EKS は、既存のすべてのクラスターを、対応する Kubernetes マイナーバージョン用の最新の Amazon EKS プラットフォームバージョンに自動的にアップグレードします。既存の Amazon EKS プラットフォームバージョンの自動アップグレードは、段階的にロールアウトされます。役割アウトプロセスには時間がかかる場合があります。最新の Amazon EKS プラットフォームバージョンの機能がすぐに必要な場合は新しい Amazon EKS クラスターを作成する必要があります。

クラスターが現在のプラットフォームバージョンより 2 バージョン以上遅れている場合、Amazon EKS がクラスターを自動的に更新できなかった可能性があります。この原因の詳細については「[the section called “Amazon EKS プラットフォームのバージョンは現在のプラットフォームバージョンより 2 つ以上遅れています”](#)」を参照してください。

- Amazon EKS は対応するパッチバージョンを適用して新しいノード AMI を発行する可能性があります。ただし、すべてのパッチバージョンは、特定の Kubernetes マイナーバージョンにおいて、EKS コントロールプレーンとノード AMI との間で互換性を持ちます。

新しい Amazon EKS プラットフォームバージョンにより、重大な変更が発生したり、サービスが中断されたりすることはありません。

クラスターは常に、指定された Kubernetes バージョン用に入手可能な、最新の Amazon EKS プラットフォームバージョン (eks.<n>) で作成されます。クラスターを新しい Kubernetes マイナーバージョンに更新した場合、そのクラスターは、更新した Kubernetes マイナーバージョンでの、現在の Amazon EKS プラットフォームバージョンを受け取ります。

最新および間近にリリースされた、Amazon EKS プラットフォームのバージョンを以下の表に示します。

Note

AWS は2024 年 6 月に公開された一部のプラットフォームバージョンを最近無効にしました。該当のプラットフォームバージョンには安定性の問題がありました。アクションは不要です。

この特定のドキュメントページへのすべてのソースファイルの変更の通知を受け取るには、RSS リーダーを使用して次の URL をサブスクライブできます。

```
https://github.com/awsdocs/amazon-eks-user-guide/commits/mainline/latest/ug/clusters/platform-versions.adoc.atom
```

Kubernetes バージョン 1.33

次の受付コントローラーはすべての 1.33 プラットフォームバージョンで有効です:

NodeRestriction、ExtendedResourceToleration、NamespaceLifecycle、LimitRanger、Serv

Kubernetes バージョ ン	EKS プラットフォー ムのバージョン	リリースノート	リリース日
1.33.1	eks.5	セキュリティの修正 および機能強化が行 われた新しいプラッ トフォームバージョ ン。	2025 年 6 月 11 日
1.33.1	eks.4	EKS 用 Kubernte s バージョン 1.33 の初回リリース。 詳細については、 「 the section called “Kubernetes 1.33” 」 を参照してくださ い。	2025 年 5 月 28 日

Kubernetes バージョン 1.32

次の受付コントローラーはすべての 1.32 プラットフォームバージョンで有効です:

NodeRestriction、ExtendedResourceToleration、NamespaceLifecycle、LimitRanger、Serv

Kubernetes バージョ ン	EKS プラットフォー ムのバージョン	リリースノート	リリース日
1.32.5	eks.12	セキュリティの修正 および機能強化が行 われた新しいプラッ トフォームバージョ ン。	2025 年 6 月 11 日

Kubernetes バージョン	EKS プラットフォームのバージョン	リリースノート	リリース日
1.32.5	eks.11	セキュリティの修正 および機能強化が行 われた新しいプラッ トフォームバージョ ン。	2025 年 5 月 30 日
1.32.3	eks.10	セキュリティの修正 および機能強化が行 われた新しいプラッ トフォームバージョ ン。	2025 年 5 月 16 日
1.32.3	eks.9	セキュリティの修正 および機能強化が行 われた新しいプラッ トフォームバージョ ン。	2025 年 4 月 29 日
1.32.3	eks.8	セキュリティの修正 および機能強化が行 われた新しいプラッ トフォームバージョ ン。	2025 年 4 月 18 日
1.32.3	eks.7	セキュリティの修正 および機能強化が行 われた新しいプラッ トフォームバージョ ン。	2025 年 4 月 18 日
1.32.3	eks.6	セキュリティの修正 および機能強化が行 われた新しいプラッ トフォームバージョ ン。	2025 年 4 月 2 日

Kubernetes バージョン	EKS プラットフォームのバージョン	リリースノート	リリース日
1.32.2	eks.5	セキュリティの修正および機能強化が行われた新しいプラットフォームバージョン。	2025 年 3 月 17 日
1.32.2	eks.4	セキュリティの修正および機能強化が行われた新しいプラットフォームバージョン。	2025 年 3 月 4 日
1.32.1	eks.3	セキュリティの修正および機能強化が行われた新しいプラットフォームバージョン。	2025 年 2 月 24 日
1.32.0	eks.2	EKS 用 Kubernetes バージョン 1.32 の初回リリース。 詳細については、 「the section called “Kubernetes 1.32”」 を参照してください。	2025 年 1 月

Kubernetes バージョン 1.31

次の受付コントローラーはすべての 1.31 プラットフォームバージョンで有効です:

NodeRestriction、ExtendedResourceToleration、NamespaceLifecycle、LimitRanger、Serv

Kubernetes バージョン	EKS プラットフォームのバージョン	リリースノート	リリース日
1.31.9	eks.28	セキュリティの修正 および機能強化が行 われた新しいプラッ トフォームバージョ ン。	2025 年 6 月 11 日
1.31.9	eks.27	セキュリティの修正 および機能強化が行 われた新しいプラッ トフォームバージョ ン。	2025 年 5 月 30 日
1.31.7	eks.26	セキュリティの修正 および機能強化が行 われた新しいプラッ トフォームバージョ ン。	2025 年 5 月 16 日
1.31.7	eks.25	セキュリティの修正 および機能強化が行 われた新しいプラッ トフォームバージョ ン。	2025 年 4 月 29 日
1.31.7	eks.24	セキュリティの修正 および機能強化が行 われた新しいプラッ トフォームバージョ ン。	2025 年 4 月 18 日
1.31.7	eks.23	セキュリティの修正 および機能強化が行 われた新しいプラッ	2025 年 4 月 18 日

Kubernetes バージョン	EKS プラットフォームのバージョン	リリースノート	リリース日
		トフォームバージョン。	
1.31.7	eks.22	セキュリティの修正および機能強化が行われた新しいプラットフォームバージョン。	2025 年 4 月 2 日
1.31.6	eks.21	セキュリティの修正および機能強化が行われた新しいプラットフォームバージョン。	2025 年 3 月 17 日
1.31.6	eks.20	セキュリティの修正および機能強化が行われた新しいプラットフォームバージョン。	2025 年 3 月 4 日
1.31.5	eks.19	セキュリティの修正および機能強化が行われた新しいプラットフォームバージョン。	2025 年 2 月 24 日
1.31.5	eks.18	セキュリティの修正および機能強化が行われた新しいプラットフォームバージョン。	2025 年 2 月 24 日

Kubernetes バージョン	EKS プラットフォームのバージョン	リリースノート	リリース日
1.31.4	eks.17	セキュリティの修正 および機能強化が行 われた新しいプラッ トフォームバージョ ン。	2025年1月17日
1.31.2	eks.12	Amazon EKS Hybrid Nodes のサポート とコントロールプ レーンのオブザーバ ビリティの強化機 能を備えた新しい プラットフォーム バージョン。「 the section called “ハイ ブリッドノード” 」 と「 Amazon EKS enhances performan ce observability 」をそ れぞれ参照してくだ さい。	2024 年 11 月 15 日
1.31.1	eks.6	セキュリティの修正 および機能強化が行 われた新しいプラッ トフォームバージョ ン。	2024 年 10 月 21 日

Kubernetes バージョン	EKS プラットフォームのバージョン	リリースノート	リリース日
1.31.0	eks.4	EKS 用 Kubernetes バージョン 1.31 の初回リリース。 詳細については、「 the section called “Kubernetes 1.31” 」を参照してください。	2024 年 9 月 26 日

Kubernetes バージョン 1.30

次の受付コントローラーはすべての 1.30 プラットフォームバージョンで有効です:

NodeRestriction、ExtendedResourceToleration、NamespaceLifecycle、LimitRanger、ServiceAccountTokenProjection

Kubernetes バージョン	EKS プラットフォームのバージョン	リリースノート	リリース日
1.30.13	eks.36	セキュリティの修正および機能強化が行われた新しいプラットフォームバージョン。	2025 年 6 月 11 日
1.30.13	eks.35	セキュリティの修正および機能強化が行われた新しいプラットフォームバージョン。	2025 年 5 月 30 日
1.30.11	eks.34	セキュリティの修正および機能強化が行われた新しいプラットフォームバージョン。	2025 年 5 月 16 日

Kubernetes バージョン	EKS プラットフォームのバージョン	リリースノート	リリース日
		トフォームバージョン。	
1.30.11	eks.33	セキュリティの修正および機能強化が行われた新しいプラットフォームバージョン。	2025 年 4 月 29 日
1.30.11	eks.32	セキュリティの修正および機能強化が行われた新しいプラットフォームバージョン。	2025 年 4 月 18 日
1.30.11	eks.31	セキュリティの修正および機能強化が行われた新しいプラットフォームバージョン。	2025 年 4 月 18 日
1.30.11	eks.30	セキュリティの修正および機能強化が行われた新しいプラットフォームバージョン。	2025 年 4 月 2 日
1.30.10	eks.29	セキュリティの修正および機能強化が行われた新しいプラットフォームバージョン。	2025 年 3 月 17 日

Kubernetes バージョン	EKS プラットフォームのバージョン	リリースノート	リリース日
1.30.10	eks.28	セキュリティの修正 および機能強化が行 われた新しいプラッ トフォームバージョ ン。	2025 年 3 月 4 日
1.30.9	eks.27	セキュリティの修正 および機能強化が行 われた新しいプラッ トフォームバージョ ン。	2025 年 2 月 24 日
1.30.8	eks.25	セキュリティの修正 および機能強化が行 われた新しいプラッ トフォームバージョ ン。	2025年1月17日
1.30.8	eks.24	セキュリティの修正 および機能強化が行 われた新しいプラッ トフォームバージョ ン。	2025 年 1 月 3 日
1.30.7	eks.23	セキュリティの修正 および機能強化が行 われた新しいプラッ トフォームバージョ ン。	2024 年 12 月 13 日
1.30.6	eks.22	セキュリティの修正 および機能強化が行 われた新しいプラッ トフォームバージョ ン。	2024 年 12 月 13 日

Kubernetes バージョン	EKS プラットフォームのバージョン	リリースノート	リリース日
1.30.6	eks.21	セキュリティの修正および機能強化が行われた新しいプラットフォームバージョン。	2024 年 12 月 13 日
1.30.6	eks.20	Amazon EKS Hybrid Nodes のサポートとコントロールプレーンのオブザーバビリティの強化機能を備えた新しいプラットフォームバージョン。「 the section called “ハイブリッドノード” 」と「 Amazon EKS enhances performance observability 」をそれぞれ参照してください。	2024 年 11 月 15 日
1.30.5	eks.12	セキュリティの修正および機能強化が行われた新しいプラットフォームバージョン。	2024 年 10 月 21 日
1.30.4	eks.8	セキュリティの修正および機能強化が行われた新しいプラットフォームバージョン。	2024 年 9 月 3 日

Kubernetes バージョン	EKS プラットフォームのバージョン	リリースノート	リリース日
1.30.3	eks.7	セキュリティの修正 および機能強化が行 われた新しいプラッ トフォームバージョ ン。	2024 年 8 月 28 日
1.30.3	eks.6	セキュリティの修正 および機能強化が行 われた新しいプラッ トフォームバージョ ン。	2024 年 8 月 9 日
1.30.2	eks.5	セキュリティの修正 および機能強化が行 われた新しいプラッ トフォームバージョ ン。	2024 年 7 月 2 日
1.30.0	eks.2	EKS 用 Kuberne te s バージョン 1.30 の初回リリース。 詳細については、 「 the section called “Kubernetes 1.30” 」 を参照してくださ い。	2024 年 5 月 23 日

Kubernetes バージョン 1.29

次の受付コントローラーはすべての 1.29 プラットフォームバージョンで有効です:
NodeRestriction、ExtendedResourceToleration、NamespaceLifecycle、LimitRanger、Serv

Kubernetes バージョン	EKS プラットフォームのバージョン	リリースノート	リリース日
1.29.15	eks.39	セキュリティの修正 および機能強化が行 われた新しいプラッ トフォームバージョ ン。	2025 年 6 月 11 日
1.29.15	eks.38	セキュリティの修正 および機能強化が行 われた新しいプラッ トフォームバージョ ン。	2025 年 5 月 30 日
1.29.15	eks.37	セキュリティの修正 および機能強化が行 われた新しいプラッ トフォームバージョ ン。	2025 年 5 月 16 日
1.29.15	eks.36	セキュリティの修正 および機能強化が行 われた新しいプラッ トフォームバージョ ン。	2025 年 4 月 29 日
1.29.15	eks.35	セキュリティの修正 および機能強化が行 われた新しいプラッ トフォームバージョ ン。	2025 年 4 月 18 日
1.29.15	eks.34	セキュリティの修正 および機能強化が行 われた新しいプラッ	2025 年 4 月 18 日

Kubernetes バージョン	EKS プラットフォームのバージョン	リリースノート	リリース日
		トフォームバージョン。	
1.29.15	eks.33	セキュリティの修正および機能強化が行われた新しいプラットフォームバージョン。	2025 年 4 月 2 日
1.29.14	eks.32	セキュリティの修正および機能強化が行われた新しいプラットフォームバージョン。	2025 年 3 月 17 日
1.29.14	eks.31	セキュリティの修正および機能強化が行われた新しいプラットフォームバージョン。	2025 年 3 月 4 日
1.29.13	eks.30	セキュリティの修正および機能強化が行われた新しいプラットフォームバージョン。	2025 年 2 月 24 日
1.29.13	eks.29	セキュリティの修正および機能強化が行われた新しいプラットフォームバージョン。	2025 年 2 月 24 日

Kubernetes バージョン	EKS プラットフォームのバージョン	リリースノート	リリース日
1.29.12	eks.28	セキュリティの修正 および機能強化が行 われた新しいプラッ トフォームバージョ ン。	2025 年 1 月 20 日
1.29.12	eks.27	セキュリティの修正 および機能強化が行 われた新しいプラッ トフォームバージョ ン。	2025 年 1 月 3 日
1.29.11	eks.26	セキュリティの修正 および機能強化が行 われた新しいプラッ トフォームバージョ ン。	2024 年 12 月 13 日
1.29.10	eks.25	セキュリティの修正 および機能強化が行 われた新しいプラッ トフォームバージョ ン。	2024 年 12 月 13 日
1.29.10	eks.24	セキュリティの修正 および機能強化が行 われた新しいプラッ トフォームバージョ ン。	2024 年 12 月 13 日

Kubernetes バージョン	EKS プラットフォームのバージョン	リリースノート	リリース日
1.29.10	eks.23	Amazon EKS Hybrid Nodes のサポートとコントロールプレーンのオブザーバビリティの強化機能を備えた新しいプラットフォームバージョン。「 the section called “ハイブリッドノード” 」と「 Amazon EKS enhances performance observability 」をそれぞれ参照してください。	2024 年 11 月 15 日
1.29.9	eks.17	セキュリティの修正および機能強化が行われた新しいプラットフォームバージョン。	2024 年 10 月 21 日
1.29.8	eks.13	セキュリティの修正および機能強化が行われた新しいプラットフォームバージョン。	2024 年 9 月 3 日
1.29.7	eks.12	セキュリティの修正および機能強化が行われた新しいプラットフォームバージョン。	2024 年 8 月 28 日

Kubernetes バージョン	EKS プラットフォームのバージョン	リリースノート	リリース日
1.29.7	eks.11	セキュリティの修正および機能強化が行われた新しいプラットフォームバージョン。	2024 年 8 月 9 日
1.29.6	eks.10	セキュリティの修正および機能強化が行われた新しいプラットフォームバージョン。	2024 年 7 月 2 日
1.29.4	eks.7	CoreDNS 自動スケーリング、セキュリティの修正および機能強化が行われた、新しいプラットフォームバージョン。CoreDNS 自動スケーリングの詳細については「 the section called “をスケールして DNS トラフィックを増やす” 」を参照してください。	2024 年 5 月 16 日
1.29.3	eks.6	セキュリティの修正および機能強化が行われた新しいプラットフォームバージョン。	2024 年 4 月 18 日

Kubernetes バージョン	EKS プラットフォームのバージョン	リリースノート	リリース日
1.29.1	eks.5	セキュリティの修正および機能強化が行われた新しいプラットフォームバージョン。	2024 年 3 月 29 日
1.29.1	eks.4	セキュリティの修正および機能強化が行われた新しいプラットフォームバージョン。	2024 年 3 月 20 日
1.29.1	eks.3	セキュリティの修正および機能強化が行われた新しいプラットフォームバージョン。	2024 年 3 月 12 日
1.29.0	eks.1	EKS 用 Kubernetes バージョン 1.29 の初回リリース。 詳細については、「 the section called “Kubernetes 1.29” 」を参照してください。	2024 年 1 月 23 日

Kubernetes バージョン 1.28

次の受付コントローラーはすべての 1.28 プラットフォームバージョンで有効です:

NodeRestriction、ExtendedResourceToleration、NamespaceLifecycle、LimitRanger、Serv

Kubernetes バージョン	EKS プラットフォームのバージョン	リリースノート	リリース日
1.28.15	eks.45	セキュリティの修正 および機能強化が行 われた新しいプラッ トフォームバージョ ン。	2025 年 6 月 11 日
1.28.15	eks.44	セキュリティの修正 および機能強化が行 われた新しいプラッ トフォームバージョ ン。	5 月 30 日 2025 年
1.28.15	eks.43	セキュリティの修正 および機能強化が行 われた新しいプラッ トフォームバージョ ン。	2025 年 5 月 16 日
1.28.15	eks.42	セキュリティの修正 および機能強化が行 われた新しいプラッ トフォームバージョ ン。	2025 年 4 月 29 日
1.28.15	eks.41	セキュリティの修正 および機能強化が行 われた新しいプラッ トフォームバージョ ン。	2025 年 4 月 18 日
1.28.15	eks.40	セキュリティの修正 および機能強化が行 われた新しいプラッ	2025 年 4 月 18 日

Kubernetes バージョン	EKS プラットフォームのバージョン	リリースノート	リリース日
		トフォームバージョン。	
1.28.15	eks.39	セキュリティの修正および機能強化が行われた新しいプラットフォームバージョン。	2025 年 4 月 2 日
1.28.15	eks.38	セキュリティの修正および機能強化が行われた新しいプラットフォームバージョン。	2025 年 3 月 17 日
1.28.15	eks.37	セキュリティの修正および機能強化が行われた新しいプラットフォームバージョン。	2025 年 3 月 4 日
1.28.15	eks.36	セキュリティの修正および機能強化が行われた新しいプラットフォームバージョン。	2025 年 2 月 24 日
1.28.15	eks.34	セキュリティの修正および機能強化が行われた新しいプラットフォームバージョン。	2025年1月17日

Kubernetes バージョン	EKS プラットフォームのバージョン	リリースノート	リリース日
1.28.15	eks.33	セキュリティの修正 および機能強化が行 われた新しいプラッ トフォームバージョ ン。	2025 年 1 月 3 日
1.28.15	eks.32	セキュリティの修正 および機能強化が行 われた新しいプラッ トフォームバージョ ン。	2024 年 12 月 13 日
1.28.15	eks.31	セキュリティの修正 および機能強化が行 われた新しいプラッ トフォームバージョ ン。	2024 年 12 月 13 日
1.28.15	eks.30	セキュリティの修正 および機能強化が行 われた新しいプラッ トフォームバージョ ン。	2024 年 12 月 13 日

Kubernetes バージョン	EKS プラットフォームのバージョン	リリースノート	リリース日
1.28.15	eks.29	Amazon EKS Hybrid Nodes のサポートとコントロールプレーンのオブザーバビリティの強化機能を備えた新しいプラットフォームバージョン。「 the section called “ハイブリッドノード” 」と「 Amazon EKS enhances performance observability 」をそれぞれ参照してください。	2024 年 11 月 15 日
1.28.14	eks.23	セキュリティの修正および機能強化が行われた新しいプラットフォームバージョン。	2024 年 10 月 21 日
1.28.13	eks.19	セキュリティの修正および機能強化が行われた新しいプラットフォームバージョン。	2024 年 9 月 3 日
1.28.12	eks.18	セキュリティの修正および機能強化が行われた新しいプラットフォームバージョン。	2024 年 8 月 28 日

Kubernetes バージョン	EKS プラットフォームのバージョン	リリースノート	リリース日
1.28.11	eks.17	セキュリティの修正および機能強化が行われた新しいプラットフォームバージョン。	2024 年 8 月 9 日
1.28.11	eks.16	セキュリティの修正および機能強化が行われた新しいプラットフォームバージョン。	2024 年 7 月 2 日
1.28.9	eks.13	CoreDNS 自動スケーリング、セキュリティの修正および機能強化が行われた、新しいプラットフォームバージョン。CoreDNS 自動スケーリングの詳細については「 the section called “をスケールして DNS トラフィックを増やす” 」を参照してください。	2024 年 5 月 16 日
1.28.8	eks.12	セキュリティの修正および機能強化が行われた新しいプラットフォームバージョン。	2024 年 4 月 18 日

Kubernetes バージョン	EKS プラットフォームのバージョン	リリースノート	リリース日
1.28.7	eks.11	セキュリティの修正 および機能強化が行 われた新しいプラッ トフォームバージョ ン。	2024 年 3 月 29 日
1.28.7	eks.10	セキュリティの修正 および機能強化が行 われた新しいプラッ トフォームバージョ ン。	2024 年 3 月 20 日
1.28.6	eks.9	セキュリティの修正 および機能強化が行 われた新しいプラッ トフォームバージョ ン。	2024 年 3 月 12 日
1.28.5	eks.7	セキュリティの修正 および機能強化が行 われた新しいプラッ トフォームバージョ ン。	2024 年 1 月 17 日
1.28.4	eks.6	アクセスエントリ 、 セキュリティの修正 および機能強化が行 われた新しいプラッ トフォームバージョ ン。	2023 年 12 月 14 日

Kubernetes バージョン	EKS プラットフォームのバージョン	リリースノート	リリース日
1.28.4	eks.5	セキュリティの修正および機能強化が行われた新しいプラットフォームバージョン。	2023 年 12 月 12 日
1.28.3	eks.4	EKS Pod Identity がポッドに AWS サービスへのアクセス権を付与する方法の説明 、セキュリティ修正、および機能強化を含む新しいプラットフォームバージョン。	2023 年 11 月 10 日
1.28.3	eks.3	セキュリティの修正および機能強化が行われた新しいプラットフォームバージョン。	2023 年 11 月 3 日
1.28.2	eks.2	セキュリティの修正および機能強化が行われた新しいプラットフォームバージョン。	2023 年 10 月 16 日

Kubernetes バージョン	EKS プラットフォームのバージョン	リリースノート	リリース日
1.28.1	eks.1	EKS 用 Kubernetes バージョン 1.28 の初回リリース。 詳細については、「 the section called “Kubernetes 1.28” 」を参照してください。	2023 年 9 月 26 日

Kubernetes バージョン 1.27

次の受付コントローラーはすべての 1.27 プラットフォームバージョンで有効です:

NodeRestriction、ExtendedResourceToleration、NamespaceLifecycle、LimitRanger、ServiceAccountTokenProjection

Kubernetes バージョン	EKS プラットフォームのバージョン	リリースノート	リリース日
1.27.16	eks.49	セキュリティの修正および機能強化が行われた新しいプラットフォームバージョン。	2025 年 6 月 11 日
1.27.16	eks.48	セキュリティの修正および機能強化が行われた新しいプラットフォームバージョン。	2025 年 5 月 30 日
1.27.16	eks.47	セキュリティの修正および機能強化が行われた新しいプラットフォームバージョン。	2025 年 5 月 16 日

Kubernetes バージョン	EKS プラットフォームのバージョン	リリースノート	リリース日
		トフォームバージョン。	
1.27.16	eks.46	セキュリティの修正および機能強化が行われた新しいプラットフォームバージョン。	2025 年 4 月 29 日
1.27.16	eks.45	セキュリティの修正および機能強化が行われた新しいプラットフォームバージョン。	2025 年 4 月 18 日
1.27.16	eks.44	セキュリティの修正および機能強化が行われた新しいプラットフォームバージョン。	2025 年 4 月 18 日
1.27.16	eks.43	セキュリティの修正および機能強化が行われた新しいプラットフォームバージョン。	2025 年 4 月 2 日
1.27.16	eks.42	セキュリティの修正および機能強化が行われた新しいプラットフォームバージョン。	2025 年 3 月 17 日

Kubernetes バージョン	EKS プラットフォームのバージョン	リリースノート	リリース日
1.27.16	eks.41	セキュリティの修正 および機能強化が行 われた新しいプラッ トフォームバージョ ン。	2025 年 3 月 4 日
1.27.16	eks.40	セキュリティの修正 および機能強化が行 われた新しいプラッ トフォームバージョ ン。	2025 年 3 月 4 日
1.27.16	eks.38	セキュリティの修正 および機能強化が行 われた新しいプラッ トフォームバージョ ン。	2025年1月17日
1.27.16	eks.37	セキュリティの修正 および機能強化が行 われた新しいプラッ トフォームバージョ ン。	2025 年 1 月 3 日
1.27.16	eks.36	セキュリティの修正 および機能強化が行 われた新しいプラッ トフォームバージョ ン。	2024 年 12 月 13 日
1.27.16	eks.35	セキュリティの修正 および機能強化が行 われた新しいプラッ トフォームバージョ ン。	2024 年 12 月 13 日

Kubernetes バージョン	EKS プラットフォームのバージョン	リリースノート	リリース日
1.27.16	eks.34	セキュリティの修正 および機能強化が行 われた新しいプラッ トフォームバージョ ン。	2024 年 12 月 13 日
1.27.16	eks.33	Amazon EKS Hybrid Nodes のサポート、 セキュリティの修 正と機能強化つき の新しいプラット フォームバージョ ン。Amazon EKS Hybrid Nodes の詳 細については「 the section called “ハイブ リッドノード” 」を参 照してください。	2024 年 11 月 15 日
1.27.16	eks.27	セキュリティの修正 および機能強化が行 われた新しいプラッ トフォームバージョ ン。	2024 年 10 月 21 日
1.27.16	eks.23	セキュリティの修正 および機能強化が行 われた新しいプラッ トフォームバージョ ン。	2024 年 9 月 3 日

Kubernetes バージョン	EKS プラットフォームのバージョン	リリースノート	リリース日
1.27.16	eks.22	セキュリティの修正および機能強化が行われた新しいプラットフォームバージョン。	2024 年 8 月 28 日
1.27.16	eks.21	セキュリティの修正および機能強化が行われた新しいプラットフォームバージョン。	2024 年 8 月 9 日
1.27.15	eks.20	セキュリティの修正および機能強化が行われた新しいプラットフォームバージョン。	2024 年 7 月 2 日
1.27.13	eks.17	CoreDNS 自動スケーリング、セキュリティの修正および機能強化が行われた、新しいプラットフォームバージョン。CoreDNS 自動スケーリングの詳細については「 the section called “をスケールして DNS トラフィックを増やす” 」を参照してください。	2024 年 5 月 16 日

Kubernetes バージョン	EKS プラットフォームのバージョン	リリースノート	リリース日
1.27.12	eks.16	セキュリティの修正 および機能強化が行 われた新しいプラッ トフォームバージョ ン。	2024 年 4 月 18 日
1.27.11	eks.15	セキュリティの修正 および機能強化が行 われた新しいプラッ トフォームバージョ ン。	2024 年 3 月 29 日
1.27.11	eks.14	セキュリティの修正 および機能強化が行 われた新しいプラッ トフォームバージョ ン。	2024 年 3 月 20 日
1.27.10	eks.13	セキュリティの修正 および機能強化が行 われた新しいプラッ トフォームバージョ ン。	2024 年 3 月 12 日
1.27.9	eks.11	セキュリティの修正 および機能強化が行 われた新しいプラッ トフォームバージョ ン。	2024 年 1 月 17 日

Kubernetes バージョン	EKS プラットフォームのバージョン	リリースノート	リリース日
1.27.8	eks.10	アクセスエントリ 、セキュリティの修正および機能強化が行われた新しいプラットフォームバージョン。	2023 年 12 月 14 日
1.27.8	eks.9	セキュリティの修正および機能強化が行われた新しいプラットフォームバージョン。	2023 年 12 月 12 日
1.27.7	eks.8	EKS Pod Identity がポッドに AWS サービスへのアクセス権を付与する方法の説明 、セキュリティ修正、および機能強化を含む新しいプラットフォームバージョン。	2023 年 11 月 10 日
1.27.7	eks.7	セキュリティの修正および機能強化が行われた新しいプラットフォームバージョン。	2023 年 11 月 3 日
1.27.6	eks.6	セキュリティの修正および機能強化が行われた新しいプラットフォームバージョン。	2023 年 10 月 16 日

Kubernetes バージョン	EKS プラットフォームのバージョン	リリースノート	リリース日
1.27.4	eks.5	セキュリティの修正および機能強化が行われた新しいプラットフォームバージョン。	2023 年 8 月 30 日
1.27.4	eks.4	セキュリティの修正および機能強化が行われた新しいプラットフォームバージョン。	2023 年 7 月 30 日
1.27.3	eks.3	セキュリティの修正および機能強化が行われた新しいプラットフォームバージョン。	2023 年 6 月 30 日
1.27.2	eks.2	セキュリティの修正および機能強化が行われた新しいプラットフォームバージョン。	2023 年 6 月 9 日
1.27.1	eks.1	EKS 用 Kubernetes バージョン 1.27 の初回リリース。 詳細については、「 the section called “Kubernetes 1.27” 」を参照してください。	2023 年 5 月 24 日

Kubernetes バージョン 1.26

次の受付コントローラーはすべての 1.26 プラットフォームバージョンで有効です:

NodeRestriction、ExtendedResourceToleration、NamespaceLifecycle、LimitRanger、ServiceAccountTokenProjection

Kubernetes バージョン	EKS プラットフォームのバージョン	リリースノート	リリース日
1.26.15	eks.51	セキュリティの修正 および機能強化が行 われた新しいプラッ トフォームバージョ ン。	2025 年 6 月 11 日
1.26.15	eks.50	セキュリティの修正 および機能強化が行 われた新しいプラッ トフォームバージョ ン。	2025 年 5 月 30 日
1.26.15	eks.49	セキュリティの修正 および機能強化が行 われた新しいプラッ トフォームバージョ ン。	2025 年 5 月 16 日
1.26.15	eks.48	セキュリティの修正 および機能強化が行 われた新しいプラッ トフォームバージョ ン。	2025 年 4 月 29 日
1.26.15	eks.47	セキュリティの修正 および機能強化が行 われた新しいプラッ トフォームバージョ ン。	2025 年 4 月 18 日

Kubernetes バージョン	EKS プラットフォームのバージョン	リリースノート	リリース日
1.26.15	eks.46	セキュリティの修正 および機能強化が行 われた新しいプラッ トフォームバージョ ン。	2025 年 4 月 18 日
1.26.15	eks.45	セキュリティの修正 および機能強化が行 われた新しいプラッ トフォームバージョ ン。	2025 年 4 月 2 日
1.26.15	eks.44	セキュリティの修正 および機能強化が行 われた新しいプラッ トフォームバージョ ン。	2025 年 3 月 17 日
1.26.15	eks.43	セキュリティの修正 および機能強化が行 われた新しいプラッ トフォームバージョ ン。	2025 年 3 月 4 日
1.26.15	eks.42	セキュリティの修正 および機能強化が行 われた新しいプラッ トフォームバージョ ン。	2025 年 2 月 24 日
1.26.15	eks.40	セキュリティの修正 および機能強化が行 われた新しいプラッ トフォームバージョ ン。	2025年1月17日

Kubernetes バージョン	EKS プラットフォームのバージョン	リリースノート	リリース日
1.26.15	eks.39	セキュリティの修正 および機能強化が行 われた新しいプラッ トフォームバージョ ン。	2025 年 1 月 3 日
1.26.15	eks.38	セキュリティの修正 および機能強化が行 われた新しいプラッ トフォームバージョ ン。	2024 年 12 月 13 日
1.26.15	eks.37	セキュリティの修正 および機能強化が行 われた新しいプラッ トフォームバージョ ン。	2024 年 12 月 13 日
1.26.15	eks.36	セキュリティの修正 および機能強化が行 われた新しいプラッ トフォームバージョ ン。	2024 年 12 月 13 日

Kubernetes バージョン	EKS プラットフォームのバージョン	リリースノート	リリース日
1.26.15	eks.35	Amazon EKS Hybrid Nodes のサポート、セキュリティの修正と機能強化つきの新しいプラットフォームバージョン。Amazon EKS Hybrid Nodes の詳細については「 the section called “ハイブリッドノード” 」を参照してください。	2024 年 11 月 15 日
1.26.15	eks.28	セキュリティの修正および機能強化が行われた新しいプラットフォームバージョン。	2024 年 10 月 21 日
1.26.15	eks.24	セキュリティの修正および機能強化が行われた新しいプラットフォームバージョン。	2024 年 9 月 3 日
1.26.15	eks.23	セキュリティの修正および機能強化が行われた新しいプラットフォームバージョン。	2024 年 8 月 28 日

Kubernetes バージョン	EKS プラットフォームのバージョン	リリースノート	リリース日
1.26.15	eks.22	セキュリティの修正および機能強化が行われた新しいプラットフォームバージョン。	2024 年 8 月 9 日
1.26.15	eks.21	セキュリティの修正および機能強化が行われた新しいプラットフォームバージョン。	2024 年 7 月 2 日
1.26.15	eks.18	CoreDNS 自動スケーリング、セキュリティの修正および機能強化が行われた、新しいプラットフォームバージョン。CoreDNS 自動スケーリングの詳細については「 the section called “をスケールして DNS トラフィックを増やす” 」を参照してください。	2024 年 5 月 16 日
1.26.15	eks.17	セキュリティの修正および機能強化が行われた新しいプラットフォームバージョン。	2024 年 4 月 18 日

Kubernetes バージョン	EKS プラットフォームのバージョン	リリースノート	リリース日
1.26.14	eks.16	セキュリティの修正 および機能強化が行 われた新しいプラッ トフォームバージョ ン。	2024 年 3 月 29 日
1.26.14	eks.15	セキュリティの修正 および機能強化が行 われた新しいプラッ トフォームバージョ ン。	2024 年 3 月 20 日
1.26.13	eks.14	セキュリティの修正 および機能強化が行 われた新しいプラッ トフォームバージョ ン。	2024 年 3 月 12 日
1.26.12	eks.12	セキュリティの修正 および機能強化が行 われた新しいプラッ トフォームバージョ ン。	2024 年 1 月 17 日
1.26.11	eks.11	アクセスエントリ 、 セキュリティの修正 および機能強化が行 われた新しいプラッ トフォームバージョ ン。	2023 年 12 月 14 日

Kubernetes バージョン	EKS プラットフォームのバージョン	リリースノート	リリース日
1.26.11	eks.10	セキュリティの修正 および機能強化が行 われた新しいプラッ トフォームバージョ ン。	2023 年 12 月 12 日
1.26.10	eks.9	EKS Pod Identity が ポッドに AWS サー ビスへのアクセス権 を付与する方法の説 明 、セキュリティ修 正、および機能強化 を含む新しいプラッ トフォームバージョ ン。	2023 年 11 月 10 日
1.26.10	eks.8	セキュリティの修正 および機能強化が行 われた新しいプラッ トフォームバージョ ン。	2023 年 11 月 3 日
1.26.9	eks.7	セキュリティの修正 および機能強化が行 われた新しいプラッ トフォームバージョ ン。	2023 年 10 月 16 日
1.26.7	eks.6	セキュリティの修正 および機能強化が行 われた新しいプラッ トフォームバージョ ン。	2023 年 8 月 30 日

Kubernetes バージョン	EKS プラットフォームのバージョン	リリースノート	リリース日
1.26.7	eks.5	セキュリティの修正および機能強化が行われた新しいプラットフォームバージョン。	2023 年 7 月 30 日
1.26.6	eks.4	セキュリティの修正および機能強化が行われた新しいプラットフォームバージョン。	2023 年 6 月 30 日
1.26.5	eks.3	セキュリティの修正および機能強化が行われた新しいプラットフォームバージョン。	2023 年 6 月 9 日
1.26.4	eks.2	セキュリティの修正および機能強化が行われた新しいプラットフォームバージョン。	2023 年 5 月 5 日
1.26.2	eks.1	EKS 用 Kubernetes バージョン 1.26 の初回リリース。 詳細については、「 the section called “Kubernetes 1.26” 」を参照してください。	2023 年 4 月 11 日

最新のプラットフォームバージョンを取得する

1. Amazon EKS コンソールを開きます。
2. ナビゲーションペインで [Clusters] (クラスター) を選択してください。
3. クラスターのリストで、プラットフォームバージョンを確認する [クラスター名] を選択してください。
4. [概要] タブを選択してください。
5. [プラットフォームバージョン] は[詳細] セクションで参照できます。
6. プラットフォームバージョンを確認するクラスターの [名前] を確認します。
7. 次のコマンドを実行してください。

```
aws eks describe-cluster --name my-cluster --query cluster.platformVersion
```

出力例は次のとおりです。

```
"eks.10"
```

プラットフォームバージョンを変更する

EKS クラスターのプラットフォームバージョンを変更することはできません。ある Kubernetes バージョンのために新しい Amazon EKS プラットフォームバージョンが使用可能になると、EKS は既存のすべてのクラスターを、対応する Kubernetes バージョンの最新の Amazon EKS プラットフォームバージョンに自動的にアップグレードします。既存の Amazon EKS プラットフォームバージョンの自動アップグレードは段階的に役割アウトされます。AWS コンソールまたは CLI を使用してプラットフォームバージョンを変更することはできません。

Kubernetes バージョンをアップグレードすると、クラスターはその Kubernetes バージョンの最新のプラットフォームバージョンに移行します。

Karpenter と Cluster Autoscaler を使用したクラスターコンピューティングのスケールリング

自動スケールリングとは、需要の変化に合わせてリソースを自動的に増減させる機能です。これは Kubernetes の主要な機能であり、この機能がなければ、手動で実行するために膨大な人的リソースが必要になります。

EKS Auto Mode

Amazon EKS Auto Mode は、クラスターコンピューティングリソースを自動的にスケールします。ポッドが既存のノードに収まらない場合、EKS Auto Mode は新しいノードを作成します。また、EKS Auto Mode はワークロードの統合とノードの削除も行います。EKS Auto Mode は Karpenter に基づいて構築されます。

詳細については、以下を参照してください。

- [EKS Auto Mode](#)
- [the section called “ノードプールの作成”](#)
- [the section called “拡張ワークロードのデプロイ”](#)

その他のソリューション

Amazon EKS はさらに 2 つの自動スケーリング製品をサポートしています。

Karpenter

Karpenter は、アプリケーションの可用性とクラスターの効率の向上に役立つ柔軟で高性能な Kubernetes クラスターオートスケーラーです。Karpenter は、アプリケーションの負荷の変化に応じて、適切なサイズのコンピューティングリソース (Amazon EC2 インスタンスなど) を 1 分未満で起動します。Kubernetes と AWS との統合を通じて、Karpenter は、ワークロードの要件を正確に満たすジャストインタイムコンピューティングリソースをプロビジョンできます。Karpenter は、クラスターワークロードの具体的な要件に基づいて、新しいコンピューティングリソースを自動的にプロビジョンします。これには、コンピューティング、ストレージ、高速化、スケジューリングの各要件が含まれます。Amazon EKS は Karpenter を使用したクラスターをサポートしていますが、Karpenter は適合するいずれの Kubernetes クラスターでも動作します。詳細については、[Karpenter](#) のドキュメントを参照してください。

Important

Karpenter は、AWS ユーザーが Kubernetes クラスターでインストール、設定、管理を行うオープンソースソフトウェアです。AWS は、Amazon EKS クラスターで互換性のあるバージョンを使用して Karpenter を変更せずに実行した場合にテクニカルサポートを提供します。他のカスタマーマネージドソフトウェアと同様に、お客様が Karpenter コントローラーの可用性とセキュリティ、および、Karpenter コントローラーやその実行中の Kubernetes クラスターをアップグレードするときの適切なテスト手順を維持すること

が重要です。Karpenter の AWS サービスレベルアグリーメント (SLA) はなく、お客様は Karpenter によって起動された EC2 インスタンスがビジネス要件を満たしていることを確認する必要があります。

Cluster Autoscaler

Kubernetes の Cluster Autoscalerは、ポッドが失敗したり、他のノードに再スケジュールされた場合に、クラスター内のノード数を自動的に調整します。Cluster Autoscaler は Auto Scaling グループを使用します。詳細については、「[AWS の Cluster Autoscaler](#)」を参照してください。

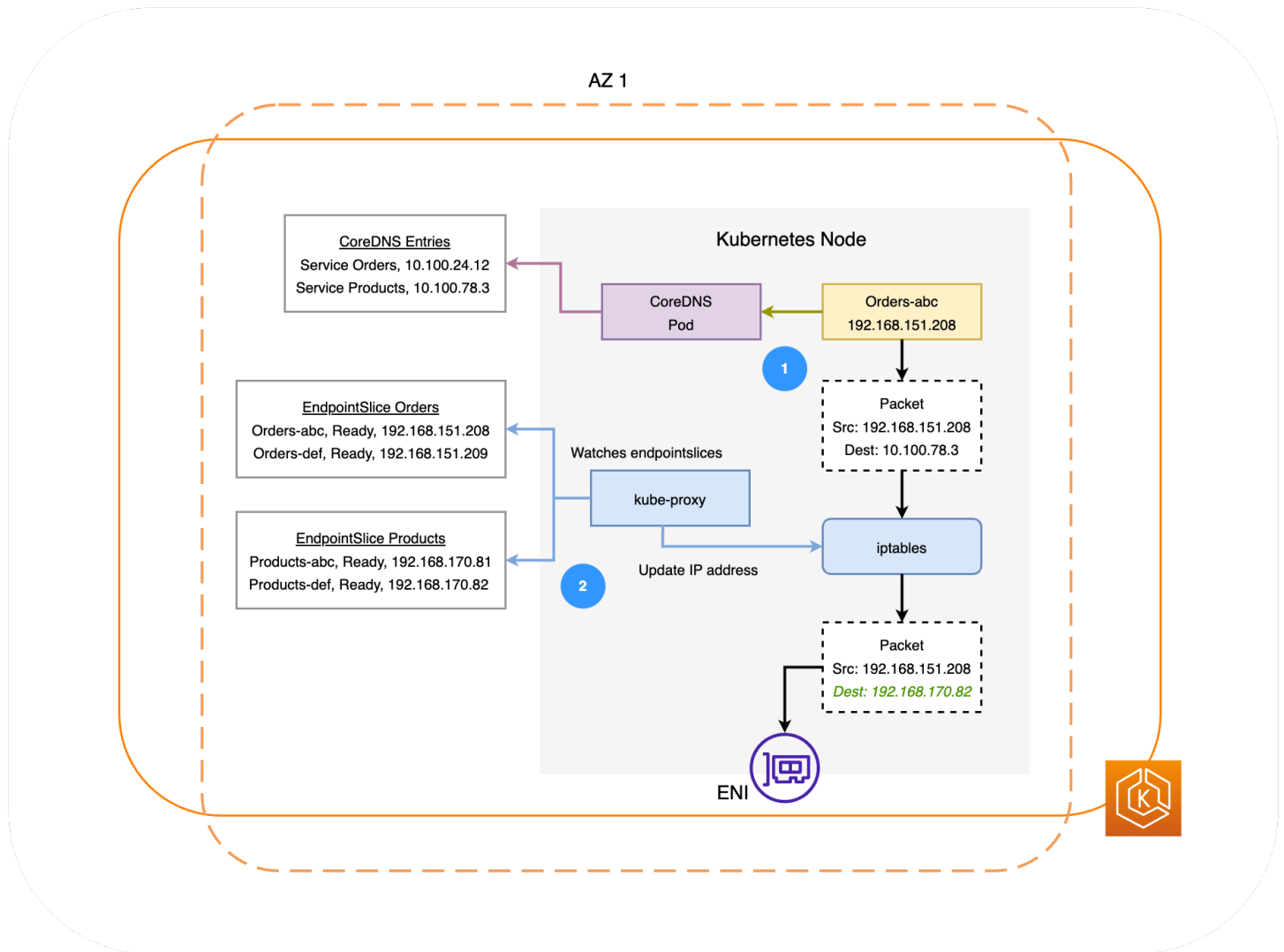
Amazon EKS での Amazon Application Recovery Controller (ARC) のゾーンシフトの詳細

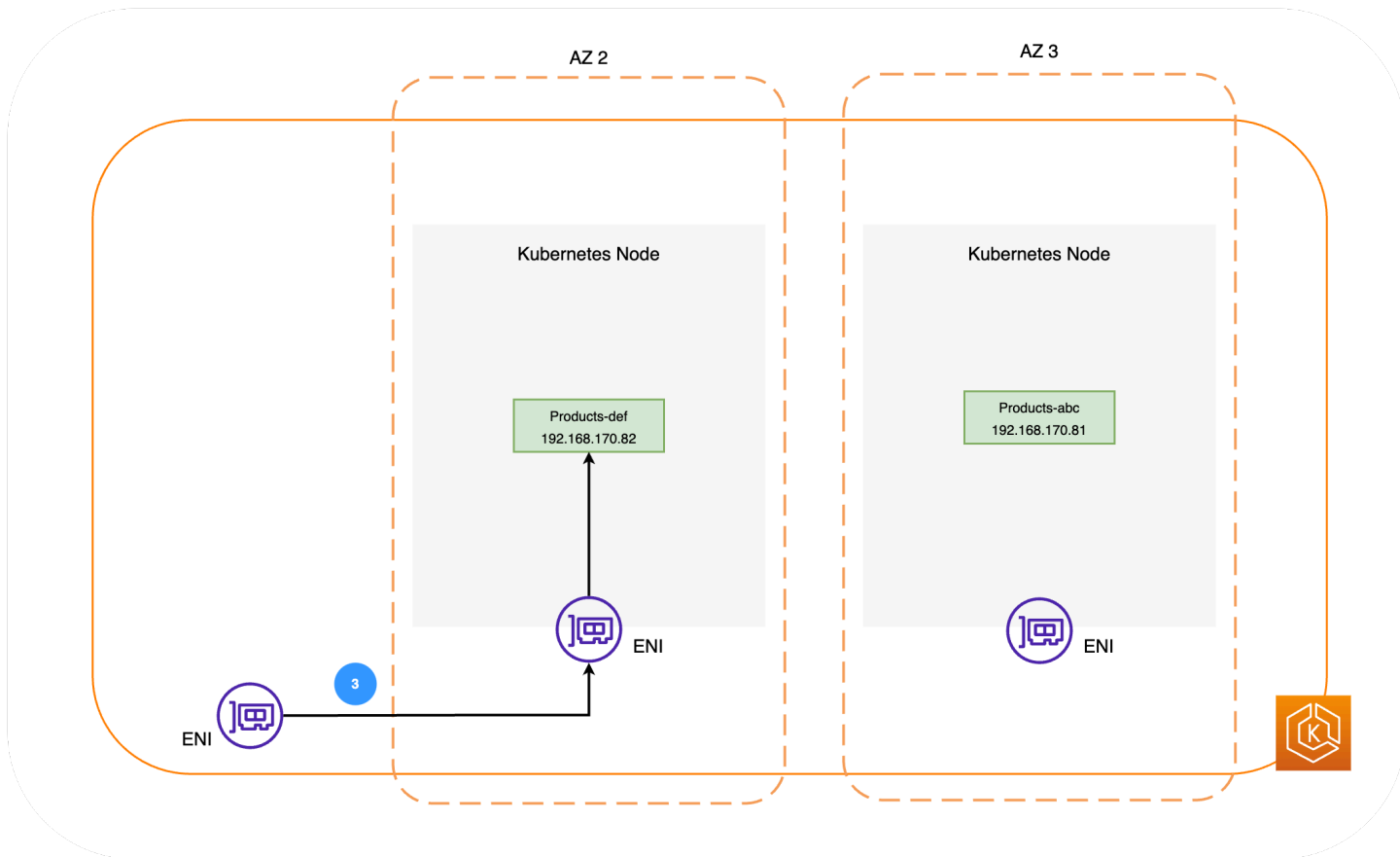
Kubernetes には、アベイラビリティーゾーン (AZ) のヘルスの低下や障害などのイベントに対するアプリケーションの回復力を高めるネイティブ機能があります。Amazon EKS クラスターでワークロードを実行する場合、[Amazon Application Recovery Controller \(ARC\) のゾーンシフト](#)または[ゾーンオートシフト](#)を使用して、アプリケーション環境の耐障害性とアプリケーション復旧をさらに改善できます。ARC ゾーンシフトは、ゾーンシフトが期限切れになるまで、またはキャンセルするまで、障害のある AZ からリソースのトラフィックを移動できる一時的な手段として設計されています。必要に応じてゾーンシフトを拡張できます。

EKS クラスターのゾーンシフトを開始することも、ゾーンオートシフトを有効にして AWS がゾーンシフトを実行できるようにすることもできます。このシフトは、正常な AZ のワーカーノードで実行されているポッドのネットワークエンドポイントのみを使用するようにクラスター内の東西ネットワークトラフィックのフローを更新します。さらに、EKS クラスター内のアプリケーションの入力トラフィックを処理する ALB または NLB は、トラフィックを正常な AZ 内のターゲットに自動的にルーティングします。可用性を最大限に高めることを目標にしているお客様にとって、AZ で障害が発生した場合は、障害のある AZ が回復するまでその AZ からすべてのトラフィックを移動できることが重要です。そのためには、[ARC ゾーンシフトで ALB または NLB を有効にする](#)こともできます。

ポッド間の東西ネットワークトラフィックフローを理解する

次の図は、2 つのワークロードの例である Orders と Products を示しています。この例の目的は、さまざまな AZ でワークロードとポッドが通信する方法を示します。





1. Orders が Products と通信するには、まず送信先のサービスの DNS 名を解決する必要があります。Orders は CoreDNS と通信して、そのサービスの仮想 IP アドレス (クラスター IP) を取得します。Orders が Products のサービス名を解決すると、そのターゲット IP にトラフィックが送信されます。
2. kube-proxy はクラスター内のすべてのノードで実行され、[EndpointSlices](#) for Services を継続的に監視します。サービスが作成されると、EndpointSlice コントローラーによってバックグラウンドで EndpointSlice が作成され、管理されます。各 EndpointSlice には、ポッドアドレスのサブセットと、それらが実行されているノードを含むエンドポイントのリストまたは表があります。kube-proxy は、ノードで iptables を使用して、これらのポッドエンドポイントごとにルーティングルールを設定します。また、kube-proxy は、基本的なロードバランシングも行います。サービスのクラスター IP 宛てのトラフィックを、ポッドの IP アドレスに直接送信するようリダイレクトします。kube-proxy は、送信接続の送信先 IP を書き換えることでこれを行います。
3. 次に、ネットワークパケットは、各ノードの ENI を介して AZ 2 の Products ポッドに送信されます (上の図を参照)。

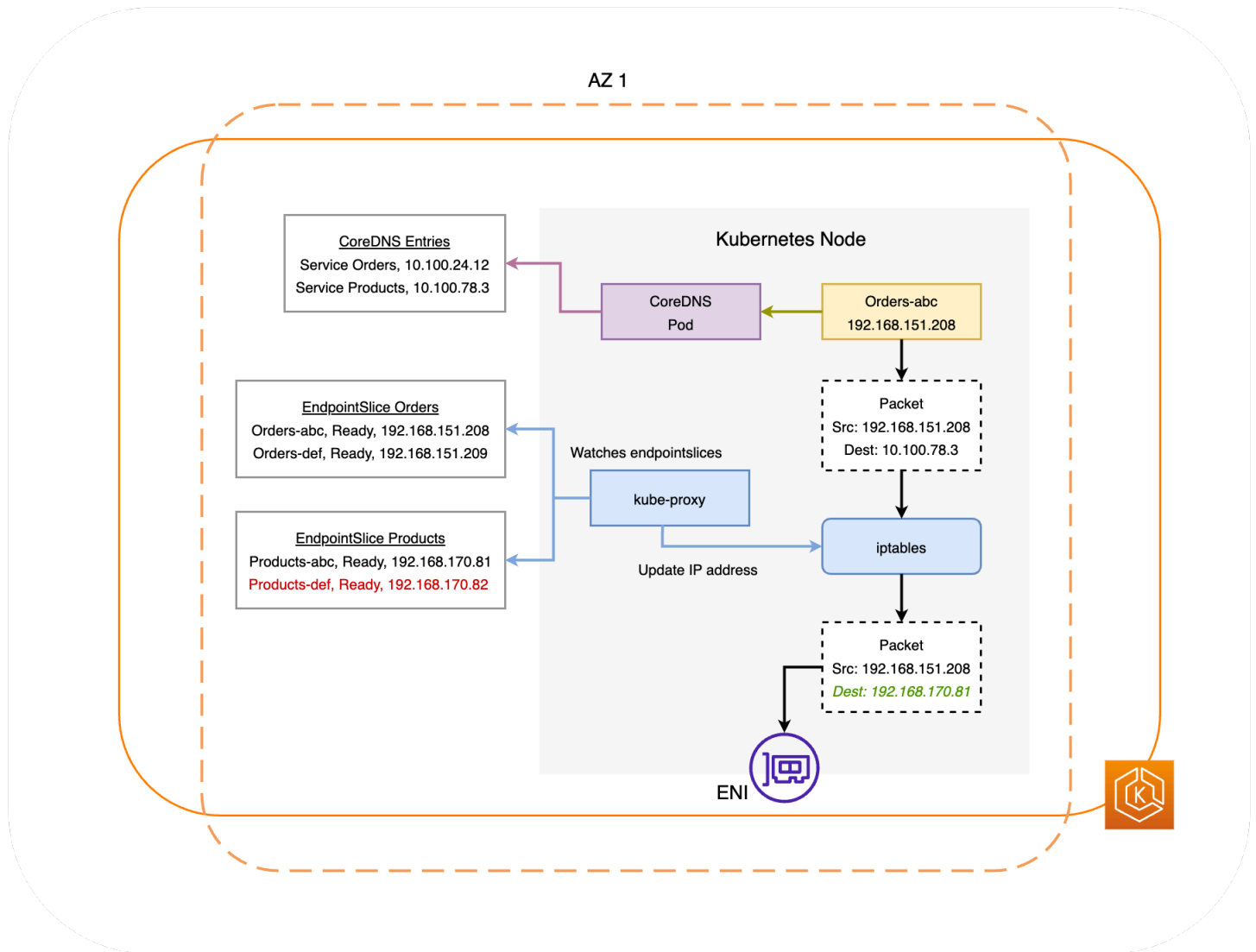
Amazon EKS での ARC ゾーンシフトについて

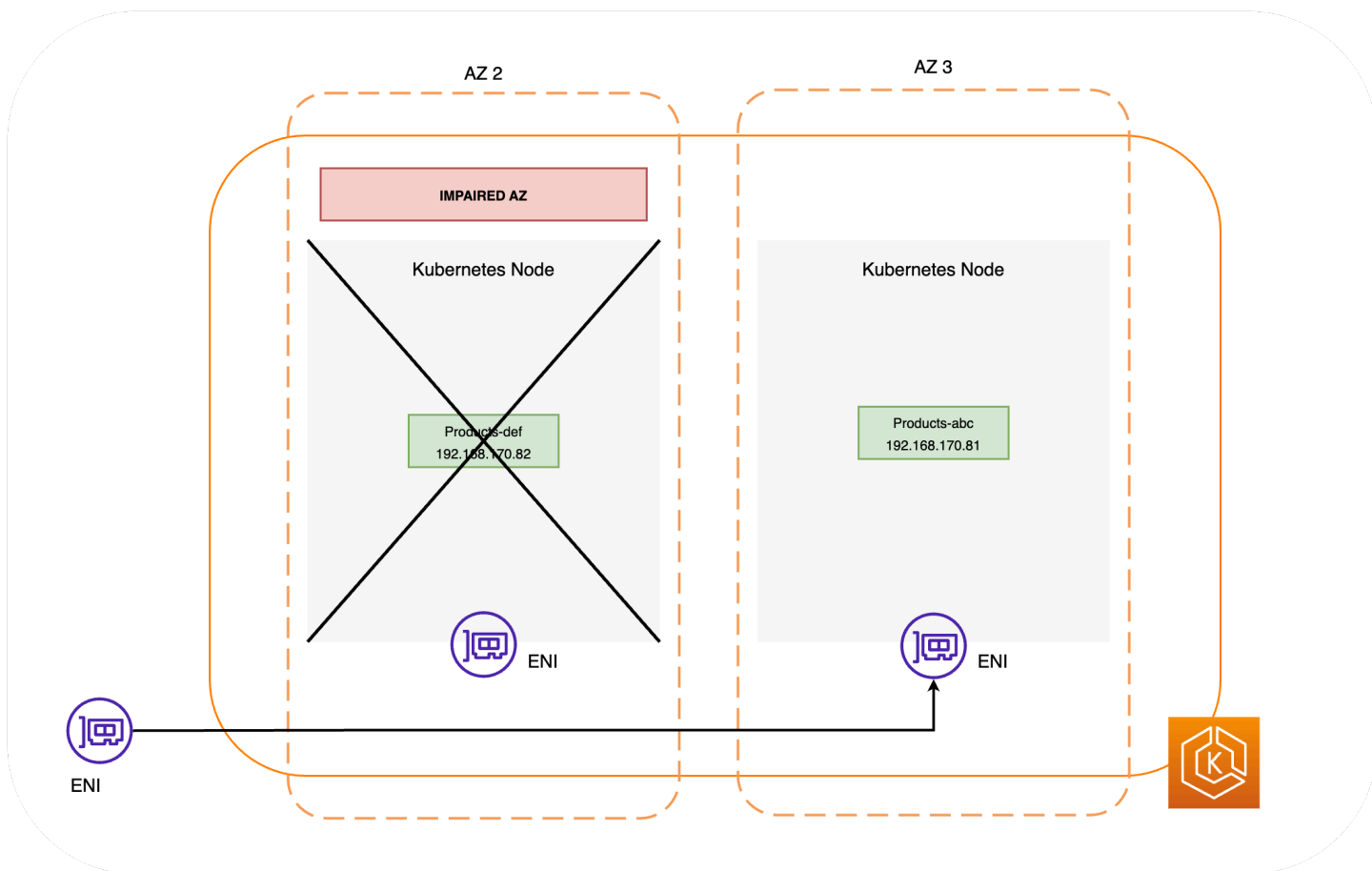
環境に AZ の障害がある場合は、EKS クラスター環境のゾーンシフトを開始できます。または、ゾーンオートシフトを使用して AWS がこれを管理できます。ゾーンオートシフトでは、AWS は AZ の全体的なヘルスをモニタリングし、クラスター環境内の障害のある AZ からトラフィックを自動的に移動することで、潜在的な AZ 障害に対応します。

ARC で Amazon EKS クラスターのゾーンシフトを有効にすると、ARC コンソール、AWS CLI、またはゾーンシフト API やゾーンオートシフト API を使用して、ゾーンシフトをトリガーしたり、ゾーンオートシフトを有効にしたりできます。EKS ゾーンシフト中、次のことが自動的に行われます。

- 影響を受ける AZ 内のすべてのノードが遮断されます。これにより、Kubernetes スケジューラーが異常な AZ のノードに新しいポッドをスケジューリングできなくなります。
- [マネージドノードグループ](#)を使用している場合、[アベイラビリティゾーンの再分散](#)は停止され、Auto Scalingグループ (ASG) が更新され、新しい EKS データプレーンノードが正常な AZ でのみ起動されるようにします。
- 異常な AZ のノードは終了されず、ポッドはこれらのノードから削除されます。これは、ゾーンシフトの有効期限が切れたりキャンセルされたりしたときに、トラフィックがフル容量を保持している AZ に安全に戻ることができるようにするためです。
- EndpointSlice コントローラーは、障害のある AZ 内のすべてのポッドエンドポイントを検索し、関連する EndpointSlices から削除します。これにより、正常な AZ のポッドエンドポイントのみがネットワークトラフィックの受信対象になります。ゾーンシフトがキャンセルまたは期限切れになると、EndpointSlice コントローラーは EndpointSlices を更新して、復元された AZ にエンドポイントを含めます。

以下の図は、EKS ゾーンシフトによって、クラスター環境で正常なポッドエンドポイントのみがターゲットにされる方法についての大まかなフローを示しています。





EKS ゾーンシフトの要件

ゾーンシフトを EKS で正常に機能させるには、事前に AZ の障害に対する回復性を実現できるようにクラスター環境を設定する必要があります。以下は、従う必要があるステップのリストです。

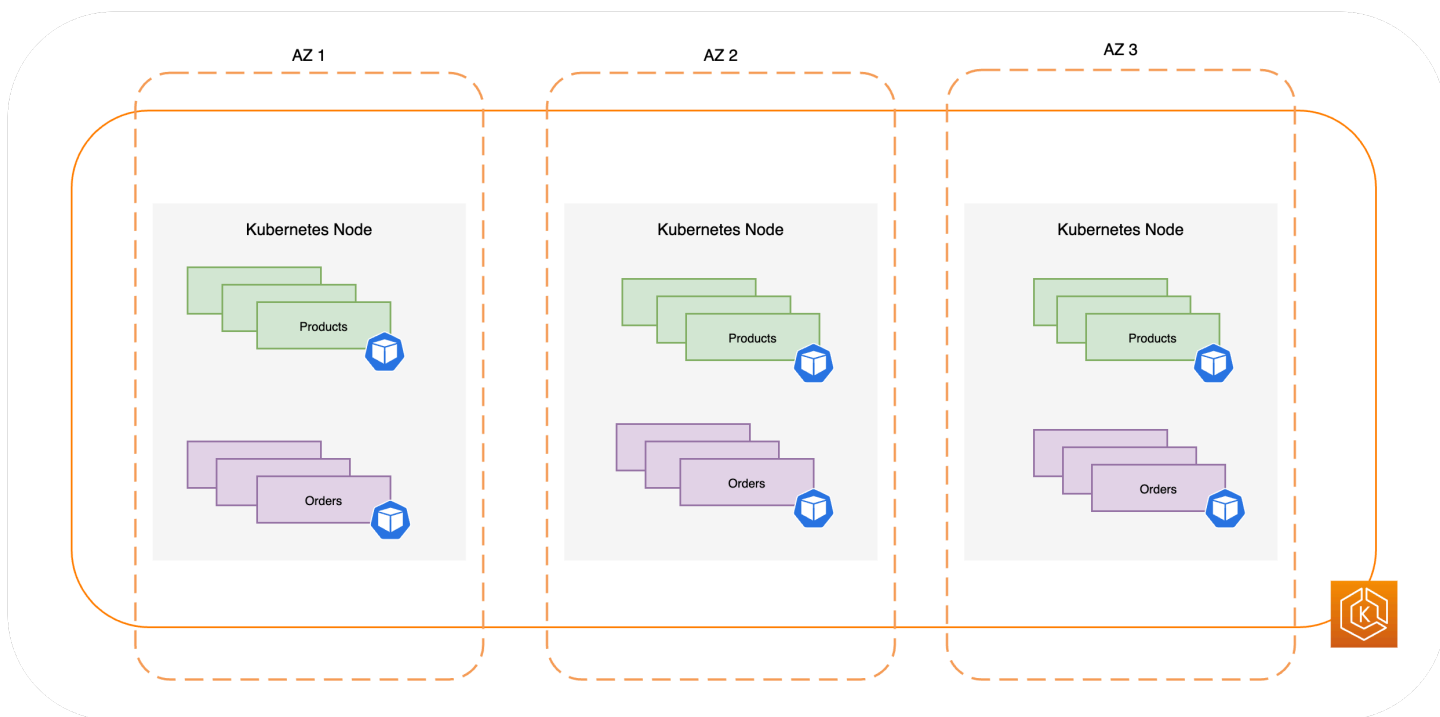
- 複数の AZ にクラスターのワーカーノードをプロビジョニングする
- 1 つの AZ の削除に耐えるのに十分なコンピューティング容量をプロビジョニングする
- すべての AZ でポッド (CoreDNS を含む) をプリスケールする
- すべての AZ に複数のポッドレプリカを分散して、単一の AZ から移動することで十分な容量を確保できるようにします。
- 相互依存ポッドまたは関連ポッドを同じ AZ に配置する
- ゾーンシフトを手動で開始することで、少ない AZ でクラスター環境が期待どおりに動作することをテストします。または、ゾーンオートシフトを有効にし、オートシフトの練習運用に頼ることもできます。これは、ゾーンシフトが EKS で機能するためには必要ありませんが、強くお勧めします。

複数の AZ に EKS ワーカーノードをプロビジョニングする

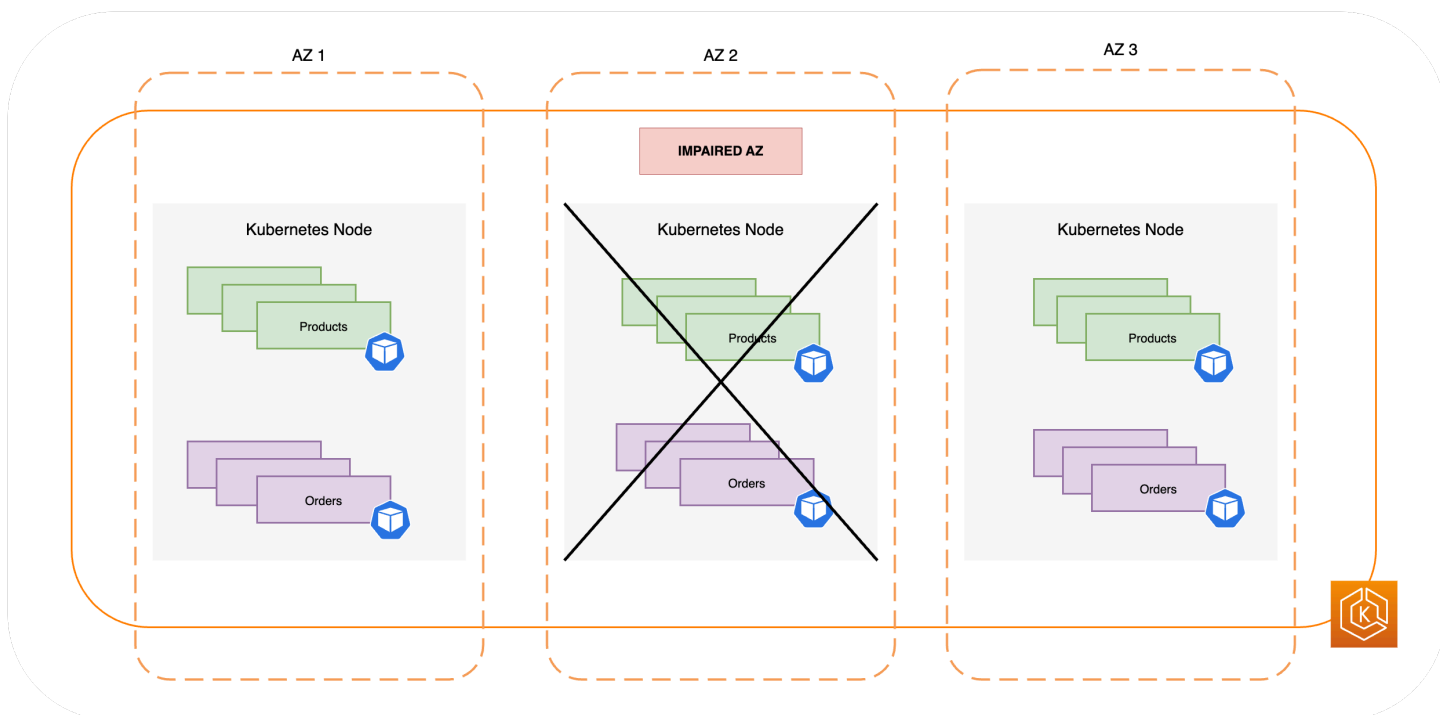
AWS リージョンには、アベイラビリティーゾーン (AZ) と呼ばれる物理データセンターがある複数の別々の場所があります。AZ は、リージョン全体に影響を与える可能性のある同時障害を回避するために、互いに物理的に離れた場所に配置されています。EKS クラスターをプロビジョニングする場合は、リージョン内の複数の AZ にワーカーノードをデプロイする必要があります。これにより、クラスター環境は 1 つの AZ の障害に対する回復力が向上し、他の AZ で実行されているアプリケーションの高可用性 (HA) を維持できます。影響を受ける AZ からゾーンシフトを開始すると、EKS 環境のクラスター内ネットワークは自動的に更新されて正常な AZ のみが使用されますが、クラスターの高可用性は維持されます。

EKS 環境にこのようなマルチ AZ 設定を実施することで、システム全体の信頼性が向上します。ただし、マルチ AZ 環境は、アプリケーションデータの転送と処理方法に大きな役割を果たし、環境のネットワーク料金に影響します。特に、頻繁な egress クロスゾーントラフィック (AZ 間に分散されたトラフィック) は、ネットワーク関連のコストに大きな影響を与える可能性があります。さまざまな戦略を適用して、EKS クラスター内のポッド間のクロスゾーントラフィックの量を制御し、関連するコストを削減できます。高可用性 EKS 環境を実行するときにネットワークコストを最適化する方法の詳細については、[このベストプラクティスガイド](#)を参照してください。

以下の図は、3 つの正常な AZ を持つ高可用性 EKS 環境を示しています。



次の図は、3 つの AZ を持つ EKS 環境が AZ の障害に対して回復力があり、他の 2 つの正常な AZ のために高可用性を維持する方法を示しています。



1 つの AZ の削除に耐えるのに十分なコンピューティング容量をプロビジョニングする

EKSデータプレーンでコンピューティングインフラストラクチャのリソース使用率とコストを最適化するには、コンピューティング容量をワークロード要件に合わせることをお勧めします。ただし、すべてのワーカーノードがフル容量である場合、新しいポッドをスケジュールする前に、新しいワーカーノードを EKS データプレーンに追加する必要があります。重要なワークロードを実行する場合、通常、負荷の急増やノードのヘルスの問題などに対処するために、冗長な容量をオンラインで運用することをお勧めします。ゾーンシフトを使用する予定がある場合、AZ 全体の容量を取り除くことを計画していることになります。そのため、AZ がオフラインになった状態でも負荷を処理できるように、冗長なコンピューティング容量を調整する必要があります。

コンピューティングをスケーリングする場合、EKS データプレーンに新しいノードを追加するプロセスには時間がかかるため、特にゾーンの障害が発生した場合に、アプリケーションのリアルタイムのパフォーマンスと可用性に影響を与える可能性があります。EKS 環境は、エンドユーザーやクライアントのエクスペリエンスの低下を避けるために、AZ を失った場合の負荷を吸収する回復力がある必要があります。これは、新しいポッドが必要な時点からワーカーノードで実際にスケジュールされる時点までの遅延を最小限に抑えるか、排除することを意味します。

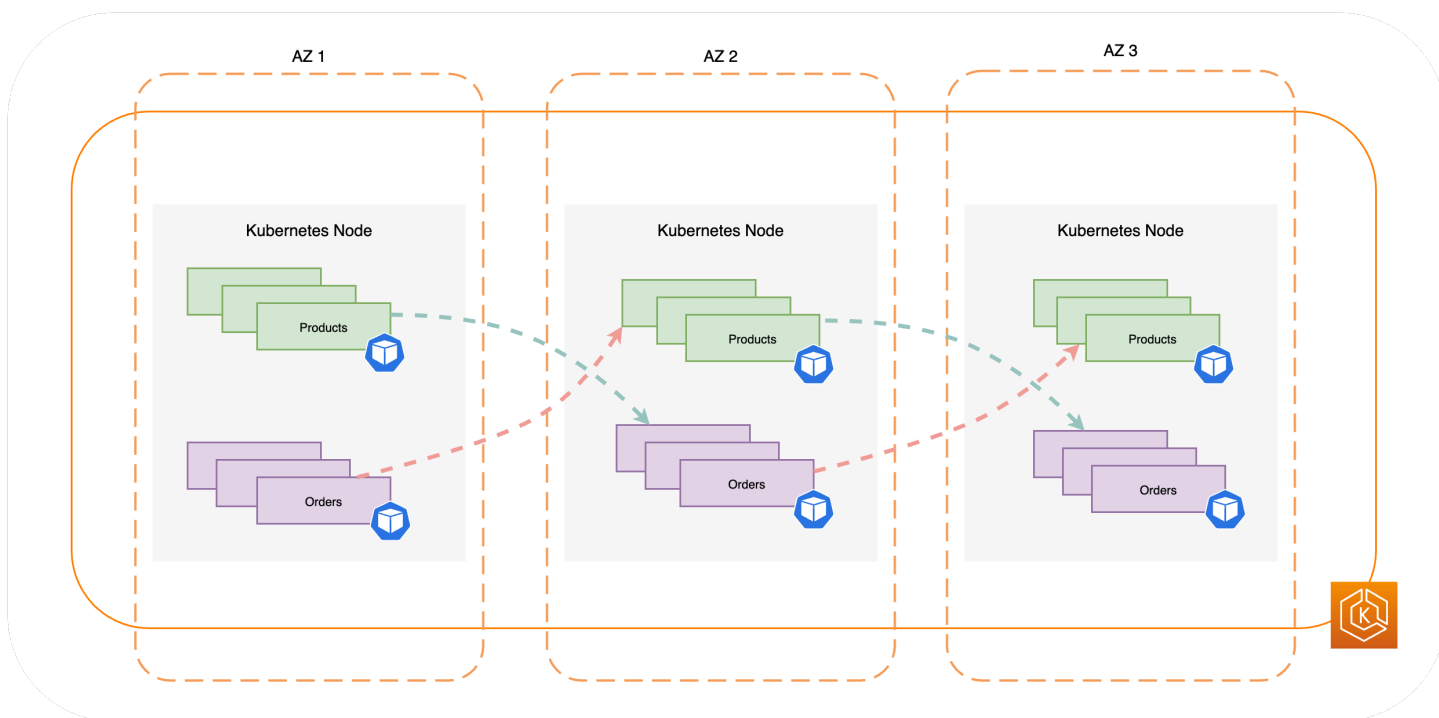
さらに、ゾーンの障害が発生した場合は、潜在的なコンピューティング容量の制約のリスクを軽減する必要があります。これにより、新しく必要なノードが正常な AZ の EKS データプレーンに追加されるのを防ぐことができます。

これを実現するには、特に環境内の AZ が 1 つ少ない場合に Kubernetes スケジューラーが新しいポッドの配置に使用できる既存の容量を持つように、各 AZ の一部のワーカーノードでコンピューティング容量をオーバープロビジョニングする必要があります。

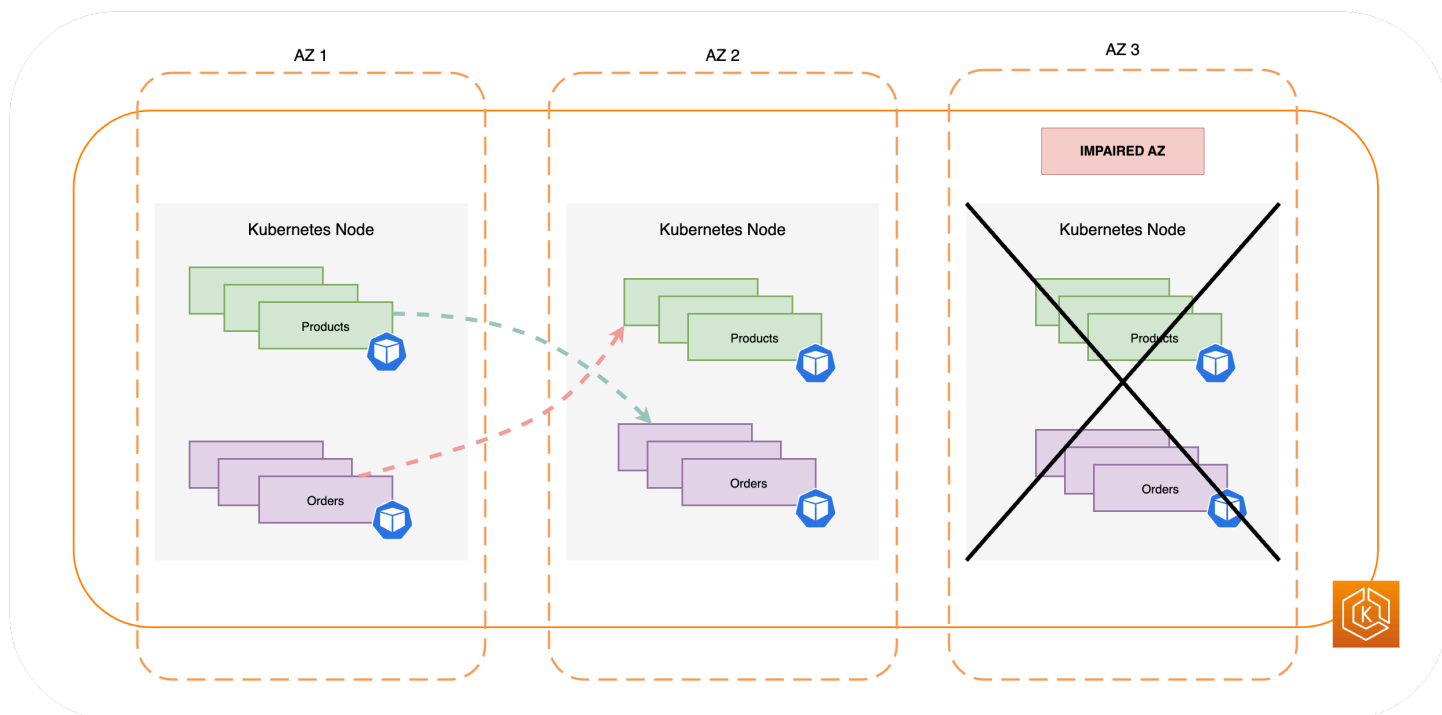
複数のポッドレプリカを AZ 間で実行して分散する

Kubernetes を使用すると、1 つのアプリケーションの複数のインスタンス (ポッドレプリカ) を実行してワークロードを事前スケールできます。アプリケーションに対して複数のポッドレプリカを実行すると、単一障害点がなくなり、1 つのレプリカへのリソースの負担が軽減されることで、全体的なパフォーマンスが向上します。ただし、アプリケーションの高可用性と耐障害性向上を実現するには、異なる障害ドメイン (トポロジドメイン、ここでは AZ) にまたがってアプリケーションの複数のレプリカを分散させて実行する必要があります。[トポロジ分散制約](#)では、アプリケーションが既存の静的安定性を持つように設定できるため、AZ の障害が発生した場合、正常な AZ に十分なレプリカがあり、発生する可能性があるトラフィックのスパイクや急増をすぐに処理できます。

下図は、すべての AZ が正常である場合の、EKS 環境と東西トラフィックフローを示しています。



下図は、1 つの AZ で障害が発生し、ゾーンシフトを開始したときの、東から西へのトラフィックフローを持つ EKS 環境を示しています。



以下のコードスニペットは、この Kubernetes 機能を使用してワークロードを設定する方法の例です。

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: orders
spec:
  replicas: 9
  selector:
    matchLabels:
      app: orders
  template:
    metadata:
      labels:
        app: orders
        tier: backend
    spec:
      topologySpreadConstraints:
        - maxSkew: 1
          topologyKey: "topology.kubernetes.io/zone"
          whenUnsatisfiable: ScheduleAnyway
          labelSelector:
            matchLabels:
```

```
app: orders
```

最も重要なのは、DNS サーバーソフトウェア (CoreDNS/kube-dns) の複数のレプリカを実行し、それらがデフォルトでまだ設定されていない場合は、同様のトポロジ分散制約を適用することです。これにより、正常な AZ に十分な DNS ポッドがあり、単一の AZ 障害がある場合にクラスター内の他の通信ポッドのサービス検出リクエストを引き続き処理できます。[CoreDNS EKS アドオン](#)には、複数の AZ に使用可能なノードがある場合に、クラスターのアベイラビリティゾーンに CoreDNS ポッドを分散するためのデフォルト設定があります。これらのデフォルト設定を独自のカスタム設定に置き換えることもできます。

[Helm で CoreDNS](#) をインストールする場合、[values.yaml ファイル](#)で `replicaCount` を更新して、各 AZ に十分な数のレプリカを確保できます。さらに、これらのレプリカがクラスター環境内の異なる AZ に分散されるようにするには、同じ `values.yaml` ファイルで `topologySpreadConstraints` プロパティを更新する必要があります。以下のコードスニペットは、これに対して CoreDNS を設定する方法を示しています。

CoreDNS Helm の `values.yaml`

```
replicaCount: 6
topologySpreadConstraints:
  - maxSkew: 1
    topologyKey: topology.kubernetes.io/zone
    whenUnsatisfiable: ScheduleAnyway
    labelSelector:
      matchLabels:
        k8s-app: kube-dns
```

AZ の障害が発生した場合は、CoreDNS の自動スケーリングシステムを使用して CoreDNS ポッドの負荷の増加を吸収できます。必要な DNS インスタンスの数は、クラスターで実行されているワークロードの数によって異なります。CoreDNS は CPU バインドであり、[Horizontal Pod Autoscaler \(HPA\)](#) を使用して CPU に基づいてスケールできます。以下は、ニーズに合わせて変更できる例です。

```
apiVersion: autoscaling/v1
kind: HorizontalPodAutoscaler
metadata:
  name: coredns
  namespace: default
spec:
  maxReplicas: 20
```

```
minReplicas: 2
scaleTargetRef:
  apiVersion: apps/v1
  kind: Deployment
  name: coredns
targetCPUUtilizationPercentage: 50
```

また、EKS は、CoreDNS の EKS アドオンバージョンで CoreDNS デプロイの自動スケーリングを管理できます。この CoreDNS オートスケーラーは、ノード数や CPU コア数など、クラスターの状態を継続的にモニタリングします。この情報に基づいて、コントローラーは EKS クラスター内の CoreDNS デプロイのレプリカ数を動的に調整します。

[CoreDNS EKS アドオン で自動スケーリング設定](#)を有効にするには、次のオプションの設定を追加する必要があります。

```
{
  "autoScaling": {
    "enabled": true
  }
}
```

[NodeLocal DNS](#) または [クラスター比例オートスケーラー](#) を使用して CoreDNS をスケールすることもできます。CoreDNS の水平スケーリングの詳細については、[ここ](#)を参照してください。

同じ AZ で相互依存ポッドを同じ場所に配置する

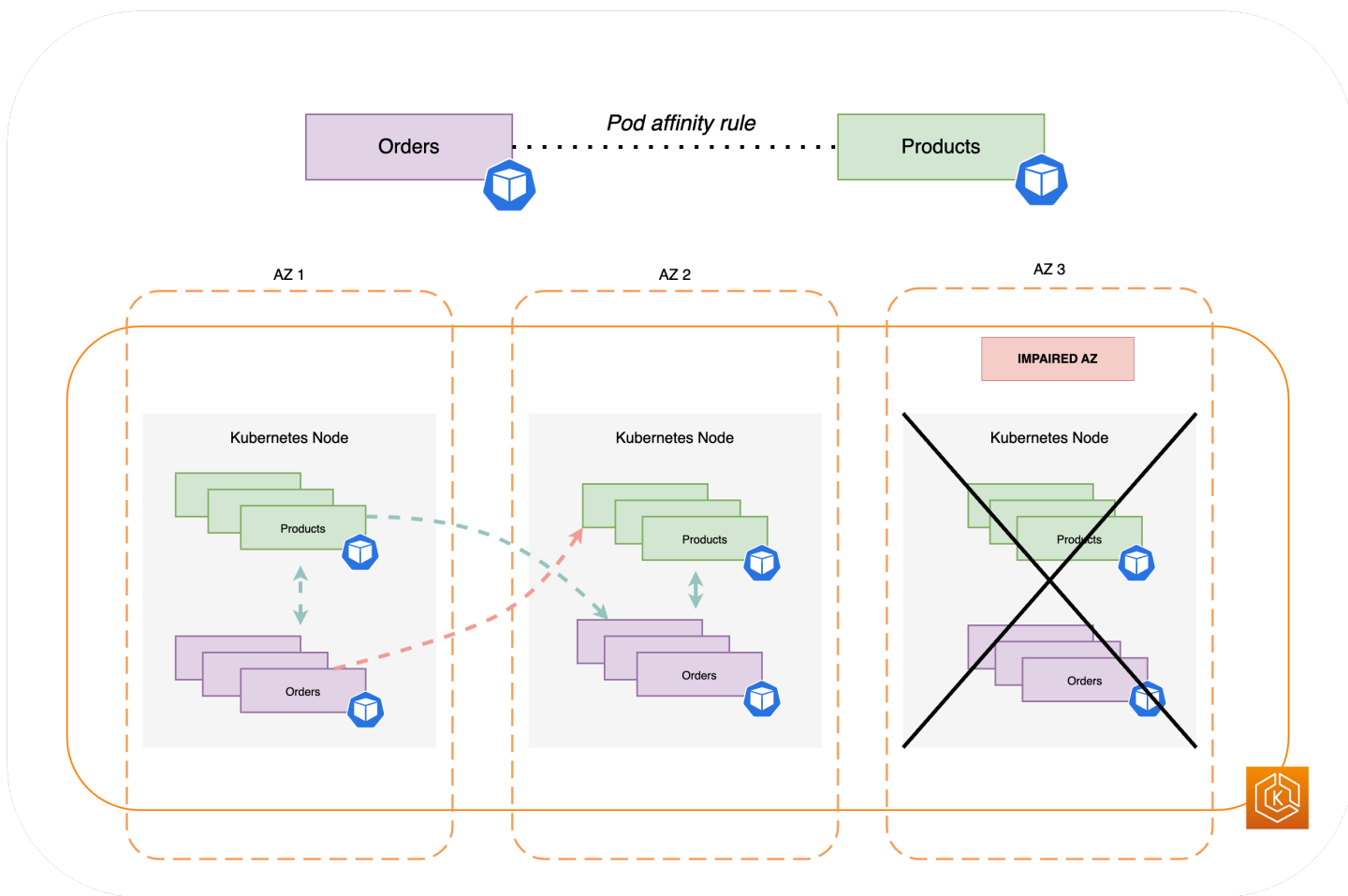
ほとんどの場合、エンドツーエンドプロセスの実行を成功させるために相互に通信する必要がある個別のワークロードを実行している可能性があります。異なるアプリケーションが異なる AZ に分散しているが、同じ AZ に配置されていない場合、単一の AZ の障害が基盤となるエンドツーエンドのプロセスに影響を与える可能性があります。例えば、アプリケーション A には AZ 1 と AZ 2 に複数のレプリカがあるが、アプリケーション B には AZ 3 にすべてのレプリカがある場合、AZ 3 が失われると、これら 2 つのワークロード (アプリケーション A と B) 間のエンドツーエンドプロセスに影響します。トポロジ分散制約とポッドのアフィニティを組み合わせると、すべての AZ にポッドを分散させ、特定のポッド間の関係を設定して、それらが一緒に配置されるようにすることで、アプリケーションの回復性を強化できます。

[ポッドのアフィニティのルール](#)を使用すると、ワークロード間の関係を定義して Kubernetes スケジューラーの動作に影響を与え、同じワーカーノードまたは同じ AZ にポッドを配置できます。これらのスケジューリング制約をどの程度厳格にするかを設定することもできます。

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: products
  namespace: ecommerce
  labels:
    app.kubernetes.io/version: "0.1.6"

  spec:
    serviceName: graphql-service-account
    affinity:
      podAffinity:
        requiredDuringSchedulingIgnoredDuringExecution:
          - labelSelector:
              matchExpressions:
                - key: app
                  operator: In
                  values:
                    - orders
            topologyKey: "kubernetes.io/hostname"
```

下図は、ポッドアフィニティルールを使用して同じノードに共に配置されたポッドを示しています。



クラスター環境が AZ の損失を処理できることをテストする

上記の要件を完了したら、次の重要なステップは、AZ の損失を処理するのに十分なコンピューティングとワークロード容量があることをテストすることです。これを行うには、EKS でゾーンシフトを手動でトリガーします。または、ゾーンオートシフトを有効にして練習実行を設定して、クラスター環境内の AZ を 1 つ減らしてアプリケーションが期待どおりに機能することをテストすることもできます。

よくある質問

この機能を使用する理由は何ですか？

EKS クラスターで ARC ゾーンシフトまたはゾーンオートシフトを使用することで、クラスター内ネットワークトラフィックを障害のある AZ から移行する迅速な復旧プロセスを自動化することで、Kubernetes アプリケーションの可用性をより適切に維持できます。ARC を使用すると、障害のある AZ イベント中に回復期間を延長することの多い、長く複雑なステップを回避できます。

この機能は他の AWS サービスとどのように連携しますか？

EKS は ARC と統合され、AWS で復旧オペレーションを実行するためのプライマリインターフェイスを提供します。クラスター内トラフィックが障害のある AZ から適切にルーティングされるように、Kubernetes データプレーンで実行されているポッドのネットワークエンドポイントのリストが変更されます。クラスターへの外部トラフィックのルーティングに AWS Load Balancer を使用している場合は、ロードバランサーを ARC に登録し、ゾーンシフトをトリガーして、トラフィックが機能低下したゾーンに流れるのを防ぐことができます。この機能は、EKS マネージド型ノードグループ (MNG) によって作成された Amazon EC2 Auto Scaling グループ (ASG) とも相互作用します。障害のある AZ が新しい Kubernetes ポッドまたはノードの起動に使用されるのを防ぐため、EKS は障害のある AZ を ASG から削除します。

この機能はデフォルトの Kubernetes 保護とどのように異なりますか？

この機能は、顧客が回復力を維持するのに役立ついくつかの Kubernetes ネイティブ組み込み保護と連携して機能します。ポッドの準備状況とライブネスプローブを設定して、ポッドがいつトラフィックを取るかを決定できます。これらのプローブで障害が発生すると、Kubernetes はこれらのポッドをサービスのターゲットとして削除し、トラフィックはポッドに送信されなくなります。これは便利ですが、ゾーンが機能低下したときに障害が発生することが保証されるように、これらのヘルスチェックを設定するのは簡単ではありません。ARC ゾーンシフト機能には、Kubernetes のネイティブ保護が十分でない場合に、機能低下した AZ を完全に分離するのに役立つ追加のセーフティネットが用意されています。また、アーキテクチャの運用準備状況と回復性を簡単にテストする方法も用意されています。

AWS はユーザーに代わってゾーンシフトをトリガーできますか？

はい。ARC ゾーンシフトを完全に自動化する方法が必要な場合は、ARC ゾーンオートシフトを有効にできます。ゾーンオートシフトでは、AWS を使用して EKS クラスターの AZ の状態を監視し、AZ の障害が検出されるとシフトを自動的にトリガーできます。

この機能を使用し、ワーカーノードとワークロードが事前にスケーリングされていない場合はどうなりますか？

ゾーンシフト中に事前スケールされておらず、追加のノードやポッドのプロビジョニングに依存している場合、復旧が遅れるリスクがあります。Kubernetes データプレーンに新しいノードを追加するプロセスには時間がかかるため、特にゾーンの障害が発生した場合に、アプリケーションのリアルタイムのパフォーマンスと可用性に影響を与える可能性があります。さらに、ゾーンの障害が発生した場合は、コンピューティング容量の制約により、新たに必要なノードが正常な AZ に追加できなくなる可能性があります。

ワークロードが事前にスケールされておらず、クラスター内のすべての AZ に分散されていない場合、ゾーンの障害により、影響を受ける AZ のワーカーノードでのみ実行されているアプリケーション

ンの可用性に影響する可能性があります。アプリケーションの可用性が完全に停止するリスクを軽減するため、EKS にはフェイルセーフ機能があります。この機能により、ワークロードのすべてのエンドポイントが障害が発生している AZ にある場合でも、障害ゾーンのポッドエンドポイントにトラフィックを送信することが可能です。ただし、ゾーンの問題が発生した場合でも可用性を維持するために、アプリケーションを事前にスケールしてすべての AZ に分散することを強くお勧めします。

ステートフルアプリケーションを実行している場合はどうなりますか？

ステートフルアプリケーションを実行している場合は、ユースケースとアーキテクチャに応じて耐障害性を評価する必要があります。アクティブ/スタンバイのアーキテクチャまたはパターンがある場合、アクティブが障害のある AZ にある可能性があります。アプリケーションレベルでは、スタンバイがアクティブ化されていない場合、アプリケーションに問題が発生する可能性があります。また、正常な AZ で新しい Kubernetes ポッドを起動すると、障害のある AZ にバインドされた永続的なボリュームにアタッチできないため、問題が発生する可能性があります。

この機能は Karpenter で機能しますか？

Karpenter のサポートは現在、ARC ゾーンシフトと EKS のゾーンオートシフトでは利用できません。AZ に障害がある場合は、異常な AZ を削除して、関連する Karpenter NodePool 設定を調整し、新しいワーカーノードが正常な AZ でのみ起動されるようにすることができます。

この機能は EKS Fargate で機能しますか？

この機能は EKS Fargate では機能しません。デフォルトでは、EKS Fargate がゾーンヘルスイベントを認識すると、ポッドは他の AZ での動作を優先します。

EKS マネージド Kubernetes コントロールプレーンは影響を受けますか？

いいえ。デフォルトでは、Amazon EKS は、複数の AZ で Kubernetes コントロールプレーンを実行およびスケールして、高可用性を実現します。ARC ゾーンシフトとゾーンオートシフトは、Kubernetes データプレーンでのみ動作します。

この新機能に関連するコストはありますか？

EKS クラスターでは、ARC ゾーンシフトとゾーンオートシフトを追加料金なしで使用できます。ただし、プロビジョニングされたインスタンスの料金は引き続きお支払いいただきます。この機能を使用する前に、Kubernetes データプレーンを事前にスケールすることを強くお勧めします。コストとアプリケーションの可用性の適切なバランスを考慮する必要があります。

その他のリソース

- [ゾーンシフトの仕組み](#)

- [ARC のゾーンシフトのベストプラクティス](#)
- [ゾーンシフトおよびゾーンオートシフトでサポートされているリソースとシナリオ](#)
- [Amazon EKS での回復力のあるワークロードの運用](#)
- [ポッドの優先度とオーバープロビジョニングにより Kubernetes ノードスケーリングの遅延を排除する](#)
- [CoreDNS ポッドを高い DNS トラフィックにスケールする](#)

EKS ゾーンシフトを有効にしてアベイラビリティゾーンの障害を回避する

Amazon Application Recovery Controller (ARC) は、アベイラビリティゾーン (AZ) 間でアプリケーションのリカバリを管理および調整し、Amazon EKS を含む多くのサービスと連携動作します。ARC ゾーンシフトの EKS サポートでは、クラスター内ネットワークトラフィックを障害のある AZ から移行できます。また、AWS に AZ の健全性を監視させ、異常な AZ から別の場所へ一時的にネットワークトラフィックを切り替える権限を付与することもできます。

EKS ゾーンシフトの使用方法:

1. Amazon Application Recovery Controller (ARC) で EKS クラスターを有効にします。これは、Amazon EKS コンソール、AWS CLI、CloudFormation、または eksctl を使用してクラスターレベルで行われます。
2. 有効にすると、ARC コンソール、AWS CLI、またはゾーンシフト API およびゾーンオートシフト API を使用して、ゾーンシフトまたはゾーンオートシフトを管理できます。

EKS クラスターを ARC に登録した後も、ARC を設定する必要があります。例えば、ARC コンソールを使用してゾーンオートシフトを設定できます。

EKS ゾーンシフトの仕組みと、障害のあるアベイラビリティゾーンを処理するためのワークロードの設計方法についての詳細は、「[the section called “ゾーンシフトの詳細”](#)」を参照してください。

考慮事項:

- EKS Auto Mode は Amazon Application Recovery Controller、Zone Shift、Zonal Autoshift をサポートしていません

Amazon Application Recovery Controller とは

Amazon Application Recovery Controller (ARC) は、AWS 上で実行しているアプリケーションのリカバリ操作の準備や迅速な実行に役立ちます。ゾーンシフトを使用すると、サポートされているリソースのトラフィックを アベイラビリティゾーン (AZ) から AWS リージョン内の正常な AZ に一時的に移動することで、AZ の障害からすばやく回復できます。

[Amazon Application Recovery Controller \(ARC\) の詳細はこちら](#)

ゾーンシフトとは

ゾーンシフトは ARC の機能であり、EKS クラスターや Elastic Load Balancer などのリソースのトラフィックを AWS リージョンのアベイラビリティゾーンから移動できます。これにより、問題をすばやく軽減し、アプリケーションをすばやく回復できます。例えば、不適切なデプロイが原因でレイテンシーの問題が発生していたり、アベイラビリティゾーンで障害が発生していたりする場合、トラフィックをシフトすることを選択できます。ゾーンシフトでは、事前設定手順は必要ありません。

[ARC ゾーンシフトの詳細はこちら](#)

ゾーンオートシフトとは

ゾーンオートシフトは ARC の機能であり、有効にすると AWS がユーザーに代わってサポートされているリソースのトラフィックを AZ から AWS リージョンの正常な AZ に移動できます。AWS は、顧客に影響を与える可能性のある、リージョン内の AZ の障害が内部テレメトリによって示されると、オートシフトを開始します。内部テレメトリには、AWS ネットワーク、Amazon EC2、Elastic Load Balancing サービスなど、複数のソースからのメトリクスが組み込まれています。

AWS は、インジケータが問題や潜在的な問題がなくなったことを示すと、オートシフトを終了します。

[ARC ゾーンオートシフトの詳細はこちら](#)

オートシフト中の EKS の動作

EKS は、障害のある AZ へのトラフィックを停止するようにネットワーク設定を更新します。さらに、マネージドノードグループを使用している場合、EKS はゾーンシフト中に正常な AZ にある新しいノードのみを起動します。シフトの有効期限が切れたりキャンセルされたりすると、以前に異常として検出された AZ を含めるようにネットワーク設定が復元されます。

[EKS ゾーンシフトの詳細はこちら](#)

EKS クラスターを Amazon Application Recovery Controller (ARC) に登録する (AWS コンソール)

1. ARC に登録する EKS クラスターの名前とリージョンを見つけます。
2. そのリージョンの [EKS コンソール](#) に移動し、クラスターを選択します。
3. [クラスター情報] ページの [概要] タブを選択します。
4. ゾーンシフトの見出しで、[管理] ボタンを選択します。
5. [EKS ゾーンシフト] に [有効] または [無効] を選択します。

これで、EKS クラスターが ARC に登録されました。

AWS にアベイラビリティーゾーンの障害を検出して回避させるには、ARC ゾーンオートシフトを設定する必要があります。例えば、ARC コンソールで以下のことを行うことができます。

次のステップ

- [ゾーンオートシフトを有効にする](#) 方法の詳細
- [ゾーンシフトを手動で開始](#) する方法の詳細

Amazon EKS でのアクセスコントロールの仕組みについて説明します。

Amazon EKS クラスターへのアクセスを管理する方法について説明します。Amazon EKS を使用するには、Kubernetes と AWS Identity and Access Management (AWS IAM) がどのようにアクセスコントロールをするのかについての知識が必要になります。

このセクションには以下が含まれます。

[the section called “Kubernetes API アクセス”](#) — アプリケーションまたはユーザーが Kubernetes API を認証できるようにする方法について説明します。アクセスエントリ、aws-auth ConfigMap、または外部の OIDC プロバイダーを使用できます。

[the section called “クラスターリソースにアクセスする”](#) — Amazon EKS クラスターと通信するように AWS Management Consoleを設定する方法について説明します。コンソールを使用して、名前空間、ノード、Pod など、クラスター内の Kubernetes リソースを表示します。

[the section called “kubectl を使用してクラスターにアクセスする”](#) — Amazon EKS クラスターと通信するように kubectl を設定する方法について説明します。AWS CLI を使用して kubeconfig ファイルを作成します。

[the section called “AWS へのワークロードアクセス”](#) —Kubernetes サービスアカウントを AWS IAM ロールに関連付ける方法について説明します。サービスアカウント (IRSA) には、Pod Identity または IAM ロールを使用できます。

一般的なタスク

- 開発者に Kubernetes API へのアクセスを許可します。AWS Management Consoleに Kubernetes リソースを表示します。
 - 解決策: [アクセスエントリを使用](#)して、Kubernetes RBAC アクセス許可を AWS IAM ユーザーまたはロールに関連付けます。
- AWS 認証情報を使用して、kubectl を Amazon EKS クラスターと通信するように設定します。
 - 解決策: AWS CLI を使用して [kubeconfig ファイルを作成](#)します。
- Ping Identity など外部 ID プロバイダーを使用して、Kubernetes API に対してユーザーを認証します。

- 解決策: [外部の OIDC プロバイダーをリンクします](#)。
- Kubernetes クラスターのワークロードに AWS API を呼び出す機能を付与します。
- 解決策: [Pod Identity を使用](#)して AWS IAM ロールを Kubernetes サービスアカウントに関連付けます。

背景

- [Kubernetes サービスアカウントの仕組みを学びます](#)。
- [Kubernetes ロールベースアクセスコントロール \(RBAC\) モデルを確認します](#)。
- AWS リソースへのアクセスを管理する方法については、「[AWS IAM ユーザーガイド](#)」を参照してください。あるいは、無料の [AWS IAM の使用に関する入門トレーニング](#)も受講できます。

EKS Auto Mode に関する考慮事項

EKS Auto Mode は、EKS Pod Identity および EKS EKS アクセスエントリと統合されます。

- EKS Auto Mode は、アクセスエントリを使用して EKS コントロールプレーンに Kubernetes アクセス許可を付与します。例えば、アクセスポリシーにより、EKS Auto Mode はネットワークエンドポイントとサービスに関する情報を読み取ることができます。
- EKS Auto Mode クラスターのアクセスエントリを無効にすることはできません。
- 任意で aws-auth ConfigMap を有効にできます。
- EKS Auto Mode のアクセスエントリは自動的に設定されます。このアクセスエントリは表示できますが、変更することはできません。
- NodeClass を使用してカスタムノード IAM ロールを作成する場合は、AmazonEKSAutoNodePolicy アクセスポリシーを使用してロールのアクセスエントリを作成する必要があります。
- AWS サービスのアクセス許可をワークロードに付与する場合は、EKS Pod Identity を使用します。
- EKS Auto Mode クラスターに Pod Identity エージェントをインストールする必要はありません。

IAM ユーザーおよびロールに Kubernetes API へのアクセスを付与する

クラスターには Kubernetes API エンドポイントがあります。Kubectl はこの API を使用します。この API は、次の 2 種類の ID を使用して認証することができます。

- AWS Identity and Access Management (IAM) プリンシパル (ロールまたはユーザー) — このタイプには IAM への認証が必要です。[IAM](#) ユーザーとして、または ID ソースから提供された認証情報を使用して、[フェデレーテッド ID](#) で AWS にサインインできます。IAM ロールを使用して前もって管理者により ID フェデレーションが設定されている場合、フェデレーテッド ID としてのみサインインできます。フェデレーションを使用して AWS にアクセスする場合、間接的に[ロールを引き受ける](#)ことになります。ユーザーがこのタイプの ID を使用すると、次のことが行われます。
- ユーザーがクラスター上の Kubernetes オブジェクトを操作できるように Kubernetes アクセス許可を割り当てることができます。IAM プリンシパルにアクセス許可を割り当てて、クラスター上の Kubernetes オブジェクトにアクセスできるようにする方法の詳細については、「[the section called “許可を付与する”](#)」を参照してください。
- IAM アクセス許可を割り当てて、ユーザーが Amazon EKS API、AWS CLI、AWS CloudFormation、AWS Management Console、または `eksctl` を使用して Amazon EKS クラスターとそのリソースを操作できるようにすることができます。詳細については、サービス認可リファレンスの「[Amazon Elastic Kubernetes Service で定義されるアクション](#)」を参照してください。
- ノードは IAM ロールを引き受けることでクラスターに参加します。IAM プリンシパルを使用してクラスターにアクセスできます。プリンシパルは、Amazon EKS コントロールプレーンで実行される [AWS IAM Authenticator for Kubernetes](#) によって提供されます。
- 独自の OpenID Connect (OIDC) プロバイダーのユーザー — このタイプでは、[OIDC](#) プロバイダーへの認証が必要です。Amazon EKS クラスターで独自の OIDC プロバイダーを設定する方法については、「[the section called “OIDC プロバイダーをリンクする”](#)」を参照してください。ユーザーがこのタイプの ID を使用すると、次のことが行われます。
- ユーザーがクラスター上の Kubernetes オブジェクトを操作できるように Kubernetes アクセス許可を割り当てることができます。
- IAM アクセス許可を割り当てて、ユーザーが Amazon EKS API、AWS CLI、AWS CloudFormation、AWS Management Console、または `eksctl` を使用して Amazon EKS クラスターとそのリソースを操作することはできません。

クラスターでは両方のタイプの ID を使用できます。IAM の認証方法を無効にすることはできません。OIDC 認証は任意です。

IAM アイデンティティと Kubernetes のアクセス許可を関連付ける

[AWS IAM Authenticator for Kubernetes](#) はクラスターのコントロールプレーンにインストールされます。これにより、クラスター上の Kubernetes リソースへのアクセスを許可する [AWS Identity and Access Management](#) (IAM) プリンシパル (ロールとユーザー) が有効になります。次のいずれかの方法を使用して、IAM プリンシパルにクラスター上の Kubernetes オブジェクトへのアクセスを許可できます。

- アクセスエントリの作成 — クラスターが、クラスターの Kubernetes バージョンの「[前提条件](#)」セクションに記載されているプラットフォームバージョンと同じかそれ以降の場合は、このオプションを使用することをお勧めします。

アクセスエントリを使用して、クラスター外から IAM プリンシパルの Kubernetes アクセス許可を管理します。クラスターへのアクセスの追加および管理には、EKS API、AWS コマンドラインインターフェイス、AWS SDK、AWS CloudFormation、および AWS Management Console を使用できます。つまり、クラスターを作成したのと同じツールでユーザーを管理できるということです。

開始するには、「[アクセスエントリを使用するように認証モードを変更する](#)」に従い、次に「[既存の aws-auth ConfigMap エントリをアクセスエントリに移行する](#)」に従います。

- **aws-auth ConfigMap** へのエントリの追加 — クラスターのプラットフォームバージョンが「[前提条件](#)」セクションに記載されているバージョンよりも前の場合は、このオプションを使用する必要があります。クラスターのプラットフォームバージョンが、クラスターの Kubernetes バージョンの「[前提条件](#)」セクションに記載されているプラットフォームバージョンと同じかそれ以降で、ConfigMap にエントリを追加した場合は、それらのエントリをアクセスエントリに移行することをお勧めします。ただし、マネージド型ノードグループで使用する IAM ロールのエントリや Fargate プロファイルなど、Amazon EKS が ConfigMap に追加したエントリは移行できません。詳細については、「[the section called “Kubernetes API アクセス”](#)」を参照してください。
- aws-auth ConfigMap オプションを使用する必要がある場合は、`eksctl create iamidentitymapping` コマンドを使用してエントリを ConfigMap に追加できます。詳細については、eksctl ドキュメントの「[IAM ユーザーとロールの管理](#)」を参照してください。

クラスター認証モードを設定する

各クラスターには認証モードがあります。認証モードによって、IAM プリンシパルがクラスター上の Kubernetes オブジェクトにアクセスできるようにするために使用できる方法が決まります。認証モードは 3 つあります。

Important

アクセス入力メソッドを一度有効にすると、無効にすることはできません。ConfigMap メソッドがクラスターの作成時に有効になっていない場合、後で有効にすることはできません。アクセスエントリの導入前に作成されたすべてのクラスターでは、ConfigMap メソッドが有効になっています。クラスターでハイブリッドノードを使用している場合は、API または API_AND_CONFIG_MAP クラスター認証モードを使用する必要があります。

クラスター内部の **aws-auth ConfigMap**

これは Amazon EKS クラスターのオリジナルの認証モードです。クラスターを作成した IAM プリンシパルは、`kubectl` を使用してクラスターにアクセスできる初期ユーザーです。初期ユーザーは、aws-auth ConfigMap のリストに他のユーザーを追加し、クラスター内の他のユーザーに影響を与えるアクセス許可を割り当てる必要があります。ConfigMap には管理するエントリがないため、これらの他のユーザーは初期ユーザーを管理したり削除したりできません。

ConfigMap とアクセスエントリの両方

この認証モードでは、両方の方法を使用して IAM プリンシパルをクラスターに追加できます。それぞれの方法には別々のエントリが保存されることに注意してください。例えば、AWS CLI からアクセスエントリを追加しても、aws-auth ConfigMap は更新されません。

アクセスエントリのみ

この認証モードでは、EKS API、AWS コマンドラインインターフェイス、AWS SDK、AWS CloudFormation、および AWS Management Console を使用して IAM プリンシパルのクラスターへのアクセスを管理できます。

各アクセスエントリにはタイプがあり、プリンシパルを特定の名前空間に制限するアクセススコープと、事前設定された再利用可能なアクセス許可ポリシーを設定するアクセスポリシーを組み合わせて使用できます。また、STANDARD タイプと Kubernetes RBAC グループを使用してカスタムアクセス許可を割り当てることもできます。

[Authentication mode] (認証モード)	方法
ConfigMap のみ (CONFIG_MAP)	aws-auth ConfigMap
EKS API と ConfigMap (API_AND_CONFIG_MAP)	EKS API、AWS コマンドラインインターフェイス、AWS SDK、AWS CloudFormation、AWS Management Console、および aws-auth ConfigMap のアクセスエントリ
EKS API のみ (API)	EKS API、AWS コマンドラインインターフェイス、AWS SDK、AWS CloudFormation、および AWS Management Console のアクセスエントリ

Note

Amazon EKS Auto Mode にはアクセスエントリが必要です。

EKS アクセスエントリを使用して Kubernetes へのアクセスを IAM ユーザーに許可する

EKS アクセスエントリとは

EKS アクセスエントリはユーザーに Kubernetes API へのアクセスを許可する最善の方法です。例えば、アクセスエントリを使用して、開発者に kubectl を使用するアクセス権を付与できます。

基本的に、EKS アクセスエントリは一連の Kubernetes アクセス許可を IAM ロールなどの IAM アイデンティティに関連付けます。例えば、開発者は IAM ロールを引き受け、それを使用して EKS クラスターを認証できます。

Kubernetes アクセス許可をアタッチして、次の 2 つの方法でエントリにアクセスできます：

- アクセス ポリシーを使用します。アクセスポリシーは、AWS によって管理される事前定義された Kubernetes アクセス許可テンプレートです。詳細については「[the section called “アクセスポリシーを確認する”](#)」を参照してください。

- Kubernetes グループを参照してください。IAM アイデンティティを Kubernetes グループに関連付けると、グループのアクセス許可を付与する Kubernetes リソースを作成できます。詳細については Kubernetes ドキュメントの「[RBAC 認可を使用する](#)」を参照してください。

利点

Amazon EKS クラスターアクセス管理を使用すると、Amazon EKS API を介して Kubernetes クラスターの認証と認可を直接制御できます。この機能により、ユーザーのアクセス許可を管理するときに AWS API と Kubernetes API を切り替える必要がなくなり、アクセス管理が簡素化されます。アクセスエントリとアクセスポリシーを使用して、クラスター作成者からのクラスター管理者権限を変更または取り消す機能など、IAM AWS プリンシパルの詳細な権限を定義できます。

この機能は、AWS CloudFormation、Terraform、AWS CDK などの Infrastructure as Code (IaC) ツールと統合され、クラスターの作成時にアクセス構成を定義できます。設定ミスが発生した場合は、Kubernetes API に直接アクセスする必要なしに、Amazon EKS API を介してクラスターアクセスを復元できます。この一元化されたアプローチは CloudTrail 監査ログ記録や多要素認証などの既存の AWS IAM 機能を活用することで、運用上のオーバーヘッドを削減し、セキュリティを向上させます。

使用を開始する

1. 使用する IAM アイデンティティとアクセスポリシーを決定します。
 - [the section called “アクセスポリシーを確認する”](#)
2. クラスターで EKS アクセスエントリを有効にします。サポートされているプラットフォームバージョンがあることを確認します。
 - [the section called “\[Authentication mode\] \(認証モード\)”](#)
3. IAM アイデンティティを Kubernetes 権限に関連付けるアクセスエントリを作成します。
 - [the section called “アクセスエントリを作成する”](#)
4. IAM アイデンティティを使用してクラスターを認証します。
 - [the section called “AWS CLI の設定”](#)
 - [the section called “kubectl と eksctl のセットアップ”](#)

レガシークラスターアクセス設定

この機能が導入される前に作成されたクラスター (プラットフォームバージョン要件で指定されているものよりも前の初期プラットフォームバージョンを持つクラスター) で EKS アクセスエントリを

有効にすると、EKS は既存のアクセス許可を反映するアクセスエントリを自動的に作成します。このアクセスエントリには以下が表示されます。

- クラスターを最初に作成した IAM アイデンティティ
- クラスターの作成時にそのアイデンティティに付与される管理アクセス許可

 Note

以前はこの管理アクセスは自動的に付与され、変更できませんでした。EKS アクセス エントリを有効にすると、このレガシー アクセス構成を表示および削除できるようになります。

アクセスポリシーをアクセスエントリに関連付ける

1 つ以上のアクセスポリシーを タイプ STANDARD のアクセスエントリに割り当てることができます。Amazon EKS は、クラスターで正しく機能するために必要なアクセス許可を他のタイプのアクセスエントリに自動的に付与します。Amazon EKS アクセスポリシーには、IAM アクセス許可ではなく Kubernetes アクセス許可が含まれます。アクセスポリシーをアクセスエントリに関連付ける前に、各アクセスポリシーに含まれる Kubernetes アクセス許可を既に理解できていることを確かめてください。詳細については、「[the section called “アクセスポリシーを確認する”](#)」を参照してください。どのアクセスポリシーも要件を満たさない場合は、アクセスポリシーをアクセスエントリに関連付けしないでください。代わりに、アクセスエントリに 1 つ以上のグループ名を指定し、Kubernetes ロールベースのアクセス制御オブジェクトを作成して管理します。詳細については、「[the section called “アクセスエントリを作成する”](#)」を参照してください。

- 既存のアクセスエントリ。作成する場合は「[the section called “アクセスエントリを作成する”](#)」を参照してください。
- ListAccessEntries、DescribeAccessEntry、UpdateAccessEntry、ListAccessPolicies、および DisassociateAccessPolicy のアクセス許可を持つ AWS Identity and Access Management ロールまたはユーザー。詳細については、「サービス認証リファレンス」の「[Actions defined by Amazon Elastic Kubernetes Service](#)」を参照してください。

アクセスポリシーをアクセスエントリに関連付ける前に、以下の要件を考慮してください。

- 各アクセスエントリには複数のアクセスポリシーを関連付けることができますが、各ポリシーを 1 つのアクセスエントリに関連付けることができるのは 1 回だけです。複数のアクセスポリシー

を関連付ける場合、アクセスエントリの IAM プリンシパルには、関連するすべてのアクセスポリシーに含まれるすべてのアクセス許可が付与されます。

- アクセスポリシーをクラスター上のすべてのリソースに範囲指定することも、1 つ以上の Kubernetes 名前空間の名前を指定して範囲指定することもできます。名前空間の名前にはワイルドカード文字を使用できます。例えば、dev- で始まるすべての名前空間にアクセスポリシーを適用する場合は、名前空間の名前として dev-* を指定できます。名前空間がクラスターに存在し、スペルがクラスター上の実際の名前空間の名前と一致していることを確認してください。Amazon EKS は、クラスター上の名前空間のスペルや存在を確認しません。
- アクセスポリシーをアクセスエントリに関連付けた後、アクセスポリシーのアクセス範囲を変更できます。アクセスポリシーの範囲を Kubernetes 名前空間に設定した場合は、必要に応じて関連付けの名前空間を追加および削除できます。
- グループ名も指定されているアクセスエントリにアクセスポリシーを関連付けると、IAM プリンシパルには、関連するすべてのアクセスポリシーのすべてのアクセス許可が付与されます。また、グループ名を指定する Kubernetes Role および RoleBinding オブジェクトで指定されている Kubernetes Role または ClusterRole オブジェクトのすべてのアクセス許可も保持されます。
- `kubectl auth can-i --list` コマンドを実行しても、コマンドを実行したときに使用している IAM プリンシパルのアクセスエントリに関連付けられているアクセスポリシーによって割り当てられた Kubernetes アクセス許可は表示されません。このコマンドは、アクセスエントリに指定したグループ名またはユーザー名にバインドした Kubernetes Role または ClusterRole オブジェクトでアクセス許可を付与した場合にのみ、Kubernetes アクセス許可を表示します。
- `--as username` または `--as-group group-name` で `kubectl` コマンドを使用するなど、クラスター上の Kubernetes オブジェクトを操作する際に Kubernetes ユーザーまたはグループになりすますと、Kubernetes RBAC 認可の使用を強制することになります。その結果、IAM プリンシパルには、アクセスエントリに関連付けられたアクセスポリシーによって割り当てられるアクセス許可はありません。IAM プリンシパルを偽装するユーザーまたはグループが持つ Kubernetes アクセス許可は、グループ名またはユーザー名にバインドした Kubernetes Role または ClusterRole オブジェクトで付与した Kubernetes アクセス許可だけです。IAM プリンシパルが関連するアクセスポリシー内のアクセス許可を持つようにするには、Kubernetes ユーザーまたはグループを偽装しないでください。IAM プリンシパルには、アクセスエントリに指定したグループ名またはユーザー名にバインドした Kubernetes Role または ClusterRole オブジェクトで付与したアクセス許可も引き続き保持されます。詳細については、Kubernetes ドキュメントの「[ユーザーの偽装](#)」を参照してください。

AWS Management Console または AWS CLI を使用してアクセスポリシーをアクセスエントリに関連付けることができます。

AWS Management Console

1. [アマゾン EKS コンソール](#)を開きます。
2. アクセスポリシーを関連付けるアクセスエントリがあるクラスターの名前を選択します。
3. [リモートアクセス] タブを選択してください。
4. アクセスエントリのタイプが標準の場合は、Amazon EKS アクセスポリシーを関連付けたり関連付けを解除したりできます。アクセスエントリのタイプが標準以外の場合、このオプションは使用できません。
5. [アクセスポリシーを関連付ける] を選択します。
6. [ポリシー名] には、IAM プリンシパルに付与したいアクセス許可を持つポリシーを選択します。各ポリシーに含まれるアクセス許可を表示するには、「[the section called “アクセスポリシーを確認する”](#)」を参照してください。
7. [アクセススコープ] では、アクセス範囲を選択します。[クラスター] を選択すると、すべての Kubernetes 名前空間のリソースについて、アクセスポリシー内のアクセス許可が IAM プリンシパルに付与されます。[Kubernetes 名前空間] を選択した場合は、[新しい名前空間の追加] を選択できます。表示される [名前空間] フィールドには、クラスター上の Kubernetes 名前空間の名前を入力できます。IAM プリンシパルに複数の名前空間のアクセス許可を持たせたい場合は、複数の名前空間を入力できます。
8. [アクセスポリシーを追加] を選択します。

AWS CLI

1. ご使用のデバイスまたは AWS CloudShell で、バージョン 2.12.3 以降、または AWS コマンドラインインターフェイス (AWS CLI) のバージョン 1.27.160 以降がインストールおよび設定されていること。現在のバージョンを確認するには「`aws --version | cut -d / -f2 | cut -d ' ' -f1`」を参照してください。yum、apt-get、macOS 用の Homebrew などのパッケージマネージャーは、多くの場合 AWS CLI の最新バージョンより数バージョン古くなっています。最新バージョンをインストールするには「AWS コマンドラインインターフェイスユーザーガイド」の「[インストール](#)」および「[aws configure を使用したクイック設定](#)」を参照してください。AWS クラウドシェル にインストールされている AWS CLI バージョンも最新バージョンより数バージョン遅れることがあります。更新するには、「AWS CloudShell ユーザーガイド」の「[ホームディレクトリへの AWS CLI のインストール](#)」を参照してください。
2. 利用可能なアクセスポリシーを表示します。

```
aws eks list-access-policies --output table
```


出力例は次のとおりです。

```
-----
|                                     ListAccessPolicies
|
+-----+
+
||                                     accessPolicies
||
|+-----+
+-----+|
||                                     arn
name                                     ||
|+-----+
+-----+|
|| {arn-aws}eks::aws:cluster-access-policy/AmazonEKSAAdminPolicy |
AmazonEKSAAdminPolicy ||
|| {arn-aws}eks::aws:cluster-access-policy/AmazonEKSClusterAdminPolicy |
AmazonEKSClusterAdminPolicy ||
|| {arn-aws}eks::aws:cluster-access-policy/AmazonEKSEditPolicy |
AmazonEKSEditPolicy ||
|| {arn-aws}eks::aws:cluster-access-policy/AmazonEKSVIEWPolicy |
AmazonEKSVIEWPolicy ||
|+-----+
+-----+|
```

各ポリシーに含まれるアクセス許可を表示するには、「[the section called “アクセスポリシーを確認する”](#)」を参照してください。

3. 既存のアクセスエントリを表示します。##### の部分は自分のクラスター名に置き換えます。

```
aws eks list-access-entries --cluster-name my-cluster
```

出力例は次のとおりです。

```
{
  "accessEntries": [
    "arn:aws:iam::111122223333:role/my-role",
    "arn:aws:iam::111122223333:user/my-user"
  ]
}
```

4. アクセスポリシーをアクセスエントリに関連付けます。次の例では、AmazonEKSVIEWPolicy アクセスポリシーをアクセスエントリに関連付けます。*my-role* IAM ロールがクラスター上の Kubernetes オブジェクトにアクセスしようとするたびに、Amazon EKS は、ポリシー内のアクセス許可を使用して *my-namespace1* と *my-namespace2* の Kubernetes 名前空間のみの Kubernetes オブジェクトにアクセスするアクセス許可をロールに付与します。*my-cluster* をクラスターの名前、*111122223333* を AWS アカウント ID に、*my-role* を Amazon EKS に Kubernetes クラスターオブジェクトへのアクセスを許可してほしい IAM ロールの名前に置き換えます。

```
aws eks associate-access-policy --cluster-name my-cluster --principal-arn
arn:aws:iam::111122223333:role/my-role \
    --access-scope type=namespace,namespaces=my-namespace1,my-namespace2 --policy-arn
arn:aws:eks::aws:cluster-access-policy/AmazonEKSVIEWPolicy
```

IAM プリンシパルにクラスター全体のアクセス許可を持たせたい場合

は、`type=namespace,namespaces=my-namespace1,my-namespace2` を `type=cluster` に置き換えてください。複数のアクセスポリシーをアクセスエントリに関連付ける場合は、それぞれに固有のアクセスポリシーを指定してコマンドを複数回実行します。関連するアクセスポリシーにはそれぞれ独自の範囲があります。

Note

関連するアクセスポリシーの範囲を後で変更する場合は、新しい範囲で前のコマンドをもう一度実行してください。例えば、*my-namespace2* を削除したい場合は、`type=namespace,namespaces=my-namespace1` のみを使用してコマンドを再実行します。範囲を namespace から cluster に変更する場合は、`type=cluster` を使用して `type=namespace,namespaces=my-namespace1,my-namespace2` を削除してコマンドを再実行します。

5. どのアクセスポリシーをアクセスエントリに関連付けるかを決定します。

```
aws eks list-associated-access-policies --cluster-name my-cluster --principal-arn
arn:aws:iam::111122223333:role/my-role
```

出力例は次のとおりです。

```
{
  "clusterName": "my-cluster",
```

```

    "principalArn": "arn:aws:iam::111122223333",
    "associatedAccessPolicies": [
      {
        "policyArn": "arn:aws:eks::aws:cluster-access-policy/
AmazonEKSVIEWPolicy",
        "accessScope": {
          "type": "cluster",
          "namespaces": []
        },
        "associatedAt": "2023-04-17T15:25:21.675000-04:00",
        "modifiedAt": "2023-04-17T15:25:21.675000-04:00"
      },
      {
        "policyArn": "arn:aws:eks::aws:cluster-access-policy/
AmazonEKSAAdminPolicy",
        "accessScope": {
          "type": "namespace",
          "namespaces": [
            "my-namespace1",
            "my-namespace2"
          ]
        },
        "associatedAt": "2023-04-17T15:02:06.511000-04:00",
        "modifiedAt": "2023-04-17T15:02:06.511000-04:00"
      }
    ]
  }
}

```

前の例では、このアクセスエントリの IAM プリンシパルには、クラスター上のすべての名前空間に対する表示アクセス許可と、2 つの Kubernetes 名前空間に対する管理者アクセス許可があります。

6. アクセスポリシーとアクセスエントリの関連付けを解除します。この例では、AmazonEKSAAdminPolicy ポリシーとアクセスエントリの関連付けが解除されます。ただし、IAM プリンシパルは *my-namespace1* と *my-namespace2* 名前空間のオブジェクトに対する AmazonEKSVIEWPolicy アクセスポリシー内のアクセス許可を保持します。これは、そのアクセスポリシーがアクセスエントリとの関連付けを解除されていないためです。

```

aws eks disassociate-access-policy --cluster-name my-cluster --principal-arn
arn:aws:iam::111122223333:role/my-role \
  --policy-arn arn:aws:eks::aws:cluster-access-policy/AmazonEKSAAdminPolicy

```

利用可能なアクセスポリシーを一覧表示するには、「[the section called “アクセスポリシーを確認する”](#)」を参照してください。

既存の **aws-auth ConfigMap** エントリをアクセスエントリに移行する

クラスター上の aws-auth ConfigMap にエントリを追加した場合は、aws-auth ConfigMap 内の既存のエントリのアクセスエントリを作成することをお勧めします。アクセスエントリを作成したら、そのエントリを ConfigMap から削除できます。aws-authConfigMap のエントリに[アクセスポリシー](#)を関連付けることはできません。アクセスポリシーを IAM プリンシパルに関連付けるには、アクセスエントリを作成します。

Important

[マネージドノードグループ](#)または [Fargate プロファイル](#)をクラスターに追加したときに Amazon EKS によって作成された既存の aws-auth ConfigMap エントリは削除しないでください。Amazon EKS が ConfigMap で作成したエントリを削除すると、クラスターは正しく機能しなくなります。ただし、[セルフマネージド型](#)ノードグループのアクセスエントリを作成した後で、それらのエントリを削除することはできます。

前提条件

- アクセスエントリとアクセスポリシーに関する知識。詳細については、「[the section called “許可を付与する”](#)」および「[the section called “アソシエイトアクセスポリシー”](#)」を参照してください。
- 「[the section called “許可を付与する”](#)」のトピックの前提条件に記載されているバージョンと同じかそれ以降のプラットフォームバージョンの既存のクラスター。
- デバイスまたは AWS CloudShell にインストールされている eksctl コマンドラインツールのバージョン 0.210.0 以降。eksctl をインストールまたはアップグレードするにはeksctl ドキュメントの「[インストール](#)」を参照してください。
- kube-system 名前空間内の aws-auth ConfigMap を変更する Kubernetes アクセス許可。
- CreateAccessEntry および ListAccessEntries のアクセス許可を持つ AWS Identity and Access Management ロールまたはユーザー。詳細については、サービス認可リファレンスの「[Amazon Elastic Kubernetes Service で定義されるアクション](#)」を参照してください。

eksctl

1. aws-auth ConfigMap 内の既存のエントリを表示します。##### の部分は自分のクラスター名に置き換えます。

```
eksctl get iamidentitymapping --cluster my-cluster
```

出力例は次のとおりです。

ARN	USERNAME	GROUPS
arn:aws:iam::111122223333:role/EKS-my-cluster-Admins	Admins	system:masters
arn:aws:iam::111122223333:role/EKS-my-cluster-my-namespace-Viewers	my-namespace-Viewers	Viewers
arn:aws:iam::111122223333:role/EKS-my-cluster-self-managed-ng-1	system:node:{{EC2PrivateDNSName}}	system:bootstrappers,system:nodes
arn:aws:iam::111122223333:user/my-user	my-user	
arn:aws:iam::111122223333:role/EKS-my-cluster-fargateprofile1	system:node:{{SessionName}}	system:bootstrappers,system:nodes,system:node-proxier
arn:aws:iam::111122223333:role/EKS-my-cluster-managed-ng	system:node:{{EC2PrivateDNSName}}	system:bootstrappers,system:nodes

2. 前の出力で返された作成済み ConfigMap エントリのいずれかへの [the section called “アクセスエントリを作成する”](#)。アクセスエントリを作成するときは、出力で返されるARN、USERNAME、GROUPS および ACCOUNT と同じ値を指定してください。出力例では、最後の2つのエントリを除くすべてのエントリのアクセスエントリを作成します。これらのエントリは Fargate プロファイルとマネージド型ノードグループ用に Amazon EKS によって作成されたものだからです。
3. 作成したすべてのアクセスエントリを ConfigMap から削除します。ConfigMap からエントリを削除しない場合、IAM プリンシパル ARN のアクセスエントリの設定によって ConfigMap エントリが上書きされます。**111122223333** は AWS アカウント ID に、**EKS-my-cluster-my-namespace-Viewers** は、ConfigMap のエントリに含まれるロールの名前に置き換えます。削除するエントリが IAM ロールではなく IAM ユーザー用のものである場合は、role を user に、**EKS-my-cluster-my-namespace-Viewers** をユーザー名に置き換えます。

```
eksctl delete iamidentitymapping --arn arn:aws:iam::111122223333:role/EKS-my-cluster-my-namespace-Viewers --cluster my-cluster
```

アクセスポリシーアクセス許可を確認する

アクセスポリシーには、Kubernetes verbs (アクセス許可) と resources を含む rules が含まれます。アクセスポリシーには IAM のアクセス許可やリソースは含まれません。Kubernetes Role および ClusterRole オブジェクトと同様に、アクセスポリシーには allow rules のみが含まれます。アクセスポリシーの内容は変更できません。独自のアクセスポリシーを作成することはできません。アクセスポリシーのアクセス許可がニーズに合わない場合は、Kubernetes RBAC オブジェクトを作成し、アクセスエントリのグループ名を指定します。詳細については、「[the section called “アクセスエントリを作成する”](#)」を参照してください。アクセスポリシーに含まれるアクセス許可は、Kubernetes ユーザー向けのクラスターロールのアクセス許可と似ています。詳細については、Kubernetes ドキュメントの「[ユーザー向け Role](#)」を参照してください。

アクセスポリシーを選択し、その内容を確認します。各アクセスポリシーの各テーブルの各行は個別のルールです。

AmazonEKSAAdminPolicy

このアクセスポリシーには、リソースに対するほとんどのアクセス許可を IAM プリンシパルに付与するアクセス許可が含まれています。アクセスエントリに関連付けられている場合、そのアクセス範囲は通常 1 つ以上の Kubernetes 名前空間です。IAM プリンシパルにクラスター上のすべてのリソースへの管理者アクセスを許可する場合は、代わりに [the section called “AmazonEKSClusterAdminPolicy”](#) アクセスポリシーをアクセスエントリに関連付けてください。

ARN — `arn:aws:eks::aws:cluster-access-policy/AmazonEKSAAdminPolicy`

Kubernetes API グループ	Kubernetes resources	Kubernetes 動詞 (許可)
apps	daemonsets , deployments , deployments/rollback , deployments/scale , replicaset , replicaset/scale , statefulsets , statefulsets/scale	create, delete, deletecollection , patch, update

Kubernetes API グループ	Kubernetes resources	Kubernetes 動詞 (許可)
apps	controllerrevisions , daemonsets , daemonsets/status , deployments , deployments/scale , deployments/status , replicaset , replicaset/scale , replicaset/status , statefulsets , statefulsets/scale , statefulsets/status	get, list, watch
authorization.k8s.io	localsubjectaccess reviews	create
autoscaling	horizontalpodautoscalers	create, delete, deletecollection , patch, update
autoscaling	horizontalpodautoscalers , horizontalpodautoscalers/status	get, list, watch
batch	cronjobs, jobs	create, delete, deletecollection , patch, update
batch	cronjobs, cronjobs/status , jobs, jobs/status	get, list, watch
discovery.k8s.io	endpointslices	get, list, watch

Kubernetes API グループ	Kubernetes resources	Kubernetes 動詞 (許可)
extensions	daemonsets , deployments , deployments/rollback , deployments/scale , ingresses , networkpolicies , replicaset , replicaset/scale , replicationcontrollers/scale	create, delete, deletecollection , patch, update
extensions	daemonsets , daemonsets/status , deployments , deployments/scale , deployments/status , ingresses , ingresses/status , networkpolicies , replicaset , replicaset/scale , replicaset/status , replicationcontrollers/scale	get, list, watch
networking.k8s.io	ingresses , ingresses/status , networkpolicies	get, list, watch
networking.k8s.io	ingresses , networkpolicies	create, delete, deletecollection , patch, update
policy	poddisruptionbudgets	create, delete, deletecollection , patch, update
policy	poddisruptionbudgets , poddisruptionbudgets/status	get, list, watch

Kubernetes API グループ	Kubernetes resources	Kubernetes 動詞 (許可)
rbac.authorization.k8s.io	rolebindings , roles	create, delete, deletecollection , get, list, patch, update, watch
	configmaps , endpoints , persistentvolumeclaims , persistentvolumeclaims/status , pods, replicationcontrollers , replicationcontrollers/scale , serviceaccounts , services, services/status	get, list, watch
	pods/attach , pods/exec , pods/portforward , pods/proxy , secrets, services/proxy	get, list, watch
	configmaps , events, persistentvolumeclaims , replicationcontrollers , replicationcontrollers/scale , secrets, serviceaccounts , services, services/proxy	create, delete, deletecollection , patch, update
	pods, pods/attach , pods/exec , pods/portforward , pods/proxy	create, delete, deletecollection , patch, update
	serviceaccounts	impersonate

Kubernetes API グループ	Kubernetes resources	Kubernetes 動詞 (許可)
	bindings, events, limitranges , namespaces/status , pods/log, pods/status , replicationcontrollers/status , resourcequotas , resourcequotas/status	get, list, watch
	namespaces	get,list, watch

AmazonEKSClusterAdminPolicy

このアクセスポリシーには、IAM プリンシパル管理者にクラスターへのアクセス権を付与するアクセス許可が含まれています。アクセスエントリに関連付けられている場合、そのアクセス範囲は通常、Kubernetes 名前空間ではなくクラスターになります。IAM プリンシパルの管理範囲をより限定したい場合は、代わりに [the section called “AmazonEKSAAdminPolicy”](#) アクセスポリシーをアクセスエントリに関連付けることを検討してください。

ARN — `arn:aws:eks::aws:cluster-access-policy/AmazonEKSClusterAdminPolicy`

Kubernetes API グループ	Kubernetes nonResourceURL	Kubernetes resources	Kubernetes 動詞 (許可)
*		*	*
	*		*

AmazonEKSAAdminViewPolicy

このアクセスポリシーには、IAM プリンシパルにクラスターの全リソースをリスト/表示するアクセス権を付与するアクセス許可が含まれています。これには [Kubernetes シークレット](#) が含まれることに注意してください。

ARN — `arn:aws:eks::aws:cluster-access-policy/AmazonEKSAAdminViewPolicy`

Kubernetes API グループ	Kubernetes resources	Kubernetes 動詞 (許可)
*	*	get, list, watch

AmazonEKSEditPolicy

このアクセスポリシーには、IAM プリンシパルがほとんどの Kubernetes リソースを編集できるようにするアクセス許可が含まれています。

ARN — `arn:aws:eks::aws:cluster-access-policy/AmazonEKSEditPolicy`

Kubernetes API グループ	Kubernetes resources	Kubernetes 動詞 (許可)
apps	daemonsets , deployments , deployments/rollback , deployments/scale , replicase ts , replicaset s/scale, statefulsets , statefulsets/scale	create, delete, deletecol lection , patch, update
apps	controllerrevisions , daemonsets , daemonset s/status , deployments , deployments/scale , deployments/status , replicaset s , replicase ts/scale , replicase ts/status , statefuls ets , statefulsets/ scale , statefulsets/ status	get, list, watch
autoscaling	horizontalpodautos calers , horizonta lpodautoscalers/st atus	get, list, watch

Kubernetes API グループ	Kubernetes resources	Kubernetes 動詞 (許可)
autoscaling	horizontalpodautoscalers	create, delete, deletecollection , patch, update
batch	cronjobs, jobs	create, delete, deletecollection , patch, update
batch	cronjobs, cronjobs/status , jobs, jobs/status	get, list, watch
discovery.k8s.io	endpointslices	get, list, watch
extensions	daemonsets , deployments , deployments/rollback , deployments/scale , ingresses , networkpolicies , replicaset , replicaset/scale , replicationcontrollers/scale	create, delete, deletecollection , patch, update
extensions	daemonsets , daemonsets/status , deployments , deployments/scale , deployments/status , ingresses , ingresses/status , networkpolicies , replicaset , replicaset/scale , replicaset/status , replicationcontrollers/scale	get, list, watch
networking.k8s.io	ingresses , networkpolicies	create, delete, deletecollection , patch, update

Kubernetes API グループ	Kubernetes resources	Kubernetes 動詞 (許可)
networking.k8s.io	ingresses , ingresses /status , networkpolicies	get, list, watch
policy	poddisruptionbudgets	create, delete, deletecollection , patch, update
policy	poddisruptionbudgets , poddisruptionbudgets/status	get, list, watch
	namespaces	get, list, watch
	pods/attach , pods/exec , pods/portforward , pods/proxy , secrets, services/proxy	get, list, watch
	serviceaccounts	impersonate
	pods, pods/attach , pods/exec , pods/portforward , pods/proxy	create, delete, deletecollection , patch, update
	configmaps , events, persistentvolumeclaims , replicationcontrollers , replicationcontrollers/scale , secrets, serviceaccounts , services, services/proxy	create, delete, deletecollection , patch, update

Kubernetes API グループ	Kubernetes resources	Kubernetes 動詞 (許可)
	configmaps , endpoints , persistentvolumeclaims , persistentvolumeclaims/status , pods, replicationcontrollers , replicationcontrollers/scale , serviceaccounts , services, services/status	get, list, watch
	bindings, events, limitranges , namespaces/status , pods/log, pods/status , replicationcontrollers/status , resourcequotas , resourcequotas/status	get, list, watch

AmazonEKSVIEWPolicy

このアクセスポリシーには、IAM プリンシパルがほとんどの Kubernetes リソースを表示できるようにするアクセス許可が含まれています。

ARN — `arn:aws:eks::aws:cluster-access-policy/AmazonEKSVIEWPolicy`

Kubernetes API グループ	Kubernetes resources	Kubernetes 動詞 (許可)
apps	controllerrevisions , daemonsets , daemonsets/status , deployments , deployments/scale , deployments/status , replicaset , replicase	get, list, watch

Kubernetes API グループ	Kubernetes resources	Kubernetes 動詞 (許可)
	ts/scale , replicase ts/status , statefuls ets , statefulsets/ scale , statefulsets/ status	
autoscaling	horizontalpodautos calers , horizonta lpodautoscalers/st atus	get, list, watch
batch	cronjobs, cronjobs/ status , jobs, jobs/stat us	get, list, watch
discovery.k8s.io	endpointslices	get, list, watch
extensions	daemonsets , daemonset s/status , deploymen ts , deployments/scale , deployments/status , ingresses , ingresses /status , networkpo licies , replicaset s , replicaset s/scale , replicaset s/status , replicationcontrol lers/scale	get, list, watch
networking.k8s.io	ingresses , ingresses /status , networkpo licies	get, list, watch
policy	poddisruptionbudgets , poddisruptionbudge ts/status	get, list, watch

Kubernetes API グループ	Kubernetes resources	Kubernetes 動詞 (許可)
	configmaps , endpoints , persistentvolumeclaims , persistentvolumeclaims/status , pods, replicationcontrollers , replicationcontrollers/scale , serviceaccounts , services, services/status	get, list, watch
	bindings、 events、 limitranges 、 namespaces/ status、 pods/log、 pods/ status 、 replicationcontrollers/ status 、 resourcequotas 、 resourcequotas/status	get, list, watch
	namespaces	get, list, watch

AmazonEKSAutoNodePolicy

ARN — `arn:aws:eks::aws:cluster-access-policy/AmazonEKSAutoNodePolicy`

このポリシーには、Amazon EKS コンポーネントが以下のタスクを完了できるようにする次のアクセス許可が含まれています。

- kube-proxy – ネットワークエンドポイントおよびサービスをモニタリングし、関連するイベントを管理します。これにより、クラスター全体のネットワークプロキシ機能が有効になります。
- ipamd – AWS VPC ネットワーキングリソースと Container Network Interface (CNI) を管理します。これにより、IP アドレス管理デーモンがポッドネットワーキングを処理できるようになります。

- `coredns` – エンドポイントやサービスなどのサービス検出リソースにアクセスします。これにより、クラスター内の DNS 解決が有効になります。
- `ebs-csi-driver` – Amazon EBS ボリュームのストレージ関連リソースを操作します。これにより、永続ボリュームの動的なプロビジョニングとアタッチが可能になります。
- `neuron` – AWS Neuron デバイスのノードとポッドをモニタリングします。これにより、AWS Inferentia および Trainium アクセラレーターを管理できます。
- `node-monitoring-agent` – ノード診断およびイベントにアクセスします。これにより、クラスターのヘルスマニタリングと診断の収集が可能になります。

各コンポーネントは専用のサービスアカウントを使用し、その特定の機能に必要なアクセス許可のみに制限されます。

NodeClass でノード IAM ロールを手動で指定する場合は、新しいノード IAM ロールをこのアクセスポリシーに関連付けるアクセスエントリを作成する必要があります。

AmazonEKSBLOCKStoragePolicy

Note

このポリシーは AWS サービスリンクロール専用で、カスターマネージドロールには使用できません。

ARN — `arn:aws:eks::aws:cluster-access-policy/AmazonEKSBLOCKStoragePolicy`

このポリシーには、Amazon EKS がストレージオペレーションのリーダー選出および調整リソースを管理できるようにするアクセス許可が含まれています。

- `coordination.k8s.io` – リーダー選出用のリースオブジェクトを作成および管理します。これにより、EKS ストレージコンポーネントは、リーダー選出メカニズムを通じてクラスター全体のアクティビティを調整できます。

ポリシーは、クラスター内の他の調整リソースへのアクセス競合を防ぐために、EKS ストレージコンポーネントで使用される特定のリースリソースを対象とします。

Amazon EKS は、Auto Mode が有効の場合、クラスター IAM ロールのこのアクセスポリシーを使用してアクセスエントリを自動的に作成し、ブロックストレージ機能が正しく機能するのに必要なアクセス許可を設定します。

AmazonEKSLoadBalancingPolicy

ARN — `arn:aws:eks::aws:cluster-access-policy/AmazonEKSLoadBalancingPolicy`

このポリシーには、Amazon EKS がロードバランシングのリーダー選出リソースを管理できるようにするアクセス許可が含まれています。

- `coordination.k8s.io` – リーダー選出用のリースオブジェクトを作成および管理します。これにより、EKS ロードバランシングコンポーネントはリーダーを選出することで、複数のレプリカ間でアクティビティを調整できます。

このポリシーは、クラスター内の他のリースリソースへのアクセスを防止しながら、適切な調整を実現するために、特にロードバランシングリースリソースを対象とします。

Amazon EKS は、Auto Mode が有効の場合、クラスター IAM ロールのこのアクセスポリシーを使用してアクセスエントリを自動的に作成し、ネットワーク機能が正しく機能するのに必要なアクセス許可を設定します。

AmazonEKSNetworkingPolicy

ARN — `arn:aws:eks::aws:cluster-access-policy/AmazonEKSNetworkingPolicy`

このポリシーには、Amazon EKS がネットワーキングのリーダー選出リソースを管理できるようにするアクセス許可が含まれています。

- `coordination.k8s.io` – リーダー選出用のリースオブジェクトを作成および管理します。これにより、EKS ネットワークコンポーネントはリーダーを選出することで、IP アドレス割り当てアクティビティを調整できます。

このポリシーは、クラスター内の他のリースリソースへのアクセスを防止しながら、適切な調整を実現するために、特にネットワークリースリソースを対象とします。

Amazon EKS は、Auto Mode が有効の場合、クラスター IAM ロールのこのアクセスポリシーを使用してアクセスエントリを自動的に作成し、ネットワーク機能が正しく機能するのに必要なアクセス許可を設定します。

AmazonEKSComputePolicy

Note

このポリシーは AWS サービスリンクロール専用で、カスターマネージドロールには使用できません。

ARN — `arn:aws:eks::aws:cluster-access-policy/AmazonEKSComputePolicy`

このポリシーには、Amazon EKS がコンピューティングオペレーションのリーダー選出リソースを管理できるようにするアクセス許可が含まれています。

- `coordination.k8s.io` – リーダー選出用のリースオブジェクトを作成および管理します。これにより、EKS コンピューティングコンポーネントは、リーダーを選出することでノードスケリングアクティビティを調整できます。

このポリシーは、クラスター内のすべてのリースリソースへの基本的な読み取りアクセス (`get`、`watch`) を許可しながら、特にコンピューティング管理リースリソースを対象とします。

Amazon EKS は、Auto Mode が有効の場合、クラスター IAM ロールのこのアクセスポリシーを使用してアクセスエントリを自動的に作成し、ネットワーク機能が正しく機能するのに必要なアクセス許可を設定します。

AmazonEKSBlockStorageClusterPolicy

Note

このポリシーは AWS サービスリンクロール専用で、カスターマネージドロールには使用できません。

ARN — `arn:aws:eks::aws:cluster-access-policy/AmazonEKSBlockStorageClusterPolicy`

このポリシーは、Amazon EKS Auto Mode のブロックストレージ機能に必要なアクセス許可を付与します。これにより、Amazon EKS クラスター内のブロックストレージリソースを効率的に管理できます。ポリシーには以下のアクセス権限が含まれています。

CSI ドライバー管理:

- CSI ドライバー (特にブロックストレージ用) の作成、読み取り、更新、削除を行います。

ボリューム管理:

- 永続ボリュームの一覧表示、監視、作成、更新、パッチ適用、削除を行います。
- 永続ボリュームのクレームを一覧表示、監視、更新します。
- パッチ永続ボリュームのクレームのステータス。

ノードとポッドのインタラクション:

- ノードおよびポッドの情報を読み取ります。
- ストレージオペレーションに関連するイベントを管理します。

ストレージクラスおよび属性:

- ストレージクラスと CSI ノードを読み取ります。
- ボリューム属性クラスを読み取ります。

ボリュームのアタッチ:

- ボリュームのアタッチとそのステータスを一覧表示、監視、変更します。

スナップショットオペレーション:

- ボリュームのスナップショット、スナップショットコンテンツ、スナップショットクラスを管理します。
- ボリュームグループスナップショットおよび関連リソースのオペレーションを処理します。

このポリシーは、Auto Mode で実行されている Amazon EKS クラスター内の包括的なブロックストレージ管理をサポートするように設計されています。ブロックストレージボリュームのプロビジョニング、アタッチ、サイズ変更、スナップショット作成など、さまざまなオペレーションに対するアクセス許可を組み合わせます。

Amazon EKS は、Auto Mode が有効の場合、クラスター IAM ロールのこのアクセスポリシーを使用してアクセスエントリを自動的に作成し、ブロックストレージ機能が正しく機能するのに必要なアクセス許可を設定します。

AmazonEKSClusterPolicy

Note

このポリシーは AWS サービスリンクロール専用で、カスターマネージドロールには使用できません。

ARN — `arn:aws:eks::aws:cluster-access-policy/AmazonEKSClusterPolicy`

このポリシーは、Amazon EKS Auto Mode のコンピューティング管理機能に必要なアクセス許可を付与します。これにより、Amazon EKS クラスター内のコンピューティングリソースを効率的にオーケストレーションおよびスケールできます。ポリシーには以下のアクセス権限が含まれています。

ノード管理:

- NodePools および NodeClaims のステータスの作成、読み取り、更新、削除、管理を行います。
- 作成、変更、削除を含め、NodeClasses を管理します。

スケジューリングとリソース管理:

- ポッド、ノード、永続ボリューム、永続ボリュームのクレーム、レプリケーションコントローラー、名前空間への読み取りアクセス。
- ストレージクラス、CSI ノード、ボリュームのアタッチへの読み取りアクセス。
- デプロイ、デーモンセット、レプリカセット、ステートフルセットを一覧表示および監視します。
- ポッド中断の予算を読み取ります。

イベント処理:

- クラスターイベントの作成、読み取り、管理を行います。

ノードのプロビジョニング解除とポッドエビクション:

- ノードの更新、パッチ適用、削除を行います。
- 必要に応じて、ポッドエビクションを作成してポッドを削除します。

カスタムリソース定義 (CRD) 管理:

- 新しい CRD を作成します。
- ノード管理に関連する特定の CRD (NodeClasses、NodePools、NodeClaims、NodeDiagnostics) を管理します。

このポリシーは、Auto Mode で実行されている Amazon EKS クラスター内の包括的なコンピューティング管理をサポートするように設計されています。ノードのプロビジョニング、スケジューリング、スケーリング、リソースの最適化など、さまざまなオペレーションのアクセス許可を組み合わせます。

Amazon EKS は、Auto Mode が有効の場合、クラスター IAM ロールのこのアクセスポリシーを使用してアクセスエントリを自動的に作成し、コンピューティング管理機能が正しく機能するのに必要なアクセス許可を設定します。

AmazonEKSLoadBalancingClusterPolicy

ARN — `arn:aws:eks::aws:cluster-access-policy/AmazonEKSLoadBalancingClusterPolicy`

このポリシーは、Amazon EKS Auto Mode のロードバランシング機能に必要なアクセス許可を付与します。これにより、Amazon EKS クラスター内のロードバランシングリソースを効率的に管理および設定できます。ポリシーには以下のアクセス権限が含まれています。

イベントとリソースの管理:

- イベントを作成してパッチを適用します。
- ポッド、ノード、エンドポイント、名前空間への読み取りアクセス。
- ポッドのステータスを更新します。

サービスおよびイングレス管理:

- サービスとそのステータスの完全な管理。
- イングレスとそのステータスの包括的な制御。
- エンドポイントスライスとイングレスクラスへの読み取りアクセス。

ターゲットグループのバインディング:

- ターゲットグループのバインディングとそのステータスを作成および変更します。
- イングレスクラスパラメータへの読み取りアクセス。

カスタムリソース定義 (CRD) 管理:

- すべての CRD の作成および読み取りを行います。
- targetgroupbindings.eks.amazonaws.com および ingressclassparams.eks.amazonaws.com CRD の特定の管理。

ウェブフック設定:

- 変異および検証ウェブフック設定の作成および読み取りを行います。
- eks-load-balancing-webhook 設定を管理します。

このポリシーは、Auto Mode で実行されている Amazon EKS クラスター内の包括的なロードバランシング管理をサポートするように設計されています。サービスの公開、イングレスルーティング、AWS ロードバランシングサービスとの統合など、さまざまなオペレーションのアクセス許可を組み合わせます。

Amazon EKS は、Auto Mode が有効の場合、クラスター IAM ロールのこのアクセスポリシーを使用してアクセスエントリを自動的に作成し、ロードバランシング機能が正しく機能するのに必要なアクセス許可を設定します。

AmazonEKSNetworkingClusterPolicy

ARN — `arn:aws:eks::aws:cluster-access-policy/`

AmazonEKSNetworkingClusterPolicy

AmazonEKSNetworkingClusterPolicy

このポリシーは、Amazon EKS Auto Mode のネットワーキング機能に必要なアクセス許可を付与します。これにより、Amazon EKS クラスター内のネットワーキングリソースを効率的に管理および設定できます。ポリシーには以下のアクセス権限が含まれています。

ノードとポッドの管理:

- NodeClasses とそのステータスへの読み取りアクセス。
- NodeClaims とそのステータスへの読み取りアクセス。

- ポッドへの読み取りアクセス。

CNI ノード管理:

- 作成、読み取り、更新、削除、パッチ適用などの、CNINodes とそのステータスのアクセス許可。

カスタムリソース定義 (CRD) 管理:

- すべての CRD の作成および読み取りを行います。
- `cninodes.eks.amazonaws.com` CRD の特定の管理 (更新、パッチ適用、削除)。

イベント管理:

- イベントを作成してパッチを適用します。

このポリシーは、Auto Mode で実行されている Amazon EKS クラスター内の包括的なネットワーク管理をサポートするように設計されています。ノードネットワーキング設定、CNI (Container Network Interface) 管理、関連するカスタムリソース処理など、さまざまなオペレーションのアクセス許可を組み合わせます。

このポリシーにより、ネットワーキングコンポーネントはノード関連のリソースを操作し、CNI 固有のノード設定を管理し、クラスター内のネットワーキングオペレーションに不可欠なカスタムリソースを処理できます。

Amazon EKS は、Auto Mode が有効の場合、クラスター IAM ロールのこのアクセスポリシーを使用してアクセスエントリを自動的に作成し、ネットワーク機能が正しく機能するのに必要なアクセス許可を設定します。

AmazonEKSHybridPolicy

Note

このポリシーは AWS サービスリンクロール専用で、カスタマーマネージドロールには使用できません。

このアクセスポリシーには、クラスターのノードへの EKS アクセスを許可するアクセス許可が含まれています。アクセスエントリに関連付けられている場合、そのアクセス範囲は通常、Kubernetes

名前空間ではなくクラスターになります。このポリシーは、Amazon EKS Hybrid Nodes で使用されます。

ARN — `arn:aws:eks::aws:cluster-access-policy/AmazonEKSHybridPolicy`

Kubernetes API グループ	Kubernetes nonResourceURL	Kubernetes resources	Kubernetes 動詞 (許可)
*		nodes	list

AmazonEKSClusterInsightsPolicy

 Note

このポリシーは AWS サービスリンクロール専用で、カスターマネージドロールには使用できません。

ARN — `arn:aws:eks::aws:cluster-access-policy/AmazonEKSClusterInsightsPolicy`

このポリシーは、Amazon EKS クラスターインサイト機能の読み取り専用アクセス許可を付与します。ポリシーには以下のアクセス権限が含まれています。

ノードアクセス: - クラスターノードの一覧表示および表示 - ノードのステータス情報の読み取り

DaemonSet アクセス: - kube-proxy 設定に対する読み取りアクセス

このポリシーは、クラスターインサイトの EKS サービスによって自動的に管理されます。詳細については、「[the section called “クラスターのインサイト”](#)」を参照してください。

アクセスポリシーの更新

アクセスポリシーが導入されてからのアクセスポリシーへの更新に関する詳細を表示します。このページの変更に関する自動通知については、[ドキュメント履歴](#) の RSS フィードを購読してください。

変更	説明	日付
EKS クラスターインサイトのポリシーを追加する	AmazonEKSClusterInsightsPolicy を発行	2024 年 12 月 2 日
Amazon EKS Hybrid のポリシーを追加	AmazonEKSHybridPolicy を発行	2024 年 12 月 2 日
Amazon EKS Auto Mode のポリシーを追加	このアクセスポリシーは、クラスター IAM ロールとノード IAM ロールに Kubernetes API を呼び出すアクセス許可を付与します。AWS ではこれらを使用して、ストレージ、コンピューティング、およびネットワークリソースのルーチンタスクを自動化します。	2024 年 12 月 2 日
AmazonEKSAAdminView Policy を追加します。	Secrets などのリソースを含む、拡張ビューアクセス用の新しいポリシーを追加します。	2024 年 4 月 23 日
アクセスポリシーが導入されました。	Amazon EKS はアクセスポリシーを導入しました。	2023 年 5 月 29 日

アクセスエントリを使用するように認証モードを変更する

アクセスエントリを使い始めるにはクラスターの認証モードを API_AND_CONFIG_MAP または API モードに変更する必要があります。これにより、アクセスエントリ用の API が追加されます。

AWS コンソール

1. [Amazon EKS コンソール](#)を開きます。
2. アクセスエントリを作成するクラスターの名前を選択してください。
3. [リモートアクセス] タブを選択してください。

4. [認証モード] にはクラスターの現在の認証モードが表示されます。モードが EKS API と表示されている場合は既にアクセスエントリを追加でき、残りの手順は省略できます。
5. [アクセスの管理] を選択してください。
6. クラスター認証モードの場合は、EKS API でモードを選択してください。認証モードを、EKS API およびアクセスエントリを削除するモードに戻すことはできないことに注意してください。
7. [Save changes] (変更の保存) をクリックします。Amazon EKS がクラスターの更新を開始し、クラスターのステータスが「更新中」に変わり、変更が [更新履歴] タブに記録されます。
8. クラスターのステータスが [アクティブ] に戻るまで待ちます。クラスターが [アクティブ] になったら、「[the section called “アクセスエントリを作成する”](#)」の手順に従って IAM プリンシパルのクラスターへのアクセスを追加できます。

AWS CLI

1. 「AWS コマンドラインインターフェイスユーザーガイド」の「[インストール](#)」の説明に従って、AWS CLI をインストールします。
2. 以下のコマンドを実行してください。##### の部分は自分のクラスター名に置き換えます。ConfigMap メソッドを完全に無効にする場合はAPI_AND_CONFIG_MAP を API に置き換えてください。

Amazon EKS がクラスターの更新を開始し、クラスターのステータスが [UPDATING] に変わり、変更が `aws eks list-updates` に記録されます。

```
aws eks update-cluster-config --name my-cluster --access-config
authenticationMode=API_AND_CONFIG_MAP
```

3. クラスターのステータスが [アクティブ] に戻るまで待ちます。クラスターが [アクティブ] になったら、「[the section called “アクセスエントリを作成する”](#)」の手順に従って IAM プリンシパルのクラスターへのアクセスを追加できます。

必要なプラットフォームバージョン

アクセスエントリを使用するには、クラスターのプラットフォームバージョンが、次の表に記載されているバージョンと同じかそれ以降、または表に記載されている Kubernetes バージョンよりも新しいバージョンである必要があります。Kubernetes バージョンがリストにない場合は、すべてのプラットフォームバージョンでアクセスエントリがサポートされます。

Kubernetes バージョン	プラットフォームバージョン
1.30	eks.2
1.29	eks.1
1.28	eks.6
1.27	eks.10
1.26	eks.11

詳細については「[the section called “プラットフォームのバージョン”](#)」を参照してください。

アクセスエントリを作成する

アクセスエントリを作成する前に、次の点を考慮してください：

- 適切に設定された認証モード。「[the section called “\[Authentication mode\] \(認証モード\)”](#)」を参照してください。
- アクセスエントリには1 つだけの既存の IAM プリンシパルの Amazon リソースネーム (ARN) が含まれます。IAM プリンシパルは複数のアクセスエントリに含めることはできません。指定する ARN に関するその他の考慮事項
 - IAM のベストプラクティスでは長期認証情報を持つ IAM ユーザーではなく、短期認証情報を持つ IAM ロールを使用してクラスターにアクセスすることを推奨しています。詳細は「IAM ユーザーガイド」の「[人間のユーザーが一時的な認証情報を使用して AWS にアクセスするにはID プロバイダーとのフェデレーションの使用が必要です](#)」を参照してください。
 - ARN が IAM ロール用の場合はパスを含めることができます。aws-auth ConfigMap エントリの ARN にはパスを含めることはできません。例えば、ARN は `arn:aws:iam::<111122223333>:role/<development/apps/my-role>` または `arn:aws:iam::<111122223333>:role/<my-role>` にすることができます。
 - アクセスエントリのタイプが STANDARD 以外の場合 (タイプに関する次の考慮事項を参照)、ARN はクラスターと同じ AWS アカウントにある必要があります。タイプが STANDARD の場合、ARN はクラスターが属するアカウントと同じ、または異なる AWS アカウントでもかまいません。
- アクセスエントリの作成後に IAM プリンシパルを変更することはできません。

- この ARN を含む IAM プリンシパルを削除しても、アクセスエントリは自動的に削除されません。削除する IAM プリンシパルの ARN を含むアクセスエントリを削除することをお勧めします。アクセスエントリを削除せず、IAM プリンシパルを再作成すると、同じ ARN があっても、アクセスエントリは機能しません。これは再作成された IAM プリンシパルの ARN は同じですが、再作成された IAM プリンシパルの `roleID` または `userID` (`aws sts get-caller-identity` AWS CLI コマンドで確認できます) は元の IAM プリンシパルの ARN とは異なるためです。アクセスエントリの IAM プリンシパルの `roleID` または `userID` が表示されない場合でも、Amazon EKS はそれをアクセスエントリとともに保存します。
- 各アクセスエントリにはタイプがあります。タイプを指定しない場合、Amazon EKS は自動的にタイプを STANDARD に設定します。
 - EC2_LINUX - Linux または Bottlerocket セルフマネージド型ノードで使用される IAM ロール用
 - EC2_WINDOWS - Windows セルフマネージドノードで使用される IAM ロール用
 - FARGATE_LINUX - AWS Fargate (Fargate) で使用される IAM ロール用
 - HYBRID_LINUX - ハイブリッドノードで使用される IAM ロール用
 - STANDARD - 指定しない場合のデフォルトタイプ
 - EC2 - EKS Auto Mode カスタムノードクラス用。詳細については、[「the section called “ノードクラスのアクセスエントリを作成する”」](#) を参照してください。
 - アクセスエントリの作成後にタイプを変更することはできません。
- マネージドノードグループまたは Fargate プロファイルに使用される IAM ロールのアクセスエントリを作成する必要はありません。EKS はアクセス エントリを作成するか (有効な場合)、認証構成マップを更新します (アクセス エントリが利用できない場合)。
- アクセスエントリのタイプが STANDARD の場合、アクセスエントリのユーザー名を指定できます。ユーザー名の値を指定しない場合、Amazon EKS はアクセスエントリのタイプと、指定した IAM プリンシパルが IAM ロールか IAM ユーザーかに応じて、以下のいずれかの値を設定します。独自のユーザー名を指定する特別な理由がない限り、ユーザー名を指定せず、Amazon EKS に自動生成させることをお勧めします。独自のユーザー名を指定する場合:
 - `system:`、`eks:`、`aws:`、`amazon:`、または `iam:` で始めることはできません。
 - ユーザー名が IAM ロール用の場合は、ユーザー名の末尾に `{{SessionName}}` または `{{SessionNameRaw}}` を追加することをお勧めします。ユーザー名に `{{SessionName}}` または `{{SessionNameRaw}}` を追加する場合、ユーザー名の `{{SessionName}}` の前にコロンを含める必要があります。このルールを引き受けると、ルールを引き受けるときに指定された AWS STS セッションの名前が自動的にクラスターに渡され、CloudTrail ログに表示されます。例えば、`john{{SessionName}}` というユーザー名にはできません。ユーザー名は `:john{{SessionName}}` または `jo:hn{{SessionName}}` でなければなりません。コ

コロンは `{{SessionName}}` の前になければいけません。次の表の Amazon EKS によって生成されたユーザー名には ARN が含まれています。ARN にはコロンが含まれているため、この要件を満たしています。ユーザー名に `{{SessionName}}` を含めなければ、コロンは不要です。`{{SessionName}}` の場合、セッション名の特殊文字「@」は「-」に置き換えられます。`{{SessionNameRaw}}` の場合、セッション名の特殊文字はすべて保持されます。

IAM プリンシパルタイプ	タイプ	Amazon EKS が自動的に設定するユーザー名の値
ユーザー	STANDARD	ユーザーの ARN。例: arn:aws:iam::<111122223333>:user/<my-user>
ロール	STANDARD	想定されるロールの STS ARN。Amazon EKS では <code>{{SessionName}}</code> がロールに追加されます。 例: arn:aws:sts::<111122223333>:assumed-role/<my-role>/{{SessionName}} 指定したロールの ARN にパスが含まれている場合、Amazon EKS は生成されたユーザー名からそのパスを削除してください。
ロール	EC2_LINUX 、または EC2_Windows	system:node:{{EC2PrivateDNSName}}
ロール	FARGATE_LINUX	system:node:{{SessionName}}

IAM プリンシパルタイプ	タイプ	Amazon EKS が自動的に設定するユーザー名の値
ロール	HYBRID_LINUX	system:node:{{SessionName}}

アクセスエントリの作成後にユーザー名を変更することができます。

- アクセスエントリのタイプが STANDARD で、Kubernetes RBAC 認可を使用する場合はアクセスエントリに 1 つ以上のグループ名を追加できます。アクセスエントリを作成したら、グループ名を追加および削除できます。IAM プリンシパルがクラスター上の Kubernetes オブジェクトにアクセスできるようにするには Kubernetes ロールベース認可 (RBAC) オブジェクトを作成して管理する必要があります。クラスター上で Kubernetes RoleBinding または ClusterRoleBinding オブジェクトを作成し、kind: Group の subject としてグループ名を指定します。Kubernetes は、Kubernetes Role で指定したクラスターオブジェクト、またはバインディングの roleRef でも指定した ClusterRole オブジェクトへの IAM プリンシパルアクセスを許可します。グループ名を指定する場合は、Kubernetes ロールベース認可 (RBAC) オブジェクトについて理解しておくことをお勧めします。詳細については Kubernetes ドキュメントの「[RBAC 認可を使用する](#)」を参照してください。

Important

Amazon EKS はクラスターに存在する Kubernetes RBAC オブジェクトに、指定したグループ名が含まれていることを確認しません。例えば、現時点で存在しないグループのアクセスエントリを作成すると、EKS はエラーを返す代わりにグループを作成します。

クラスター上の Kubernetes オブジェクトに IAM プリンシパルがアクセスするのを許可する Kubernetes の代わりに、またはそれに加えて、Amazon EKS アクセスポリシーをアクセスエントリに関連付けることができます。Amazon EKS は IAM プリンシパルがアクセスポリシーのアクセス許可を使用してクラスター上の Kubernetes オブジェクトにアクセスすることを許可します。アクセスポリシーのアクセス許可の範囲は指定した Kubernetes 名前空間に限定できます。アクセスポリシーを使用する場合、Kubernetes RBAC オブジェクトを管理する必要はありません。詳細については、「[the section called “アソシエイトアクセスポリシー”](#)」を参照してください。

- タイプが EC2_LINUX または EC2_Windows のアクセスエントリを作成する場合、アクセスエントリを作成する IAM プリンシパルには iam:PassRole アクセス許可が必要です。詳細について

は「IAM ユーザーガイド」の「[AWS サービスにロールを渡す許可をユーザーに付与する](#)」を参照してください。

- 標準的な [IAM 動作](#)と同様に、アクセスエントリの作成と更新は最終的には一貫性があり、最初の API コールが正常に戻ってから有効になるまでに数秒かかる場合があります。発生する可能性のあるこれらの遅延を考慮して、アプリケーションを設計する必要があります。アプリケーションの重要で高可用性のコードパスにはアクセスエントリの作成や更新を含めないことをお勧めします。代わりに、実行頻度が低い別の初期化またはセットアップルーチンに の変更を加えます。また、本番稼働ワークフローが依存する前に、変更が伝達済みであることを確認します。
- アクセスエントリは[サービスリンクロール](#)をサポートしていません。プリンシパル ARN がサービスリンクロールである場合、アクセスエントリを作成することはできません。サービスリンクロールは `arn:aws:iam::*:role/aws-service-role/*` 形式の ARN によって識別できます。

AWS Management Console または AWS CLI を使用してアクセスエントリを作成できます。

AWS Management Console

- [Amazon EKS コンソール](#)を開きます。
- アクセスエントリを作成するクラスターの名前を選択してください。
- [リモートアクセス] タブを選択してください。
- [アクセスエントリの作成] を選択してください。
- IAM プリンシパルには既存の IAM ロールまたはユーザーを選択してください。IAM のベストプラクティスでは長期認証情報を持つ IAM ユーザーではなく、短期認証情報を持つ IAM ロールを使用してクラスターにアクセスすることを推奨しています。詳細は「IAM ユーザーガイド」の「[人間のユーザーが一時的な認証情報を使用して AWS にアクセスするにはID プロバイダーとのフェデレーションの使用が必要です](#)」を参照してください。
- [タイプ] で、アクセスエントリがセルフマネージド型の Amazon EC2 ノードに使用されるノードロール用のものである場合は[EC2 Linux] または [EC2 Windows] を選択してください。それ以外の場合はデフォルト (標準) をそのまま使用します。
- 選択した タイプが標準で、ユーザー名を指定する場合はユーザー名を入力します。
- 選択したタイプが標準で、IAM プリンシパルに Kubernetes RBAC 認可を使用したい場合はグループに 1 つ以上の名前を指定します。グループ名を指定せず、Amazon EKS 認可を使用する場合は後のステップで、またはアクセスエントリの作成後にアクセスポリシーを関連付けることができます。
- (オプション) [タグ] ではアクセスエントリにラベルを割り当てます。例えば、同じタグを持つすべてのリソースを検索しやすくするためです。

10[次へ] を選択します。

11[アクセスポリシーを追加] ページで、選択したタイプが標準で、Amazon EKS が IAM プリンシパルにクラスター上の Kubernetes オブジェクトへのアクセス許可を与えることを許可する場合は次のステップを実行してください。それ以外の場合は[次へ] を選択してください。

- a. [ポリシー名] にはアクセスポリシーを選択してください。アクセスポリシーのアクセス許可は表示できませんが、Kubernetes ユーザー向け ClusterRole オブジェクトと同様のアクセス許可が含まれています。詳細については、Kubernetes ドキュメントの「[ユーザー向け Role](#)」を参照してください。
- b. 以下のオプションのいずれかを選択してください：
 - クラスター — Amazon EKS が IAM プリンシパルに、クラスター上のすべての Kubernetes オブジェクトのアクセスポリシー内のアクセス許可があるように承認する場合はこのオプションを選択してください。
 - Kubernetes 名前空間 — Amazon EKS が IAM プリンシパルに、クラスター上の特定の Kubernetes 名前空間のすべての Kubernetes オブジェクトのアクセスポリシー内のアクセス許可があるように承認する場合はこのオプションを選択してください。[名前空間] にはクラスター上の Kubernetes 名前空間の名前を入力します。名前空間をさらに追加する場合は[新しい名前空間を追加する] を選択し、名前空間名を入力します。
- c. ポリシーを追加するには[ポリシーを追加] を選択してください。ポリシーごとに異なる範囲を設定できますが、各ポリシーは 1 回しか追加できません。
- d. [次へ] を選択してください。

12.アクセスエントリの設定を確認してください。何かが正しくないと思われる場合は[戻る] を選択してステップに戻り、エラーを修正します。設定が正しければ、[作成] を選択してください。

AWS CLI

1. AWS コマンドラインインターフェイスユーザーガイドの「[インストール](#)」の説明に従って、AWS CLI をインストールします。
2. アクセスエントリを作成するには以下のいずれかの例を使用してアクセスエントリを作成できます：
 - セルフマネージド型 Amazon EC2 Linux ノードグループのアクセスエントリを作成します。`#####` はクラスターの名前、`111122223333` は自分の AWS アカウント ID に、`EKS-#####-self-managed-ng-1` は [ノード IAM ロール](#) の名前に置き換えます。ノードグループが Windows ノードグループの場合は `EC2_LINUX` を `EC2_Windows` に置き換えてください。

```
aws eks create-access-entry --cluster-name my-cluster --principal-arn
arn:aws:iam::111122223333:role/EKS-my-cluster-self-managed-ng-1 --type EC2_LINUX
```

STANDARD 以外のタイプを指定する場合、--kubernetes-groups オプションは使用できません。タイプが STANDARD 以外の値であるため、このアクセスエントリにアクセスポリシーを関連付けることはできません。

- Amazon EC2 セルフマネージド型ノードグループには使用されていない IAM ロールを許可するアクセスエントリを作成し、Kubernetes にクラスターへのアクセスを許可してもらいます。**#####** をクラスターの名前に、**111122223333** を AWS アカウント ID に、**my-role** を IAM ロールの名前に置き換えます。**Viewers** はクラスター上の Kubernetes RoleBinding または ClusterRoleBinding オブジェクトで指定したグループの名前に置き換えます。

```
aws eks create-access-entry --cluster-name my-cluster --principal-arn
arn:aws:iam::111122223333:role/my-role --type STANDARD --user Viewers --
kubernetes-groups Viewers
```

- IAM ユーザーがクラスターに対して認証できるようにするアクセスエントリを作成します。これも可能であるためこの例が示されていますが、IAM のベストプラクティスでは長期認証情報を持つ IAM ユーザーではなく、短期認証情報を持つ IAM ロールを使用してクラスターにアクセスすることを推奨しています。詳細は「IAM ユーザーガイド」の「[人間のユーザーが一時的な認証情報を使用して AWS にアクセスするには ID プロバイダーとのフェデレーションの使用が必要です](#)」を参照してください。

```
aws eks create-access-entry --cluster-name my-cluster --principal-arn
arn:aws:iam::111122223333:user/my-user --type STANDARD --username my-user
```

このユーザーに Kubernetes API ディスカバリーロールのアクセス許可よりも多くのクラスターへのアクセスを許可したい場合は --kubernetes-groups オプションが使用されていないため、アクセスエントリにアクセスポリシーを関連付ける必要があります。詳細については、「[the section called “アソシエイトアクセスポリシー”](#)」と Kubernetes ドキュメントの「[API ディスカバリーロール](#)」を参照してください。

アクセスエントリを更新する

AWS Management Console または AWS を使用してアクセスエントリを更新できます。

AWS Management Console

1. [アマゾン EKS コンソール](#)を開きます。
2. アクセスエントリを作成するクラスターの名前を選択してください。
3. [リモートアクセス] タブを選択してください。
4. 更新するアクセスエントリを選択します。
5. [編集] を選択します。
6. [ユーザー名] では、既存の値を変更できます。
7. [グループ]では、既存のグループ名を削除したり、新しいグループ名を追加したりできます。system:nodes または system bootstrappers というグループ名が存在する場合は、削除しないでください。これらのグループを削除すると、クラスターが正しく機能しなくなる可能性があります。グループ名を指定せず、Amazon EKS 認可を使用する場合は、後のステップで、[アクセスポリシー](#)を関連付けます。
8. [タグ] では、アクセスエントリにラベルを割り当てます。例えば、同じタグを持つすべてのリソースを検索しやすくするためです。また、既存のタグを削除することもできます。
9. [Save changes] (変更の保存) をクリックします。
10. アクセスポリシーをエントリに関連付ける場合は、[the section called “アソシエイトアクセスポリシー”](#)を参照してください。

AWS CLI

1. AWS コマンドラインインターフェイスユーザーガイドの「[インストール](#)」の説明に従って、AWS CLI をインストールします。
2. アクセスエントリを更新するには、*my-cluster* はクラスターの名前、*111122223333* は AWS アカウント ID に、*EKS-my-cluster-my-namespace-Viewers* は IAM ロールの名前に置き換えます。

```
aws eks update-access-entry --cluster-name my-cluster --principal-arn
arn:aws:iam::111122223333:role/EKS-my-cluster-my-namespace-Viewers --kubernetes-
groups Viewers
```

アクセスエントリのタイプが STANDARD 以外の値の場合、--kubernetes-groups オプションは使用できません。また、アクセスポリシーを STANDARD 以外のタイプのアクセスエントリに関連付けることもできません。

アクセスエントリを削除する

アクセスエントリを誤って削除したことが判明した場合は、いつでも再作成できます。削除するアクセスエントリがアクセスポリシーに関連付けられている場合、その関連付けは自動的に削除されます。アクセスエントリを削除する前に、アクセスポリシーとアクセスエントリの関連付けを解除する必要はありません。

AWS Management Console または AWS CLI を使用してアクセスエントリを削除できます。

AWS Management Console

1. [アマゾン EKS コンソール](#)を開きます。
2. アクセスエントリを削除するクラスターの名前を選択します。
3. [リモートアクセス] タブを選択してください。
4. [アクセスエントリ] リストで、削除するアクセスエントリを選択します。
5. [Delete] を選択します。
6. 確認ダイアログボックスで、[削除] を選択します。

AWS CLI

1. AWS コマンドラインインターフェイスユーザーガイドの「[インストール](#)」の説明に従って、AWS CLI をインストールします。
2. アクセスエントリを削除するには、*my-cluster* をクラスターの名前、*111122223333* を AWS アカウント ID に、*my-role* をクラスターにアクセスさせたくない IAM ロールの名前に置き換えます。

```
aws eks delete-access-entry --cluster-name my-cluster --principal-arn
arn:aws:iam::111122223333:role/my-role
```

ConfigMap を使用して IAM ユーザーに Kubernetes へのアクセスを許可する

⚠ Important

aws-auth ConfigMap は非推奨になりました。Kubernetes API へのアクセスを管理する推奨手段については、「[the section called “許可を付与する”](#)」を参照してください。

[IAM プリンシパル](#)を使用してクラスターにアクセスします。プリンシパルは、Amazon EKS コントロールプレーンで実行される [AWS IAM Authenticator for Kubernetes](#) によって有効になります。オーセンティケーターは、その設定情報を aws-auth ConfigMap から取得します。すべての aws-auth ConfigMap 設定については、GitHub の [Full Configuration Format](#) を参照してください。

Amazon EKS クラスターに IAM プリンシパルを追加する

Amazon EKS クラスターを作成すると、このクラスターを作成する [IAM プリンシパル](#) は、Amazon EKS コントロールプレーンのクラスターロールベースアクセスコントロール (RBAC) 設定で、system:masters 許可が自動的に付与されます。このプリンシパルは表示可能な設定の中には表示されません。したがって、どのプリンシパルが最初にクラスターを作成したのかを追跡する必要があります。追加の IAM プリンシパルがクラスターとインタラクションできるようにするには、Kubernetes 内の aws-auth ConfigMap を編集し、aws-auth ConfigMap で指定した group の名前を使用して、Kubernetes rolebinding または clusterrolebinding を作成します。

ℹ Note

Kubernetes のロールベースアクセスコントロール (RBAC) 設定の詳細については、Kubernetes ドキュメントの「[RBAC 認可を使用する](#)」を参照してください。

1. クラスターへのアクセスに kubectl が使用している認証情報を判別します。使用中のコンピュータ上で、次のコマンドを使用して、kubectl が使用する認証情報を判別できます。デフォルトのパスを使用しない場合は、`~/.kube/config` を kubeconfig ファイルへのパスに置き換えます。

```
cat ~/.kube/config
```

出力例は次のとおりです。

```
[...]
contexts:
- context:
    cluster: my-cluster.region-code.eksctl.io
    user: admin@my-cluster.region-code.eksctl.io
    name: admin@my-cluster.region-code.eksctl.io
current-context: admin@my-cluster.region-code.eksctl.io
[...]
```

前述の出力例では、*admin* という名前のユーザーの認証情報が *my-cluster* という名前のクラスター用に設定されています。このユーザーがクラスターを作成したユーザーである場合は、すでにそのクラスターにアクセスできます。クラスターを作成したユーザーでない場合は、以降の手順を実行して、他の IAM プリンシパルのクラスターへのアクセスを有効にする必要があります。[IAM のベストプラクティス](#)では、ユーザーではなくロールに許可を付与することが推奨されています。次のコマンドを使用すると、現在クラスターへのアクセスが可能な、他のプリンシパルを確認できます。

```
kubectl describe -n kube-system configmap/aws-auth
```

出力例は次のとおりです。

```
Name:          aws-auth
Namespace:     kube-system
Labels:        <none>
Annotations:   <none>

Data
====
mapRoles:
----
- groups:
  - system:bootstrappers
  - system:nodes
  rolearn: arn:aws:iam::111122223333:role/my-node-role
  username: system:node:{{EC2PrivateDNSName}}

BinaryData
=====
```

```
Events:  <none>
```

前の例は、デフォルトの aws-auth ConfigMap です。ノードインスタンスのロールだけがクラスターにアクセスできます。

2. IAM プリンシパルをマッピングできる既存の Kubernetes roles と rolebindings または clusterroles と clusterrolebindings があることを確認してください。これらのリソースの詳細については、Kubernetes ドキュメントの「[RBAC 認可を使用する](#)」を参照してください。
 - a. 既存の Kubernetes の roles または clusterroles を表示します。Roles は namespace スコープされ、clusterroles はクラスターにスコープされます。

```
kubectl get roles -A
```

```
kubectl get clusterroles
```

- b. 前の出力で返された role または clusterrole の詳細を表示します。クラスター内で IAM プリンシパルに必要となる許可 (rules) があることを確認します。

前のコマンド出力で返された *role-name* を、role の名前に置き換えます。*kube-system* を、role の名前空間に置き換えます。

```
kubectl describe role role-name -n kube-system
```

cluster-role-name を、前のコマンドの出力で返された clusterrole 名に置き換えます。

```
kubectl describe clusterrole cluster-role-name
```

- c. 既存の Kubernetes の rolebindings または clusterrolebindings を表示します。Rolebindings は namespace スコープされ、clusterrolebindings はクラスターにスコープされます。

```
kubectl get rolebindings -A
```

```
kubectl get clusterrolebindings
```

- d. `rolebinding` または `clusterrolebinding` の詳細を表示して、前のステップで `roleRef` としてリストされた `role` または `clusterrole`、そして `subjects` にリストされたグループ名があることを確認します。

role-binding-name を、前のコマンドの出力で返された `rolebinding` の名前に置き換えます。*kube-system* は、`rolebinding` の `namespace` で置き換えます。

```
kubectl describe rolebinding role-binding-name -n kube-system
```

出力例は次のとおりです。

```
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: eks-console-dashboard-restricted-access-role-binding
  namespace: default
subjects:
- kind: Group
  name: eks-console-dashboard-restricted-access-group
  apiGroup: rbac.authorization.k8s.io
roleRef:
  kind: Role
  name: eks-console-dashboard-restricted-access-role
  apiGroup: rbac.authorization.k8s.io
```

cluster-role-binding-name は、前のコマンドの出力で返された `clusterrolebinding` の名前で置き換えます。

```
kubectl describe clusterrolebinding cluster-role-binding-name
```

出力例は次のとおりです。

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: eks-console-dashboard-full-access-binding
subjects:
- kind: Group
  name: eks-console-dashboard-full-access-group
  apiGroup: rbac.authorization.k8s.io
```



```
roleRef:
  kind: ClusterRole
  name: eks-console-dashboard-full-access-clusterrole
  apiGroup: rbac.authorization.k8s.io
```

3. aws-auth ConfigMap を編集します。eksctl のようなツールにより ConfigMap を更新することができます。あるいは、手動で編集しての更新も可能です。

Important

ConfigMap の編集には、eksctl や、その他のツールを使用することをお勧めします。使用できる他のツールについては、Amazon EKS ベストプラクティスガイドの「[ツールを使用して aws-auth ConfigMap を変更する](#)」を参照してください。aws-auth ConfigMap の形式が不適切な場合、クラスターへのアクセスが失われる場合があります。

- [eksctl で configmap を編集する](#) ステップを表示します。
- [configmap を手動で編集する](#) ステップを表示します。

Eksctl で Configmap を編集する

1. デバイスまたは AWS CloudShell にインストールされている eksctl コマンドラインツールのバージョン 0.210.0 以降が必要です。eksctl をインストールまたはアップグレードするには eksctl ドキュメントの「[インストール](#)」を参照してください。
2. ConfigMap に現在のマッピングを表示します。##### の部分は自分のクラスター名に置き換えます。*region-code* を、クラスターのある AWS リージョンに置き換えます。

```
eksctl get iamidentitymapping --cluster my-cluster --region=region-code
```

出力例は次のとおりです。

```
ARN                                     USERNAME                                GROUPS
ACCOUNT
arn:aws:iam::111122223333:role/eksctl-my-cluster-my-nodegroup-
NodeInstanceRole-1XLS7754U3ZPA    system:node:{{EC2PrivateDNSName}}
system:bootstrappers,system:nodes
```

3. ロールのマッピングを追加します。*my-role* を実際のロールの名前に置き換えます。*eks-console-dashboard-full-access-group* を、Kubernetes の RoleBinding または ClusterRoleBinding オブジェクトで指定されたグループの名前に置き換えます。*111122223333* は、ご自分のアカウント ID に置き換えます。*###*を任意の名前に置き換えることができます。

```
eksctl create iamidentitymapping --cluster my-cluster --region=region-code \
    --arn arn:aws:iam::111122223333:role/my-role --username admin --group eks-
console-dashboard-full-access-group \
    --no-duplicate-arns
```

⚠ Important

ロールの ARN には、role/my-team/developers/my-role などのパスを含めることはできません。ロールの ARN は、arn:aws:iam::*111122223333*:role/*my-role* のような形式にする必要があります。この例では、my-team/developers/ を削除する必要があります。

出力例は次のとおりです。

```
[...]
2022-05-09 14:51:20 [#] adding identity "{arn-aws}iam::111122223333:role/my-role" to
auth ConfigMap
```

4. ユーザーのマッピングを追加します。[IAM のベストプラクティス](#)では、ユーザーではなくロールに許可を付与することが推奨されています。*my-user* を、ご自身のユーザー名に置き換えます。*eks-console-dashboard-restricted-access-group* を、Kubernetes RoleBinding または ClusterRoleBinding オブジェクトで指定されたグループの名前に置き換えます。*111122223333* は、ご自分のアカウント ID に置き換えます。*my-user* を任意の名前に置き換えることができます。

```
eksctl create iamidentitymapping --cluster my-cluster --region=region-code \
    --arn arn:aws:iam::111122223333:user/my-user --username my-user --group eks-
console-dashboard-restricted-access-group \
    --no-duplicate-arns
```

出力例は次のとおりです。

```
[...]
2022-05-09 14:53:48 [#] adding identity "arn:aws:iam::111122223333:user/my-user" to
auth ConfigMap
```

5. ConfigMap のマッピングを再度表示します。

```
eksctl get iamidentitymapping --cluster my-cluster --region=region-code
```

出力例は次のとおりです。

ARN	USERNAME ACCOUNT	GROUPS
arn:aws:iam::111122223333:role/eksctl-my-cluster-my-nodegroup-NodeInstanceRole-1XLS7754U3ZPA	system:node:{{EC2PrivateDNSName}}	
arn:aws:iam::111122223333:role/admin	system:bootstrappers,system:nodes	
arn:aws:iam::111122223333:role/my-role	my-user	eks-console-dashboard-full-access-
arn:aws:iam::111122223333:user/my-user	my-user	eks-console-dashboard-restricted-
arn:aws:iam::111122223333:role/my-role	my-user	access-group

Configmap を手動で編集する

1. 編集する ConfigMap を開きます。

```
kubectl edit -n kube-system configmap/aws-auth
```

Note

「Error from server (NotFound): configmaps "aws-auth" not found」というエラーが表示された場合は、「[aws-auth ConfigMap をクラスターに適用する](#)」の手順を使用して、ストック ConfigMap を適用します。

2. IAM プリンシパルを ConfigMap に追加します。IAM グループは IAM プリンシパルではないため、ConfigMap に追加できません。

- IAM ロールを (たとえば、[フェデレーティッドユーザー](#)に) 追加するには、ConfigMap の mapRoles セクションの data の下にロールの詳細を追加します。ファイルに存在しない場合は、このセクションを追加します。各エントリは、次のパラメータをサポートしています。
 - rolearn: 追加する IAM ロールの ARN。この値にパスを含めることはできません。例えば、arn:aws:iam::**111122223333**:role/my-team/developers/**role-name** のような ARN を指定することはできません。ARN は、arn:aws:iam::**111122223333**:role/**role-name** のようにする必要があります。
 - username: IAM ロールにマッピングする Kubernetes 内のユーザー名。
 - groups: ロールをマップする Kubernetes グループのグループまたはリスト。グループには、デフォルトグループか、clusterrolebinding または rolebinding で指定されたグループを使用します。詳細については、Kubernetes ドキュメントの「[デフォルト Role と ClusterRoleBinding](#)」を参照してください。
- IAM ユーザーを追加するには: [IAM のベストプラクティス](#)では、ユーザーではなくロールに許可を付与することが推奨されています。ConfigMap の mapUsers セクション (data の下) にユーザーの詳細を追加します。ファイルに存在しない場合は、このセクションを追加します。各エントリは、次のパラメータをサポートしています。
 - userarn: 追加する IAM ユーザーの ARN。
 - username: IAM ユーザーにマッピングする Kubernetes 内のユーザー名。
 - groups: ユーザーをマップする Kubernetes グループのグループまたはリスト。グループには、デフォルトグループか、clusterrolebinding または rolebinding で指定されたグループを使用します。詳細については、Kubernetes ドキュメントの「[デフォルト Role と ClusterRoleBinding](#)」を参照してください。

3. 例えば、次の YAML ブロックには次が含まれます。

- ノードがそれ自体をクラスターに登録できるように、my-console-viewer-role IAM ノードインスタンスを Kubernetes グループにマッピングしている mapRoles セクション。および、すべてのクラスターですべての Kubernetes リソースを表示することが可能な、Kubernetes グループにマッピングされた IAM ロール。my-console-viewer-role IAM ロールに必要な IAM と Kubernetes グループのアクセス許可のリストについては、「[the section called “必要なアクセス許可”](#)」を参照してください。
- デフォルトの AWS アカウントからの admin IAM ユーザーを system:masters Kubernetes グループにマッピングする mapUsers セクション。および、特定の名前空間で Kubernetes リソースを表示することが可能な、Kubernetes グループにマッピングされた、他の AWS アカウントからの my-user ユーザー。my-user IAM ユーザーに必要な IAM と Kubernetes グループ

のアクセス許可のリストについては、「[the section called “必要なアクセス許可”](#)」を参照してください。

必要に応じて行を追加または削除し、すべての###を、実際の値で置き換えます。

```
# Please edit the object below. Lines beginning with a '#' will be ignored,
# and an empty file will abort the edit. If an error occurs while saving this file
# will be
# reopened with the relevant failures.
#
apiVersion: v1
data:
  mapRoles: |
    - groups:
      - system:bootstrappers
      - system:nodes
      rolearn: arn:aws:iam::111122223333:role/my-role
      username: system:node:{{EC2PrivateDNSName}}
    - groups:
      - eks-console-dashboard-full-access-group
      rolearn: arn:aws:iam::111122223333:role/my-console-viewer-role
      username: my-console-viewer-role
  mapUsers: |
    - groups:
      - system:masters
      userarn: arn:aws:iam::111122223333:user/admin
      username: admin
    - groups:
      - eks-console-dashboard-restricted-access-group
      userarn: arn:aws:iam::444455556666:user/my-user
      username: my-user
```

4. ファイルを保存し、テキストエディタを終了します。

aws-auth ConfigMap をクラスターに適用する

マネージド型ノードグループを作成するとき、または aws-auth を使用してノードグループを作成するときに、ConfigMap eksctl が自動的に作成され、クラスターに適用されます。これはノードをクラスターに追加するために当初作成されたものですが、この ConfigMap を使用して、ロールベースアクセスコントロール (RBAC) アクセスを IAM プリンシパルに追加することもできます。セ

ルフマネージド型ノードが起動済みで、クラスターに aws-auth ConfigMap を適用していない場合は、次の手順に従って適用できます。

1. aws-auth ConfigMap が適用済みであるかどうかを確認します。

```
kubectl describe configmap -n kube-system aws-auth
```

「Error from server (NotFound): configmaps "aws-auth" not found」というエラーが表示された場合は、以下のステップを実行してストック ConfigMap を適用します。

2. AWS オーセンティケーター設定マップをダウンロード、編集、適用します。

- a. 設定マップをダウンロードします。

```
curl -O https://s3.us-west-2.amazonaws.com/amazon-eks/cloudformation/2020-10-29/aws-auth-cm.yaml
```

- b. aws-auth-cm.yaml ファイルで、ノードに関連付けられている IAM ロールの Amazon リソースネーム (ARN) に rolearn を設定します。これを行うには、テキストエディタを使用するか、*my-node-instance-role* を置き換えて次のコマンドを実行します。

```
sed -i.bak -e 's|<ARN of instance role (not instance profile)>|my-node-instance-role|' aws-auth-cm.yaml
```

このファイルの他の行は変更しないでください。

Important

ロールの ARN には、role/my-team/developers/my-role などのパスを含めることはできません。ロールの ARN は、arn:aws:iam::*111122223333*:role/*my-role* のような形式にする必要があります。この例では、my-team/developers/ を削除する必要があります。

ノードグループの AWS CloudFormation スタック出力を検査し、次の値を探することができます。

- InstanceRoleARN – eksctl で作成されたノードグループ用
- NodeInstanceRole – AWS Management Console で、Amazon EKS で発行された AWS CloudFormation テンプレートで作成されたノードグループ用

- c. 設定を適用します。このコマンドが完了するまで数分かかることがあります。

```
kubectl apply -f aws-auth-cm.yaml
```

Note

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

3. ノードのステータスを監視し、Ready ステータスになるまで待機します。

```
kubectl get nodes --watch
```

Ctrl+C を入力して、シェルプロンプトに戻ります。

外部 OIDC プロバイダーを使用して Kubernetes へのアクセスをユーザーに許可する

Amazon EKS では、クラスターに対してユーザーを認証する方法として OpenID Connect (OIDC) ID プロバイダーがサポートされます。OIDC ID プロバイダーはAWS アイデンティティとアクセス管理 (IAM) と併用、または代替として使用できます。IAM の使用の詳細については、「[the section called “Kubernetes API アクセス”](#)」を参照してください。クラスターに認証を設定した後は、Kubernetes roles および clusterroles を作成しロールにアクセス許可を割り当て、さらに、このロールを Kubernetes rolebindings および clusterrolebindings を使用する ID にバインドできます。詳細についてはKubernetes ドキュメントの「[RBAC 認可を使用する](#)」を参照してください。

- クラスターには OIDC ID プロバイダーを 1 つ関連付けることができます。
- Kubernetes は OIDC ID プロバイダーを提供しません。既存のパブリック OIDC ID プロバイダーを使用することも、独自の ID プロバイダーを実行することもできます。認定プロバイダーのリストについては、OpenID サイトの「[OpenID Certification](#)」(OpenID 認定) を参照してください。
- Amazon EKS が署名キーを検出できるように、OIDC ID プロバイダーの発行者 URL は公的にアクセス可能である必要があります。Amazon EKS は、自己署名証明書を持つ OIDC ID プロバイダーをサポートしていません。
- ノードをクラスターに結合する時に必要なため、クラスターへの IAM 認証を無効にすることはできません。

- Amazon EKS クラスターは、OIDC ID プロバイダーユーザーではなく、AWS [IAM プリンシパル](#)が作成する必要があります。これは、クラスター作成者が Kubernetes API ではなく Amazon EKS API と対話するためです。
- コントロールプレーンで CloudWatch ログが有効になっている場合、OIDC ID プロバイダー認証ユーザーは、クラスターの監査ログに一覧表示されます。詳細については、「[the section called “コントロールプレーンログを有効または無効にする”](#)」を参照してください。
- OIDC プロバイダーのアカウントで AWS Management Consoleにサインインすることはできません。AWS アイデンティティとアクセス管理 アカウントで AWS Management Consoleにサインインした場合にのみ、[the section called “クラスターリソースにアクセスする”](#) が可能となります。

OIDC ID プロバイダーを関連付ける

OIDC ID プロバイダーをクラスターに関連付けるには、プロバイダーから以下の情報が必要です。

発行者 URL

API サーバーがトークン検証のために公開署名キーを検出できるようにする OIDC ID プロバイダーの URL。URL は `https://` で始まり、プロバイダーの OIDC ID トークンで `iss` クレームに対応している必要があります。OIDC 標準に従って、パスコンポーネントは許可されますが、クエリパラメータは許可されません。通常 URL は <https://server.example.org> または <https://example.com> のように、ホスト名のみで構成されます。この URL は以下 `.well-known/openid-configuration` のレベルを指し、インターネット経由で公的にアクセス可能である必要があります。

クライアント ID (オーディエンスとも呼ばれる)

OIDC ID プロバイダーに認証リクエストを行うクライアントアプリケーションの ID。

`eksctl` または AWS Management Console を使用して、ID プロバイダーを関連付けることができます。

`eksctl` を使用して ID プロバイダーを関連付ける

1. 以下の内容を含む `associate-identity-provider.yaml` という名前のファイルを作成します。`###` は実際に使用する値に置き換えます。`identityProviders` セクションの値は、OIDC ID プロバイダーから取得します。値は `identityProviders` での `name`、`type`、`issuerUrl`、および `clientId` の設定のみで必要です。

```
---
```



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

metadata:
  name: my-cluster
  region: your-region-code

identityProviders:
  - name: my-provider
    type: oidc
    issuerUrl: https://example.com
    clientId: kubernetes
    usernameClaim: email
    usernamePrefix: my-username-prefix
    groupsClaim: my-claim
    groupsPrefix: my-groups-prefix
    requiredClaims:
      string: string
    tags:
      env: dev
```

Important

groupsPrefix または usernamePrefix に対しては system: またはその文字列の一部でも指定しないでください。

2. プロバイダーを作成します。

```
eksctl associate identityprovider -f associate-identity-provider.yaml
```

3. kubectl を使用してクラスターと OIDC ID プロバイダーを操作する方法については、Kubernetes ドキュメントの「[kubectl の使用](#)」を参照してください。

AWS コンソールを使用して ID プロバイダーを関連付ける

1. [アマゾン EKS コンソール](#)を開きます。
2. クラスターを選択し、[アクセス] タブを選択してください。
3. [OIDC ID プロバイダー] セクションで、[ID プロバイダーを関連付け] を選択します。
4. [Associate OIDC Identity Provider] (OIDC ID プロバイダーの関連付け) ページで、以下のオプションを入力または選択した上で、[Associate] (関連付け) を選択します。

- [Name] (名前) に、プロバイダーの一意の名前を入力します。
- [Issuer URL] (発行者 URL) に、プロバイダーの URL を入力してください。この URL はインターネットからアクセス可能である必要があります。
- [クライアント ID] に、OIDC ID プロバイダーのクライアント ID (別称、対象者) を入力します。
- [Username claim] (ユーザー名のクレーム) に、ユーザー名として使用するクレームを入力してください。
- [Groups claim] (グループクレーム) に、ユーザーのグループとして使用するクレームを入力します。
- (オプション) [Advanced options] (詳細オプション) で、以下の情報を入力または選択してください。
 - [Username prefix] (ユーザー名のプレフィックス) – ユーザー名クレームの前に付加するプレフィックスを入力してください。プレフィックスは既存の名前との衝突を防ぐために、ユーザー名クレームの前に付加されます。値を入力せず、ユーザーネームが email 以外の場合、プレフィックスはデフォルトで [Issuer URL] (発行者 URL) の値になります。値に - を使用すると、すべてのプレフィックスを無効にできます system: またはその文字列の一部でも指定しないでください。
 - [Groups prefix] (グループプレフィックス) – グループクレームの前に付加するプレフィックスを入力してください。プレフィックスは system: groups など既存の名前との衝突を防ぐために、グループクレームの前に付加されます。例えば、値 oidc: は oidc:engineering および oidc:infra などのグループ名を作成します。system: またはその文字列の一部でも指定しないでください。
 - [Required claims] (必要なクレーム) – [Add claim] (クレームを追加) 選択し、クライアント ID トークンに必要なクレームを記述する、キーと値のペアを 1 つ以上入力してください。ペアは ID トークンで必要なクレームを記述します。設定されている場合、各クレームは一致する値を持つ ID トークンに存在することが検証されます。
 - a. kubectl を使用してクラスターと OIDC ID プロバイダーを操作する方法については、Kubernetes ドキュメントの「[kubectl の使用](#)」を参照してください。

IAM ポリシーの例

OIDC ID プロバイダーとクラスターの関連付けを防ぐには、以下の IAM ポリシーを作成して、Amazon EKS 管理者の IAM アカウントに関連付けます。詳細については「IAM ユーザーガイド」の「[IAM ポリシーを作成する](#)」と「[IAM ID アクセス許可の追加](#)」、および「サービス認証リファレンス」の「[Actions](#)」に関するページを参照してください。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "denyOIDC",
      "Effect": "Deny",
      "Action": [
        "eks:AssociateIdentityProviderConfig"
      ],
      "Resource": "arn:aws:eks:us-west-2.amazonaws.com:111122223333:cluster/*"
    },
    {
      "Sid": "eksAdmin",
      "Effect": "Allow",
      "Action": [
        "eks:*"
      ],
      "Resource": "*"
    }
  ]
}
```

以下のポリシーの例では、clientID が kubernetes で、issuerUrl が [https://cognito-idp.us-west-2amazonaws.com/](https://cognito-idp.us-west-2.amazonaws.com/) の場合、OIDC ID プロバイダーの関連付けを許可します。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "AllowCognitoOnly",
      "Effect": "Deny",
      "Action": "eks:AssociateIdentityProviderConfig",
      "Resource": "arn:aws:eks:us-west-2:111122223333:cluster/my-instance",
      "Condition": {
        "StringNotLikeIfExists": {
          "eks:issuerUrl": "https://cognito-idp.us-west-2.amazonaws.com/*"
        }
      }
    },
    {
      "Sid": "DenyOtherClients",
      "Effect": "Deny",
```

```
    "Action": "eks:AssociateIdentityProviderConfig",
    "Resource": "arn:aws:eks:us-west-2:111122223333:cluster/my-instance",
    "Condition": {
      "StringNotEquals": {
        "eks:clientId": "kubernetes"
      }
    }
  },
  {
    "Sid": "AllowOthers",
    "Effect": "Allow",
    "Action": "eks:*",
    "Resource": "*"
  }
]
```

OIDC ID プロバイダーとクラスターの関連付けを解除します。

OIDC ID プロバイダーとクラスターの関連付けを解除する場合、プロバイダーに含まれるユーザーはクラスターにアクセスできなくなります。ただし、[IAM プリンシパル](#)を使用して、クラスターへ引き続きアクセスすることは可能です。

1. [アマゾン EKS コンソール](#)を開きます。
2. [OIDC Identity Providers] (OIDC ID プロバイダー) セクションで、[Disassociate] (関連付けの解除) を選択しID プロバイダー名を入力した上で、[Disassociate] を選択します。

AWS Management Console に Kubernetes リソースを表示する

クラスターにデプロイされた Kubernetes リソースを AWS Management Console で表示できます。Kubernetes リソースは、AWS CLIまたは [eksctl](#) で表示することはできません。コマンドラインツールを使用して Kubernetes リソースを表示するには、[kubect](#)を使用します。

Note

AWS Management Console の [リソース] タブと [コンピューティング] タブの [ノード] セクションを表示するには、使用している [IAM プリンシパル](#)に特定の IAM と Kubernetes 許可が必要です。詳細については、「[the section called “必要なアクセス許可”](#)」を参照してください。

1. [Amazon EKS コンソール](#)を開きます。
2. [Clusters] (クラスター) のリストで、表示したい Kubernetes リソースを含んでいるクラスターを選択します。
3. [リソース] タブを選択します。
4. リソースを表示したい [リソースタイプ] グループを選択します。例えば、[ワークロード] などです。そのグループ内のリソースタイプのリストが表示されます。
5. リソースタイプを選択します。例えば、[Workloads] (ワークロード) グループ内の [Deployments] (デプロイメント) などです。リソースタイプの簡単な説明、リソースタイプの詳細な説明が参照できる Kubernetes ドキュメントへのリンク、およびクラスターにデプロイされていてそのタイプに分類されるリソースのリストが表示されます。リストが空の場合、そのタイプのリソースはクラスターにデプロイされていません。
6. 詳細情報を表示するには、そのリソースを選択します。次の例を試してください。
 - [ワークロード] グループを選択し、[デプロイメント] のリソースタイプを選択し、次に [CoreDNS] リソースを選択します。リソースを選択すると、デフォルトで [構造化ビュー] が表示されます。リソースタイプによっては、[構造化ビュー] に [ポッド] セクションが表示されます。このセクションでは、ワークロードによって管理される Pod が一覧表示されます。一覧表示されている任意の Pod を選択し、Pod の情報を表示できます。すべてのリソースタイプについて、[Structured View] (構造化ビュー) で情報が表示されるわけではありません。リソース表示画面で右上隅にある [Raw view] (raw ビュー) を選択すると、そのリソースについて Kubernetes API からの完全な JSON レスポンスが表示されます。
 - [Cluster] (クラスター) グループを選択してから、[Nodes] (ノード) リソースタイプを選択します。クラスター内のすべてのノードのリストが表示されます。ノードは、任意の [Amazon EKS ノードタイプ](#) となります。これはクラスターの [コンピューティング] () タブを選択したときに [ノード] セクションに表示されるのと同じリストです。リストからノードリソースを選択します。[構造化ビュー] では、[ポッド] セクションも表示されます。このセクションでは、ノード上で実行中のすべての Pod が表示されます。

必要なアクセス許可

AWS Management Console の [リソース] タブと [コンピューティング] タブの [ノード] セクションを表示するには、使用している [IAM プリンシパル](#) に特定の最小限の IAM と Kubernetes 許可が必要です。次のステップを実行して、IAM プリンシパルに必要な許可を割り当てます。

1. `eks:AccessKubernetesApi`、および `Kubernet` リソースを表示するために必要なその他の IAM アクセス許可が、使用している IAM プリンシパルに割り当てられていることを確認してく

ださい。IAM プリンシパルの許可を編集する方法の詳細については、「IAM ユーザーガイド」の「[プリンシパルへのアクセスの制御](#)」を参照してください。ロールのアクセス許可を編集する方法の詳細については、IAM ユーザーガイドの「[ロールのアクセス許可ポリシーの変更 \(コンソール\)](#)」を参照してください。

次のポリシーの例には、アカウント内のすべてのクラスター内にある Kubernetes リソースを表示するために必要な、プリンシパルへの許可が含まれています。**111122223333** を自分の AWS アカウント ID に置き換えます。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "eks:ListFargateProfiles",
        "eks:DescribeNodegroup",
        "eks:ListNodegroups",
        "eks:ListUpdates",
        "eks:AccessKubernetesApi",
        "eks:ListAddons",
        "eks:DescribeCluster",
        "eks:DescribeAddonVersions",
        "eks:ListClusters",
        "eks:ListIdentityProviderConfigs",
        "iam:ListRoles"
      ],
      "Resource": "*"
    },
    {
      "Effect": "Allow",
      "Action": "ssm:GetParameter",
      "Resource": "arn:aws:ssm*:111122223333:parameter/*"
    }
  ]
}
```

[接続されたクラスター](#)内のノードを表示するには、[Amazon EKS コネクタの IAM ロール](#)がクラスター内のプリンシパルを偽装できる必要があります。これにより、[Amazon EKS Connector](#) はプリンシパルを Kubernetes ユーザーにマッピングできます。

2. Kubernetes リソースを表示するのに必要なアクセス許可を持つ Kubernetes role または clusterrole にバインドされている Kubernetes rolebinding または clusterrolebinding を作成します。Kubernetes のロールとロールバインドの詳細については、Kubernetes ドキュメントの「[RBAC 認可を使用する](#)」を参照してください。次のマニフェストのいずれかをクラスターに適用して、必要な Kubernetes アクセス許可を持つ role と rolebinding、または clusterrole と clusterrolebinding を作成できます。

すべての名前空間の Kubernetes リソースを表示する

- ファイル内のグループ名は eks-console-dashboard-full-access-group です。次のコマンドを使用して、クラスターにマニフェストを適用します。

```
kubectl apply -f https://s3.us-west-2.amazonaws.com/amazon-eks/docs/eks-console-full-access.yaml
```

特定の名前空間の Kubernetes リソースを表示する

- このファイルの名前空間は default です。ファイル内のグループ名は eks-console-dashboard-restricted-access-group です。次のコマンドを使用して、クラスターにマニフェストを適用します。

```
kubectl apply -f https://s3.us-west-2.amazonaws.com/amazon-eks/docs/eks-console-restricted-access.yaml
```

ファイル内の Kubernetes グループ名、名前空間、アクセス許可、またはその他の設定を変更する必要がある場合は、ファイルをダウンロードして編集してからクラスターに適用します。

- a. 次のいずれかのコマンドを使って、ファイルをダウンロードします。

```
curl -O https://s3.us-west-2.amazonaws.com/amazon-eks/docs/eks-console-full-access.yaml
```

```
curl -O https://s3.us-west-2.amazonaws.com/amazon-eks/docs/eks-console-restricted-access.yaml
```

- b. 必要に応じてファイルを編集します。
- c. 次のコマンドのいずれかを使用して、クラスターにマニフェストを適用します。

```
kubectl apply -f eks-console-full-access.yaml
```

```
kubectl apply -f eks-console-restricted-access.yaml
```

- aws-auth ConfigMap のユーザーまたはグループに [IAM プリンシパル](#) をマッピングします。eksctl のようなツールにより ConfigMap を更新することができます。あるいは、手動で編集しての更新も可能です。

Important

ConfigMap の編集には、eksctl や、その他のツールを使用することをお勧めします。使用できる他のツールについては、Amazon EKS ベストプラクティスガイドの「[ツールを使用して aws-auth ConfigMap を変更する](#)」を参照してください。aws-auth ConfigMap の形式が不適切な場合、クラスターへのアクセスが失われる場合があります。

eksctl で編集する

- デバイスまたは AWS クラウドシェル にインストールされている eksctl コマンドラインツールのバージョン 0.210.0 以降が必要です。eksctl をインストールまたはアップグレードするには eksctl ドキュメントの「[インストール](#)」を参照してください。
- ConfigMap に現在のマッピングを表示します。##### の部分は自分のクラスター名に置き換えます。region-code を、クラスターのある AWS リージョンに置き換えます。

```
eksctl get iamidentitymapping --cluster my-cluster --region=region-code
```

出力例は次のとおりです。

ARN	USERNAME	GROUPS
ACCOUNT		
arn:aws:iam::111122223333:role/eksctl-my-cluster-my-nodegroup-		
NodeInstanceRole-1XLS7754U3ZPA	system:node:{{EC2PrivateDNSName}}	
	system:bootstrappers,system:nodes	

- ロールのマッピングを追加します。この例では、最初のステップで *my-console-viewer-role* という名前のロールに IAM アクセス許可をアタッチしたと仮定します。&ExampleAWSAccountNo1 は、ご自分のアカウント ID に置き換えます。

```
eksctl create iamidentitymapping \
```



```
--cluster my-cluster \  
--region=region-code \  
--arn arn:aws:iam::111122223333:role/my-console-viewer-role \  
--group eks-console-dashboard-full-access-group \  
--no-duplicate-arns
```

Important

ロールの ARN には、role/my-team/developers/my-role などのパスを含めることはできません。ロールの ARN は、arn:aws:iam::**111122223333**:role/**my-role** のような形式にする必要があります。この例では、my-team/developers/ を削除する必要があります。

出力例は次のとおりです。

```
[...]
2022-05-09 14:51:20 [#] adding identity "arn:aws:iam::111122223333:role/my-console-viewer-role" to auth ConfigMap
```

4. ユーザーのマッピングを追加します。[IAM のベストプラクティス](#)では、ユーザーではなくロールに許可を付与することが推奨されています。この例では、最初のステップで **my-user** という名前のユーザーに IAM アクセス許可をアタッチしたと仮定します。**&ExampleAWSAccountNo1** は、ご自分のアカウント ID に置き換えます。

```
eksctl create iamidentitymapping \  
--cluster my-cluster \  
--region=region-code \  
--arn arn:aws:iam::111122223333:user/my-user \  
--group eks-console-dashboard-restricted-access-group \  
--no-duplicate-arns
```

出力例は次のとおりです。

```
[...]
2022-05-09 14:53:48 [#] adding identity "arn:aws:iam::111122223333:user/my-user" to auth ConfigMap
```

5. ConfigMap のマッピングを再度表示します。

```
eksctl get iamidentitymapping --cluster my-cluster --region=region-code
```

出力例は次のとおりです。

ARN	USERNAME ACCOUNT	GROUPS
arn:aws:iam::111122223333:role/eksctl-my-cluster-my-nodegroup-NodeInstanceRole-1XLS7754U3ZPA	system:node:{{EC2PrivateDNSName}}	
arn:aws:iam::111122223333:role/my-console-viewer-role	system:bootstrappers,system:nodes	
		eks-console-dashboard-full-access-group
arn:aws:iam::111122223333:user/my-user		
		eks-console-dashboard-restricted-access-group

ConfigMap を手動で編集する

aws-auth ConfigMap にユーザまたはロールを追加する方法の詳細については、「[the section called “Amazon EKS クラスタに IAM プリンシパルを追加する”](#)」を参照してください。

1. 編集する aws-auth ConfigMap を開きます。

```
kubectl edit -n kube-system configmap/aws-auth
```

2. aws-auth ConfigMap にマッピングを追加しますが、既存のマッピングを置き換えないでください。次の例では、最初のステップで追加した許可を持つ [IAM プリンシパル](#)と、前のステップで作成した Kubernetes グループの間のマッピングを追加します。

- **my-console-viewer-role** ロールと eks-console-dashboard-full-access-group。
- **my-user** ユーザーと eks-console-dashboard-restricted-access-group。

これらの例では、最初のステップで **my-console-viewer-role** という名前のロール、および **my-user** という名前のユーザーに、IAM アクセス許可をアタッチしたと仮定します。**111122223333** を自分の AWS アカウント ID に置き換えます。

```
apiVersion: v1
data:
```

```
mapRoles: |
  - groups:
    - eks-console-dashboard-full-access-group
    rolearn: arn:aws:iam::111122223333:role/my-console-viewer-role
    username: my-console-viewer-role
mapUsers: |
  - groups:
    - eks-console-dashboard-restricted-access-group
    userarn: arn:aws:iam::111122223333:user/my-user
    username: my-user
```

Important

ロールの ARN には、role/my-team/developers/my-console-viewer-role などのパスを含めることはできません。ロールの ARN は、arn:aws:iam::**111122223333**:role/**my-console-viewer-role** のような形式にする必要があります。この例では、my-team/developers/ を削除する必要があります。

3. ファイルを保存し、テキストエディタを終了します。

kubeconfig ファイルを作成して kubectl を EKS クラスターに接続する

このトピックでは、クラスター用の kubeconfig ファイルを作成します (または既存のファイルを更新します)。

kubectl コマンドラインツールは、kubeconfig ファイルの設定情報を使用して、クラスターの API サーバーと通信します。詳細については、Kubernetes ドキュメントの「[kubeconfig ファイルを使用してクラスターアクセスを組織する](#)」を参照してください。

Amazon EKS はクラスター認証に kubectl で aws eks get-token コマンドを使用します。デフォルトでは、AWS CLI は次のコマンドで返されるものと同じ認証情報を使用します。

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

- 既存の Amazon EKS クラスター。デプロイするには「[はじめに](#)」を参照してください。

- デバイスまたは AWS クラウドシェル に、kubectl コマンドラインツールがインストールされていること。バージョンはご使用のクラスターの Kubernetes バージョンと同じか、1 つ前のマイナーバージョン以前、あるいはそれより新しいバージョンが使用できます。例えば、クラスターのバージョンが 1.29 である場合、kubectl のバージョン 1.28、1.29、または 1.30 が使用できます。kubectl をインストールまたはアップグレードする方法については「[the section called “kubectl と eksctl のセットアップ”](#)」を参照してください。
- ご使用のデバイスまたは AWS クラウドシェル で、バージョン 2.12.3 以降、または AWS コマンドラインインターフェイス (AWS CLI のバージョン 1.27.160 以降がインストールおよび設定されていること。現在のバージョンを確認するには「aws --version | cut -d / -f2 | cut -d ' ' -f1」を参照してください。yum、apt-get、macOS 用の Homebrew などのパッケージマネージャーは、多くの場合 AWS CLI の最新バージョンより数バージョン古くなっています。最新バージョンをインストールするには「AWS コマンドラインインターフェイスユーザーガイド」の「[インストール](#)」および「[aws configure を使用したクイック設定](#)」を参照してください。AWS クラウドシェル にインストールされている AWS CLI バージョンも最新バージョンより数バージョン遅れることがあります。更新するには、「AWS CloudShell ユーザーガイド」の「[ホームディレクトリへの AWS CLI のインストール](#)」を参照してください。
- 指定したクラスターに対して eks:DescribeCluster API アクションを使用するアクセス許可を持つ IAM ユーザーまたはロール。詳細については「[the section called “アイデンティティベースのポリシー”](#)」を参照してください。独自の OpenID Connect プロバイダーの ID を使用してクラスターにアクセスする場合は、Kubernetes ドキュメントの「[kubectl の使用](#)」を参照して、kubeconfig ファイルを作成または更新してください。

kubeconfig ファイルを自動で作成する

- ご使用のデバイスまたは AWS CloudShell で、バージョン 2.12.3 以降、または AWS コマンドラインインターフェイス (AWS CLI) のバージョン 1.27.160 以降がインストールおよび設定されていること。現在のバージョンを確認するには「aws --version | cut -d / -f2 | cut -d ' ' -f1」を参照してください。yum、apt-get、macOS 用の Homebrew などのパッケージマネージャーは、多くの場合 AWS CLI の最新バージョンより数バージョン古くなっています。最新バージョンをインストールするには「AWS コマンドラインインターフェイスユーザーガイド」の「[インストール](#)」および「[aws configure を使用したクイック設定](#)」を参照してください。AWS クラウドシェル にインストールされている AWS CLI バージョンも最新バージョンより数バージョン遅れることがあります。更新するには、「AWS CloudShell ユーザーガイド」の「[ホームディレクトリへの AWS CLI のインストール](#)」を参照してください。

- 指定したクラスターに対して `eks:DescribeCluster` API アクションを使用するアクセス許可。詳細については「[the section called “アイデンティティベースのポリシー”](#)」を参照してください。
- クラスター用の `kubeconfig` ファイルを作成もしくは更新します。`region-code` はクラスターがある AWS リージョンに、`my-cluster` はクラスターの名前に置き換えます。

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

デフォルトでは、最終的な設定ファイルは、ホームディレクトリのデフォルト `kubeconfig` パス (`.kube`) に作成されるか、その場所で既存 `config` ファイルとマージされます。別のパスは `--kubeconfig` オプションを使用して指定できます。

`kubectl` コマンドを発行する場合、認証に使用する `--role-arn` オプションで IAM ロール ARN を指定できます。それ以外の場合は、デフォルトの AWS CLI または SDK の認証情報チェーンの中の [IAM プリンシパル](#) が使用されます。デフォルトの AWS CLI または SDK の ID を表示するには、`aws sts get-caller-identity` コマンドを実行します。

使用可能なすべてのオプションについては、`aws eks update-kubeconfig help` コマンドを実行するか、「AWS CLI コマンドリファレンス」の「[update-kubeconfig](#)」を参照してください。

- 設定をテストします。

```
kubectl get svc
```

出力例は次のとおりです。

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

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

Kubernetes ワークロードに Kubernetes サービスアカウントを使用して AWS へのアクセスを許可する

A Kubernetes service account provides an identity for processes that run in a Pod. For more information see [サービスアカウントの管理](#) in the Kubernetes documentation. If your Pod needs

access to AWS services, you can map the service account to an AWS Identity and Access Management identity to grant that access. For more information, see [the section called “IRSA による認証情報”](#) or [the section called “Pod Identity”](#).

サービスアカウントトークン

[BoundServiceAccountTokenVolume](#) 機能は、Kubernetes バージョンでデフォルトで有効になっています。この機能により、Kubernetes で実行されているワークロードは、対象者、時間、および鍵に紐付けされた JSON ウェブトークンをリクエストできるようになり、サービスアカウントトークンのセキュリティが向上します。サービスアカウントトークンの有効期限は 1 時間です。Kubernetes の以前のバージョンでは、トークンに有効期限はありませんでした。つまり、これらのトークンに依存するクライアントは 1 時間以内にトークンを更新する必要があります。以下の [Kubernetes クライアント SDK](#) は、必要な期間内にトークンを自動的に更新します。

- Go バージョン 0.15.7 以降
- Python バージョン 12.0.0 以降
- Java バージョン 9.0.0 以降
- JavaScript バージョン 0.10.3 以降
- Ruby master ブランチ
- Haskell バージョン 0.3.0.0
- C# バージョン 7.0.5 以降

ワークロードで古いバージョンのクライアントを使用している場合は、更新する必要があります。有効期限付きの新しいサービスアカウントトークンにクライアントがスムーズに移行できるようにするため、Kubernetes ではデフォルトの 1 時間を超えてサービスアカウントトークンの有効期限が延長されます。Amazon EKS クラスターの場合、延長できる有効期限は 90 日です。Amazon EKS クラスターの Kubernetes API サーバーでは、90 日を超えるトークンのリクエストは拒否されます。アプリケーションとその依存関係を確認し、Kubernetes クライアント SDK が前記のバージョンと同じかそれ以降であることを確認することをお勧めします。

API サーバーが 1 時間を超える古いトークンによるリクエストを受信すると、API 監査ログイベントに `annotations.authentication.k8s.io/stale-token` で注釈を付けます。注釈の値は次の例のようになります。

```
subject: system:serviceaccount:common:fluent-bit, seconds after warning threshold: 4185802.
```

クラスターで[コントロールプレーンのログ記録](#)が有効になっている場合、注釈は監査ログに記録されます。以下の [CloudWatch Log Insights](#) クエリを使用すると、Amazon EKS クラスター内で古いトークンを使っているすべての Pod を特定できます。

```
fields @timestamp
|filter @logStream like /kube-apiserver-audit/
|filter @message like /seconds after warning threshold/
|parse @message "subject: *, seconds after warning threshold:*\" as subject,
elapsedtime
```

subject は、Pod が使用したサービスアカウントを示します。elapsedtime は、最新のトークンを読み込んでからの経過時間 (秒単位) を示します。elapsedtime が 90 日 (7,776,000 秒間) を超えると、API サーバーへのリクエストは拒否されます。トークンの自動更新を行う前記のバージョンのいずれかを使用するよう、アプリケーションの Kubernetes クライアント SDK をプロアクティブに更新する必要があります。使用しているサービスアカウントトークンが 90 日に近く、トークンの有効期限が切れるまでクライアント SDK のバージョンを更新するのに十分な時間がない場合、既存の Pod を終了して新しい Pod を作成できます。サービスアカウントトークンが再フェッチされるので、クライアントバージョン SDK を更新するのに 90 日が追加されたことになります。

Pod がデプロイの一部である場合、高可用性を維持しながら Pod を終了する方法として、次のコマンドによるロールアウトの実行をお勧めします。*my-deployment* をデプロイの名前で置き換えます。

```
kubectl rollout restart deployment/my-deployment
```

クラスターアドオン

以下のクラスターアドオンが更新され、サービスアカウントトークンを自動的に再フェッチする Kubernetes クライアント SDK が使用できるようになりました。リストされているバージョン、またはそれ以降のバージョンを、クラスターにインストールすることをお勧めします。

- Amazon VPC CNI Plugin for Kubernetes およびメトリクスヘルパーのプラグインのバージョン 1.8.0 以降。現在のバージョンを確認したり、更新したりするには、「[the section called “Amazon VPC CNI”](#)」および「[cni-metrics-helper](#)」を参照してください。
- CoreDNS のバージョン 1.8.4 以降。現在のバージョンを確認したり、更新したりするには、「[the section called “CoreDNS”](#)」を参照してください。

- AWS Load Balancer Controller のバージョン 2.0.0 以降。現在のバージョンを確認したり、更新したりするには、「[the section called “AWS ロードバランサーコントローラー”](#)」を参照してください。
- 現在の kube-proxy バージョン 現在のバージョンを確認したり、更新したりするには、「[the section called “kube-proxy”](#)」を参照してください。
- AWS for Fluent Bit バージョン 2.25.0 以降。現在のバージョンを更新するには、GitHub で「[Releases](#)」(リリース) を参照してください。
- Fluentd イメージバージョン [1.14.6-1.2](#) 以降、および Kubernetes メタデータバージョン [2.11.1](#) 以降用の Fluentd フィルタープラグイン。

Amazon Elastic Kubernetes Service クラスター上のワークロードへの AWS Identity and Access Management 許可の付与

Amazon EKS には、Amazon EKS クラスターで実行されるワークロードに AWS Identity and Access Management 許可を付与する方法が 2 つあります。サービスアカウント用の IAM ロールと EKS Pod Identity です。

サービスアカウントの IAM ロール

サービスアカウント用の IAM ロール (IRSA) は、Amazon S3 バケットや Amazon DynamoDB テーブルなど、他の様々な AWS リソースにアクセスするためのきめ細かい IAM アクセス許可で AWS 上で動作する Kubernetes アプリケーションを構成します。同じ Amazon EKS クラスター内で複数のアプリケーションを同時に実行し、各アプリケーションに必要な最小限のアクセス権限のみを持たせることができます。IRSA は、Amazon EKS、Amazon EKS Anywhere、AWS での Red Hat OpenShift サービス、Amazon EC2 インスタンスでのセルフマネージド型 Kubernetes クラスターなどの、AWS でサポートされるさまざまな Kubernetes デプロイオプションをサポートするように構築されました。そのため、IRSA は IAM のような基本的な AWS サービスを使用して構築されており、Amazon EKS サービスや EKS API に直接依存することはありませんでした。詳細については、「[the section called “IRSA による認証情報”](#)」を参照してください。

EKS Pod Identity

EKS Pod Identity は、Amazon S3 バケット、Amazon DynamoDB テーブルなどのさまざまな AWS リソースにアクセスするためのアプリケーションを認証するためのシンプルなワークフローをクラスター管理者に提供します。EKS Pod Identity は EKS 専用であるため、クラスター管理者が Kubernetes アプリケーションを設定して IAM アクセス許可を取得する方法が簡単になります。これらの権限は、AWS Management Console、EKS API、および AWS CLI から直接行

う、より少ない手順で簡単に設定できるようになり、クラスター内のどの Kubernetes オブジェクトに対しても実行する必要がありません。クラスター管理者は EKS サービスと IAM サービスを切り替えたり、特権的な IAM オペレーションを使用してアプリケーションに必要な権限を設定したりする必要がありません。新しいクラスターを作成するときにロールの信頼ポリシーを更新しなくても、IAM ロールを複数のクラスターで使用できるようになりました。EKS Pod Identity が提供する IAM 認証情報には、クラスター名、名前空間、サービスアカウント名などの属性を含むロールセッションタグが含まれます。ロールセッションタグを使用すると、管理者は一致するタグに基づいて AWS リソースへのアクセスを許可することで、複数のサービスアカウントで機能する単一のロールを作成できます。詳細については、「[the section called “Pod Identity”](#)」を参照してください。

EKS Pod Identity と IRSA の比較

大まかに言うと、EKS Pod Identity と IRSA はどちらも、Kubernetes クラスター上で実行されているアプリケーションに IAM アクセス許可を付与できるようにします。ただし、設定方法、サポートされる制限、有効になる機能は根本的に異なります。以下では、両ソリューションの重要な側面をいくつか比較します。

属性	EKS Pod Identity	IRSA
ロール拡張性	新しく導入された Amazon EKS サービスプリンシパル <code>pods.eks.amazonaws.com</code> との信頼を確立するには、各ロールを一度設定する必要があります。この 1 回限りのステップを済ませると、そのロールを新しいクラスターで使用するたびにそのロールの信頼ポリシーを更新する必要がなくなります。	新しいクラスターでロールを使用するたびに、IAM ロールの信頼ポリシーを新しい EKS クラスター OIDC プロバイダーエンドポイントで更新する必要があります。
クラスターのスケーラビリティ	EKS Pod Identity では、ユーザーが IAM OIDC プロバイダーをセットアップする必要がないため、この制限は適用されません。	各 EKS クラスターには、OpenID Connect (OIDC) 発行者の URL が関連付けられています。IRSA を使用するには、IAM の EKS クラスターごとに

属性	EKS Pod Identity	IRSA
		固有の OpenID Connect プロバイダーを作成する必要があります。IAM のデフォルトのグローバル制限は、AWS アカウントにつき 100 の OIDC プロバイダーです。IRSA で AWS アカウントにつき 100 を超える EKS クラスターを使用する予定の場合は、IAM OIDC プロバイダーの制限に達します。
ロールのスケラビリティ	EKS Pod Identity では、ユーザーが信頼ポリシーで IAM ロールとサービスアカウント間の信頼関係を定義する必要がないため、この制限は適用されません。	IRSA では、IAM ロールとサービスアカウント間の信頼関係を、ロールの信頼ポリシーで定義します。デフォルトでは、信頼ポリシーの長さは 2048 です。つまり、通常、1 つの信頼ポリシーで 4 つの信頼関係を定義できます。信頼ポリシーの長さの上限を増やすことはできますが、通常、1 つの信頼ポリシー内で作成できる信頼関係は最大 8 つに制限されます。

属性	EKS Pod Identity	IRSA
ロールの再利用性	EKS Pod Identity によって提供される一時的な AWS STS 認証情報には、クラスター名、名前空間、サービスアカウント名などのロールセッションタグが含まれます。ロールセッションタグを使用すると、管理者は、アタッチされたタグに基づいて AWS リソースへのアクセスを許可することで、複数のサービスアカウントで使用する単一の IAM ロールを作成し、そのロールを有効な権限を変えることができます。これは、属性ベースのアクセス制御 (ABAC) とも呼ばれます。詳細については、「the section called “Pod にアクセス権を付与する”」を参照してください。	AWS STS セッションタグはサポートされていません。ロールはクラスター間で再利用できますが、すべてのポッドにそのロールのすべての権限が付与されます。
サポートされている環境	EKS Pod Identity は Amazon EKS でのみ使用できます。	IRSA は、Amazon EKS、Amazon EKS Anywhere、AWS での Red Hat OpenShift サービス、Amazon EC2 インスタンスでのセルフマネージド型 Kubernetes クラスターなどに使用できます。

属性	EKS Pod Identity	IRSA
サポートされている EKS バージョン	サポート対象のすべての EKS クラスターバージョン。特定のバージョンについては、「 the section called “EKS Pod Identity クラスターバージョン” 」を参照してください。	サポート対象のすべての EKS クラスターバージョン。

サービスアカウントの IAM ロール

ポッドのコンテナ内のアプリケーションは AWS SDK または AWS CLI で、AWS Identity and Access Management (IAM) アクセス許可を使用した AWS サービスへの API リクエストを行うことができます。アプリケーションは AWS 認証情報で AWS API リクエストに署名する必要があります。サービスアカウントの IAM ロール (IRSA) には、Amazon EC2 インスタンスプロファイルから Amazon EC2 インスタンスに認証情報を提供する場合と同じような方法で、アプリケーションの認証情報を管理する機能があります。AWS 認証情報を作成してコンテナに配布したり、Amazon EC2 インスタンスのロールを使用したりする必要はありません。IAM ロールを Kubernetes サービスアカウントと関連付けて、サービスアカウントを使用するように Pod を設定できます。[AWS Outposts の Amazon EKS 用ローカルクラスター](#)で、サービスアカウントに IAM ロールを使用することはできません。

サービスアカウントの IAM ロールには、次の利点があります。

- 最小特権 – IAM アクセス許可の範囲をサービスアカウントに設定すると、そのサービスアカウントを使用する Pod にのみそのアクセス許可を付与できます。また、この機能により、kiam や kube2iam などのサードパーティーのソリューションが不要になります。
- 認証情報の分離 – Pod 内のコンテナは、そのコンテナが使用するサービスアカウントに関連付けられている IAM ロールの認証情報のみを取得できます。コンテナは、他の Pod 内のコンテナで使われている認証情報にアクセスすることはできません。サービスアカウントの IAM ロールを使用する場合、Pod 内のコンテナには [Amazon EKS ノード IAM ロール](#)に割り当てられたアクセス許可も付与されます (ただし、Pod から [Amazon EC2 インスタンスメタデータサービス \(IMDS\)](#) へのアクセスをブロックしていない場合に限ります)。詳細については「[ワーカーノードに割り当てられたインスタンスプロファイルへのアクセスを制限する](#)」を参照してください。
- 監査性 – AWS CloudTrail にアクセスイベントのログ記録を利用することで、遡及的な監査を確実に行えます。

以下の手順を実行してサービスアカウントの IAM ロールを有効にします。

1. [クラスターの IAM OIDC プロバイダーを作成する](#) – この手順は、クラスターごとに 1 回だけ実行する必要があります。

 Note

EKS VPC エンドポイントを有効にすると、その VPC 内から EKS OIDC サービスエンドポイントにアクセスできなくなります。そのため、VPC で `eksctl` を使用して OIDC プロバイダーを作成するなどの操作は機能せず、<https://oidc.eks.region.amazonaws.com> をリクエストしようとするときタイムアウトになります。エラーメッセージの例は次のとおりです。

```
server cant find oidc.eks.region.amazonaws.com: NXDOMAIN
```

このステップを完了するには、VPC の外部、たとえばインターネットに接続された AWS CloudShell 内またはコンピューター上でコマンドを実行します。または、Route 53 Resolver などのスプリットホライズン条件付きリゾルバーを VPC に作成して、OIDC 発行者 URL のために別のリゾルバーを使用し、VPC DNS は使用しないようにすることもできます。CoreDNS での条件付き転送の例については、GitHub の「[Amazon EKS feature request](#)」を参照してください。

2. [Kubernetes サービスアカウントに IAM ロールを割り当てる](#) – この手順は、アプリケーションに付与する固有のアクセス許可のセットごとに実行します。
3. [Kubernetes サービスアカウントを使用するように Pod を設定する](#) – この手順は、AWS サービスへのアクセスを必要とする Pod ごとに実行します。
4. [AWS SDK で IRSA を使用する](#) – ワークロードがサポートされているバージョンの AWS SDK を使用していること、およびワークロードがデフォルトの認証情報チェーンを使用していることを確認します。

IAM、Kubernetes、OpenID Connect (OIDC) の背景情報

2014 年に、OpenID Connect (OIDC) を使用したフェデレーティッドアイデンティティのサポートが、AWS Identity and Access Management に追加されました。この機能により、サポートされている ID プロバイダーで AWS API コールを認証し、有効な OIDC JSON ウェブトークン (JWT) を受け取ることができます。このトークンを AWS STS の `AssumeRoleWithWebIdentity` API オペレー

ションに渡し、一時的な IAM ロールの認証情報を受け取ることができます。これらの認証情報を使用して、Amazon S3 や DynamoDB などの任意の AWS サービスとやり取りできます。

各 JWT トークンは署名キーペアによって署名されます。キーは Amazon EKS が管理する OIDC プロバイダーで提供され、プライベートキーは 7 日ごとにローテーションされます。Amazon EKS はパブリックキーの有効期限が切れるまで保持します。外部 OIDC クライアントに接続する場合は、パブリックキーが期限切れになる前に、キーを更新する必要があることに注意してください。[署名キーを取得して OIDC トークンを検証する](#)方法について説明します。

Kubernetes は、独自の内部 ID システムとして長い間、サービスアカウントを使用してきました。ポッドは、Kubernetes API サーバーのみが検証できる自動マウントトークン (OIDC JWT ではない) を使用して Kubernetes API サーバーで認証できます。これらのレガシーサービスアカウントトークンは期限切れにならず、署名キーの更新は難しいプロセスです。Kubernetes バージョン 1.12 では、新しい ProjectedServiceAccountToken 機能のサポートが追加されました。この機能は、サービスアカウントアイデンティティを含む OIDC JSON ウェブトークンで、設定可能なオーディエンスをサポートします。

Amazon EKS は、ProjectedServiceAccountToken JSON ウェブトークンの署名キーを含むクラスターごとにパブリック OIDC 検出エンドポイントをホストするため、IAM などの外部システムで、Kubernetes によって発行された OIDC トークンを検証して受け入れることができます。

クラスターの IAM OIDC プロバイダーを作成するには

クラスターには、[OpenID Connect](#) (OIDC) 発行者の URL が関連付けられています。サービスアカウントで AWS Identity and Access Management (IAM) ロールを使用するには、クラスターの OIDC 発行者 URL に対して IAM OIDC プロバイダーが存在する必要があります。

- 既存の Amazon EKS クラスター。デプロイするには「[はじめに](#)」を参照してください。
- ご使用のデバイスまたは AWS CloudShell で、バージョン 2.12.3 以降、または AWS コマンドラインインターフェイス (AWS CLI) のバージョン 1.27.160 以降がインストールおよび設定されていること。現在のバージョンを確認するには「`aws --version | cut -d / -f2 | cut -d ' ' -f1`」を参照してください。yum、apt-get、macOS 用の Homebrew などのパッケージマネージャーは、多くの場合 AWS CLI の最新バージョンより数バージョン古くなっています。最新バージョンをインストールするには「AWS コマンドラインインターフェイスユーザーガイド」の「[インストール](#)」および「[aws configure を使用したクイック設定](#)」を参照してください。AWS クラウドシェルにインストールされている AWS CLI バージョンも最新バージョンより数バージョン遅れることがあります。更新するには「AWS クラウドシェル ユーザーガイド」の「[ホームディレクトリへの AWS CLI のインストール](#)」を参照してください。

- デバイスまたは AWS クラウドシェル に、kubectl コマンドラインツールがインストールされていること。バージョンはご使用のクラスターの Kubernetes バージョンと同じか、1 つ前のマイナーバージョン以前、あるいはそれより新しいバージョンが使用できます。例えば、クラスターのバージョンが 1.29 である場合、kubectl のバージョン 1.28、1.29、または 1.30 が使用できます。kubectl をインストールまたはアップグレードする方法については「[the section called “kubectl と eksctl のセットアップ”](#)」を参照してください。
- クラスター構成を含む既存の kubectl config ファイル。kubectl config ファイルの作成については、「[the section called “kubectl を使用してクラスターにアクセスする”](#)」を参照してください。

eksctl または AWS Management Console を使用して、クラスターの IAM OIDC プロバイダーを作成できます。

OIDC プロバイダーの作成 (eksctl)

1. デバイスまたは AWS CloudShell にインストールされている eksctl コマンドラインツールのバージョン 0.210.0 以降。eksctl をインストールまたはアップグレードするには eksctl ドキュメントの「[インストール](#)」を参照してください。
2. クラスターの OIDC 発行者 ID を決定します。

クラスターの OIDC 発行者 ID を取得し、変数に格納します。<my-cluster> を独自の値に置き換えます。

```
cluster_name=<my-cluster>
oidc_id=$(aws eks describe-cluster --name $cluster_name --query
  "cluster.identity.oidc.issuer" --output text | cut -d '/' -f 5)
echo $oidc_id
```

3. クラスターの発行者 ID を持つ IAM OIDC プロバイダーが既にアカウントにあるかどうかを確認します。

```
aws iam list-open-id-connect-providers | grep $oidc_id | cut -d "/" -f4
```

出力が返された場合は、既にクラスター用の IAM OIDC プロバイダーが存在するため、次の手順をスキップできます。出力が返されない場合はクラスター用の IAM OIDC プロバイダーを作成する必要があります。

4. 次のコマンドを使用して、クラスターの IAM OIDC ID プロバイダーを作成します。


```
eksctl utils associate-iam-oidc-provider --cluster $cluster_name --approve
```

Note

EKS VPC エンドポイントを有効にすると、その VPC 内から EKS OIDC サービスエンドポイントにアクセスできなくなります。そのため、VPC で eksctl を使用して OIDC プロバイダーを作成するなどの操作は機能せず、タイムアウトします。エラーメッセージの例は次のとおりです。

```
** server cant find oidc.eks.<region-code>.amazonaws.com: NXDOMAIN
```

このステップを完了するには、VPC の外部、たとえばインターネットに接続された AWS CloudShell 内またはコンピューター上でコマンドを実行します。または、Route 53 Resolver などのスプリットホライズン条件付きリゾルバーを VPC に作成して、OIDC 発行者 URL のために別のリゾルバーを使用し、VPC DNS は使用しないようにすることもできます。CoreDNS での条件付き転送の例については、GitHub の「[Amazon EKS feature request](#)」を参照してください。

OIDC プロバイダーの作成 (AWS コンソール)

1. [Amazon EKS コンソール](#)を開きます。
2. 左側ペインで、[クラスター] を選択し、[クラスター] ページでご自身のクラスターの名前を選択します。
3. [概要] タブの [詳細] セクションで、[OpenID Connect プロバイダーの URL] の値を書き留めます。
4. IAM コンソール (<https://console.aws.amazon.com/iam/>) を開きます。
5. 左のナビゲーションペインの [アクセス管理] で [ID プロバイダー] を選択します。[プロバイダー] がクラスターの URL と一致するリストに表示されている場合は、クラスターのプロバイダーがすでに存在します。クラスターの URL に一致するプロバイダーがリストにない場合、プロバイダーを作成する必要があります。
6. プロバイダーを作成するには、[プロバイダーを追加] を選択します。
7. [プロバイダータイプ] に、[OpenID Connect] を選択します。
8. [プロバイダー URL] に、クラスター用の OIDC プロバイダー URL 入力します。
9. [対象者] には sts.amazonaws.com を入力します。

10.(オプション) 任意のタグ、例えば、このプロバイダーのクラスターを特定するためのタグを追加します。

11[プロバイダーを追加] をクリックします。

次のステップ: [the section called “IAM ロールを割り当てる”](#)

IAM ロールを Kubernetes サービスアカウントに割り当てる

このトピックでは、AWS Identity and Access Management (IAM) ロール を引き受けるように Kubernetes サービスアカウントを設定する方法について説明します。任意の Pod はサービスアカウントを使用するように設定すると、ロールにアクセス許可がある AWS サービスすべてにアクセスできます。

前提条件

- 既存のクラスター。まだ所有していない場合は、[はじめに](#) でのガイドのいずれかに従って作成できます。
- クラスターの既存 IAM OpenID Connect (OIDC) プロバイダー 既に所有中かどうかの確認、または作成方法については「[the section called “IAM OIDC プロバイダー”](#)」を参照してください。
- ご使用のデバイスまたは AWS CloudShell で、バージョン 2.12.3 以降、または AWS コマンドラインインターフェイス (AWS CLI) のバージョン 1.27.160 以降がインストールおよび設定されていること。現在のバージョンを確認するには「`aws --version | cut -d / -f2 | cut -d ' ' -f1`」を参照してください。yum、apt-get、macOS 用の Homebrew などのパッケージマネージャーは、多くの場合 AWS CLI の最新バージョンより数バージョン古くなっています。最新バージョンをインストールするには「AWS コマンドラインインターフェイスユーザーガイド」の「[インストール](#)」および「[aws configure を使用したクイック設定](#)」を参照してください。AWS クラウドシェル にインストールされている AWS CLI バージョンも最新バージョンより数バージョン遅れることがあります。更新するには「AWS クラウドシェル ユーザーガイド」の「[ホームディレクトリへの AWS CLI のインストール](#)」を参照してください。
- デバイスまたは AWS クラウドシェル に、kubectl コマンドラインツールがインストールされていること。バージョンはご使用のクラスターの Kubernetes バージョンと同じか、1 つ前のマイナーバージョン以前、あるいはそれより新しいバージョンが使用できます。例えば、クラスターのバージョンが 1.29 である場合、kubectl のバージョン 1.28、1.29、または 1.30 が使用できます。kubectl をインストールまたはアップグレードする方法については「[the section called “kubectl と eksctl のセットアップ”](#)」を参照してください。

- クラスター構成を含む既存の `kubectl config` ファイル。 `kubectl config` ファイルの作成については、「[the section called “kubectl を使用してクラスターにアクセスする”](#)」を参照してください。

ステップ 1: IAM ポリシーの作成

既存の IAM ポリシーを IAM ロールに関連付ける場合は、次のステップにスキップします。

1. IAM ポリシーを作成します。ポリシーを自作することも、必要となるアクセス権限のいくつかは既に付与されている AWS 管理ポリシーをコピーし、特定の要件に応じてカスタマイズすることもできます。詳細については IAM ユーザーガイド の「[IAM ポリシーの作成](#)」を参照してください。
2. Pod にアクセスさせる AWS サービスの権限を含むファイルを作成します。すべての AWS サービスに対するアクションの全リストについては、「[サービス認可リファレンス](#)」を参照してください。

次のコマンドを実行して、Amazon S3 バケットへの読み取り専用アクセスを許可するサンプルポリシーファイルを作成できます。必要に応じて、このバケットに設定情報またはブートストラップスクリプトを格納すると、Pod 内のコンテナがバケットからファイルを読み取り、アプリケーションにロードできます。このサンプルポリシーを作成する場合は、次のコンテンツをデバイスにコピーします。*my-pod-secrets-bucket* をバケット名に置き換え、コマンドを実行します。

```
cat >my-policy.json <<EOF
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "s3:GetObject",
      "Resource": "arn:aws:s3:::my-pod-secrets-bucket"
    }
  ]
}
EOF
```

3. IAM ポリシーを作成します。

```
aws iam create-policy --policy-name my-policy --policy-document file://my-policy.json
```

ステップ 2: IAM ロールを作成して関連付ける

IAM ロールを作成し、Kubernetes サービスアカウントに関連付けます。eksctl または AWS CLI を使用できます。

ロールの作成と関連付け (eksctl)

デバイスまたは AWS CloudShell にインストールされている eksctl コマンドラインツールのバージョン 0.210.0 以降。eksctl をインストールまたはアップグレードするにはeksctl ドキュメントの「[インストール](#)」を参照してください。

my-service-account を、eksctl で作成する Kubernetes サービスアカウントの名前に置き換え、IAM ロールに関連付けます。*default* は、eksctl でサービスアカウントを作成する名前空間に置き換えます。*#####* の部分は自分のクラスター名に置き換えます。*my-role* は、サービスアカウントに関連付けるロールの名前に置き換えます。まだ存在しない場合は、eksctl によって作成されます。*111122223333* をアカウント ID に置き換え、*my-policy* を既存のポリシー名に置き換えます。

```
eksctl create iamserviceaccount --name my-service-account --namespace default --cluster my-cluster --role-name my-role \
    --attach-policy-arn arn:aws:iam::111122223333:policy/my-policy --approve
```

⚠ Important

ロールまたはサービスアカウントが既に存在する場合、前のコマンドは失敗する可能性があります。eksctl には、そのような状況で利用できるさまざまなオプションがあります。詳細については、eksctl create iamserviceaccount --help を実行してください。

ロールの作成と関連付け (AWS CLI)

IAM ロールを引き受ける既存の Kubernetes サービスアカウントがある場合は、この手順を省略できます。

1. Kubernetes サービスアカウントを作成します。次のコンテンツをデバイスにコピーします。*my-service-account* を目的の名前に置き換え、必要に応じて *default* を別の名前空間に置き換えます。*default* を変更する場合、名前空間は既に存在している必要があります。

```
cat >my-service-account.yaml <<EOF
```

```
apiVersion: v1
kind: ServiceAccount
metadata:
  name: my-service-account
  namespace: default
EOF
kubectl apply -f my-service-account.yaml
```

2. 次のコマンドを使用して、AWS アカウント ID を環境変数に設定します。

```
account_id=$(aws sts get-caller-identity --query "Account" --output text)
```

3. 次のコマンドを使用して、クラスターの OIDC ID プロバイダーを環境変数に設定します。####
の部分は自分のクラスター名に置き換えます。

```
oidc_provider=$(aws eks describe-cluster --name my-cluster --region $AWS_REGION --  
query "cluster.identity.oidc.issuer" --output text | sed -e "s/^https:\\/\\/\\/" )
```

4. サービスアカウントの名前空間と名前の変数を設定します。*my-service-account* を、ロールを引き受けさせる Kubernetes サービスアカウントに置き換えます。*default* は、サービスアカウントの名前空間に置き換えます。

```
export namespace=default
export service_account=my-service-account
```

5. IAM ロール用の信頼ポリシーファイルを作成するには、次のコマンドを実行します。名前空間内のすべてのサービスアカウントにロールの使用を許可する場合は、次の内容をデバイスにコピーします。*StringEquals* を *StringLike* に置き換え、*\$service_account* を * に置き換えます。以下の *StringEquals* または *StringLike* 条件に複数のエントリを追加して、複数のサービスアカウントまたは名前空間がロールを引き受けられるようにできます。クラスターが属するアカウントとは異なる AWS アカウントのロールがロールを引き受けられるようにする方法については、「[the section called “クロスアカウント IAM”](#)」を参照してください。

```
cat >trust-relationship.json <<EOF
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Federated": "arn:aws:iam::$account_id:oidc-provider/$oidc_provider"
      }
    }
  ]
}
```

```

    },
    "Action": "sts:AssumeRoleWithWebIdentity",
    "Condition": {
      "StringEquals": {
        "$oidc_provider:aud": "sts.amazonaws.com",
        "$oidc_provider:sub": "system:serviceaccount:$namespace:$service_account"
      }
    }
  }
]
}
EOF

```

6. ロールを作成します。*my-role* を IAM ロールの名前に置き換え、*my-role-description* をロールの説明に置き換えます。

```
aws iam create-role --role-name my-role --assume-role-policy-document file:///trust-relationship.json --description "my-role-description"
```

7. IAM ポリシーをロールにアタッチします。*my-role* を IAM ロールの名前に置き換え、*my-policy* を、作成した既存のポリシーの名前に置き換えます。

```
aws iam attach-role-policy --role-name my-role --policy-arn=arn:aws:iam::
$account_id:policy/my-policy
```

8. サービスアカウントに、サービスアカウントで引き受ける IAM ロールの Amazon リソースネーム (ARN) の注釈を付けます。*my-role* を既存の IAM ロールの名前に置き換えます。前の手順で、クラスターが属するアカウントとは異なる AWS アカウントのロールに、ロールの引き受けを許可したと仮定します。その場合、AWS アカウントおよび他のアカウントからのロールを必ず指定してください。詳細については、「[the section called “クロスアカウント IAM”](#)」を参照してください。

```
kubectl annotate serviceaccount -n $namespace $service_account eks.amazonaws.com/
role-arn=arn:aws:iam::$account_id:role/my-role
```

9. (オプション) [サービスアカウントの AWS Security Token Service エンドポイントを設定します](#)。AWS では、グローバルエンドポイントの代わりにリージョン AWS STS エンドポイントを使用することをお勧めしています。これにより、レイテンシーが減少し、組み込みの冗長性が提供され、セッショントークンの有効性が向上します。

ステップ 3: 設定を確認する

1. IAM ロールの信頼ポリシーが正しく設定されていることを確認します。

```
aws iam get-role --role-name my-role --query Role.AssumeRolePolicyDocument
```

出力例は次のとおりです。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Federated": "arn:aws:iam::111122223333:oidc-provider/
oidc.eks.region-code.amazonaws.com/id/EXAMPLED539D4633E53DE1B71EXAMPLE"
      },
      "Action": "sts:AssumeRoleWithWebIdentity",
      "Condition": {
        "StringEquals": {
          "oidc.eks.region-code.amazonaws.com/id/
EXAMPLED539D4633E53DE1B71EXAMPLE:sub": "system:serviceaccount:default:my-service-
account",
          "oidc.eks.region-code.amazonaws.com/id/
EXAMPLED539D4633E53DE1B71EXAMPLE:aud": "sts.amazonaws.com"
        }
      }
    }
  ]
}
```

2. 前の手順でロールにアタッチしたポリシーが、そのロールにアタッチされていることを確認します。

```
aws iam list-attached-role-policies --role-name my-role --query
"AttachedPolicies[].PolicyArn" --output text
```

出力例は次のとおりです。

```
arn:aws:iam::111122223333:policy/my-policy
```

3. 使用するポリシーの Amazon リソースネーム (ARN) を保存する変数を設定します。*my-policy* を、アクセス許可を確認するポリシーの名前に置き換えます。

```
export policy_arn=arn:aws:iam::111122223333:policy/my-policy
```

4. ポリシーのデフォルトバージョンを確認します。

```
aws iam get-policy --policy-arn $policy_arn
```

出力例は次のとおりです。

```
{
  "Policy": {
    "PolicyName": "my-policy",
    "PolicyId": "EXAMPLEBIOWGLDEXAMPLE",
    "Arn": "arn:aws:iam::111122223333:policy/my-policy",
    "Path": "/",
    "DefaultVersionId": "v1",
    [...]
  }
}
```

5. ポリシーの内容を表示して、Pod で必要な権限がすべて含まれていることを確認します。必要であれば、次のコマンドの **1** を、前の出力で返されたバージョンに置き換えます。

```
aws iam get-policy-version --policy-arn $policy_arn --version-id v1
```

出力例は次のとおりです。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "s3:GetObject",
      "Resource": "arn:aws:s3:::my-pod-secrets-bucket"
    }
  ]
}
```

前の手順でサンプルポリシーを作成した場合、出力は同じになります。別のポリシーを作成した場合、####の内容は異なります。

6. Kubernetes サービスアカウントにロールが注釈されていることを確認します。

```
kubectl describe serviceaccount my-service-account -n default
```

出力例は次のとおりです。

```
Name:                my-service-account
Namespace:           default
Annotations:         eks.amazonaws.com/role-arn: arn:aws:iam::111122223333:role/my-
                    role
Image pull secrets:   <none>
Mountable secrets:    my-service-account-token-qqjfl
Tokens:              my-service-account-token-qqjfl
[...]
```

次のステップ

- [the section called “Pod への割り当て”](#)

Kubernetes サービスアカウントを使用するように Pod を設定するには

Pod が AWS サービスにアクセスする必要がある場合は、Kubernetes サービスアカウントを使用するように設定する必要があります。サービスアカウントは、AWS サービスにアクセスする権限がある AWS Identity and Access Management (IAM) ロールに関連付ける必要があります。

- 既存のクラスター。まだ所有していない場合は、[はじめに](#) でのガイドのいずれかを参照しながら作成できます。
- クラスターの既存 IAM OpenID Connect (OIDC) プロバイダー 既に所有中かどうかの確認、または作成方法については「[the section called “IAM OIDC プロバイダー”](#)」を参照してください。
- IAM ロールに関連付けられている既存の Kubernetes サービスアカウント。サービスアカウントには、IAM ロールの Amazon リソースネーム (ARN) の注釈を付ける必要があります。ロールには、Pod が AWS サービスを使用するアクセス許可を含む IAM ポリシーが関連付けられている必要があります。サービスアカウントとロールの作成および設定方法については、「[the section called “IAM ロールを割り当てる”](#)」を参照してください。

- ご使用のデバイスまたは AWS CloudShell で、バージョン 2.12.3 以降、または AWS コマンドラインインターフェイス (AWS CLI) のバージョン 1.27.160 以降がインストールおよび設定されていること。現在のバージョンを確認するには「`aws --version | cut -d / -f2 | cut -d ' ' -f1`」を参照してください。yum、apt-get、macOS 用の Homebrew などのパッケージマネージャーは、多くの場合 AWS CLI の最新バージョンより数バージョン古くなっています。最新バージョンをインストールするには「AWS コマンドラインインターフェイスユーザーガイド」の「[インストール](#)」および「[aws configure を使用したクイック設定](#)」を参照してください。AWS クラウドシェル にインストールされている AWS CLI バージョンも最新バージョンより数バージョン遅れることがあります。更新するには「AWS クラウドシェル ユーザーガイド」の「[ホームディレクトリへの AWS CLI のインストール](#)」を参照してください。
 - デバイスまたは AWS クラウドシェル に、kubectl コマンドラインツールがインストールされていること。バージョンはご使用のクラスターの Kubernetes バージョンと同じか、1 つ前のマイナーバージョン以前、あるいはそれより新しいバージョンが使用できます。例えば、クラスターのバージョンが 1.29 である場合、kubectl のバージョン 1.28、1.29、または 1.30 が使用できます。kubectl をインストールまたはアップグレードする方法については「[the section called “kubectl と eksctl のセットアップ”](#)」を参照してください。
 - クラスター構成を含む既存の kubectl config ファイル。kubectl config ファイルの作成については、「[the section called “kubectl を使用してクラスターにアクセスする”](#)」を参照してください。
1. 次コマンドを使用して、Pod をデプロイして設定を確認できるデプロイマニフェストを作成します。*example* の値は独自の値に置き換えます。

```
cat >my-deployment.yaml <<EOF
apiVersion: apps/v1
kind: Deployment
metadata:
  name: my-app
spec:
  selector:
    matchLabels:
      app: my-app
  template:
    metadata:
      labels:
        app: my-app
    spec:
      serviceAccountName: my-service-account
      containers:
      - name: my-app
```

```
image: public.ecr.aws/nginx/nginx:X.XX
EOF
```

2. マニフェストをクラスターにデプロイします。

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

3. Pod に必要な環境変数が存在することを確認してください。

a. 前の手順のデプロイ時にデプロイされた Pod を確認します。

```
kubectl get pods | grep my-app
```

出力例は次のとおりです。

```
my-app-6f4dfff6cb-76cv9    1/1    Running    0    3m28s
```

b. Pod が使用している IAM ロールの ARN を確認します。

```
kubectl describe pod my-app-6f4dfff6cb-76cv9 | grep AWS_ROLE_ARN:
```

出力例は次のとおりです。

```
AWS_ROLE_ARN:                arn:aws:iam::111122223333:role/my-role
```

ロール ARN は、既存のサービスアカウントに注釈を付けたロール ARN と一致する必要があります。サービスアカウントへの注釈付けの詳細については、「[the section called “IAM ロールを割り当てる”](#)」を参照してください。

c. Pod にウェブ ID トークンファイルのマウントがあることを確認します。

```
kubectl describe pod my-app-6f4dfff6cb-76cv9 | grep AWS_WEB_IDENTITY_TOKEN_FILE:
```

出力例は次のとおりです。

```
AWS_WEB_IDENTITY_TOKEN_FILE: /var/run/secrets/eks.amazonaws.com/serviceaccount/
token
```

kubelet が Pod に代わってトークンをリクエストして格納します。デフォルトで kubelet は、トークンが合計有効期限の 80% を超えている場合、またはトークンが 24 時間を超

えている場合、そのトークンを更新します。Pod 仕様の設定を使用して、デフォルトのサービスアカウントを除くすべてのアカウントの有効期限を変更できます。詳細については、Kubernetes ドキュメントの「[Service Account Token Volume Projection \(サービスアカウントトークンボリュームのプロジェクト\)](#)」を参照してください。

クラスターの [Amazon EKS Pod Identity ウェブフック](#) は、次の注釈が付いたサービスアカウントを使用している Pod をモニタリングします。

```
eks.amazonaws.com/role-arn: arn:aws:iam::111122223333:role/my-role
```

このウェブフックは、そうした Pod に以前の環境変数を適用します。クラスターでは、環境変数とトークンファイルのマウントを設定するために、ウェブフックを使用する必要はありません。これらの環境変数を保持するよう、Pod を手動で設定できます。[AWS SDK のサポートされているバージョン](#) は、最初に認証情報チェーンプロバイダーでこれらの環境変数を探します。ロールの認証情報は、この条件を満たす Pod で使用されます。

4. ロールにアタッチされた IAM ポリシーで割り当てたアクセス許可を使用して、Pod が AWS サービスとやり取りできることを確認します。

 Note

サービスアカウントに関連付けられた IAM ロールの AWS 認証情報を Pod が使用する場合、AWS CLI またはその Pod のコンテナ内にある他の SDK は、そのロールから提供される認証情報を使用します。[Amazon EKS ノードの IAM ロール](#) に提供された認証情報へのアクセスを制限しない場合、Pod は引き続きこれらの認証情報にアクセスできます。詳細については「[ワーカーノードに割り当てられたインスタンスプロフィールへのアクセスを制限する](#)」を参照してください。

Pod が想定通りにサービスとやり取りできない場合は、次の手順を実行して、すべてが正しく設定されていることを確認してください。

- a. OpenID Connect ウェブアイデンティティトークンファイルを介した IAM ロールの引き受けをサポートする AWS SDK バージョンを Pod が使用していることを確認します。詳細については、「[the section called “サポートされている SDK”](#)」を参照してください。
- b. デプロイがサービスアカウントを使用していることを確認します。

```
kubectl describe deployment my-app | grep "Service Account"
```

出力例は次のとおりです。

```
Service Account:  my-service-account
```

- c. それでも Pod がサービスにアクセスできない場合は、「[Kubernetes サービスアカウントに IAM ロールを割り当てる](#)」で説明されている[手順](#)を参照して、ロールとサービスアカウントが正しく設定されていることを確認します。

サービスアカウントの AWS Security Token Service エンドポイントを設定する

[サービスアカウントの IAM ロール](#)で Kubernetes サービスアカウントを使用している場合、クラスターとプラットフォームバージョンが同じであるか、次の表にリストされているものより後であれば、サービスアカウントで使用される AWS Security Token Service エンドポイントのタイプを設定できます。Kubernetes またはプラットフォームのバージョンが表に記載されているバージョンよりも前の場合、サービスアカウントはグローバルエンドポイントのみを使用できます。

Kubernetes バージョン	プラットフォームバージョン	デフォルトのエンドポイントタイプ
1.31	eks.4	リージョン別
1.30	eks.2	リージョン別
1.29	eks.1	リージョン別
1.28	eks.1	リージョン別
1.27	eks.1	リージョン別
1.26	eks.1	リージョン別

AWS では、グローバルエンドポイントの代わりに地域の AWS STS エンドポイントを使用することを推奨しています。これにより、レイテンシーが減少し、組み込みの冗長性が提供され、セッショントークンの有効性が向上します。Pod が実行中の AWS リージョンで、AWS Security Token Service がアクティブであること。さらに、AWS リージョン内のサービスに障害が発生した場合に別の AWS リージョンを使用できるよう、アプリケーションに冗長性が組み込まれている必要があります。

ます。詳細については、IAM ユーザーガイドの「[AWS リージョンでの AWS STS の管理](#)」を参照してください。

- 既存のクラスター。まだ所有していない場合は、[はじめに](#) でのガイドのいずれかを参照しながら作成できます。
- クラスター用の既存の IAM OIDC プロバイダー。詳細については、「[the section called “IAM OIDC プロバイダー”](#)」を参照してください。
- [サービスアカウント用の Amazon EKS IAM](#) 機能を使用するように設定された、既存の Kubernetes サービスアカウント。

次の例ではすべて、[Amazon VPC CNI プラグイン](#) で使用される aws-node Kubernetes サービスアカウントを使用しています。*example values* は、ご自身のサービスアカウント、ポッド、名前空間、その他のリソースに置き換えることができます。

1. エンドポイントを変更したいサービスアカウントを使用する Pod を選択します。Pod を実行する AWS リージョンを決定します。*aws-node-6mfgv* を Pod 名に置き換え、*kube-system* をポッドの名前空間に置き換えます。

```
kubectl describe pod aws-node-6mfgv -n kube-system |grep Node:
```

出力例は次のとおりです。

```
ip-192-168-79-166.us-west-2/192.168.79.166
```

前の出力では、Pod は us-west-2 AWS リージョンのノードで実行されています。

2. Pod のサービスアカウントが使用しているエンドポイントタイプを確認します。

```
kubectl describe pod aws-node-6mfgv -n kube-system |grep AWS_STS_REGIONAL_ENDPOINTS
```

出力例は次のとおりです。

```
AWS_STS_REGIONAL_ENDPOINTS: regional
```

現在のエンドポイントがグローバルの場合、この出力には *global* が返されます。出力が返されない場合、デフォルトのエンドポイントタイプが、上書きされないまま使用されています。

3. クラスターまたはプラットフォームのバージョンが表に示されているバージョンと同じかそれ以降の場合は、次のコマンドのいずれかを使用して、サービスアカウントで使われるエンドポイントタイプをデフォルトタイプから別のタイプに変更できます。`aws-node` は、サービスアカウントの名前に置き換え、`kube-system` は、サービスアカウントの名前空間に置き換えます。
- デフォルトまたは現在のエンドポイントタイプがグローバルであり、それをリージョン別に変更する場合は、以下の手順を実行します。

```
kubectl annotate serviceaccount -n kube-system aws-node eks.amazonaws.com/sts-regional-endpoints=true
```

Pod のコンテナで実行されているアプリケーションで、事前署名された S3 URL を生成するために[サービスアカウントの IAM ロール](#)を使用している場合、リージョン別エンドポイントの URL の形式は、次の例のようになります。

```
https://bucket.s3.us-west-2.amazonaws.com/path?...&X-Amz-Credential=your-access-key-id/date/us-west-2/s3/aws4_request&...
```

- デフォルトまたは現在のエンドポイントタイプがリージョン別であり、それをグローバルに変更する場合は、次の手順を実行します。

```
kubectl annotate serviceaccount -n kube-system aws-node eks.amazonaws.com/sts-regional-endpoints=false
```

アプリケーションが、AWS STS のグローバルエンドポイントに対し明示的にリクエストを行っており、Amazon EKS クラスターにおいて、リージョン別エンドポイントを使用する際のデフォルトの動作がオーバーライドされていない場合は、リクエストが失敗しエラーが返されます。詳細については、「[the section called “ポッドコンテナは次のエラーを受け取ります: An error occurred \(SignatureDoesNotMatch\) when calling the GetCallerIdentity operation: Credential should be scoped to a valid region”](#)」を参照してください。

Pod のコンテナで実行されているアプリケーションで、事前署名された S3 URL を生成するために[サービスアカウントの IAM ロール](#)を使用している場合、グローバルエンドポイントの URL の形式は、次の例のようになります。

```
https://bucket.s3.amazonaws.com/path?...&X-Amz-Credential=your-access-key-id/date/us-west-2/s3/aws4_request&...
```

使用している自動化処理で、事前署名付き URL に特定の形式を想定している場合、または事前署名付き URL を使用するアプリケーションやダウンストリームの依存関係に、ターゲットとして想定する AWS リージョンがある場合、適切な AWS STS エンドポイントを使用するために必要な変更を加えます。

4. 認証情報環境変数を適用するために、サービスアカウントに関連付けられている既存の Pod を削除して再作成します。変更するウェブフックは、既に実行されている Pod には適用されません *Pod*、*kube-system*、および *-l k8s-app=aws-node* を、アノテーションを設定する Pod の情報に置き換えることができます。

```
kubectl delete Pods -n kube-system -l k8s-app=aws-node
```

5. Pod がすべて再起動したことを確認します。

```
kubectl get Pods -n kube-system -l k8s-app=aws-node
```

6. Pod のいずれかの環境変数を表示します。AWS_STS_REGIONAL_ENDPOINTS の値が、以前のステップで設定した値であることを確認します。

```
kubectl describe pod aws-node-kzbtr -n kube-system |grep AWS_STS_REGIONAL_ENDPOINTS
```

出力例は次のとおりです。

```
AWS_STS_REGIONAL_ENDPOINTS=regional
```

IRSA で別のアカウントに対して認証する

別のアカウントのクラスターから ID プロバイダーを作成するか、連鎖した AssumeRole オペレーションを使用することで、クロスアカウントの IAM アクセス許可を設定できます。次の例では、アカウント A は、サービスアカウントの IAM ロールをサポートする Amazon EKS クラスターを所有しています。そのクラスターで実行されている Pod は、アカウント B の IAM アクセス許可を引き受ける必要があります。

Example 別のアカウントのクラスターから ID プロバイダーを作成する

Example

この例では、アカウント A はアカウント B にクラスターからの OpenID Connect (OIDC) 発行者 URL を提供します。アカウント B は、アカウント A のクラスターからの OIDC 発行者 URL を使用

して、「[クラスターの IAM OIDC プロバイダーを作成する](#)」および「[the section called “IAM ロールを割り当てる”](#)」の手順に従います。次に、クラスター管理者は、アカウント A のクラスターのサービスアカウントに、アカウント B (**444455556666**) のロールを使用するように注釈を付けます。

```
apiVersion: v1
kind: ServiceAccount
metadata:
  annotations:
    eks.amazonaws.com/role-arn: arn:aws:iam::444455556666:role/account-b-role
```

Example 連鎖された **AssumeRole** オペレーションを使用する

Example

この例では、アカウント B は、アカウント A のクラスターの Pod に付与するためのアクセス許可を持つ、IAM ポリシーを作成しています。アカウント B (**444455556666**) は、アカウント A (**111122223333**) への **AssumeRole** 許可を付与する信頼関係を持つ IAM ロールにそのポリシーをアタッチします。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "AWS": "arn:aws:iam::111122223333:root"
      },
      "Action": "sts:AssumeRole",
      "Condition": {}
    }
  ]
}
```

アカウント A は、クラスターの OIDC 発行者アドレスで作成された ID プロバイダーから認証情報を取得する信頼ポリシーを持つロールを作成します。

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



```

    "Federated": "arn:aws:iam::111122223333:oidc-provider/oidc.eks.region-
code.amazonaws.com/id/EXAMPLED539D4633E53DE1B71EXAMPLE"
  },
  "Action": "sts:AssumeRoleWithWebIdentity"
}
]
}

```

アカウント A は、アカウント B が作成したロールを引き受けるための以下のアクセス許可を持つポリシーをそのロールにアタッチします。

```

{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "sts:AssumeRole",
      "Resource": "arn:aws:iam::444455556666:role/account-b-role"
    }
  ]
}

```

アカウント B のロールを引き受ける Pod のアプリケーションコードは、account_b_role と account_a_role の 2 つのプロファイルを使用します。account_b_role プロファイルでは、ソースとして account_a_role プロファイルを使用します。AWS CLI では、~/.aws/config ファイルは以下のようになります。

```

[profile account_b_role]
source_profile = account_a_role
role_arn=arn:aws:iam::444455556666:role/account-b-role

[profile account_a_role]
web_identity_token_file = /var/run/secrets/eks.amazonaws.com/serviceaccount/token
role_arn=arn:aws:iam::111122223333:role/account-a-role

```

他の AWS SDK の連鎖されたプロファイルを指定するには、使用する SDK のドキュメントを参照してください。詳細については、「[AWS で構築するツール](#)」を参照してください。

AWS SDK で IRSA を使用する

認証情報の使用

サービスアカウントの IAM ロール (IRSA) の認証情報を使用するために、コードで任意の AWS SDK を使用して SDK を含む AWS サービスのクライアントを作成できます。デフォルトでは、SDK は使用する AWS Identity and Access Management 認証情報を一連の場所で検索します。クライアントの作成時や SDK の初期化時に認証情報プロバイダーを指定しなかった場合は、サービスアカウントの認証情報用の IAM ロールが使用されます。

これがうまくいくのは、サービスアカウント用の IAM ロールがデフォルトの認証情報チェーンのステップとして追加されたからです。現在、ワークロードが認証情報チェーンの前にある認証情報を使用している場合は、同じワークロードのサービスアカウントに IAM ロールを設定しても、その認証情報は引き続き使用されます。

SDK は、AssumeRoleWithWebIdentity アクションを使用してサービスアカウント OIDC トークンを AWS Security Token Service からの一時的な認証情報と自動的に交換します。Amazon EKS とこの SDK アクションは、有効期限が切れる前に一時認証情報を更新することで、引き続きローテーションを行います。

[サービスアカウントの IAM ロール](#)を使用すると、Pod 内のコンテナは、OpenID Connect ウェブ ID トークンファイルを介した IAM ロールの引き受けをサポートする AWS SDK バージョンを使用する必要があります。お使いの AWS SDK には、必ず次のバージョン以降を使用してください。

- Java (バージョン 2) – [2.10.11](#)
- Java – [1.11.704](#)
- Go – [1.23.13](#)
- Python (Boto3) – [1.9.220](#)
- Python (botocore) – [1.12.200](#)
- AWS CLI – [1.16.232](#)
- ノード – [2.525.0](#) と [3.27.0](#)
- Ruby – [3.58.0](#)
- C++ – [1.7.174](#)
- .NET – [3.3.659.1](#) - `AWSSDK.SecurityToken` も含めることを確認します。
- PHP – [3.110.7](#)

[Cluster Autoscaler](#)、[AWS Load Balancer Controller](#) を使用してインターネットトラフィックをルーティングする、[Amazon VPC CNI plugin for Kubernetes](#) など、一般的な Kubernetes アドオンの多くは、サービスアカウントの IAM ロールをサポートしています。

サポートされている SDK を使用していることを確認するには、コンテナを構築する際に、「[AWS での構築ツール](#)」で、希望する SDK のインストール手順に従ってください。

考慮事項

Java

Java を使用する場合は、クラスパスに sts を含める必要があります。詳細については、Java SDK ドキュメントの「[WebIdentityTokenFileCredentialsProvider](#)」を参照してください。

OIDC トークンを検証するための署名キーを取得する

Kubernetes は、ProjectedServiceAccountToken を各 Kubernetes サービスアカウントに発行します。このトークンは OIDC トークンであり、さらに JSON ウェブトークン (JWT) の一種でもあります。Amazon EKS は、外部システムでトークンを検証できるように、トークンの署名キーを含むクラスターごとにパブリック OIDC エンドポイントをホストします。

ProjectedServiceAccountToken を検証するには、JSON ウェブキーセット (JWKS) と呼ばれる OIDC パブリック署名キーを取得する必要があります。アプリケーションでこれらのキーを使用してトークンを検証します。例えば、[PyJWT Python ライブラリ](#)を使用して、これらのキーを使用してトークンを検証できます。ProjectedServiceAccountToken の詳細については、「[the section called “IAM、Kubernetes、OpenID Connect \(OIDC\) の背景情報”](#)」を参照してください。

前提条件

- クラスターの既存の AWS Identity and Access Management (IAM) OpenID Connect (OIDC) プロバイダー。既に存在しているかどうかを確認する、または作成するには「[the section called “IAM OIDC プロバイダー”](#)」を参照してください。
- AWS CLI – Amazon EKS など AWS のサービスを実行するためのコマンドラインツールです。詳細については、AWS コマンドラインインターフェイスユーザーガイドの「[インストール](#)」を参照してください。AWS CLI のインストール後は設定も行っておくことをお勧めします。詳細については、AWS コマンドラインインターフェイスユーザーガイドの「[aws configure を使用したクイック設定](#)」を参照してください。

手順

1. AWS CLI を使用して、Amazon EKS クラスターの OIDC URL を取得します。

```
$ aws eks describe-cluster --name my-cluster --query 'cluster.identity.oidc.issuer'
"https://oidc.eks.us-west-2.amazonaws.com/id/8EBDXXXX00BAE"
```

2. curl または同様のツールを使用して、パブリック署名キーを取得します。結果は [JSON ウェブキーセット \(JWKS\)](#) です。

 Important

Amazon EKS は、OIDC エンドポイントへの呼び出しをスロットリングします。パブリック署名キーをキャッシュする必要があります。レスポンスに含まれる cache-control ヘッダーを考慮します。

 Important

Amazon EKS は、OIDC 署名キーを 7 日ごとにローテーションします。

```
$ curl https://oidc.eks.us-west-2.amazonaws.com/id/8EBDXXXX00BAE/keys
{"keys":
[{"kty":"RSA","kid":"2284XXXX4a40","use":"sig","alg":"RS256","n":"wk1bXXXXMVfQ","e":"AQAB"}]}
```

EKS Pod Identity が Pod に AWS サービスへのアクセス権を付与する仕組みを学ぶ

ポッドのコンテナ内のアプリケーションは AWS SDK または AWS CLI で、AWS Identity and Access Management (IAM) アクセス許可を使用した AWS サービスへの API リクエストを行うことができます。アプリケーションは AWS 認証情報で AWS API リクエストに署名する必要があります。

EKS Pod Identity は、Amazon EC2 インスタンスプロファイルから Amazon EC2 インスタンスに認証情報を提供する場合と同じような方法で、アプリケーションの認証情報を管理する機能があります。AWS 認証情報を作成してコンテナに配布したり、Amazon EC2 インスタンスのロールを使用したりする必要はありません。IAM ロールを Kubernetes サービスアカウントと関連付けて、サービスアカウントを使用するように Pod を設定できます。

各 EKS Pod Identity の関連付けは、指定されたクラスター内のネームスペース内のサービスアカウントにロールをマップします。複数のクラスターに同じアプリケーションがある場合、ロールの信頼ポリシーを変更することなく、各クラスターで同一の関連付けを行うことができます。

ポッドが関連付けられたサービスアカウントを使用する場合、Amazon EKS はポッドのコンテナに環境変数を設定します。環境変数は、AWS CLI を含む AWS SDK が EKS Pod Identity の認証情報を使用するように設定します。

EKS Pod Identity の利点

EKS Pod Identity には次の利点があります。

- 最小特権 – IAM アクセス許可の範囲をサービスアカウントに設定すると、そのサービスアカウントを使用する Pod にのみそのアクセス許可を付与できます。また、この機能により、kiam や kube2iam などのサードパーティーのソリューションが不要になります。
- 認証情報の分離 – Pod 内のコンテナは、そのコンテナが使用するサービスアカウントに関連付けられている IAM ロールの認証情報のみを取得できます。コンテナは、他の Pod 内のコンテナで使われている認証情報にアクセスすることはできません。Pod Identity を使用する場合、Pod 内のコンテナには [the section called “ノードの IAM ロール”](#) に割り当てられたアクセス許可も付与されます (ただし、Pod から [Amazon EC2 インスタンスメタデータサービス \(IMDS\)](#) へのアクセスをブロックしていない場合)。詳細については「[ワーカーノードに割り当てられたインスタンスプロファイルへのアクセスを制限する](#)」を参照してください。
- 監査性 – 遡及的な監査を確実に行うため、AWS CloudTrail を介してアクセスとイベントのロギングを利用できます。

EKS Pod Identity は、OIDC ID プロバイダーを使用しないため、[the section called “IRSA による認証情報”](#) よりも簡単な方法です。EKS Pod Identity には次の機能強化があります。

- 独立オペレーション – 多くの組織では、OIDC ID プロバイダーの作成を Kubernetes クラスターの管理とは別のチームが担当しています。EKS Pod Identity には明確な役割分担があり、EKS Pod Identity の関連付けの設定はすべて Amazon EKS で行われ、IAM アクセス許可の設定はすべて IAM で行われます。
- 再利用性 – EKS Pod Identity は、サービスアカウントの IAM ロールが使用するクラスターごとの個別のプリンシパルの代わりに、単一の IAM プリンシパルを使用します。IAM 管理者は以下のプリンシパルを任意のロールの信頼ポリシーに追加して、EKS Pod Identity で使用できるようにします。

```
"Principal": {
  "Service": "pods.eks.amazonaws.com"
}
```

- スケーラビリティ – 一時的な認証情報の各セットは、各 Pod で実行する各 AWS SDK ではなく、EKS Pod Identity 内の EKS Auth サービスによって引き継がれます。次に、各ノードで実行される Amazon EKS Pod Identity エージェントが SDK に認証情報を発行します。そのため、負荷はノードごとに 1 回に減り、各ポッドで重複することはありません。詳細については、「[the section called “仕組み”](#)」を参照してください。

この 2 つの選択肢を比較する詳細については、「[the section called “AWS へのワークロードアクセス”](#)」を参照してください。

EKS Pod Identity のセットアップの概要

以下の手順を実行して EKS Pod Identity を有効にします。

1. [the section called “エージェントのセットアップ”](#) — この手順は、クラスターごとに 1 回だけ実行します。クラスターで EKS 自動モードが有効になっている場合は、このステップを完了する必要はありません。
2. [the section called “IAM ロールを割り当てる”](#) — この手順は、アプリケーションに付与する固有の権限セットごとに実行します。
3. [the section called “Pod サービスアカウント”](#) — この手順を AWS サービスへのアクセスが必要な Pod ごとに実行します。
4. [the section called “サポートされている SDK”](#) — ワークロードがサポートされているバージョンの AWS SDK を使用していること、およびワークロードがデフォルトの認証情報チェーンを使用していることを確認します。

EKS Pod Identity に関する考慮事項

- 各クラスター内の Kubernetes サービスアカウントごとに 1 つの IAM ロールを関連付けることができます。EKS Pod Identity の関連付けを編集することで、サービスアカウントにマップされるロールを変更できます。
- 関連付けることができるのはクラスターと同じ AWS アカウントにあるロールだけです。別のアカウントから、EKS Pod Identity が使用するよう設定したこのアカウントのロールに、アクセスを委任できます。アクセス権の委任および AssumeRole については、「IAM ユーザーガイド」の「[AWS アカウント間の IAM ロールを使用したアクセスの委任](#)」を参照してください。
- EKS Pod Identity エージェントが必要です。ノード上で Kubernetes DaemonSet として動作し、その動作するノード上の Pod にのみ認証情報を提供します。EKS Pod Identity エージェントの互

換性の詳細については、以下のセクション [the section called “EKS Pod Identity の制限事項”](#) を参照してください。

- AWS IAM 動作と同様に、EKS Pod Identity の関連付けは最終的には一貫性があり、最初の API コールが正常に戻ってから有効になるまでに数秒かかる場合があります。発生する可能性のあるこれらの遅延を考慮して、アプリケーションを設計する必要があります。アプリケーションの重要な高可用性のコードパスには、Pod Identity 関連付けの作成/更新を含めないことをお勧めします。代わりに、実行頻度が低い別の初期化またはセットアップルーチンに の変更を加えます。
- Pod Identity Agent とともにポッドのセキュリティグループを使用している場合は、AWS VPC CNI のために `POD_SECURITY_GROUP_ENFORCING_MODE` フラグを設定する必要がある場合があります。ポッドのセキュリティグループに関する考慮事項の詳細については、「[the section called “ポッドのセキュリティグループ”](#)」を参照してください。
- EKS Pod Identity エージェント はノードの `hostNetwork` を使用し、ノード上のリンクローカルアドレス上のポート 80 とポート 2703 を使用します。このアドレスは、IPv4 クラスターの場合は 169.254.170.23、IPv6 クラスターの場合は `[fd00:ec2::23]` になります。

IPv6 アドレスを無効にするか、または localhost IPv6 IP アドレスを禁止すると、エージェントは起動できません。IPv6 を使用できないノードでエージェントを起動するには、「[the section called “IPv6 を無効にする”](#)」の手順に従って IPv6 設定を無効にします。

- Pod がプロキシを使用する場合は、Pod に挿入される `no_proxy/NO_PROXY` 環境変数に IPv4 に対しては 169.254.170.23 を、IPv6 に対しては `[fd00:ec2::23]` を間違いなく追加する必要があります。そうしないと、アプリケーション Pod から `eks-pod-identity-agent` DaemonSets へのリクエストが失敗します。リクエストはプロキシに送信されるものの、プロキシが IP をルーティングできないからです。

EKS Pod Identity クラスターバージョン

EKS Pod Identity を使用するには、クラスターのプラットフォームバージョンが、次の表に記載されているバージョンと同じかそれ以降、または表に記載されている Kubernetes バージョンよりも新しいバージョンである必要があります。

Kubernetes バージョン	プラットフォームバージョン
Kubernetes バージョンの記載がない	サポートされるプラットフォームのすべてのバージョン
1.28	eks.4

Kubernetes バージョン	プラットフォームバージョン
1.27	eks.8
1.26	eks.9

EKS Pod Identity の制限事項

EKS Pod Identity は以下で使用できます。

- 前のトピック [the section called “EKS Pod Identity クラスターバージョン”](#) に記載されている Amazon EKS クラスターバージョン。
- Linux Amazon EC2 インスタンスであるクラスターのワーカーノード。

EKS Pod Identity は以下では使用できません。

- AWS Outposts。
- Amazon EKS Anywhere
- Amazon EC2 で作成して実行する Kubernetes クラスター。EKS Pod Identity コンポーネントは Amazon EKS でのみ使用できます。

EKS Pod Identity は以下では使用できません。

- Linux Amazon EC2 インスタンス以外の場所で実行されるポッド。AWS Fargate (Fargate) 上で実行される Linux ポッドと Windows ポッドはサポートされていません。Windows Amazon EC2 インスタンスで実行されるポッドはサポートされていません。

EKS Pod Identity の仕組みを理解する

Amazon EKS Pod Identity の関連付けは、Amazon EC2 インスタンスプロファイルから Amazon EC2 インスタンスに認証情報を提供する場合と同じような方法で、アプリケーションの認証情報を管理する機能があります。

Amazon EKS Pod Identity は、追加の EKS Auth API と各ノードで実行されるエージェントポッドを使用して、ワークロードに認証情報を提供します。

Amazon EKS アドオン、自己管理型コントローラ、オペレータ、その他のアドオンなどのアドオンでは、作成者は最新の AWS SDK を使用するようにソフトウェアを更新する必要があります。EKS Pod Identity と Amazon EKS によって作成されたアドオンとの互換性のリストについては、前の [the section called “EKS Pod Identity の制限事項”](#) セクションを参照してください。

EKS Pod Identity をコードで使用する

コードでは、AWS SDK を使用して AWS サービスにアクセスできます。SDK を使用して AWS サービスのクライアントを作成するコードを記述すると、デフォルトで SDK は使用する AWS Identity and Access Management 認証情報を一連の場所で検索します。有効な認証情報が見つかったら、検索は停止されます。使用されるデフォルトの場所について詳しくは、「AWS SDK and Tools リファレンスガイド」の「[認証情報プロバイダーチェーン](#)」を参照してください。

EKS Pod Identity がコンテナ認証情報プロバイダーに追加されました。このプロバイダーはデフォルトの認証情報チェーンのステップで検索されます。現在、ワークロードが認証情報チェーンの前にある認証情報を使用している場合、同じワークロードに EKS Pod Identity の関連付けを設定しても、それらの認証情報は引き続き使用されます。この方法では、古い認証情報を削除する前に、まず関連付けを作成することで、他の種類の認証情報から安全に移行できます。

コンテナ認証情報プロバイダーは、各ノードで実行されるエージェントからの一時的な認証情報を提供します。Amazon EKS では、エージェントは Amazon EKS Pod Identity エージェントであり、Amazon Elastic Container Service ではエージェントは amazon-ecs-agent です。SDK は環境変数を使用して接続するエージェントを検索します。

一方、サービスアカウント用の IAM ロールは、AWS SDK が AssumeRoleWithWebIdentity を使用して AWS Security Token Service と交換しなければならないウェブ ID トークンを提供します。

EKS Pod Identity エージェントが Pod とどのように連携するか

1. Amazon EKS が起動した新しい Pod で EKS Pod Identity 関連付けのあるサービスアカウントが使用されている場合、クラスターは次のコンテンツを Pod マニフェストに追加します。

```
env:
- name: AWS_CONTAINER_AUTHORIZATION_TOKEN_FILE
  value: "/var/run/secrets/pods.eks.amazonaws.com/serviceaccount/eks-pod-identity-token"
- name: AWS_CONTAINER_CREDENTIALS_FULL_URI
  value: "http://169.254.170.23/v1/credentials"
volumeMounts:
- mountPath: "/var/run/secrets/pods.eks.amazonaws.com/serviceaccount/"
```

```
name: eks-pod-identity-token
volumes:
- name: eks-pod-identity-token
  projected:
    defaultMode: 420
    sources:
    - serviceAccountToken:
        audience: pods.eks.amazonaws.com
        expirationSeconds: 86400 # 24 hours
        path: eks-pod-identity-token
```

2. Kubernetes は Pod を実行するノードを選択します。次に、ノード上の Amazon EKS Pod Identity エージェントは [AssumeRoleForPodIdentity](#) アクションを使用して EKS Auth API から一時的な認証情報を取得します。
3. EKS Pod Identity エージェントは、コンテナ内で実行する AWS SDK でこれらの認証情報を利用できるようにします。
4. デフォルトの認証情報チェーンを使用する認証情報プロバイダーを指定しなくても、アプリケーションで SDK を使用できます。または、コンテナ認証情報プロバイダーを指定します。使用されるデフォルトの場所について詳しくは、「AWS SDK and Tools リファレンスガイド」の「[認証情報プロバイダーチェーン](#)」を参照してください。
5. SDK は環境変数を使用して EKS Pod Identity エージェントに接続し、認証情報を取得します。

Note

ワークロードが現在、認証情報チェーンの早い段階にある認証情報を使用している場合、同じワークロードに EKS Pod Identity の関連付けを設定しても、その認証情報は引き続き使用されます。

Amazon EKS Pod Identity エージェントのセットアップ

EKS Pod Identity の関連付けは、Amazon EC2 インスタンスプロファイルから Amazon EC2 インスタンスに認証情報を提供する場合と同じような方法で、アプリケーションの認証情報を管理する機能があります。

Amazon EKS Pod Identity は、追加の EKS Auth API と各ノードで実行されるエージェントポッドを使用して、ワークロードに認証情報を提供します。

i Tip

EKS Auto Mode クラスターに EKS Pod Identity エージェントをインストールする必要はありません。この機能は EKS Auto Mode に組み込まれています。

考慮事項

- EKS Pod Identity エージェントはデフォルトでポッドが認証情報をリクエストするために IPv4 と IPv6 のアドレスをリッスンします。エージェントは、IPv4 のループバック (localhost) IP アドレス 169.254.170.23 と IPv6 の localhost IP アドレス [fd00:ec2::23] を使用します。
- IPv6 アドレスを無効にするか、または localhost IPv6 IP アドレスを禁止すると、エージェントは起動できません。IPv6 を使用できないノードでエージェントを起動するには、「[the section called “IPv6 を無効にする”](#)」の手順に従って IPv6 設定を無効にします。

Amazon EKS Pod Identity エージェントの作成**エージェントの前提条件**

- 既存の アマゾン EKS クラスター。デプロイするには「[はじめに](#)」を参照してください。クラスターバージョンとプラットフォームバージョンは、[EKS Pod Identity クラスターバージョン](#)に記載されているバージョン以降である必要があります。
- ノードロールには、エージェントが EKS Auth API で AssumeRoleForPodIdentity アクションを実行する権限があります。[AWS マネージドポリシー: AmazonEKSTaskRolePolicy](#) を使用するか、次のようなカスタムポリシーを追加できます。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "eks-auth:AssumeRoleForPodIdentity"
      ],
      "Resource": "*"
    }
  ]
}
```

このアクションをタグで制限して、エージェントを使用するポッドが引き受けることができるロールを制限できます。

- ノードは Amazon ECR にアクセスしてイメージをダウンロードできます。アドオンのコンテナイメージは、「[Amazon EKS アドオンの Amazon コンテナイメージレジストリの表示](#)」に記載されているレジストリにあります。

なお、イメージの場所を変更して、AWS Management Console の [オプションの設定] や AWS CLI の `--configuration-values` で EKS アドオンの `imagePullSecrets` を提供することができます。

- ノードは Amazon EKS Auth API にアクセスできます。プライベートクラスターでは、AWS PrivateLink の `eks-auth` エンドポイントが必要です。

AWS コンソールでエージェントを設定する

1. [アマゾン EKS コンソール](#)を開きます。
2. 左のナビゲーションペインで、[クラスター] を選択し、次にアドオンを設定するEKS Pod Identity エージェントを選択します。
3. [アドオン] タブを選択してください。
4. [その他のアドオンを入手] を選択してください。
5. EKS Pod Identity のアドオンボックスの右上にあるボックスを選択し、[次へ] を選択します。
6. [選択したアドオン設定を構成する] ページの [バージョン] ドロップダウンリストで任意のバージョンを選択します。
7. (オプション) [オプションの設定] を展開して追加の設定を入力します。例えば、代替のコンテナイメージの場所と `ImagePullSecrets` を指定できます。許可されたキーのある JSON スキーマは、[アドオン設定スキーマ] に表示されます。

設定キーと値を [設定値] に入力します。

8. [Next] を選択します。
9. EKS Pod Identity がクラスター上で実行されていることを確認してください。

```
kubectl get pods -n kube-system | grep 'eks-pod-identity-agent'
```

出力例は次のとおりです。

eks-pod-identity-agent-gmqp7	1/1	Running
1 (24h ago) 24h		
eks-pod-identity-agent-prnsh	1/1	Running
1 (24h ago) 24h		

これで、クラスターで EKS Pod Identity の関連付けを使用できるようになりました。詳細については、「[the section called “IAM ロールを割り当てる”](#)」を参照してください。

AWS CLI でエージェントを設定する

1. 次の AWS CLI コマンドを実行します。my-cluster を自分のクラスター名に置き換えます。

```
aws eks create-addon --cluster-name my-cluster --addon-name eks-pod-identity-agent --addon-version v1.0.0-eksbuild.1
```

Note

EKS Pod Identity エージェントは、サービスアカウントの IAM ロール用の service-account-role-arn を使用しません。EKS Pod Identity エージェントにはノードロールの権限を付与する必要があります。

2. EKS Pod Identity がクラスター上で実行されていることを確認してください。

```
kubectl get pods -n kube-system | grep 'eks-pod-identity-agent'
```

出力例は次のとおりです。

eks-pod-identity-agent-gmqp7	1/1	Running
1 (24h ago) 24h		
eks-pod-identity-agent-prnsh	1/1	Running
1 (24h ago) 24h		

これで、クラスターで EKS Pod Identity の関連付けを使用できるようになりました。詳細については、「[the section called “IAM ロールを割り当てる”](#)」を参照してください。

IAM ロールを Kubernetes サービスアカウントに割り当てる

このトピックでは、EKS Pod Identity に関連付けられている AWS Identity and Access Management (IAM) ロールを引き受けるように Kubernetes サービスアカウントを設定する方法について説明します。任意の Pod はサービスアカウントを使用するように設定すると、ロールにアクセス許可がある AWS サービスすべてにアクセスできます。

EKS Pod Identity の関連付けを作成するには、1 つの手順しかありません。AWS Management Console、AWS CLI、AWS SDK、AWS CloudFormation などのツールを使用して EKS で関連付けを作成します。クラスター内の関連付けに関するデータやメタデータはどの Kubernetes オブジェクトにもなく、サービスアカウントに注釈を追加することはありません。

前提条件

- 既存のクラスター。まだ所有していない場合は、[はじめに](#) でのガイドのいずれかに従って作成できます。
- 関連付けを作成している IAM プリンシパルには `iam:PassRole` が必要です。
- ご使用のデバイスまたは AWS CloudShell に最新バージョンの AWS CLI がインストールおよび設定されていること。現在のバージョンは `aws --version | cut -d / -f2 | cut -d ' ' -f1` で確認できます。yum、apt-get、macOS 用の Homebrew などのパッケージマネージャーは、多くの場合 AWS CLI の最新バージョンより数バージョン古くなっています。最新バージョンをインストールするには、「AWS コマンドラインインターフェイスユーザーガイド」の「[インストール](#)」および「[aws configure を使用したクイック設定](#)」を参照してください。AWS CloudShell にインストールされている AWS CLI バージョンは、最新バージョンより数バージョン遅れている可能性もあります。更新するには、「AWS CloudShell ユーザーガイド」の「[ホームディレクトリへの AWS CLI のインストール](#)」を参照してください。
- デバイスまたは AWS クラウドシェル に、kubectl コマンドラインツールがインストールされていること。バージョンはご使用のクラスターの Kubernetes バージョンと同じか、1 つ前のマイナーバージョン以前、あるいはそれより新しいバージョンが使用できます。例えば、クラスターのバージョンが 1.29 である場合、kubectl のバージョン 1.28、1.29、または 1.30 が使用できます。kubectl をインストールまたはアップグレードする方法については「[the section called “kubectl と eksctl のセットアップ”](#)」を参照してください。
- クラスター構成を含む既存の kubectl config ファイル。kubectl config ファイルの作成については、「[the section called “kubectl を使用してクラスターにアクセスする”](#)」を参照してください。

Pod Identity の関連付けを作成する (AWS コンソール)

1. [アマゾン EKS コンソール](#)を開きます。
2. 左のナビゲーションペインで、[クラスター] を選択し、次にアドオンを設定するEKS Pod Identity エージェントを選択します。
3. [アクセス] タブを選択します。
4. [Pod Identity の関連付け] で [作成] を選択します。
5. [IAM ロール] には、ワークロードに付与する権限を持つ IAM ロールを選択します。

Note

リストには、EKS Pod Identity による使用を許可する以下の信頼ポリシーを持つロールのみが含まれています。

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

`sts:AssumeRole` - EKS Pod Identity は一時的な認証情報をポッドに渡す前に、`AssumeRole` を使用して IAM ロールを引き継ぎます。

`sts:TagSession` - EKS Pod Identity は、AWS STS へのリクエストにセッションタグを含めるために `TagSession` を使用します。

これらのタグを信頼ポリシーの条件キーを使用して、このロールを使用できるサービスアカウント、名前空間、およびクラスターを制限できます。

アマゾン EKS 条件キーのリストについては「サービス認可リファレンス」の「[アマゾン エラスティックKubernetesサービスによって定義された条件](#)」を参照してください。条件キーを使用できるアクションとリソースについては「[アマゾン エラスティックKubernetesサービスで定義されるアクション](#)」を参照してください。

6. [Kubernetes 名前空間] では、サービスアカウントとワークロードが含まれている Kubernetes 名前空間を選択します。オプションで、クラスターに存在しない名前でも名前空間名で指定できます。
7. Kubernetes サービスアカウントでは、使用する Kubernetes サービスアカウントを選択します。Kubernetes ワークロードのマニフェストには、このサービスアカウントを指定する必要があります。オプションで、クラスターに存在しないサービスアカウントを名前でも指定できます。
8. (オプション) [タグ] で [タグを追加] を選択し、キーと値のペアにメタデータを追加します。これらのタグは関連付けに適用され、IAM ポリシーで使用できます。

このステップでは、複数のリージョンを追加できます。

9. [Create] (作成) を選択します。

Pod Identity の関連付けを作成する (AWS CLI)

1. 既存の IAM ポリシーを IAM ロールに関連付ける場合は、次のステップにスキップします。

IAM ポリシーを作成します。ポリシーを自作することも、必要となるアクセス権限のいくつかは既に付与されている AWS 管理ポリシーをコピーし、特定の要件に応じてカスタマイズすることもできます。詳細についてはIAM ユーザーガイドの「[IAM ポリシーの作成](#)」を参照してください。

- a. Pod にアクセスさせる AWS サービスの権限を含むファイルを作成します。すべての AWS サービスに対するアクションの全リストについては、「[サービス認可リファレンス](#)」を参照してください。

次のコマンドを実行して、Amazon S3 バケットへの読み取り専用アクセスを許可するサンプルポリシーファイルを作成できます。必要に応じて、このバケットに設定情報またはブートストラップスクリプトを格納すると、Pod 内のコンテナがバケットからファイルを読み取り、アプリケーションにロードできます。このサンプルポリシーを作成する場合は、次のコンテンツを

デバイスにコピーします。*my-pod-secrets-bucket* をバケット名に置き換え、コマンドを実行します。

```
cat >my-policy.json <<EOF
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "s3:GetObject",
      "Resource": "arn:aws:s3:::my-pod-secrets-bucket"
    }
  ]
}
EOF
```

b. IAM ポリシーを作成します。

```
aws iam create-policy --policy-name my-policy --policy-document file://my-policy.json
```

2. IAM ロールを作成し、Kubernetes サービスアカウントに関連付けます。

a. IAM ロールを引き受ける既存の Kubernetes サービスアカウントがある場合は、この手順を省略できます。

Kubernetes サービスアカウントを作成します。次のコンテンツをデバイスにコピーします。*my-service-account* を目的の名前に置き換え、必要に応じて *default* を別の名前空間に置き換えます。*default* を変更する場合、名前空間は既に存在している必要があります。

```
cat >my-service-account.yaml <<EOF
apiVersion: v1
kind: ServiceAccount
metadata:
  name: my-service-account
  namespace: default
EOF
kubectl apply -f my-service-account.yaml
```

以下のコマンドを実行してください。

```
kubectl apply -f my-service-account.yaml
```

- b. IAM ロール用の信頼ポリシーファイルを作成するには、次のコマンドを実行します。

```
cat >trust-relationship.json <<EOF
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "AllowEksAuthToAssumeRoleForPodIdentity",
      "Effect": "Allow",
      "Principal": {
        "Service": "pods.eks.amazonaws.com"
      },
      "Action": [
        "sts:AssumeRole",
        "sts:TagSession"
      ]
    }
  ]
}
EOF
```

- c. ロールを作成します。*my-role* を IAM ロールの名前に置き換え、*my-role-description* をロールの説明に置き換えます。

```
aws iam create-role --role-name my-role --assume-role-policy-document file://
trust-relationship.json --description "my-role-description"
```

- d. IAM ポリシーをロールにアタッチします。*my-role* を IAM ロールの名前に置き換え、*my-policy* を、作成した既存のポリシーの名前に置き換えます。

```
aws iam attach-role-policy --role-name my-role --policy-
arn=arn:aws:iam::111122223333:policy/my-policy
```

 Note

サービスアカウントの IAM ロールとは異なり、EKS Pod Identity はサービスアカウントの注釈を使用しません。

- e. 次のコマンドを実行して、関連付けを作成します。my-cluster をクラスターの名前で置き換え、**my-service-account** を目的の名前に置き換え、必要に応じて **default** を別の名前空間に置き換えます。

```
aws eks create-pod-identity-association --cluster-name my-cluster --role-arn
arn:aws:iam::111122223333:role/my-role --namespace default --service-account my-
service-account
```

出力例は次のとおりです。

```
{
  "association": {
    "clusterName": "my-cluster",
    "namespace": "default",
    "serviceAccount": "my-service-account",
    "roleArn": "arn:aws:iam::111122223333:role/my-role",
    "associationArn": "arn:aws::111122223333:podidentityassociation/my-
cluster/a-abcdefghijklmnop1",
    "associationId": "a-abcdefghijklmnop1",
    "tags": {},
    "createdAt": 1700862734.922,
    "modifiedAt": 1700862734.922
  }
}
```

Note

名前空間とサービスアカウントは、クラスターには存在しない名前で指定できます。EKS Pod Identity の関連付けを機能させるには、名前空間、サービスアカウント、およびサービスアカウントを使用するワークロードを作成する必要があります。

設定の確認

1. IAM ロールの信頼ポリシーが正しく設定されていることを確認します。

```
aws iam get-role --role-name my-role --query Role.AssumeRolePolicyDocument
```

出力例は次のとおりです。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "Allow EKS Auth service to assume this role for Pod Identities",
      "Effect": "Allow",
      "Principal": {
        "Service": "pods.eks.amazonaws.com"
      },
      "Action": [
        "sts:AssumeRole",
        "sts:TagSession"
      ]
    }
  ]
}
```

2. 前の手順でロールにアタッチしたポリシーが、そのロールにアタッチされていることを確認します。

```
aws iam list-attached-role-policies --role-name my-role --query
'AttachedPolicies[].PolicyArn' --output text
```

出力例は次のとおりです。

```
arn:aws:iam::111122223333:policy/my-policy
```

3. 使用するポリシーの Amazon リソースネーム (ARN) を保存する変数を設定します。*my-policy* を、アクセス許可を確認するポリシーの名前に置き換えます。

```
export policy_arn=arn:aws:iam::111122223333:policy/my-policy
```

4. ポリシーのデフォルトバージョンを確認します。

```
aws iam get-policy --policy-arn $policy_arn
```

出力例は次のとおりです。

```
{
```

```
"Policy": {
  "PolicyName": "my-policy",
  "PolicyId": "EXAMPLEBIOWGLDEXAMPLE",
  "Arn": "arn:aws:iam::111122223333:policy/my-policy",
  "Path": "/",
  "DefaultVersionId": "v1",
  [...]
}
```

5. ポリシーの内容を表示して、Pod で必要な権限がすべて含まれていることを確認します。必要であれば、次のコマンドの **1** を、前の出力で返されたバージョンに置き換えます。

```
aws iam get-policy-version --policy-arn $policy_arn --version-id v1
```

出力例は次のとおりです。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "s3:GetObject",
      "Resource": "arn:aws:s3:::my-pod-secrets-bucket"
    }
  ]
}
```

前の手順でサンプルポリシーを作成した場合、出力は同じになります。別のポリシーを作成した場合、####の内容は異なります。

次のステップ

[the section called “Pod サービスアカウント”](#)

EKS Pod Identity のターゲット IAM ロールを使用して AWS リソースにアクセスする

Amazon Elastic Kubernetes Service (Amazon EKS) でアプリケーションを実行する場合、同じまたは異なる AWS アカウントに存在する AWS リソースへのアクセスが必要になることがあります。このガイドでは、EKS Pod Identity を使用してこれらのアカウント間のアクセスを設定する方法を示します。これにより、Kubernetes ポッドから他の AWS リソースにアクセスできるようになります。

前提条件

作業を開始する前に、次の手順が完了していることを確認してください。

- [Amazon EKS Pod Identity エージェントのセットアップ](#)
- [EKS Pod Identity ロールの作成](#)

仕組み

Pod Identity を使用すると、EKS クラスター内のアプリケーションから、ロールチェーンと呼ばれるプロセスを通じてアカウント間の AWS リソースにアクセスできます。Pod Identity の関連付けを作成する際は、EKS クラスターと同じアカウント内にある [EKS Pod Identity ロール](#)と、ユーザーの AWS リソース (S3 バケットや DynamoDB テーブルなど) が含まれたアカウントのターゲット IAM ロールという 2 つの IAM ロールを指定できます。[IAM PassRole](#) の要件により、[EKS Pod Identity ロール](#)は EKS クラスターのアカウント内に存在する必要がありますが、ターゲット IAM ロールは任意の AWS アカウントに存在できます。PassRole を使用すると、AWS エンティティはロールの引き受けを別のサービスに委任できます。EKS Pod Identity は PassRole を使用してロールを Kubernetes サービスアカウントに接続し、ロールとそれを渡す ID の両方が EKS クラスターと同じ AWS アカウントに存在する必要があります。アプリケーションポッドから AWS リソースにアクセスする必要がある場合、ポッドで Pod Identity の認証情報が要求されます。次に、Pod Identity は 2 つのロールの引き受けを順番に自動実行します。つまり、最初に [EKS Pod Identity ロール](#)を引き受け、次にその認証情報を使用してターゲット IAM ロールを引き受けます。このプロセスにより、ターゲットロールで定義されたアクセス許可を含む一時的な認証情報がポッドに提供され、他の AWS アカウントのリソースへの安全なアクセスが実現します。

キャッシュに関する考慮事項

キャッシュメカニズムにより、既存の Pod Identity の関連付けで IAM ロールを更新しても、EKS クラスターで実行されているポッドですぐに有効にならない場合があります。Pod Identity エージェントは、認証情報の取得時に関連付けの設定に基づいて IAM 認証情報をキャッシュします。関連付けに [EKS Pod Identity ロール](#) ARN のみが含まれ、ターゲット IAM ロールが含まれていない場合、キャッシュされた認証情報は 6 時間有効となります。関連付けに [EKS Pod Identity ロール](#) ARN とターゲット IAM ロールの両方が含まれている場合、キャッシュされた認証情報は 59 分間有効です。[EKS Pod Identity ロール](#) ARN の更新やターゲット IAM ロールの追加など、既存の関連付けを変更しても、既存のキャッシュはリセットされません。その結果、エージェントはキャッシュされた認証情報が更新されるまで更新を認識しません。変更をより迅速に適用するには、既存のポッドを再作成できます。再作成しない場合は、キャッシュの有効期限が切れるまで待機する必要があります。

ステップ 1: ターゲット IAM ロールを作成して関連付ける

このステップでは、ターゲット IAM ロールを作成して設定することで、安全な信頼チェーンを確立します。デモンストレーションでは、2 つの AWS アカウント間で信頼チェーンを確立するための新しいターゲット IAM ロールを作成します。EKS クラスターの AWS アカウント内にある [EKS Pod Identity ロール](#) (例: eks-pod-identity-primary-role) で、ターゲットアカウントでターゲット IAM ロール (例: eks-pod-identity-aws-resources) を引き受ける権限を取得し、Amazon S3 バケットなどの AWS リソースへのアクセスを有効にします。

ターゲット IAM ロールを作成する

1. [Amazon IAM コンソール](#)を開きます。
2. 上部のナビゲーションバーで、ターゲット IAM ロールの AWS リソース (S3 バケットや DynamoDB テーブルなど) を含むアカウントにサインインしていることを確認します。
3. 左のナビゲーションペインで、[ロール] を選択してください。
4. [ロールの作成] ボタンを選択し、「信頼できるエンティティタイプ」の下にある AWS アカウントを選択します。
5. 別の AWS アカウントを選択し、AWS のアカウント番号 ([EKS Pod Identity ロール](#)が存在するアカウント) を入力し、[次へ] を選択します。
6. ロールに関連付けるアクセス許可ポリシー (AmazonS3FullAccess など) を追加し、[次へ] を選択します。
7. 「MyCustomIAMTargetRole」などのロール名を入力し、[ロールの作成] を選択します。

ターゲット IAM ロールの信頼ポリシーを更新する

1. ロールを作成すると、[ロール] リストに戻ります。前の手順で作成した新規ロールを検索し、選択します (例: MyCustomIAMTargetRole)。
2. [信頼関係] タブを選択します。
3. 右側の [信頼ポリシーを編集] をクリックします。
4. ポリシーエディタで、デフォルトの JSON を信頼ポリシーに置き換えます。IAM ロール ARN のロール名と 111122223333 のプレースホルダー値を、EKS クラスターをホストする AWS アカウント ID に置き換えます。例:

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

```
"Statement": [
  {
    "Effect": "Allow",
    "Principal": {
      "AWS": "arn:aws:iam::111122223333:role/eks-pod-identity-primary-role"
    },
    "Action": "sts:AssumeRole",
  },
  {
    "Effect": "Allow",
    "Principal": {
      "AWS": "arn:aws:iam::111122223333:role/eks-pod-identity-primary-role"
    },
    "Action": "sts:TagSession"
  }
]
```

EKS Pod Identity ロールのアクセス許可ポリシーを更新する

このステップでは、ターゲット IAM ロール ARN をリソースとして追加し、Amazon EKS クラスターに関連付けられた [EKS Pod Identity ロール](#) のアクセス許可ポリシーを更新します。

1. [Amazon EKS コンソール](#)を開きます。
2. 左のナビゲーションペインで [クラスター] を選択し、次に EKS クラスターの名前を選択します。
3. [リモートアクセス] タブを選択してください。
4. [Pod Identity の関連付け] で、[EKS Pod Identity ロール](#) を選択します。
5. [アクセス許可]、[許可を追加] を選択し、次に [インラインポリシーを作成] をクリックします。
6. 右側で [JSON] を選択します。
7. ポリシーエディタで、デフォルトの JSON をアクセス許可ポリシーに置き換えます。IAM ロール ARN のロール名と 222233334444 のプレースホルダー値をターゲット IAM ロールに置き換えます。例:

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



```
        "sts:AssumeRole",
        "sts:TagSession"
    ],
    "Resource": "arn:aws:iam::222233334444:role/eks-pod-identity-aws-resources"
}
]
```

ステップ 2: ターゲット IAM ロールを Kubernetes サービスアカウントに関連付ける

このステップでは、ターゲット IAM ロールと EKS クラスター内の Kubernetes サービスアカウントとの関連付けを作成します。

1. [Amazon EKS コンソール](#)を開きます。
2. 左のナビゲーションペインで [クラスター] を選択し、次に関連付けを追加するクラスター名を選択します。
3. [アクセス] タブを選択します。
4. [Pod Identity の関連付け] で [作成] を選択します。
5. ワークロードが引き受ける IAM ロールの [EKS Pod Identity ロール](#)を選択します。
6. [EKS Pod Identity ロール](#)が引き受けるターゲット IAM ロールを [ターゲット IAM ロール] で選択します。
7. [Kubernetes 名前空間] フィールドに、関連付けを作成する名前空間の名前を入力します (例: my-app-namespace)。これにより、サービスアカウントが存在する場所が定義されます。
8. [Kubernetes サービスアカウント] フィールドに、IAM 認証情報を使用するサービスアカウントの名前 (例: my-service-account) を入力します。これにより、IAM ロールがサービスアカウントにリンクされます。
9. [作成] を選択して関連付けを作成します。

(オプション) ステップ 3: IAM ターゲットロールに外部アクセス許可を追加する

お客様の AWS リソースへのアクセス権をサードパーティーに付与 (委任) することが必要になる場合があります。例えば、Example Corp という第三者企業を雇って、コストを最適化するために AWS アカウントをモニタリングする業務を依頼するとします。Example Corp がお客様の毎日の支出を追跡するには、お客様の AWS リソースにアクセスする必要があります。この場合、IAM ターゲットロールの信頼ポリシーに ExternalId を追加して、[混乱した代理](#)の問題が発生する可能性を回避することをお勧めします。

信頼ポリシーを編集する

1. ロールを作成すると、[ロール] リストに戻ります。前のステップで作成した新しいロール (例: MyCustomIAMTargetRole) を検索し、クリックします。
2. [信頼関係] タブを選択します。
3. 右側の [信頼ポリシーを編集] をクリックします。
4. ポリシーエディタで、デフォルトの JSON を信頼ポリシーに置き換えます。aws-region/other-account/cluster-name/namespace/service-account-name の ExternalId プレースホルダー値を置き換えます。ここで「region」はクラスターの AWS リージョン、「111122223333」は他の AWS アカウント ID、「cluster-name」は EKS クラスター名、「namespace」は Kubernetes 名前空間、「service-account-name」は Kubernetes サービスアカウント名です。例：

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "AWS": "arn:aws:iam::111122223333:role/eks-pod-identity-primary-role"
      },
      "Action": "sts:AssumeRole",
      "Condition": {
        "StringEquals": {
          "sts:ExternalId": "region/111122223333/cluster-name/namespace/
service-account-name"
        }
      },
    },
    {
      "Effect": "Allow",
      "Principal": {
        "AWS": "arn:aws:iam::111122223333:role/eks-pod-identity-primary-role"
      },
      "Action": "sts:TagSession"
    }
  ]
}
```

サービスアカウントを使用して AWS サービスにアクセスするように Pod を設定する

Pod が AWS サービスにアクセスする必要がある場合は、Kubernetes サービスアカウントを使用するように設定する必要があります。サービスアカウントは、AWS サービスにアクセスする権限がある AWS Identity and Access Management (IAM) ロールに関連付ける必要があります。

- 既存のクラスター。まだ所有していない場合は、[はじめに](#) でのガイドのいずれかを参照しながら作成できます。
 - 既存の Kubernetes サービスアカウントと、そのサービスアカウントを IAM ロールに関連付ける EKS Pod Identity 関連付け。ロールには、Pod が AWS サービスを使用するアクセス許可を含む IAM ポリシーが関連付けられている必要があります。サービスアカウントとロールの作成および設定方法については、「[the section called “IAM ロールを割り当てる”](#)」を参照してください。
 - ご使用のデバイスまたは AWS CloudShell に最新バージョンの AWS CLI がインストールおよび設定されていること。現在のバージョンは `aws --version | cut -d / -f2 | cut -d ' ' -f1` で確認できます。yum、apt-get、macOS 用の Homebrew などのパッケージマネージャーは、多くの場合 AWS CLI の最新バージョンより数バージョン古くなっています。最新バージョンをインストールするには、「AWS コマンドラインインターフェイスユーザーガイド」の「[インストール](#)」および「[aws configure を使用したクイック設定](#)」を参照してください。AWS CloudShell にインストールされている AWS CLI バージョンは、最新バージョンより数バージョン遅れている可能性もあります。更新するには、「AWS CloudShell ユーザーガイド」の「[ホームディレクトリへの AWS CLI のインストール](#)」を参照してください。
 - デバイスまたは AWS クラウドシェル に、kubectl コマンドラインツールがインストールされていること。バージョンはご使用のクラスターの Kubernetes バージョンと同じか、1 つ前のマイナーバージョン以前、あるいはそれより新しいバージョンが使用できます。例えば、クラスターのバージョンが 1.29 である場合、kubectl のバージョン 1.28、1.29、または 1.30 が使用できます。kubectl をインストールまたはアップグレードする方法については「[the section called “kubectl と eksctl のセットアップ”](#)」を参照してください。
 - クラスター構成を含む既存の kubectl config ファイル。kubectl config ファイルの作成については、「[the section called “kubectl を使用してクラスターにアクセスする”](#)」を参照してください。
1. 次コマンドを使用して、Pod をデプロイして設定を確認できるデプロイマニフェストを作成します。*example* の値は独自の値に置き換えます。

```
cat >my-deployment.yaml <<EOF
apiVersion: apps/v1
kind: Deployment
metadata:
```

```
name: my-app
spec:
  selector:
    matchLabels:
      app: my-app
  template:
    metadata:
      labels:
        app: my-app
    spec:
      serviceAccountName: my-service-account
      containers:
      - name: my-app
        image: public.ecr.aws/nginx/nginx:X.XX
EOF
```

2. マニフェストをクラスターにデプロイします。

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

3. Pod に必要な環境変数が存在することを確認してください。

a. 前の手順のデプロイ時にデプロイされた Pod を確認します。

```
kubectl get pods | grep my-app
```

出力例は次のとおりです。

```
my-app-6f4dfff6cb-76cv9    1/1    Running    0    3m28s
```

b. Pod にサービスアカウントトークンのファイルがマウントされていることを確認します。

```
kubectl describe pod my-app-6f4dfff6cb-76cv9 | grep
AWS_CONTAINER_AUTHORIZATION_TOKEN_FILE:
```

出力例は次のとおりです。

```
AWS_CONTAINER_AUTHORIZATION_TOKEN_FILE: /var/run/secrets/
pods.eks.amazonaws.com/serviceaccount/eks-pod-identity-token
```

4. ロールにアタッチされた IAM ポリシーで割り当てたアクセス許可を使用して、Pod が AWS サービスとやり取りできることを確認します。

 Note

サービスアカウントに関連付けられた IAM ロールの AWS 認証情報を Pod が使用する場合、AWS CLI またはその Pod のコンテナ内にある他の SDK は、そのロールから提供される認証情報を使用します。[Amazon EKS ノードの IAM ロール](#)に提供された認証情報へのアクセスを制限しない場合、Pod は引き続きこれらの認証情報にアクセスできます。詳細については「[ワーカーノードに割り当てられたインスタンスプロファイルへのアクセスを制限する](#)」を参照してください。

Pod が想定通りにサービスとやり取りできない場合は、次の手順を実行して、すべてが正しく設定されていることを確認してください。

- a. EKS Pod Identity 関連付けを介した IAM ロールの引き受けをサポートする AWS SDK バージョンを Pod が使用していることを確認します。詳細については、「[the section called “サポートされている SDK”](#)」を参照してください。
- b. デプロイがサービスアカウントを使用していることを確認します。

```
kubectl describe deployment my-app | grep "Service Account"
```

出力例は次のとおりです。

```
Service Account:  my-service-account
```

タグに基づいてポッドに AWS リソースへのアクセス権を付与する

属性ベースのアクセス制御 (ABAC) は、複数の属性をまとめたポリシーを利用して、ユーザーに権限を付与するものです。EKS Pod Identity は、クラスター名、名前空間、サービスアカウント名などの属性が含まれたタグを各 Pod の一時認証情報にアタッチします。これらのロールセッションタグを使用すると、管理者は一致するタグに基づいて AWS リソースへのアクセスを許可することで、複数のサービスアカウントで機能する単一のロールを作成できます。ロールセッションタグのサポートを追加することで、同じ IAM ロールと IAM ポリシーを再利用しながら、クラスター間およびクラスター内のワークロード間のセキュリティ境界を厳しくすることができます。

タグ付きポリシーの例

以下は、対応するオブジェクトが EKS クラスター名でタグ付けされたときに s3:GetObject アクセス許可を付与する IAM ポリシーの例です。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "s3:ListBucket"
      ],
      "Resource": "*"
    },
    {
      "Effect": "Allow",
      "Action": [
        "s3:GetObject",
        "s3:GetObjectTagging"
      ],
      "Resource": "*",
      "Condition": {
        "StringEquals": {
          "s3:ExistingObjectTag/eks-cluster-name": "${aws:PrincipalTag/eks-cluster-name}"
        }
      }
    }
  ]
}
```

セッションタグを有効または無効にする

EKS Pod Identity で、ロールを引き受けるときに事前定義されたセッションタグのセットを追加します。これらのセッションタグを使用すると、管理者は一致するタグに基づいて AWS リソースへのアクセスを許可することで、リソース間で機能する単一のロールを作成できます。

セッションタグを有効にする

セッションタグは EKS Pod Identity で自動的に有効になります。ユーザー側のアクションは必要ありません。デフォルトでは、EKS Pod Identity によって事前定義されたタグのセットがセッションに

アタッチされます。ポリシーでこれらのタグを参照するには、`${aws:PrincipalTag/ }` 構文の後にタグキーを使用します。例えば、`${aws:PrincipalTag/kubernetes-namespace}`。

- `eks-cluster-arn`
- `eks-cluster-name`
- `kubernetes-namespace`
- `kubernetes-service-account`
- `kubernetes-pod-name`
- `kubernetes-pod-uid`

セッションタグを無効にする

AWS では、インラインセッションポリシー、管理ポリシー ARN、およびセッションタグが、個別の制限を持つひとまとめのバイナリ形式に圧縮されます。ひとまとめのバイナリ形式がサイズ制限を超過したことを示す `PackedPolicyTooLarge` エラーが表示された場合は、EKS Pod Identity によって追加されたセッションタグを無効にすることでサイズを縮小できます。これらのセッションタグを無効にするには、次の手順に従います。

1. [Amazon EKS コンソール](#)を開きます。
2. 左のナビゲーションペインで [クラスター] を選択し、次に修正するクラスターの名前を選択します。
3. [リモートアクセス] タブを選択してください。
4. [Pod Identity の関連付け] で、[関連付け ID] で変更する関連付け ID を選択し、[編集] を選択します。
5. [セッションタグ] で、[セッションタグを無効にする] を選択します。
6. [Save changes] (変更の保存) をクリックします。

アカウント間のコピー

EKS Pod Identity によって追加されるセッションタグはすべて推移的です。タグのキーと値は、ワークロードがロールを別のアカウントに切り替えるために使用するすべての `AssumeRole` アクションに渡されます。これらのタグを他のアカウントのポリシーで使用して、クロスアカウントシナリオでのアクセスを制限できます。詳細については、IAM ユーザーガイドの「[セッションタグを使用したロールの連鎖](#)」を参照してください。

カスタムタグ

EKS Pod Identity は、実行する AssumeRole アクションにカスタムタグを追加できません。ただし、IAM ロールに適用するタグは、常に同じ形式 (`${aws:PrincipalTag/MyCustomTag}`) のように、`${aws:PrincipalTag/` の後にキーが続く) で使用できます。

Note

`sts:AssumeRole` リクエストによってセッションに追加されたタグは、競合が発生した場合に優先されます。例えば、次のように言います。

- EKS が顧客ロールを引き継ぐときに Amazon EKS がセッションにキー `eks-cluster-name` と値 `my-cluster` を追加します。
- また、値 `my-own-cluster` を含む IAM ロールに `eks-cluster-name` タグも追加します。

この場合は前者が優先され、`eks-cluster-name` タグの値は `my-cluster` になります。

AWS SDK で Pod Identity を使用する

EKS Pod Identity 認証情報の使用

EKS Pod Identity の関連付けからの認証情報を使用するには、コードで任意の AWS SDK を使用して SDK を使用する AWS サービスのクライアントを作成できます。デフォルトでは、SDK は使用する AWS Identity and Access Management 認証情報を一連の場所で検索します。クライアントの作成時や SDK の初期化時に認証情報プロバイダーを指定しなかった場合は、EKS Pod Identity 認証情報が使用されます。

これがうまくいくのは、EKS Pod Identity がデフォルトの認証情報チェーンのステップで検索されるコンテナ認証情報プロバイダーに追加されているからです。現在、ワークロードが認証情報チェーンの前にある認証情報を使用している場合は、同じワークロードに EKS Pod Identity の関連付けを設定しても、その認証情報は引き続き使用されます。

EKS Pod Identity の仕組みの詳細については、「[the section called “仕組み”](#)」を参照してください。

「[Learn how EKS Pod Identity grants pods access to AWS services](#)」に従うと、Pod 内のコンテナでは EKS Pod Identity エージェントからの IAM ロールの引き受けをサポートする AWS SDK バージョン

ジョンを使用する必要があります。お使いの AWS SDK には、必ず次のバージョン以降を使用してください。

- Java (バージョン 2) – [2.21.30](#)
- Java – [1.12.746](#)
- [バージョン1に移動 – v1.47.11](#)
- [バージョン 2 に移動 – 2023-11-14 リリース](#)
- Python (Boto3) – [1.34.41](#)
- Python (botocore) – [1.34.41](#)
- AWS CLI – [1.30.0](#)

AWS CLI – [2.15.0](#)

- JavaScript v2 – [2.1550.0](#)
- JavaScript v3 – [v3.458.0](#)
- Kotlin – [v1.0.1](#)
- Ruby – [3.188.0](#)
- Rust – [release-2024-03-13](#)
- C++ – [1.11.263](#)
- .NET – [3.7.734.0](#)
- PowerShell – [4.1.502](#)
- PHP – [3.287.1](#)

サポートされている SDK を使用していることを確認するには、コンテナを構築する際に、「[AWS での構築ツール](#)」で、希望する SDK のインストール手順に従ってください。

EKS Pod Identity をサポートするアドオンのリストについては、「[the section called “Pod Identity サポートリファレンス”](#)」を参照してください。

EKS Pod Identity エージェントで IPv6 を無効にする

AWS Management Console

1. EKS Pod Identity エージェントで IPv6 を無効にするには、EKS アドオンの [オプション設定] に次の設定を追加します。

- a. [アマゾン EKS コンソール](#)を開きます。

- b. 左のナビゲーションペインで、[クラスター] を選択し、次にアドオンを設定するクラスター名を選択してください。
- c. [アドオン] タブを選択してください。
- d. EKS Pod Identity エージェントのアドオンボックスの右上にあるボックスを選択し、[編集] を選択します。
- e. EKS Pod Identity エージェントの設定ページの場合:
 - i. 使用する [バージョン] を選択します。前のステップと同じバージョンを保持し、別のアクションでバージョンと設定を更新することをお勧めします。
 - ii. [オプションの構成設定] を展開します。
 - iii. [設定値] で JSON キー "agent": とにネストした JSON オブジェクトの値に、キー "additionalArgs": を指定します。結果のテキストは有効な JSON オブジェクトでなければなりません。このキーと値だけがテキストボックス内のデータである場合はキーと値を中括弧 { } で囲みます。次の例はネットワークポリシーが有効になっていることを示しています :

```
{
  "agent": {
    "additionalArgs": {
      "-b": "169.254.170.23"
    }
  }
}
```

この設定では、エージェントが使用する唯一のアドレスに IPv4 アドレスを設定します。

- f. EKS Pod Identity エージェントのポッドを置き換えて新しい設定を適用するには、[変更を保存] を選択します。

Amazon EKS は、EKS Pod Identity エージェントの Kubernetes DaemonSet のロールアウトを使用して EKS アドオンに変更を適用します。ロールアウトのステータスは AWS Management Console のアドオン [更新履歴] と `kubectl rollout status daemonset/eks-pod-identity-agent --namespace kube-system` で追跡できます。

`kubectl rollout` は以下のコマンドを実行してください :

```
$ kubectl rollout
history -- View rollout history
```

```
pause      -- Mark the provided resource as paused
restart    -- Restart a resource
resume     -- Resume a paused resource
status     -- Show the status of the rollout
undo       -- Undo a previous rollout
```

ロールアウトに時間がかかりすぎる場合、Amazon EKS はロールアウトを取り消し、[アドオン更新] のタイプと [失敗] のステータスのメッセージがアドオンの [更新履歴] に追加されます。問題を調査するには、ロールアウトの履歴から開始し、EKS Pod Identity エージェントのポッドで `kubectl logs` を実行して EKS Pod Identity エージェントのログを確認します。

2. [更新履歴] の新しいエントリのステータスが [成功] の場合、ロールアウトが完了し、アドオンはすべての EKS Pod Identity エージェントのポッドで新しい設定を使用していることを意味します。

AWS CLI

1. EKS Pod Identity エージェントで IPv6 を無効にするには、EKS アドオンの [オプション設定] に次の設定を追加します。

次の AWS CLI コマンドを実行します。my-cluster をクラスターの名前に置き換え、IAM ロール ARN を使用するロールに置き換えます。

```
aws eks update-addon --cluster-name my-cluster --addon-name eks-pod-identity-agent \
  --resolve-conflicts PRESERVE --configuration-values '{"agent":{"additionalArgs":
  { "-b": "169.254.170.23"}}}'
```

この設定では、エージェントが使用する唯一のアドレスに IPv4 アドレスを設定します。

Amazon EKS は、EKS Pod Identity エージェントの Kubernetes DaemonSet のロールアウトを使用して EKS アドオンに変更を適用します。ロールアウトのステータスは AWS Management Console のアドオン [更新履歴] と `kubectl rollout status daemonset/eks-pod-identity-agent --namespace kube-system` で追跡できます。

`kubectl rollout` は以下のコマンドを実行してください：

```
kubectl rollout

history  -- View rollout history
pause   -- Mark the provided resource as paused
```

```
restart  -- Restart a resource
resume   -- Resume a paused resource
status   -- Show the status of the rollout
undo     -- Undo a previous rollout
```

ロールアウトに時間がかかりすぎる場合、Amazon EKS はロールアウトを取り消し、[アドオン更新] のタイプと [失敗] のステータスのメッセージがアドオンの [更新履歴] に追加されます。問題を調査するには、ロールアウトの履歴から開始し、EKS Pod Identity エージェントのポッドで `kubectl logs` を実行して EKS Pod Identity エージェントのログを確認します。

EKS Pod Identity で必須の信頼ポリシーを使用して IAM ロールを作成する

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

sts:AssumeRole

EKS Pod Identity は一時的な認証情報をポッドに渡す前に、AssumeRole を使用して IAM ロールを引き継ぎます。

sts:TagSession

EKS Pod Identity は、AWS STS へのリクエストにセッションタグを含めるために TagSession を使用します。

これらのタグを信頼ポリシーの条件キーを使用して、このロールを使用できるサービスアカウント、名前空間、およびクラスターを制限できます。

Amazon EKS 条件キーのリストについては「サービス認可リファレンス」の「[Amazon Elastic Kubernetes Service によって定義された条件](#)」を参照してください。条件キーを使用できるアクションとリソースについては「[Amazon Elastic Kubernetes Service で定義されるアクション](#)」を参照してください。

ノードを使用してコンピューティングリソースを管理する

Kubernetes ノードはコンテナ化されたアプリケーションを実行するマシンです。各ノードには以下のコンポーネントがあります：

- [コンテナランタイム](#) - コンテナの実行を担当するソフトウェア。
- [kubelet](#) - コンテナが正常で、関連する Pod 内で実行されていることを確認します。
- [kube-proxy](#) - Pod との通信を許可するネットワークルールを維持します。

詳細については、Kubernetes ドキュメントの「[ノード](#)」を参照してください。

Amazon EKS クラスターは、任意の [EKS Auto Mode マネージド型ノード](#)、[セルフマネージド型ノード](#)、[Amazon EKS マネージド型ノードグループ](#)、[AWS Fargate](#)、[Amazon EKS Hybrid Nodes](#) を組み合わせて、Pod をスケジュールできます。クラスターにデプロイされているノードの詳細については [the section called “クラスターリソースにアクセスする”](#) を参照してください。

Note

ハイブリッドノードを除くノードはクラスターの作成時に選択したサブネットと同じ VPC 内にある必要があります。ただし、ノードは同じサブネット内にある必要はありません。

コンピューティングオプションの比較

次の表に、要件を満たすための最適なオプションを決定する際に、評価すべき基準をいくつか示します。セルフマネージド型ノードはリストされているすべての条件をサポートするもう 1 つのオプションですが、より多くの手動メンテナンスが必要です。詳細については、「[the section called “セルフマネージド型ノード”](#)」を参照してください。

Note

Bottlerocket には、この表の一般的な情報と特に異なる点があります。詳しくは、GitHub 上で Bottlerocket の [ドキュメント](#) を参照してください。

条件	EKS マネージド型 ノードグループ	EKS 自動モードI	アマゾン EKS ハイブ リッドノード
AWS アウトポスト に デプロイ可能	いいえ	いいえ	いいえ
AWS ローカルゾーン にデプロイ可能	はい	いいえ	いいえ
Windows を必要とす るコンテナを実行で きる	はい	いいえ	いいえ
Linux を必要とするコ ンテナを実行可能	はい	あり	はい
インフェクエンティ ア チップを必要とす るワークロードを実 行可能	はい – アマゾンリ ナックス ノードのみ	はい	いいえ
GPU を必要とする ワークロードを実行 可能	はい – アマゾンリ ナックス ノードのみ	あり	あり
Arm プロセッサを必 要とするワークロー ドを実行可能	あり	あり	あり
AWS Bottlerocket を 実行可能	あり	はい	いいえ
ポッドはCPU、メモ リ、ストレージ、お よびネットワークリ ソースを他のポッド と共有します。	あり	あり	あり

条件	EKS マネージド型 ノードグループ	EKS 自動モードI	アマゾン EKS ハイブ リッドノード
Amazon EC2 インスタンスをデプロイおよび管理する必要がある	はい	いいえ - EC2 マネージドインスタンス について説明します	はい - オンプレミスの物理マシンまたは仮想マシンは選択したツールで管理されます。
Amazon EC2 インスタンスのオペレーティングシステムの保護、保守、パッチ適用を行う必要がある	はい	いいえ	はい - 物理マシンまたは仮想マシンで実行されているオペレーティングシステムは選択したツールで管理されます。
ノードをデプロイする時に、追加の kubelet 引数などのブートストラップ引数を提供できます。	はい - eksctl、またはカスタム AMI を含む 起動テンプレート を使用します。	いいえ - NodeClass を使用してノードを設定します	はい - nodeadm を使用してブートストラップ引数をカスタマイズできます。 「 the section called “ハイブリッドノード用 nodeadm” 」を参照してください。
ノードに割り当てられた IP アドレスとは異なる CIDR ブロックから、IP アドレスを Pod に割り当てることができます。	はい - カスタム AMI の起動テンプレートを使用します。詳細については「 the section called “起動テンプレート” 」を参照してください。	いいえ	はい - 「 the section called “CNI を設定する” 」を参照してください。
ノードに SSH 接続可能	はい	いいえ - ノードのトラブルシューティング方法について説明します	はい

条件	EKS マネージド型 ノードグループ	EKS 自動モードI	アマゾン EKS ハイブ リッドノード
独自のカスタム AMI をノードにデプロイ 可能	はい – 起動テンプレート を使用します	いいえ	はい
独自のカスタム CNI をノードにデプロイ 可能	はい – カスタム AMI の 起動テンプレート を使用します	いいえ	はい
ノード AMI を自分で 更新する必要がある	はい - アマゾン EKS 最適化 AMI をデプロイ した場合、更新が 利用可能になると ア マゾン EKS コンソール で通知されます。 コンソールではワン クリックで更新を実 行できます。カスタ ム AMI をデプロイし た場合、更新が利用 可能になっても アマ ゾン EKS コンソール に通知されません。 更新は自分で実行す る必要があります。	いいえ	はい - 物理マシンまた は仮想マシンで実行 されているオペレー ティングシステムは 選択したツールで管 理されます。「 the section called “オペ レーティングシステ ムを準備する” 」を参 照してください。

条件	EKS マネージド型 ノードグループ	EKS 自動モードI	アマゾン EKS ハイブ リッドノード
ノードの Kubernetes バージョンを自分で 更新する必要があります。	はい - アマゾン EKS 最適化 AMI をデプロ イした場合、更新が 利用可能になると ア マゾン EKS コンソール で通知されます。 コンソールではワン クリックで更新を実 行できます。カスタ ム AMI をデプロイし た場合、更新が利用 可能になっても アマ ゾン EKS コンソール に通知されません。 更新は自分で実行す る必要があります。	いいえ	はい - 独自のツール または nodeadm を 使用して、ハイブ リッドノードのアップ グレードを管理し ます。「 the section called “ハイブリッ ドノードをアップグ レードする” 」を参照 してください。
Pod で Amazon EBS ストレージを使用可 能	あり	はい。統合された機 能として使用できま す。 ストレージクラ スを作成 する方法に ついて説明します。	いいえ
Pod で Amazon EFS ストレージを使用可 能	あり	はい	いいえ
Pod で Amazon FSx for Lustre ストレージ を使用可能	あり	はい	いいえ
Network Load Balancer をサービス に使用可能	あり	あり	はい - ターゲットタイ プ ip を使用する必要 があります。

条件	EKS マネージド型 ノードグループ	EKS 自動モードI	アマゾン EKS ハイブ リッドノード
ポッドはパブリック サブネットで実行可 能	はい	あり	いいえ - ポッドはオン プレミス環境で実行 されます。
それぞれの Pod に、 異なる VPC セキュリ ティグループを割り 当て可能	はい : Linux ノードの み	いいえ	いいえ
Kubernetes DaemonSets を実行 可能	はい	あり	あり
Pod マニフェストで HostPort および HostNetwork をサ ポートします	はい	あり	あり
AWS が利用可能な リージョン	すべての アマゾン EKS 対応リージョン	すべての アマゾン EKS 対応リージョン	AWS Govクラウド (米国) リージョンと 中国リージョンを除 く Amazon EKS がサ ポートするすべての リージョン 。
Amazon EC2 専有ホ ストでコンテナを実 行できます	はい	いいえ	いいえ

条件	EKS マネージド型 ノードグループ	EKS 自動モードI	アマゾン EKS ハイブリッドノード
料金	複数の Pod を実行する Amazon EC2 インスタンスのコスト。詳細については アマゾン EC2 料金表 を参照してください。	クラスターで EKS 自動モードI が有効になっている場合は自動モードI のコンピューティング機能を使用して起動されたインスタンスに対しては標準の EC2 インスタンス料金に加えて個別の料金がかかります。この金額は起動されたインスタンスタイプとクラスターが配置されている AWS リージョンによって異なります。詳細については「 アマゾン EKS の料金 」を参照してください。	ハイブリッドノードの vCPU の 1 時間あたりのコスト。詳細については「 アマゾン EKS の料金 」を参照してください。

マネージドノードグループを使用してノードライフサイクルを簡素化する

Amazon EKS マネージド型ノードグループは、Amazon EKS Kubernetes クラスターのノード (Amazon EC2 インスタンス) のプロビジョニングとライフサイクル管理を自動化します。

Amazon EKS マネージド型ノードグループでは、Kubernetes アプリケーションを実行するためのコンピューティング性能を提供する Amazon EC2 インスタンスを個別にプロビジョニングまたは登録する必要はありません。1 回の操作で、クラスターのノードを作成、自動的に更新、または終了できます。ノードを更新し、終了させることで、ノードを自動的にドレーンし、アプリケーションを利用できる状態にしておきます。

すべてのマネージド型ノードは、Amazon EKS によって管理される Amazon EC2 Auto Scaling グループの一部としてプロビジョニングされます。インスタンスや Auto Scaling グループを含むすべてのリソースは、AWS アカウント内で実行されます。各ノードグループは、定義された複数のアベイラビリティゾーンで実行できます。

マネージド型ノードグループは、ノードのヘルスを継続的にモニタリングするノード自動修復をオプションで利用することもできます。検出された問題に自動的に対応し、可能な場合はノードを置き換えます。これにより、最小限の手動操作でクラスターの全体的な可用性が向上します。詳細については、「[the section called “ノードのヘルス”](#)」を参照してください。

Amazon EKS コンソール、`eksctl`、AWS CLI、AWS API、または AWS CloudFormation を含む Infrastructure as Code ツールを使用し、新規または既存のクラスターにマネージド型ノードグループを追加できます。マネージド型ノードグループの一部として起動されたノードは、自動的に Kubernetes [Cluster Autoscaler](#) によって自動検出用にタグ付けされます。ノードグループを使用して、Kubernetes ラベルをノードに適用し、いつでも更新できます。

Amazon EKS マネージド型ノードグループの使用に追加料金はかかりません。お支払いいただくのはプロビジョニングした AWS リソースの分だけです。これには、Amazon EC2 インスタンス、Amazon EBS ボリューム、Amazon EKS クラスター時間、その他の AWS インフラストラクチャが含まれます。最低料金や前払いの義務は発生しません。

新しい Amazon EKS クラスターおよびマネージド型ノードグループの使用を開始するには、「[the section called “クラスターを作成する \(コンソールと CLI\)”](#)」を参照してください。

既存のクラスターにマネージド型ノードグループを追加するには、「[the section called “作成”](#)」を参照してください。

マネージド型ノードグループの概念

- Amazon EKS マネージド型ノードグループは、Amazon EC2 インスタンスを作成し、管理します。
- すべてのマネージド型ノードは、Amazon EKS によって管理される Amazon EC2 Auto Scaling グループの一部としてプロビジョニングされます。さらに、Amazon EC2 インスタンスや Auto Scaling グループを含むすべてのリソースは、AWS アカウント内で実行されます。
- マネージド型ノードグループの Auto Scaling グループは、グループの作成時に指定するすべてのサブネットにまたがります。
- Amazon EKS は、Kubernetes [Cluster Autoscaler](#) の使用を設定するようにマネージド型ノードグループリソースをタグ付けします。

⚠ Important

Amazon EBS ボリュームによってバックアップされ、Kubernetes [Cluster Autoscaler](#) を使用する複数のアベイラビリティゾーンにわたってステートフルアプリケーションを実行している場合、それぞれが単一のアベイラビリティゾーンにスコープされる複数のノードグループを設定する必要があります。また、`--balance-similar-node-groups` 機能を有効にする必要があります。

- カスタム起動テンプレートを使用すると、マネージド型ノードをデプロイするときに、柔軟性とカスタマイズのレベルを向上させることができます。例えば、追加の `kubelet` 引数を指定して、カスタム AMI を使用できます。詳細については、「[the section called “起動テンプレート”](#)」を参照してください。マネージド型ノードグループを最初に作成するときにカスタム起動テンプレートを使用しない場合は、自動生成された起動テンプレートを使用できます。自動生成されたテンプレートを手動で変更しないでください。エラーが発生します。
- Amazon EKS は、マネージド型ノードグループで CVE およびセキュリティパッチの責任共有モデルに従います。マネージド型ノードが Amazon EKS 最適化 AMI を実行する場合、バグや問題が報告されると、Amazon EKS が AMI のパッチ適用バージョンの構築を担当します。修正を公開できます。ただし、これらのパッチが適用された AMI バージョンのマネージド型ノードグループへのデプロイはユーザーが担当します。マネージド型ノードでカスタム AMI を実行する場合、バグや問題が報告され、AMI をデプロイする場合は、ユーザーがパッチ適用バージョンの構築を担当します。詳細については、「[the section called “更新”](#)」を参照してください。
- Amazon EKS マネージド型ノードグループは、パブリックサブネットとプライベートサブネットの両方で起動できます。2020 年 4 月 22 日以降にパブリックサブネットでマネージド型ノードグループを起動した場合、インスタンスがクラスターに正常に参加するには、サブネットで `MapPublicIpOnLaunch` を `true` に設定する必要があります。パブリックサブネットが `eksctl`、または 2020 年 3 月 26 日以降に [Amazon EKS が販売した AWS CloudFormation テンプレート](#) を使用して作成された場合、この設定はすでに `true` に設定されています。パブリックサブネットが 2020 年 3 月 26 日より前に作成されている場合は、設定を手動で変更する必要があります。詳細については「[サブネットの IPv4 アドレス指定属性の変更](#)」を参照してください。
- マネージド型ノードグループをプライベートサブネットにデプロイする場合、Amazon ECR にアクセスしてコンテナイメージをプルできることを確認します。これを行うには、NAT ゲートウェイをサブネットのルートテーブルに接続するか、次の [AWS PrivateLink VPC エンドポイント](#) を追加します。
 - Amazon ECR API のエンドポイントインターフェイス — `com.amazonaws.region-code.ecr.api`

- Amazon ECR Docker レジストリ API のエンドポイントインターフェイス — `com.amazonaws.region-code.ecr.dkr`
- Amazon S3 ゲートウェイのエンドポイント - `com.amazonaws.region-code.s3`

その他の一般的に使用されるサービスとエンドポイントについては、「[the section called “プライベートクラスター”](#)」を参照してください。

- マネージドノードグループは、[AWS Outposts](#) または [AWS Wavelength](#) にはデプロイできません。マネージドノードグループは、[AWS Local Zones](#) で作成できます。詳細については、「[the section called “AWS Local Zones”](#)」を参照してください。
- 1つのクラスター内に複数のマネージド型ノードグループを作成できます。例えば、一部のワークロード用に、標準 Amazon EKS 最適化 Amazon Linux AMI を使用するノードグループを作成し、GPU サポートを必要とするワークロード用に GPU バリエーションを使用する別のノードグループを作成できます。
- マネージド型ノードグループで [Amazon EC2 インスタンスのステータスチェック](#) に障害が発生した場合、Amazon EKS は問題の診断に役立つエラーコードを返します。詳細については、「[the section called “マネージド型ノードグループのエラー”](#)」を参照してください。
- Amazon EKS は、マネージド型ノードグループインスタンスに Kubernetes ラベルを追加します。これらの Amazon EKS によって提供されたラベルには、プレフィックス `eks.amazonaws.com` が付与されます。
- Amazon EKS は、終了または更新時に Kubernetes API を使用してノードを自動的にドレーンします。
- AZRebalance を使用してノードを終了、または目的のノード数を減らす際には、ポッドの停止状態の予算は考慮されません。これらのアクションはノード内にある Pod を削除しようとします。ただし、15 分が経過すると、ノード内のすべての Pod が終了しているかどうかにかかわらず、ノードは終了します。ノードが終了するまでの期間を延長するには、Auto Scaling グループにライフサイクルフックを追加します。詳細については、「[Amazon EC2 Auto Scaling ユーザーガイド](#)」の「[ライフサイクルフックの追加](#)」を参照してください。
- スポット中断通知またはキャパシティリバランス通知を受け取った後、ドレインプロセスを正しく実行するには、CapacityRebalance を true に設定する必要があります。
- マネージドノードグループを更新すると、Pod に設定した Pod 中断予算が考慮されます。詳細については、「[the section called “動作の詳細を更新する”](#)」を参照してください。
- Amazon EKS マネージド型ノードグループの使用に、追加料金はかかりません。プロビジョニングした AWS リソースに対してのみ料金が発生します。

- ノードの Amazon EBS ボリュームを暗号化する場合は、起動テンプレートを使用してノードをデプロイできます。起動テンプレートを使用せずに、暗号化された Amazon EBS ボリュームを持つマネージド型ノードをデプロイするには、アカウントに作成された新しい Amazon EBS ボリュームをすべて暗号化してください。詳細については、「Amazon EC2 ユーザーガイド」の「[デフォルトで暗号化](#)」を参照してください。

マネージド型ノードグループのキャパシティータイプ

マネージド型ノードグループを作成する場合、キャパシティータイプをオンデマンドまたはスポットのいずれかから選択できます。Amazon EKS は、Amazon EC2 Auto Scaling グループを使用して、マネージド型ノードグループをデプロイします。これにはオンデマンドのみか、Amazon EC2 スポットインスタンスのみが含まれます。単一の Kubernetes クラスター内で、耐障害性のあるアプリケーションの Pod をスポットのマネージド型ノードグループにスケジュールし、非フォールトトレラントなアプリケーションの Pod をオンデマンドのノードグループにスケジュールできます。デフォルトでは、マネージド型ノードグループはオンデマンド Amazon EC2 インスタンスをデプロイします。

オンデマンド

オンデマンドインスタンスでは、長期契約なしで、コンピューティング性能に対して秒単位で支払います。

デフォルトでは、キャパシティータイプを指定しない場合、マネージド型ノードグループはオンデマンドインスタンスでプロビジョニングされます。マネージド型ノードグループは、ユーザーの代わりに Amazon EC2 Auto Scaling グループを次のように設定します。

- オンデマンドキャパシティーをプロビジョニングするための配分戦略は、`prioritized` に設定されます。マネージド型ノードグループは、API 内部で渡されたインスタンスタイプの優先度を使用して、オンデマンドキャパシティーを満たすときに最初に使用するインスタンスタイプを決定します。例えば、3 つのインスタンスタイプを `c5.large`、`c4.large`、`c3.large` の順序で指定するとします。オンデマンドインスタンスが起動されると、マネージド型ノードグループは `c5.large`、`c4.large`、`c3.large` の順でオンデマンドキャパシティーを満たします。詳しくは、Amazon EC2 Auto Scaling ユーザーガイドの [Amazon EC2 Auto Scaling グループ](#) を参照してください。
- Amazon EKS は、キャパシティータイプを `eks.amazonaws.com/capacityType: ON_DEMAND` に指定しているマネージド型ノードグループ内のすべてのノードに、次の Kubernetes ラベルを追加します。このラベルを使用すると、オンデマンドノードで、ステートフルまたは非フォールトトレラントなアプリケーションをスケジュールできます。

スポット

Amazon EC2 スポットインスタンスは、オンデマンドの料金から大きな割引で提供される、Amazon EC2 の予備容量です。Amazon EC2 スポットインスタンスは、EC2 から容量の返却を求められると 2 分間の中断通知をもって中断されることがあります。詳細については、「Amazon EC2 ユーザーガイド」の「[スポットインスタンス](#)」を参照してください。Amazon EC2 スポットインスタンスを使用してマネージド型ノードグループを構成し、Amazon EKS クラスターで実行されているコンピューティングノードのコストを最適化できます。

マネージド型ノードグループ内でスポットインスタンスを使用するには、キャパシティータイプを spot に設定してマネージド型ノードグループを作成してください。マネージド型ノードグループは、ユーザーの代わりに Amazon EC2 Auto Scaling グループを設定し、次のようなスポットベストプラクティスを適用します。

- Spot ノードが最適な Spot キャパシティープールにプロビジョニングされるように、割り当て戦略を以下のいずれかに設定します。
 - price-capacity-optimized (PCO) – Kubernetes バージョン 1.28 以降のクラスターに新しいノードグループを作成すると、割り当て戦略は price-capacity-optimized に設定されます。ただし、Amazon EKS マネージドノードグループが PCO をサポートし始める前に capacity-optimized で作成済みのノードグループの割り当て戦略は変更されません。
 - capacity-optimized (CO) – Kubernetes バージョン 1.27 以前のクラスターに新しいノードグループを作成すると、割り当て戦略は capacity-optimized に設定されます。

容量を割り当てるために利用可能なスポットキャパシティープールの数を増やすには、複数のインスタンスタイプを使用するようにマネージド型ノードグループを設定します。

- スポットノードが中断するリスクが高い場合に、Amazon EKS がスポットノードを適切にドレインおよび再調整して、アプリケーションの中断を最小限に抑えられるように、Amazon EC2 Spot Capacity Rebalancing が有効化されています。詳細については、Amazon EC2 Auto Scaling ユーザーガイドの [Amazon EC2 Auto Scaling 容量の再調整](#) を参照してください。
- スポットノードが再調整の推奨事項を受け取ると、Amazon EKS は自動的に新しい代替スポットノードの起動を試みます。
- 代替スポットノードが Ready 状態になる前にスポットの 2 分間の中断通知が到着すると、Amazon EKS は再調整に関する推奨事項を受け取ったスポットノードのドレインを開始します。Amazon EKS はベストエフォートベースでノードをドレインします。そのため、Amazon EKS が既存のノードをドレインする前に、代替ノードがクラスターに参加するのを待機するという保証はありません。

- 代替スポットノードがブートストラップされ、Kubernetes 上で Ready 状態になると、Amazon EKS は再調整に関する推奨事項を受信したスポットノードを遮断し、ドレインします。スポットノードを遮断すると、サービスコントローラーがこのスポットノードに新しいリクエストを送信しないようにします。正常でアクティブなスポットノードのリストからも削除されます。スポットノードをドレインすると、実行中の Pod が適切に削除されます。
- Amazon EKS は、キャパシティータイプを `eks.amazonaws.com/capacityType: SPOT` に指定しているマネージド型ノードグループ内のすべてのノードに、次の Kubernetes ラベルを追加します。このラベルを使用して、スポットノードで耐障害性のあるアプリケーションをスケジュールできます。

オンデマンドキャパシティーとスポットキャパシティーのどちらでノードグループをデプロイするかを決定するときは、次の条件を考慮する必要があります。

- スポットインスタンスは、ステートレスかつフォールトトレラントで、柔軟性の高いアプリケーションに適しています。これには、バッチトレーニングワークロード、機械学習トレーニングワークロード、Apache Spark などのビッグデータ ETL、キュー処理アプリケーション、ステートレス API エンドポイントなどがあります。スポットは Amazon EC2 の予備容量であり、時間の経過とともに変化する可能性があるため、中断されても問題ないワークロードには、スポットキャパシティーを使用することをお勧めします。具体的には、スポット容量は、必要な容量が使用できない期間を許容できるワークロードに適しています。
- 非フォールトトレラントなアプリケーションには、オンデマンドを使用することをお勧めします。これには、モニタリングツールやオペレーションツールなどのクラスター管理ツール、StatefulSets を必要とするデプロイ、データベースなどのステートフルなアプリケーションなどが含まれます。
- スポットインスタンスの使用中にアプリケーションの可用性を最大化するには、スポットのマネージド型ノードグループが複数のインスタンスタイプを使用するように構成することをお勧めします。複数のインスタンスタイプを使用する場合は、次のルールを適用することをお勧めします。
- マネージド型ノードグループ内で [Cluster Autoscaler](#) を使用している場合、同じサイズの vCPU とメモリリソースを持つインスタンスタイプを柔軟に組み合わせて使用することをお勧めします。これは、クラスター内のノードが期待どおりにスケールされるようにするためです。例えば、4 つの vCPU と 8 GiB のメモリが必要な場合は、`c3.xlarge`、`c4.xlarge`、`c5.xlarge`、`c5d.xlarge`、`c5a.xlarge`、`c5n.xlarge`、またはその他の同等なインスタンスタイプを使用してください。
- アプリケーションの可用性を高めるために、複数のスポットマネージド型ノードグループをデプロイすることをお勧めします。これを行うには、各グループが、同

じ vCPU とメモリリソースを持つ、インスタンスタイプの柔軟な組み合わせを使用する必要があります。例えば、4 つの vCPU および 8 GiB のメモリが必要な場合は、c3.xlarge、c4.xlarge、c5.xlarge、c5d.xlarge、c5a.xlarge、c5n.xlarge、または他の同等なインスタンスタイプのマネージド型ノードグループを 1 つ作成し、m3.xlarge、m4.xlarge、m5.xlarge、m5d.xlarge、m5a.xlarge、m5n.xlarge、または他の同等なインスタンスタイプで 2 つ目のマネージド型ノードグループを作成することをお勧めします。

- カスタム起動テンプレートを使用しているスポットキャパシティタイプでノードグループをデプロイする場合、API を使用して複数のインスタンスタイプを渡します。起動テンプレートを使って 1 つのインスタンスタイプを渡さないでください。起動テンプレートを使用したノードグループのデプロイについての詳細は、「[the section called “起動テンプレート”](#)」をご覧ください。

クラスターのマネージドノードグループを作成する

このトピックでは、Amazon EKS クラスターに登録しているノードの、Amazon EKS マネージド型ノードグループを起動する方法を説明します。ノードがクラスターに参加したら、それらのノードに Kubernetes アプリケーションをデプロイ可能になります。

Amazon EKS マネージド型ノードグループを初めて起動する場合は、[はじめに](#) のガイドのいずれかに従うことをお勧めします。このガイドでは、ノードを使用して Amazon EKS クラスターを作成するためのウォークスルーを説明します。

Important

- Amazon EKS ノードは、標準の Amazon EC2 インスタンスです。通常の Amazon EC2 料金に基づいて請求されます。詳細については、[Amazon EC2 の料金表](#)を参照してください。
- AWS Outposts または AWS Wavelength が有効になっている AWS リージョンでは、マネージドノードを作成できません。ユーザーは代わりに、セルフマネージド型ノードを作成できます。詳細については[the section called “Amazon Linux”](#)、[the section called “Windows”](#)、および[the section called “Bottlerocket”](#)を参照してください。また、Outpost に自己管理型 Amazon Linux ノードグループを作成することもできます。詳細については、「[the section called “ノード”](#)」を参照してください。
- Amazon EKS 最適化 Linux または Bottlerocket に含まれる bootstrap.sh ファイル用に [AMI ID を指定](#)しない場合、マネージドノードグループは maxPods の値に最大数を適用し

ます。vCPU が 30 未満のインスタンスの場合、最大数は 110 です。30 を超える vCPU を持つインスタンスの場合、最大数は 250 に跳ね上がります。これらの数値は、内部の Amazon EKS スケーラビリティチームのテストによる [Kubernetes スケーラビリティのしきい値](#)と推奨設定に基づいています。詳細については、「[Amazon VPC CNI プラグインがノードあたりのポッド数の制限を引き上げ](#)」のブログ記事を参照してください。

- 既存の Amazon EKS クラスター。デプロイするには「[the section called “クラスターを作成する”](#)」を参照してください。
- ノードが使用する既存の IAM ロール。作成する場合は「[the section called “ノードの IAM ロール”](#)」を参照してください。この役割に VPC CNI のポリシーがどちらも含まれていない場合、VPC CNI ポッドには以下の別の役割が必要です。
- (オプションですが推奨) 必要な IAM ポリシーがアタッチされた独自の IAM ロールで構成された Amazon VPC CNI plugin for Kubernetes アドオン。詳細については、「[the section called “IRSA 用の設定”](#)」を参照してください。
- 「[最適な Amazon EC2 ノードインスタンスタイプを選択する](#)」に記載されている考慮事項に精通していること。選択したインスタンスタイプによってはクラスターと VPC に関する追加の前提条件がある場合もあります。
- Windows マネージドノードグループを追加するには、まずクラスターの Windows サポートを有効にする必要があります。詳細については、「[the section called “Windows サポートを有効にする”](#)」を参照してください。

マネージド型ノードグループを作成するには、以下のいずれかを使用します。

- [the section called “eksctl”](#)
- [the section called “AWS Management Console”](#)

eksctl

eksctl を使用してマネージド型ノードグループを作成する

この手順には、eksctl バージョン 0.210.0 以降が必要です。お使いのバージョンは、以下のコマンドを使用して確認できます。

```
eksctl version
```

eksctl のインストールまたはアップグレードの手順については、eksctl ドキュメントの「[インストール](#)」を参照してください。

1. (オプション) [AmazonEKS_CNI_Policy] マネージド IAM ポリシーが [Amazon EKS ノードの IAM ロール](#) にアタッチされている場合は、代わりに Kubernetes aws-node サービスアカウントに関連付けた IAM ロールに割り当ててをお勧めします。詳細については、「[the section called “IRSA 用の設定”](#)」を参照してください。
2. カスタム起動テンプレートの有無にかかわらず、マネージド型ノードグループを作成します。起動テンプレートを手動で指定すると、ノードグループをより詳細にカスタマイズできます。たとえば、カスタム AMI をデプロイしたり、Amazon EKS 最適化 AMI の bootstrap.sh スクリプトの引数を指定したりできます。すべての使用可能なオプションとデフォルト値の一覧を表示するには、次のコマンドを入力します。

```
eksctl create nodegroup --help
```

次のコマンドで、*my-cluster* をクラスターの名前に置き換え、*my-mng* をノードグループの名前に置き換えます。ノードグループ名は 63 文字以下である必要があります。先頭は文字または数字でなければなりませんが、残りの文字にはハイフンおよびアンダースコアを含めることもできます。

Important

マネージド型ノードグループを最初に作成する際にカスタム起動テンプレートを使用しない場合は、後でノードグループに使用しないでください。カスタム起動テンプレートを指定しなかった場合、システムにより起動テンプレートが自動生成されます。これを手動で変更することはお勧めしません。自動生成された起動テンプレートを手動で変更すると、エラーが発生する場合があります。

起動テンプレートなし

eksctl は、デフォルトの Amazon EC2 起動テンプレートをアカウント内に作成し、指定したオプションに基づいて作成した起動テンプレートを使用してノードグループをデプロイします。--node-type の値を指定する前に「[the section called “Amazon EC2 インスタンスタイプ”](#)」を参照してください。

ami-family を許可されているキーワードに置き換えます。詳細については、「eksctl ドキュメント」の「[Setting the node AMI Family](#)」(ノード AMI ファミリーの設定) を参照してください。*my-*

key を Amazon EC2 キーペアまたはパブリックキーの名前に置き換えます。このキーは起動後のノードに SSH 接続するために使用されます。

 Note

Windows の場合、このコマンドは SSH を有効にしません。代わりに Amazon EC2 キーペアをインスタンスに関連付け、インスタンスに RDP できるようにします。

Amazon EC2 キーペアをまだ持っていない場合は、AWS Management Console で作成できます。Linux の詳細については、「Amazon EC2 ユーザーガイド」の「[Amazon EC2 key pairs and Linux instances](#)」を参照してください。Windows の詳細については、「Amazon EC2 ユーザーガイド」の「[Amazon EC2 key pairs and Windows instances](#)」を参照してください。

次の条件が true の場合、IMDS への Pod アクセスをブロックすることをお勧めします。

- IAM ロールをすべての Kubernetes サービスアカウントに割り当てることによって、必要最小限のアクセス許可のみを Pod に付与しようとしている。
- クラスター内の Pod が、現在の AWS リージョンの取得など、その他の理由で Amazon EC2 インスタンスメタデータサービス (IMDS) へのアクセスを必要としていない。

詳細については、「[ワーカーノードに割り当てられたインスタンスプロファイルへのアクセスを制限する](#)」を参照してください。

IMDS への Pod のアクセスをブロックするには、`--disable-pod-imds` オプションを次のコマンドに追加します。

```
eksctl create nodegroup \  
  --cluster my-cluster \  
  --region region-code \  
  --name my-mng \  
  --node-ami-family ami-family \  
  --node-type m5.large \  
  --nodes 3 \  
  --nodes-min 2 \  
  --nodes-max 4 \  
  --ssh-access \  
  --ssh-public-key my-key
```


ご使用のインスタンスでは、オプションで大量の IP アドレスをポッドに割り当てることや、インスタンスではなく CIDR ブロックから Pod に IP アドレスを割り当てることができます。また、そのインスタンスはインターネットにアクセスできないクラスターにデプロイすることも可能です。詳細については、追加オプションの[the section called “IP アドレスを増やす”](#)、[the section called “カスタム ネットワーキング”](#)および [the section called “プライベートクラスター”](#) を参照して、前のコマンドに追加します。

マネージドノードグループは、インスタンスタイプに基づいて、ノードグループの各ノードで実行できる Pod の最大数に対して 1 つの値を計算して適用します。異なるインスタンスタイプを持つノードグループを作成する場合、すべてのインスタンスタイプで計算された最小値が、ノードグループ内のすべてのインスタンスタイプで実行できる Pod の最大数として適用されます。マネージドノードグループは、[各 Amazon EC2 インスタンスタイプの Amazon EKS の推奨最大ポッド数](#)で参照するスクリプトを使用して値を計算します。

起動テンプレートの使用

起動テンプレートがすでに存在している必要があり、また、「[起動テンプレート設定の基本](#)」で指定されている要件を満たしている必要があります。次の条件が true の場合、IMDS への Pod アクセスをブロックすることをお勧めします。

- IAM ロールをすべての Kubernetes サービスアカウントに割り当てることによって、必要最小限のアクセス許可のみを Pod に付与しようとしている。
- クラスター内の Pod が、現在の AWS リージョンの取得など、その他の理由で Amazon EC2 インスタンスメタデータサービス (IMDS) へのアクセスを必要としていない。

詳細については、「[ワーカーノードに割り当てられたインスタンスプロフィールへのアクセスを制限する](#)」を参照してください。

IMDS への Pod のアクセスをブロックするには、起動テンプレートで必要な設定を行います。

- a. 次のコンテンツをデバイスにコピーします。*example values* を置き換えたら、変更したコマンドを実行して `eks-nodegroup.yaml` ファイルを作成します。起動テンプレートなしでデプロイしたときに指定したいいくつかの設定は、起動テンプレートに移動されます。`version` を指定しない場合は、テンプレートのデフォルトバージョンが使用されます。

```
cat >eks-nodegroup.yaml <<EOF
apiVersion: eksctl.io/v1alpha5
kind: ClusterConfig
metadata:
```

```
name: my-cluster
region: region-code
managedNodeGroups:
- name: my-mng
  launchTemplate:
    id: lt-id
    version: "1"
EOF
```

eksctl 設定ファイル使用の詳細については、eksctl ドキュメントの「[Config file schema](#)」を参照してください。オプションで、インスタンスはポッドに大量の IP アドレスを割り当てたり、インスタンスのブロックとは異なる CIDR ブロックから IP アドレスをポッドに割り当てたり、アウトバウンドのインターネットアクセスのないクラスターにデプロイしたりできます。詳細については、「[the section called “IP アドレスを増やす”](#)」、「[the section called “カスタムネットワーキング”](#)」、および「[the section called “プライベートクラスター”](#)」を参照して、設定ファイルに追加する追加オプションを確認してください。

起動テンプレートで AMI ID を指定しなかった場合、マネージドノードグループは、インスタンスタイプに基づいて、ノードグループの各ノードで実行できる Pod の最大数に対して 1 つの値を計算して適用します。異なるインスタンスタイプを持つノードグループを作成する場合、すべてのインスタンスタイプで計算された最小値が、ノードグループ内のすべてのインスタンスタイプで実行できる Pod の最大数として適用されます。マネージドノードグループは、[各 Amazon EC2 インスタンスタイプの Amazon EKS の推奨最大ポッド数](#)で参照するスクリプトを使用して値を計算します。

起動テンプレートで AMI ID を指定した場合、[カスタムネットワーク](#)を使用しているか、または[インスタンスに割り当てられている IP アドレスの数を増やす](#)場合には、ノードグループの各ノードで実行できる Pod の最大数を指定します。詳細については、「[the section called “各 Amazon EC2 インスタンスタイプの Amazon EKS の推奨最大ポッド数”](#)」を参照してください。

b. 次のコマンドでノードグループをデプロイします。

```
eksctl create nodegroup --config-file eks-nodegroup.yaml
```

AWS Management Console

AWS Management Console を使用してマネージド型ノードグループを作成する

1. クラスターステータスが ACTIVE と表示されるまで待ちます。まだ ACTIVE ではないクラスターにはマネージド型ノードグループを作成できません。
2. [Amazon EKS コンソール](#)を開きます。
3. マネージド型ノードグループを作成するクラスターの名前を選択します。
4. [コンピューティング] タブを選択します。
5. [ノードグループを追加] を選択します。
6. [ノードグループの設定] ページで、必要に応じてパラメータを指定し、[次へ] を選択します。
 - [名前] – マネージド型ノードグループの一意の名前を入力します。ノードグループ名は 63 文字以下である必要があります。先頭は文字または数字でなければなりません。残りの文字にはハイフンおよびアンダースコアを含めることもできます。
 - [ノード IAM ロール] – ノードグループで使用するノードインスタンスロールを選択します。詳細については、「[the section called “ノードの IAM ロール”](#)」を参照してください。

 Important

- クラスターの作成に使用したロールは使用できません。
 - セルフマネージド型ノードグループによって現在使用されていないロールを使用することをお勧めします。それ以外の場合は、新しいセルフマネージド型ノードグループで使います。詳細については、「[the section called “削除”](#)」を参照してください。
-
- 起動テンプレートを使用する — (オプション) 既存の起動テンプレートを使用するかどうかを選択します。[起動テンプレート名] を選択します。次に、[起動テンプレートのバージョン] を選択します。バージョンを選択しない場合、Amazon EKS はテンプレートのデフォルトのバージョンを使用します。起動テンプレートを使用すると、ノードグループを詳細にカスタマイズできます。これにより、カスタム AMI のデプロイ、ポッドへの大量の IP アドレスの割り当て、インスタンスのブロックとは異なる CIDR ブロックからのポッドに対する IP アドレスの割り当て、さらにアウトバウンドのインターネットアクセスのないクラスターに対するノードのデプロイが可能になります。詳細については[the section called “IP アドレスを増やす”](#)、[the section called “カスタムネットワーキング”](#)、および[the section called “プライベートクラスター”](#)を参照してください。

起動テンプレートは、「[起動テンプレートを使用してマネージドノードをカスタマイズする](#)」の要件を満たしている必要があります。独自の起動テンプレートを使用しない場合、Amazon EKS API はデフォルトの Amazon EC2 起動テンプレートをアカウントに作成し、デフォルトの起動テンプレートを使用してノードグループをデプロイします。

[サービスアカウントの IAM ロール](#)を実装する場合は、AWS サービスへのアクセス許可を必要とするすべての Pod に必要なアクセス許可を直接割り当て、クラスター内の Pod が、現在の AWS リージョンを取得するなどの理由で IMDS にアクセスしないようにします。また、起動テンプレートでホストネットワークを使用しないポッドの、IMDS へのアクセスを無効にすることもできます。詳細については、「[ワーカーノードに割り当てられたインスタンスプロフィールへのアクセスを制限する](#)」を参照してください。

- [Kubernetes ラベル]: (オプション) マネージド型ノードグループ内のノードに Kubernetes ラベルを適用するように選択できます。
- Kubernetes テイント - (オプション) 管理対象ノードグループ内のノードに Kubernetes テイントを適用することを選択できます。[効果] メニューでの利用可能なオプションは **NoSchedule**、**NoExecute**、および **PreferNoSchedule** です。詳細については、「[the section called “GPU ノードにテイントを適用する”](#)」を参照してください。
- [タグ] - (オプション) Amazon EKS マネージド型ノードグループにタグを付けることを選択できます。これらのタグは、Auto Scaling グループやインスタンスなど、ノードグループ内の他のリソースには伝達されません。詳細については、「[the section called “リソースのタグ付け”](#)」を参照してください。

7. [コンピューティング構成とスケーリングの設定] ページで、必要に応じてパラメータを指定し、[次へ] を選択します。

- [AMI タイプ] - AMI タイプを選択します。Arm インスタンスをデプロイする場合は、デプロイする前に「[Amazon EKS optimized Arm Amazon Linux AMIs](#)」の考慮事項を確認してください。

前のページで起動テンプレートを指定し、起動テンプレートで AMI を指定した場合は、値を選択できません。テンプレートの値が表示されます。テンプレートで指定された AMI は、「[AMI の指定](#)」の要件を満たしている必要があります。

- [キャパシティータイプ] - キャパシティータイプを選択します。キャパシティータイプの選択の詳細については、「[the section called “マネージド型ノードグループのキャパシティータイプ”](#)」を参照してください。同じノードグループ内で、異なるキャパシティータイプを混在させることはできません。両方のキャパシティータイプを使用したい場合は、キャパシティータイプとインスタンスタイプをそれぞれに持つ、別々のノードグループを作成します。GPU アクセラレーテッドワーカーノードのプロビジョニングとスケーリングについては、「[the section called “MNG の GPU を予約する”](#)」を参照してください。
- [インスタンスタイプ] - デフォルトで 1 つまたは複数のインスタンスタイプが指定されています。デフォルトのインスタンスタイプを削除するには、インスタンスタイプの右側にある X を選択します。マネージド型ノードグループで使用するインスタンスタイプを選択します。詳細については、「[the section called “Amazon EC2 インスタンスタイプ”](#)」を参照してください。

コンソールには、一般的に使用されるインスタンスタイプのセットが表示されます。表示されていないインスタンスタイプを持つマネージド型ノードグループの作成が必要な場合は、eksctl、AWS CLI、AWS CloudFormation、または SDK を使用して、ノードグループを作成します。前のページで起動テンプレートを指定した場合、起動テンプレートでインスタンスタイプを指定する必要があるため、値を選択できません。起動テンプレートの値が表示されます。[キャパシティータイプ] で [スポット] を選択した場合は、可用性を高めるために、複数のインスタンスタイプを指定することをお勧めします。

- [ディスクサイズ] – ノードのルートボリュームに使用するディスクサイズ (GiB 単位) を入力します。

前のページで起動テンプレートを指定した場合は、値を起動テンプレートで指定するため、値を選択できません。

- [必要なサイズ] – マネージド型ノードグループが起動時に保持する必要があるノードの現在の数を指定します。

 Note

Amazon EKS は、ノードグループを自動的にスケールインまたはスケールアウトしません。ただし、これを行うように Kubernetes Cluster Autoscaler を設定することはできません。詳細については、「[AWS の Cluster Autoscaler](#)」を参照してください。

- [最小サイズ] – マネージド型ノードグループがスケールインできるノードの最小数を指定します。
- [最大サイズ] – マネージド型ノードグループがスケールアウトできるノードの最大数を指定します。
- ノードグループの更新設定 – (オプション) 並行して更新するノードの数または割合を選択できます。これらのノードは、更新中は使用できません。[使用できない最大値] で、次のいずれかのオプションを選択し、その [値] を指定します:
 - [数値] – 並行して更新できるノードグループ内のノード数を選択して指定します。
 - [パーセンテージ] – 並行して更新できるノードグループ内のノードの割合を選択して指定します。ノードグループに多数のノードがある場合に便利です。
- ノードの自動修復設定 – (オプション) [ノードの自動修復を有効にする] チェックボックスを有効にすると、Amazon EKS は検出された問題が発生したときにノードを自動的に置き換えます。詳細については、「[the section called “ノードのヘルス”](#)」を参照してください。

8. [ネットワーキングを指定] ページで、必要に応じてパラメータを指定し、[次へ] を選択します。

- [サブネット]: マネージド型ノードを起動するサブネットを選択します。

 Important

Amazon EBS ボリュームによってバックアップされ、Kubernetes [Cluster Autoscaler](#) を使用する複数のアベイラビリティゾーンにわたってステートフルアプリケーションを実行している場合、それぞれが単一のアベイラビリティゾーンにスコープされる複数のノードグループを設定する必要があります。また、`--balance-similar-node-groups` 機能を有効にする必要があります。

 Important

- パブリックサブネットを選択し、クラスターでパブリック API サーバーのエンドポイントのみが有効になっている場合は、サブネットの `MapPublicIPOnLaunch` に `true` をセットして、インスタンスがクラスターに正常に参加できるようにします。サブネットが 2020 年 3 月 26 日以降に `eksctl` または [Amazon EKS で発行された AWS CloudFormation テンプレート](#) を使用して作成された場合は、この設定はすでに `true` に設定されています。サブネットが 2020 年 3 月 26 日より前に `eksctl` または AWS CloudFormation テンプレートを使用して作成されている場合は、設定を手動で変更する必要があります。詳細については「[サブネットの IPv4 アドレス指定属性の変更](#)」を参照してください。
 - 起動テンプレートを使用しており、複数のネットワークインターフェイスを指定している場合には、たとえば `MapPublicIpOnLaunch` が `true` に設定されていても、Amazon EC2 はパブリック IPv4 アドレスの自動的な割り当てを行いません。このシナリオでノードがクラスターに参加するには、クラスターのプライベート API サーバーエンドポイントを有効にするか、NAT ゲートウェイなどの別の方法によってアウトバウンドインターネットアクセスを提供する、プライベートサブネットでノードを起動する必要があります。詳細については、「Amazon EC2 ユーザーガイド」の「[Amazon EC2 インスタンスの IP アドレス指定](#)」を参照してください。
- ノードへの SSH アクセスの設定 (オプション)。SSH を有効にすることにより、インスタンスに接続し、問題がある場合に診断情報を収集できます。ノードグループを作成するときは、リモートアクセスを有効にすることを強くお勧めします。ノードグループの作成後にリモートアクセスを有効にすることはできません。

起動テンプレートの使用を選択した場合、このオプションは表示されません。ノードへのリモートアクセスを有効にするには、起動テンプレートでキーペアを指定し、起動テンプレートで指定したセキュリティグループのノードに対して適切なポートが開いていることを確認します。詳細については、「[the section called “カスタムセキュリティグループを使用する”](#)」を参照してください。

 Note

Windows の場合、このコマンドは SSH を有効にしません。代わりに Amazon EC2 キーペアをインスタンスに関連付け、インスタンスに RDP できるようにします。

- [SSH キーペア] (オプション) の場合は、使用する Amazon EC2 SSH キーを選択します。Linux の詳細については、「Amazon EC2 ユーザーガイド」の「[Amazon EC2 key pairs and Linux instances](#)」を参照してください。Windows の詳細については、「Amazon EC2 ユーザーガイド」の「[Amazon EC2 key pairs and Windows instances](#)」を参照してください。起動テンプレートを使用することを選択した場合、選択することはできません。Bottlerocket AMI を使用するノードグループに Amazon EC2 SSH キーが提供されると、管理コンテナも有効になります。詳細については、GitHub の [Admin container](#) を参照してください。
 - [次からの SSH リモートアクセスを許可する] の場合、特定のインスタンスへのアクセスを制限するには、それらのインスタンスに関連付けられているセキュリティグループを選択します。特定のセキュリティグループを選択しないと、インターネット上のどの場所 (0.0.0.0/0) からでも SSH アクセスが許可されます。
9. [確認と作成] ページで、マネージド型ノードグループの設定を確認し、[作成] を選択してください。

ノードがクラスターに参加できない場合はトラブルシューティングの章にある「[the section called “ノードをクラスターに結合できません”](#)」を参照してください。

10. ノードのステータスを監視し、Ready ステータスになるまで待機します。

```
kubectl get nodes --watch
```

11. (GPU ノードのみ) GPU インスタスタイプと Amazon EKS 最適化高速 AMI を選択した場合は、クラスター上の DaemonSet として [Kubernetes 用の NVIDIA デバイスプラグイン](#) を適用する必要があります。次のコマンドを実行する前に、`vX.X.X` を必要となる [NVIDIA/k8s-device-plugin](#) バージョンに置き換えます。

```
kubectl apply -f https://raw.githubusercontent.com/NVIDIA/k8s-device-plugin/vX.X.X/
deployments/static/nvidia-device-plugin.yml
```

Kubernetes アドオンをインストールする

ノードが関連付けられた Amazon EKS クラスターが実行中になったところで、Kubernetes アドオンのインストールとクラスターへのアプリケーションのデプロイを開始できます。以下のトピックはクラスターの機能を拡張するのに役立ちます。

- クラスターを作成した [IAM プリンシパル](#) は、kubectl または AWS Management Console を使用して Kubernetes API サーバーを呼び出すことができる唯一のプリンシパルです。他の IAM プリンシパルがクラスターにアクセスできるようにする場合はそれらを追加する必要があります。詳細については、「[the section called “Kubernetes API アクセス”](#)」および「[the section called “必要なアクセス許可”](#)」を参照してください。
- 次の条件が true の場合、IMDS への Pod アクセスをブロックすることをお勧めします。
 - IAM ロールをすべての Kubernetes サービスアカウントに割り当てることによって、必要最小限のアクセス許可のみを Pod に付与しようとしている。
 - クラスター内の Pod が、現在の AWS リージョンの取得など、その他の理由で Amazon EC2 インスタンスメタデータサービス (IMDS) へのアクセスを必要としていない。

詳細については「[ワーカーノードに割り当てられたインスタンスプロファイルへのアクセスを制限する](#)」を参照してください。

- ノードグループ内のノード数を自動的に調整するように Kubernetes [Cluster Autoscaler](#) を設定します。
- [サンプルアプリケーション](#) をクラスターにデプロイします。
- クラスターを管理するための重要なツールを使用して、[クラスターリソースを整理およびモニタリング](#)します。

クラスターのためにマネージドノードグループを更新する

マネージド型ノードグループの更新を開始すると、Amazon EKS はノードを自動的に更新し、「[ノードの更新の各フェーズを理解する](#)」でリストされた手順を完了します。Amazon EKS 最適化 AMI を使用している場合、Amazon EKS は最新のセキュリティパッチとオペレーティングシステムの更新を、最新の AMI リリースバージョンの一部として自動的にノードに適用します。

いくつかのシナリオでは、Amazon EKS マネージド型ノードグループのバージョンや設定を更新すると便利です。

- Amazon EKS クラスターの Kubernetes バージョンを更新し、同じ Kubernetes バージョンを使用するようにノードを更新する場合。
- マネージド型ノードグループでは、新しい AMI リリースバージョンを使用できます。AMI バージョンの詳細については、次のセクションを参照してください。
 - [the section called “バージョン情報を取得する”](#)
 - [the section called “Bottlerocket”](#)
 - [the section called “バージョン情報を取得する”](#)
- マネージド型ノードグループ内のインスタンスの最小数、最大数、または必要な数を調整する場合。
- マネージド型ノードグループのインスタンスで Kubernetes ラベルを追加または削除する場合。
- マネージド型ノードグループに AWS タグを追加または削除する場合。
- カスタム AMI の更新などの設定の変更を伴う、新しいバージョンの起動テンプレートをデプロイする必要があります。
- Amazon VPC CNI アドオンのバージョン 1.9.0 以降をデプロイし、プレフィックス委任のアドオンを有効にし、ノードグループ内の新しい AWS Nitro System インスタンスで大幅に増加した Pod の数をサポートしたい場合。詳細については、「[the section called “IP アドレスを増やす”](#)」を参照してください。
- Windows ノードの IP プレフィックスの委任を有効にし、ノードグループ内の新しい AWS Nitro System インスタンスで、大幅に増加したポッドをサポートした場合。詳細については、「[the section called “IP アドレスを増やす”](#)」を参照してください。

マネージドノードグループの Kubernetes バージョンに対して、新しい AMI リリースバージョンがある場合は、ノードグループのバージョンを更新して、新しい AMI バージョンを使用することができます。同様に、クラスターがノードグループより新しい Kubernetes バージョンを実行している場合、クラスターの Kubernetes バージョンに一致する最新の AMI リリースバージョンを使用するようにノードグループを更新できます。

スケーリングオペレーションまたは更新によってマネージドノードグループ内のノードが終了すると、そのノードの Pod が最初にドレインされます。詳細については、「[the section called “動作の詳細を更新する”](#)」を参照してください。

ノードグループバージョンの更新

ノードグループのバージョンは、次のいずれかを使用して更新できます。

- [the section called “eksctl”](#)
- [the section called “AWS Management Console”](#)

更新するバージョンは、コントロールプレーンバージョンよりも新しいバージョンにすることはできません。

eksctl

eksctl を使用してマネージドノードグループを更新する

次のコマンドを使用して、マネージドノードグループを、ノードに現在デプロイされているのと同じ Kubernetes バージョンの最新 AMI リリースに更新します。各####は独自の値に置き換えます。

```
eksctl upgrade nodegroup \  
  --name=node-group-name \  
  --cluster=my-cluster \  
  --region=region-code
```

Note

起動テンプレートでデプロイされたノードグループを、新しい起動テンプレートのバージョンにアップグレードする場合は、上記のコマンドに `--launch-template-version version-number` を追加します。起動テンプレートは、「[起動テンプレートを使用してマネージドノードをカスタマイズする](#)」に記載されている要件を満たしている必要があります。起動テンプレートにカスタム AMI が含まれている場合、AMI は「[Specifying an AMI](#)」(AMI の指定) の要件を満たしている必要があります。ノードグループを新しいバージョンの起動テンプレートにアップグレードすると、指定した起動テンプレートバージョンの新しい設定と一致するように、すべてのノードがリサイクルされます。

起動テンプレートなしでデプロイしたノードグループを、新しい起動テンプレートバージョンに直接アップグレードすることはできません。代わりに、起動テンプレートを使用して新しいノードグループをデプロイし、新しい起動テンプレートバージョンにノードグループを更新する必要があります。

コントロールプレーンの Kubernetes バージョンと同じバージョンにノードグループをアップグレードできます。例えば、Kubernetes 1.32 を実行しているクラスターがある場合、次のコマンドを使用して、現在 Kubernetes 1.31 を実行しているノードをバージョン 1.32 にアップグレードできます。

```
eksctl upgrade nodegroup \  
  --name=node-group-name \  
  --cluster=my-cluster \  
  --region=region-code \  
  --kubernetes-version=1.32
```

AWS Management Console

AWS Management Console を使用してマネージドノードグループを更新する

1. [アマゾン EKS コンソール](#)を開きます。
2. 更新するノードグループを含むクラスターを選択します。
3. 少なくとも 1 つのノードグループに利用可能な更新がある場合、ページの上部に利用可能な更新について通知するボックスが表示されます。[Compute] (コンピューティング) タブを選択すると、利用可能な更新があるノードグループの [Node Groups] (ノードグループ) 表の、[AMI release version] (AMI リリースバージョン) 列に、[Update now] (今すぐ更新) が表示されます。ノードグループを更新するには、[Update now] (今すぐ更新) を選択します。

カスタム AMI でデプロイされたノードグループの通知は表示されません。ノードがカスタム AMI でデプロイされている場合は、次の手順を実行して、新しく更新されたカスタム AMI をデプロイします。

- a. AMI の新しいバージョンを作成します。
 - b. 新しい AMI ID を使用して、新しい起動テンプレートのバージョンを作成します。
 - c. 起動テンプレートの新しいバージョンにノードを更新する
4. [Update node group version] (ノードグループバージョンの更新) ダイアログボックスで、次のオプションを有効または無効にします。
 - [Update node group version] (ノードグループのバージョンの更新) - カスタム AMI をデプロイした場合、あるいは Amazon EKS に最適化された AMI が現在のクラスターで最新のバージョンである場合には、このオプションは利用できません。
 - [Change launch template version] (起動テンプレートバージョンの変更) - ノードグループがカスタムの起動テンプレートを使用せずにデプロイされている場合、このオプションは利用できません。カスタム起動テンプレートを使用してデプロイされたノードグループの、起動テンプ

レートのバージョンのみ更新できます。ノードグループを更新する [Launch template version] (起動テンプレートバージョン) を選択します。ノードグループがカスタム AMI で設定されている場合は、選択するバージョンも AMI を指定する必要があります。新しいバージョンの起動テンプレートにアップグレードすると、指定した起動テンプレートバージョンの新しい設定と一致するように、すべてのノードがリサイクルされます。

5. [Update strategy] (更新戦略) で、次のいずれかのオプションを選択します。

- [ローリング更新]: このオプションは、クラスターの Pod の中断予算を尊重します。Pod 中断予算の問題により、Amazon EKS がこのノードグループで実行されている Pod を正常にドレインできない場合、更新が失敗します。
- [強制更新]: このオプションは Pod の中断予算を尊重しません。ノードの再起動を強制的に実行することにより、Pod の中断予算の問題に関係なく更新が行われます。

6. [Update] (更新) を選択します。

ノードグループ設定の編集

マネージド型ノードグループの設定の一部を変更できます。

1. [アマゾン EKS コンソール](#)を開きます。
2. 編集するノードグループを含むクラスターを選択します。
3. [Compute] (コンピューティング) タブを選択します。
4. 編集するノードグループを選択し、次に [Edit] (編集) を選択します。
5. (オプション) [Edit Node Group] (ノードグループの編集) ページで以下を実行します。
 - a. ノードグループのスケールリング設定を編集します。
 - [Desired size] (必要なサイズ) - マネージド型ノードグループが保持する必要があるノードの現在の数を指定します。
 - [最小サイズ] - マネージド型ノードグループがスケールインできるノードの最小数を指定します。
 - [最大サイズ]: マネージド型ノードグループがスケールアウトできるノードの最大数を指定します。ノードグループでサポートされるノードの最大数については、「[the section called "Service Quotas"](#)」を参照してください。
 - b. (オプション) ノードグループ内のノードに [Kubernetes ラベル] を追加または削除します。ここに示すラベルは、Amazon EKS で適用したラベルのみです。ここには表示されていない他のラベルがノードに存在する可能性があります。

- c. (オプション) ノードグループ内のノードに [Kubernetes テイント] を追加または削除します。追加されたテイントは、 **NoSchedule** 、 **NoExecute** 、または **PreferNoSchedule** の影響があります。詳細については、「[the section called “GPU ノードにテイントを適用する”](#)」を参照してください。
- d. (オプション) ノードグループリソースに [Tags] (タグ) を追加または削除します。これらのタグは、Amazon EKS ノードグループにのみ適用されます。これらは、Amazon EC2 インスタンスやサブネットなど、ノードグループの他のリソースには伝達されません。
- e. (オプション) ノードグループの更新設定を編集します。[Number] (数値) または [Percentage] (パーセンテージ) のいずれかを選択します。
 - [数値]: 並行して更新できるノードグループ内のノード数を選択して指定します。これらのノードは、更新中は使用できません。
 - [パーセンテージ]: 並行して更新できるノードグループ内のノードの割合を選択して指定します。これらのノードは、更新中は使用できません。ノードグループに多数のノードがある場合に便利です。
- f. 編集が終了したら、[変更の保存] を選択します。

Important

ノードグループ設定を更新する場合、[NodegroupScalingConfig](#) を変更しても Pod 中断予算 (PDB) は考慮されません。[ノードグループの更新](#) プロセス (アップグレードフェーズ中にノードをドレインし、PDB を考慮する) とは異なり、スケーリング設定を更新すると、Auto Scaling グループ (ASG) スケールダウンコールによってノードがすぐに終了します。これは、スケールダウン先のターゲットサイズに関係なく、PDB を考慮せずに発生します。つまり、Amazon EKS マネージドノードグループの `desiredSize` を減らすと、ノードが終了するとすぐに、PDB を尊重せずに Pod が削除されます。

ノードの更新の各フェーズを理解する

Amazon EKS マネージド型ワーカーノードのアップグレード戦略には、次のセクションで説明する 4 つの異なるフェーズがあります。

セットアップフェーズ

セットアップフェーズには、次の手順があります。

1. ノードグループに関連付けられた Auto Scaling グループ用に新しい Amazon EC2 起動テンプレートバージョンを作成します。新しい起動テンプレートバージョンでは、ターゲットの AMI またはカスタム起動テンプレートバージョンを更新に使用します。
2. 最新の起動テンプレートバージョンを使用するように、Auto Scaling グループを更新します。
3. ノードグループの `updateConfig` プロパティを使用して、並列でアップグレードするノードの最大数を決定します。使用不可の最大値には、クォータとして 100 ノードがあります。デフォルト値は 1 ノードです。詳細については、Amazon EKS API リファレンス内の [updateConfig](#) プロパティを参照してください。

スケールアップフェーズ

マネージド型ノードグループ内のノードをアップグレードするとき、アップグレードされたノードは、アップグレードされているノードと同じアベイラビリティーゾーンで起動されます。この配置を保証するために、Amazon EC2 のアベイラビリティーゾーンの再調整を使用します。詳細については、[Amazon EC2 Auto Scaling ユーザーガイド](#) のアベイラビリティーゾーンの再調整を参照してください。この要件を満たすために、マネージド型ノードグループのアベイラビリティーゾーンごとに最大 2 つのインスタンスを起動できます。

スケールアップフェーズには、次の手順があります。

1. Auto Scaling グループの最大サイズと希望のサイズを、次のうちいずれか大きい方だけ増加させます。
 - Auto Scaling グループがデプロイされているアベイラビリティーゾーン数の最大 2 倍。
 - アップグレードができない最大値。

たとえば、ノードグループのアベイラビリティーゾーンが 5 つで、`maxUnavailable` が 1 である場合、アップグレードプロセスで最大 10 個のノードを起動できます。しかし、`maxUnavailable` が 20 (または 10 より大きいいずれかの値) である場合、プロセスにより起動される新しいノードは 20 個です。
2. Auto Scaling グループのスケールリング後、最新の設定を使用するノードがノードグループに存在するかどうかをチェックします。この手順は、次の基準を満たす場合にのみ成功します。
 - ノードが存在するすべてのアベイラビリティーゾーンで、少なくとも 1 つの新しいノードが起動されます。
 - 新しいノードはすべて、Ready 状態である必要があります。
 - 新しいノードには Amazon EKS 適用ラベルが必要です。

これは、通常のノードグループのワーカーノード上の Amazon EKS 適用ラベルです。

- `eks.amazonaws.com/nodegroup-image=$amiName`
- `eks.amazonaws.com/nodegroup=$nodeGroupName`

これは、カスタム起動テンプレートまたは AMI ノードグループのワーカーノード上の Amazon EKS 適用ラベルです。

- `eks.amazonaws.com/nodegroup-image=$amiName`
- `eks.amazonaws.com/nodegroup=$nodeGroupName`
- `eks.amazonaws.com/sourceLaunchTemplateId=$launchTemplateId`
- `eks.amazonaws.com/sourceLaunchTemplateVersion=$launchTemplateVersion`

3. 新しい Pod がスケジュールされないように、ノードをスケジュール不可としてマークします。また、ノードを終了する前にロードバランサーからノードを削除するために、ノードに `node.kubernetes.io/exclude-from-external-load-balancers=true` のラベルを付けます。

このフェーズで `NodeCreationFailure` エラーが発生する既知の原因を次に示します。

アベイラビリティゾーンの容量が不十分です

アベイラビリティゾーンに、リクエストされたインスタンスタイプの容量がない可能性があります。マネージド型ノードグループを作成するときは、複数のインスタンスタイプを設定することをお勧めします。

アカウント内の EC2 インスタンス制限

Service Quotas を使用してアカウントが同時に実行できる Amazon EC2 インスタンスの数を増やす必要がある場合があります。詳細については、Linux インスタンス向け Amazon Elastic Compute Cloud ユーザーガイドの [EC2 の Service Quotas](#) を参照してください。

カスタムユーザーデータ

カスタムユーザーデータは、ブートストラッププロセスを中断することがあります。このシナリオでは、ノードで `kubelet` が起動しないか、ノードで想定される Amazon EKS ラベルが取得されない可能性があります。詳細については、「[the section called “AMI を指定する”](#)」を参照してください。

ノードを異常な状態または準備ができていない状態にする変更

ノードのディスク負荷、メモリ負荷、および同様の条件により、ノードが Ready 状態にならない可能性があります。

各ノードのブートストラップにかかる時間は通常 15 分以内

ブートストラップしてクラスターに加わるまでに 15 分以上かかるノードがあると、アップグレードがタイムアウトします。この時間は、新しいノードが必要になってからクラスターに加わるまで、新しいノードのブートストラップに要した合計時間です。マネージドノードグループをアップグレードする場合は、Auto Scaling グループのサイズが大きくなると、すぐに時間の測定が始まります。

アップグレードフェーズ

アップグレードフェーズの動作には 2 通りの方法があり、更新戦略によって異なります。デフォルトと最小という 2 つの更新戦略があります。

通常は、デフォルト戦略をお勧めします。古いノードを終了する前に新しいノードを作成するという戦略で、アップグレードフェーズ中も使用可能な容量が維持されます。最小は、リソースやコスト (例えば、GPU などのハードウェアアクセラレーター) が限られている場合に便利な戦略です。新しいノードを作成する前に古いノードを終了するため、合計容量が設定しておいた数量を超えることはありません。

デフォルト更新戦略のステップは次のとおりです。

1. Auto Scaling グループ内のノード数を (必要な数だけ) 増やして、ノードグループに追加でノードを作成します。
2. ノードグループのために設定されている使用不可の最大数を上限として、アップグレードが必要なノードをランダムに選択します。
3. ノードから Pod をドレインします。Pod が 15 分以内にノードを離れず、強制フラグがない場合は、PodEvictionFailure というエラーが表示されて、アップグレードフェーズは失敗します。この状況になった場合、update-nodegroup-version リクエストで強制フラグを適用して Pod を削除できます。
4. すべての Pod が削除されたらノードを遮断し、60 秒間待機します。これは、サービスコントローラーがこのノードに新しくリクエストを送信しないようにするためと、アクティブなノードのリストからこのノードを削除するために行われます。
5. 遮断されたノードの Auto Scaling グループに終了リクエストを送信します。

6. 以前のバージョンの起動テンプレートでデプロイされたノードグループ内にノードが存在しなくなるまで、以前のアップグレード手順を繰り返します。

最小更新戦略のステップは次のとおりです。

1. ノードグループのすべてのノードを最初に遮断して、サービスコントローラーがこれらのノードに新しいリクエストを送信しないようにします。
2. ノードグループのために設定されている使用不可の最大数を上限として、アップグレードが必要なノードをランダムに選択します。
3. 選択したノードからポッドをドレインします。Pod が 15 分以内にノードを離れず、強制フラグがない場合は、PodEvictionFailure というエラーが表示されて、アップグレードフェーズは失敗します。この状況になった場合、update-nodegroup-version リクエストで強制フラグを適用して Pod を削除できます。
4. すべてのポッドを削除し 60 秒間待機すると、終了リクエストが、選択したノードの Auto Scaling グループに送信されます。Auto Scaling グループは、不足分の容量を補うために (選択したノードと同数の) 新しいノードを作成します。
5. 以前のバージョンの起動テンプレートでデプロイされたノードグループ内にノードが存在しなくなるまで、以前のアップグレード手順を繰り返します。

アップグレードフェーズ中の PodEvictionFailure エラー

このフェーズで PodEvictionFailure エラーが発生する既知の原因を次に示します。

アグレッシブ PDB

Pod にアグレッシブ PDB が定義されているか、同じ Pod を指す複数の PDB が存在します。

すべてのテイントを許容するデプロイ

すべての Pod が削除されたら、前のステップでノードが[テイント](#)されているため、ノードは空になるはずですが、デプロイがすべてのテイントを許容する場合、ノードが空でない可能性が高くなり、Pod のエビクションが失敗します。

スケールダウンフェーズ

スケールダウンフェーズでは、Auto Scaling グループの最大サイズと希望するサイズが 1 ずつ減り、更新が開始される前の値に戻ります。

ワークフローのスケールダウンフェーズ中に Cluster Autoscaler がノードグループをスケールアップしているとアップグレードワークフローが判断した場合、ノードグループを元のサイズに戻すことなく、すぐに終了します。

起動テンプレートを使用してマネージドノードをカスタマイズする

自分の起動テンプレートを使用することで、マネージド型ノードをデプロイでき、最高レベルのカスタマイズを実現できます。起動テンプレートを使用すると、次のような機能が可能となります。

- ノードをデプロイする時に、追加の [kubenet](#) 引数などのブートストラップ引数を提供できます。
- ノードに割り当てられた IP アドレスとは異なる CIDR ブロックから、IP アドレスを Pod に割り当てます。
- 独自のカスタム AMI をノードにデプロイします。
- 独自のカスタム CNI をノードにデプロイします。

最初にマネージドノードグループを作成するときに独自の起動テンプレートを指定する場合にも、後で高い柔軟性を得ることができます。自分の起動テンプレートを使用してマネージド型ノードグループをデプロイする限り、同じ起動テンプレートの異なるバージョンとして繰り返し更新できます。ノードグループを別のバージョンの起動テンプレートに更新すると、指定した起動テンプレートバージョンの新しい設定と一致するように、グループ内のすべてのノードがリサイクルされます。

マネージド型ノードグループは常に Amazon EC2 Auto Scaling グループで使用する起動テンプレートでデプロイされます。起動テンプレートを指定しない場合、Amazon EKS API はアカウントのデフォルト値を使用して起動テンプレートを自動的に作成します。ただし、自動生成された起動テンプレートを変更することは推奨しません。さらに、カスタム起動テンプレートを使用していない既存のノードグループは直接更新できません。代わりに、カスタム起動テンプレートを使用して、新しいノードグループを作成する必要があります。

起動テンプレート設定の基本

AWS Management Console、AWS CLI、または AWS SDK を使用して、Amazon EC2 Auto Scaling 起動テンプレートを作成できます。詳細については「Amazon EC2 Auto Scaling ユーザーガイド」の「[自動スケーリング グループの起動テンプレートを作成する](#)」を参照してください。起動テンプレートの一部の設定はマネージド型ノードで使われている設定と似ています。起動テンプレートを使用してノードグループをデプロイまたは更新する場合、一部の設定はノードグループ設定または起動テンプレートのいずれかで指定する必要があります。両方の場所で設定を指定しないでください。存在してはいけない場所に設定が存在する場合、ノードグループの作成や更新などの操作は失敗します。

次の表に、起動テンプレートで禁止されている設定を示します。また、マネージド型ノードグループの設定に必要な同様の設定がある場合はその設定も示します。リスト化されている設定はコンソールに表示される設定です。AWS CLI と SDK では類似するが異なる名前がある場合があります。

起動テンプレート — 禁止	Amazon EKS ノードグループ設定
[ネットワークインターフェイス] ([ネットワークインターフェイスを追加]) の下の [サブネット]	[Specify networking] (ネットワーキングを指定) ページの [Node group network configuration] (ノードグループのネットワーク設定) の下の [Subnets] (サブネット)
[高度な詳細] の下の [IAM インスタンスプロファイル]	[Configure Node group] (ノードグループを設定) ページの [Node group configuration] (ノードグループ設定) の下の [Node IAM role] (ノード IAM 役割)
[高度な詳細] の下の [シャットダウン動作] および [停止 – 休止動作]。起動テンプレートのどちらの設定でも、[起動テンプレート設定に含めない] のデフォルトを保持します。	同等のものではありません。Amazon EKS は自動スケーリンググループではなく、インスタンスのライフサイクルを管理する必要があります。

次の表に、マネージド型ノードグループ構成で禁止される設定を示します。また、起動テンプレートに必要な同様の設定がある場合はその設定も示します。リスト化されている設定はコンソールに表示される設定です。AWS CLI と SDK とで名前が似ている場合があります。

Amazon EKS ノードグループ設定 — 禁止	起動テンプレート
(起動テンプレートでカスタム AMI を指定した場合のみ) [Set compute and scaling configuration] (コンピューティングとスケーリングの設定を実行) ページの [Node Group compute configuration] (ノードグループのコンピューティング設定) の下の [AMI type] (AMI タイプ) — [Specified in launch template] (起動テンプレートで指定) と、指定した AMI ID がコンソールに表示されます。	[Launch template contents] (起動テンプレートのコンテンツ) での [Application and OS Images (Amazon Machine Image)] (アプリケーションと OS のイメージ (Amazon マシンイメージ)) - 次のいずれかの要件がある場合は ID を指定する必要があります。 カスタム AMI の使用。 AMI を指定する に記載されている要件を満たさない AMI を指定

Amazon EKS ノードグループ設定 — 禁止	起動テンプレート
起動テンプレートで [Application and OS Images (Amazon Machine Image)] (アプリケーションと OS のイメージ (Amazon マシンイメージ)) が指定されていない場合はノードグループ設定で AMI を選択できます。	すると、ノードグループのデプロイが失敗します。 • Amazon EKS に最適化された AMI に含まれる <code>bootstrap.sh</code> ファイルに引数を提供するためのユーザーデータを提供する必要がある。インスタンスでは、Pod に多数の IP アドレスを割り当てたり、インスタンスとは異なる CIDR ブロックから Pod に IP アドレスを割り当てたりできます。さらに、アウトバウンドのインターネットアクセスのないプライベートクラスターをデプロイすることもできます。詳細については、以下の各トピックを参照してください。 • プレフィックスを使用して Amazon EKS ノードに割り当てる IP アドレスを増やす • カスタムネットワーキングを使用して代替サブネットに Pod をデプロイする • インターネットアクセスが制限されたプライベートクラスターをデプロイする • AMI を指定する
[Set compute and scaling configuration] (コンピューティングとスケーリングの設定を実行) ページの [Node Group compute configuration] (ノードグループのコンピューティング設定) の下の [Disk size] (ディスクサイズ) — [Specified in launch template] (起動テンプレートで指定) がコンソールに表示されます。	[ストレージ (ボリューム)] ([新しいボリュームの追加]) の下の [サイズ]。これを起動テンプレートで指定する必要があります。

Amazon EKS ノードグループ設定 — 禁止	起動テンプレート
[Specify Networking] (ネットワーキングを指定) ページの [Node group configuration] (ノードグループ設定) の下の [SSH key pair] (SSH キーペア) — コンソールに起動テンプレートで指定したキーまたは [Not specified in launch template] (起動テンプレートで指定されていません) が表示されます。	[キーペア (ログイン)] の下の [キーペア名]。
起動テンプレートを使用する場合はリモートアクセスを許可するソースセキュリティグループを指定できません。	インスタンスの場合、[ネットワーク設定] の下の [セキュリティグループ]、または [ネットワークインターフェイス] ([ネットワークインターフェイスを追加]) の下の [セキュリティグループ]。両方設定することはできません。詳細については「the section called “カスタムセキュリティグループを使用する”」を参照してください。

Note

- 起動テンプレートを使用してノードグループをデプロイする場合は起動テンプレート内の [起動テンプレートのコンテンツ] で 0 または 1 の [インスタンスタイプ] を指定します。またはコンソールの [コンピューティングとスケーリングの構成を設定する] ページにある [インスタンスタイプ] で 0 から 20 までのインスタンスタイプを指定できます。または Amazon EKS API を使用する他のツールを使用して行うこともできます。起動テンプレートでインスタンスタイプを指定し、その起動テンプレートを使用してノードグループをデプロイする場合、コンソールや、Amazon EKS API を使用する他のツールを使用してインスタンスタイプを指定することはできません。起動テンプレートやコンソール、または Amazon EKS API を使用する他のツールを使用してインスタンスタイプを指定しない場合は t3.medium インスタンスタイプが使用されます。ノードグループがスポットキャパシティータイプを使用している場合はコンソールを使用して複数のインスタンスタイプを指定することをお勧めします。詳細については「[the section called “マネージド型ノードグループのキャパシティータイプ”](#)」を参照してください。

- ノードグループにデプロイするコンテナが、インスタンスメタデータサービスバージョン 2 を使用している場合は起動テンプレートの [メタデータレスポンスのホップ制限] を 2 に設定してください。詳細については「Amazon EC2 ユーザーガイド」の「[Instance metadata and user data](#)」(インスタンスメタデータとユーザーデータ) を参照してください。カスタム起動テンプレートを使用せずにマネージド型ノードグループをデプロイする場合、この値はデフォルトの起動テンプレートのノードグループに対して自動的に設定されます。
- 起動テンプレートでは、インスタンスタイプを柔軟に選択できるようにする InstanceRequirements 機能はサポートされていません。

Amazon EC2 インスタンスへのタグ付け

起動テンプレートの TagSpecification パラメータを使用して、ノードグループの Amazon EC2 インスタンスに適用するタグを指定します。CreateNodegroup または UpdateNodegroupVersion API を呼び出す IAM エンティティには ec2:RunInstances および ec2:CreateTags へのアクセス許可が必要です。また、起動テンプレートにタグが追加される必要があります。

カスタムセキュリティグループを使用する

起動テンプレートでカスタムの Amazon EC2 [セキュリティグループ](#) を指定し、ノードグループ内のインスタンスに適用できます。これはインスタンスレベルのセキュリティグループのパラメータ内か、ネットワークインターフェイス設定のパラメータの一部として指定できます。しかし、インスタンスレベルとネットワークインターフェイス、両方のセキュリティグループを指定して、起動テンプレートを作成することはできません。マネージドノード型グループでカスタムセキュリティグループを使用する際に適用される、次の条件を考慮してください。

- AWS Management Console を使用する場合、Amazon EKS では単一のネットワークインターフェイス仕様の起動テンプレートのみ使用できます。
- デフォルトでは Amazon EKS は [クラスターセキュリティグループ](#) をノードグループ内のインスタンスに追加して、ノードとコントロールプレーンとの間の通信を容易にします。前述のいずれかのオプションを使用して、起動テンプレートでカスタムセキュリティグループを指定した場合、Amazon EKS はクラスターセキュリティグループを追加しません。したがって、セキュリティグループのインバウンドルールとアウトバウンドルールで、クラスターのエンドポイントとの通信が有効になっていることを確認する必要があります。セキュリティグループのルールが正しく

ない場合、ワーカーノードはクラスターに参加できません。セキュリティグループルールの詳細については[the section called “セキュリティグループの要件”](#)を参照してください。

- ノードグループ内のインスタンスへの SSH アクセスが必要な場合はそのアクセスを許可するセキュリティグループを含めてください。

Amazon EC2 ユーザーデータ

起動テンプレートにはカスタムユーザーデータのセクションが含まれています。このセクションでは、個々のカスタム AMI を手動で作成しなくても、ノードグループの構成設定を指定できます。Bottlerocket で使用できる設定の詳細については、GitHub で [Using user data](#) を参照してください。

cloud-init を使用すると、インスタンス起動時に起動テンプレート内の Amazon EC2 ユーザーデータを提供できます。詳細については「[cloud-init ドキュメント](#)」を参照してください。ユーザーデータを使用すると、一般的な設定操作を実行できます。これには次の操作が含まれます。

- [ユーザーまたはグループを含む](#)
- [パッケージのインストール](#)

マネージド型ノードグループで使用される起動テンプレートの Amazon EC2 ユーザーデータは、Amazon Linux AMI の場合は、[MIME マルチパートアーカイブ](#)、Bottlerocket AMI の場合は TOML の形式であることが必要です。これはユーザーデータが、ノードがクラスターに参加するために必要な Amazon EKS ユーザーデータにマージされるためです。kubelet を起動または変更するコマンドをユーザーデータに指定しないでください。これは Amazon EKS によってマージされたユーザーデータの一部として実行されます。ノードへのラベル設定などの一部の kubelet パラメータはマネージド型ノードグループ API 経由で直接設定できます。

Note

手動での起動や、カスタムの設定パラメータの受け渡しなど、高度な kubelet のカスタマイズについては[the section called “AMI を指定する”](#)を参照してください。起動テンプレートでカスタム AMI ID が指定されている場合、Amazon EKS はユーザーデータをマージしません。

次の詳細はユーザーデータセクションについて説明しています。

Amazon Linux 2 ユーザーデータ

複数のユーザーデータブロックと単一の MIME マルチパートファイルを組み合わせることができます。例えば、カスタムパッケージをインストールするユーザーデータのシェルスクリプトを、Docker デーモンを設定するクラウドブートフックに組み合わせることができます。MIME マルチパートファイルには次のコンポーネントが含まれます。

- コンテンツタイプとパートバウンダリの宣言: `Content-Type: multipart/mixed; boundary=="MYBOUNDARY=="`
- MIME バージョンの宣言: `MIME-Version: 1.0`
- 次のコンポーネントを含む 1 つ以上のユーザーデータブロック:
 - ユーザーデータブロックの始まりを示す開始境界: `--==MYBOUNDARY==`
 - ブロックのコンテンツの種類の宣言: `Content-Type: text/cloud-config; charset="us-ascii"`。コンテンツタイプの詳細については[cloud-init](#) のドキュメントを参照してください。
 - ユーザーデータのコンテンツ (例えば、シェルコマンドや `cloud-init` ディレクティブのリスト)。
 - MIME マルチパートファイルの終わりを示す、終了境界: `--==MYBOUNDARY==--`

自分で MIME マルチパートファイルを作成するときに使用できる例は次の通りです。

```
MIME-Version: 1.0
Content-Type: multipart/mixed; boundary=="MYBOUNDARY=="

--==MYBOUNDARY==
Content-Type: text/x-shellscript; charset="us-ascii"

#!/bin/bash
echo "Running custom user data script"

--==MYBOUNDARY==--
```

Amazon Linux 2023 ユーザーデータ

Amazon Linux 2023 (AL2023) ではYAML 設定スキーマを使用する新しいノード初期化プロセス `nodeadm` が導入されています。セルフマネージド型ノードグループまたは起動テンプレートを持つ AMI を使用している場合は新しいノードグループの作成時に追加のクラスターメタデータを明示的に指定する必要があります。最低限必要なパラメータの例を以下に示します。ここで、`apiServerEndpoint`、`certificateAuthority`、サービスの `cidr` が必要になります。

```

---
apiVersion: node.eks.aws/v1alpha1
kind: NodeConfig
spec:
  cluster:
    name: my-cluster
    apiServerEndpoint: https://example.com
    certificateAuthority: Y2VydGlmaWNhdGVBdXRob3JpdHk=
    cidr: 10.100.0.0/16

```

通常、この設定はユーザーデータにそのまま設定するか、MIME マルチパートドキュメントに埋め込まれます。

```

MIME-Version: 1.0
Content-Type: multipart/mixed; boundary="BOUNDARY"

--BOUNDARY
Content-Type: application/node.eks.aws

---
apiVersion: node.eks.aws/v1alpha1
kind: NodeConfig spec: [...]

--BOUNDARY--

```

AL2 ではこれらのパラメータからのメタデータは Amazon EKS DescribeCluster API コールから検出されていました。AL2023 ではノードの大規模なスケールアップ中に API コールによってスロットリングが発生するリスクがあるため、この動作が変更されました。この変更は起動テンプレートのないマネージド型ノードグループを使用している場合や、Karpenter を使用している場合には影響しません。詳細については、「Amazon EKS API リファレンス」の `certificateAuthority`、サービスの `cidr`、[DescribeCluster](#) を参照してください。

次に、AL2023 ユーザーデータの詳しい例を示します。ノードをカスタマイズするシェルスクリプト (パッケージのインストールやコンテナイメージの事前キャッシュなど) を必要な `nodeadm` 設定と結合しています。この例では、よく使用されるカスタマイズを示しています。* 追加でシステムパッケージをインストールする * コンテナイメージを事前キャッシュしてポッドの起動時間を改善する * HTTP プロキシ設定を設定する * ノードのラベル付けに関する `kubelet` フラグを設定する

```

MIME-Version: 1.0

```

```
Content-Type: multipart/mixed; boundary="BOUNDARY"

--BOUNDARY
Content-Type: text/x-shellscript; charset="us-ascii"

#!/bin/bash
set -o errexit
set -o pipefail
set -o nounset

# Install additional packages
yum install -y htop jq iptables-services

# Pre-cache commonly used container images
nohup docker pull public.ecr.aws/eks-distro/kubernetes/pause:3.2 &

# Configure HTTP proxy if needed
cat > /etc/profile.d/http-proxy.sh << 'EOF'
export HTTP_PROXY="http://proxy.example.com:3128"
export HTTPS_PROXY="http://proxy.example.com:3128"
export NO_PROXY="localhost,127.0.0.1,169.254.169.254,.internal"
EOF

--BOUNDARY
Content-Type: application/node.eks.aws

apiVersion: node.eks.aws/v1alpha1
kind: NodeConfig
spec:
  cluster:
    name: my-cluster
    apiServerEndpoint: https://example.com
    certificateAuthority: Y2VydGlmawNhdGVBdXRob3JpdHk=
    cidr: 10.100.0.0/16
  kubelet:
    config:
      clusterDNS:
        - 10.100.0.10
    flags:
      - --node-labels=app=my-app,environment=production

--BOUNDARY--
```


Bottlerocket ユーザーデータ

Bottlerocket はユーザーデータを TOML 形式で構造化しています。Amazon EKS が提供するユーザーデータとマージするユーザーデータを提供できます。たとえば、追加の kubelet 設定を指定できます。

```
[settings.kubernetes.system-reserved]
cpu = "10m"
memory = "100Mi"
ephemeral-storage= "1Gi"
```

サポートされる設定について詳しくは[Bottlerocket のドキュメント](#)を参照してください。ユーザーデータにノードラベルと[テイント](#)を設定できます。ただし、代わりにノードグループ内にこれらを設定することをお勧めします。この場合、Amazon EKS によりこれらの設定が適用されます。

ユーザーデータをマージしても、書式設定は保持されませんが、コンテンツは同じままです。ユーザーデータに指定した設定はAmazon EKS によって構成された設定よりも優先されます。したがって、settings.kubernetes.max-pods または settings.kubernetes.cluster-dns-ip を設定した場合、ユーザーデータのこれらの値がノードに適用されます。

Amazon EKS で、有効な TOML がすべてサポートされるわけではありません。以下はサポートされていない既知の形式の一覧です。

- 引用符で囲まれたキー内の引用符: 'quoted "value"' = "value"
- 値内のエスケープされた引用符: str = "I'm a string. \"You can quote me\""
- 浮動小数点と整数の混在: numbers = [0.1, 0.2, 0.5, 1, 2, 5]
- 配列内の混合型: contributors = ["[foo@example.com](#)", { name = "Baz", email = "[baz@example.com](#)" }]
- 引用符付きキーを含む括弧で囲まれたヘッダー: [foo."bar.baz"]

Windows ユーザーデータ

Windows ユーザーデータでは、PowerShell コマンドを使用します。マネージド型ノードグループを作成すると、カスタムユーザーデータが Amazon EKS マネージドユーザーデータと結合されます。まず PowerShell コマンドが表示され、その後にマネージドユーザーデータコマンドが続きます。そのいずれも 1 つの <powershell></powershell> タグ内にあります。

⚠ Important

Windows ノードグループを作成すると、Amazon EKS は Linux ベースのノードがクラスターに参加できるように aws-auth ConfigMap を更新します。このサービスは、Windows AMI に対するアクセス許可を自動的に設定しません。Windows ノードを使用している場合は、アクセスエントリ API を利用するか、aws-auth ConfigMap を直接更新することで、アクセスを管理する必要があります。詳細については、「[the section called “Windows サポートを有効にする”](#)」を参照してください。

i Note

起動テンプレートに AMI ID が指定されていない場合はユーザーデータで Windows Amazon EKS ブートストラップスクリプトを使用して Amazon EKS を設定しないでください。

ユーザーデータの例は次のとおりです。

```
<powershell>
Write-Host "Running custom user data script"
</powershell>
```

AMI を指定する

以下のいずれかの要件がある場合は起動テンプレートの ImageId フィールドで AMI ID を指定します。追加情報については要件を選択してください。

Amazon EKS に最適化された Linux/Bottlerocket AMI に含まれる **bootstrap.sh** ファイルに引数を渡すためのユーザーデータを提供する

ブートストラップとはインスタンスの起動時に実行できるコマンドの追加を表す用語です。例えば、ブートストラップでは追加の [kubelet](#) 引数を使用できます。起動テンプレートを指定せずに、eksctl を使用して引数を bootstrap.sh スクリプトに渡すことができます。または起動テンプレートのユーザーデータセクションに情報を指定することでこれを行うことができます。

起動テンプレートを指定しない eksctl

`my-nodegroup.yaml` という名前のファイルを以下の内容で作成します。各#####は独自の値に置き換えます。--apiserver-endpoint、--b64-cluster-ca、および --dns-cluster-ip 引数はオプションです。しかし、これらを定義すると、bootstrap.sh スクリプトによる describeCluster 呼び出しを避けることができます。これはプライベートクラスターのセットアップや、ノードを頻繁にスケールインおよびスケールアウトするクラスターで役立ちます。bootstrap.sh スクリプトの詳細については、GitHub で「[bootstrap.sh](#)」ファイルを参照してください。

- 必須となる引数はクラスター名 (#####) のみです。
- ami-`1234567890abcdef0` に最適化された AMI ID を取得するには、次のセクションを参照してください。
 - [推奨 Amazon Linux AMI ID を取得する](#)
 - [推奨 Bottlerocket AMI ID を取得する](#)
 - [推奨 Microsoft Windows AMI ID を取得する](#)
- クラスターの ##### を取得するには次のコマンドを実行してください。

```
aws eks describe-cluster --query "cluster.certificateAuthority.data" --output text
--name my-cluster --region region-code
```

- クラスターの `api-server-endpoint` を取得するには次のコマンドを実行してください。

```
aws eks describe-cluster --query "cluster.endpoint" --output text --name my-
cluster --region region-code
```

- 最終的に、--dns-cluster-ip の値はサービス CIDR を示す .10 となります。クラスターの `service-cidr` を取得するには次のコマンドを実行してください。例えば、戻り値が ipv4 10.100.0.0/16 であれば、自分の値は `10.100.0.10` です。

```
aws eks describe-cluster --query "cluster.kubernetesNetworkConfig.serviceIpv4Cidr"
--output text --name my-cluster --region region-code
```

- この例ではカスタムの max-pods 値を設定するために kubelet 引数を指定します。その際、Amazon EKS 最適化 AMI に含まれている bootstrap.sh スクリプトを使用します。ノードグループ名は 63 文字以下である必要があります。先頭は文字または数字でなければなりません。残りの文字にはハイフンおよびアンダースコアを含めることもできます。`my-max-pods-value` を選択する方法については「[the section called “各 Amazon EC2 インスタンスタイプの Amazon EKS の推奨最大ポッド数”](#)」を参照してください。

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

metadata:
  name: my-cluster
  region: region-code

managedNodeGroups:
- name: my-nodegroup
  ami: ami-1234567890abcdef0
  instanceType: m5.large
  privateNetworking: true
  disableIMDSv1: true
  labels: { x86-al2-specified-mng }
  overrideBootstrapCommand: |
    #!/bin/bash
    /etc/eks/bootstrap.sh my-cluster \
      --b64-cluster-ca certificate-authority \
      --apiserver-endpoint api-server-endpoint \
      --dns-cluster-ip service-cidr.10 \
      --kubelet-extra-args '--max-pods=my-max-pods-value' \
      --use-max-pods false
```

利用可能な eksctl config ファイルオプションについては「eksctl ドキュメント」の「[Config file schema](#)」(Config ファイルスキーマ)を参照してください。eksctl ユーティリティは引き続き起動テンプレートを作成し、config ファイルに提供したデータを使用してユーザーデータを設定します。

次のコマンドを使用して、ノードグループを作成します。

```
eksctl create nodegroup --config-file=my-nodegroup.yaml
```

起動テンプレートのユーザーデータ

起動テンプレートのユーザーデータセクションで次の情報を指定します。各#####は独自の値に置き換えます。--apiserver-endpoint、--b64-cluster-ca、および --dns-cluster-ip 引数はオプションです。しかし、これらを定義すると、bootstrap.sh スクリプトによる describeCluster 呼び出しを避けることができます。これはプライベートクラスターのセットアップや、ノードを頻繁にスケールインおよびスケールアウトするクラスターで役立ちま

す。bootstrap.sh スクリプトの詳細については、GitHub で「[bootstrap.sh](#)」ファイルを参照してください。

- 必須となる引数はクラスター名 (#####) のみです。
- クラスターの ##### を取得するには次のコマンドを実行してください。

```
aws eks describe-cluster --query "cluster.certificateAuthority.data" --output text
--name my-cluster --region region-code
```

- クラスターの *api-server-endpoint* を取得するには次のコマンドを実行してください。

```
aws eks describe-cluster --query "cluster.endpoint" --output text --name my-
cluster --region region-code
```

- 最終的に、--dns-cluster-ip の値はサービス CIDR を示す .10 となります。クラスターの *service-cidr* を取得するには次のコマンドを実行してください。例えば、戻り値が ipv4 10.100.0.0/16 であれば、自分の値は *10.100.0.10* です。

```
aws eks describe-cluster --query "cluster.kubernetesNetworkConfig.serviceIpv4Cidr"
--output text --name my-cluster --region region-code
```

- この例ではカスタムの max-pods 値を設定するために kubelet 引数を指定します。その際、Amazon EKS 最適化 AMI に含まれている bootstrap.sh スクリプトを使用します。*my-max-pods-value* を選択する方法については「[the section called “各 Amazon EC2 インスタンスタイプの Amazon EKS の推奨最大ポッド数”](#)」を参照してください。

```
MIME-Version: 1.0
Content-Type: multipart/mixed; boundary=="MYBOUNDARY=="

--MYBOUNDARY==
Content-Type: text/x-shellscript; charset="us-ascii"

#!/bin/bash
set -ex
/etc/eks/bootstrap.sh my-cluster \
  --b64-cluster-ca certificate-authority \
  --apiserver-endpoint api-server-endpoint \
  --dns-cluster-ip service-cidr.10 \
  --kubelet-extra-args '--max-pods=my-max-pods-value' \
  --use-max-pods false
```

```
--==MYBOUNDARY==--
```

Amazon EKS に最適化された Windows AMI に含まれる **Start-EKSBootstrap.ps1** ファイルに引数を渡すためのユーザーデータを提供する

ブートストラップとはインスタンスの起動時に実行できるコマンドの追加を表す用語です。起動テンプレートを指定せずに、eksctl を使用して引数を Start-EKSBootstrap.ps1 スクリプトに渡すことができます。または起動テンプレートのユーザーデータセクションに情報を指定することでこれを行うことができます。

カスタム Windows AMI ID を指定する場合は、次の考慮事項に留意してください。

- 起動テンプレートを使用し、必要なブートストラップコマンドをユーザーデータセクションに入力する必要があります。目的の Windows ID を取得するには、「[最適化された Windows AMI を使用してノードを作成する](#)」のテーブルを使用できます。
- いくつかの制限と条件があります。例えば、eks:kube-proxy-windows を AWS IAM 認証設定マップに追加する必要があります。詳細については「[the section called “AMI ID を指定する場合の制限と条件”](#)」を参照してください。

起動テンプレートのユーザーデータセクションで次の情報を指定します。各#####は独自の値に置き換えます。-APIServerEndpoint、-Base64ClusterCA、および -DNSClusterIP 引数はオプションです。しかし、これらを定義すると、Start-EKSBootstrap.ps1 スクリプトによる describeCluster 呼び出しを避けることができます。

- 必須となる引数はクラスター名 (#####) のみです。
- クラスターの ##### を取得するには次のコマンドを実行してください。

```
aws eks describe-cluster --query "cluster.certificateAuthority.data" --output text --name my-cluster --region region-code
```

- クラスターの *api-server-endpoint* を取得するには次のコマンドを実行してください。

```
aws eks describe-cluster --query "cluster.endpoint" --output text --name my-cluster --region region-code
```

- 最終的に、--dns-cluster-ip の値はサービス CIDR を示す .10 となります。クラスターの *service-cidr* を取得するには次のコマンドを実行してください。例えば、戻り値が ipv4 10.100.0.0/16 であれば、自分の値は *10.100.0.10* です。

```
aws eks describe-cluster --query "cluster.kubernetesNetworkConfig.serviceIpv4Cidr" --  
output text --name my-cluster --region region-code
```

- その他の引数については「[the section called “ブートストラップスクリプトの設定パラメータ”](#)」を参照してください。

Note

カスタムサービス CIDR を使用している場合は `-ServiceCIDR` パラメータを使用して指定する必要があります。そうしないと、クラスター内の Pod の DNS 解決が失敗します。

```
<powershell>  
[string]$EKSBootstrapScriptFile = "$env:ProgramFiles\Amazon\EKS\Start-EKSBootstrap.ps1"  
& $EKSBootstrapScriptFile -EKSClusterName my-cluster `   
  -Base64ClusterCA certificate-authority `   
  -APIServerEndpoint api-server-endpoint `   
  -DNSClusterIP service-cidr.10  
</powershell>
```

特定のセキュリティ、コンプライアンス、または社内的なポリシー要件のために、カスタム AMI を実行する

詳細については「Amazon EC2 ユーザーガイド」の「[Amazon マシンイメージ \(AMI\)](#)」を参照してください。Amazon EKS AMI ビルド仕様には Amazon Linux をベースにしたカスタム Amazon EKS AMI を構築するためのリソースと設定スクリプトが含まれています。詳細については、GitHub の「[Amazon EKS AMI ビルド仕様](#)」を参照してください。他のオペレーティングシステムがインストールされたカスタム AMI を作成するには、GitHub の「[Amazon EKS Sample Custom AMIs](#)」を参照してください。

マネージドノードグループで使用される起動テンプレートで AMI ID に動的パラメータ参照を使用することはできません。

Important

AMI を指定する際、Amazon EKS ではユーザーデータをマージしません。代わりに、ノードのクラスターへの参加に必要な bootstrap コマンドを、ユーザーが提供する必要があります。

ます。ノードがクラスターに参加できない場合、Amazon EKS CreateNodegroup および UpdateNodegroupVersion アクションも失敗します。

AMI ID を指定する場合の制限と条件

以下に、マネージド型ノードグループで AMI ID を指定する際の制限と条件を示します。

- 起動テンプレートで AMI ID を指定する場合と、AMI ID を指定しない場合は新しいノードグループを作成して切り替える必要があります。
- 新しい AMI バージョンが利用可能になっても、コンソールでは通知を表示しません。ノードグループを新しい AMI バージョンに更新するには更新された AMI ID で新しいバージョンの起動テンプレートを作成する必要があります。次に、新しい起動テンプレートバージョンでノードグループを更新する必要があります。
- AMI ID を指定した場合はAPI の次のフィールドを設定することはできません。
 - amiType
 - releaseVersion
 - version
- AMI ID を指定する場合、API で設定されたすべての taints は非同期に適用されます。ノードがクラスターに参加する前にテイントを適用するには--register-with-taints コマンドラインフラグを使用してユーザーデータ内の kubelet にテイントを渡す必要があります。詳細については、Kubernetes ドキュメントの「[kubelet](#)」を参照してください。
- Windows マネージド型ノードグループのカスタム AMI ID を指定するときは、AWS IAM オーセンティケーター設定マップに eks:kube-proxy-windows を追加します。これはDNS が正しく機能するために必要です。

a. AWS IAM 認証 設定マップを編集用に関きます。

```
kubectl edit -n kube-system cm aws-auth
```

b. このエントリを、Windows ノードに関連付けられた各 rolearn の下にある groups リストに追加します。設定マップは [aws-auth-cm-windows.yaml](#) に似ているはずです。

```
- eks:kube-proxy-windows
```

c. ファイルを保存し、テキストエディタを終了します。

クラスターからマネージドノードグループを削除する

このトピックでは、Amazon EKS マネージド型ノードグループを削除する方法について説明します。マネージド型ノードグループを削除すると、まず Amazon EKS が Auto Scaling グループの最小サイズ、最大サイズ、および必要なサイズをゼロに設定します。これにより、ノードグループがスケールダウンされます。

各インスタンスが終了する前に、Amazon EKS はそのノードから Pod を排出するシグナルを送信します。数分後に Pod が排出されない場合、Amazon EKS により Auto Scaling がインスタンスの終了を継続します。すべてのインスタンスが終了すると、Auto Scaling グループは削除されます。

Important

クラスター内の、他のマネージド型ノードグループで使用されていないノード IAM ロールを使用している、マネージド型ノードグループを削除すると、そのロールは aws-auth ConfigMap から削除されます。クラスター内のいずれかのセルフマネージド型ノードグループが同じノード IAM ロールを使用している場合、セルフマネージド型ノードは NotReady ステータスに移行します。さらに、クラスター操作も中断されます。クラスターのプラットフォームバージョンが「[EKS アクセスエントリを使用して Kubernetes へのアクセスを IAM ユーザーに許可する](#)」の前提条件セクションに記載されている最低バージョン以上である場合、セルフマネージドノードグループにのみ使用しているロールのマッピングを追加するには、「[the section called “アクセスエントリを作成する”](#)」を参照してください。プラットフォームバージョンがアクセスエントリに必要な最小バージョンより前の場合は、エントリを aws-auth ConfigMap に追加し直すことができます。詳細については、ターミナルに `eksctl create iamidentitymapping --help` を入力してください。

マネージド型ノードグループを削除するには、以下を使用します。

- [the section called “eksctl”](#)
- [the section called “AWS Management Console”](#)
- [the section called “AWS CLI”](#)

eksctl

マネージド型ノードグループを **eksctl** を使用して削除する

次のコマンドを入力します。各####は独自の値に置き換えます。

```
eksctl delete nodegroup \  
  --cluster my-cluster \  
  --name my-mng \  
  --region region-code
```

その他のオプションについては、eksctl ドキュメントの「[ノードグループの削除と排出](#)」を参照してください。

AWS Management Console

マネージド型ノードグループを AWS Management Console を使用して削除する

1. [アマゾン EKS コンソール](#)を開きます。
2. [クラスター] ページで、削除するノードグループを含むクラスターを選択します。
3. 選択したクラスターページで、[コンピューティング] タブを選択します。
4. [Node groups] (ノードグループ) セクションで、削除するノードグループを選択してください。その後、[削除] をクリックします。
5. [ノードグループの削除] 確認ダイアログボックスで、ノードグループの名前を入力します。その後、[削除] をクリックします。

AWS CLI

マネージド型ノードグループを AWS CLI を使用して削除する

1. 次のコマンドを入力します。各#####は独自の値に置き換えます。

```
aws eks delete-nodegroup \  
  --cluster-name my-cluster \  
  --nodegroup-name my-mng \  
  --region region-code
```

2. キーボードの矢印キーを使用して、応答出力をスクロールします。終了したら q キーを押します。

その他のオプションについては、「AWS CLI コマンドリファレンス」の [delete-nodegroup](#) コマンドを参照してください。

セルフマネージドノードでノードを自分自身で維持する

クラスターには、Pod がスケジュールされている Amazon EC2 ノードが 1 つ以上含まれています。Amazon EKS ノードは AWS アカウントで実行され、クラスター API サーバーエンドポイントを介してクラスターのコントロールプレーンに接続します。Amazon EC2 料金に基づいて請求されます。詳細については「[Amazon EC2 料金](#)」を参照してください。

クラスターには、複数のノードグループを含めることができます。各ノードグループには、[Amazon EC2 Auto Scaling グループ](#)にデプロイされる 1 つ以上のノードが含まれます。グループ内のノードのインスタンスタイプは、[Karpenter](#) による[属性ベースのインスタンスタイプの選択](#)を使用するときなどに、異なる場合があります。ノードグループ内のすべてのインスタンスは、[Amazon EKS ノードの IAM ロール](#)を使用する必要があります。

Amazon EKS は、Amazon EKS 最適化 AMI と呼ばれる特殊な Amazon マシンイメージ (AMI) を提供します。AMI は Amazon EKS と連携するように設定されています。そのコンポーネントには containerd、kubelet、および AWS IAM Authenticator が含まれます。また、AMI にはクラスターのコントロールプレーンを自動的に検出して接続を許可する、特別な[ブートストラップスクリプト](#)も含まれます。

CIDR ブロックを使用してクラスターのパブリックエンドポイントへのアクセスを制限する場合は、プライベートエンドポイントアクセスも有効にすることをお勧めします。これは、ノードがクラスターと通信できるようにするためです。プライベートエンドポイントが有効になっていない場合、パブリックアクセスに指定する CIDR ブロックに、VPC からの出力ソースを含める必要があります。詳細については、「[the section called “クラスターエンドポイントのアクセス”](#)」を参照してください。

Amazon EKS クラスターにセルフマネージド型ノードを追加するには、次のトピックを参照してください。セルフマネージドノードを手動で起動する場合は、各ノードに次のタグを追加すると共に、<cluster-name> がクラスターに一致することを確認します。詳細については、「[個々のリソースでのタグの追加と削除](#)」を参照してください。このガイドの手順に従うと、必要なタグがノードに自動的に追加されます。

キー	値
kubernetes.io/cluster/<cluster-name>	owned

⚠ Important

Amazon EC2 インスタンスメタデータサービス (IMDS) のタグは、EKS ノードと互換性がありません。インスタンスメタデータタグを有効にすると、タグ値でのスラッシュ (「/」) の使用が防止されます。この制限により、特に Karpenter や Cluster Autoscaler などのノード管理ツールを使用する場合、インスタンスの起動が失敗する可能性があります。これらのサービスは、スラッシュを含むタグに依存して適切に機能しているためです。

Kubernetes の一般的な視点から見たノードの詳細については、Kubernetes のドキュメントの「[Nodes](#)」を参照してください。

トピック

- [セルフマネージド Amazon Linux ノードを作成する](#)
- [セルフマネージド Bottlerocket ノードの作成](#)
- [セルフマネージド Microsoft Windows ノードの作成](#)
- [セルフマネージドの Ubuntu Linux ノードを作成する](#)
- [クラスターのためにセルフマネージドノードを更新する](#)

セルフマネージド Amazon Linux ノードを作成する

このトピックでは、Amazon EKS クラスターに登録されている Linux ノードの Auto Scaling グループを起動する方法を説明します。ノードがクラスターに参加したら、それらのノードに Kubernetes アプリケーションをデプロイ可能になります。eksctl または AWS Management Console を使用して、セルフマネージド型の Amazon Linux ノードを起動することもできます。AWS アウトポスト でノードを起動する必要がある場合は「[the section called “ノード”](#)」を参照してください。

- 既存の Amazon EKS クラスター。デプロイするには「[the section called “クラスターを作成する”](#)」を参照してください。AWS アウトポスト、AWS 波長、または AWS ローカルゾーンs が有効になっている AWS リージョンにサブネットがある場合はクラスターの作成時にそれらのサブネットが渡されるべきではありません。
- ノードが使用する既存の IAM 役割。作成する場合は「[the section called “ノードの IAM ロール”](#)」を参照してください。この役割に VPC CNI のポリシーがどちらも含まれていない場合、VPC CNI ポッドには以下の別の役割が必要です。

- (オプションですが推奨) 必要な IAM ポリシーがアタッチされた独自の IAM ロールで構成された Amazon VPC CNI plugin for Kubernetes アドオン。詳細については、「[the section called “IRSA 用の設定”](#)」を参照してください。
- 「[最適な Amazon EC2 ノードインスタンスタイプを選択する](#)」に記載されている考慮事項に精通していること。選択したインスタンスタイプによってはクラスターと VPC に関する追加の前提条件がある場合もあります。

セルフマネージド Linux ノードは次のいずれかを使用して起動できます。

- [the section called “eksctl”](#)
- [the section called “AWS Management Console”](#)

eksctl

eksctl を使用してセルフマネージド型の Linux ノードを起動する

1. デバイスまたは AWS クラウドシェル にインストールされている eksctl コマンドラインツールのバージョン 0.210.0 以降をインストールします。eksctl をインストールまたはアップグレードするにはeksctl ドキュメントの「[インストール](#)」を参照してください。
2. (オプション) [AmazonEKS_CNI_Policy] マネージド IAM ポリシーが [Amazon EKS ノードの IAM ロール](#) にアタッチされている場合は、代わりに Kubernetes aws-node サービスアカウントに関連付けた IAM ロールに割り当てることをお勧めします。詳細については、「[the section called “IRSA 用の設定”](#)」を参照してください。
3. 次のコマンドは既存のクラスターにノードグループを作成します。**al-nodes** を、ノードグループの名前に置き換えます。ノードグループ名は 63 文字以下である必要があります。先頭は文字または数字でなければなりませんが、残りの文字にはハイフンおよびアンダースコアを含めることもできます。**#####** の部分は自分のクラスター名に置き換えます。この名前には英数字 (大文字と小文字が区別されます) とハイフンのみを使用できます。先頭の文字は英数字である必要があります。また、100 文字より長くすることはできません。名前はクラスターを作成する AWS リージョンおよび AWS アカウント内で一意である必要があります。残りの**#####**は独自の値に置き換えます。デフォルトでは、ノードはコントロールプレーンと同じ Kubernetes バージョンで作成されます。

--node-type の値を選択する前に、「[最適な Amazon EC2 ノードインスタンスタイプを選択する](#)」を確認してください。

my-key を Amazon EC2 キーペアまたはパブリックキーの名前に置き換えます。このキーは起動後のノードに SSH 接続するために使用されます。Amazon EC2 キーペアをまだ持っていない場合は AWS Management Console で作成できます。詳細については「Amazon EC2 ユーザーガイド」の「[Amazon EC2 キーペア](#)」を参照してください。

次のコマンドを使用して、ノードグループを作成します。

 Important

ノードグループを AWS アウトポスト、波長、または ローカルゾーンs サブネットにデプロイする場合、追加の考慮事項があります。

- クラスターを作成したときに、サブネットが渡されていない必要があります。
- ノードグループはサブネットと [volumeType](#): gp2 を指定する設定ファイルを使用して作成する必要があります。詳細については「eksctl ドキュメント」の「[設定ファイルからノードグループを作成する](#)」と「[設定ファイルのスキーマ](#)」を参照してください。

```
eksctl create nodegroup \  
  --cluster my-cluster \  
  --name al-nodes \  
  --node-type t3.medium \  
  --nodes 3 \  
  --nodes-min 1 \  
  --nodes-max 4 \  
  --ssh-access \  
  --managed=false \  
  --ssh-public-key my-key
```

次のノードグループをデプロイするには

- デフォルト設定よりもはるかに多くの IP アドレスを Pod に割り当てることができるノードグループについては、「[the section called “IP アドレスを増やす”](#)」を参照してください。
- インスタンスのブロックとは異なる CIDR ブロックから IPv4 アドレスを Pod に割り当てることができるノードグループについては、「[the section called “カスタムネットワーキング”](#)」を参照してください。
- Pod とサービスに IPv6 アドレスを割り当てることができるノードグループについては、「[the section called “IPv6”](#)」を参照してください。

- アウトバウンドインターネットアクセスがない場合は「[the section called “プライベートクラスター”](#)」を参照してください。

すべての使用可能なオプションとデフォルト値の一覧を表示するには次のコマンドを入力してください。

```
eksctl create nodegroup --help
```

ノードがクラスターに参加できない場合はトラブルシューティングの章にある「[the section called “ノードをクラスターに結合できません”](#)」を参照してください。

出力例は次のとおりです。ノードの作成中に、複数の行が出力されます。出力の最後の行は次のサンプル行が表示されます。

```
[#] created 1 nodegroup(s) in cluster "my-cluster"
```

4. (オプション) [サンプルアプリケーション](#) をデプロイして、クラスターと Linux ノードをテストします。
5. 次の条件が true の場合、IMDS への Pod アクセスをブロックすることをお勧めします。
 - IAM ロールをすべての Kubernetes サービスアカウントに割り当てることによって、必要最小限のアクセス許可のみを Pod に付与しようとしている。
 - クラスター内の Pod が、現在の AWS リージョンの取得など、その他の理由で Amazon EC2 インスタンスメタデータサービス (IMDS) へのアクセスを必要としていない。

詳細については「[ワーカーノードに割り当てられたインスタンスプロファイルへのアクセスを制限する](#)」を参照してください。

AWS Management Console

ステップ 1: AWS Management Console を使用してセルフマネージド型の リナックス ノードを起動する`

1. AWS クラウドフォーメーション テンプレートの最新バージョンをダウンロードします。

```
curl -O https://s3.us-west-2.amazonaws.com/amazon-eks/cloudformation/2022-12-23/  
amazon-eks-nodegroup.yaml
```


2. クラスターステータスが ACTIVE と表示されるまで待ちます。クラスターがアクティブになる前にノードを起動した場合、ノードはクラスターへの登録に失敗し、再起動する必要があります。
3. [AWS クラウドフォーメーション コンソール](#)を開きます。
4. [スタックの作成] を選択し、[新しいリソースを使用 (標準)] を選択してください。
5. [テンプレートの指定] で、[テンプレートファイルのアップロード] を選択し、[ファイルを選択] を選択してください。
6. ダウンロードした `amazon-eks-nodegroup.yaml` ファイルを選択してください。
7. [次へ] を選択してください。
8. [スタックの詳細の指定] ページで、必要に応じて次のパラメータを入力し、[次へ] を選択してください。
 - スタック名: AWS クラウドフォーメーション スタックのスタック名を選択してください。例えば、`#####-nodes` という名前にすることができます。この名前には英数字 (大文字と小文字が区別されます) とハイフンのみを使用できます。先頭の文字は英数字である必要があります。また、100 文字より長くすることはできません。名前はクラスターを作成する AWS リージョンおよび AWS アカウント内で一意である必要があります。
 - [クラスター名]: Amazon EKS クラスターの作成時に使用した名前を入力してください。この名前はクラスター名と同じにする必要があります。同じでない場合、ノードはクラスターに参加できません。
 - [クラスター制御プレーンセキュリティグループ]: [VPC](#) の作成時に生成した AWS クラウドフォーメーション 出力の [セキュリティグループs] 値を選択してください。

次のステップでは該当するグループを取得する 1 つのオペレーションを説明します。

- a. [Amazon EKS コンソール](#)を開きます。
- b. クラスターの名前を選択してください。
- c. [ネットワーキング] タブを選択してください。
- d. [クラスター制御プレーンセキュリティグループ] ドロップダウンリストから選択する場合は[追加のセキュリティグループ] の値をリファレンスとして使用します。
- [ノードグループ名]: ノードグループの名前を入力してください。この名前はノードに対して作成される自動スケーリングノードグループを識別するために後で使用できます。ノードグループ名は 63 文字以下である必要があります。先頭は文字または数字でなければなりませんが、残りの文字にはハイフンおよびアンダースコアを含めることもできます。
- [ノードオートスケーリンググループ最小サイズ]: ノードの 自動スケーリング グループがスケールインできる最小ノード数を入力してください。

- ノード自動スケーリンググループ希望容量: スタック作成時にスケーリングする必要のあるノード数を入力してください。
- [NodeAutoScalingGroupMaxSize]: ノードの自動スケーリンググループがスケールアウトできる最大ノード数を入力してください。
- [ノードインスタンス型]: ノードのインスタンスタイプを選択してください。詳細については、「[the section called “Amazon EC2 インスタンスタイプ”](#)」を参照してください。
- [NodeImageIdSSMParam]: 最新の Amazon EKS 最適化 AMI の Amazon EC2 Systems Manager のパラメータが、可変 Kubernetes バージョン用に事前設定されています。Amazon EKS でサポートされている別の Kubernetes マイナーバージョンを使用するには、**1.XX** を別の[サポートされているバージョン](#)に置き換えます。クラスターと同じ Kubernetes バージョンを指定することをお勧めします。

amazon-linux-2 を別の AMI タイプに置き換えることもできます。詳細については「[the section called “最新の ID を取得する”](#)」を参照してください。

 Note

Amazon EKS ノード AMI は Amazon リナックス をベースとしています。[Amazon Linux セキュリティセンター](#)で Amazon Linux 2 のセキュリティもしくはプライバシーに関するイベントを追跡したり、関連する [RSS フィード](#)をサブスクライブできます。セキュリティおよびプライバシーイベントには問題の概要、影響を受けるパッケージ、および問題を修正するためにインスタンスを更新する方法などがあります。

- ノードイメージId: (オプション) (Amazon EKS 最適化 AMI の代わりに) 独自のカスタム AMI を使用している場合はAWS リージョンのノード AMI ID を入力してください。ここで値を指定すると、[ノードイメージIdSSMParam] フィールドの値はすべて上書きされます。
- [ノードボリュームサイズ]: ノードのルートボリュームのサイズを GiB 単位で指定します。
- [ノードボリューム型]: ノードのルートボリュームタイプを指定します。
- [キー名]: 起動後に、SSH を使用してノードに接続するときに使用できる Amazon EC2 SSH キーペアの名前を入力してください。Amazon EC2 キーペアをまだ持っていない場合はAWS Management Console で作成できます。詳細については「アマゾン EC2 ユーザーガイド」の「[アマゾン EC2 キーペア](#)」を参照してください。

 Note

ここでキーペアを指定しないと、AWS クラウドフォーメーション スタックの作成は失敗します。

- [ブートストラップ引数]: kubelet の追加引数など、ノードブートストラップスクリプトに渡すオプションの引数を指定します。詳細については、GitHub の [bootstrap script usage information](#) を参照してください。

次のノードグループをデプロイするには

- デフォルト設定よりもはるかに多くの IP アドレスを Pod に割り当てることができるノードグループについては、「[the section called “IP アドレスを増やす”](#)」を参照してください。
- インスタンスのブロックとは異なる CIDR ブロックから IPv4 アドレスを Pod に割り当てることができるノードグループについては、「[the section called “カスタムネットワーキング”](#)」を参照してください。
- Pod とサービスに IPv6 アドレスを割り当てることができるノードグループについては、「[the section called “IPv6”](#)」を参照してください。
- アウトバウンドインターネットアクセスがない場合は「[the section called “プライベートクラスタ”](#)」を参照してください。
- [無効IMDSv1]: 各ノードはデフォルトでインスタンスメタデータサービスバージョン 1 (IMDSv1) および IMDSv2 をサポートします。IMDSv1 は無効にできます。今後ノードグループのノードおよび Pod が IMDSv1 を使用しないようにするには、DisableIMDSv1 を true に設定します。IDMS の詳細については「[インスタンスメタデータサービスの設定](#)」を参照してください。ノードでのそれへのアクセス制限について詳しくは[ワーカーノードに割り当てられたインスタンスプロファイルへのアクセスを制限する](#)を参照してください。
- [VpcId]: 作成した [VPC](#) の ID を入力してください。
- [Subnets]: VPC 用に作成したサブネットを選択してください。「[Amazon EKS クラスターの Amazon VPC を作成する](#)」で説明されている手順で VPC を作成した場合は起動するノードの VPC 内のプライベートサブネットのみを指定します。クラスターの [ネットワーキング] タブから、各サブネットリンクを開き、プライベートのサブネットを確認できます。

 Important

- いずれかのサブネットがパブリックサブネットである場合はパブリック IP アドレスの自動割り当て設定を有効にする必要があります。この設定がパブリックサブネット

に対して有効になっていない場合、そのパブリックサブネットにデプロイするノードにはパブリック IP アドレスが割り当てられず、クラスターやその他の AWS のサービスと通信できなくなります。2020 年 3 月 26 日以前に、[Amazon EKS AWS クラウドフォーメーション VPC テンプレート](#)を使用して、または eksctl を使用してサブネットがデプロイされた場合、パブリックサブネットではパブリック IP アドレスの自動割り当てが無効になります。サブネットのパブリック IP アドレス割り当てを有効にする方法については「[サブネットのパブリック IPv4 アドレス属性の変更](#)」を参照してください。ノードがプライベートサブネットにデプロイされている場合、NAT ゲートウェイを介してクラスターや他の AWS のサービスと通信できます。

- サブネットにインターネットアクセスがない場合は「[インターネットアクセスが制限されたプライベートクラスターをデプロイする](#)」の考慮事項と追加の手順を確認してください。
- AWS アウトポスト、波長、または ローカルゾーンs サブネット を選択する場合はクラスターの作成時にサブネットを渡さないようにします。

9. [スタックオプションの設定] ページで、希望する設定を選択し、[次へ] を選択してください。
- 10[AWS クラウドフォーメーション が IAM リソースを作成する可能性を認識しています] の左にあるチェックボックスを選択して、[スタックの作成] を選択してください。
- 11スタックの作成が完了したら、コンソールで選択し、[出力] を選択してください。
- 12作成されたノードグループの [ノードインスタンス役割] を記録します。これはAmazon EKS ノードを設定する際、必要になります。

ステップ 2: ノードを有効にしてクラスターに参加する

Note

アウトバウンドのインターネットアクセスのないプライベート VPC でノードを起動する場合はノードが VPC 内からクラスターに参加できるようにしてください。

1. aws-auth ConfigMap がすでにあるかどうかを確認します。

```
kubectl describe configmap -n kube-system aws-auth
```

2. aws-auth ConfigMap が表示されている場合は必要に応じて更新してください。
 - a. 編集する ConfigMap を開きます。

```
kubectl edit -n kube-system configmap/aws-auth
```

- b. 必要に応じて新しい mapRoles エントリを追加します。rolearn 値を、前の手順で記録した [ノードインスタンス役割] 値に設定します。

```
[...]
data:
  mapRoles: |
    - rolearn: <ARN of instance role (not instance profile)>
      username: system:node:{{EC2PrivateDNSName}}
      groups:
        - system:bootstrappers
        - system:nodes
[...]
```

- c. ファイルを保存し、テキストエディタを終了します。
3. 「Error from server (NotFound): configmaps "aws-auth" not found」というエラーが表示されたら、ストック ConfigMap を適用してください。
- a. 設定マップをダウンロードします。

```
curl -O https://s3.us-west-2.amazonaws.com/amazon-eks/cloudformation/2020-10-29/
aws-auth-cm.yaml
```

- b. aws-auth-cm.yaml ファイルで、rolearn 値を前の手順で記録した [ノードインスタンス役割] 値に設定します。これを行うにはテキストエディタを使用するか、##### を置き換えて次のコマンドを実行してください。

```
sed -i.bak -e 's|<ARN of instance role (not instance profile)>|my-node-instance-
role|' aws-auth-cm.yaml
```

- c. 設定を適用します。このコマンドが完了するまで数分かかることがあります。

```
kubectl apply -f aws-auth-cm.yaml
```

4. ノードのステータスを監視し、Ready ステータスになるまで待機します。

```
kubectl get nodes --watch
```

Ctrl+C を入力して、シェルプロンプトに戻ります。

Note

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

ノードがクラスターに参加できない場合はトラブルシューティングの章にある「[the section called “ノードをクラスターに結合できません”](#)」を参照してください。

5. (GPU ノードのみ) GPU インスタンスタイプと Amazon EKS 最適化アクセラレーション AMI を選択した場合はクラスター上の DaemonSet として [Kubernetes 用の NVIDIA デバイスプラグイン](#) を適用する必要があります。次のコマンドを実行する前に、`vX.X.X` を必要となる [NVIDIA/k8s-device-plugin](#) バージョンに置き換えます。

```
kubectl apply -f https://raw.githubusercontent.com/NVIDIA/k8s-device-plugin/vX.X.X/
deployments/static/nvidia-device-plugin.yml
```

ステップ 3: その他のアクション

1. (オプション) [サンプルアプリケーション](#) をデプロイして、クラスターと Linux ノードをテストします。
2. (オプション) AmazonEKS_CNI_Policy マネージド IAM ポリシー (IPv4 クラスターがある場合) または [AmazonEKS_CNI_IPv6_Policy](#) (IPv6 クラスターがある場合には [ユーザー自身が作成したもの](#)) が [Amazon EKS ノードの IAM ロール](#) にアタッチされている場合、代わりに Kubernetes aws-node サービスアカウントに関連付けた IAM ロールに割り当てることをお勧めします。詳細については、「[the section called “IRSA 用の設定”](#)」を参照してください。
3. 次の条件が true の場合、IMDS への Pod アクセスをブロックすることをお勧めします。
 - IAM ロールをすべての Kubernetes サービスアカウントに割り当てることによって、必要最小限のアクセス許可のみを Pod に付与しようとしている。
 - クラスター内の Pod が、現在の AWS リージョンの取得など、その他の理由で Amazon EC2 インスタンスメタデータサービス (IMDS) へのアクセスを必要としていない。

詳細については「[ワーカーノードに割り当てられたインスタンスプロファイルへのアクセスを制限する](#)」を参照してください。

セルフマネージド Bottlerocket ノードの作成

Note

マネージド型ノードグループでは、ユースケースにいくつかの利点がある場合があります。詳細については、「[the section called “マネージド型ノードグループ”](#)」を参照してください。

このトピックでは、Amazon EKS クラスターに登録する [Bottlerocket](#) ノードの Auto Scaling グループを起動する方法について説明します。Bottlerocket は AWS が提供する Linux ベースのオープンソースのオペレーティングシステムで、仮想マシンやベアメタルホスト上でコンテナを実行するために使用できます。ノードがクラスターに参加したら、それらのノードに Kubernetes アプリケーションをデプロイ可能になります。Bottlerocket の詳細については、GitHub の [Using a Bottlerocket AMI with Amazon EKS](#) と eksctl ドキュメント内の [Custom AMI support](#) を参照してください。

インプレースアップグレードについて詳しくは、GitHub の [Bottlerocket Update Operator](#) を参照してください。

Important

- Amazon EKS ノードは標準の Amazon EC2 インスタンスであり、通常の Amazon EC2 インスタンス価格に基づいて請求されます。詳細については「[Amazon EC2 料金](#)」を参照してください。
- AWS Outposts 上の Amazon EKS 拡張クラスターで Bottlerocket ノードを起動できますが、AWS Outposts 上のローカルクラスターでは起動できません。詳細については、「[Amazon EKS on AWS Outposts](#)」を参照してください。
- x86 または Arm プロセッサを使用して Amazon EC2 インスタンスにデプロイできます。ただし、Inferentia チップがあるインスタンスにデプロイすることはできません。
- Bottlerocket は、AWS CloudFormation と互換性があります。ただし、Amazon EKS の Bottlerocket ノードをデプロイするためにコピーできる公式の CloudFormation テンプレートはありません。
- Bottlerocket のイメージには、SSH サーバーやシェルは付属していません。SSH で管理者コンテナを有効にし、ユーザーデータとともにブートストラップの設定ステップを渡すために、帯域外のアクセス方式を使用できます。詳細については、GitHub の「[bottleRocket readme.md](#)」のセクションを参照してください。

- [探査](#)
- [管理者コンテナ](#)
- [Kubernetes 設定](#)

この手順には、eksctl バージョン 0.210.0 以降が必要です。お使いのバージョンは、以下のコマンドを使用して確認できます。

```
eksctl version
```

eksctl のインストールまたはアップグレードの手順については、eksctl ドキュメントの「[インストール](#)」を参照してください。注意: この手順は、eksctl で作成されたクラスターでのみ機能します。

1. 次のコンテンツをデバイスにコピーします。##### の部分は自分のクラスター名に置き換えます。この名前には英数字 (大文字と小文字が区別されます) とハイフンのみを使用できます。先頭の文字は英数字である必要があります。また、100 文字より長くすることはできません。名前は、クラスターを作成する AWS リージョンおよび AWS アカウント内で一意である必要があります。*ng-bottlerocket* をノードグループの名前に置き換えます。ノードグループ名は 63 文字以下である必要があります。先頭は文字または数字でなければなりません。残りの文字にはハイフンおよびアンダースコアを含めることもできます。Arm インスタンスにデプロイするには、*m5.large* を Arm インスタンスタイプに置き換えます。*my-ec2-keypair-name* を、起動後に、SSH を使用してノードに接続するときに使用できる Amazon EC2 SSH キーペアの名前に置き換えます。Amazon EC2 キーペアをまだ持っていない場合は、AWS Management Console で作成できます。詳細については「Amazon EC2 ユーザーガイド」の「[Amazon EC2 キーペア](#)」を参照してください。残りの##### はすべて独自の値に置き換えます。置き換えが完了したら、変更したコマンドを実行して bottlerocket.yaml ファイルを作成します。

Arm Amazon EC2 インスタンスタイプを指定する場合は、デプロイする前に「[Amazon EKS 最適化 Arm Amazon Linux AMI](#)」の考慮事項を確認してください。カスタム AMI を使用したデプロイの手順については、GitHub の [Building Bottlerocket](#) と、eksctl ドキュメントの [Custom AMI support](#) を参照してください。マネージド型ノードグループをデプロイするには、起動テンプレートを使用してカスタム AMI をデプロイします。詳細については、「[the section called “起動テンプレート”](#)」を参照してください。

 Important

ノードグループを AWS Outposts、AWS Wavelength、または AWS Local Zone サブネットにデプロイする場合、クラスターの作成時に AWS Outposts、AWS Wavelength、または AWS Local Zone サブネットは渡さないでください。次の例では、サブネットを指定する必要があります。詳細については、eksctl ドキュメントの「[設定ファイルからノードグループを作成する](#)」と「[Config ファイルのスキーマ](#)」を参照してください。*region-code* を、クラスターのある AWS リージョンに置き換えます。

```
cat >bottlerocket.yaml <<EOF
---
apiVersion: eksctl.io/v1alpha5
kind: ClusterConfig

metadata:
  name: my-cluster
  region: region-code
  version: '1.32'

iam:
  withOIDC: true

nodeGroups:
- name: ng-bottlerocket
  instanceType: m5.large
  desiredCapacity: 3
  amiFamily: Bottlerocket
  ami: auto-ssm
  iam:
    attachPolicyARNs:
    - arn:aws:iam::aws:policy/AmazonEKSWorkerNodePolicy
    - arn:aws:iam::aws:policy/AmazonEC2ContainerRegistryReadOnly
    - arn:aws:iam::aws:policy/AmazonSSMManagedInstanceCore
    - arn:aws:iam::aws:policy/AmazonEKS_CNI_Policy
  ssh:
    allow: true
    publicKeyName: my-ec2-keypair-name
EOF
```


2. 次のコマンドでノードをデプロイします。

```
eksctl create nodegroup --config-file=bottlerocket.yaml
```

出力例は次のとおりです。

ノードの作成中に、複数の行が出力されます。出力の最後の行は次のサンプル行が表示されます。

```
[#] created 1 nodegroup(s) in cluster "my-cluster"
```

3. (オプション) [Amazon EBS CSI プラグイン](#)を使用して、Bottlerocket ノード上に Kubernetes [永続ボリューム](#)を作成します。デフォルトの Amazon EBS ドライバーは、Bottlerocket に含まれていないファイルシステムツールを利用します。ドライバーを使用してストレージクラスを作成する方法の詳細については、「[the section called "Amazon EBS"](#)」を参照してください。
4. (オプション) デフォルトでは、kube-proxy は `nf_conntrack_max` カーネルパラメータを、Bottlerocket がブート時に設定するものとは異なるデフォルト値に設定します。Bottlerocket の[デフォルト設定](#)を保持するには、次のコマンドを使用して kube-proxy 設定を編集します。

```
kubectrl edit -n kube-system daemonset kube-proxy
```

次の例にある kube-proxy 引数に、`--conntrack-max-per-core` と `--conntrack-min` を追加します。0 の設定は、変更がないことを意味します。

```
containers:
- command:
  - kube-proxy
  - --v=2
  - --config=/var/lib/kube-proxy-config/config
  - --conntrack-max-per-core=0
  - --conntrack-min=0
```

5. (オプション) [サンプルアプリケーション](#)をデプロイして、Bottlerocket ノードをテストします。

6. 次の条件が true の場合、IMDS への Pod アクセスをブロックすることをお勧めします。

- IAM ロールをすべての Kubernetes サービスアカウントに割り当てることによって、必要最小限のアクセス許可のみを Pod に付与しようとしている。
- クラスター内の Pod が、現在の AWS リージョンの取得など、その他の理由で Amazon EC2 インスタンスメタデータサービス (IMDS) へのアクセスを必要としていない。

詳細については「[ワーカーノードに割り当てられたインスタンスプロファイルへのアクセスを制限する](#)」を参照してください。

セルフマネージド Microsoft Windows ノードの作成

このトピックでは、Amazon EKS クラスターに登録されている Windows ノードの Auto Scaling グループの起動方法について説明します。ノードがクラスターに参加したら、それらのノードに Kubernetes アプリケーションをデプロイ可能になります。

Important

- Amazon EKS ノードは標準の Amazon EC2 インスタンスであり、通常の Amazon EC2 インスタンス価格に基づいて請求されます。詳細については「[Amazon EC2 料金](#)」を参照してください。
- AWS アウトポスト 上の Amazon EKS 拡張クラスターで Windows ノードを起動できますが、AWS アウトポスト 上のローカルクラスターでは起動できません。詳細については、「[Amazon EKS on AWS Outposts](#)」を参照してください。

クラスターに対して Windows サポートを有効にします。Windows ノードグループを起動する前に、重要な考慮事項を確認することをお勧めします。詳細については、「[the section called “Windows サポートを有効にする”](#)」を参照してください。

次のいずれかを使用して、セルフマネージド型の Windows ノードを起動できます。

- [the section called “eksctl”](#)
- [the section called “AWS Management Console”](#)

eksctl

eksctl <2> を使用して自己管理型 Windows ノードを起動します。

この手順ではeksctl をインストール済みで、お使いの eksctl バージョンが 0.210.0 以上であることが必要です。お使いのバージョンは以下のコマンドを使用して確認できます。

```
eksctl version
```

eksctl のインストールまたはアップグレードの手順についてはeksctl ドキュメントの「[インストール](#)」を参照してください。

 Note

この手順はeksctl で作成されたクラスターに対してのみ機能します。

1. (オプション) AmazonEKS_CNI_Policy マネージド IAM ポリシー (IPv4 クラスターがある場合) または [AmazonEKS_CNI_IPv6_Policy](#) (IPv6 クラスターがある場合には[ユーザー自身が作成したもの](#)) が [Amazon EKS ノードの IAM ロール](#) にアタッチされている場合、代わりに Kubernetes aws-node サービスアカウントに関連付けた IAM ロールに割り当てることをお勧めします。詳細については、「[the section called “IRSA 用の設定”](#)」を参照してください。
2. この手順では既存のクラスターがあることを前提としています。Windows ノードグループの追加先となる Amazon EKS クラスターと Amazon Linux ノードグループがまだ存在しない場合は、[the section called “クラスターを作成する \(eksctl\)”](#) に従うことをお勧めします。このガイドはAmazon Linux ノードで Amazon EKS クラスターを作成する方法についての完全なチュートリアルを提供します。

次のコマンドを使用して、ノードグループを作成します。[#####](#) を、クラスターのある AWS リージョンに置き換えます。[#####](#) の部分は自分のクラスター名に置き換えます。この名前には英数字 (大文字と小文字が区別されます) とハイフンのみを使用できます。先頭の文字は英数字である必要があります。また、100 文字より長くすることはできません。名前はクラスターを作成する AWS リージョンおよび AWS アカウント内で一意である必要があります。[ng-windows](#) をノードグループの名前に置き換えます。ノードグループ名は 63 文字以下である必要があります。先頭は文字または数字でなければなりませんが、残りの文字にはハイフンおよびアンダースコアを含めることもできます。[2019](#) を 2022 に置き換えて Windows Server 2022 を使用できます。残りの [#####](#) は独自の値に置き換えます。

 Important

ノードグループを AWS アウトポスト、AWS 波長、または AWS ローカルゾーン サブネットにデプロイする場合、クラスターの作成時に AWS アウトポスト、波長、またはローカルゾーン サブネットは渡さないでください。AWS アウトポスト、波長、またはローカルゾーンs サブネットを指定した設定ファイルを使用して、ノードグループを作成します。詳細についてはeksctl ドキュメントの「[設定ファイルからノードグループを作成する](#)」と「[設定ファイルのスキーマ](#)」を参照してください。

```
eksctl create nodegroup \  
  --region region-code \  
  --cluster my-cluster \  
  --name ng-windows \  
  --node-type t2.large \  
  --nodes 3 \  
  --nodes-min 1 \  
  --nodes-max 4 \  
  --managed=false \  
  --node-ami-family WindowsServer2019FullContainer
```

Note

- ノードがクラスターに参加できない場合はトラブルシューティングガイドの「[the section called “ノードをクラスターに結合できません”](#)」を参照してください。
- eksctl コマンドで使用可能なオプションを表示するには次のコマンドを入力してください。

```
eksctl command -help
```

出力例は次のとおりです。ノードの作成中に、複数の行が出力されます。出力の最後の行は次のサンプル行が表示されます。

```
[#] created 1 nodegroup(s) in cluster "my-cluster"
```

3. (オプション) [サンプルアプリケーション](#)をデプロイして、クラスターと Windows ノードをテストします。
4. 次の条件が true の場合、IMDS への Pod アクセスをブロックすることをお勧めします。
 - IAM ロールをすべての Kubernetes サービスアカウントに割り当てることによって、必要最小限のアクセス許可のみを Pod に付与しようとしている。
 - クラスター内の Pod が、現在の AWS リージョンの取得など、その他の理由で Amazon EC2 インスタンスメタデータサービス (IMDS) へのアクセスを必要としていない。

詳細については、「[ワーカーノードに割り当てられたインスタンスプロファイルへのアクセスを制限する](#)」を参照してください。

AWS Management Console

前提条件

- 既存の Amazon EKS クラスターと Linux ノードグループ。これらのリソースがない場合は[はじめに](#)のガイドのいずれかに従って作成することをお勧めします。これらのガイドでは、Linux ノードで Amazon EKS クラスターを作成する方法について説明しています。
- Amazon EKS クラスターの要件を満たす、既存の VPC およびセキュリティグループ。詳細については[the section called “VPC およびサブネットの要件”](#)および[the section called “セキュリティグループの要件”](#)を参照してください。[はじめに](#)のガイドでは要件を満たす VPC を作成します。または「[Amazon EKS クラスターの Amazon VPC を作成する](#)」に従って手動で作成することもできます。
- Amazon EKS クラスターの要件を満たす VPC およびセキュリティグループを使用している、既存の Amazon EKS クラスター。詳細については「[the section called “クラスターを作成する”](#)」を参照してください。AWS アウトポスト、AWS 波長、または AWS ローカルゾーンが有効になっている AWS リージョンにサブネットがある場合はクラスターの作成時にそれらのサブネットが渡されるべきではありません。

ステップ 1: AWS Management Console を使用してセルフマネージド型の Windows ノードを起動する

1. クラスターステータスが ACTIVE と表示されるまで待ちます。クラスターがアクティブになる前にノードを起動した場合、ノードはクラスターへの登録に失敗し、再起動が必要になります。
2. [AWS クラウドフォーメーション コンソール](#)を開きます
3. [スタックの作成] を選択してください。
4. [テンプレートを指定] で、[Amazon S3 URL] を選択してください。
5. 次の URL をコピーして、[Amazon S3 URL] に貼り付けます。

```
https://s3.us-west-2.amazonaws.com/amazon-eks/cloudformation/2023-02-09/amazon-eks-windows-nodegroup.yaml
```

6. [次へ] を 2 回選択してください。
7. [スタックのクイック作成] ページで、必要に応じて以下のパラメータを入力してください。
 - スタック名: AWS クラウドフォーメーション スタックのスタック名を選択してください。例えば、my-cluster-nodes という名前にすることができます。
 - [クラスター名]: Amazon EKS クラスターの作成時に使用した名前を入力してください。

 Important

この名前は「[ステップ 1: Amazon EKS クラスターを作成する](#)」で使用した名前と完全に一致する必要があります。そうしないと、ノードはクラスターに参加できません。

- クラスター制御プレーンセキュリティグループ: [VPC](#) を作成する際に生成した AWS クラウド フォーマション 出力からセキュリティグループを選択してください。次のステップでは該当するグループを取得する 1 つの方法を説明します。
 - a. [Amazon EKS コンソール](#)を開きます。
 - b. クラスターの名前を選択してください。
 - c. [ネットワーキング] タブを選択してください。
 - d. [クラスター制御プレーンセキュリティグループ] ドロップダウンリストから選択する場合は[追加のセキュリティグループ] の値をリファレンスとして使用します。
- [ノードグループ名]: ノードグループの名前を入力してください。この名前はノードに対して作成される自動スケーリングノードグループを識別するために後で使用できます。ノードグループ名は 63 文字以下である必要があります。先頭は文字または数字でなければなりませんが、残りの文字にはハイフンおよびアンダースコアを含めることもできます。
- [ノードオートスケーリンググループ最小サイズ]: ノードの 自動スケーリング グループがスケールインできる最小ノード数を入力してください。
- ノード自動スケーリンググループ希望容量: スタック作成時にスケーリングする必要のあるノード数を入力してください。
- [NodeAutoScalingGroupMaxSize]: ノードの 自動スケーリング グループがスケールアウトできる最大ノード数を入力してください。
- [ノードインスタンス型]: ノードのインスタンスタイプを選択してください。詳細については、「[the section called “Amazon EC2 インスタンスタイプ”](#)」を参照してください。

 Note

最新バージョンの [Amazon VPC CNI plugin for Kubernetes](#) でサポートされているインスタンスタイプは、GitHub 上の [vpc_ip_resource_limit.go](#) にリストされています。サポートされている最新のインスタンスタイプを利用するには、CNI のバージョンを更新することが必要になる場合があります。詳細については、「[the section called “Amazon VPC CNI”](#)」を参照してください。

- [NodeImageIdSSMParam]: 現在推奨されている Amazon EKS 最適化 Windows Core AMI ID の Amazon EC2 Systems Manager パラメータが事前設定されています。Windows の通常版を使用する場合は、**Core** を Full に置き換えます。
- ノードイメージId: (オプション) (Amazon EKS 最適化 AMI の代わりに) 独自のカスタム AMI を使用している場合はAWS リージョンのノード AMI ID を入力してください。このフィールドに値を指定すると、[ノードイメージIdSSMParam] フィールドの値はすべて上書きされます。
- [ノードボリュームサイズ]: ノードのルートボリュームのサイズを GiB 単位で指定します。
- [キー名]: 起動後に、SSH を使用してノードに接続するときに使用できる Amazon EC2 SSH キーペアの名前を入力してください。Amazon EC2 キーペアをまだ持っていない場合はAWS Management Console で作成できます。詳細については「Amazon EC2 ユーザーガイド」の「[Amazon EC2 キーペア](#)」を参照してください。

 Note

ここでキーペアを指定しないと、AWS クラウドフォーメーション スタックの作成は失敗します。

- [ブートストラップ引数]: kubelet を使用して、-KubeletExtraArgs の追加引数など、ノードブートストラップスクリプトに渡すオプションの引数を指定します。
- [無効IMDSv1]: 各ノードはデフォルトでインスタンスメタデータサービスバージョン 1 (IMDSv1) および IMDSv2 をサポートします。IMDSv1 は無効にできます。今後ノードグループのノードおよび Pod が IMDSv1 を使用しないようにするには、DisableIMDSv1 を true に設定します。IDMS の詳細については「[インスタンスメタデータサービスの設定](#)」を参照してください。
- [VpcId]: 作成した [VPC](#) の ID を選択します。
- [NodeSecurityGroups]: [VPC](#) の作成時に Linux ノードグループ用に作成したセキュリティグループを選択します。Linux ノードに複数のセキュリティグループがアタッチされている場合、それらのすべてを指定します。これは、たとえば、Linux ノードグループが eksctl を使用して作成された場合に行います。
- [Subnets]: 作成したサブネットを選択してください。「[Amazon EKS クラスターの Amazon VPC を作成する](#)」で説明されている手順で VPC を作成した場合は起動するノードの VPC 内のプライベートサブネットのみを指定します。

⚠ Important

- いずれかのサブネットがパブリックサブネットである場合はパブリック IP アドレスの自動割り当て設定を有効にする必要があります。この設定がパブリックサブネットに対して有効になっていない場合、そのパブリックサブネットにデプロイするノードにはパブリック IP アドレスが割り当てられず、クラスターやその他の AWS のサービスと通信できなくなります。2020 年 3 月 26 日以前に、[Amazon EKS AWS クラウドフォーメーション VPC テンプレート](#)を使用して、または eksctl を使用してサブネットがデプロイされた場合、パブリックサブネットではパブリック IP アドレスの自動割り当てが無効になります。サブネットのパブリック IP アドレス割り当てを有効にする方法については「[サブネットのパブリック IPv4 アドレス属性の変更](#)」を参照してください。ノードがプライベートサブネットにデプロイされている場合、NAT ゲートウェイを介してクラスターや他の AWS のサービスと通信できます。
- サブネットにインターネットアクセスがない場合は「[インターネットアクセスが制限されたプライベートクラスターをデプロイする](#)」の考慮事項と追加の手順を確認してください
- AWS アウトポスト、波長、または ローカルゾーンs サブネット を選択する場合はクラスターの作成時にサブネットを渡さないようにします。

8. スタックが IAM リソースを作成する可能性があることを確認し、[スタックの作成] を選択してください。
9. スタックの作成が完了したら、コンソールで選択し、[出力] を選択してください。
10. 作成されたノードグループの [NodeInstanceRole] を記録します。これは、Amazon EKS Windows ノードを設定するときに必要になります。

ステップ 2: ノードを有効にしてクラスターに参加する

1. aws-auth ConfigMap がすでにあるかどうかを確認します。

```
kubectl describe configmap -n kube-system aws-auth
```

2. aws-auth ConfigMap が表示されている場合は必要に応じて更新してください。

- a. 編集する ConfigMap を開きます。

```
kubectl edit -n kube-system configmap/aws-auth
```


- b. 必要に応じて新しい mapRoles エントリを追加します。rolearn 値を、前の手順で記録した [ノードインスタンス役割] の値に設定します。

```
[...]
data:
  mapRoles: |
    - rolearn: <ARN of linux instance role (not instance profile)>
      username: system:node:{{EC2PrivateDNSName}}
      groups:
        - system:bootstrappers
        - system:nodes
    - rolearn: <ARN of windows instance role (not instance profile)>
      username: system:node:{{EC2PrivateDNSName}}
      groups:
        - system:bootstrappers
        - system:nodes
        - eks:kube-proxy-windows
[...]
```

- c. ファイルを保存し、テキストエディタを終了します。

3. 「Error from server (NotFound): configmaps "aws-auth" not found」というエラーが表示されたら、ストック ConfigMap を適用してください。

- a. 設定マップをダウンロードします。

```
curl -O https://s3.us-west-2.amazonaws.com/amazon-eks/cloudformation/2020-10-29/
aws-auth-cm-windows.yaml
```

- b. aws-auth-cm-windows.yaml ファイルで、rolearn 値を、前の手順で記録した該当する [ノードインスタンス役割] 値に設定します。これを行うにはテキストエディタを使用するか、この#####を置き換えて次のコマンドを実行してください。

```
sed -i.bak -e 's|<ARN of linux instance role (not instance profile)>|my-node-
linux-instance-role|' \
    -e 's|<ARN of windows instance role (not instance profile)>|my-node-windows-
instance-role|' aws-auth-cm-windows.yaml
```

Important

- このファイルの他の行は変更しないでください。

- Windows ノードと Linux ノードの両方に同じ IAM ロールを使用しないでください。

c. 設定を適用します。このコマンドが完了するまで数分かかることがあります。

```
kubectl apply -f aws-auth-cm-windows.yaml
```

4. ノードのステータスを監視し、Ready ステータスになるまで待機します。

```
kubectl get nodes --watch
```

Ctrl+C を入力して、シェルプロンプトに戻ります。

Note

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

ノードがクラスターに参加できない場合はトラブルシューティングの章にある「[the section called “ノードをクラスターに結合できません”](#)」を参照してください。

ステップ 3: その他のアクション

1. (オプション) [サンプルアプリケーション](#) をデプロイして、クラスターと Windows ノードをテストします。
2. (オプション) AmazonEKS_CNI_Policy マネージド IAM ポリシー (IPv4 クラスターがある場合) または [AmazonEKS_CNI_IPv6_Policy](#) (IPv6 クラスターがある場合には [ユーザー自身が作成したもの](#)) が [Amazon EKS ノードの IAM ロール](#) にアタッチされている場合、代わりに Kubernetes aws-node サービスアカウントに関連付けた IAM ロールに割り当てることをお勧めします。詳細については、「[the section called “IRSA 用の設定”](#)」を参照してください。
3. 次の条件が true の場合、IMDS への Pod アクセスをブロックすることをお勧めします。
 - IAM ロールをすべての Kubernetes サービスアカウントに割り当てることによって、必要最小限のアクセス許可のみを Pod に付与しようとしている。
 - クラスター内の Pod が、現在の AWS リージョンの取得など、その他の理由で Amazon EC2 インスタンスメタデータサービス (IMDS) へのアクセスを必要としていない。

詳細については「[ワーカーノードに割り当てられたインスタンスプロファイルへのアクセスを制限する](#)」を参照してください。

セルフマネージドの Ubuntu Linux ノードを作成する

Note

マネージド型ノードグループでは、ユースケースにいくつかの利点がある場合があります。詳細については、「[the section called “マネージド型ノードグループ”](#)」を参照してください。

このトピックでは、[Amazon Elastic Kubernetes Service \(EKS\) ノード上で Ubuntu](#) の Auto Scaling グループを起動するか、あるいは Amazon EKS クラスターに登録される [Amazon Elastic Kubernetes Service \(EKS\) ノード上で Ubuntu Pro](#) の Auto Scaling グループを起動する方法について説明します。EKS で動作する Ubuntu と Ubuntu Pro は、公式の Ubuntu Minimal LTS に基づいており、AWS と共同で開発されているカスタム AWS カーネルを搭載し、これまで特に EKS 向けに構築されています。Ubuntu Pro は、EKS 延長サポート期間、カーネルライブパッチ、FIPS の遵守、および無制限の Pro コンテナを実行する機能をサポートすることで、さらなるセキュリティの強化も図っています。

ノードがクラスターに参加したら、それらのノードにコンテナ化したアプリケーションをデプロイできるようになります。詳細については、「[Ubuntu on AWS](#)」のドキュメントおよび eksctl ドキュメントの「[Custom AMI support](#)」を参照してください。

Important

- Amazon EKS ノードは標準の Amazon EC2 インスタンスであり、通常の Amazon EC2 インスタンス価格に基づいて請求されます。詳細については「[Amazon EC2 料金](#)」を参照してください。
- AWS Outposts 上の Amazon EKS 拡張クラスターで Ubuntu ノードを起動できますが、AWS Outposts 上のローカルクラスターでは起動できません。詳細については、「[Amazon EKS on AWS Outposts](#)」を参照してください。

- x86 または Arm プロセッサを使用して Amazon EC2 インスタンスにデプロイできます。ただし、インスタンスに Inferentia チップがある場合は、先に [Neuron SDK](#) をインストールしておく必要があります。

この手順には、eksctl バージョン 0.210.0 以降が必要です。お使いのバージョンは、以下のコマンドを使用して確認できます。

```
eksctl version
```

eksctl のインストールまたはアップグレードの手順については、eksctl ドキュメントの「[インストール](#)」を参照してください。注意: この手順は、eksctl で作成されたクラスターでのみ機能します。

1. 次のコンテンツをデバイスにコピーします。my-cluster を自分のクラスター名に置き換えます。この名前には英数字 (大文字と小文字が区別されます) とハイフンのみを使用できます。先頭の文字はアルファベット文字である必要があります。また、100 文字より長くすることはできません。ng-ubuntu をノードグループの名前に置き換えます。ノードグループ名は 63 文字以下である必要があります。先頭は文字または数字でなければなりませんが、残りの文字にはハイフンおよびアンダースコアを含めることもできます。Arm インスタンスにデプロイするには、m5.large を Arm インスタンスタイプに置き換えます。my-ec2-keypair-name を、起動後に、SSH を使用してノードに接続するときに使用できる Amazon EC2 SSH キーペアの名前に置き換えます。Amazon EC2 キーペアをまだ持っていない場合は、AWS Management Console で作成できます。詳細については、「Amazon EC2 ユーザーガイド」の「[Amazon EC2 キーペア](#)」を参照してください。残りの#####はすべて独自の値に置き換えます。置き換えが完了したら、変更したコマンドを実行して ubuntu.yaml ファイルを作成します。

Important

ノードグループを AWS Outposts、AWS Wavelength、または AWS Local Zone サブネットにデプロイする場合、クラスターの作成時に AWS Outposts、AWS Wavelength、または AWS Local Zone サブネットは渡さないでください。次の例では、サブネットを指定する必要があります。詳細については、eksctl ドキュメントの「[設定ファイルからノードグループを作成する](#)」と「[Config ファイルのスキーマ](#)」を参照してください。**region-code** を、クラスターのある AWS リージョンに置き換えます。

```
cat >ubuntu.yaml <<EOF
---
apiVersion: eksctl.io/v1alpha5
kind: ClusterConfig

metadata:
  name: my-cluster
  region: region-code
  version: '1.32'

iam:
  withOIDC: true

nodeGroups:
- name: ng-ubuntu
  instanceType: m5.large
  desiredCapacity: 3
  amiFamily: Ubuntu2204
  iam:
    attachPolicyARNs:
      - arn:aws:iam::aws:policy/AmazonEKSWorkerNodePolicy
      - arn:aws:iam::aws:policy/AmazonEC2ContainerRegistryReadOnly
      - arn:aws:iam::aws:policy/AmazonSSMManagedInstanceCore
      - arn:aws:iam::aws:policy/AmazonEKS_CNI_Policy
  ssh:
    allow: true
    publicKeyName: my-ec2-keypair-name
EOF
```

Ubuntu Pro ノードグループを作成するには、amiFamily 値を UbuntuPro2204 に変更します。

2. 次のコマンドでノードをデプロイします。

```
eksctl create nodegroup --config-file=ubuntu.yaml
```

出力例は次のとおりです。

ノードの作成中に、複数の行が出力されます。出力の最後の行は次のサンプル行が表示されます。

```
[#] created 1 nodegroup(s) in cluster "my-cluster"
```

3. (オプション) [サンプルアプリケーション](#)をデプロイして Ubuntu ノードをテストします。
4. 次の条件が true の場合、IMDS への Pod アクセスをブロックすることをお勧めします。
 - IAM ロールをすべての Kubernetes サービスアカウントに割り当てることによって、必要最小限のアクセス許可のみを Pod に付与しようとしている。
 - クラスター内の Pod が、現在の AWS リージョンの取得など、その他の理由で Amazon EC2 インスタンスメタデータサービス (IMDS) へのアクセスを必要としていない。

詳細については「[ワーカーノードに割り当てられたインスタンスプロファイルへのアクセスを制限する](#)」を参照してください。

クラスターのためにセルフマネージドノードを更新する

Amazon EKS に最適化された最新の AMI がリリースされたら、セルフマネージド型ノードグループのノードを新しい AMI に置き換えることを検討してください。同様に、Amazon EKS クラスターの Kubernetes バージョンを更新した場合は、ノードを更新して、同じ Kubernetes バージョンのノードを使用します。

Important

このトピックでは、セルフマネージド型ノードのノードの更新について説明します。[マネージドノードグループ](#)を使用している場合は、「[the section called “更新”](#)」を参照してください。

クラスター内のセルフマネージド型ノードグループを更新して新しい AMI を使用するには、2 つの基本的な方法があります。

[アプリケーションを新しいノードグループに移行する](#)

新しいノードグループを作成し、Pod をそのグループに移行します。新しいノードグループへの移行は、既存の AWS CloudFormation スタックで AMI ID を単に更新するよりも適切です。これは、移行プロセスが古いノードグループを NoSchedule として[ティント](#)に設定し、新しいスタックが既存の Pod ワークロードを受け入れる準備ができた後にそのノードをドレインするためです。

AWS CloudFormation ノードスタックを更新する

既存のノードグループの AWS CloudFormation スタックを更新して、新しい AMI が使用されるようにします。この方法は、eksctl を使用して作成されたノードグループではサポートされていません。

アプリケーションを新しいノードグループに移行する

このトピックは、新しいノードグループを作成し、既存のアプリケーションを新しいグループに適切に移行し、クラスターから古いノードグループを削除する方法について説明します。新しいノードグループに移行するには、eksctl または AWS Management Console を使用します。

- [the section called “eksctl”](#)
- [the section called “AWS Management Console および AWS CLI”](#)

eksctl

eksctl を使用してアプリケーションを新しいノードグループに移行する

eksctl を移行に使用方法の詳細については、eksctl ドキュメントの「[管理対象外のノードグループ](#)」を参照してください。

この手順には、eksctl バージョン 0.210.0 以降が必要です。お使いのバージョンは、以下のコマンドを使用して確認できます。

```
eksctl version
```

eksctl のインストールまたはアップグレードの手順については、eksctl ドキュメントの「[インストール](#)」を参照してください。

Note

この手順は、eksctl で作成されたクラスターとノードグループに対してのみ機能します。

1. **my-cluster** をクラスター名に置き換えて、既存のノードグループの名前を取得します。

```
eksctl get nodegroups --cluster=my-cluster
```

出力例は次のとおりです。

CLUSTER	NODEGROUP	CREATED	MIN SIZE	MAX SIZE
DESIRED CAPACITY	INSTANCE TYPE	IMAGE ID		
default	standard-nodes	2019-05-01T22:26:58Z	1	4
	t3.medium	ami-05a71d034119ffc12		3

2. 次のコマンドを使用して、eksctl で新しいノードグループを起動します。コマンドのすべての####を独自の値に置き換えます。バージョン番号をお使いのコントロールプレーンの Kubernetes バージョンよりも新しいものにすることはできません。また、コントロールプレーンの Kubernetes バージョンより 3 つ以上前のマイナーなバージョンにすることもできません。コントロールプレーンと同じバージョンを使用することをお勧めします。

次の条件が true の場合、IMDS への Pod アクセスをブロックすることをお勧めします。

- IAM ロールをすべての Kubernetes サービスアカウントに割り当てることによって、必要最小限のアクセス許可のみを Pod に付与しようとしている。
- クラスター内の Pod が、現在の AWS リージョンの取得など、その他の理由で Amazon EC2 インスタンスメタデータサービス (IMDS) へのアクセスを必要としていない。

詳細については「[ワーカーノードに割り当てられたインスタンスプロファイルへのアクセスを制限する](#)」を参照してください。

Pod から IMDS へのアクセスをブロックするには、次のコマンドに `--disable-pod-imds` オプションを追加します。

Note

使用可能なフラグとその説明については、「<https://eksctl.io/>」を参照してください。

```
eksctl create nodegroup \
  --cluster my-cluster \
  --version 1.32 \
  --name standard-nodes-new \
  --node-type t3.medium \
  --nodes 3 \
  --nodes-min 1 \
  --nodes-max 4 \
  --managed=false
```


3. 前のコマンドが完了したら、次のコマンドを使用して、すべてのノードが Ready 状態になったことを確認します。

```
kubectl get nodes
```

4. 次のコマンドを使用して、元のノードグループを削除します。このコマンドで、すべての#####を自分のクラスター名とノードグループ名に置き換えます。

```
eksctl delete nodegroup --cluster my-cluster --name standard-nodes-old
```

AWS Management Console および AWS CLI

AWS Management Console と AWS CLI を使用してアプリケーションを新しいノードグループに移行する

1. 新しいノードグループを起動するには、「[セルフマネージド Amazon Linux ノードを作成する](#)」で説明されている手順に従います。
2. スタックの作成が完了したら、コンソールで選択し、[出力] を選択してください。
3. 作成されたノードグループの NodeInstanceRole を記録します。Amazon EKS ノードをクラスターに追加するには、これが必要です。

Note

追加の IAM ポリシーを古いノードグループの IAM ロールにアタッチした場合は、この同じポリシーを新しいノードグループの IAM ロールにアタッチして、新しいグループでその機能を管理します。これは、たとえば、Kubernetes [Cluster Autoscaler](#) のアクセス許可を追加した場合に適用されます。

4. 相互通信できるように、両方のノードグループのセキュリティグループを更新します。詳細については、「[the section called “セキュリティグループの要件”](#)」を参照してください。
 - a. 両方のノードグループのセキュリティグループ ID を記録します。これは、AWS CloudFormation スタックからの出力に、NodeSecurityGroup の値として表示されます。

次の AWS CLI コマンドを使用して、スタック名からセキュリティグループ ID を取得します。これらのコマンドで、oldNodes は、古いノードスタックの AWS CloudFormation スタック名、newNodes は、移行先のスタックの名前です。各#####は独自の値に置き換えます。

```
oldNodes="old_node_CFN_stack_name"
newNodes="new_node_CFN_stack_name"

oldSecGroup=$(aws cloudformation describe-stack-resources --stack-name $oldNodes \
--query 'StackResources[?
ResourceType=='AWS::EC2::SecurityGroup`].PhysicalResourceId' \
--output text)
newSecGroup=$(aws cloudformation describe-stack-resources --stack-name $newNodes \
--query 'StackResources[?
ResourceType=='AWS::EC2::SecurityGroup`].PhysicalResourceId' \
--output text)
```

- b. 互いにトラフィックを受け入れるように、各ノードのセキュリティグループに進入ルールを追加します。

以下の AWS CLI コマンドでは、他のセキュリティグループからのすべてのプロトコルのトラフィックをすべて許可するインバウンドルールを各セキュリティグループに追加します。この設定により、ワークロードを新しいグループに移行している際に、各ノードグループ内の Pod が相互に通信できるようになります。

```
aws ec2 authorize-security-group-ingress --group-id $oldSecGroup \
--source-group $newSecGroup --protocol -1
aws ec2 authorize-security-group-ingress --group-id $newSecGroup \
--source-group $oldSecGroup --protocol -1
```

5. `aws-auth configmap` を編集して、新しいノードインスタンスロールを RBAC にマッピングします。

```
kubectl edit configmap -n kube-system aws-auth
```

新しいノードグループの新しい `mapRoles` エントリを追加します。

```
apiVersion: v1
data:
  mapRoles: |
    - rolearn: ARN of instance role (not instance profile)
      username: system:node:{{EC2PrivateDNSName}}
      groups:
        - system:bootstrappers
        - system:nodes>
```

```
- rolearn: arn:aws:iam::111122223333:role/nodes-1-16-NodeInstanceRole-
U11V27W93CX5
  username: system:node:{{EC2PrivateDNSName}}
  groups:
    - system:bootstrappers
    - system:nodes
```

ARN (#####) スニペットを、[前の手順](#)で記録した NodeInstanceRole の値に置き換えます。次に、更新された configmap を適用するには、ファイルを保存して閉じます。

6. ノードのステータスを監視し、新しいノードがクラスターに結合され、Ready ステータスになるまで待機します。

```
kubectl get nodes --watch
```

7. (オプション) [Kubernetes Cluster Autoscaler](#) を使用している場合は、スケーリングアクションの競合を回避するために、デプロイをゼロ (0) レプリカにスケールダウンします。

```
kubectl scale deployments/cluster-autoscaler --replicas=0 -n kube-system
```

8. 以下のコマンドを使用して、NoSchedule で削除する各ノードをテイントに設定します。その目的は、ノードを置き換えたときにそのノードで新しい Pod がスケジュールまたは再スケジュールされないようにすることです。詳細については、Kubernetes ドキュメントの「[テイントと容認](#)」を参照してください。

```
kubectl taint nodes node_name key=value:NoSchedule
```

ノードを新しい Kubernetes バージョンにアップグレードする場合は、次のコードスニペットを使用して、特定の Kubernetes バージョン (この場合は 1.30) のすべてのノードを識別してテイントに設定できます。バージョン番号をお使いのコントロールプレーンの Kubernetes バージョンよりも新しいものにすることはできません。また、コントロールプレーンの Kubernetes バージョンより 3 つ以上前のマイナーなバージョンにすることもできません。コントロールプレーンと同じバージョンを使用することをお勧めします。

```
K8S_VERSION=1.30
nodes=$(kubectl get nodes -o jsonpath="{.items[?(@.status.nodeInfo.kubeletVersion==\"v$K8S_VERSION\")].metadata.name}")
for node in ${nodes[@]}
do
```

```
echo "Tainting $node"
kubectl taint nodes $node key=value:NoSchedule
done
```

9. クラスターの DNS プロバイダーを決定します。

```
kubectl get deployments -l k8s-app=kube-dns -n kube-system
```

出力例は次のとおりです。このクラスターでは、DNS 解決に CoreDNS を使用していますが、クラスターからは kube-dns が返される場合もあります。

NAME	DESIRED	CURRENT	UP-TO-DATE	AVAILABLE	AGE
coredns	1	1	1	1	31m

10 現在のデプロイメントで実行しているレプリカが 2 つ未満の場合は、そのデプロイメントを 2 つのレプリカにスケールアウトします。前のコマンドの出力が kubedns を返していた場合は、**coredns** を kubedns に置き換えます。

```
kubectl scale deployments/coredns --replicas=2 -n kube-system
```

11 次のコマンドを使用して、クラスターから削除する各ノードをドレーンします。

```
kubectl drain node_name --ignore-daemonsets --delete-local-data
```

ノードを新しい Kubernetes バージョンにアップグレードする場合は、次のコードスニペットを使用して、特定の Kubernetes バージョン (この場合は **1.30**) のすべてのノードを識別してドレーンします。

```
K8S_VERSION=1.30
nodes=$(kubectl get nodes -o jsonpath="{.items[?(@.status.nodeInfo.kubeletVersion==\"v${K8S_VERSION}\")].metadata.name}")
for node in ${nodes[@]}
do
    echo "Draining $node"
    kubectl drain $node --ignore-daemonsets --delete-local-data
done
```

12 古いノードのドレーンが完了したら、以前承認したセキュリティグループのインバウンドルールを取り消します。次に、AWS CloudFormation スタックを削除してインスタンスを終了してください。

 Note

さらに IAM ポリシーを古いノードグループの IAM ロールにアタッチした場合 ([Kubernetes Cluster Autoscaler](#) に対するアクセス許可を追加する場合など) は、ロールからその追加ポリシーをデタッチしてから、AWS CloudFormation スタックを削除できます。

- a. 先ほどノードのセキュリティグループ用に作成したインバウンドルールを取り消します。これらのコマンドで、oldNodes は、古いノードスタックの AWS CloudFormation スタック名、newNodes は、移行先のスタックの名前です。

```
oldNodes="old_node_CFN_stack_name"
newNodes="new_node_CFN_stack_name"

oldSecGroup=$(aws cloudformation describe-stack-resources --stack-name $oldNodes \
--query 'StackResources[?
ResourceType=='AWS::EC2::SecurityGroup'].PhysicalResourceId' \
--output text)
newSecGroup=$(aws cloudformation describe-stack-resources --stack-name $newNodes \
--query 'StackResources[?
ResourceType=='AWS::EC2::SecurityGroup'].PhysicalResourceId' \
--output text)
aws ec2 revoke-security-group-ingress --group-id $oldSecGroup \
--source-group $newSecGroup --protocol -1
aws ec2 revoke-security-group-ingress --group-id $newSecGroup \
--source-group $oldSecGroup --protocol -1
```

- b. [AWS CloudFormation コンソール](#)を開きます。
- c. 古いノードスタックを選択します。
- d. [削除] を選択します。
- e. [スタックの削除] 確認ダイアログボックスで、[スタックを削除] をクリックします。
- 13.aws-auth configmap を編集して、RBAC から古いノードインスタンスロールを削除します。

```
kubectl edit configmap -n kube-system aws-auth
```

古いノードグループの mapRoles エントリを削除します。

```
apiVersion: v1
data:
  mapRoles: |
    - rolearn: arn:aws:iam::111122223333:role/nodes-1-16-NodeInstanceRole-
W70725MZQFF8
      username: system:node:{{EC2PrivateDNSName}}
      groups:
        - system:bootstrappers
        - system:nodes
    - rolearn: arn:aws:iam::111122223333:role/nodes-1-15-NodeInstanceRole-
U11V27W93CX5
      username: system:node:{{EC2PrivateDNSName}}
      groups:
        - system:bootstrappers
        - system:nodes>
```

更新された configmap を適用するには、ファイルを保存して閉じます。

- 14.(オプション) Kubernetes [Cluster Autoscaler](#) を使用している場合は、デプロイのスケーリングを 1 レプリカに戻します。

 Note

必要に応じて、新しい Auto Scaling グループにタグ (k8s.io/cluster-autoscaler/enabled, k8s.io/cluster-autoscaler/my-cluster など) を付け、新しくタグ付けた Auto Scaling グループを参照するように、Cluster Autoscaler デプロイのコマンドを更新する必要があります。詳細については、「[AWS の Cluster Autoscaler](#)」を参照してください。

```
kubectl scale deployments/cluster-autoscaler --replicas=1 -n kube-system
```

- 15.(オプション) 最新バージョンの [Amazon VPC CNI Plugin for Kubernetes](#) を使用していることを確認します。サポートされている最新のインスタスタイプを利用するには CNI のバージョンを更新することが必要になる場合があります。詳細については、「[the section called “Amazon VPC CNI”](#)」を参照してください。

- 16.クラスターが DNS 解決 ([\[migrate-determine-dns-step\]](#) を参照) に kube-dns を使用している場合は、kube-dns デプロイを 1 レプリカにスケールインします。

```
kubectl scale deployments/kube-dns --replicas=1 -n kube-system
```

AWS CloudFormation ノードスタックを更新する

このトピックは、新しい AMI を使用して既存の AWS CloudFormation セルフマネージド型ノードスタックを更新する方法について説明します。この手順を使用して、クラスターの更新後にノードを新しいバージョンの Kubernetes に更新することができます。それ以外の場合は、Kubernetes の既存バージョン用の Amazon EKS に最適化された最新の AMI に更新することができます。

Important

このトピックでは、セルフマネージド型ノードのノードの更新について説明します。「[マネージドノードグループを使用してノードライフサイクルを簡素化する](#)」方法の詳細については、「[the section called “更新”](#)」を参照してください。

最新のデフォルトの Amazon EKS ノード AWS CloudFormation テンプレートは、古い AMI を一度に 1 つずつ削除する前に、新しい AMI を使用してインスタンスをクラスターに起動するように設定されています。この設定により、ローリング更新中にクラスター内で Auto Scaling グループのアクティブなインスタンスが常に必要数確保されるようになります。

Note

この方法は、eksctl を使用して作成されたノードグループではサポートされていません。eksctl を使用してクラスターまたはノードグループを作成した場合は、「[the section called “移行”](#)」を参照してください。

1. クラスターの DNS プロバイダーを決定します。

```
kubectl get deployments -l k8s-app=kube-dns -n kube-system
```

出力例は次のとおりです。このクラスターでは、DNS 解決に CoreDNS を使用していますが、クラスターからは kube-dns が返される場合があります。使用しているバージョンの kubectl によって、出力が異なる場合があります。

NAME	DESIRED	CURRENT	UP-TO-DATE	AVAILABLE	AGE
coredns	1	1	1	1	31m

2. 現在のデプロイメントで実行しているレプリカが 2 つ未満の場合は、そのデプロイメントを 2 つのレプリカにスケールアウトします。前のコマンドの出力が `kubedns` を返していた場合は、`coredns` を `kube-dns` に置き換えます。

```
kubectl scale deployments/coredns --replicas=2 -n kube-system
```

3. (オプション) [Kubernetes Cluster Autoscaler](#) を使用している場合は、スケーリングアクションの競合を回避するために、デプロイをゼロ (0) レプリカにスケールダウンします。

```
kubectl scale deployments/cluster-autoscaler --replicas=0 -n kube-system
```

4. 現在のノードグループのインスタンスタイプと必要なインスタンス数を決定します。グループの AWS CloudFormation テンプレートを更新する場合は、後でこれらの値を入力します。
- Amazon EC2 コンソール (<https://console.aws.amazon.com/ec2/>) を開きます。
 - 左ナビゲーションペインの [Launch Configurations] (起動設定) を選択し、既存のノードの起動設定のインスタンスタイプを書き留めます。
 - 左ナビゲーションペインの [Auto Scaling] (グループ) を選択し、既存の Auto Scaling グループのノードの、インスタンスの [Desired] (必要) 数を書き留めます。
5. [AWS CloudFormation コンソール](#)を開きます。
6. ノードグループスタックを選択し、[更新] を選択します。
7. [Replace current template] (現在のテンプレートを置き換える) を選択し、Amazon S3 URL を選択します。
8. Amazon S3 URL で、ノードの AWS CloudFormation テンプレートの最新バージョンを使用していることを確認するために以下の URL をテキストエリアに貼り付けます。続いて、[次へ] を選択します。
- ```
https://s3.us-west-2.amazonaws.com/amazon-eks/cloudformation/2022-12-23/amazon-eks-nodegroup.yaml
```
9. [Specify stack details] (スタックの詳細の指定) ページで、以下のパラメータを指定し、[Next] (次へ) を選択します。



- [NodeAutoScalingGroupDesiredCapacity] – [前のステップ](#)で記録した必要なインスタンス数を入力します。または、スタック更新時にスケーリングする必要のある新しいノード数を入力します。
- NodeAutoScalingGroupMaxSize - ノードの Auto Scaling グループがスケールアウトする最大ノード数を入力します。この値は、必要な容量よりも少なくとも 1 ノード多い必要があります。これは、更新時にノード数を減らさずにノードのローリング更新を実行できるようにするためです。
- NodeInstanceType — [前のステップ](#)で記録したインスタンスタイプを選択します。または、ノードに別のインスタンスタイプを選択します。異なるインスタンスタイプを選択する前に、「[最適な Amazon EC2 ノードインスタンスタイプを選択する](#)」を確認してください。各 Amazon EC2 インスタンスタイプは最大数の Elastic Network Interface (ネットワークインターフェイス) をサポートし、各ネットワークインターフェイスは最大数の IP アドレスをサポートします。ワーカーノードと Pod には、それぞれ IP アドレスが割り当てられています。そのため、各 Amazon EC2 ノードで実行する Pod の最大数をサポートするインスタンスタイプを選択することが重要です。インスタンスタイプごとにサポートされるネットワークインターフェイスと IP アドレスのリストについては、「[各インスタンスタイプのネットワークインターフェイスごとの IP アドレス](#)」を参照してください。例えば、m5.large インスタンスタイプは、ワーカーノードと Pod に対して最大 30 の IP アドレスをサポートします。

 Note

最新バージョンの [Amazon VPC CNI plugin for Kubernetes](#) でサポートされているインスタンスタイプは、GitHub 上の [vpc\\_ip\\_resource\\_limit.go](#) に示されています。サポートされている最新のインスタンスタイプを利用するには、Amazon VPC CNI Plugin for Kubernetes のバージョンを更新することが必要になる場合があります。詳細については、「[the section called “Amazon VPC CNI”](#)」を参照してください。

 Important

AWS リージョンによっては利用できないインスタンスタイプがあります。

- [NodeImageIdSSMParam]: 更新する AMI ID の Amazon EC2 Systems Manager パラメータです。次の値では、Kubernetes バージョン 1.32 用の最新の Amazon EKS 最適化 AMI が使用されています。

```
/aws/service/eks/optimized-ami/1.32/amazon-linux-2/recommended/image_id
```

1.32 は、同一の[サポートされている Kubernetes バージョン](#)に置き換えることができます。または、コントロールプレーンで実行されている Kubernetes バージョンより 1 つ前までのバージョンである必要があります。ノードは、コントロールプレーンと同じバージョンにしておくことをお勧めします。[amazon-linux-2](#) を別の AMI タイプに置き換えることもできます。詳細については、「[the section called “最新の ID を取得する”](#)」を参照してください。

#### Note

Amazon EC2 Systems Manager パラメータを使用すると、AMI ID を検索して指定することなく、後でノードを更新できます。AWS CloudFormation スタックがこの値を使用している場合、スタックの更新によって常に、指定された Kubernetes バージョンで推奨される最新の Amazon EKS 最適化 AMI が起動されます。これは、テンプレートの値を変更しない場合も同様です。

- Nodelmageld - 独自のカスタム AMI を使用するには、使用する AMI の ID を入力します。

#### Important

この値は、NodelmageldSSMPParam に指定されたすべての値を上書きします。NodelmageldSSMPParam 値を使用する場合は、Nodelmageld の値が空白であることを確認します。

- [DisableIMDSv1] – 各ノードは、デフォルトでインスタンスメタデータサービスバージョン 1 (IMDSv1) および IMDSv2 をサポートします。ただし、IMDSv1 を無効にできます。ノードグループでスケジュールされているノードまたは Pod で IMDSv1 を使用しない場合は、[true] を選択します。IDMS の詳細については「[インスタンスメタデータサービスの設定](#)」を参照してください。サービスアカウントの IAM ロールを実装した場合は、AWS のサービスへのアクセスを必要とするすべての Pod に対し、必要なアクセス許可を直接割り当てます。これにより、クラスター内の Pod は、現在の AWS リージョンの取得など、その他の理由で IMDS へのアクセスを必要としません。次に、ホストネットワークを使用しない Pod の IMDSv2 へのアクセスを無効にすることもできます。詳細については、[ワーカーノードに割り当てられたインスタンスプロファイルへのアクセスを制限する](#)を参照してください。

10(オプション) オプション ページで、スタックリソースをタグ付けします。[Next] を選択します。

- 11[確認] ページで情報を確認し、スタックで IAM リソースを作成することを承認して、[スタックの更新] を選択します。

 Note

クラスター内の各ノードの更新には数分かかります。すべてのノードの更新が完了するのを待ってから、次の手順を実行します。

- 12.クラスターの DNS プロバイダーが kube-dns の場合は、kube-dns デプロイを 1 レプリカにスケールインします。

```
kubectl scale deployments/kube-dns --replicas=1 -n kube-system
```

- 13(オプション) Kubernetes [Cluster Autoscaler](#) を使用している場合は、デプロイのスケーリングを目的のレプリカ数に戻します。

```
kubectl scale deployments/cluster-autoscaler --replicas=1 -n kube-system
```

- 14(オプション) 最新バージョンの [Amazon VPC CNI Plugin for Kubernetes](#) を使用していることを確認します。サポートされている最新のインスタンスタイプを利用するには、Amazon VPC CNI Plugin for Kubernetes のバージョンを更新することが必要になる場合があります。詳細については、「[the section called “Amazon VPC CNI”](#)」を参照してください。

## AWS Fargate を使用してコンピューティング管理を簡素化する

このトピックでは、Amazon EKS を使用して AWS Fargate で Kubernetes Pod を実行する方法について説明します。Fargate は、[コンテナ](#)に適切なサイズのコンピューティング能力をオンデマンドで提供するテクノロジーです。Fargate を使用すると、コンテナを実行するために仮想マシンのグループをプロビジョニング、設定、スケールする必要がありません。これにより、サーバータイプの選択、ノードグループをスケールするタイミングの決定、クラスターのパッキングの最適化を行う必要がなくなります。

Fargate で起動する Pod と、[Fargate プロファイル](#)での実行方法を制御できます。Fargate プロファイルは Amazon EKS クラスターの一部として定義されています。Amazon EKS は、Kubernetes が提供するアップストリームの拡張可能なモデルを使用して AWS により構築されたコントローラーを使用して、Kubernetes と Fargate を統合します。これらのコントローラーは、Amazon EKS マネージド Kubernetes コントロールプレーンの一部として実行され、ネイティブ Kubernetes Pod を Fargate にスケジューリングします。Fargate コントローラーには、いくつかの変更と検証アドミッ

ションコントローラーに加えて、デフォルトの Kubernetes スケジューラーとともに実行される新しいスケジューラが含まれています。Fargate で実行する条件を満たす Pod を起動すると、クラスターで実行されている Fargate コントローラーは Pod を認識し、更新し、Fargate にスケジューリングします。

このトピックでは、Fargate で実行されている Pod のさまざまなコンポーネントについて説明し、Amazon EKS で Fargate を使用する際の特別な考慮事項を示します。

## AWS Fargate 考慮事項

Amazon EKS での Fargate の使用についての考慮事項を以下に示します。

- Fargate で実行される各 Pod には、独自の分離境界があります。基本となるカーネル、CPU リソース、メモリリソース、または Elastic Network Interface を別の Pod と共有しません。
- Network Load Balancer と Application Load Balancer (ALBs) は、IPターゲットでのみ Fargate で使用できます。詳細については、[the section called “ネットワークロードバランサーを作成する”](#) および [the section called “アプリケーション負荷分散”](#) を参照してください。
- Fargate 公開サービスは、ターゲットタイプの IP モードでのみ実行され、ノード IP モードでは実行されません。マネージド型ノードで実行されているサービスと Fargate で実行されているサービスからの接続を確認するためのおすすめの方法は、サービス名を使用して接続することです。
- Fargate で実行するには、ポッドがスケジューリングされた時点で Fargate プロファイルと一致する必要があります。Fargate プロファイルに一致しない Pod は Pending としてスタックする可能性があります。一致する Fargate プロファイルが存在する場合、Fargate に再スケジューリングするために作成した保留中の Pod を削除できます。
- デーモンセットは Fargate ではサポートされていません。アプリケーションにデーモンが必要な場合は、そのデーモンを Pod のサイドカーコンテナとして実行するように再設定します。
- 特権を持つコンテナは Fargate ではサポートされていません。
- Fargate で実行されている Pod は、HostPort または HostNetwork を Pod マニフェストに指定できません。
- Fargate Pod の場合、デフォルトの `nofile` および `nproc` のソフトリミットは 1024、ハードリミットは 65535 です。
- GPU は現在、Fargate では使用できません。
- Fargate で実行されているポッドは、プライベートサブネットでのみサポートされています (AWS のサービスへの NAT ゲートウェイアクセスができますが、インターネットゲートウェイへの直接ルーティングはできません)。そのため、クラスターの VPC ではプライベートサブネットを使用で

きるようにする必要があります。アウトバウンドインターネットアクセスのないクラスターについては、「」を参照してください[the section called “プライベートクラスター”](#)

- 「[Vertical Pod Autoscaler でポッドリソースを調整する](#)」を使用して Fargate Pod の CPU とメモリの適切な初期サイズを設定し、「[Horizontal Pod Autoscaler を使用してポッドデプロイをスケールする](#)」を使用してそれらの Pod をスケールできます。Vertical Pod Autoscaler で、より大きい CPU とメモリの組み合わせを持つ Fargate に Pod を自動的に再デプロイする場合は、正しい機能を確認するため、Vertical Pod Autoscaler のモードを Auto または Recreate に設定します。詳細については、GitHub で [Vertical Pod Autoscaler](#) のドキュメントを参照してください。
- VPC で DNS 解決と DNS ホスト名を有効にする必要があります。詳細については、「[VPC の DNS サポートを更新する](#)」を参照してください。
- Amazon EKS Fargate は、仮想マシン (VM) 内の各ポッドを分離することで、Kubernetes アプリケーションの多重防御を追加します。この VM 境界は、コンテナエスケープが発生した場合に他のポッドが使用するホストベースのリソースへのアクセスを防ぎます。これは、コンテナ化されたアプリケーションを攻撃し、コンテナ外のリソースにアクセスする一般的な方法です。

Amazon EKS を使用しても、[責任共有モデル](#)に基づくお客様の責任は変わりません。クラスターのセキュリティとガバナンスのコントロールの設定について、慎重に検討する必要があります。アプリケーションを分離する最も安全な方法は、常に別のクラスターで実行することです。

- Fargate プロファイルでは、VPC セカンダリ CIDR ブロックからのサブネットの指定がサポートされています。セカンダリ CIDR ブロックを指定することもできます。サブネットの使用できる IP アドレスの数は限られています。結果的に、クラスター内に作成できる Pod の数も制限されます。Pod に別のサブネットを使用することで、使用可能な IP アドレスの数を増やすことができます。詳細については、「[IPv4 CIDR ブロックの VPC への追加](#)」を参照してください。
- Amazon EC2 インスタンスメタデータサービス (IMDS) は、Fargate ノードにデプロイされた Pod では使用できません。IAM 認証情報を必要とする Fargate にデプロイされた Pod がある場合は、[サービスアカウントの IAM ロール](#)を使用して Pod に割り当てます。Pod が IMDS を通じて利用可能な他の情報にアクセスする必要がある場合は、この情報を Pod 仕様にハードコーディングする必要があります。これには、Pod がデプロイされる AWS リージョンまたはアベイラビリティゾーンが含まれます。
- AWS Outposts、AWS Wavelength、または AWS Local Zones に Fargate Pod をデプロイすることはできません。
- Amazon EKS は定期的に Fargate Pod にパッチを適用して安全に保つ必要があります。影響を軽減する方法で更新を試みますが、Pod のエビクションが正常に行われなかった場合、Pod を削除しなければならない場合があります。中断を最小限に抑えるために行えるアクションがいくつかあります。詳細については、「[the section called “OS パッチ適用イベント”](#)」を参照してください。

- [Amazon VPC CNI plugin for Amazon EKS](#) は、Fargate ノードにインストールされています。Fargate ノードでは [Amazon EKS クラスターの代替 CNI プラグイン](#) を使用することはできません。
- Fargate 上で実行される Pod は、ドライバーの手動インストールステップなしで、Amazon EFS ファイルシステムを自動的にマウントします。Fargate ノードでは動的永続ボリュームプロビジョニングを使用できませんが、静的プロビジョニングを使用することはできます。
- Amazon EKS は Fargate Spot をサポートしていません。
- Amazon EBS ボリュームを Fargate Pod にマウントすることはできません。
- Amazon EBS CSI コントローラーは Fargate ノードで実行できますが、Amazon EBS CSI ノード DaemonSet は Amazon EC2 インスタンスでのみ実行できます。
- [Kubernetes Job](#) が Completed または Failed とマークされた後も、Job が作成した Pod は通常存在し続けます。この動作によりログと結果を表示できますが、Fargate を使用する場合、その後 Job をクリーンアップしないとコストが発生します。

Job が完了または失敗した後に、関連する Pod を自動的に削除するには、有効期限 (TTL) コントローラーを使用して期間を指定できます。次の例では、Job マニフェストで `.spec.ttlSecondsAfterFinished` を指定する方法を示しています。

```
apiVersion: batch/v1
kind: Job
metadata:
 name: busybox
spec:
 template:
 spec:
 containers:
 - name: busybox
 image: busybox
 command: ["/bin/sh", "-c", "sleep 10"]
 restartPolicy: Never
 ttlSecondsAfterFinished: 60 # <-- TTL controller
```

## Fargate 比較テーブル

| 条件                                   | AWS Fargate |
|--------------------------------------|-------------|
| <a href="#">AWS Outposts</a> にデプロイ可能 | いいえ         |



| 条件                                                               | AWS Fargate                                            |
|------------------------------------------------------------------|--------------------------------------------------------|
| <a href="#">AWS Local Zone</a> にデプロイ可能                           | いいえ                                                    |
| Windows を必要とするコンテナを実行できる                                         | いいえ                                                    |
| Linux を必要とするコンテナを実行可能                                            | はい                                                     |
| インフュクエンティア チップを必要とするワークロードを実行可能                                  | いいえ                                                    |
| GPU を必要とするワークロードを実行可能                                            | いいえ                                                    |
| Arm プロセッサを必要とするワークロードを実行可能                                       | いいえ                                                    |
| AWS <a href="#">Bottlerocket</a> を実行可能                           | いいえ                                                    |
| Pod は、カーネルランタイム環境を他の Pod と共有します。                                 | いいえ: 各 Pod には専用のカーネルがあります。                             |
| ポッドはCPU、メモリ、ストレージ、およびネットワークリソースを他のポッドと共有します。                     | いいえ — 各 Pod には専用のリソースがあり、リソース使用率を最大化するために個別にサイズ設定できます。 |
| Pod 仕様で要求されているよりも多くのハードウェアとメモリを、Pod が使用できます。                     | いいえ: より大きな vCPU およびメモリ設定を使用して、Pod を再デプロイできます。          |
| Amazon EC2 インスタンスをデプロイおよび管理する必要がある                               | いいえ                                                    |
| Amazon EC2 インスタンスのオペレーティングシステムの保護、保守、パッチ適用を行う必要がある               | いいえ                                                    |
| ノードをデプロイする時に、追加の <a href="#">kubelet</a> 引数などのブートストラップ引数を提供できます。 | いいえ                                                    |

| 条件                                                               | AWS Fargate                                               |
|------------------------------------------------------------------|-----------------------------------------------------------|
| ノードに割り当てられた IP アドレスとは異なる CIDR ブロックから、IP アドレスを Pod に割り当てることができます。 | いいえ                                                       |
| ノードに SSH 接続可能                                                    | いいえ – SSH 接続を行うノードホストオペレーティングシステムはありません。                  |
| 独自のカスタム AMI をノードにデプロイ可能                                          | いいえ                                                       |
| 独自のカスタム CNI をノードにデプロイ可能                                          | いいえ                                                       |
| ノード AMI を自分で更新する必要がある                                            | いいえ                                                       |
| ノードの Kubernetes バージョンを自分で更新する必要があります。                            | いいえ: ノードを管理しません。                                          |
| Pod で Amazon EBS ストレージを使用可能                                      | いいえ                                                       |
| Pod で Amazon EFS ストレージを使用可能                                      | <a href="#">あり</a>                                        |
| Pod で Amazon FSx for Lustre ストレージを使用可能                           | いいえ                                                       |
| Network Load Balancer をサービスに使用可能                                 | はい、「 <a href="#">Network Load Balancer を作成する</a> 」を使用する場合 |
| ポッドはパブリックサブネットで実行可能                                              | いいえ                                                       |
| それぞれの Pod に、異なる VPC セキュリティグループを割り当て可能                            | はい                                                        |
| Kubernetes DaemonSets を実行可能                                      | いいえ                                                       |
| Pod マニフェストで HostPort および HostNetwork をサポートします                    | いいえ                                                       |
| AWS が利用可能なリージョン                                                  | <a href="#">一部の Amazon EKS 対応リージョン</a>                    |



| 条件                           | AWS Fargate                                                                                         |
|------------------------------|-----------------------------------------------------------------------------------------------------|
| Amazon EC2 専有ホストでコンテナを実行できます | いいえ                                                                                                 |
| 料金                           | 個々のFargate メモリとCPU構成のコスト。各 Pod には、独自のコストがあります。詳細については、「 <a href="#">AWS Fargate の料金</a> 」を参照してください。 |

## クラスターの AWS Fargate の使用を開始する

このトピックでは、Amazon EKS クラスターを使用して AWS Fargate で Pod の実行を開始する方法について説明します。

CIDR ブロックを使用してクラスターのパブリックエンドポイントへのアクセスを制限する場合は、プライベートエンドポイントアクセスも有効にすることをお勧めします。こうすることで、Fargate Pod がクラスターと通信できるようになります。プライベートエンドポイントが有効になっていない場合、パブリックアクセスに指定する CIDR ブロックに、VPC からのアウトバウンドソースを含める必要があります。詳細については、「[the section called “クラスターエンドポイントのアクセス”](#)」を参照してください。

### 前提条件

既存のクラスター。既存の Amazon EKS クラスターがなければ、[はじめに](#) を参照してください。

### ステップ 1: 既存のノードが Fargate Pod と通信できることを確認する

ノードのない新しいクラスター、またはマネージド型ノードグループ ([the section called “マネージド型ノードグループ”](#) を参照) のみを持つクラスターを使用している場合は、「[the section called “ステップ 2: Fargate Pod 実行ロールを作成する”](#)」に進みます。

既にそれに関連付けられているノードがある既存のクラスターで作業していると仮定します。これらのノードの Pod が Fargate で実行されている Pod と自由に通信できることを確認する必要があります。Fargate で実行されている Pod は、関連付けられているクラスターのクラスターセキュリティグループを使用するように自動的に設定されます。クラスター内の既存のノードが、クラスターセキュリティグループとの間でトラフィックを送受信できることを確認します。マネージドノードグループもクラスターセキュリティグループを使用するように自動的に設定されるため、この互換性

を変更または確認する必要はありません (「[the section called “マネージド型ノードグループ”](#)」を参照)。

eksctl または Amazon EKS マネージド型 AWS CloudFormation テンプレートで作成された既存のノードグループの場合、クラスターセキュリティグループをノードに手動で追加できます。または、ノードグループの Auto Scaling グループ起動テンプレートを変更して、クラスターセキュリティグループをインスタンスにアタッチすることもできます。詳細については、Amazon VPC ユーザーガイドの[インスタンスのセキュリティグループを変更する](#)を参照してください。

AWS Management Console で、クラスターの [Networking] (ネットワーク) セクションでクラスターのクラスターセキュリティグループを確認できます。これを行うには、次の AWS CLI コマンドを実行します。このコマンドを使用する場合は、<my-cluster> を自分のクラスター名に置き換えます。

```
aws eks describe-cluster --name <my-cluster> --query
cluster.resourcesVpcConfig.clusterSecurityGroupId
```

## ステップ 2: Fargate Pod 実行ロールを作成する

クラスターが AWS Fargate で Pod を作成する場合、Fargate インフラストラクチャで実行されるコンポーネントは、ユーザーに代わって AWS API を呼び出す必要があります。Amazon EKS の Pod 実行ロールにより、これらを行うための IAM アクセス許可が付与されます。AWS Fargate Pod 実行ロールを作成するには、「[the section called “ポッド実行IAMロール”](#)」を参照してください。

### Note

--fargate オプションを使用して eksctl でクラスターを作成した場合は、クラスターに Pod 実行ロールが既にあり、これは eksctl-my-cluster-FargatePodExecutionRole-ABCDEFGHijkl というパターンで IAM コンソールに表示されます。同様に、eksctl を使用して Fargate プロファイルを作成する場合、eksctl では Pod 実行ロールを作成します (まだ存在しない場合)。

## ステップ 3: クラスターの Fargate プロファイルを作成する

クラスターの Fargate で実行されている Pod をスケジューリングする前に、起動時に Fargate を使用する Pod を指定する Fargate プロファイルを定義する必要があります。詳細については、「[the section called “プロファイルを定義する”](#)」を参照してください。

**Note**

--fargate オプションを使用して eksctl でクラスターを作成した場合、クラスターの Fargate プロファイルは、kube-system と default の名前空間のすべての Pod のセレクトラを使用して既に作成されています。Fargate で使用するその他の名前空間の Fargate プロファイルを作成するには、以下の手順に従います。

Fargate プロファイルを作成するには、次のツールのいずれかを使用します。

- [the section called “eksctl”](#)
- [the section called “AWS Management Console”](#)

**eksctl**

この手順には、eksctl バージョン 0.210.0 以降が必要です。お使いのバージョンは、以下のコマンドを使用して確認できます。

```
eksctl version
```

eksctl のインストールまたはアップグレードの手順については、eksctl ドキュメントの「[インストール](#)」を参照してください。

**eksctl** で、Fargate プロファイルを作成するには

以下の eksctl コマンドで Fargate プロファイルを作成し、すべての <example value> を自分の値に置き換えます。名前空間を指定する必要があります。ただし、--labels オプションは必須ではありません。

```
eksctl create fargateprofile \
 --cluster <my-cluster> \
 --name <my-fargate-profile> \
 --namespace <my-kubernetes-namespace> \
 --labels <key=value>
```

<my-kubernetes-namespace> および <key=value> ラベルには、特定のワイルドカードを使用できます。詳細については、「[the section called “Fargate プロファイルのワイルドカード”](#)」を参照してください。

## AWS Management Console

AWS Management Console で、Fargate プロファイルを作成するには

1. [Amazon EKS コンソール](#)を開きます。
2. Fargate プロファイルを作成するクラスターを選択します。
3. [コンピューティング] タブを開きます。
4. [Fargate プロファイル] で、[Fargate プロファイルを追加] を選択します。
5. [Fargate プロファイルを設定] ページで、次の操作を行います。
  - a. [名前] に Fargate プロファイルの名前を入力します。名前は一意である必要があります。
  - b. [Pod 実行ロール] で、Fargate プロファイルで使用する Pod 実行ロールを選択します。eks-fargate-pods.amazonaws.com サービスプリンシパルを持つ IAM ロールのみが表示されます。ここにロールが表示されない場合は、ロールを作成する必要があります。詳細については、「[the section called “ポッド実行IAMロール”](#)」を参照してください。
  - c. 選択した [サブネット] を必要に応じて変更します。

 Note

Fargate で実行される Pod では、プライベートサブネットのみがサポートされます。

- d. [Tags] (タグ) では、オプションで Fargate プロファイルにタグを付けることができます。これらのタグは、Pod など、プロファイルに関連付けられた他のリソースには伝達されません。
  - e. [次へ] を選択します。
6. [Pod の選択を設定] ページで、次の操作を行います。
    - a. [名前空間] に、Pod と照合する名前空間を入力します。
      - kube-system または default など、特定の名前空間を使用して照合できます。
      - 特定のワイルドカード (例: prod-\*) を使用して、複数の名前空間 (例: prod-deployment および prod-test) と照合することができます。詳細については、「[the section called “Fargate プロファイルのワイルドカード”](#)」を参照してください。
    - b. (オプション) セレクタに Kubernetes ラベルを追加します。特に、指定された名前空間内の Pod が一致する必要があるものにそれらを追加します。
      - ラベル infrastructure: fargate をセレクターに追加して、infrastructure: fargate Kubernetes ラベルも持つ指定された名前空間の Pod のみがセレクターと一致するようにすることができます。

- 特定のワイルドカード (例: key?: value?) を使用して、複数の名前空間 (例: keya: valuea および keyb: valueb) と照合することができます。詳細については、「[the section called “Fargate プロファイルのワイルドカード”](#)」を参照してください。

c. [Next] (次へ) を選択します。

7. [確認と作成] ページで、Fargate プロファイルの情報を確認し、[作成] を選択します。

## ステップ 4: CoreDNS を更新する

デフォルトでは、CoreDNS は Amazon EKS クラスターの Amazon EC2 インフラストラクチャで実行するように設定されています。クラスター内で Fargate でのみ Pod を実行する場合は、次の手順を実行します。

### Note

--fargate オプションを使用して eksctl でクラスターを作成した場合は、[the section called “次のステップ”](#) にスキップできます。

1. 次のコマンドを使用して、CoreDNS の Fargate プロファイルを作成します。<my-cluster> をクラスター名、<111122223333> をアカウント ID、<AmazonEKSFargatePodExecutionRole> をポッド実行ロール名に置き換え、また、<0000000000000000a>、<0000000000000000b>、<0000000000000000c> をプライベートサブネットの ID に置き換えます。Pod 実行ロールがない場合は、最初に作成する必要があります (「[the section called “ステップ 2: Fargate Pod 実行ロールを作成する”](#)」を参照)。

### Important

ロール ARN に / 以外の[パス](#)を含めることはできません。例えば、ロールの名前が development/apps/AmazonEKSFargatePodExecutionRole の場合、ロールの ARN を指定するときに AmazonEKSFargatePodExecutionRole に変更する必要があります。ロール ARN の形式は arn:aws:iam::<111122223333>:role/<AmazonEKSFargatePodExecutionRole> であることが必要です。

```
aws eks create-fargate-profile \
 --fargate-profile-name coredns \
 --cluster-name <my-cluster> \

```

```
--pod-execution-role-arn arn:aws:iam::<111122223333>:role/
<AmazonEKSFargatePodExecutionRole> \
--selectors namespace=kube-system,labels={k8s-app=kube-dns} \
--subnets subnet-<0000000000000000a> subnet-<0000000000000000b> subnet-
<0000000000000000c>
```

2. coredns デプロイのロールアウトをトリガーします。

```
kubectl rollout restart -n kube-system deployment coredns
```

## 次のステップ

- 以下のワークフローを使用すると、Fargate で実行するために既存のアプリケーションの移行を開始できます。
  - a. アプリケーションの Kubernetes 名前空間と Kubernetes ラベルに一致する [the section called “Fargate プロファイルの作成”](#)。
  - b. 既存のいずれかの Pod を削除および再作成し、Fargate でスケジューリングされるようにします。<namespace> と <deployment-type> を変更して、特定のポッドを更新します。

```
kubectl rollout restart -n <namespace> deployment <deployment-type>
```

- [the section called “アプリケーション負荷分散”](#) をデプロイして、Fargate で実行されている Pod が Ingress オブジェクトを使用できるようにします。
- [the section called “Vertical Pod Autoscaler”](#) を使用して Fargate Pod の CPU とメモリの適切な初期サイズを設定し、[the section called “Horizontal Pod Autoscaler”](#) を使用してそれらの Pod をスケールできます。Vertical Pod Autoscaler で、より上位の CPU とメモリの組み合わせを持つ Fargate に Pod を自動的に再デプロイする場合は、Vertical Pod Autoscaler のモードを Auto または Recreate に設定します。これは、正しい機能を保証するためです。詳細については、GitHub で [Vertical Pod Autoscaler](#) のドキュメントを参照してください。
- [これらの手順](#)に従って、アプリケーションをモニタリングするための [AWS Distro for OpenTelemetry](#) (ADOT) コレクターをセットアップします。

## 起動時にどの Pod が AWS Fargate を使用するのかを定義する

クラスターの Fargate で Pod をスケジューリングする前に、起動時に Fargate を使用する Pod を指定する Fargate プロファイルを少なくとも 1 つ定義する必要があります。

管理者は、Fargate プロファイルを使用してどの Pod が Fargate で実行されるかを宣言できます。これはプロファイルのセレクターを介して行うことができます。1 つのプロファイルに最大 5 つのセレクターを追加できます。各セレクターには名前空間が含まれている必要があります。セレクターにはラベルを含めることもできます。ラベルフィールドは、複数のオプションのキーと値のペアで構成されます。セレクターに一致するポッドは Fargate でスケジューリングされます。ポッドは、セレクターで指定された名前空間とラベルを使用して照合されます。名前空間セレクターがラベルなしで定義されている場合、Amazon EKS は、プロファイルを使用して、その名前空間で実行されるすべての Pod を Fargate にスケジューリングしようとします。スケジューリングされる Pod が Fargate プロファイル内のいずれかのセレクターと一致する場合、その Pod は Fargate でスケジューリングされます。

Pod が複数の Fargate プロファイルと一致する場合、次の Kubernetes ラベルを Pod 仕様に追加することで、Pod がどのプロファイルを使用するかを指定することができます：

`eks.amazonaws.com/fargate-profile: my-fargate-profile`。Pod を Fargate にスケジューリングするには、そのプロファイル内のセレクターと一致する必要があります。Kubernetes アフィニティ/アンチアフィニティルールは適用されないため、Amazon EKS Fargate Pod では必要ありません。

Fargate プロファイルを作成する際は、Pod 実行ロールを指定する必要があります。この実行ロールは、プロファイルを使用して Fargate インフラストラクチャで実行される Amazon EKS コンポーネントのためのものです。承認のためにクラスターの Kubernetes [ロールベースのアクセスコントロール](#) (RBAC) に追加されます。これにより、Fargate インフラストラクチャで実行されている kubelet が Amazon EKS クラスターに登録され、クラスター内でノードとして表示されるようになります。Pod 実行ロールは、Amazon ECR イメージリポジトリへの読み取りアクセスを許可するために、Fargate インフラストラクチャへの IAM アクセス許可も提供します。詳細については、「[the section called “ポッド実行IAMロール”](#)」を参照してください。

Fargate プロファイルは変更できません。ただし、新しい更新されたプロファイルを作成して既存のプロファイルを置き換え、その後元のプロファイルを削除することはできます。

#### Note

Fargate プロファイルを使用して実行中の Pod は、プロファイルが削除されると停止され、保留状態になります。



クラスターのいずれかの Fargate プロファイルが DELETING ステータスである場合は、Fargate プロファイルの削除が完了するのを待ってから、そのクラスターに他のプロファイルを作成する必要があります。

#### Note

Fargate は現在 Kubernetes [topologySpreadConstraints](#) をサポートしていません。

Amazon EKS と Fargate は、Fargate プロファイルで定義されている各サブネットに Pod を分散させます。ただし、分散が偏ってしまう場合があります。均等な分散が必要な場合は、2 つの Fargate プロファイルを使用してください。2 つのレプリカをデプロイし、ダウンタイムを発生させたくないシナリオでは、均等な分散が重要になります。各プロファイルにはサブネットを 1 つだけ含めることをお勧めします。

## Fargate プロファイルのコンポーネント

Fargate プロファイルには、次のコンポーネントが含まれています。

### ポッド実行ロール

クラスターが AWS Fargate で Pod を作成するとき、Fargate インフラストラクチャ上で実行されている kubelet は、ユーザーに代わって AWS API を呼び出す必要があります。例えば、Amazon ECR からコンテナイメージを取得するためのコールを行う必要があります。Amazon EKS の Pod 実行ロールにより、これらを行うための IAM アクセス許可が付与されます。

Fargate プロファイルを作成する際は、Pod で使用する Pod 実行ロールを指定する必要があります。このロールは、承認のためにクラスターの Kubernetes [ロールベースのアクセスコントロール](#) (RBAC) に追加されます。これにより、Fargate インフラストラクチャで実行されている kubelet が Amazon EKS クラスターに登録され、クラスター内でノードとして表示されるようになります。詳細については、「[the section called “ポッド実行IAMロール”](#)」を参照してください。

### サブネット

このプロファイルを使用する Pod を起動するサブネットの ID。現時点では、Fargate で実行されている Pod にはパブリック IP アドレスが割り当てられていません。したがって、このパラメータには、プライベートサブネット (インターネットゲートウェイへの直接ルートなし) のみが受け入れられます。



## セレクター

この Fargate プロファイルを使用するために Pod に一致するセレクター。1 つの Fargate プロファイルに最大 5 つのセレクターを指定できます。セレクターには次のコンポーネントがあります:

- 名前空間 – セレクターの名前空間を指定する必要があります。セレクターはこの名前空間で作成された Pod のみを照合します。ただし、複数のセレクターを作成して複数の名前空間をターゲットにすることはできます。
- ラベル – オプションで、セレクターに一致する Kubernetes ラベルを指定できます。セレクターは、セレクターで指定されているすべてのラベルを持つ Pod のみに一致します。

## Fargate プロファイルのワイルドカード

名前空間、ラベルキー、およびラベル値のセレクター基準では、Kubernetes で許可されている文字に加えて、\* および ? の使用が許可されています。

- \* は、文字なし、1 文字、または複数の文字を表します。例: `prod*` は `prod` および `prod-metrics` を表します。
- ? は 1 文字を表します (例: `value?` は `valuea` を表します)。しかし、`value` および `value-a` を表すことはできません。? は厳密に 1 文字だけを表すためです。

これらのワイルドカード文字は、任意の位置や組み合わせで使用できます (例: `prod*`、`*dev`、および `frontend*?`)。正規表現など、他のワイルドカードやパターンマッチング形式はサポートされていません。

Pod 仕様内の名前空間とラベルに一致するプロファイルが複数ある場合、Fargate はプロファイル名による英数字ソートに基づいてプロファイルを選択します。例えば、(beta-workload という名前の) プロファイル A と (prod-workload という名前の) プロファイル B の両方に、起動する Pod に一致するセレクターがある場合、Fargate は Pod にプロファイル A (beta-workload) を選択します。Pod には、プロファイル A のラベルが付いています (例: `eks.amazonaws.com/fargate-profile=beta-workload`)。

ワイルドカードを使用する新しいプロファイルに既存の Fargate Pod を移行するには、次の 2 つの方法があります。

- 一致するセレクトターを持つ新しいプロファイルを作成し、古いプロファイルを削除します。古いプロファイルでラベル付けされたポッドは、一致する新しいプロファイルに再スケジュールされます。
- ワークロードを移行したいものの、各 Fargate Pod にどの Fargate ラベルが付いているかわからない場合は、次の方法を使用できます。新しいプロファイルを作成し、同じクラスター上のプロファイルの中でアルファベット順で最初に並べられる名前を付けます。次に、新しいプロファイルに移行する必要がある Fargate Pod をリサイクルします。

## Fargate プロファイルの作成

このセクションでは、Fargate プロファイルを作成する方法について説明します。また、Fargate プロファイルに使用する Pod 実行ロールを作成しておく必要があります。詳細については、「[the section called “ポッド実行IAMロール”](#)」を参照してください。Fargate で実行されている Pod は、プライベートサブネットでのみサポートされています (AWS サービスへの [NAT ゲートウェイ](#) アクセスはできますが、インターネットゲートウェイへの直接ルーティングはできません)。これは、クラスターの VPC がプライベートサブネットを使用できるようにするためです。

プロファイルは以下を使用して作成できます。

- [the section called “eksctl”](#)
- [the section called “AWS Management Console”](#)

## eksctl

**eksctl** で、Fargate プロファイルを作成するには

以下の **eksctl** コマンドで Fargate プロファイルを作成し、すべてのサンプル値を自分の値に置き換えます。名前空間を指定する必要があります。ただし、`--labels` オプションは必須ではありません。

```
eksctl create fargateprofile \
 --cluster my-cluster \
 --name my-fargate-profile \
 --namespace my-kubernetes-namespace \
 --labels key=value
```

my-kubernetes-namespace および key=value ラベルには、特定のワイルドカードを使用できます。詳細については、「[the section called “Fargate プロファイルのワイルドカード”](#)」を参照してください。

## AWS Management Console

AWS Management Console で、Fargate プロファイルを作成するには

1. [アマゾン EKS コンソール](#)を開きます。
2. Fargate プロファイルを作成するクラスターを選択します。
3. [コンピューティング] タブを開きます。
4. [Fargate プロファイル] で、[Fargate プロファイルを追加] を選択します。
5. [Fargate プロファイルを設定] ページで、次の操作を行います。
  - a. [名前] に、Fargate プロファイルの一意の名前 (my-profile など) を入力します。
  - b. [Pod 実行ロール] で、Fargate プロファイルで使用する Pod 実行ロールを選択します。eks-fargate-pods.amazonaws.com サービスプリンシパルを持つ IAM ロールのみが表示されます。ここにロールが表示されない場合は、ロールを作成する必要があります。詳細については、「[the section called “ポッド実行IAMロール”](#)」を参照してください。
  - c. 選択した [サブネット] を必要に応じて変更します。

### Note

Fargate で実行される Pod では、プライベートサブネットのみがサポートされます。

- d. [Tags] (タグ) では、オプションで Fargate プロファイルにタグを付けることができます。これらのタグは、Pod など、プロファイルに関連付けられた他のリソースには伝達されません。
  - e. [Next] を選択します。
6. [Pod の選択を設定] ページで、次の操作を行います。
    - a. [名前空間] に、Pod と照合する名前空間を入力します。
      - kube-system または default など、特定の名前空間を使用して照合できます。
      - 特定のワイルドカード (例: prod-\*) を使用して、複数の名前空間 (例: prod-deployment および prod-test) と照合することができます。詳細については、「[the section called “Fargate プロファイルのワイルドカード”](#)」を参照してください。
    - b. (オプション) セレクタに Kubernetes ラベルを追加します。特に、指定された名前空間内の Pod が一致する必要があるものにそれらを追加します。

- ラベル `infrastructure: fargate` をセクターに追加して、`infrastructure: fargate` Kubernetes ラベルも持つ指定された名前空間の Pod のみがセクターと一致するようにすることができます。
- 特定のワイルドカード (例: `key?: value?`) を使用して、複数の名前空間 (例: `keya: valuea` および `keyb: valueb`) と照合することができます。詳細については、「[the section called “Fargate プロファイルのワイルドカード”](#)」を参照してください。

c. [Next] を選択します。

7. [確認と作成] ページで、Fargate プロファイルの情報を確認し、[作成] を選択します。

## Fargate プロファイルを削除する

このトピックでは、Fargate プロファイルを削除する方法について説明します。Fargate プロファイルを削除すると、そのプロファイルで Fargate にスケジューリングされていた Pod はすべて削除されます。これらの Pod が別の Fargate プロファイルと一致する場合、それらの Pod はそのプロファイルで Fargate にスケジューリングされます。Fargate プロファイルがどのプロファイルとも一致しなくなった場合は、Fargate へのスケジューリングは行われず、保留中のままになる可能性があります。

一度にクラスター内の 1 つの Fargate プロファイルのみが、DELETING ステータスになることができます。そのクラスター内の他のプロファイルを削除する前に、Fargate プロファイルの削除が完了するまでお待ちください。

プロファイルは、次のいずれかのツールを使用して削除できます。

- [the section called “eksctl”](#)
- [the section called “AWS Management Console”](#)
- [the section called “AWS CLI”](#)

### eksctl

**eksctl** で、Fargate プロファイルを削除する

クラスターからプロファイルを削除するには、次のコマンドを使用します。各 `#####` は独自の値に置き換えます。

```
eksctl delete fargateprofile --name my-profile --cluster my-cluster
```

## AWS Management Console

AWS Management Console で、Fargate プロファイルを削除する

1. [アマゾン EKS コンソール](#)を開きます。
2. 左のナビゲーションペインで [Clusters] (クラスター) を選択してください。クラスターの一覧で Fargate プロファイルを削除するクラスターを選択します。
3. [コンピューティング] タブを開きます。
4. 削除する Fargate プロファイルを選択し、[削除] を選択します。
5. [Fargate プロファイルの削除] ページで、プロファイルの名前を入力し、[削除] を選択します。

## AWS CLI

AWS CLI で、Fargate プロファイルを削除する

クラスターからプロファイルを削除するには、次のコマンドを使用します。各#####は独自の値に置き換えます。

```
aws eks delete-fargate-profile --fargate-profile-name my-profile --cluster-name my-cluster
```

## Fargate Pod 設定の詳細を理解する

このセクションでは、AWS Fargate で Kubernetes Pod を実行するための固有の Pod 設定の詳細について説明します。

### ポッド CPU とメモリ

Kubernetes では、Pod 内の各コンテナに割り当てられるリクエスト、最小量の vCPU およびメモリリソースを定義できます。Pod は Kubernetes によってスケジュールされ、少なくとも各 Pod に対してリクエストされたリソースがコンピューティングリソース上で利用可能になるようにします。詳細については、Kubernetes のドキュメントの「[Managing Compute Resources for Containers](#)」を参照してください。

#### Note

Amazon EKS Fargate はノードごとに Pod を 1 つだけ実行するため、リソースが少ない場合に Pod を削除するシナリオは発生しません。すべての Amazon EKS Fargate Pod は保証さ

れた優先度で実行されるため、リクエストされた CPU とメモリは、すべてのコンテナの制限に等しくする必要があります。詳細については、Kubernetes ドキュメントの「[Configure Quality of Service for Pods \(ポッド用にサービスの品質を設定する\)](#)」を参照してください。

Pod が Fargate にスケジュールされている場合、Pod 仕様内の vCPU とメモリの予約によって、Pod にプロビジョニングする CPU とメモリの量が決まります。

- Init コンテナからの最大リクエストは、Init リクエスト vCPU およびメモリ要件を決定するために使用されます。
- 実行時間の長いコンテナのリクエストは、実行時間の長いリクエスト vCPU とメモリ要件を決定するために合計されます。
- 前記の 2 つの値のうち大きい方が、Pod 用に使用する vCPU とメモリリクエストに対して選択されます。
- Fargate は、必要な Kubernetes コンポーネント (kubelet、kube-proxy、および containerd) の各 Pod のメモリ予約に 256 MB を追加します。

Fargate では、Pod の実行に必要なリソースを常に確保するために、vCPU とメモリのリクエストの合計に最も近い、次に示すコンピューティング設定に切り上げられます。

vCPU とメモリの組み合わせを指定しない場合は、使用可能な最小の組み合わせ (.25 vCPU と 0.5 GB のメモリ) が使用されます。

次の表は、Fargate で実行されている Pod で使用可能な vCPU とメモリの組み合わせを示しています。

| vCPU 値   | メモリの値                              |
|----------|------------------------------------|
| .25 vCPU | 0.5 GB、1 GB、2 GB                   |
| .5 vCPU  | 1 GB、2 GB、3 GB、4 GB                |
| 1 vCPU   | 2 GB、3 GB、4 GB、5 GB、6 GB、7 GB、8 GB |
| 2 vCPU   | 4 GB ~ 16 GB (1 GB のインクリメント)       |
| 4 vCPU   | 8 GB ~ 30 GB (1 GB のインクリメント)       |

| vCPU 値  | メモリの値                          |
|---------|--------------------------------|
| 8 vCPU  | 16 GB ~ 60 GB (4 GB のインクリメント)  |
| 16 vCPU | 32 GB ~ 120 GB (8 GB のインクリメント) |

Kubernetes コンポーネント用に予約されている追加メモリにより、要求よりも多くの vCPU による Fargate タスクがプロビジョニングされる可能性があります。例えば、1 つの vCPU と 8 GB のメモリを要求すると、メモリ要求に 256 MB が追加され、2 つの vCPU と 9 GB のメモリの Fargate タスクがプロビジョニングされます。これは、1 つの vCPU と 9 GB のメモリのタスクがないためです。

Fargate で実行されている Pod のサイズと、Kubernetes が `kubectl get nodes` でレポートするノードサイズに相関はありません。レポートされるノードサイズは、多くの場合 Pod のキャパシティーよりも大きくなります。次のコマンドを使用して、Pod キャパシティーを確認できます。`default` を Pod の名前空間に、`pod-name` を Pod の名前に置き換えます。

```
kubectl describe pod --namespace default pod-name
```

出力例は次のとおりです。

```
[...]
annotations:
 CapacityProvisioned: 0.25vCPU 0.5GB
[...]
```

CapacityProvisioned アノテーションは、実際の Pod キャパシティーを表し、これで Fargate で実行されている Pod のコストが決まります。コンピューティング設定の料金情報については、「[AWS Fargate の料金](#)」を参照してください。

## Fargate ストレージ

Fargate 上で実行される Pod は、ドライバーの手動インストールステップなしで、Amazon EFS ファイルシステムを自動的にマウントします。Fargate ノードでは動的永続ボリュームプロビジョニングを使用できませんが、静的プロビジョニングを使用することはできます。詳細については、GitHub の「[Amazon EFS CSI ドライバー](#)」を参照してください。

プロビジョニングすると、Fargate で実行されている各 Pod に対して、デフォルトで 20 GiB のエフェメラルストレージが割り当てられます。このタイプのストレージは、Pod の停止後に削除されます。Fargate に起動される新しい Pod では、エフェメラルストレージボリュームの暗号化がデ



フォルトで有効になっています。エフェメラル Pod ストレージは、AWS Fargate で管理されるキーを使用して AES-256 暗号化アルゴリズムで暗号化されます。

#### Note

Fargate で実行される Amazon EKS Pod のデフォルトで使用可能なストレージは、20 GiB 未満です。これは、一部のスペースが kubelet と Pod 内に読み込まれるその他の Kubernetes モジュールによって使用されるためです。

エフェメラルストレージの総量は、最大 175 GiB まで増やすことができます。Kubernetes でサイズを設定するには、Pod 内の各コンテナに対して、ephemeral-storage リソースのリクエストを指定します。Kubernetes が Pod をスケジュールすると、各 Pod に対するリソースリクエストの合計が Fargate タスクの容量を下回ることが保証されます。詳細については、Kubernetes ドキュメントの「[Resource Management for Pods and Containers](#)」を参照してください。

Amazon EKS Fargate は、システム使用目的でリクエストされた数よりも大きなエフェメラルストレージをプロビジョニングします。たとえば、100 GiB のリクエストでは、115 GiB のエフェメラルストレージの Fargate タスクがプロビジョニングされます。

## AWS Fargate OS パッチ適用イベントのアクションを設定する

Amazon EKS は定期的に AWS Fargate ノードの OS にパッチを適用し、ノードを安全に保ちます。パッチ適用プロセスの一環として、ノードをリサイクルして OS パッチをインストールします。更新は、サービスへの影響を最小限に抑える方法で試行されます。ただし、Pod のエビクションが正常に行われない場合は、Pod を削除する必要がある場合があります。以下に示しているのは、中断の可能性を最小限に抑えるために実行できるアクションです。

- 適切な Pod の中断予算 (PDB) を設定して、同時にダウンする Pod の数を制御します。
- Pod が削除される前に、失敗したエビクションに対応する Amazon EventBridge ルールを作成します。
- 受信した通知に記載されているエビクション日より前に、影響を受けるポッドを手動で再起動します。
- AWS User Notifications で通知設定を作成します。

Amazon EKS は、Kubernetes コミュニティと緊密に協力して、できるだけ早く、バグ修正とセキュリティパッチが入手可能になるようにしています。すべての Fargate Pod は最新の Kubernetes パツ



チバージョンで起動されます。このバージョンは、Amazon EKS からクラスターの Kubernetes バージョンとして入手できます。以前のパッチバージョンの Pod をお持ちの場合、Amazon EKS はそれをリサイクルして最新バージョンに更新することがあります。これにより、Pod に最新のセキュリティアップデートが確実に装備されます。そのため、クリティカルな[共通脆弱性識別子](#) (CVE) 問題があっても、最新の状態に保たれてセキュリティリスクが軽減されています。

AWS Fargate OS が更新されると、Amazon EKS は影響を受けるリソースと今後のポッドの工ビクション日を含む通知を送信します。指定された工ビクション日が不都合な場合は、通知に記載された工ビクション日より前に、影響を受けるポッドを手動で再起動するオプションを利用できます。通知を受け取る前に作成されたポッドは、工ビクションの対象となります。ポッドを手動で再起動する方法の詳細については、[Kubernetes ドキュメント](#)を参照してください。

Pod がリサイクルされるときに一度にダウンする Pod の数を制限するために、Pod の中断予算 (PDB) を設定できます。PDB を使用して、更新の実行を許可しながら、各アプリケーションの要件に基づいて最小限使用可能なポッドを定義できます。PDB の最小可用性は 100% 未満である必要があります。詳細については、Kubernetes ドキュメントの「[アプリケーションに Disruption Budget 指定する](#)」を参照してください。

Amazon EKS は [Eviction API](#) を使用して、アプリケーションに設定した PDB を尊重しながら Pod を安全に排出します。ポッドの工ビクションは、影響を最小限に抑えるために、アベイラビリティゾーンごとに行われます。工ビクションが成功すると、新しい Pod には最新のパッチが適用され、それ以上のアクションは必要ありません。

Pod の工ビクションが失敗すると、Amazon EKS は工ビクションに失敗した Pod の詳細を含むイベントをアカウントに送信します。スケジュールされた終了時刻より前にメッセージに対応できます。具体的な時刻は、パッチの緊急性によって異なります。その時刻になると、Amazon EKS は Pod の工ビクションを再び試行します。ただし、今回は、工ビクションが失敗しても新しいイベントは送信されません。工ビクションが再び失敗すると、新しい Pod に最新のパッチが適用できるように、既存の Pod が定期的に削除されます。

Pod の工ビクションが失敗したときに受信されるイベントのサンプルを次に示します。これには、クラスター、Pod 名、Pod 名前空間、Fargate プロファイル、およびスケジュールされた終了時刻に関する詳細が含まれます。

```
{
 "version": "0",
 "id": "12345678-90ab-cdef-0123-4567890abcde",
 "detail-type": "EKS Fargate Pod Scheduled Termination",
 "source": "aws.eks",
```

```
"account": "111122223333",
"time": "2021-06-27T12:52:44Z",
"region": "region-code",
"resources": [
 "default/my-database-deployment"
],
"detail": {
 "clusterName": "my-cluster",
 "fargateProfileName": "my-fargate-profile",
 "podName": "my-pod-name",
 "podNamespace": "default",
 "evictErrorMessage": "Cannot evict pod as it would violate the pod's disruption budget",
 "scheduledTerminationTime": "2021-06-30T12:52:44.832Z[UTC]"
}
```

さらに、Pod に複数の PDB が関連付けられていると、エビクション失敗イベントが発生する可能性があります。このイベントは次のエラーメッセージを返します。

```
"evictErrorMessage": "This pod has multiple PodDisruptionBudget, which the eviction subresource does not support",
```

このイベントに基づいて、目的のアクションを作成できます。例えば、Pod の中断予算 (PDB) を調整して、Pod のエビクションの方法を制御できます。具体的には、利用可能な Pod の目標パーセンテージを指定する PDB から始めたとします。アップグレード中に Pod が強制終了される前に、PDB を異なる割合の Pod に調整できます。このイベントを受信するには、クラスターが属する AWS アカウントおよび AWS リージョンで Amazon EventBridge ルールを作成する必要があります。ルールでは、次のカスタムパターンを使用する必要があります。EventBridge ルールの作成の詳細については、Amazon EventBridge ユーザーガイドの「[イベントに反応する Amazon EventBridge ルールの作成](#)」を参照してください。

```
{
 "source": ["aws.eks"],
 "detail-type": ["EKS Fargate Pod Scheduled Termination"]
}
```

キャプチャするためのイベントに適切なターゲットを設定できます。詳細については、「Amazon EventBridge ユーザーガイド」の「[Amazon EventBridge ターゲット](#)」を参照してください。AWS User Notifications で通知設定を作成することもできます。AWS Management Console を

使用して通知を作成する場合は、[イベントルール] で、[AWS 名] に [Elastic Kubernetes Service (EKS)] を選択し、[イベントタイプ] に [EKS Fargate Pod のスケジュールされた終了時刻] を選択します。詳細については、「AWS User Notifications ユーザーガイド」の「[AWS User Notifications の開始方法](#)」を参照してください。

EKS Pod エビクションに関するよくある質問については、AWS re:Post の「[よくある質問: Fargate Pod のエビクション通知](#)」を参照してください。

## AWS Fargate アプリケーションと使用状況のメトリクスを収集する

AWS Fargate のシステムメトリクスおよび CloudWatch 使用状況メトリクスを収集できます。

### アプリケーションメトリクス

Amazon EKS や AWS Fargate で動作するアプリケーションには、AWS Distro for OpenTelemetry (ADOT) を使用できます。ADOT では、システムメトリクスを収集し、CloudWatch コンテナインサイトダッシュボードに送信できます。Fargate で実行されているアプリケーションで ADOT の使用を開始するには、「ADOT ドキュメント」の「[Using CloudWatch Container Insights with AWS Distro for OpenTelemetry](#)」( Distro for OpenTelemetry を活用した CloudWatch Container Insights の使用) を参照してください。

### 使用状況メトリクス

CloudWatch 使用状況メトリクスを使用して、アカウントのリソースの使用状況を把握できます。これらのメトリクスを使用して、CloudWatch グラフやダッシュボードで現在のサービスの使用状況を可視化できます。

AWS Fargate 使用状況メトリクスは、AWS Service Quotas に対応しています。使用量がサービスクォータに近づいたときに警告するアラームを設定することもできます。Fargate のサービスのクォータの詳細については、「[the section called “Service Quotas”](#)」を参照してください。

AWS Fargate は、AWS/Usage 名前空間に以下のメトリクスを公開します。

| メトリクス         | 説明                                                               |
|---------------|------------------------------------------------------------------|
| ResourceCount | アカウントで実行されている指定されたリソースの合計数。リソースは、メトリクスに関連付けられたディメンションによって定義されます。 |

次のディメンションは、AWS Fargate によって公開される使用状況メトリクスを絞り込むために使用されます。

| ディメンション  | 説明                                                                                                                                                                                                                                                                                                    |
|----------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Service  | リソースを含む AWS のサービスの名前。AWS Fargate 使用状況メトリクスの場合、このディメンションの値は Fargate です。                                                                                                                                                                                                                                |
| Type     | 報告されるエンティティのタイプ。現在、AWS Fargate 使用状況メトリクスの有効な値は Resource のみです。                                                                                                                                                                                                                                         |
| Resource | <p>実行中のリソースのタイプ。</p> <p>現在、AWS Fargate は Fargate のオンデマンド 使用状況に関する情報を返します。Fargate のオンデマンド使用状況のリソース値は OnDemand です。</p> <p>[注意] =====</p> <p>Fargate のオンデマンド使用状況とは、Fargate を使用した Amazon EKS Pod、Fargate 起動タイプを使用した Amazon ECS タスク、FARGATE キャパシティープロバイダーを使用した Amazon ECS タスクを組み合わせたものです。</p> <p>=====</p> |
| Class    | 追跡されているリソースのクラス。現在、AWS Fargate はクラスディメンションを使用していません。                                                                                                                                                                                                                                                  |

Fargate のリソース使用状況メトリクスをモニタリングするための CloudWatch アラームの作成

AWS Fargate は、Fargate オンデマンドリソース使用状況の AWS Service Quotas に対応する CloudWatch 使用状況メトリクスを提供します。Service Quotas コンソールでは、使用状況をグラフ

で可視化できます。また、使用量が Service Quotas に近づいたときに警告するアラームも設定できます。詳細については、「[the section called “メトリクスを収集する”](#)」を参照してください。

以下の手順を使用して、Fargate リソース使用状況メトリクスに基づく CloudWatch アラームを作成します。

1. Service Quotas のコンソールを開きます。<https://console.aws.amazon.com/servicequotas/>
2. 左のナビゲーションペインで [AWS サービス] を選択します。
3. [AWS サービス] リストから、[AWS Fargate] を探して選択します。
4. [Service quotas] (サービスクォータ) リストで、アラームを作成する Fargate 使用量クォータを選択します。
5. Amazon CloudWatch アラームのセクションで [Create] (作成) を選択します。
6. [アラームのしきい値] で、適用されたクォータ値からアラーム値として設定する値の割合を選択します。
7. [アラーム名] にアラームの名前を入力し、[Create (作成)] を選択します。

## クラスターの AWS Fargate ログ記録を開始する

Fargate の Amazon EKS では、Fluent Bit をベースにした組み込みのログルーターが利用できます。Fluent Bit コンテナをサイドカーとして明示的に実行する必要はなく、この実行は Amazon によって行われます。必要となるのは、ログルーターの設定だけです。設定は専用の ConfigMap を介して行い、その際は以下の基準を満たす必要があります。

- 名前のついた aws-logging
- aws-observability と呼ばれる専用の名前空間での作成
- 5,300 文字以内にしてください。

ConfigMap を作成すると、Fargate の Amazon EKS は自動的にそれを検出しログルーターの設定を行います。Fargate は、Fluent Bit の AWS のバージョンを使用しています。これは、AWS によって管理される Fluent Bit の上流対応のディストリビューションです。詳細については、GitHub の「[AWS for Fluent Bit](#)」を参照してください。

ログルーターを使用すると、AWS のさまざまなサービスをログの分析と保管に使用できます。Fargate からは、Amazon CloudWatch、Amazon OpenSearch Service に対し直接ログをストリーミングできます。また、[Amazon Data Firehose](#) 経由で [Amazon S3](#)、[Amazon Kinesis Data Streams](#)、およびパートナーツールなどの送信先にログをストリーミングすることも可能です。

- Fargate Pod をデプロイする既存の Kubernetes 名前空間を指定する既存の Fargate プロファイル。詳細については、「[the section called “ステップ 3: クラスターの Fargate プロファイルを作成する”](#)」を参照してください。
- 既存の Fargate Pod 実行ロール。詳細については、「[the section called “ステップ 2: Fargate Pod 実行ロールを作成する”](#)」を参照してください。

## ログルーターの設定

### ⚠ Important

ログを正常に公開するには、そのクラスターが属している VPC からログの送信先までの、ネットワークアクセスが必要になります。これには、ユーザーによる VPC の出力ルール、カスタマイズが関係します。詳細については、「Amazon CloudWatch Logs ユーザーガイド」の「[Using CloudWatch Logs with interface VPC endpoints](#)」を参照してください。

以下のステップでは、すべての#####を独自の値に置き換えます。

1. aws-observability という名前で、専用の Kubernetes 名前空間を作成します。

- a. 次の内容をコンピュータ上の aws-observability-namespace.yaml という名前のファイルに保存します。name の値は aws-observability である必要があり、aws-observability: enabled ラベルが必須です。

```
kind: Namespace
apiVersion: v1
metadata:
 name: aws-observability
 labels:
 aws-observability: enabled
```

- b. 名前空間を作成します。

```
kubectl apply -f aws-observability-namespace.yaml
```

2. データ値 Fluent Conf を使用して ConfigMap を作成し、コンテナログを送信先に送ります。Fluent Conf とは Fluent Bit であり、コンテナログを任意のログ送信先にルーティングするために使用される、高速で軽量なログプロセッサ構成言語です。詳細については、Fluent Bit ドキュメントの「[Configuration File](#)」を参照してください。

 Important

典型的な Fluent Conf に含まれている主なセクションは、ServiceInput、Filter および Output です。ただし、Fargate のログルーターでは、以下だけを受け入れます。

- Filter および Output セクション。
  - Parser セクション。
- 他のセクションを提供した場合、拒否されます。

Fargate ログルーターは Service および Input セクションを管理します。次の Input セクションがありますが、これらは変更できず、ConfigMap には必要ありません。ただし、メモリバッファ制限やログに適用されるタグなどの洞察は得られます。

```
[INPUT]
 Name tail
 Buffer_Max_Size 66KB
 DB /var/log/flb_kube.db
 Mem_Buf_Limit 45MB
 Path /var/log/containers/*.log
 Read_From_Head On
 Refresh_Interval 10
 Rotate_Wait 30
 Skip_Long_Lines On
 Tag kube.*
```

ConfigMap の作成時は、以下の (Fargate がフィールドの検証に使用する) ルールを考慮に入れます。

- [FILTER]、[OUTPUT]および [PARSER] は、それぞれが対応するキーにより指定する必要があります。例: [FILTER] が filters.conf の下にある必要があります。filters.conf には、複数の [FILTER] を含められます。また、[OUTPUT] および [PARSER] セクションは、それぞれと対応するキーの下に置く必要があります。複数の [OUTPUT] セクションを指定することで、ログを異なる送信先に同時にルーティングできます。
- Fargate は各セクションに必要なキーを検証します。Name および match がそれぞれの [FILTER] および [OUTPUT] に必要です。Name および format がそれぞれの [PARSER] に必要です。キーの大文字と小文字は区別されません。
- \${ENV\_VAR} などの環境変数は ConfigMap では許可されていません。



- インデントは、それぞれの `filters.conf`、`output.conf`、および `parsers.conf` の中で、ディレクティブまたはキーと値のペアで同じである必要があります。キーと値のペアは、ディレクティブよりも深いインデントにする必要があります。
- Fargate は、サポートされている次のフィルターに対して検証します。grep、parser、record\_modifier、rewrite\_tag、throttle、nest、modify、および kubernetes。
- Fargate は、サポートされている次の出力に対して検証します。es、firehose、kinesis\_firehose、cloudwatch、cloudwatch\_logs、および kinesis。
- ログ記録を有効にするには、サポートされている Output プラグインが少なくとも 1 つ ConfigMap にあることが必要です。Filter および Parser は、ログ記録を有効にするために必要ありません。

また、希望の設定を使用して Amazon EC2 で Fluent Bit を実行し、検証によって発生する問題をトラブルシューティングすることもできます。以下のいずれかの例に従って、ConfigMap を作成します。

 Important

Amazon EKS Fargate のログ記録では、ConfigMap での動的設定をサポートしていません。ConfigMap に対する任意の変更は、新しい Pod に対してのみ適用されます。既存の Pod には、これらの変更は適用されません。

例を使用して、必要なログ送信先用に ConfigMap を作成します。

 Note

また、Amazon Kinesis Data Streams をログの宛先として使用できます。Kinesis Data Streams を使用する場合は、ポッド実行ロールに `kinesis:PutRecords` 権限が付与されていることを確認してください。詳細については、「[Fluent Bit: 公式マニュアル](#)」の Amazon Kinesis Data Streams の「[許可](#)」を参照してください。



## Example

### CloudWatch

CloudWatch を使用する場合、次の 2 つの出力オプションがあります。

- [C で記述された出力プラグイン](#)
- [Golang で記述された出力プラグイン](#)

次の例は、cloudwatch\_logs プラグインを使用して CloudWatch にログを送信する方法を示しています。

- a. 次の内容を aws-logging-cloudwatch-configmap.yaml という名前のファイルに保存します。##### を、クラスターのある AWS リージョンに置き換えます。[OUTPUT] のパラメータは必須です。

```
kind: ConfigMap
apiVersion: v1
metadata:
 name: aws-logging
 namespace: aws-observability
data:
 flb_log_cw: "false" # Set to true to ship Fluent Bit process logs to
CloudWatch.
 filters.conf: |
 [FILTER]
 Name parser
 Match *
 Key_name log
 Parser criot
 [FILTER]
 Name kubernetes
 Match kube.*
 Merge_Log On
 Keep_Log Off
 Buffer_Size 0
 Kube_Meta_Cache_TTL 300s
 output.conf: |
 [OUTPUT]
 Name cloudwatch_logs
 Match kube.*
 region region-code
 log_group_name my-logs
```

```

 log_stream_prefix from-fluent-bit-
 log_retention_days 60
 auto_create_group true
 parsers.conf: |
 [PARSER]
 Name crio
 Format Regex
 Regex ^(?<time>[^\]+) (?<stream>stdout|stderr) (?<logtag>P|F) (?
<log>.*)$
 Time_Key time
 Time_Format %Y-%m-%dT%H:%M:%S.%L%z

```

- b. マニフェストをクラスターに適用します。

```
kubectl apply -f aws-logging-cloudwatch-configmap.yaml
```

## Amazon OpenSearch Service

ログを Amazon OpenSearch Service に送信したい場合、C で書かれたプラグインである、[es](#) 出力を使用することができます。以下の例では、OpenSearch にログを送信するためにプラグインを使用する方法について説明します。

- a. 次の内容を `aws-logging-opensearch-configmap.yaml` という名前のファイルに保存します。各 `#####` は独自の値に置き換えます。

```

kind: ConfigMap
apiVersion: v1
metadata:
 name: aws-logging
 namespace: aws-observability
data:
 output.conf: |
 [OUTPUT]
 Name es
 Match *
 Host search-example-gjxdcilagiprbqlqn42jsty66y.region-
code.es.amazonaws.com
 Port 443
 Index example
 Type example_type
 AWS_Auth On
 AWS_Region region-code
 tls On

```

- b. マニフェストをクラスターに適用します。

```
kubectl apply -f aws-logging-opensearch-configmap.yaml
```

## Firehose

Firehose にログを送信する場合、次の二つの出力オプションがあります。

- [kinesis\\_firehose](#) – C 言語で記述された出力プラグイン。
- [firehose](#) — Go 言語で記述された出力プラグイン。

次の例は、Firehose にログを送信するために `kinesis_firehose` プラグインを使用する方法を示しています。

- a. 次の内容を `aws-logging-firehose-configmap.yaml` という名前のファイルに保存します。*region-code* を、クラスターのある AWS リージョンに置き換えます。

```
kind: ConfigMap
apiVersion: v1
metadata:
 name: aws-logging
 namespace: aws-observability
data:
 output.conf: |
 [OUTPUT]
 Name kinesis_firehose
 Match *
 region region-code
 delivery_stream my-stream-firehose
```

- b. マニフェストをクラスターに適用します。

```
kubectl apply -f aws-logging-firehose-configmap.yaml
```

3. 目的の宛先にログを送信できるように Fargate Pod 実行ロールに対するアクセス許可を設定します。

- a. 送信先の IAM ポリシーをコンピュータにダウンロードします。

## Example

### CloudWatch

CloudWatch IAM ポリシーをコンピュータにダウンロードします。GitHub で[ポリシーの表示](#)をすることもできます。

```
curl -O https://raw.githubusercontent.com/aws-samples/amazon-eks-fluent-logging-examples/mainline/examples/fargate/cloudwatchlogs/permissions.json
```

### Amazon OpenSearch Service

OpenSearch IAM ポリシーをコンピュータにダウンロードします。GitHub で[ポリシーの表示](#)をすることもできます。

```
curl -O https://raw.githubusercontent.com/aws-samples/amazon-eks-fluent-logging-examples/mainline/examples/fargate/amazon-elasticsearch/permissions.json
```

OpenSearch Dashboards のアクセスコントロールが適切に設定されていることを確認します。OpenSearch Dashboards の `all_access` role には、Fargate Pod 実行ロールと IAM ロールがマッピングされている必要があります。同様のマッピングが、`security_manager` ロールに対しても必要です。以前のマッピングを追加するには、Menu、Security、Roles の順にクリックした後、それぞれに対応するロールを選択します。詳細については、「[CloudWatch Logs が Amazon ES ドメインにストリーミングされるようにトラブルシューティングする方法を教えてください。](#)」を参照してください。

### Firehose

Firehose IAM ポリシーをコンピュータにダウンロードします。GitHub で[ポリシーの表示](#)をすることもできます。

```
curl -O https://raw.githubusercontent.com/aws-samples/amazon-eks-fluent-logging-examples/mainline/examples/fargate/kinesis-firehose/permissions.json
```

- b. ダウンロードしたポリシーファイルから IAM ポリシーを作成します。

```
aws iam create-policy --policy-name eks-fargate-logging-policy --policy-document file://permissions.json
```

- c. 次のコマンドを使用して、Fargate プロファイルに指定されたポッド実行ロールに IAM ポリシーを添付します。**111122223333** は、ご自分のアカウント ID に置き換えます。**AmazonEKSFargatePodExecutionRole** を Pod 実行ロールに置き換えます (詳細については、「[the section called “ステップ 2: Fargate Pod 実行ロールを作成する”](#)」を参照してください)。

```
aws iam attach-role-policy \
 --policy-arn arn:aws:iam::111122223333:policy/eks-fargate-logging-policy \
 --role-name AmazonEKSFargatePodExecutionRole
```

## Kubernetes フィルターのサポート

Fluent Bit Kubernetes フィルターを使用すると、Kubernetes メタデータをログファイルに追加できます。フィルターの詳細については、Fluent Bit ドキュメントの [Kubernetes](#) を参照してください。API サーバーエンドポイントを使用して、フィルターを適用できます。

```
filters.conf: |
 [FILTER]
 Name kubernetes
 Match kube.*
 Merge_Log On
 Buffer_Size 0
 Kube_Meta_Cache_TTL 300s
```

### Important

- Kube\_URL、Kube\_CA\_File、Kube\_Token\_Command、および Kube\_Token\_File はサービス所有の設定パラメータであるため、指定しないでください。Amazon EKS Fargate がこれらの値を設定します。
- Kube\_Meta\_Cache\_TTL は、Fluent Bit が最新のメタデータを取得するために API サーバーと通信するまで待機する時間です。Kube\_Meta\_Cache\_TTL が指定されていない場合、Amazon EKS Fargate は API サーバーの負荷を軽減するためにデフォルト値である 30 分を追加します。

## Fluent Bit プロセスログをアカウントに送付するには

必要に応じて、Fluent Bit プロセスログは、以下の ConfigMap を使用して Amazon CloudWatch に送付できます。Fluent Bit プロセスログを CloudWatch に送信するには、追加のログの取り込みおよびストレージのコストがかかります。*region-code* を、クラスターのある AWS リージョンに置き換えます。

```
kind: ConfigMap
apiVersion: v1
metadata:
 name: aws-logging
 namespace: aws-observability
 labels:
data:
 # Configuration files: server, input, filters and output
 # =====
 flb_log_cw: "true" # Ships Fluent Bit process logs to CloudWatch.

output.conf: |
 [OUTPUT]
 Name cloudwatch
 Match kube.*
 region region-code
 log_group_name fluent-bit-cloudwatch
 log_stream_prefix from-fluent-bit-
 auto_create_group true
```

ログは、CloudWatch 内のクラスターと同じ AWS リージョンにあります。ロググループ名は *my-cluster*-fluent-bit-logs で、Fluent Bit のログストリーム名は fluent-bit-*podname*-*pod-namespace* です。

### Note

- プロセスログは、Fluent Bit プロセスが正常に開始された場合にのみ送付されます。Fluent Bit の起動中に障害が発生すると、プロセスログが失われます。プロセスログは CloudWatch にのみ送付できます。
- 送付プロセスログをアカウントにデバッグするには、前述の ConfigMap を適用してプロセスログを取得します。Fluent Bit が起動に失敗するのは、通常、起動中に ConfigMap が Fluent Bit によって解析または受け入れられていないためです。

## Fluent Bit プロセスログの送信を停止するには

Fluent Bit プロセスログを CloudWatch に送信するには、追加のログの取り込みおよびストレージのコストがかかります。既存の ConfigMap 設定でプロセスログを除外するには、次のステップを実行します。

1. Fargate ログ記録を有効にした後、Amazon EKS クラスターの Fluent Bit プロセスログ用に自動的に作成された CloudWatch ロググループを見つけます。これはフォーマット *my-cluster-fluent-bit-logs* に従います。
2. CloudWatch ロググループ内の各 Pod のプロセスログ用に作成された既存の CloudWatch ログストリームを削除します。
3. ConfigMap を編集して `flb_log_cw: "false"` を設定します。
4. クラスター内の既存の Pod を再起動します。

## アプリケーションをテストする

1. サンプル Pod をデプロイします。
  - a. 次の内容をコンピュータ上の `sample-app.yaml` という名前のファイルに保存します。

```
apiVersion: apps/v1
kind: Deployment
metadata:
 name: sample-app
 namespace: same-namespace-as-your-fargate-profile
spec:
 replicas: 3
 selector:
 matchLabels:
 app: nginx
 template:
 metadata:
 labels:
 app: nginx
 spec:
 containers:
 - name: nginx
 image: nginx:latest
 ports:
 - name: http
```

```
containerPort: 80
```

b. マニフェストをクラスターに適用します。

```
kubectl apply -f sample-app.yaml
```

2. ConfigMap で設定した送信先を使用して、NGINX ログを表示します。

## サイズに関する考慮事項

ログルーター用のメモリは、最大 50 MB に収まるようにすることをお勧めします。アプリケーションで非常に高いスループットでログが生成されることが予想される場合は、最大 100 MB を想定して計画する必要があります。

## トラブルシューティング

ConfigMap が無効になっているなど、何らかの理由でログ機能が有効または無効になっているかどうか、および無効になっている理由を確認するには、Pod イベントを `kubectl describe pod pod-name` でチェックしてください。出力には、次の出力例のように、ロギングが有効かどうかを明確にする Pod イベントが含まれる場合があります。

```
[...]
Annotations: CapacityProvisioned: 0.25vCPU 0.5GB
 Logging: LoggingDisabled: LOGGING_CONFIGMAP_NOT_FOUND
[...]
Events:
 Type Reason Age From
 ---- -
 Warning LoggingDisabled <unknown> fargate-scheduler
 Disabled logging because aws-logging configmap was not found. configmap
 "aws-logging" not found
```

Pod イベントは一時的なもので、その期間は設定によります。`kubectl describe pod pod-name` を使用して Pod のアノテーションを表示することもできます。Pod アノテーションには、ロギング機能が有効か無効かとその理由に関する情報があります。



## 最適な Amazon EC2 ノードインスタンスタイプを選択する

Amazon EC2 では、ワーカーノード用のインスタンスタイプが幅広く用意されています。インスタンスタイプごとに、コンピューティング、メモリ、ストレージの異なる機能が提供されます。また、各インスタンスファミリーは、これらの機能に基づきグループ化されています。リストについては、「Amazon EC2 ユーザーガイド」の「[利用可能なインスタンスタイプ](#)」を参照してください。Amazon EKS は、(特定の) サポートを有効にするために、Amazon EC2 AMI のいくつかのバリエーションをリリースしています。選択したインスタンスタイプに Amazon EKS との互換性があることを確認する際は、次の基準を考慮してください。

- mac ファミリーは、現在、すべての Amazon EKS AMI でサポートされていません。
- Arm およびアクセラレータ付きではない Amazon EKS AMI では、g3、g4、inf、および p ファミリーはサポートされません。
- アクセラレータ付き Amazon EKS AMI は、a、c、hpc、m、および t ファミリーをサポートしていません。
- Arm ベースのインスタンスの場合、Amazon Linux 2023 (AL2023) は Graviton2 以降のプロセッサを使用するインスタンスタイプのみをサポートします。AL2023 は A1 インスタンスをサポートしていません。

Amazon EKS でサポートされているインスタンスタイプを選択する場合は、各タイプで次の機能を考慮してください。

### ノードグループ内のインスタンス数

特に多くの DaemonSet が存在する場合などは、一般的に、数少ない大型のインスタンスの使用が適しています。各インスタンスには API サーバーへの API コールが必要です。したがって、インスタンス数が多いほど、API サーバーのロードが高くなります。

### オペレーティングシステム

[Linux](#)、[Windows](#)、および [Bottlerocket](#) に対応しているインスタンスタイプを確認します。Windows インスタンスを作成する前に、「[Deploy Windows nodes on EKS clusters](#)」を確認してください。

### ハードウェアアーキテクチャ

x86 または Arm のどちらが必要か検討します。Arm インスタンスをデプロイする前に、「[Amazon EKS optimized Arm Amazon Linux AMIs](#)」を確認してください。Nitro System ([Linux](#)

または [Windows](#)) 上に構築されたインスタンス、または [アクセラレータ付き](#) 機能が必要かを検討します。アクセラレータ付き機能は、Linux の Amazon EKS でのみ使用できます。

## Pod の最大数

各 Pod には独自の IP アドレスが割り当てられているため、インスタンスタイプでサポートされている IP アドレスの数が、インスタンスで実行できる Pod の数を決定するための要因になります。インスタンスタイプがサポートする Pod の数を手動で確認するには、「[the section called “各 Amazon EC2 インスタンスタイプの Amazon EKS の推奨最大ポッド数”](#)」を参照してください。

### Note

Amazon EKS に最適化された Amazon Linux 2 AMI の v20220406 以降を使用している場合、AMI を最新版にアップグレードしなくても、新しいインスタンスタイプを使用できます。これらの AMI は、max-pods の値が [eni-max-pods.txt](#) ファイルにリストされていない場合、必要な値を自動計算します。現在プレビュー中のインスタンスタイプは、Amazon EKS のデフォルトではサポートされていない場合があります。これらのタイプでの max-pods 値は、依然として AMI 内の eni-max-pods.txt に追加する必要があります

[AWS Nitro システム](#) のインスタンスタイプには、サポートできる IP アドレスの数を、非 Nitro のシステムのインスタンスタイプよりも大幅に多くできるオプションがあります。ただし、インスタンスに割り当てられたすべての IP アドレスが Pod で使用できるわけではありません。インスタンスに割り当てる IP アドレスの数を大幅に増やすには、クラスターにバージョン 1.9.0 以降の Amazon VPC CNI アドオンをインストールし、適切に設定する必要があります。詳細については、「[the section called “IP アドレスを増やす”](#)」を参照してください。インスタンスに最大数の IP アドレスを割り当てるには、バージョン 1.10.1 以降の Amazon VPC CNI アドオンをクラスターにインストールし、IPv6 ファミリーを使用してそのクラスターをデプロイする必要があります。

## IP ファミリー

クラスターで IPv4 ファミリーを使用している場合、サポートされているすべてのインスタンスタイプが使用可能になります。これによりクラスターは、プライベート IPv4 アドレスを Pod とサービスに割り当てることができます。ただし、クラスターに IPv6 ファミリーを使用する場合は、[AWS Nitro システム](#) インスタンスタイプ、またはベアメタルインスタンスタイプを使用する必要があります。Windows インスタンスでは、IPv4 のみがサポートされています。クラス

ターでは、バージョン 1.10.1 以降の Amazon VPC CNI アドオンを実行している必要があります。IPv6 の使用の詳細については、「[the section called “IPv6”](#)」を参照してください。

実行している Amazon VPC CNI アドオンのバージョン

最新バージョンの [Amazon VPC CNI Plugin for Kubernetes](#) は、[こちらのインスタンスタイプ](#)をサポートしています。サポートされている最新のインスタンスタイプを利用するには、Amazon VPC CNI アドオンのバージョンを更新する必要があります。詳細については、「[the section called “Amazon VPC CNI”](#)」を参照してください。最新バージョンでは、Amazon EKS で使用できる最新の機能をサポートしています。以前のバージョンでは、すべての機能がサポートされているわけではありません。さまざまなバージョンでサポートされている機能は、GitHub の [Changelog](#) で確認できます。

ノードを作成している AWS リージョン

AWS リージョンによっては使用できないインスタンスタイプがあります。

Pod にセキュリティグループを使用しているかどうか

Pod にセキュリティグループを使用している場合は、特定のインスタンスタイプのみがサポートされます。詳細については、「[the section called “ポッドのセキュリティグループ”](#)」を参照してください。

## 各 Amazon EC2 インスタンスタイプの Amazon EKS の推奨最大ポッド数

各 Pod には独自の IP アドレスが割り当てられているため、インスタンスタイプでサポートされている IP アドレスの数が、インスタンスで実行できる Pod の数を決定するための要因になります。Amazon EKS には、ダウンロードして実行できるスクリプトが用意されており、各インスタンスタイプで実行する Amazon EKS で推奨される最大のポッド数を決定できます。このスクリプトでは、各インスタンスのハードウェア属性と設定オプションを使用して、ポッドの最大数を決定します。これらの手順で返された番号を使用して、[インスタンスが属するサブネットとは異なるサブネットから IP アドレスをポッドに割り当てる機能](#)や、[インスタンスの IP アドレス数を大幅に増やす機能](#)を有効にできます。複数のインスタンスタイプを持つマネージドノードグループを使用している場合は、それらすべてのインスタンスタイプで機能する値を使用します。

1. 各インスタンスタイプにおけるポッドの最大数を計算するために使用できるスクリプトをダウンロードします。

```
curl -O https://raw.githubusercontent.com/awslabs/amazon-eks-ami/master/templates/al2/runtime/max-pods-calculator.sh
```

2. コンピュータ上で、そのスクリプトを実行可能としてマークします。

```
chmod +x max-pods-calculator.sh
```

3. **m5.large** をデプロイ予定のインスタンスタイプに置き換え、**1.9.0-eksbuild.1** を Amazon VPC CNI アドオンバージョンに置き換えて、そのスクリプトを実行します。アドオンバージョンを確認するには、「[Amazon VPC CNI を使用してポッドに IP を割り当てる](#)」の更新手順を参照してください。

```
./max-pods-calculator.sh --instance-type m5.large --cni-version 1.9.0-eksbuild.1
```

出力例は次のとおりです。

29

次のオプションをスクリプトに追加して、オプションの機能を使用する際にサポートされる最大のポッド数を確認できます。

- `--cni-custom-networking-enabled` — インスタンスとは異なるサブネットから IP アドレスを割り当てる場合は、このオプションを使用します。詳細については、「[the section called “カスタムネットワーキング”](#)」を参照してください。同じサンプル値を使用して前のスクリプトにこのオプションを追加すると、20 が得られます。
- `--cni-prefix-delegation-enabled` — 各 elastic network interface にかなり多くの IP アドレスを割り当てる場合は、このオプションを使用します。この機能を使用するには、Nitro System で実行する Amazon Linux インスタンスと、Amazon VPC CNI アドオンのバージョン 1.9.0 以降が必要です。詳細については、「[the section called “IP アドレスを増やす”](#)」を参照してください。同じサンプル値を使用して前のスクリプトにこのオプションを追加すると、110 が得られます。

`--help` オプションを指定してスクリプトを実行し、使用可能なすべてのオプションを表示することもできます。

#### Note

最大 Pod 計算スクリプトは、[Kubernetes スケーラビリティのしきい値](#)と推奨設定に基づいて戻り値を 110 に制限します。インスタンスタイプに 30 を超える vCPU がある場合、この制限は、内部の Amazon EKS スケーラビリティチームのテストに基づく数値である 250 に

跳ね上がります。詳細については、「[Amazon VPC CNI プラグインがノードあたりのポッド数の制限を引き上げ](#)」のブログ記事を参照してください。

## EKS Auto Mode に関する考慮事項

EKS Auto Mode では、ノード上のポッドの数が以下のいずれか低い方に制限されます。

- ハード上限の 110 ポッド
- 上記の最大ポッド数計算の結果。

## 事前構築済みの最適化されたイメージを使用してノードを作成する

マネージドノードグループまたはセルフマネージド型ノードを使用する場合、事前に構築された Amazon EKS 最適化 [Amazon マシンイメージ](#) (AMI)、または独自のカスタム AMI のいずれかを使用してノードをデプロイできます。ハイブリッドノードを実行している場合は、「[the section called “オペレーティングシステムを準備する”](#)」を参照してください。Amazon EKS 最適化 AMI の各タイプの詳細については、以下のいずれかのトピックを参照してください。独自のカスタム AMI を作成する手順については、「[the section called “カスタムビルド”](#)」を参照してください。

Amazon EKS Auto Mode では、EKS は AMI の選択と更新を含め EC2 インスタンスを管理します。

### トピック

- [EKS AL2 AMI と AL2 高速化 AMI の移行機能ガイド](#)
- [最適化された Amazon Linux AMI を使用してノードを作成する](#)
- [最適化された Bottlerocket AMI を使用してノードを作成する](#)
- [最適化された Ubuntu Linux AMI を使用してノードを作成する](#)
- [最適化された Windows AMI を使用してノードを作成する](#)

## EKS AL2 AMI と AL2 高速化 AMI の移行機能ガイド

AWS は、EKS AL2 最適化 AMI と AL2 高速化 AMI に対するサポートを、2025 年 11 月 26 日に終了します。お客様は、こちらのサポート終了 (EOS) の日付 (2025 年 11 月 26 日) 以降も EKS AL2 AMI をご使用いただけますが、EKS はこの日以降、新しい Kubernetes バージョンや AL2 AMI の更新 (マイナーリリース、パッチ、バグ修正を含む) をリリースすることはありません。当社はお客様に、Amazon Linux 2023 (AL2023) または Bottlerocket AMI へのアップグレードを推奨しています。

- AL2023 では、事前設定済みのセキュリティポリシー、SELinux の Permissive モード、IMDSv2 専用モードのデフォルトでの有効化、起動時間の最適化、セキュリティとパフォーマンスを向上させるパッケージ管理の改良によって、OS レベルの直接アクセスや広範なノード変更など大幅なカスタマイズを必要とするインフラストラクチャに最適な、セキュリティの機能や設定をデフォルトで組み込んだ「セキュアバイデフォルト」のアプローチを実現しています。
- Bottlerocket は、ノードのカスタマイズを最小限に抑えたコンテナネイティブアプローチに最適な、専用のコンテナ最適化設計により、セキュリティの強化や起動時間の短縮、アタックサーフェスの縮小を実現して、効率の向上を可能にします。

あるいは、EOS 日 (2025 年 11 月 26 日) まで [the section called “カスタムビルド”](#) か、Amazon Linux 2 の EOS 日 (2026 年 6 月 30 日) までは Amazon Linux 2 ベースインスタンスを使用してカスタムの AMI を構築するという方法もあります。詳細については、「[Amazon Linux 2023 に関するよくある質問](#)」、「[Bottlerocket のよくある質問](#)」、または「[the section called “AL2023 にアップグレードする”](#)」 「[the section called “Bottlerocket”](#)」のドキュメントにある詳しい移行ガイドを参照してください。

## 移行とサポートに関するよくある質問

AL2 から AL2023 AMI に移行するにはどうすればよいですか？

詳細なアプリケーションワークロードテストと文書化されたロールバック手順を含む移行計画を作成、実装し、EKS の公式ドキュメントの「[Amazon Linux 2 から Amazon Linux 2023 にアップグレードする](#)」の手順に従うことが推奨されます。

EKS 最適化 AL2 AMI の EKS サポート終了 (EOS) 日以降に、カスタム AL2 AMI を構築することは可能ですか？

当社は、AL2023 または Bottlerocket 用として正式にサポートされ公開されている EKS 最適化 AMI に移行されることを推奨しますが、EKS AL2 最適化 AMI と AL2 高速化 AMI は、AL2 AMI の EOS 日 (2025 年 11 月 26 日) まで構築することが可能です。また、Amazon Linux 2 の EOS 日 (2026 年 6 月 30 日) までは、Amazon Linux 2 ベースインスタンスを使用してカスタムの AMI を構築することができます。カスタム EKS AL2 最適化 AMI および AL2 高速化 AMI の構築手順については、EKS 公式ドキュメント内の「[カスタム Amazon Linux AMI を構築する](#)」を参照してください。



EKS Kubernetes のバージョンサポートポリシーは、Amazon Linux ディストリビューションに適用されますか？

適用されません。EKS AL2 最適化 AMI と AL2 高速化 AMI の EOS 日は、EKS による Kubernetes バージョンの標準サポートおよび延長サポートのスケジュールとは無関係です。EKS の延長サポートをご利用いただいている場合でも、AL2023 または Bottlerocket への移行が必要になります。

cgroupv1 から cgroupv2 に変更すると、自分の移行にどのような影響が及びますか？

[Kubernetes コミュニティ](#) は、cgroupv1 サポート (AL2 で使用) をメンテナンスモードに移行しました。つまり、今後は新機能は追加されず、重要なセキュリティ修正およびバグ修正のみが行われます。cgroupv2 を Kubernetes に導入するには、OS、カーネル、コンテナランタイム、Kubernetes コンポーネント間で互換性を確保する必要があります。そのためには、AL2023、Bottlerocket、Red Hat Enterprise Linux (RHEL) 9+、Ubuntu 22.04+、Debian 11+ など、cgroupv2 をデフォルトで有効にする Linux ディストリビューションが必要になります。これらのディストリビューションにはカーネルバージョン 5.8 以降が付属しています。こちらは Kubernetes で cgroupv2 をサポートするための最小要件です。詳細については「[About cgroup v2](#)」を参照してください。

カスタム AL2 AMI で Neuron が必要な場合、どうすればよいですか？

AL2 ベースの AMI では、完全に Neuron をベースとするアプリケーションをネイティブに実行することはできません。AL2 AMI で AWS Neuron を使用する場合は、Neuron がサポートしているコンテナを使って、非 AL2 Linux ディストリビューション (Ubuntu 22.04、Amazon Linux 2023 など) でアプリケーションをコンテナ化し、そのコンテナを Neuron ドライバー (aws-neuronx-dkms) がインストールされた AL2 ベースの AMI にデプロイする必要があります。

## 互換性とバージョン

AL2 AMI でサポートされている Kubernetes バージョン

Kubernetes バージョン 1.32 は Amazon EKS が Amazon Linux 2 (AL2) AMI をリリースする最後のバージョンです。[サポートされている](#) 1.32 までの Kubernetes バージョンに対して、EKS は、2025 年 11 月 26 日まで AL2 AMI (AL2\_ARM\_64、AL2\_x86\_64) と AL2 高速化 AMI (AL2\_x86\_64\_GPU) をリリースします。この日を過ぎると、EKS はすべての Kubernetes バージョンで AL2 最適化 AMI と AL2 高速化 AMI のリリースを停止します。EKS AL2 最適化 AMI と AL2 高速化 AMI の EOS 日は、EKS による Kubernetes バージョンの標準サポートおよび延長サポートのスケジュールとは無関係ですのでご注意ください。

## AL2、AL2023、Bottlerocket AMI の NVIDIA ドライバーの比較

| ドライバーのブランチ | Amazon Linux 2 AMI | Amazon Linux 2023 AMI | Bottlerocket AMI | サポート終了日                    |
|------------|--------------------|-----------------------|------------------|----------------------------|
| R535       | サポート外              | サポート外                 | サポート             | <a href="#">2027 年 9 月</a> |
| R550       | サポート               | サポート                  | サポート外            | <a href="#">2025 年 4 月</a> |
| R560       | サポート外              | サポート                  | サポート外            | <a href="#">2025 年 3 月</a> |
| R570       | サポート外              | サポート                  | 近日公開予定           | <a href="#">2026 年 2 月</a> |

詳細については、[Nvidia のリリースドキュメント](#)を参照してください。

## AL2、AL2023、Bottlerocket AMI の NVIDIA CUDA バージョン比較

| CUDA バージョン           | AL2 のサポート | AL2023 のサポート | Bottlerocket のサポート |
|----------------------|-----------|--------------|--------------------|
| <a href="#">10.1</a> | サポート      | サポートされません    | サポート外              |
| <a href="#">11.8</a> | サポート      | サポート対象       | サポート               |
| <a href="#">12.0</a> | サポート外     | サポート対象       | サポート               |
| <a href="#">12.5</a> | サポート外     | サポート対象       | サポート               |

詳細については、[CUDA のリリースドキュメント](#)を参照してください。

## AL2、AL2023、Bottlerocket AMI でサポートされているドライバーと Linux カーネルバージョンの比較

| コンポーネント    | AL2 AMI ソース    | AL2023 AMI ソース  | Bottlerocket AMI ソース |
|------------|----------------|-----------------|----------------------|
| 基本 OS の互換性 | RHEL7/CentOS 7 | Fedora/CentOS 9 | 該当なし                 |



| コンポーネント          | AL2 AMI ソース                    | AL2023 AMI ソース                        | Bottlerocket AMI ソース                  |
|------------------|--------------------------------|---------------------------------------|---------------------------------------|
| CUDA Toolkit     | <a href="#">CUDA 11.x–12.x</a> | <a href="#">CUDA 12.5 以降</a>          | CUDA 11.x (12.5 が間もなくリリース)            |
| NVIDIA GPU ドライバー | <a href="#">R550</a>           | <a href="#">R565</a> (R570 が間もなくリリース) | <a href="#">R535</a> (R570 が間もなくリリース) |
| AWS Neuron ドライバー | <a href="#">2.19</a>           | <a href="#">2.19 以降</a>               | 2.20                                  |
| Linux Kernel     | <a href="#">5.10</a>           | <a href="#">6.1、6.12</a>              | 5.15、6.1 (6.12 が間もなくリリース)             |

## AL2 AMI と AWS Neuron の互換性

[AWS Neuron の 2.20 のリリース](#)以降は、EKS AL ベースの AMI で使用される Neuron ランタイム (aws-neuronx-runtime-lib) では、Amazon Linux 2 (AL2) がサポートされなくなります。現在、Amazon Linux 2 をサポートしている AWS Neuron パッケージは、Neuron ドライバー (aws-neuronx-dkms) のみです。したがって、AL2 ベースの AMI では、Neuron をベースとするアプリケーションをネイティブに実行することはできません。AL2023 AMI で Neuron をセットアップする方法については、「[AWS Neuron Setup](#)」のガイドを参照してください。

## Kubernetes と AL2 AMI の互換性

Kubernetes コミュニティは、cgroupv1 サポート (AL2 で使用) をメンテナンスモードに移行しました。つまり、今後は新機能は追加されず、重要なセキュリティ修正およびバグ修正のみが行われます。cgroupv2 に依存する Kubernetes の機能 (MemoryQoS や拡張リソース分離など) は、AL2 では使用することができません。さらに、Amazon EKS Kubernetes バージョン 1.32 は、AL2 AMI をサポートする最後のバージョンとなります。最新の Kubernetes バージョンとの互換性を維持するため、当社は AL2023 または Bottlerocket への移行を推奨しています。これらは cgroupv2 がデフォルトで有効になります。

## Linux バージョンと AL2 AMI の互換性

Amazon Linux 2 (AL2) は、2026 年 6 月 30 日のサポート終了 (EOS) 日まで AWS でサポートされます。ただし、AL2 はリリースから長い時間が経過しているため、新しいアプリケーションや機能に対する、より広範な Linux コミュニティによるサポートは、ますます制限されています。AL2 AMI

は [Linux カーネル 5.10](#) をベースにしており、一方の AL2023 は [Linux カーネル 6.10](#) を使用しています。AL2023 と異なり、AL2 はより広範な Linux コミュニティからのサポートが、制限されています。つまり、AL2 の古いカーネルバージョンと連携するには、アップストリームの多くの Linux パッケージやツールをバックポートする必要があります。最新の Linux 機能やセキュリティ強化機能の中には、カーネルが古いために使用できないものがあり、オープンソースのプロジェクトの多くが、5.10 などの古いカーネルバージョンのサポートを廃止するか制限しています。

### AL2023 に含まれていない非推奨パッケージ

AL2023 に含まれていない、または AL2023 で変更された最も一般的なパッケージには、次のようなものがあります。

- [Amazon Linux 2 の一部のソースバイナリパッケージ](#) は、Amazon Linux 2023 で利用できなくなりました。
- Amazon Linux による、AL2023 のさまざまなバージョンのパッケージ ([amazon-linux-extras システム](#) など) のサポート方法が変更されています。
- [Enterprise Linux \(EPEL\) の追加パッケージ](#) は AL2023 ではサポートされていません。
- [32-bit アプリケーション](#) は AL2023 ではサポートされていません。
- AL2023 は [mate-desktop](#) パッケージを作成していません。

詳細については、「[AL2 と AL2023 の比較](#)」を参照してください。

### AL2、AL2023、Bottlerocket 間の FIPS 検証の比較

Amazon Linux 2 (AL2)、Amazon Linux 2023 (AL2023)、Bottlerocket は、連邦情報処理標準 (FIPS) 適合性をサポートしています。

- AL2 は FIPS 140-2 に基づいて認定され、AL2023 は FIPS 140-3 に基づいて認定されています。AL2023 で FIPS モードを有効にするには、Amazon EC2 インスタンスに必要なパッケージをインストールし、「[Enable FIPS Mode on AL2023](#)」に記載された手順に従って設定を行います。詳細については、「[Amazon Linux 2023 に関するよくある質問](#)」を参照してください。
- Bottlerocket は、カーネルとユーザースペースのコンポーネントを、FIPS 140-3 Cryptographic Module Validation Program (暗号モジュール試験及び認証制度) に送信された暗号モジュールの使用に制限する、FIPS 専用のバリエーションを提供します。

## EKS AMI ドライバーとバージョンの変更ログ

EKS AMI のコンポーネントとそのバージョンのリストの完全版は、GitHub の「[Amazon EKS AMI Release Notes](#)」を参照してください。

## 最適化された Amazon Linux AMI を使用してノードを作成する

Amazon EKS 最適化 Amazon Linux AMI は、Amazon Linux 2 (AL2) および Amazon Linux 2023 (AL2023) 上に構築されています。Amazon EKS ノードのベースイメージとして機能するように構成されています。AMI は、Amazon EKS と連携するように構成されており、次のコンポーネントが含まれています。

- kubelet
- AWS IAM オーセンティケーター
- containerd

### Note

- Amazon Linux のセキュリティまたはプライバシーに関するイベントを、[Amazon Linux セキュリティセンター](#)で追跡できます。そのためには目的のバージョンのタブを選択してください。該当する RSS フィードをサブスクライブすることもできます。セキュリティおよびプライバシーイベントには問題の概要、影響を受けるパッケージ、および問題を修正するためにインスタンスを更新する方法などがあります。
- 高速または Arm AMI をデプロイする前に、[Amazon EKS 最適化高速 Amazon Linux AMI](#) および [the section called “Amazon EKS 最適化 Arm Amazon Linux AMI”](#) の情報を確認してください。
- Amazon EC2 P2 インスタンスは、NVIDIA ドライバーバージョン 470 以前を必要とするため、Amazon EKS ではサポートされていません。
- バージョン 1.30 以降のクラスターで新しく作成されたマネージド型ノードグループは、ノードオペレーティングシステムとして自動的に AL2023 を使用するようデフォルトで設定されます。以前は、新しいノードグループはデフォルトで AL2 を使用するよう設定されていました。新しいノードグループを作成するときに AL2 を AMI タイプとして選択すれば、AL2 を引き続き使用できます。
- Amazon EKS は、2025 年 11 月 26 日以降、EKS 最適化 Amazon Linux 2 (AL2) AMI を公開しなくなります。また、Kubernetes バージョン 1.32 は Amazon EKS が AL2 AMI を

リリースする最後のバージョンです。バージョン 1.33 以降、Amazon EKS は引き続き AL2023 および Bottlerocket ベースの AMI をリリースします。

## Amazon EKS 最適化高速 Amazon Linux AMI

Amazon EKS 最適化高速 Amazon Linux AMI は、標準的な Amazon EKS 最適化 Amazon Linux AMI 上に構築されています。これは、Amazon EKS ノードのオプションのイメージとして機能し、GPU、[Inferentia](#)、[Trainium](#) ベースのワークロードをサポートするように設定されています。

標準の Amazon EKS 最適化 AMI 設定に加えて、高速 AMI には、以下が備わっています。

- NVIDIA ドライバー
- `nvidia-container-toolkit`
- AWS Neuron ドライバー

高速 AMI に含まれる最新のコンポーネント一覧については、GitHub の `amazon-eks-ami` の「[Releases](#)」を参照してください。

### Note

- ノードの AWS CloudFormation テンプレートには、適用可能なインスタンスタイプを指定するようにしてください。Amazon EKS 最適化高速 AMI を使用することで、[NVIDIA のクラウドエンドユーザーライセンス契約 \(EULA\)](#) に同意したものとみなされます。
- Amazon EKS 最適化高速 AMI は、以前は、GPU をサポートする Amazon EKS 最適化 AMI と呼ばれていたものです。
- これまでのバージョンの Amazon EKS 最適化高速 AMI では、`nvidia-docker` リポジトリがインストールされていました。このリポジトリは、Amazon EKS AMI バージョン v20200529 以降では包含されなくなります。

Amazon EKS 最適化アクセラレーテッド Amazon Linux AMI でのワークロードの実行に関する詳細については、「[the section called “Linux GPU AMI を実行する”](#)」を参照してください。

## Amazon EKS 最適化 Arm Amazon Linux AMI

Arm インスタンスは、ウェブサーバー、コンテナ化されたマイクロサービス、キャッシュフリート、および分散データストアといったスケールアウト型の Arm ベースアプリケーションにおいて、コストを大幅に削減します。クラスターに Arm ノードを追加する際には、次の考慮事項を確認してください。

- クラスターが 2020 年 8 月 17 日より前にデプロイされている場合、クラスターの重要なアドオンマニフェストを 1 回だけアップグレードする必要があります。これは、クラスター内で使用中の各ハードウェアアーキテクチャのイメージを、Kubernetes が正しく取得できるようにするためです。クラスターのアドオン更新の詳細については、「[the section called “ステップ 1: アップグレードの準備をする”](#)」を参照してください。2020 年 8 月 17 日以降にクラスターをデプロイしている場合、ご使用の CoreDNS、kube-proxy、および Amazon VPC CNI Plugin for Kubernetes アドオンは、既にマルチアーキテクチャに対応しています。
- Arm ノードにデプロイされたアプリケーションは、Arm 用にコンパイルする必要があります。
- 既存のクラスターでデプロイ済みの DaemonSet がある場合、または新しいクラスターで Arm ノードと共にこれをデプロイする場合は、クラスター内のすべてのハードウェアアーキテクチャで DaemonSet が実行可能であることを確認します。
- 同じクラスター内で、Arm ノードグループと x86 ノードグループを実行することができます。その場合、Pod をデプロイできるハードウェアアーキテクチャを Kubernetes が認識できるようにするために、マルチアーキテクチャのコンテナイメージを Amazon Elastic Container Registry などのコンテナリポジトリにデプロイした上で、ノードセレクターをマニフェストに追加する作業も考慮に入れてください。詳細については、Amazon ECR ユーザーガイドの[マルチアーキテクチャイメージのプッシュ](#)と、ブログ記事 [Introducing multi-architecture container images for Amazon ECR](#) を参照してください。

### 詳細情報

Amazon EKS 最適化 Amazon Linux AMI の詳細については、以下のセクションを参照してください。

- Amazon Linux をマネージド型ノードグループと使用するには、「[the section called “マネージド型ノードグループ”](#)」を参照してください。
- セルフマネージド型の Amazon Linux ノードの起動には、「[the section called “最新の ID を取得する”](#)」を参照してください。
- バージョンについては、「[the section called “バージョン情報を取得する”](#)」を参照してください。

- Amazon EKS 最適化 Amazon Linux AMI の最新の ID を取得するには、「[the section called “最新の ID を取得する”](#)」を参照してください。
- Amazon EKS 最適化 AMI の構築に使用されているオープンソーススクリプトについては、「[the section called “カスタムビルド”](#)」を参照してください。

## Amazon Linux 2 から Amazon Linux 2023 にアップグレードする

Amazon EKS 最適化 AMI は、AL2 および AL2023 をベースとする 2 つのファミリーで使用できます。AL2023 は、クラウドアプリケーションに安全かつ安定した高パフォーマンス環境を提供するように設計された、新しい Linux ベースのオペレーティングシステムです。これは Amazon Web Services の次世代 Amazon Linux であり、サポートされているすべての Amazon EKS バージョンで利用できます。

AL2023 には、AL2 に比べていくつかの改善点があります。詳細については、「Amazon Linux 2023 ユーザーガイド」の「[Comparing AL2 and AL2023](#)」を参照してください。AL2 からいくつかのパッケージが追加、アップグレード、削除されています。アップグレードする前に、AL2023 でアプリケーションをテストすることを強くお勧めします。AL2023 のパッケージに関するすべての変更一覧については、「Amazon Linux 2023 リリースノート」の「[Package changes in Amazon Linux 2023](#)」を参照してください。

これらの変更に加えて、以下の点に注意してください。

- AL2023 では、YAML 設定スキーマを使用する新しいノード初期化プロセス nodeadm が導入されています。セルフマネージド型ノードグループまたは起動テンプレートを持つ AMI を使用している場合は、新しいノードグループの作成時に追加のクラスターメタデータを明示的に指定する必要があります。最低限必要なパラメータの[例](#)を以下に示します。ここで、apiServerEndpoint、certificateAuthority、サービスの cidr が必要になります。

```

apiVersion: node.eks.aws/v1alpha1
kind: NodeConfig
spec:
 cluster:
 name: my-cluster
 apiServerEndpoint: https://example.com
 certificateAuthority: Y2VydGhmaWNhdGVBdXRob3JpdHk=
 cidr: 10.100.0.0/16
```



AL2 では、これらのパラメータからのメタデータは Amazon EKS DescribeCluster API コールから検出されていました。AL2023 ではノードの大規模なスケールアップ中に API コールによってスロットリングが発生するリスクがあるため、この動作が変更されました。この変更は起動テンプレートのないマネージド型ノードグループを使用している場合や、Karpenter を使用している場合には影響しません。詳細については、「Amazon EKS API リファレンス」の [certificateAuthority](#)、サービスの cidr、[DescribeCluster](#) を参照してください。

- AL2023 の場合、nodeadm は、[NodeConfigSpec](#) を使用して各ノードの kubelet にパラメータを適用する形式も変更します。AL2 では、これは `--kubelet-extra-args` パラメータを使用して行われました。これは一般的に、ノードにラベルとテイントを追加するために使用されます。以下の例は、ノードへの `maxPods` と `--node-labels` の適用を示しています。

```

apiVersion: node.eks.aws/v1alpha1
kind: NodeConfig
spec:
 cluster:
 name: test-cluster
 apiServerEndpoint: https://example.com
 certificateAuthority: Y2VydGlmawNhdGVBdXRob3JpdHk=
 cidr: 10.100.0.0/16
 kubelet:
 config:
 maxPods: 110
 flags:
 - --node-labels=karpenter.sh/capacity-type=on-demand,karpenter.sh/nodepool=test
```

- AL2023 には Amazon VPC CNI バージョン 1.16.2 以降が必要です。
- AL2023 にはデフォルトで IMDSv2 が必要です。IMDSv2 には、セキュリティ体制の改善に役立ついくつかの利点があります。セッション指向の認証方式を使用しており、セッションを開始するためにシンプルな HTTP PUT リクエストでシークレットトークンを作成する必要があります。セッショントークンの有効期間は 1 秒～6 時間です。IMDSv1 から IMDSv2 への移行方法の詳細については、「[インスタンスメタデータサービスバージョン 2 の使用への移行](#)」および「[Get the full benefits of IMDSv2 and disable IMDSv1 across your AWS infrastructure](#)」を参照してください。IMDSv1 を使用する場合は、インスタンスのメタデータオプションの起動プロパティで設定を手動で上書きすることで使用可能になります。

**Note**

IMDSv2 の場合、マネージド型ノードグループのデフォルトのホップカウントは 1 に設定されています。つまり、コンテナが IMDS を使用してノードの認証情報にアクセスすることはできません。ノードの認証情報へのコンテナアクセスが必要な場合は、[カスタム Amazon EC2 起動テンプレート](#)の `HttpPutResponseHopLimit` を手動で書き換えて 2 に増やすことでアクセスが可能になります。または、IMDSv2 の代わりに [Amazon EKS Pod Identity](#) を使用して、認証情報を提供することもできます。

- AL2023 は、次世代の統合コントロールグループ階層 (cgroupv2) を備えています。cgroupv2 は、コンテナランタイムを実装するために systemd によって使用されます。AL2023 には、cgroupv1 を使用してシステムを実行できるコードが引き続き含まれていますが、この設定は推奨されません。またサポート対象でもありません。この設定は、今後の Amazon Linux のメジャーリリースで完全に削除される予定です。
- AL2023 をサポートするには、eksctl に eksctl のバージョン 0.176.0 以降が必要です。

既存のマネージド型ノードグループの場合は、起動テンプレートの使用方法に応じて、インプレースアップグレードまたは Blue/Green アップグレードを実行できます。

- マネージド型ノードグループでカスタム AMI を使用している場合は、起動テンプレートの AMI ID を入れ替えることで、インプレースアップグレードを実行できます。このアップグレード戦略を実行する前に、まずアプリケーションとユーザーデータが AL2023 に転送されていることを必ず確認してください。
- 標準の起動テンプレートまたは AMI ID を指定しないカスタム起動テンプレートでマネージド型ノードグループを使用している場合は、Blue/Green 戦略を使用してアップグレードする必要があります。通常、Blue/Green アップグレードはより複雑で、AMI タイプとして AL2023 を指定する完全に新しいノードグループを作成する必要があります。新しいノードグループの設定は、AL2 ノードグループのすべてのカスタムデータが新しい OS と互換性があることを確認して、慎重に行う必要があります。新しいノードグループがアプリケーションでテストおよび検証されると、Pod を古いノードグループから新しいノードグループに移行できます。移行が完了したら、古いノードグループを削除できます。

Karpenter を利用していて、AL2023 を使用する場合は、`EC2NodeClass amiFamily` フィールドを AL2023 に変更する必要があります。デフォルトでは、Karpenter ではドリフトが有効になっていま



す。つまり、amiFamily フィールドが変更されると、Karpenter はワーカーノードを使用可能な最新の AMI に自動的に更新します。

## Amazon Linux AMI のバージョン情報を取得する

Amazon EKS 最適化 Amazon Linux AMI は、Kubernetes バージョンと AMI のリリース日によって次の形式でバージョンングされています。

```
k8s_major_version.k8s_minor_version.k8s_patch_version-release_date
```

各 AMI リリースには、[kubelet](#)、Linux カーネル、[containerd](#) のさまざまなバージョンが含まれます。また高速 AMI には、さまざまなバージョンの NVIDIA ドライバーも含まれます。このバージョンに関する情報は、GitHub の「[Changelog](#)」で確認できます。

## 推奨 Amazon Linux AMI ID を取得する

ノードをデプロイする際に、事前構築済みの Amazon EKS 最適化 Amazon マシンイメージ (AMI) の ID を指定できます。希望する設定に合った AMI ID を取得するには、AWS Systems Manager Parameter Store API をクエリします。この API を使用すると、Amazon EKS 最適化 AMI ID を手動で検索する必要がなくなります。詳細については、「[GetParameter](#)」を参照してください。使用する [IAM プリンシパル](#) には、Amazon EKS 最適化 AMI メタデータを取得するための `ssm:GetParameter` IAM アクセス許可が必要です。

サブパラメータ `image_id` を指定する次のコマンドを使用することで、推奨される最新の Amazon EKS 最適化 Amazon Linux AMI のイメージ ID を取得できます。必要に応じてコマンドに次の変更を加え、変更したコマンドを実行してください：

- `<kubernetes-version>` をサポートされる [Amazon EKS バージョン](#) に置き換えます。
- `ami-type` は、以下のいずれかのオプションに置き換えます。Amazon EC2 インスタンスタイプの詳細については、「[Amazon EC2 インスタンスタイプ](#)」を参照してください。
- Amazon Linux 2023 (AL2023) x86 ベースのインスタンスには `amazon-linux-2023/x86_64/standard` を使用します。
- [AWS Graviton](#) ベースのインスタンスなどの AL2023 ARM インスタンスには `amazon-linux-2023/arm64/standard` を使用します。
- 最新の承認済みの AL2023 NVIDIA x86 ベースのインスタンスには、`amazon-linux-2023/x86_64/nvidia` を使用します。
- 最新の承認済みの AL2023 NVIDIA arm64 ベースのインスタンスには、`amazon-linux-2023/arm64/nvidia` を使用します。

- 最新の AL2023 [AWS Neuron](#) インスタンスには、*amazon-linux-2023/x86\_64/neuron* を使用します。
- Amazon Linux 2 (AL2) x86 ベースのインスタンスには *amazon-linux-2* を使用します。
- [AWS Graviton](#) ベースのインスタンスなどの AL2 ARM インスタンスには *amazon-linux-2-arm64* を使用します。
- NVIDIA GPU、[Inferentia](#)、および [Trainium](#) ベースのワークロードに対する AL2 [ハードウェア高速化](#) x86 ベースのインスタンスに *amazon-linux-2-gpu* を使用します。
- <region-code> を、AMI ID を必要とする [Amazon EKS がサポートされている AWS リージョン](#) で置き換えます。

```
aws ssm get-parameter --name /aws/service/eks/optimized-ami/<kubernetes-version>/<ami-type>/recommended/image_id \
 --region <region-code> --query "Parameter.Value" --output text
```

プレースホルダーの置換が行われた後のコマンドの例を以下に示します。

```
aws ssm get-parameter --name /aws/service/eks/optimized-ami/1.31/amazon-
linux-2023/x86_64/standard/recommended/image_id \
 --region us-west-2 --query "Parameter.Value" --output text
```

出力例は次のとおりです。

```
ami-1234567890abcdef0
```

## カスタム Amazon Linux AMI を構築する

### Important

Amazon EKS は、2025 年 11 月 26 日以降、EKS 最適化 Amazon Linux 2 (AL2) AMI を公開しなくなります。また、Kubernetes バージョン 1.32 は Amazon EKS が AL2 AMI をリリースする最後のバージョンです。バージョン 1.33 以降、Amazon EKS は引き続き AL2023 および Bottlerocket ベースの AMI をリリースします。

Amazon EKS に最適化された Amazon Linux (AL) AMI は、AL2 および AL2023 上に構築されており、特に Amazon EKS クラスターのノードに使用することを目的としています。Amazon EKS で

は、「[Amazon EKS AMI Build Specification](#)」リポジトリにオープンソースのビルドスクリプトが用意されており、次の方法で使用できます。\* kubelet、ランタイム環境、AWS IAM Authenticator for Kubernetes の設定を表示します。\* 独自の AL ベース AMI をゼロから構築します。

このリポジトリには、ブート時に実行される特殊な [bootstrap スクリプト](#) と [nodeadm スクリプト](#) が保存されています。これらのスクリプトによって、インスタンスの証明書データ、コントロールプレーンエンドポイント、クラスター名などを設定します。ここに置かれているものは、Amazon EKS 最適化 AMI のビルド用として最も信頼できるスクリプトと見なされるため、GitHub リポジトリの状態をフォローすることで、AMI への変更をモニタリングできます。

## 前提条件

- [AWS CLI をインストールする](#)
- [HashiCorp Packer v1.9.4+ をインストールする](#)
- [GNU Make をインストールする](#)

## クイックスタート

このセクションでは、AWS アカウントにカスタム AMI を作成するコマンドを示します。AMI のカスタマイズに使用できる設定の詳細については、「[Amazon Linux 2023](#)」ページのテンプレート変数を参照してください。

### ステップ 1. 環境をセットアップする

公式の Amazon EKS AMI リポジトリをクローンまたはフォークします。例：

```
git clone https://github.com/aws-labs/amazon-eks-ami.git
cd amazon-eks-ami
```

Packer がインストールされていることを確認します。

```
packer --version
```

### ステップ 2. カスタム AMI を作成する

さまざまなカスタム AMI のコマンド例を以下に示します。

#### 基本的な NVIDIA AL2 AMI

```
make k8s=1.31 os_distro=al2 \
 enable_accelerator=nvidia \
 nvidia_driver_major_version=560 \
 enable_efa=true
```

## 基本的な NVIDIA AL2023 AMI

```
make k8s=1.31 os_distro=al2023 \
 enable_accelerator=nvidia \
 nvidia_driver_major_version=560 \
 enable_efa=true
```

## STIG 準拠 Neuron AL2023 AMI

```
make k8s=1.31 os_distro=al2023 \
 enable_accelerator=neuron \
 enable_fips=true \
 source_ami_id=ami-0abcd1234efgh5678 \
 kms_key_id=alias/aws-stig
```

これらのコマンドを実行すると、Packer によって次の操作が行われます。\* 一時的な Amazon EC2 インスタンスを起動します。\* Kubernetes コンポーネント、ドライバー、設定をインストールします。\* AWS アカウントで AMI を作成します。

## ステップ 3. デフォルト値を表示する

デフォルト値と追加オプションを表示するには、次のコマンドを実行します。

```
make help
```

## 最適化された Bottlerocket AMI を使用してノードを作成する

[Bottlerocket](#) は、AWS がスポンサーおよびサポートするオープンソースの Linux ディストリビューションです。Bottlerocket は、コンテナワークロードをホストするために専用に構築されています。Bottlerocket を使用すると、コンテナインフラストラクチャの更新を自動化することで、コンテナ化されたデプロイの可用性を向上させ、運用コストを削減できます。Bottlerocket には、コンテナの実行に不可欠なソフトウェアのみが含まれており、リソース使用率の向上、セキュリティ上の脅威の軽減、管理オーバーヘッドの軽減が実現されます。Bottlerocket AMI には、containerd、kubelet、および AWS IAM Authenticator が含まれます。Bottlerocket は、マ

マネージド型ノードグループとセルフマネージド型ノードに加え、[Karpenter](#) でもサポートされています。

## 利点

Amazon EKS クラスターと Bottlerocket を使用すると次のようなメリットがあります。

- 運用コストが低く、管理の複雑さが軽減されたことによるアップタイムの向上 – Bottlerocket は、他の Linux ディストリビューションよりもリソースフットプリントが小さく、ブート時間が短く、セキュリティ脅威に対する脆弱性が低くなります。Bottlerocket のフットプリントが小さいため、ストレージ、コンピューティング、ネットワークリソースの使用量が減り、コストを削減できます。
- OS の自動更新によるセキュリティの向上 — Bottlerocket への更新プログラムは単一のユニットとして適用され、必要に応じてロールバックできます。これにより、システムが使用不能な状態にさらす更新プログラムの破損または失敗のリスクをなくします。Bottlerocket を使用すると、セキュリティ更新プログラムが利用可能になり次第、中断を最小限に抑えながら自動的に適用され、障害が発生した場合はロールバックできます。
- プレミアムサポート — Amazon EC2 で AWS が提供する Bottlerocket のビルドは、Amazon EC2、Amazon EKS、Amazon ECR などの AWS サービスも対象とする同じ AWS サポートプランの対象となります。

## 考慮事項

Bottlerocket を AMI タイプに使用する際、次の事項を考慮してください。

- Bottlerocket は、x86\_64 と arm64 プロセッサを搭載した Amazon EC2 インスタンスをサポートします。Bottlerocket AMI を Inferentia チップを搭載した Amazon EC2 インスタンスで使用することはお勧めしません。
- Bottlerocket イメージには、SSH サーバーまたはシェルは含まれません。帯域外のアクセス方法を使用して SSH を許可できます。これらの手法は、管理者コンテナを有効にし、ユーザーデータでブートストラップの設定ステップの渡しを可能にします。詳細については、GitHub の「[Bottlerocket OS](#)」内にある次のセクションを参照してください。
  - [探査](#)
  - [管理者コンテナ](#)
  - [Kubernetes 設定](#)
- Bottlerocket は以下のさまざまなコンテナタイプを使用します。

- デフォルトで、「[コントロールコンテナ](#)」は有効になっています。このコンテナは、Amazon EC2 Bottlerocket インスタンス上でのコマンドの実行や、シェルセッションの開始に使用できる [AWS Systems Manager エージェント](#) を実行します。詳細については、AWS Systems Manager ユーザーガイドの「[Session Manager のセットアップ](#)」を参照してください。
- ノードグループの作成時に SSH キーが与えられる場合、管理者コンテナが有効になります。admin container は、開発とテストのシナリオにのみ使用することをお勧めします。本番環境で使用することはお勧めしません。詳細については、GitHub の [Admin container](#) を参照してください。

## 詳細情報

Amazon EKS 最適化 Bottlerocket AMI の使用に関する詳細については、以下のセクションを参照してください。

- Bottlerocket の詳細については、[Bottlerocket のドキュメント](#) を参照してください。
- バージョン情報のリソースについては、「[the section called “バージョン情報を取得する”](#)」を参照してください。
- Bottlerocket をマネージド型ノードグループと使用するには、「[the section called “マネージド型ノードグループ”](#)」を参照してください。
- セルフマネージド型の Bottlerocket ノードを起動するには、「[the section called “Bottlerocket”](#)」を参照してください。
- Amazon EKS 最適化 Bottlerocket AMI の最新の ID を取得するには、「[the section called “最新の ID を取得する”](#)」を参照してください。
- コンプライアンスサポートの詳細については、「[the section called “コンプライアンスサポート”](#)」を参照してください。

## Bottlerocket AMI のバージョン情報を取得する

各 Bottlerocket AMI リリースには、[kubelet](#)、Bottlerocket カーネル、[containerd](#) のさまざまなバージョンが含まれます。また高速 AMI バリエーションには、さまざまなバージョンの NVIDIA ドライバーも含まれます。このバージョン情報は、Bottlerocket のドキュメントの「[OS](#)」のトピックで確認できます。このページから、該当する「Version Information」のサブトピックに移動します。

Bottlerocket のドキュメントは、GitHub で利用可能なバージョンよりも遅れることがあります。最新バージョンの変更のリストは、GitHub の「[Releases](#)」で確認できます。

## 推奨 Bottlerocket AMI ID を取得する

ノードをデプロイする際に、事前構築済みの Amazon EKS 最適化 Amazon マシンイメージ (AMI) の ID を指定できます。希望する設定に合った AMI ID を取得するには、AWS Systems Manager Parameter Store API をクエリします。この API を使用すると、Amazon EKS 最適化 AMI ID を手動で検索する必要がなくなります。詳細については、「[GetParameter](#)」を参照してください。使用する [IAM プリンシパル](#)には、Amazon EKS 最適化 AMI メタデータを取得するための `ssm:GetParameter` IAM アクセス許可が必要です。

サブパラメータ `image_id` を使用する次の AWS CLI コマンドを使用することで、推奨される最新の Amazon EKS 最適化 Bottlerocket AMI のイメージ ID を取得できます。必要に応じてコマンドに次の変更を加え、変更したコマンドを実行してください：

- `kubernetes-version` をサポートされる [Amazon EKS バージョン](#) に置き換えます。
- `-flavor` は、以下のいずれかのオプションに置き換えます。
  - GPU のないバリエーションの場合は `-flavor` を削除します。
  - GPU 対応バリエーションには `-nvidia` を使用します。
  - FIPS 対応バリエーションには `-fips` を使用します。
- `architecture` は、以下のいずれかのオプションに置き換えます。
  - x86 ベースのインスタンスには `x86_64` を使用します。
  - ARM インスタンスには `arm64` を使用します。
- `region-code` を、AMI ID が必要な [Amazon EKS がサポートしている AWS リージョン](#) に置き換えます。

```
aws ssm get-parameter --name /aws/service/bottlerocket/aws-k8s-kubernetes-version-flavor/architecture/latest/image_id \
 --region region-code --query "Parameter.Value" --output text
```

プレースホルダーの置換が行われた後のコマンドの例を以下に示します。

```
aws ssm get-parameter --name /aws/service/bottlerocket/aws-k8s-1.31/x86_64/latest/
image_id \
 --region us-west-2 --query "Parameter.Value" --output text
```

出力例は次のとおりです。



```
ami-1234567890abcdef0
```

## Bottlerocket を使用してコンプライアンス要件に準拠する

Bottlerocket はさまざまな組織によって定義された推奨事項に準拠しています。

- [CIS ベンチマーク](#) は Bottlerocket に定義されています。デフォルト設定では、Bottlerocket イメージには CIS レベル 1 構成プロファイルに必要なほとんどのコントロールが含まれています。CIS レベル 2 構成プロファイルに必要なコントロールを実装できます。詳細については、「AWS ブログ」の「[Amazon EKS に最適化された Bottlerocket AMI を CIS ベンチマークと照合して検証](#)」を参照してください。
- 最適化された機能セットおよび最小化したアタックサーフェスは、攻撃対象領域が小さくなるため、Bottlerocket インスタンスが PCI DSS 要件を満たすために必要な構成が少なく済みます。[Bottlerocket の CIS Benchmark](#) は、ガイダンスを強化するために優れたリソースであり、PCI DSS 要件 2.2 に基づく安全な構成標準の要件をサポートします。[Fluent Bit](#) を活用して、PCI DSS 要件 10.2 に基づくオペレーティングシステムレベルの監査ログ記録の要件をサポートすることもできます。AWS は、PCI DSS 要件 6.2 (v3.2.1 用) および要件 6.3.3 (v4.0 用) を満たすために役立つ新しい (パッチが適用された) Bottlerocket インスタンスを定期的に公開します。
- Bottlerocket は、Amazon EC2 および Amazon EKS の両方に規制対象のワークロードで使用が承認された HIPAA 対象の機能です。詳細については、「[コンプライアンスプログラムによる AWS 対象範囲内のサービス](#)」を参照してください。
- Bottlerocket AMI は FIPS 140-3 検証済み暗号化モジュールを使用するように事前設定されています。これには Amazon Linux 2023 Kernel Crypto API 暗号化モジュールや AWS-LC 暗号化モジュールなどがあります。詳細については、「[the section called “Bottlerocket FIPS AMI”](#)」を参照してください。

## Bottlerocket FIPS AMI を使用してワーカーノードを FIPS 対応にする

連邦情報処理規格 (Federal Information Processing Standards/FIPS) 公開情報 140-3 は、機密情報を保護する暗号化モジュールのセキュリティ要件を規定する、米国およびカナダ政府の基準です。Bottlerocket からは、FIPS カーネルを備えた AMI が提供されており、これによって、FIPS への準拠が容易になります。

これらの AMI は、FIPS 140-3 検証済み暗号化モジュールを使用するように事前構成されています。これには Amazon Linux 2023 Kernel Crypto API 暗号化モジュールや AWS-LC 暗号化モジュールなどがあります。



Bottlerocket FIPS AMI を使用すると、ワーカーノードは「FIPS 対応」になりますが、自動的に「FIPS に準拠」するわけではありません。詳細については、「[連邦情報処理規格 \(FIPS\) 140-3](#)」を参照してください。

### 考慮事項

- クラスターで分離サブネットを使用している場合、Amazon ECR FIPS エンドポイントにアクセスできない場合があります。これによって、ノードのブートストラップが失敗する可能性があります。ネットワーク設定で、必要な FIPS エンドポイントへのアクセスが許可されていることを確認してください。詳細については、「AWS PrivateLink ガイド」の「[Access a resource through a resource VPC endpoint](#)」を参照してください。
- Amazon ECR FIPS エンドポイントは PrivateLink 経由で使用できないため、クラスターで [PrivateLink](#) を利用するサブネットが使用されている場合、イメージのプルが失敗します。

Bottlerocket FIPS AMI を使用してマネージド型ノードグループを作成する

Bottlerocket FIPS AMI には、ワークロードをサポートする 2 つのバリエーションがあります。

- BOTTLEROCKET\_x86\_64\_FIPS
- BOTTLEROCKET\_ARM\_64\_FIPS

Bottlerocket FIPS AMI を使用してマネージド型ノードグループを作成するには、作成プロセス中に、該当する AMI タイプを選択します。詳細については、「[the section called “作成”](#)」を参照してください。

FIPS 対応バリエーションの選択の詳細については、「[the section called “最新の ID を取得する”](#)」を参照してください。

サポートされていない AWS リージョンの FIPS エンドポイントを無効にする

Bottlerocket FIPS AMI は、GovCloud (米国) AWS リージョンを含め、米国では直接サポートされています。AMI が使用可能でも、直接サポートされていない AWS リージョンでは、起動テンプレートでマネージド型ノードグループを作成すると、AMI を引き続き使用できます。

Bottlerocket FIPS AMI は、ブートストラップ中に Amazon ECR FIPS エンドポイントに依存しますが、このエンドポイントは、米国以外では利用できません。Amazon ECR FIPS エンドポイントが利用できない AWS リージョンで FIPS カーネルを目的に AMI を使用するには、次のステップを実行して FIPS エンドポイントを無効にします。

1. 以下を記述した新しい設定ファイルを作成するか、既存の設定ファイルに以下を記述します。

```
[default]
use_fips_endpoint=false
```

1. ファイルの記述を Base64 形式でエンコードします。
2. 起動テンプレートの UserData で、エンコードした次の文字列を TOML 形式で追加します。

```
[settings.aws]
config = "<your-base64-encoded-string>"
```

その他の設定については、GitHub にある、Bottlerocket の「[Description of settings](#)」を参照してください。

起動テンプレートに記述した UserData の例を次に示します。

```
[settings]
motd = "Hello from eksctl!"
[settings.aws]
config = "W2RlZmF1bHRdCnVzZV9maXBzX2VuZHBvaW50PWZhbnHNlCg==" # Base64-encoded string.
[settings.kubernetes]
api-server = "<api-server-endpoint>"
cluster-certificate = "<cluster-certificate-authority>"
cluster-name = "<cluster-name>"
...<other-settings>
```

起動テンプレート作成の詳細については、「[the section called “起動テンプレート”](#)」を参照してください。

## 最適化された Ubuntu Linux AMI を使用してノードを作成する

Canonical は Amazon EKS と提携し、クラスターで利用できるノード AMI を作成しました。

[Canonical](#) 社は、特定用途向けの Kubernetes ノード OS イメージ用のビルドを提供します。この最小化された Ubuntu イメージは Amazon EKS に最適化されたものであり、AWS が共同開発を行ったカスタム AWS カーネルが含まれます。詳細については、「[Ubuntu on Amazon Elastic Kubernetes Service \(EKS\)](#)」と「[the section called “Ubuntu Linux”](#)」を参照してください。サポートの詳細について

では、「AWS プレミアムサポートのよくある質問」の「[サードパーティソフトウェア](#)」セクションを参照してください。

## 最適化された Windows AMI を使用してノードを作成する

Windows Amazon EKS 最適化 AMI は Windows Server 2019 と Windows Server 2022 上に構築されています。Amazon EKS ノードのベースイメージとして機能するように構成されています。デフォルトでは、AMI には次のコンポーネントが含まれています。

- [kubelet](#)
- [kube-proxy](#)
- [AWS IAM Authenticator for Kubernetes](#)
- [csi-proxy](#)
- [containerd](#)

### Note

[Microsoft Security Update Guide](#) を使用して、Windows Server のセキュリティイベントやプライバシーイベントを追跡できます。

Amazon EKS は、次のバリエーションで Windows コンテナに最適化された AMI を提供しています。

- Amazon EKS 最適化 Windows Server 2019 Core AMI
- Amazon EKS 最適化 Windows Server 2019 Full AMI
- Amazon EKS 最適化 Windows Server 2022 Core AMI
- Amazon EKS 最適化 Windows Server 2022 Full AMI

### Important

- Amazon EKS 最適化 Windows Server 20H2 Core AMI は非推奨です。この AMI の新しいバージョンはリリースされません。
- 最新のセキュリティ更新プログラムがデフォルトで適用されるように、Amazon EKS は直近 4 か月間の最適化 Windows AMI を維持します。新しい AMI はそれぞれ、初回リリースから 4 か月間使用できます。この期間が過ぎると、古い AMI は非公開になり、アクセス

不可になります。セキュリティの脆弱性を回避し、サポート期間が終了した古い AMI にアクセスできなくなることを防ぐために、最新の AMI を使用することをお勧めします。非公開になった AMI へのアクセスを可能にできるかは保証できませんが、AWS サポートにチケットを提出することでアクセスをリクエストできます。

## リリースカレンダー

以下の表に、Amazon EKS 用の Windows 各バージョンについて、そのリリース日およびサポート終了日を示します。終了日が空白の場合、そのバージョンのサポートは継続されています。

| Windows のバージョン           | Amazon EKS リリース  | Amazon EKS のサポート終了 |
|--------------------------|------------------|--------------------|
| Windows Server 2022 Core | 2022 年 10 月 17 日 |                    |
| Windows Server 2022 Full | 2022 年 10 月 17 日 |                    |
| Windows Server 20H2 Core | 2021 年 8 月 12 日  | 2022 年 8 月 9 日     |
| Windows Server 2004 Core | 2020 年 8 月 19 日  | 2021 年 12 月 14 日   |
| Windows Server 2019 Core | 2019 年 10 月 7 日  |                    |
| Windows Server 2019 Full | 2019 年 10 月 7 日  |                    |
| Windows Server 1909 Core | 2019 年 10 月 7 日  | 2020 年 12 月 8 日    |

## ブートストラップスクリプトの設定パラメータ

Windows ノードを作成すると、ノード上にさまざまなパラメータが設定できるスクリプトができます。設定によりますが、このスクリプトはノード上の次のような場所で確認できます。C:\Program Files\Amazon\EKS\Start-EKSBootstrap.ps1。カスタムパラメータ値は、ブートストラップスクリプトの引数として指定できます。例えば、起動テンプレートのユーザーデータを更新できます。詳細については、「[the section called “Amazon EC2 ユーザーデータ”](#)」を参照してください。

このスクリプトには次のコマンドラインパラメータが含まれます。

- -EKSClusterName - このワーカーノードを参加させるための Amazon EKS クラスター名を指定します。

- `-KubeletExtraArgs` - kubelet の追加引数を指定します (オプション)。
- `-KubeProxyExtraArgs` - kube-proxy の追加引数を指定します (オプション)。
- `-APIServerEndpoint` — Amazon EKS クラスター API サーバーエンドポイントを指定します (オプション)。 `-Base64ClusterCA` と共に使用した場合のみ有効となります。 `Get-EKSCluster` の呼び出しをバイパスします。
- `-Base64ClusterCA` — Base64 でエンコードされたクラスターの CA の内容を指定します (オプション)。 `-APIServerEndpoint` と共に使用した場合のみ有効となります。 `Get-EKSCluster` の呼び出しをバイパスします。
- `-DNSClusterIP` — クラスター内の DNS クエリに使用する IP アドレスを上書きします (オプション)。プライマリインターフェイスの IP アドレスに基づく `10.100.0.10` または `172.20.0.10` のデフォルトです。
- `-ServiceCIDR` — クラスターサービスの宛先となる Kubernetes サービスの IP アドレス範囲をオーバーライドします。プライマリインターフェイスの IP アドレスに基づく `172.20.0.0/16` または `10.100.0.0/16` のデフォルトです。
- `-ExcludedSnatCIDRs` — 送信元ネットワークアドレス変換 (SNAT) から除外する IPv4 CIDR のリストです。つまり、VPC アドレスで呼び出し可能なポッドプライベート IP は、アウトバウンドトラフィック用のインスタンス ENI のプライマリ IPv4 アドレスの IP アドレスに変換されません。デフォルトでは、Amazon EKS Windows ノードの VPC の IPv4 CIDR が追加されます。このパラメータに CIDR を指定すると、指定した CIDR も除外されます。詳細については、「[the section called “アウトバウンドトラフィック”](#)」を参照してください。

コマンドラインパラメータに加えて、いくつかの環境変数パラメータを指定することもできます。コマンドラインパラメータを指定する場合、そのパラメータはそれぞれの環境変数よりも優先されます。ブートストラップスクリプトはマシンスコープの変数のみを読み取るため、環境変数はマシン (またはシステム) スコープとして定義する必要があります。

このスクリプトでは、次の環境変数が考慮されます。

- `SERVICE_IPV4_CIDR` — 定義については、`ServiceCIDR` コマンドラインパラメータを参照してください。
- `EXCLUDED_SNAT_CIDRS` — コンマ区切りの文字列にする必要があります。定義については `ExcludedSnatCIDRs` コマンドラインパラメータを参照してください。

## gMSA 認証サポート

Amazon EKS Windows Pod では、さまざまなタイプのグループ管理サービスアカウント (gMSA) 認証が可能です。

- Amazon EKS は認証用の Active Directory ドメインアイデンティティをサポートしています。ドメイン参加型 gMSA の詳細については、AWS ブログの「[Windows Authentication on Amazon EKS Windows pods](#)」を参照してください。
- Amazon EKS では、非ドメイン参加型 Windows ノードがポータブルなユーザーアイデンティティを使用して gMSA 認証情報を取得できるようにするプラグインが提供されています。ドメインレス gMSA の詳細については、AWS ブログの「[Domainless Windows Authentication for Amazon EKS Windows pods](#)」を参照してください。

## キャッシュされたコンテナイメージ

Amazon EKS Windows 最適化 AMI には、containerd のランタイムのためにキャッシュされたコンテナイメージが含まれます。コンテナイメージは、Amazon 管理ビルドコンポーネントを使用してカスタム AMI を構築するときにキャッシュされます。詳細については、「[the section called “Amazon 管理ビルドコンポーネントを使用する”](#)」を参照してください。

containerd ランタイムのキャッシュされたコンテナイメージは以下のとおりです。

- [amazonaws.com/eks/pause-windows](https://amazonaws.com/eks/pause-windows)
- [mcr.microsoft.com/windows/nanoserver](https://mcr.microsoft.com/windows/nanoserver)
- [mcr.microsoft.com/windows/servercore](https://mcr.microsoft.com/windows/servercore)

## 詳細情報

Amazon EKS 最適化 Windows AMI の使用に関する詳細については、以下のセクションを参照してください。

- Amazon EKS 最適化アクセラレーテッド Windows AMI でのワークロードの実行に関する詳細については、「[the section called “Windows GPU AMI を実行する”](#)」を参照してください。
- Windows をマネージド型ノードグループと使用するには、「[the section called “マネージド型ノードグループ”](#)」を参照してください。
- セルフマネージド型 Windows ノードを起動するには、「[the section called “Windows”](#)」を参照してください。

- バージョンについては、「[the section called “バージョン情報を取得する”](#)」を参照してください。
- Amazon EKS 最適化 Windows AMI の最新の ID を取得するには、「[the section called “最新の ID を取得する”](#)」を参照してください。
- Amazon EC2 Image Builder を使用して、カスタムの Amazon EKS 最適化 Windows AMI を作成するには、「[the section called “カスタムビルド”](#)」を参照してください。
- ベストプラクティスについては、「EKS ベストプラクティスガイド」の「[Amazon EKS 最適化 Windows AMI の管理](#)」を参照してください。

## eksctl を使用してセルフマネージド Windows Server 2022 ノードを作成する

次の test-windows-2022.yaml は、セルフマネージド Windows Server 2022 ノードを作成するためのリファレンスとして使用できます。各#####は独自の値に置き換えます。

### Note

セルフマネージド Windows Server 2022 ノードを実行するには、eksctl バージョン [0.116.0](#) 以降を使用する必要があります。

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

metadata:
 name: windows-2022-cluster
 region: region-code
 version: '1.32'

nodeGroups:
 - name: windows-ng
 instanceType: m5.2xlarge
 amiFamily: WindowsServer2022FullContainer
 volumeSize: 100
 minSize: 2
 maxSize: 3
 - name: linux-ng
 amiFamily: AmazonLinux2
 minSize: 2
 maxSize: 3
```

ノードグループは、次のコマンドを使用して作成できます。

```
eksctl create cluster -f test-windows-2022.yaml
```

## Windows AMI バージョンに関する情報を取得する

このトピックでは、Amazon EKS 最適化 Windows AMI のバージョンと、それに対応する [kubelet](#)、[containerd](#)、および [csi-proxy](#) のバージョンを示します。

AMI ID を含む、Amazon EKS 最適化 AMI のメタデータでは、各バリエーションをプログラマ的に取得することができます。詳細については、「[the section called “最新の ID を取得する”](#)」を参照してください。

各 AMI は次の形式により、Kubernetes のバージョンならびに AMI のリリース日によってバージョン付けされています。

```
k8s_major_version.k8s_minor_version-release_date
```

### Note

Amazon EKS マネージド型ノードグループは、Windows AMI の 2022 年 11 月以降のリリースをサポートしています。

この特定のドキュメントページへのすべてのソースファイルの変更の通知を受け取るには、RSS リーダーを使用して次の URL をサブスクライブできます。

```
https://github.com/awsdocs/amazon-eks-user-guide/commits/mainline/latest/ug/nodes/eks-ami-versions-windows.adoc.atom
```

## Amazon EKS 最適化 Windows Server 2022 Core AMI

次の表は、Amazon EKS 最適化 Windows Server 2022 Core AMI の現在および以前のバージョンを示しています。



## Example

## Kubernetes バージョン 1.33

| AMI のバージョン      | kubelet バージョン | containerd バージョン | csi-proxy バージョン | リリースノート |
|-----------------|---------------|------------------|-----------------|---------|
| 1.33-2025.05.17 | 1.33.1        | 1.7.27           | 1.2.1           |         |

## Kubernetes バージョン 1.32

| AMI のバージョン      | kubelet バージョン | containerd バージョン | csi-proxy バージョン | リリースノート |
|-----------------|---------------|------------------|-----------------|---------|
| 1.32-2025.05.17 | 1.32.5        | 1.7.20           | 1.2.1           |         |
| 1.32-2025.04.14 | 1.32.1        | 1.7.20           | 1.1.3           |         |
| 1.32-2025.03.14 | 1.32.1        | 1.7.20           | 1.1.3           |         |
| 1.32-2025.02.18 | 1.32.1        | 1.7.20           | 1.1.3           |         |
| 1.32-2025.01.15 | 1.32.0        | 1.7.20           | 1.1.3           |         |

## Kubernetes バージョン 1.31

| AMI のバージョン      | kubelet バージョン | containerd バージョン | csi-proxy バージョン | リリースノート |
|-----------------|---------------|------------------|-----------------|---------|
| 1.31-2025.05.17 | 1.31.9        | 1.7.20           | 1.2.1           |         |

| AMI のバージョン      | kubelet バージョン | containerd バージョン | csi-proxy バージョン | リリースノート                   |
|-----------------|---------------|------------------|-----------------|---------------------------|
| 1.31-2025.04.14 | 1.31.5        | 1.7.20           | 1.1.3           |                           |
| 1.31-2025.03.14 | 1.31.5        | 1.7.20           | 1.1.3           |                           |
| 1.31-2025.02.15 | 1.31.5        | 1.7.20           | 1.1.3           |                           |
| 1.31-2025.01.15 | 1.31.4        | 1.7.20           | 1.1.3           |                           |
| 1.31-2025.01.01 | 1.31.4        | 1.7.20           | 1.1.3           | CVE-2024-9042 のパッチが含まれます。 |
| 1.31-2024.12.13 | 1.31.3        | 1.7.20           | 1.1.3           |                           |
| 1.31-2024.11.12 | 1.31.1        | 1.7.20           | 1.1.3           |                           |
| 1.31-2024.10.08 | 1.31.1        | 1.7.20           | 1.1.3           |                           |
| 1.31-2024.10.01 | 1.31.1        | 1.7.20           | 1.1.3           |                           |
| 1.31-2024.09.10 | 1.31.0        | 1.7.20           | 1.1.3           |                           |

## Kubernetes バージョン 1.30

| AMI のバージョン      | kubelet バージョン | containerd バージョン | csi-proxy バージョン | リリースノート                           |
|-----------------|---------------|------------------|-----------------|-----------------------------------|
| 1.30-2025.05.17 | 1.30.13       | 1.7.20           | 1.2.1           |                                   |
| 1.30-2025.04.14 | 1.30.9        | 1.7.20           | 1.1.3           |                                   |
| 1.30-2025.03.14 | 1.30.9        | 1.7.20           | 1.1.3           | containerd を 1.7.20 にアップグレードしました。 |
| 1.30-2025.02.15 | 1.30.9        | 1.7.14           | 1.1.3           |                                   |
| 1.30-2025.01.15 | 1.30.8        | 1.7.14           | 1.1.3           |                                   |
| 1.30-2025.01.01 | 1.30.8        | 1.7.14           | 1.1.3           | CVE-2024-9042 のパッチが含まれます。         |
| 1.30-2024.12.11 | 1.30.7        | 1.7.14           | 1.1.3           |                                   |
| 1.30-2024.11.12 | 1.30.4        | 1.7.14           | 1.1.3           |                                   |
| 1.30-2024.10.08 | 1.30.4        | 1.7.14           | 1.1.3           |                                   |
| 1.30-2024.09.10 | 1.30.2        | 1.7.14           | 1.1.3           |                                   |

| AMI のバージョン      | kubelet バージョン | containerd バージョン | csi-proxy バージョン | リリースノート                           |
|-----------------|---------------|------------------|-----------------|-----------------------------------|
| 1.30-2024.08.13 | 1.30.2        | 1.7.14           | 1.1.3           |                                   |
| 1.30-2024.07.10 | 1.30.2        | 1.7.14           | 1.1.2           | CVE-2024-5321 のパッチが含まれます。         |
| 1.30-2024.06.17 | 1.30.0        | 1.7.14           | 1.1.2           | containerd を 1.7.14 にアップグレードしました。 |
| 1.30-2024.05.15 | 1.30.0        | 1.6.28           | 1.1.2           |                                   |

## Kubernetes バージョン 1.29

| AMI のバージョン      | kubelet バージョン | containerd バージョン | csi-proxy バージョン | リリースノート                           |
|-----------------|---------------|------------------|-----------------|-----------------------------------|
| 1.29-2025.05.17 | 1.29.15       | 1.7.20           | 1.2.1           |                                   |
| 1.29-2025.04.14 | 1.29.13       | 1.7.20           | 1.1.3           |                                   |
| 1.29-2025.03.14 | 1.29.13       | 1.7.20           | 1.1.3           | containerd を 1.7.20 にアップグレードしました。 |
| 1.29-2025.02.15 | 1.29.13       | 1.7.14           | 1.1.3           |                                   |

| AMI のバージョン      | kubelet バージョン | containerd バージョン | csi-proxy バージョン | リリースノート                                                          |
|-----------------|---------------|------------------|-----------------|------------------------------------------------------------------|
| 1.29-2025.01.15 | 1.29.12       | 1.7.14           | 1.1.3           |                                                                  |
| 1.29-2025.01.01 | 1.29.12       | 1.7.14           | 1.1.3           | CVE-2024-9042 のパッチが含まれます。                                        |
| 1.29-2024.12.11 | 1.29.10       | 1.7.14           | 1.1.3           |                                                                  |
| 1.29-2024.11.12 | 1.29.8        | 1.7.14           | 1.1.3           |                                                                  |
| 1.29-2024.10.08 | 1.29.8        | 1.7.14           | 1.1.3           |                                                                  |
| 1.29-2024.09.10 | 1.29.6        | 1.7.14           | 1.1.3           |                                                                  |
| 1.29-2024.08.13 | 1.29.6        | 1.7.14           | 1.1.3           |                                                                  |
| 1.29-2024.07.10 | 1.29.6        | 1.7.11           | 1.1.2           | CVE-2024-5321 のパッチが含まれます。                                        |
| 1.29-2024.06.17 | 1.29.3        | 1.7.11           | 1.1.2           |                                                                  |
| 1.29-2024.05.15 | 1.29.3        | 1.7.11           | 1.1.2           | containerd を 1.7.11 にアップグレードしました。 kubelet を 1.29.3 にアップグレードしました。 |

| AMI のバージョン      | kubelet バージョン | containerd バージョン | csi-proxy バージョン | リリースノート                                                                         |
|-----------------|---------------|------------------|-----------------|---------------------------------------------------------------------------------|
| 1.29-2024.04.09 | 1.29.0        | 1.6.28           | 1.1.2           | containerd を 1.6.28 にアップグレードしました。golang 1.22.1 を使用して CNI と csi-proxy が再構築されました。 |
| 1.29-2024.03.12 | 1.29.0        | 1.6.25           | 1.1.2           |                                                                                 |
| 1.29-2024.02.13 | 1.29.0        | 1.6.25           | 1.1.2           |                                                                                 |
| 1.29-2024.02.06 | 1.29.0        | 1.6.25           | 1.1.2           | 一時停止画像が kubelet ガベージコレクション処理によって誤って削除されてしまうバグを修正しました。                           |

| AMI のバージョン      | kubelet バージョン | containerd バージョン | csi-proxy バージョン | リリースノート                                                                                                                                                                       |
|-----------------|---------------|------------------|-----------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 1.29-2024.01.11 | 1.29.0        | 1.6.18           | 1.1.2           | Windows Server 2022 Core AMI のスタンドアロン Windows Update <a href="#">KB5034439</a> は除外しました。KB は、Amazon EKS 最適化 Windows AMI に含まれていない、個別の WinRE パーティションのある Windows インストールにのみ適用されます。 |

## Kubernetes バージョン 1.28

| AMI のバージョン      | kubelet バージョン | containerd バージョン | csi-proxy バージョン | リリースノート               |
|-----------------|---------------|------------------|-----------------|-----------------------|
| 1.28-2025.05.17 | 1.28.15       | 1.7.20           | 1.2.1           |                       |
| 1.28-2025.04.14 | 1.28.15       | 1.7.20           | 1.1.3           |                       |
| 1.28-2025.03.14 | 1.28.15       | 1.7.20           | 1.1.3           | containerd を 1.7.20 に |

| AMI のバージョン      | kubelet バージョン | containerd バージョン | csi-proxy バージョン | リリースノート                           |
|-----------------|---------------|------------------|-----------------|-----------------------------------|
|                 |               |                  |                 | アップグレードしました。                      |
| 1.28-2025.02.15 | 1.28.15       | 1.7.14           | 1.1.3           |                                   |
| 1.28-2025.01.15 | 1.28.15       | 1.7.14           | 1.1.3           |                                   |
| 1.28-2025-01-01 | 1.28.15       | 1.7.14           | 1.1.3           | CVE-2024-9042 のパッチが含まれます。         |
| 1.28-2024.12.11 | 1.28.15       | 1.7.14           | 1.1.3           |                                   |
| 1.28-2024.11.12 | 1.28.13       | 1.7.14           | 1.1.3           |                                   |
| 1.28-2024.10.08 | 1.28.13       | 1.7.14           | 1.1.3           |                                   |
| 1.28-2024.09.10 | 1.28.11       | 1.7.14           | 1.1.3           |                                   |
| 1.28-2024.08.13 | 1.28.11       | 1.7.14           | 1.1.3           |                                   |
| 1.28-2024.07.10 | 1.28.11       | 1.7.11           | 1.1.2           | CVE-2024-5321 のパッチが含まれます。         |
| 1.28-2024.06.17 | 1.28.8        | 1.7.11           | 1.1.2           | containerd を 1.7.11 にアップグレードしました。 |



| AMI のバージョン      | kubelet バージョン | containerd バージョン | csi-proxy バージョン | リリースノート                                                                         |
|-----------------|---------------|------------------|-----------------|---------------------------------------------------------------------------------|
| 1.28-2024.05.14 | 1.28.8        | 1.6.28           | 1.1.2           | containerd を 1.6.28 にアップグレードしました。kubelet を 1.28.8 にアップグレードしました。                 |
| 1.28-2024.04.09 | 1.28.5        | 1.6.25           | 1.1.2           | containerd を 1.6.25 にアップグレードしました。golang 1.22.1 を使用して CNI と csi-proxy が再構築されました。 |
| 1.28-2024.03.12 | 1.28.5        | 1.6.18           | 1.1.2           |                                                                                 |
| 1.28-2024.02.13 | 1.28.5        | 1.6.18           | 1.1.2           |                                                                                 |

| AMI のバージョン      | kubelet バージョン | containerd バージョン | csi-proxy バージョン | リリースノート                                                                                                                                                                       |
|-----------------|---------------|------------------|-----------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 1.28-2024.01.11 | 1.28.5        | 1.6.18           | 1.1.2           | Windows Server 2022 Core AMI のスタンドアロン Windows Update <a href="#">KB5034439</a> は除外しました。KB は、Amazon EKS 最適化 Windows AMI に含まれていない、個別の WinRE パーティションのある Windows インストールにのみ適用されます。 |
| 1.28-2023.12.12 | 1.28.3        | 1.6.18           | 1.1.2           |                                                                                                                                                                               |
| 1.28-2023.11.14 | 1.28.3        | 1.6.18           | 1.1.2           | CVE-2023-5528 のパッチが含まれます。                                                                                                                                                     |

| AMI のバージョン      | kubelet バージョン | containerd バージョン | csi-proxy バージョン | リリースノート                                                                                                                         |
|-----------------|---------------|------------------|-----------------|---------------------------------------------------------------------------------------------------------------------------------|
| 1.28-2023.10.19 | 1.28.2        | 1.6.18           | 1.1.2           | containerd を 1.6.18 にアップグレードしました。<br>新しい <a href="#">ブートストラップスクリプト環境変数</a> (SERVICE_I PV4_CIDR と EXCLUDED_SNAT_CIDR S )を追加しました。 |
| 1.28-2023-09.27 | 1.28.2        | 1.6.6            | 1.1.2           | kubelet の <a href="#">セキュリティ勧告</a> を修正しました。                                                                                     |
| 1.28-2023.09.12 | 1.28.1        | 1.6.6            | 1.1.2           |                                                                                                                                 |

## Kubernetes バージョン 1.27

| AMI のバージョン      | kubelet バージョン | containerd バージョン | csi-proxy バージョン | リリースノート |
|-----------------|---------------|------------------|-----------------|---------|
| 1.27-2025.05.17 | 1.27.16       | 1.7.20           | 1.2.1           |         |
| 1.27-2025.04.14 | 1.27.16       | 1.7.20           | 1.1.3           |         |

| AMI のバージョン      | kubelet バージョン | containerd バージョン | csi-proxy バージョン | リリースノート                           |
|-----------------|---------------|------------------|-----------------|-----------------------------------|
| 1.27-2025.03.14 | 1.27.16       | 1.7.20           | 1.1.3           | containerd を 1.7.20 にアップグレードしました。 |
| 1.27-2025.02.15 | 1.27.16       | 1.7.14           | 1.1.3           |                                   |
| 1.27-2025.01.15 | 1.27.16       | 1.7.14           | 1.1.3           |                                   |
| 1.27-2025.01.01 | 1.27.16       | 1.7.14           | 1.1.3           | CVE-2024-9042 のパッチが含まれます。         |
| 1.27-2024.12.11 | 1.27.16       | 1.7.14           | 1.1.3           |                                   |
| 1.27-2024.11.12 | 1.27.16       | 1.7.14           | 1.1.3           |                                   |
| 1.27-2024.10.08 | 1.27.16       | 1.7.14           | 1.1.3           |                                   |
| 1.27-2024.09.10 | 1.27.15       | 1.7.14           | 1.1.3           |                                   |
| 1.27-2024.08.13 | 1.27.15       | 1.7.14           | 1.1.3           |                                   |
| 1.27-2024.07.10 | 1.27.15       | 1.7.11           | 1.1.2           | CVE-2024-5321 のパッチが含まれます。         |

| AMI のバージョン      | kubelet バージョン | containerd バージョン | csi-proxy バージョン | リリースノート                                                                         |
|-----------------|---------------|------------------|-----------------|---------------------------------------------------------------------------------|
| 1.27-2024.06.17 | 1.27.12       | 1.7.11           | 1.1.2           | containerd を 1.7.11 にアップグレードしました。                                               |
| 1.27-2024.05.14 | 1.27.12       | 1.6.28           | 1.1.2           | containerd を 1.6.28 にアップグレードしました。kubelet を 1.27.12 にアップグレードしました。                |
| 1.27-2024.04.09 | 1.27.9        | 1.6.25           | 1.1.2           | containerd を 1.6.25 にアップグレードしました。golang 1.22.1 を使用して CNI と csi-proxy が再構築されました。 |
| 1.27-2024.03.12 | 1.27.9        | 1.6.18           | 1.1.2           |                                                                                 |
| 1.27-2024.02.13 | 1.27.9        | 1.6.18           | 1.1.2           |                                                                                 |

| AMI のバージョン      | kubelet バージョン | containerd バージョン | csi-proxy バージョン | リリースノート                                                                                                                                                                       |
|-----------------|---------------|------------------|-----------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 1.27-2024.01.11 | 1.27.9        | 1.6.18           | 1.1.2           | Windows Server 2022 Core AMI のスタンドアロン Windows Update <a href="#">KB5034439</a> は除外しました。KB は、Amazon EKS 最適化 Windows AMI に含まれていない、個別の WinRE パーティションのある Windows インストールにのみ適用されます。 |
| 1.27-2023.12.12 | 1.27.7        | 1.6.18           | 1.1.2           |                                                                                                                                                                               |
| 1.27-2023.11.14 | 1.27.7        | 1.6.18           | 1.1.2           | CVE-2023-5528 のパッチが含まれます。                                                                                                                                                     |

| AMI のバージョン      | kubelet バージョン | containerd バージョン | csi-proxy バージョン | リリースノート                                                                                                                                     |
|-----------------|---------------|------------------|-----------------|---------------------------------------------------------------------------------------------------------------------------------------------|
| 1.27-2023.10.19 | 1.27.6        | 1.6.18           | 1.1.2           | containerd を 1.6.18 にアップグレードしました。<br>新しい <a href="#">ブートストラップスクリプト環境変数</a> (SERVICE_IPV4_CIDR と EXCLUDED_SNAT_CIDRS )を追加しました。               |
| 1.27-2023-09.27 | 1.27.6        | 1.6.6            | 1.1.2           | kubelet の <a href="#">セキュリティ勧告</a> を修正しました。                                                                                                 |
| 1.27-2023.09.12 | 1.27.4        | 1.6.6            | 1.1.2           | Kubernetes API サーバーから Pod IP アドレスを取得する Kubernetes コネクタバイナリを使用するように、Amazon VPC CNI プラグインをアップグレードしました。 <a href="#">プルリクエスト #100</a> をマージしました。 |

| AMI のバージョン      | kubelet バージョン | containerd バージョン | csi-proxy バージョン | リリースノート                                                             |
|-----------------|---------------|------------------|-----------------|---------------------------------------------------------------------|
| 1.27-2023.08.17 | 1.27.4        | 1.6.6            | 1.1.2           | CVE-2023-3676 、 CVE-2023-3893 、 および CVE-2023-3955 のパッチが含まれます。       |
| 1.27-2023.08.08 | 1.27.3        | 1.6.6            | 1.1.1           |                                                                     |
| 1.27-2023.07.11 | 1.27.3        | 1.6.6            | 1.1.1           |                                                                     |
| 1.27-2023.06.20 | 1.27.1        | 1.6.6            | 1.1.1           | DNS サフィックス検索リストに正しく入力されない問題を解決しました。                                 |
| 1.27-2023.06.14 | 1.27.1        | 1.6.6            | 1.1.1           | CNI のホストポートマッピングのサポートが追加されました。プルリクエスト <a href="#">#93</a> をマージしました。 |



| AMI のバージョン      | kubelet バージョン | containerd バージョン | csi-proxy バージョン | リリースノート                                                                                             |
|-----------------|---------------|------------------|-----------------|-----------------------------------------------------------------------------------------------------|
| 1.27-2023.06.06 | 1.27.1        | 1.6.6            | 1.1.1           | ノードがプライベート Amazon ECR イメージの取り込みに失敗する原因となっていた containerd s-roadmap <a href="#">問題 #2042</a> を修正しました。 |
| 1.27-2023.05.17 | 1.27.1        | 1.6.6            | 1.1.1           |                                                                                                     |

## Kubernetes バージョン 1.26

| AMI のバージョン      | kubelet バージョン | containerd バージョン | csi-proxy バージョン | リリースノート                           |
|-----------------|---------------|------------------|-----------------|-----------------------------------|
| 1.26-2025.05.17 | 1.26.15       | 1.7.20           | 1.2.1           |                                   |
| 1.26-2025.04.14 | 1.26.15       | 1.7.20           | 1.1.3           |                                   |
| 1.26-2025.03.14 | 1.26.15       | 1.7.20           | 1.1.3           | containerd を 1.7.20 にアップグレードしました。 |
| 1.26-2025.02.15 | 1.26.15       | 1.7.14           | 1.1.3           |                                   |

| AMI のバージョン      | kubelet バージョン | containerd バージョン | csi-proxy バージョン | リリースノート                                                          |
|-----------------|---------------|------------------|-----------------|------------------------------------------------------------------|
| 1.26-2025.01.15 | 1.26.15       | 1.7.14           | 1.1.3           |                                                                  |
| 1.26-2024.12.11 | 1.26.15       | 1.7.14           | 1.1.3           |                                                                  |
| 1.26-2024.11.12 | 1.26.15       | 1.7.14           | 1.1.3           |                                                                  |
| 1.26-2024.10.08 | 1.26.15       | 1.7.14           | 1.1.3           |                                                                  |
| 1.26-2024.09.10 | 1.26.15       | 1.7.14           | 1.1.3           |                                                                  |
| 1.26-2024.08.13 | 1.26.15       | 1.7.14           | 1.1.3           |                                                                  |
| 1.26-2024.07.10 | 1.26.15       | 1.7.11           | 1.1.2           | CVE-2024-5321 のパッチが含まれます。                                        |
| 1.26-2024.06.17 | 1.26.15       | 1.7.11           | 1.1.2           | containerd を 1.7.11 にアップグレードしました。                                |
| 1.26-2024.05.14 | 1.26.15       | 1.6.28           | 1.1.2           | containerd を 1.6.28 にアップグレードしました。kubelet を 1.26.15 にアップグレードしました。 |

| AMI のバージョン      | kubelet バージョン | containerd バージョン | csi-proxy バージョン | リリースノート                                                                         |
|-----------------|---------------|------------------|-----------------|---------------------------------------------------------------------------------|
| 1.26-2024.04.09 | 1.26.12       | 1.6.25           | 1.1.2           | containerd を 1.6.25 にアップグレードしました。golang 1.22.1 を使用して CNI と csi-proxy が再構築されました。 |
| 1.26-2024.03.12 | 1.26.12       | 1.6.18           | 1.1.2           |                                                                                 |
| 1.26-2024.02.13 | 1.26.12       | 1.6.18           | 1.1.2           |                                                                                 |

| AMI のバージョン      | kubelet バージョン | containerd バージョン | csi-proxy バージョン | リリースノート                                                                                                                                                                       |
|-----------------|---------------|------------------|-----------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 1.26-2024.01.11 | 1.26.12       | 1.6.18           | 1.1.2           | Windows Server 2022 Core AMI のスタンドアロン Windows Update <a href="#">KB5034439</a> は除外しました。KB は、Amazon EKS 最適化 Windows AMI に含まれていない、個別の WinRE パーティションのある Windows インストールにのみ適用されます。 |
| 1.26-2023.12.12 | 1.26.10       | 1.6.18           | 1.1.2           |                                                                                                                                                                               |
| 1.26-2023.11.14 | 1.26.10       | 1.6.18           | 1.1.2           | CVE-2023-5528 のパッチが含まれます。                                                                                                                                                     |

| AMI のバージョン      | kubelet バージョン | containerd バージョン | csi-proxy バージョン | リリースノート                                                                                                                                                     |
|-----------------|---------------|------------------|-----------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 1.26-2023.10.19 | 1.26.9        | 1.6.18           | 1.1.2           | containerd を 1.6.18 にアップグレードしました。kubelet を 1.26.9 にアップグレードしました。<br><a href="#">新しいブートストラップスクリプト環境変数 (SERVICE_IPV4_CIDR と EXCLUDED_SNAT_CIDRS )</a> を追加しました。 |
| 1.26-2023.09.12 | 1.26.7        | 1.6.6            | 1.1.2           | Kubernetes API サーバーから Pod IP アドレスを取得する Kubernetes コネクタバイナリを使用するように、Amazon VPC CNI プラグインをアップグレードしました。 <a href="#">プルリクエスト #100</a> をマージしました。                 |

| AMI のバージョン      | kubelet バージョン | containerd バージョン | csi-proxy バージョン | リリースノート                                                                                              |
|-----------------|---------------|------------------|-----------------|------------------------------------------------------------------------------------------------------|
| 1.26-2023.08.17 | 1.26.7        | 1.6.6            | 1.1.2           | CVE-2023-3676 、 CVE-2023-3893 、 および CVE-2023-3955 のパッチが含まれます。                                        |
| 1.26-2023.08.08 | 1.26.6        | 1.6.6            | 1.1.1           |                                                                                                      |
| 1.26-2023.07.11 | 1.26.6        | 1.6.6            | 1.1.1           |                                                                                                      |
| 1.26-2023.06.20 | 1.26.4        | 1.6.6            | 1.1.1           | DNS サフィックス検索リストに正しく入力されない問題を解決しました。                                                                  |
| 1.26-2023.06.14 | 1.26.4        | 1.6.6            | 1.1.1           | Kubernetes を 1.26.4 にアップグレードしました。CNI のホストポートマッピングのサポートが追加されました。プルリクエスト <a href="#">#93</a> をマージしました。 |

| AMI のバージョン      | kubelet バージョン | containerd バージョン | csi-proxy バージョン | リリースノート                                                                                                                               |
|-----------------|---------------|------------------|-----------------|---------------------------------------------------------------------------------------------------------------------------------------|
| 1.26-2023.05.09 | 1.26.2        | 1.6.6            | 1.1.1           | ノードの再起動後にポッドでネットワーク接続の問題 <a href="#">#1126</a> を引き起こすバグを修正しました。新しい <a href="#">ブートストラップスクリプトの設定パラメータ</a> (ExcludedSnatCIDs ) を導入しました。 |
| 1.26-2023.04.26 | 1.26.2        | 1.6.6            | 1.1.1           |                                                                                                                                       |
| 1.26-2023.04.11 | 1.26.2        | 1.6.6            | 1.1.1           | サービスクラッシュ時の kubelet および kube-proxy の回復メカニズムを追加しました。                                                                                   |
| 1.26-2023.03.24 | 1.26.2        | 1.6.6            | 1.1.1           |                                                                                                                                       |

## Amazon EKS 最適化 Windows Server 2022 Full AMI

次の表は、Amazon EKS 最適化 Windows Server 2022 Full AMI の現在および以前のバージョンを示しています。

## Example

## Kubernetes バージョン 1.33

| AMI のバージョン      | kubelet バージョン | containerd バージョン | csi-proxy バージョン | リリースノート |
|-----------------|---------------|------------------|-----------------|---------|
| 1.33-2025.05.17 | 1.33.1        | 1.7.27           | 1.2.1           |         |

## Kubernetes バージョン 1.32

| AMI のバージョン      | kubelet バージョン | containerd バージョン | csi-proxy バージョン | リリースノート |
|-----------------|---------------|------------------|-----------------|---------|
| 1.32-2025.05.17 | 1.32.5        | 1.7.20           | 1.2.1           |         |
| 1.32-2025.04.14 | 1.32.1        | 1.7.20           | 1.1.3           |         |
| 1.32-2025.03.14 | 1.32.1        | 1.7.20           | 1.1.3           |         |
| 1.32-2025.02.18 | 1.32.1        | 1.7.20           | 1.1.3           |         |
| 1.32-2025.01.01 | 1.32.0        | 1.7.20           | 1.1.3           |         |

## Kubernetes バージョン 1.31

| AMI のバージョン      | kubelet バージョン | containerd バージョン | csi-proxy バージョン | リリースノート |
|-----------------|---------------|------------------|-----------------|---------|
| 1.31-2025.05.17 | 1.31.9        | 1.7.20           | 1.2.1           |         |



| AMI のバージョン      | kubelet バージョン | containerd バージョン | csi-proxy バージョン | リリースノート                   |
|-----------------|---------------|------------------|-----------------|---------------------------|
| 1.31-2025.04.14 | 1.31.5        | 1.7.20           | 1.1.3           |                           |
| 1.31-2025.03.14 | 1.31.5        | 1.7.20           | 1.1.3           |                           |
| 1.31-2025.02.15 | 1.31.5        | 1.7.20           | 1.1.3           |                           |
| 1.31-2025.01.15 | 1.31.4        | 1.7.20           | 1.1.3           |                           |
| 1.31-2025.01.01 | 1.31.4        | 1.7.20           | 1.1.3           | CVE-2024-9042 のパッチが含まれます。 |
| 1.31-2024.12.13 | 1.31.3        | 1.7.20           | 1.1.3           |                           |
| 1.31-2024.11.12 | 1.31.1        | 1.7.20           | 1.1.3           |                           |
| 1.31-2024.10.08 | 1.31.1        | 1.7.20           | 1.1.3           |                           |
| 1.31-2024.10.01 | 1.31.1        | 1.7.20           | 1.1.3           |                           |
| 1.31-2024.09.10 | 1.31.0        | 1.7.20           | 1.1.3           |                           |

## Kubernetes バージョン 1.30

| AMI のバージョン      | kubelet バージョン | containerd バージョン | csi-proxy バージョン | リリースノート                            |
|-----------------|---------------|------------------|-----------------|------------------------------------|
| 1.30-2025.05.17 | 1.30.13       | 1.7.20           | 1.2.1           |                                    |
| 1.30-2025.04.14 | 1.30.9        | 1.7.20           | 1.1.3           |                                    |
| 1.30-2025.03.14 | 1.30.9        | 1.7.20           | 1.1.3           | contianer d を 1.7.20 にアップグレードしました。 |
| 1.30-2025.02.15 | 1.30.9        | 1.7.14           | 1.1.3           |                                    |
| 1.30-2025.01.15 | 1.30.8        | 1.7.14           | 1.1.3           |                                    |
| 1.30-2025.01.01 | 1.30.8        | 1.7.14           | 1.1.3           | CVE-2024-9042 のパッチが含まれます。          |
| 1.30-2024.12.11 | 1.30.7        | 1.7.14           | 1.1.3           |                                    |
| 1.30-2024.11.12 | 1.30.4        | 1.7.14           | 1.1.3           |                                    |
| 1.30-2024.10.08 | 1.30.4        | 1.7.14           | 1.1.3           |                                    |
| 1.30-2024.09.10 | 1.30.2        | 1.7.14           | 1.1.3           |                                    |

| AMI のバージョン      | kubelet バージョン | containerd バージョン | csi-proxy バージョン | リリースノート                           |
|-----------------|---------------|------------------|-----------------|-----------------------------------|
| 1.30-2024.08.13 | 1.30.2        | 1.7.14           | 1.1.3           |                                   |
| 1.30-2024.07.10 | 1.30.2        | 1.7.14           | 1.1.2           | CVE-2024-5321 のパッチが含まれます。         |
| 1.30-2024.06.17 | 1.30.0        | 1.7.14           | 1.1.2           | containerd を 1.7.14 にアップグレードしました。 |
| 1.30-2024.05.15 | 1.30.0        | 1.6.28           | 1.1.2           |                                   |

## Kubernetes バージョン 1.29

| AMI のバージョン      | kubelet バージョン | containerd バージョン | csi-proxy バージョン | リリースノート                           |
|-----------------|---------------|------------------|-----------------|-----------------------------------|
| 1.29-2025.05.17 | 1.29.15       | 1.7.20           | 1.2.1           |                                   |
| 1.29-2025.04.14 | 1.29.13       | 1.7.20           | 1.1.3           |                                   |
| 1.29-2025.03.14 | 1.29.13       | 1.7.20           | 1.1.3           | containerd を 1.7.20 にアップグレードしました。 |
| 1.29-2025.02.15 | 1.29.13       | 1.7.14           | 1.1.3           |                                   |

| AMI のバージョン      | kubelet バージョン | containerd バージョン | csi-proxy バージョン | リリースノート                                                         |
|-----------------|---------------|------------------|-----------------|-----------------------------------------------------------------|
| 1.29-2025.01.15 | 1.29.12       | 1.7.14           | 1.1.3           |                                                                 |
| 1.29-2025.01.01 | 1.29.12       | 1.7.14           | 1.1.3           | CVE-2024-9042 のパッチが含まれます。                                       |
| 1.29-2024.12.11 | 1.29.10       | 1.7.14           | 1.1.3           |                                                                 |
| 1.29-2024.11.12 | 1.29.8        | 1.7.14           | 1.1.3           |                                                                 |
| 1.29-2024.10.08 | 1.29.8        | 1.7.14           | 1.1.3           |                                                                 |
| 1.29-2024.09.10 | 1.29.6        | 1.7.14           | 1.1.3           |                                                                 |
| 1.29-2024.08.13 | 1.29.6        | 1.7.14           | 1.1.3           |                                                                 |
| 1.29-2024.07.10 | 1.29.6        | 1.7.11           | 1.1.2           | CVE-2024-5321 のパッチが含まれます。                                       |
| 1.29-2024.06.17 | 1.29.3        | 1.7.11           | 1.1.2           |                                                                 |
| 1.29-2024.05.15 | 1.29.3        | 1.7.11           | 1.1.2           | containerd を 1.7.11 にアップグレードしました。kubelet を 1.29.3 にアップグレードしました。 |

| AMI のバージョン      | kubelet バージョン | containerd バージョン | csi-proxy バージョン | リリースノート                                                                         |
|-----------------|---------------|------------------|-----------------|---------------------------------------------------------------------------------|
| 1.29-2024.04.09 | 1.29.0        | 1.6.28           | 1.1.2           | containerd を 1.6.28 にアップグレードしました。golang 1.22.1 を使用して CNI と csi-proxy が再構築されました。 |
| 1.29-2024.03.12 | 1.29.0        | 1.6.25           | 1.1.2           |                                                                                 |
| 1.29-2024.02.13 | 1.29.0        | 1.6.25           | 1.1.2           |                                                                                 |
| 1.29-2024.02.06 | 1.29.0        | 1.6.25           | 1.1.2           | 一時停止画像が kubelet ガベージコレクション処理によって誤って削除されてしまうバグを修正しました。                           |
| 1.29-2024.01.09 | 1.29.0        | 1.6.18           | 1.1.2           |                                                                                 |

## Kubernetes バージョン 1.28

| AMI のバージョン      | kubelet バージョン | containerd バージョン | csi-proxy バージョン | リリースノート                           |
|-----------------|---------------|------------------|-----------------|-----------------------------------|
| 1.28-2025.05.17 | 1.28.15       | 1.7.20           | 1.2.1           |                                   |
| 1.28-2025.04.14 | 1.28.15       | 1.7.20           | 1.1.3           |                                   |
| 1.28-2025.03.14 | 1.28.15       | 1.7.20           | 1.1.3           | containerd を 1.7.20 にアップグレードしました。 |
| 1.28-2025.02.15 | 1.28.15       | 1.7.14           | 1.1.3           |                                   |
| 1.28-2025.01.15 | 1.28.15       | 1.7.14           | 1.1.3           |                                   |
| 1.28-2025.01.01 | 1.28.15       | 1.7.14           | 1.1.3           | CVE-2024-9042 のパッチが含まれます。         |
| 1.28-2024.12.11 | 1.28.15       | 1.7.14           | 1.1.3           |                                   |
| 1.28-2024.11.12 | 1.28.13       | 1.7.14           | 1.1.3           |                                   |
| 1.28-2024.10.08 | 1.28.13       | 1.7.14           | 1.1.3           |                                   |
| 1.28-2024.09.10 | 1.28.11       | 1.7.14           | 1.1.3           |                                   |

| AMI のバージョン      | kubelet バージョン | containerd バージョン | csi-proxy バージョン | リリースノート                                                                         |
|-----------------|---------------|------------------|-----------------|---------------------------------------------------------------------------------|
| 1.28-2024.08.13 | 1.28.11       | 1.7.14           | 1.1.3           |                                                                                 |
| 1.28-2024.07.10 | 1.28.11       | 1.7.11           | 1.1.2           | CVE-2024-5321 のパッチが含まれます。                                                       |
| 1.28-2024.06.17 | 1.28.8        | 1.7.11           | 1.1.2           | containerd を 1.7.11 にアップグレードしました。                                               |
| 1.28-2024.05.14 | 1.28.8        | 1.6.28           | 1.1.2           | containerd を 1.6.28 にアップグレードしました。kubelet を 1.28.8 にアップグレードしました。                 |
| 1.28-2024.04.09 | 1.28.5        | 1.6.25           | 1.1.2           | containerd を 1.6.25 にアップグレードしました。golang 1.22.1 を使用して CNI と csi-proxy が再構築されました。 |
| 1.28-2024.03.12 | 1.28.5        | 1.6.18           | 1.1.2           |                                                                                 |

| AMI のバージョン      | kubelet バージョン | containerd バージョン | csi-proxy バージョン | リリースノート                                                                                                                       |
|-----------------|---------------|------------------|-----------------|-------------------------------------------------------------------------------------------------------------------------------|
| 1.28-2024.02.13 | 1.28.5        | 1.6.18           | 1.1.2           |                                                                                                                               |
| 1.28-2024.01.09 | 1.28.5        | 1.6.18           | 1.1.2           |                                                                                                                               |
| 1.28-2023.12.12 | 1.28.3        | 1.6.18           | 1.1.2           |                                                                                                                               |
| 1.28-2023.11.14 | 1.28.3        | 1.6.18           | 1.1.2           | CVE-2023-5528 のパッチが含まれます。                                                                                                     |
| 1.28-2023.10.19 | 1.28.2        | 1.6.18           | 1.1.2           | containerd を 1.6.18 にアップグレードしました。<br>新しい <a href="#">ブートストラップスクリプト環境変数</a> (SERVICE_IPV4_CIDR と EXCLUDED_SNAT_CIDRS )を追加しました。 |
| 1.28-2023-09.27 | 1.28.2        | 1.6.6            | 1.1.2           | kubelet の <a href="#">セキュリティ勧告</a> を修正しました。                                                                                   |
| 1.28-2023.09.12 | 1.28.1        | 1.6.6            | 1.1.2           |                                                                                                                               |



## Kubernetes バージョン 1.27

| AMI のバージョン      | kubelet バージョン | containerd バージョン | csi-proxy バージョン | リリースノート                           |
|-----------------|---------------|------------------|-----------------|-----------------------------------|
| 1.27-2025.05.17 | 1.27.16       | 1.7.20           | 1.2.1           |                                   |
| 1.27-2025.04.14 | 1.27.16       | 1.7.20           | 1.1.3           |                                   |
| 1.27-2025.03.14 | 1.27.16       | 1.7.20           | 1.1.3           | containerd を 1.7.20 にアップグレードしました。 |
| 1.27-2025.02.15 | 1.27.16       | 1.7.14           | 1.1.3           |                                   |
| 1.27-2025.01.15 | 1.27.16       | 1.7.14           | 1.1.3           |                                   |
| 1.27-2025.01.01 | 1.27.16       | 1.7.14           | 1.1.3           | CVE-2024-9042 のパッチが含まれます。         |
| 1.27-2024.12.11 | 1.27.16       | 1.7.14           | 1.1.3           |                                   |
| 1.27-2024.11.12 | 1.27.16       | 1.7.14           | 1.1.3           |                                   |
| 1.27-2024.10.08 | 1.27.16       | 1.7.14           | 1.1.3           |                                   |
| 1.27-2024.09.10 | 1.27.15       | 1.7.14           | 1.1.3           |                                   |

| AMI のバージョン      | kubelet バージョン | containerd バージョン | csi-proxy バージョン | リリースノート                                                                         |
|-----------------|---------------|------------------|-----------------|---------------------------------------------------------------------------------|
| 1.27-2024.08.13 | 1.27.15       | 1.7.14           | 1.1.3           |                                                                                 |
| 1.27-2024.07.10 | 1.27.15       | 1.7.11           | 1.1.2           | CVE-2024-5321 のパッチが含まれます。                                                       |
| 1.27-2024.06.17 | 1.27.12       | 1.7.11           | 1.1.2           | containerd を 1.7.11 にアップグレードしました。                                               |
| 1.27-2024.05.14 | 1.27.12       | 1.6.28           | 1.1.2           | containerd を 1.6.28 にアップグレードしました。kubelet を 1.27.12 にアップグレードしました。                |
| 1.27-2024.04.09 | 1.27.9        | 1.6.25           | 1.1.2           | containerd を 1.6.25 にアップグレードしました。golang 1.22.1 を使用して CNI と csi-proxy が再構築されました。 |
| 1.27-2024.03.12 | 1.27.9        | 1.6.18           | 1.1.2           |                                                                                 |

| AMI のバージョン      | kubelet バージョン | containerd バージョン | csi-proxy バージョン | リリースノート                                                                                                                       |
|-----------------|---------------|------------------|-----------------|-------------------------------------------------------------------------------------------------------------------------------|
| 1.27-2024.02.13 | 1.27.9        | 1.6.18           | 1.1.2           |                                                                                                                               |
| 1.27-2024.01.09 | 1.27.9        | 1.6.18           | 1.1.2           |                                                                                                                               |
| 1.27-2023.12.12 | 1.27.7        | 1.6.18           | 1.1.2           |                                                                                                                               |
| 1.27-2023.11.14 | 1.27.7        | 1.6.18           | 1.1.2           | CVE-2023-5528 のパッチが含まれます。                                                                                                     |
| 1.27-2023.10.19 | 1.27.6        | 1.6.18           | 1.1.2           | containerd を 1.6.18 にアップグレードしました。<br>新しい <a href="#">ブートストラップスクリプト環境変数</a> (SERVICE_IPV4_CIDR と EXCLUDED_SNAT_CIDRS )を追加しました。 |
| 1.27-2023-09.27 | 1.27.6        | 1.6.6            | 1.1.2           | kubelet の <a href="#">セキュリティ勧告</a> を修正しました。                                                                                   |

| AMI のバージョン      | kubelet バージョン | containerd バージョン | csi-proxy バージョン | リリースノート                                                                                                                                     |
|-----------------|---------------|------------------|-----------------|---------------------------------------------------------------------------------------------------------------------------------------------|
| 1.27-2023.09.12 | 1.27.4        | 1.6.6            | 1.1.2           | Kubernetes API サーバーから Pod IP アドレスを取得する Kubernetes コネクタバイナリを使用するように、Amazon VPC CNI プラグインをアップグレードしました。 <a href="#">プルリクエスト #100</a> をマージしました。 |
| 1.27-2023.08.17 | 1.27.4        | 1.6.6            | 1.1.2           | CVE-2023-3676 、CVE-2023-3893 、および CVE-2023-3955 のパッチが含まれます。                                                                                 |
| 1.27-2023.08.08 | 1.27.3        | 1.6.6            | 1.1.1           |                                                                                                                                             |
| 1.27-2023.07.11 | 1.27.3        | 1.6.6            | 1.1.1           |                                                                                                                                             |
| 1.27-2023.06.20 | 1.27.1        | 1.6.6            | 1.1.1           | DNS サフィックス検索リストに正しく入力されない問題を解決しました。                                                                                                         |

| AMI のバージョン      | kubelet バージョン | containerd バージョン | csi-proxy バージョン | リリースノート                                                                                           |
|-----------------|---------------|------------------|-----------------|---------------------------------------------------------------------------------------------------|
| 1.27-2023.06.14 | 1.27.1        | 1.6.6            | 1.1.1           | CNI のホストポートマッピングのサポートが追加されました。プルリクエスト <a href="#">#93</a> をマージしました。                               |
| 1.27-2023.06.06 | 1.27.1        | 1.6.6            | 1.1.1           | ノードがプライベート Amazon ECR イメージの取り込みに失敗する原因となっていた containerd-roadmap <a href="#">問題 #2042</a> を修正しました。 |
| 1.27-2023.05.18 | 1.27.1        | 1.6.6            | 1.1.1           |                                                                                                   |

## Kubernetes バージョン 1.26

| AMI のバージョン      | kubelet バージョン | containerd バージョン | csi-proxy バージョン | リリースノート |
|-----------------|---------------|------------------|-----------------|---------|
| 1.26-2025.05.17 | 1.26.15       | 1.7.20           | 1.2.1           |         |
| 1.26-2025.04.14 | 1.26.15       | 1.7.20           | 1.1.3           |         |

| AMI のバージョン      | kubelet バージョン | containerd バージョン | csi-proxy バージョン | リリースノート                           |
|-----------------|---------------|------------------|-----------------|-----------------------------------|
| 1.26-2025.03.14 | 1.26.15       | 1.7.20           | 1.1.3           | containerd を 1.7.20 にアップグレードしました。 |
| 1.26-2025.02.15 | 1.26.15       | 1.7.14           | 1.1.3           |                                   |
| 1.26-2025.01.15 | 1.26.15       | 1.7.14           | 1.1.3           |                                   |
| 1.26-2024.12.11 | 1.26.15       | 1.7.14           | 1.1.3           |                                   |
| 1.26-2024.11.12 | 1.26.15       | 1.7.14           | 1.1.3           |                                   |
| 1.26-2024.10.08 | 1.26.15       | 1.7.14           | 1.1.3           |                                   |
| 1.26-2024.09.10 | 1.26.15       | 1.7.14           | 1.1.3           |                                   |
| 1.26-2024.08.13 | 1.26.15       | 1.7.14           | 1.1.3           |                                   |
| 1.26-2024.07.10 | 1.26.15       | 1.7.11           | 1.1.2           | CVE-2024-5321 のパッチが含まれます。         |
| 1.26-2024.06.17 | 1.26.15       | 1.7.11           | 1.1.2           | containerd を 1.7.11 にアップグレードしました。 |

| AMI のバージョン      | kubelet バージョン | containerd バージョン | csi-proxy バージョン | リリースノート                                                                         |
|-----------------|---------------|------------------|-----------------|---------------------------------------------------------------------------------|
| 1.26-2024.05.14 | 1.26.15       | 1.6.28           | 1.1.2           | containerd を 1.6.28 にアップグレードしました。kubelet を 1.26.15 にアップグレードしました。                |
| 1.26-2024.04.09 | 1.26.12       | 1.6.25           | 1.1.2           | containerd を 1.6.25 にアップグレードしました。golang 1.22.1 を使用して CNI と csi-proxy が再構築されました。 |
| 1.26-2024.03.12 | 1.26.12       | 1.6.18           | 1.1.2           |                                                                                 |
| 1.26-2024.02.13 | 1.26.12       | 1.6.18           | 1.1.2           |                                                                                 |
| 1.26-2024.01.09 | 1.26.12       | 1.6.18           | 1.1.2           |                                                                                 |
| 1.26-2023.12.12 | 1.26.10       | 1.6.18           | 1.1.2           |                                                                                 |
| 1.26-2023.11.14 | 1.26.10       | 1.6.18           | 1.1.2           | CVE-2023-5528 のパッチが含まれます。                                                       |

| AMI のバージョン      | kubelet バージョン | containerd バージョン | csi-proxy バージョン | リリースノート                                                                                                                                                    |
|-----------------|---------------|------------------|-----------------|------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 1.26-2023.10.19 | 1.26.9        | 1.6.18           | 1.1.2           | containerd を 1.6.18 にアップグレードしました。kubelet を 1.26.9 にアップグレードしました。<br><a href="#">新しいブートストラップクリプト環境変数 (SERVICE_IPV4_CIDR と EXCLUDED_SNAT_CIDRS )</a> を追加しました。 |
| 1.26-2023.09.12 | 1.26.7        | 1.6.6            | 1.1.2           | Kubernetes API サーバーから Pod IP アドレスを取得する Kubernetes コネクタバイナリを使用するように、Amazon VPC CNI プラグインをアップグレードしました。 <a href="#">プルリクエスト #100</a> をマージしました。                |



| AMI のバージョン      | kubelet バージョン | containerd バージョン | csi-proxy バージョン | リリースノート                                                                                              |
|-----------------|---------------|------------------|-----------------|------------------------------------------------------------------------------------------------------|
| 1.26-2023.08.17 | 1.26.7        | 1.6.6            | 1.1.2           | CVE-2023-3676 、 CVE-2023-3893 、 および CVE-2023-3955 のパッチが含まれます。                                        |
| 1.26-2023.08.08 | 1.26.6        | 1.6.6            | 1.1.1           |                                                                                                      |
| 1.26-2023.07.11 | 1.26.6        | 1.6.6            | 1.1.1           |                                                                                                      |
| 1.26-2023.06.20 | 1.26.4        | 1.6.6            | 1.1.1           | DNS サフィックス検索リストに正しく入力されない問題を解決しました。                                                                  |
| 1.26-2023.06.14 | 1.26.4        | 1.6.6            | 1.1.1           | Kubernetes を 1.26.4 にアップグレードしました。CNI のホストポートマッピングのサポートが追加されました。プルリクエスト <a href="#">#93</a> をマージしました。 |

| AMI のバージョン      | kubelet バージョン | containerd バージョン | csi-proxy バージョン | リリースノート                                                                                                                               |
|-----------------|---------------|------------------|-----------------|---------------------------------------------------------------------------------------------------------------------------------------|
| 1.26-2023.05.09 | 1.26.2        | 1.6.6            | 1.1.1           | ノードの再起動後にポッドでネットワーク接続の問題 <a href="#">#1126</a> を引き起こすバグを修正しました。新しい <a href="#">ブートストラップスクリプトの設定パラメータ</a> (ExcludedSnatCIDs ) を導入しました。 |
| 1.26-2023.04.26 | 1.26.2        | 1.6.6            | 1.1.1           |                                                                                                                                       |
| 1.26-2023.04.11 | 1.26.2        | 1.6.6            | 1.1.1           | サービスクラッシュ時の kubelet および kube-proxy の回復メカニズムを追加しました。                                                                                   |
| 1.26-2023.03.24 | 1.26.2        | 1.6.6            | 1.1.1           |                                                                                                                                       |

## Amazon EKS 最適化 Windows Server 2019 Core AMI

次の表は、Amazon EKS 最適化 Windows Server 2019 Core AMI の現在および以前のバージョンを示しています。

## Example

## Kubernetes バージョン 1.33

| AMI のバージョン      | kubelet バージョン | containerd バージョン | csi-proxy バージョン | リリースノート |
|-----------------|---------------|------------------|-----------------|---------|
| 1.33-2025.05.17 | 1.33.1        | 1.7.27           | 1.2.1           |         |

## Kubernetes バージョン 1.32

| AMI のバージョン      | kubelet バージョン | containerd バージョン | csi-proxy バージョン | リリースノート |
|-----------------|---------------|------------------|-----------------|---------|
| 1.32-2025.05.17 | 1.32.5        | 1.7.20           | 1.2.1           |         |
| 1.32-2025.04.14 | 1.32.1        | 1.7.20           | 1.1.3           |         |
| 1.32-2025.03.14 | 1.32.1        | 1.7.20           | 1.1.3           |         |
| 1.32-2025.02.18 | 1.32.1        | 1.7.20           | 1.1.3           |         |
| 1.32-2025.01.15 | 1.32.4        | 1.7.20           | 1.1.3           |         |

## Kubernetes バージョン 1.31

| AMI のバージョン      | kubelet バージョン | containerd バージョン | csi-proxy バージョン | リリースノート |
|-----------------|---------------|------------------|-----------------|---------|
| 1.31-2025.05.17 | 1.31.9        | 1.7.20           | 1.2.1           |         |

| AMI のバージョン      | kubelet バージョン | containerd バージョン | csi-proxy バージョン | リリースノート                   |
|-----------------|---------------|------------------|-----------------|---------------------------|
| 1.31-2025.04.14 | 1.31.5        | 1.7.20           | 1.1.3           |                           |
| 1.31-2025.03.14 | 1.31.5        | 1.7.20           | 1.1.3           |                           |
| 1.31-2025.02.15 | 1.31.5        | 1.7.20           | 1.1.3           |                           |
| 1.31-2025.01.15 | 1.31.4        | 1.7.20           | 1.1.3           |                           |
| 1.31-2025.01.01 | 1.31.4        | 1.7.20           | 1.1.3           | CVE-2024-9042 のパッチが含まれます。 |
| 1.31-2024.12.13 | 1.31.3        | 1.7.20           | 1.1.3           |                           |
| 1.31-2024.11.12 | 1.31.1        | 1.7.20           | 1.1.3           |                           |
| 1.31-2024.10.08 | 1.31.1        | 1.7.20           | 1.1.3           |                           |
| 1.31-2024.10.01 | 1.31.1        | 1.7.20           | 1.1.3           |                           |
| 1.31-2024.09.10 | 1.31.0        | 1.7.20           | 1.1.3           |                           |

## Kubernetes バージョン 1.30

| AMI のバージョン      | kubelet バージョン | containerd バージョン | csi-proxy バージョン | リリースノート                           |
|-----------------|---------------|------------------|-----------------|-----------------------------------|
| 1.30-2025.05.17 | 1.30.13       | 1.7.20           | 1.2.1           |                                   |
| 1.30-2025.04.14 | 1.30.9        | 1.7.20           | 1.1.3           |                                   |
| 1.30-2025.03.14 | 1.30.9        | 1.7.20           | 1.1.3           | containerd を 1.7.20 にアップグレードしました。 |
| 1.30-2025-02-15 | 1.30.9        | 1.7.14           | 1.1.3           |                                   |
| 1.30-2025.01.15 | 1.30.8        | 1.7.14           | 1.1.3           |                                   |
| 1.30-2025.01.01 | 1.30.8        | 1.7.14           | 1.1.3           | CVE-2024-9042 のパッチが含まれます。         |
| 1.30-2024.12.11 | 1.30.7        | 1.7.14           | 1.1.3           |                                   |
| 1.30-2024.11.12 | 1.30.4        | 1.7.14           | 1.1.3           |                                   |
| 1.30-2024.10.08 | 1.30.4        | 1.7.14           | 1.1.3           |                                   |
| 1.30-2024.09.10 | 1.30.2        | 1.7.14           | 1.1.3           |                                   |

| AMI のバージョン      | kubelet バージョン | containerd バージョン | csi-proxy バージョン | リリースノート                           |
|-----------------|---------------|------------------|-----------------|-----------------------------------|
| 1.30-2024.08.13 | 1.30.2        | 1.7.14           | 1.1.3           |                                   |
| 1.30-2024.07.10 | 1.30.2        | 1.7.14           | 1.1.2           | CVE-2024-5321 のパッチが含まれます。         |
| 1.30-2024.06.17 | 1.30.0        | 1.7.14           | 1.1.2           | containerd を 1.7.14 にアップグレードしました。 |
| 1.30-2024.05.15 | 1.30.0        | 1.6.28           | 1.1.2           |                                   |

## Kubernetes バージョン 1.29

| AMI のバージョン      | kubelet バージョン | containerd バージョン | csi-proxy バージョン | リリースノート                           |
|-----------------|---------------|------------------|-----------------|-----------------------------------|
| 1.29-2025.05.17 | 1.29.15       | 1.7.20           | 1.2.1           |                                   |
| 1.29-2025.04.14 | 1.29.13       | 1.7.20           | 1.1.3           |                                   |
| 1.29-2025.03.14 | 1.29.13       | 1.7.20           | 1.1.3           | containerd を 1.7.20 にアップグレードしました。 |
| 1.29-2025.02.15 | 1.29.13       | 1.7.14           | 1.1.3           |                                   |

| AMI のバージョン      | kubelet バージョン | containerd バージョン | csi-proxy バージョン | リリースノート                                                         |
|-----------------|---------------|------------------|-----------------|-----------------------------------------------------------------|
| 1.29-2025.01.15 | 1.29.12       | 1.7.14           | 1.1.3           |                                                                 |
| 1.29-2025.01.01 | 1.29.12       | 1.7.14           | 1.1.3           | CVE-2024-9042 のパッチが含まれます。                                       |
| 1.29-2024.12.11 | 1.29.10       | 1.7.14           | 1.1.3           |                                                                 |
| 1.29-2024.11.12 | 1.29.8        | 1.7.14           | 1.1.3           |                                                                 |
| 1.29-2024.10.08 | 1.29.8        | 1.7.14           | 1.1.3           |                                                                 |
| 1.29-2024.09.10 | 1.29.6        | 1.7.14           | 1.1.3           |                                                                 |
| 1.29-2024.08.13 | 1.29.6        | 1.7.14           | 1.1.3           |                                                                 |
| 1.29-2024.07.10 | 1.29.6        | 1.7.11           | 1.1.2           | CVE-2024-5321 のパッチが含まれます。                                       |
| 1.29-2024.06.17 | 1.29.3        | 1.7.11           | 1.1.2           |                                                                 |
| 1.29-2024.05.15 | 1.29.3        | 1.7.11           | 1.1.2           | containerd を 1.7.11 にアップグレードしました。kubelet を 1.29.3 にアップグレードしました。 |

| AMI のバージョン      | kubelet バージョン | containerd バージョン | csi-proxy バージョン | リリースノート                                                                         |
|-----------------|---------------|------------------|-----------------|---------------------------------------------------------------------------------|
| 1.29-2024.04.09 | 1.29.0        | 1.6.28           | 1.1.2           | containerd を 1.6.28 にアップグレードしました。golang 1.22.1 を使用して CNI と csi-proxy が再構築されました。 |
| 1.29-2024.03.13 | 1.29.0        | 1.6.25           | 1.1.2           |                                                                                 |
| 1.29-2024.02.13 | 1.29.0        | 1.6.25           | 1.1.2           |                                                                                 |
| 1.29-2024.02.06 | 1.29.0        | 1.6.25           | 1.1.2           | 一時停止画像が kubelet ガベージコレクション処理によって誤って削除されてしまうバグを修正しました。                           |
| 1.29-2024.01.09 | 1.29.0        | 1.6.18           | 1.1.2           |                                                                                 |



## Kubernetes バージョン 1.28

| AMI のバージョン      | kubelet バージョン | containerd バージョン | csi-proxy バージョン | リリースノート                           |
|-----------------|---------------|------------------|-----------------|-----------------------------------|
| 1.28-2025.05.17 | 1.28.15       | 1.7.20           | 1.2.1           |                                   |
| 1.28-2025.04.14 | 1.28.15       | 1.7.20           | 1.1.3           |                                   |
| 1.28-2025.03.14 | 1.28.15       | 1.7.20           | 1.1.3           | containerd を 1.7.20 にアップグレードしました。 |
| 1.28-2025.02.15 | 1.28.15       | 1.7.14           | 1.1.3           |                                   |
| 1.28-2025-01-15 | 1.28.15       | 1.7.14           | 1.1.3           |                                   |
| 1.28-2025-01-01 | 1.28.15       | 1.7.14           | 1.1.3           | CVE-2024-9042 のパッチが含まれます。         |
| 1.28-2024.12.11 | 1.28.15       | 1.7.14           | 1.1.3           |                                   |
| 1.28-2024.11.12 | 1.28.13       | 1.7.14           | 1.1.3           |                                   |
| 1.28-2024.10.08 | 1.28.13       | 1.7.14           | 1.1.3           |                                   |
| 1.28-2024.09.10 | 1.28.11       | 1.7.14           | 1.1.3           |                                   |

| AMI のバージョン      | kubelet バージョン | containerd バージョン | csi-proxy バージョン | リリースノート                                                                         |
|-----------------|---------------|------------------|-----------------|---------------------------------------------------------------------------------|
| 1.28-2024.08.13 | 1.28.11       | 1.7.14           | 1.1.3           |                                                                                 |
| 1.28-2024.07.10 | 1.28.11       | 1.7.11           | 1.1.2           | CVE-2024-5321 のパッチが含まれます。                                                       |
| 1.28-2024.06.17 | 1.28.8        | 1.7.11           | 1.1.2           | containerd を 1.7.11 にアップグレードしました。                                               |
| 1.28-2024.05.14 | 1.28.8        | 1.6.28           | 1.1.2           | containerd を 1.6.28 にアップグレードしました。kubelet を 1.28.8 にアップグレードしました。                 |
| 1.28-2024.04.09 | 1.28.5        | 1.6.25           | 1.1.2           | containerd を 1.6.25 にアップグレードしました。golang 1.22.1 を使用して CNI と csi-proxy が再構築されました。 |
| 1.28-2024.03.13 | 1.28.5        | 1.6.18           | 1.1.2           |                                                                                 |

| AMI のバージョン      | kubelet バージョン | containerd バージョン | csi-proxy バージョン | リリースノート                                                                                                                       |
|-----------------|---------------|------------------|-----------------|-------------------------------------------------------------------------------------------------------------------------------|
| 1.28-2024.02.13 | 1.28.5        | 1.6.18           | 1.1.2           |                                                                                                                               |
| 1.28-2024.01.09 | 1.28.5        | 1.6.18           | 1.1.2           |                                                                                                                               |
| 1.28-2023.12.12 | 1.28.3        | 1.6.18           | 1.1.2           |                                                                                                                               |
| 1.28-2023.11.14 | 1.28.3        | 1.6.18           | 1.1.2           | CVE-2023-5528 のパッチが含まれます。                                                                                                     |
| 1.28-2023.10.19 | 1.28.2        | 1.6.18           | 1.1.2           | containerd を 1.6.18 にアップグレードしました。<br>新しい <a href="#">ブートストラップスクリプト環境変数</a> (SERVICE_IPV4_CIDR と EXCLUDED_SNAT_CIDRS )を追加しました。 |
| 1.28-2023-09.27 | 1.28.2        | 1.6.6            | 1.1.2           | kubelet の <a href="#">セキュリティ勧告</a> を修正しました。                                                                                   |
| 1.28-2023.09.12 | 1.28.1        | 1.6.6            | 1.1.2           |                                                                                                                               |

## Kubernetes バージョン 1.27

| AMI のバージョン      | kubelet バージョン | containerd バージョン | csi-proxy バージョン | リリースノート                           |
|-----------------|---------------|------------------|-----------------|-----------------------------------|
| 1.27-2025.05.17 | 1.27.16       | 1.7.20           | 1.2.1           |                                   |
| 1.27-2025.04.14 | 1.27.16       | 1.7.20           | 1.1.3           |                                   |
| 1.27-2025.03.14 | 1.27.16       | 1.7.20           | 1.1.3           | containerd を 1.7.20 にアップグレードしました。 |
| 1.27-2025.02.15 | 1.27.16       | 1.7.14           | 1.1.3           |                                   |
| 1.27-2025.01.15 | 1.27.16       | 1.7.14           | 1.1.3           |                                   |
| 1.27-2025.01.01 | 1.27.16       | 1.7.14           | 1.1.3           | CVE-2024-9042 のパッチが含まれます。         |
| 1.27-2024.12.11 | 1.27.16       | 1.7.14           | 1.1.3           |                                   |
| 1.27-2024.11.12 | 1.27.16       | 1.7.14           | 1.1.3           |                                   |
| 1.27-2024.10.08 | 1.27.16       | 1.7.14           | 1.1.3           |                                   |
| 1.27-2024.09.10 | 1.27.15       | 1.7.14           | 1.1.3           |                                   |

| AMI のバージョン      | kubelet バージョン | containerd バージョン | csi-proxy バージョン | リリースノート                                                                         |
|-----------------|---------------|------------------|-----------------|---------------------------------------------------------------------------------|
| 1.27-2024.08.13 | 1.27.15       | 1.7.14           | 1.1.3           |                                                                                 |
| 1.27-2024.07.10 | 1.27.15       | 1.7.11           | 1.1.2           | CVE-2024-5321 のパッチが含まれます。                                                       |
| 1.27-2024.06.17 | 1.27.12       | 1.7.11           | 1.1.2           | containerd を 1.7.11 にアップグレードしました。                                               |
| 1.27-2024.05.14 | 1.27.12       | 1.6.28           | 1.1.2           | containerd を 1.6.28 にアップグレードしました。kubelet を 1.27.12 にアップグレードしました。                |
| 1.27-2024.04.09 | 1.27.9        | 1.6.25           | 1.1.2           | containerd を 1.6.25 にアップグレードしました。golang 1.22.1 を使用して CNI と csi-proxy が再構築されました。 |
| 1.27-2024.03.13 | 1.27.9        | 1.6.18           | 1.1.2           |                                                                                 |

| AMI のバージョン      | kubelet バージョン | containerd バージョン | csi-proxy バージョン | リリースノート                                                                                                                       |
|-----------------|---------------|------------------|-----------------|-------------------------------------------------------------------------------------------------------------------------------|
| 1.27-2024.02.13 | 1.27.9        | 1.6.18           | 1.1.2           |                                                                                                                               |
| 1.27-2024.01.09 | 1.27.9        | 1.6.18           | 1.1.2           |                                                                                                                               |
| 1.27-2023.12.12 | 1.27.7        | 1.6.18           | 1.1.2           |                                                                                                                               |
| 1.27-2023.11.14 | 1.27.7        | 1.6.18           | 1.1.2           | CVE-2023-5528 のパッチが含まれます。                                                                                                     |
| 1.27-2023.10.19 | 1.27.6        | 1.6.18           | 1.1.2           | containerd を 1.6.18 にアップグレードしました。<br>新しい <a href="#">ブートストラップスクリプト環境変数</a> (SERVICE_IPV4_CIDR と EXCLUDED_SNAT_CIDRS )を追加しました。 |
| 1.27-2023-09.27 | 1.27.6        | 1.6.6            | 1.1.2           | kubelet の <a href="#">セキュリティ勧告</a> を修正しました。                                                                                   |

| AMI のバージョン      | kubelet バージョン | containerd バージョン | csi-proxy バージョン | リリースノート                                                                                                                                     |
|-----------------|---------------|------------------|-----------------|---------------------------------------------------------------------------------------------------------------------------------------------|
| 1.27-2023.09.12 | 1.27.4        | 1.6.6            | 1.1.2           | Kubernetes API サーバーから Pod IP アドレスを取得する Kubernetes コネクタバイナリを使用するように、Amazon VPC CNI プラグインをアップグレードしました。 <a href="#">プルリクエスト #100</a> をマージしました。 |
| 1.27-2023.08.17 | 1.27.4        | 1.6.6            | 1.1.2           | CVE-2023-3676 、CVE-2023-3893 、および CVE-2023-3955 のパッチが含まれます。                                                                                 |
| 1.27-2023.08.08 | 1.27.3        | 1.6.6            | 1.1.1           |                                                                                                                                             |
| 1.27-2023.07.11 | 1.27.3        | 1.6.6            | 1.1.1           |                                                                                                                                             |
| 1.27-2023.06.20 | 1.27.1        | 1.6.6            | 1.1.1           | DNS サフィックス検索リストに正しく入力されない問題を解決しました。                                                                                                         |

| AMI のバージョン      | kubelet バージョン | containerd バージョン | csi-proxy バージョン | リリースノート                                                                                           |
|-----------------|---------------|------------------|-----------------|---------------------------------------------------------------------------------------------------|
| 1.27-2023.06.14 | 1.27.1        | 1.6.6            | 1.1.1           | CNI のホストポートマッピングのサポートが追加されました。プルリクエスト <a href="#">#93</a> をマージしました。                               |
| 1.27-2023.06.06 | 1.27.1        | 1.6.6            | 1.1.1           | ノードがプライベート Amazon ECR イメージの取り込みに失敗する原因となっていた containerd-roadmap <a href="#">問題 #2042</a> を修正しました。 |
| 1.27-2023.05.18 | 1.27.1        | 1.6.6            | 1.1.1           |                                                                                                   |

## Kubernetes バージョン 1.26

| AMI のバージョン      | kubelet バージョン | containerd バージョン | csi-proxy バージョン | リリースノート |
|-----------------|---------------|------------------|-----------------|---------|
| 1.26-2025.05.17 | 1.26.15       | 1.7.20           | 1.2.1           |         |
| 1.26-2025.04.14 | 1.26.15       | 1.7.20           | 1.1.3           |         |



| AMI のバージョン      | kubelet バージョン | containerd バージョン | csi-proxy バージョン | リリースノート                           |
|-----------------|---------------|------------------|-----------------|-----------------------------------|
| 1.26-2025.03.14 | 1.26.15       | 1.7.20           | 1.1.3           | containerd を 1.7.20 にアップグレードしました。 |
| 1.26-2025.02.15 | 1.26.15       | 1.7.14           | 1.1.3           |                                   |
| 1.26-2025.01.15 | 1.26.15       | 1.7.14           | 1.1.3           |                                   |
| 1.26-2024.12.11 | 1.26.15       | 1.7.14           | 1.1.3           |                                   |
| 1.26-2024.11.12 | 1.26.15       | 1.7.14           | 1.1.3           |                                   |
| 1.26-2024.10.09 | 1.26.15       | 1.7.14           | 1.1.3           |                                   |
| 1.26-2024.09.10 | 1.26.15       | 1.7.14           | 1.1.3           |                                   |
| 1.26-2024.08.13 | 1.26.15       | 1.7.14           | 1.1.3           |                                   |
| 1.26-2024.07.10 | 1.26.15       | 1.7.11           | 1.1.2           | CVE-2024-5321 のパッチが含まれます。         |
| 1.26-2024.06.17 | 1.26.15       | 1.7.11           | 1.1.2           | containerd を 1.7.11 にアップグレードしました。 |

| AMI のバージョン      | kubelet バージョン | containerd バージョン | csi-proxy バージョン | リリースノート                                                                         |
|-----------------|---------------|------------------|-----------------|---------------------------------------------------------------------------------|
| 1.26-2024.05.14 | 1.26.15       | 1.6.28           | 1.1.2           | containerd を 1.6.28 にアップグレードしました。kubelet を 1.26.15 にアップグレードしました。                |
| 1.26-2024.04.09 | 1.26.12       | 1.6.25           | 1.1.2           | containerd を 1.6.25 にアップグレードしました。golang 1.22.1 を使用して CNI と csi-proxy が再構築されました。 |
| 1.26-2024.03.13 | 1.26.12       | 1.6.18           | 1.1.2           |                                                                                 |
| 1.26-2024.02.13 | 1.26.12       | 1.6.18           | 1.1.2           |                                                                                 |
| 1.26-2024.01.09 | 1.26.12       | 1.6.18           | 1.1.2           |                                                                                 |
| 1.26-2023.12.12 | 1.26.10       | 1.6.18           | 1.1.2           |                                                                                 |
| 1.26-2023.11.14 | 1.26.10       | 1.6.18           | 1.1.2           | CVE-2023-5528 のパッチが含まれます。                                                       |

| AMI のバージョン      | kubelet バージョン | containerd バージョン | csi-proxy バージョン | リリースノート                                                                                                                                                    |
|-----------------|---------------|------------------|-----------------|------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 1.26-2023.10.19 | 1.26.9        | 1.6.18           | 1.1.2           | containerd を 1.6.18 にアップグレードしました。kubelet を 1.26.9 にアップグレードしました。<br><a href="#">新しいブートストラップクリプト環境変数 (SERVICE_IPV4_CIDR と EXCLUDED_SNAT_CIDRS )</a> を追加しました。 |
| 1.26-2023.09.12 | 1.26.7        | 1.6.6            | 1.1.2           | Kubernetes API サーバーから Pod IP アドレスを取得する Kubernetes コネクタバイナリを使用するように、Amazon VPC CNI プラグインをアップグレードしました。 <a href="#">プルリクエスト #100</a> をマージしました。                |

| AMI のバージョン      | kubelet バージョン | containerd バージョン | csi-proxy バージョン | リリースノート                                                                                              |
|-----------------|---------------|------------------|-----------------|------------------------------------------------------------------------------------------------------|
| 1.26-2023.08.17 | 1.26.7        | 1.6.6            | 1.1.2           | CVE-2023-3676 、 CVE-2023-3893 、 および CVE-2023-3955 のパッチが含まれます。                                        |
| 1.26-2023.08.08 | 1.26.6        | 1.6.6            | 1.1.1           |                                                                                                      |
| 1.26-2023.07.11 | 1.26.6        | 1.6.6            | 1.1.1           |                                                                                                      |
| 1.26-2023.06.20 | 1.26.4        | 1.6.6            | 1.1.1           | DNS サフィックス検索リストに正しく入力されない問題を解決しました。                                                                  |
| 1.26-2023.06.14 | 1.26.4        | 1.6.6            | 1.1.1           | Kubernetes を 1.26.4 にアップグレードしました。CNI のホストポートマッピングのサポートが追加されました。プルリクエスト <a href="#">#93</a> をマージしました。 |

| AMI のバージョン      | kubelet バージョン | containerd バージョン | csi-proxy バージョン | リリースノート                                                                                                                               |
|-----------------|---------------|------------------|-----------------|---------------------------------------------------------------------------------------------------------------------------------------|
| 1.26-2023.05.09 | 1.26.2        | 1.6.6            | 1.1.1           | ノードの再起動後にポッドでネットワーク接続の問題 <a href="#">#1126</a> を引き起こすバグを修正しました。新しい <a href="#">ブートストラップスクリプトの設定パラメータ</a> (ExcludedSnatCIDs ) を導入しました。 |
| 1.26-2023.04.26 | 1.26.2        | 1.6.6            | 1.1.1           |                                                                                                                                       |
| 1.26-2023.04.11 | 1.26.2        | 1.6.6            | 1.1.1           | サービスクラッシュ時の kubelet および kube-proxy の回復メカニズムを追加しました。                                                                                   |
| 1.26-2023.03.24 | 1.26.2        | 1.6.6            | 1.1.1           |                                                                                                                                       |

## Amazon EKS 最適化 Windows Server 2019 Full AMI

次の表は、Amazon EKS 最適化 Windows Server 2019 Full AMI の現在および以前のバージョンを示しています。

## Example

## Kubernetes バージョン 1.33

| AMI のバージョン      | kubelet バージョン | containerd バージョン | csi-proxy バージョン | リリースノート |
|-----------------|---------------|------------------|-----------------|---------|
| 1.33-2025.05.17 | 1.33.1        | 1.7.27           | 1.2.1           |         |

## Kubernetes バージョン 1.32

| AMI のバージョン      | kubelet バージョン | containerd バージョン | csi-proxy バージョン | リリースノート |
|-----------------|---------------|------------------|-----------------|---------|
| 1.32-2025.05.17 | 1.32.5        | 1.7.20           | 1.2.1           |         |
| 1.32-2025.04.14 | 1.32.1        | 1.7.20           | 1.1.3           |         |
| 1.32-2025.03.14 | 1.32.1        | 1.7.20           | 1.1.3           |         |
| 1.32-2025.02.18 | 1.32.1        | 1.7.20           | 1.1.3           |         |
| 1.32-2025.01.15 | 1.32.0        | 1.7.20           | 1.1.3           |         |

## Kubernetes バージョン 1.31

| AMI のバージョン      | kubelet バージョン | containerd バージョン | csi-proxy バージョン | リリースノート |
|-----------------|---------------|------------------|-----------------|---------|
| 1.31-2025.05.17 | 1.31.9        | 1.7.20           | 1.2.1           |         |

| AMI のバージョン      | kubelet バージョン | containerd バージョン | csi-proxy バージョン | リリースノート                   |
|-----------------|---------------|------------------|-----------------|---------------------------|
| 1.31-2025.04.14 | 1.31.5        | 1.7.20           | 1.1.3           |                           |
| 1.31-2025.03.14 | 1.31.5        | 1.7.20           | 1.1.3           |                           |
| 1.31-2025.02.15 | 1.31.5        | 1.7.20           | 1.1.3           |                           |
| 1.31-2025.01.15 | 1.31.4        | 1.7.20           | 1.1.3           |                           |
| 1.31-2025.01.01 | 1.31.4        | 1.7.20           | 1.1.3           | CVE-2024-9042 のパッチが含まれます。 |
| 1.31-2024.12.13 | 1.31.3        | 1.7.20           | 1.1.3           |                           |
| 1.31-2024.11.12 | 1.31.1        | 1.7.20           | 1.1.3           |                           |
| 1.31-2024.10.08 | 1.31.1        | 1.7.20           | 1.1.3           |                           |
| 1.31-2024.10.01 | 1.31.1        | 1.7.20           | 1.1.3           |                           |
| 1.31-2024.09.10 | 1.31.0        | 1.7.20           | 1.1.3           |                           |

## Kubernetes バージョン 1.30

| AMI のバージョン      | kubelet バージョン | containerd バージョン | csi-proxy バージョン | リリースノート                           |
|-----------------|---------------|------------------|-----------------|-----------------------------------|
| 1.30-2025.05.17 | 1.30.13       | 1.7.20           | 1.2.1           |                                   |
| 1.30-2025.04.14 | 1.30.9        | 1.7.20           | 1.1.3           |                                   |
| 1.30-2025.03.14 | 1.30.9        | 1.7.20           | 1.1.3           | containerd を 1.7.20 にアップグレードしました。 |
| 1.30-2025.02.15 | 1.30.9        | 1.7.14           | 1.1.3           |                                   |
| 1.30-2025.01.15 | 1.30.8        | 1.7.14           | 1.1.3           |                                   |
| 1.30-2025.01.01 | 1.30.8        | 1.7.14           | 1.1.3           | CVE-2024-9042 のパッチが含まれます。         |
| 1.30-2024.12.11 | 1.30.7        | 1.7.14           | 1.1.3           |                                   |
| 1.30-2024.11.12 | 1.30.4        | 1.7.14           | 1.1.3           |                                   |
| 1.30-2024.10.08 | 1.30.4        | 1.7.14           | 1.1.3           |                                   |
| 1.30-2024.09.10 | 1.30.2        | 1.7.14           | 1.1.3           |                                   |



| AMI のバージョン      | kubelet バージョン | containerd バージョン | csi-proxy バージョン | リリースノート                           |
|-----------------|---------------|------------------|-----------------|-----------------------------------|
| 1.30-2024.08.13 | 1.30.2        | 1.7.14           | 1.1.3           |                                   |
| 1.30-2024.07.10 | 1.30.2        | 1.7.14           | 1.1.2           | CVE-2024-5321 のパッチが含まれます。         |
| 1.30-2024.06.17 | 1.30.0        | 1.7.14           | 1.1.2           | containerd を 1.7.14 にアップグレードしました。 |
| 1.30-2024.05.15 | 1.30.0        | 1.6.28           | 1.1.2           |                                   |

## Kubernetes バージョン 1.29

| AMI のバージョン      | kubelet バージョン | containerd バージョン | csi-proxy バージョン | リリースノート                           |
|-----------------|---------------|------------------|-----------------|-----------------------------------|
| 1.29-2025.05.17 | 1.29.15       | 1.7.20           | 1.2.1           |                                   |
| 1.29-2025.04.14 | 1.29.13       | 1.7.20           | 1.1.3           |                                   |
| 1.29-2025.03.14 | 1.29.13       | 1.7.20           | 1.1.3           | containerd を 1.7.20 にアップグレードしました。 |
| 1.29-2025.02.15 | 1.29.13       | 1.7.14           | 1.1.3           |                                   |

| AMI のバージョン      | kubelet バージョン | containerd バージョン | csi-proxy バージョン | リリースノート                                                         |
|-----------------|---------------|------------------|-----------------|-----------------------------------------------------------------|
| 1.29-2025.01.15 | 1.29.12       | 1.7.14           | 1.1.3           |                                                                 |
| 1.29-2025.01.01 | 1.29.12       | 1.7.14           | 1.1.3           | CVE-2024-9042 のパッチが含まれます。                                       |
| 1.29-2024.12.11 | 1.29.10       | 1.7.14           | 1.1.3           |                                                                 |
| 1.29-2024.11.12 | 1.29.8        | 1.7.14           | 1.1.3           |                                                                 |
| 1.29-2024.10.08 | 1.29.8        | 1.7.14           | 1.1.3           |                                                                 |
| 1.29-2024.09.10 | 1.29.6        | 1.7.14           | 1.1.3           |                                                                 |
| 1.29-2024.08.13 | 1.29.6        | 1.7.14           | 1.1.3           |                                                                 |
| 1.29-2024.07.10 | 1.29.6        | 1.7.11           | 1.1.2           | CVE-2024-5321 のパッチが含まれます。                                       |
| 1.29-2024.06.17 | 1.29.3        | 1.7.11           | 1.1.2           |                                                                 |
| 1.29-2024.05.15 | 1.29.3        | 1.7.11           | 1.1.2           | containerd を 1.7.11 にアップグレードしました。kubelet を 1.29.3 にアップグレードしました。 |

| AMI のバージョン      | kubelet バージョン | containerd バージョン | csi-proxy バージョン | リリースノート                                                                         |
|-----------------|---------------|------------------|-----------------|---------------------------------------------------------------------------------|
| 1.29-2024.04.09 | 1.29.0        | 1.6.28           | 1.1.2           | containerd を 1.6.28 にアップグレードしました。golang 1.22.1 を使用して CNI と csi-proxy が再構築されました。 |
| 1.29-2024.03.13 | 1.29.0        | 1.6.25           | 1.1.2           |                                                                                 |
| 1.29-2024.02.13 | 1.29.0        | 1.6.25           | 1.1.2           |                                                                                 |
| 1.29-2024.02.06 | 1.29.0        | 1.6.25           | 1.1.2           | 一時停止画像が kubelet ガベージコレクション処理によって誤って削除されてしまうバグを修正しました。                           |
| 1.29-2024.01.09 | 1.29.0        | 1.6.18           | 1.1.2           |                                                                                 |

## Kubernetes バージョン 1.28

| AMI のバージョン      | kubelet バージョン | containerd バージョン | csi-proxy バージョン | リリースノート                           |
|-----------------|---------------|------------------|-----------------|-----------------------------------|
| 1.28-2025.05.17 | 1.28.15       | 1.7.20           | 1.2.1           |                                   |
| 1.28-2025.04.14 | 1.28.15       | 1.7.20           | 1.1.3           |                                   |
| 1.28-2025.03.14 | 1.28.15       | 1.7.20           | 1.1.3           | containerd を 1.7.20 にアップグレードしました。 |
| 1.28-2025.02.15 | 1.28.15       | 1.7.14           | 1.1.3           |                                   |
| 1.28-2025-01-15 | 1.28.15       | 1.7.14           | 1.1.3           |                                   |
| 1.28-2025-01-01 | 1.28.15       | 1.7.14           | 1.1.3           | CVE-2024-9042 のパッチが含まれます。         |
| 1.28-2024.12.11 | 1.28.15       | 1.7.14           | 1.1.3           |                                   |
| 1.28-2024.11.12 | 1.28.13       | 1.7.14           | 1.1.3           |                                   |
| 1.28-2024.10.08 | 1.28.13       | 1.7.14           | 1.1.3           |                                   |
| 1.28-2024.09.10 | 1.28.11       | 1.7.14           | 1.1.3           |                                   |

| AMI のバージョン      | kubelet バージョン | containerd バージョン | csi-proxy バージョン | リリースノート                                                                         |
|-----------------|---------------|------------------|-----------------|---------------------------------------------------------------------------------|
| 1.28-2024.08.13 | 1.28.11       | 1.7.14           | 1.1.3           |                                                                                 |
| 1.28-2024.07.10 | 1.28.11       | 1.7.11           | 1.1.2           | CVE-2024-5321 のパッチが含まれます。                                                       |
| 1.28-2024.06.17 | 1.28.8        | 1.7.11           | 1.1.2           | containerd を 1.7.11 にアップグレードしました。                                               |
| 1.28-2024.05.14 | 1.28.8        | 1.6.28           | 1.1.2           | containerd を 1.6.28 にアップグレードしました。kubelet を 1.28.8 にアップグレードしました。                 |
| 1.28-2024.04.09 | 1.28.5        | 1.6.25           | 1.1.2           | containerd を 1.6.25 にアップグレードしました。golang 1.22.1 を使用して CNI と csi-proxy が再構築されました。 |
| 1.28-2024.03.13 | 1.28.5        | 1.6.18           | 1.1.2           |                                                                                 |

| AMI のバージョン      | kubelet バージョン | containerd バージョン | csi-proxy バージョン | リリースノート                                                                                                                       |
|-----------------|---------------|------------------|-----------------|-------------------------------------------------------------------------------------------------------------------------------|
| 1.28-2024.02.13 | 1.28.5        | 1.6.18           | 1.1.2           |                                                                                                                               |
| 1.28-2024.01.09 | 1.28.5        | 1.6.18           | 1.1.2           |                                                                                                                               |
| 1.28-2023.12.12 | 1.28.3        | 1.6.18           | 1.1.2           |                                                                                                                               |
| 1.28-2023.11.14 | 1.28.3        | 1.6.18           | 1.1.2           | CVE-2023-5528 のパッチが含まれます。                                                                                                     |
| 1.28-2023.10.19 | 1.28.2        | 1.6.18           | 1.1.2           | containerd を 1.6.18 にアップグレードしました。<br>新しい <a href="#">ブートストラップスクリプト環境変数</a> (SERVICE_IPV4_CIDR と EXCLUDED_SNAT_CIDRS )を追加しました。 |
| 1.28-2023-09.27 | 1.28.2        | 1.6.6            | 1.1.2           | kubelet の <a href="#">セキュリティ勧告</a> を修正しました。                                                                                   |
| 1.28-2023.09.12 | 1.28.1        | 1.6.6            | 1.1.2           |                                                                                                                               |

## Kubernetes バージョン 1.27

| AMI のバージョン      | kubelet バージョン | containerd バージョン | csi-proxy バージョン | リリースノート                           |
|-----------------|---------------|------------------|-----------------|-----------------------------------|
| 1.27-2025.05.17 | 1.27.16       | 1.7.20           | 1.2.1           |                                   |
| 1.27-2025.04.14 | 1.27.16       | 1.7.20           | 1.1.3           |                                   |
| 1.27-2025.03.14 | 1.27.16       | 1.7.20           | 1.1.3           | containerd を 1.7.20 にアップグレードしました。 |
| 1.27-2025.02.15 | 1.27.16       | 1.7.14           | 1.1.3           |                                   |
| 1.27-2025.01.15 | 1.27.16       | 1.7.14           | 1.1.3           |                                   |
| 1.27-2025.01.01 | 1.27.16       | 1.7.14           | 1.1.3           | CVE-2024-9042 のパッチが含まれます。         |
| 1.27-2024.12.11 | 1.27.16       | 1.7.14           | 1.1.3           |                                   |
| 1.27-2024.11.12 | 1.27.16       | 1.7.14           | 1.1.3           |                                   |
| 1.27-2024.10.08 | 1.27.16       | 1.7.14           | 1.1.3           |                                   |
| 1.27-2024.09.10 | 1.27.15       | 1.7.14           | 1.1.3           |                                   |

| AMI のバージョン      | kubelet バージョン | containerd バージョン | csi-proxy バージョン | リリースノート                                                                         |
|-----------------|---------------|------------------|-----------------|---------------------------------------------------------------------------------|
| 1.27-2024.08.13 | 1.27.15       | 1.7.14           | 1.1.3           |                                                                                 |
| 1.27-2024.07.10 | 1.27.15       | 1.7.11           | 1.1.2           | CVE-2024-5321 のパッチが含まれます。                                                       |
| 1.27-2024.06.17 | 1.27.12       | 1.7.11           | 1.1.2           | containerd を 1.7.11 にアップグレードしました。                                               |
| 1.27-2024.05.14 | 1.27.12       | 1.6.28           | 1.1.2           | containerd を 1.6.28 にアップグレードしました。kubelet を 1.27.12 にアップグレードしました。                |
| 1.27-2024.04.09 | 1.27.9        | 1.6.25           | 1.1.2           | containerd を 1.6.25 にアップグレードしました。golang 1.22.1 を使用して CNI と csi-proxy が再構築されました。 |
| 1.27-2024.03.13 | 1.27.9        | 1.6.18           | 1.1.2           |                                                                                 |



| AMI のバージョン      | kubelet バージョン | containerd バージョン | csi-proxy バージョン | リリースノート                                                                                                                       |
|-----------------|---------------|------------------|-----------------|-------------------------------------------------------------------------------------------------------------------------------|
| 1.27-2024.02.13 | 1.27.9        | 1.6.18           | 1.1.2           |                                                                                                                               |
| 1.27-2024.01.09 | 1.27.9        | 1.6.18           | 1.1.2           |                                                                                                                               |
| 1.27-2023.12.12 | 1.27.7        | 1.6.18           | 1.1.2           |                                                                                                                               |
| 1.27-2023.11.14 | 1.27.7        | 1.6.18           | 1.1.2           | CVE-2023-5528 のパッチが含まれます。                                                                                                     |
| 1.27-2023.10.19 | 1.27.6        | 1.6.18           | 1.1.2           | containerd を 1.6.18 にアップグレードしました。<br>新しい <a href="#">ブートストラップスクリプト環境変数</a> (SERVICE_IPV4_CIDR と EXCLUDED_SNAT_CIDRS )を追加しました。 |
| 1.27-2023-09.27 | 1.27.6        | 1.6.6            | 1.1.2           | kubelet の <a href="#">セキュリティ勧告</a> を修正しました。                                                                                   |

| AMI のバージョン      | kubelet バージョン | containerd バージョン | csi-proxy バージョン | リリースノート                                                                                                                                     |
|-----------------|---------------|------------------|-----------------|---------------------------------------------------------------------------------------------------------------------------------------------|
| 1.27-2023.09.12 | 1.27.4        | 1.6.6            | 1.1.2           | Kubernetes API サーバーから Pod IP アドレスを取得する Kubernetes コネクタバイナリを使用するように、Amazon VPC CNI プラグインをアップグレードしました。 <a href="#">プルリクエスト #100</a> をマージしました。 |
| 1.27-2023.08.17 | 1.27.4        | 1.6.6            | 1.1.2           | CVE-2023-3676 、CVE-2023-3893 、および CVE-2023-3955 のパッチが含まれます。                                                                                 |
| 1.27-2023.08.08 | 1.27.3        | 1.6.6            | 1.1.1           |                                                                                                                                             |
| 1.27-2023.07.11 | 1.27.3        | 1.6.6            | 1.1.1           |                                                                                                                                             |
| 1.27-2023.06.20 | 1.27.1        | 1.6.6            | 1.1.1           | DNS サフィックス検索リストに正しく入力されない問題を解決しました。                                                                                                         |

| AMI のバージョン      | kubelet バージョン | containerd バージョン | csi-proxy バージョン | リリースノート                                                                                             |
|-----------------|---------------|------------------|-----------------|-----------------------------------------------------------------------------------------------------|
| 1.27-2023.06.14 | 1.27.1        | 1.6.6            | 1.1.1           | CNI のホストポートマッピングのサポートが追加されました。プルリクエスト <a href="#">#93</a> をマージしました。                                 |
| 1.27-2023.06.06 | 1.27.1        | 1.6.6            | 1.1.1           | ノードがプライベート Amazon ECR イメージの取り込みに失敗する原因となっていた containerd s-roadmap <a href="#">問題 #2042</a> を修正しました。 |
| 1.27-2023.05.17 | 1.27.1        | 1.6.6            | 1.1.1           |                                                                                                     |

## Kubernetes バージョン 1.26

| AMI のバージョン      | kubelet バージョン | containerd バージョン | csi-proxy バージョン | リリースノート |
|-----------------|---------------|------------------|-----------------|---------|
| 1.26-2025.05.17 | 1.26.15       | 1.7.20           | 1.2.1           |         |
| 1.26-2025.04.14 | 1.26.15       | 1.7.20           | 1.1.3           |         |

| AMI のバージョン      | kubelet バージョン | containerd バージョン | csi-proxy バージョン | リリースノート                           |
|-----------------|---------------|------------------|-----------------|-----------------------------------|
| 1.26-2025.03.14 | 1.26.15       | 1.7.20           | 1.1.3           | containerd を 1.7.20 にアップグレードしました。 |
| 1.26-2025.02.15 | 1.26.15       | 1.7.14           | 1.1.3           |                                   |
| 1.26-2025.01.15 | 1.26.15       | 1.7.14           | 1.1.3           |                                   |
| 1.26-2024.12.11 | 1.26.15       | 1.7.14           | 1.1.3           |                                   |
| 1.26-2024.11.12 | 1.26.15       | 1.7.14           | 1.1.3           |                                   |
| 1.26-2024.10.08 | 1.26.15       | 1.7.14           | 1.1.3           |                                   |
| 1.26-2024.09.10 | 1.26.15       | 1.7.14           | 1.1.3           |                                   |
| 1.26-2024.08.13 | 1.26.15       | 1.7.14           | 1.1.3           |                                   |
| 1.26-2024.07.10 | 1.26.15       | 1.7.11           | 1.1.2           | CVE-2024-5321 のパッチが含まれます。         |
| 1.26-2024.06.17 | 1.26.15       | 1.7.11           | 1.1.2           | containerd を 1.7.11 にアップグレードしました。 |

| AMI のバージョン      | kubelet バージョン | containerd バージョン | csi-proxy バージョン | リリースノート                                                                         |
|-----------------|---------------|------------------|-----------------|---------------------------------------------------------------------------------|
| 1.26-2024.05.14 | 1.26.15       | 1.6.28           | 1.1.2           | containerd を 1.6.28 にアップグレードしました。kubelet を 1.26.15 にアップグレードしました。                |
| 1.26-2024.04.09 | 1.26.12       | 1.6.25           | 1.1.2           | containerd を 1.6.25 にアップグレードしました。golang 1.22.1 を使用して CNI と csi-proxy が再構築されました。 |
| 1.26-2024.03.13 | 1.26.12       | 1.6.18           | 1.1.2           |                                                                                 |
| 1.26-2024.02.13 | 1.26.12       | 1.6.18           | 1.1.2           |                                                                                 |
| 1.26-2024.01.09 | 1.26.12       | 1.6.18           | 1.1.2           |                                                                                 |
| 1.26-2023.12.12 | 1.26.10       | 1.6.18           | 1.1.2           |                                                                                 |
| 1.26-2023.11.14 | 1.26.10       | 1.6.18           | 1.1.2           | CVE-2023-5528 のパッチが含まれます。                                                       |

| AMI のバージョン      | kubelet バージョン | containerd バージョン | csi-proxy バージョン | リリースノート                                                                                                                                                    |
|-----------------|---------------|------------------|-----------------|------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 1.26-2023.10.19 | 1.26.9        | 1.6.18           | 1.1.2           | containerd を 1.6.18 にアップグレードしました。kubelet を 1.26.9 にアップグレードしました。<br><a href="#">新しいブートストラップクリプト環境変数 (SERVICE_IPV4_CIDR と EXCLUDED_SNAT_CIDRS )</a> を追加しました。 |
| 1.26-2023.09.12 | 1.26.7        | 1.6.6            | 1.1.2           | Kubernetes API サーバーから Pod IP アドレスを取得する Kubernetes コネクタバイナリを使用するように、Amazon VPC CNI プラグインをアップグレードしました。 <a href="#">プルリクエスト #100</a> をマージしました。                |

| AMI のバージョン      | kubelet バージョン | containerd バージョン | csi-proxy バージョン | リリースノート                                                                                              |
|-----------------|---------------|------------------|-----------------|------------------------------------------------------------------------------------------------------|
| 1.26-2023.08.17 | 1.26.7        | 1.6.6            | 1.1.2           | CVE-2023-3676 、 CVE-2023-3893 、 および CVE-2023-3955 のパッチが含まれます。                                        |
| 1.26-2023.08.08 | 1.26.6        | 1.6.6            | 1.1.1           |                                                                                                      |
| 1.26-2023.07.11 | 1.26.6        | 1.6.6            | 1.1.1           |                                                                                                      |
| 1.26-2023.06.20 | 1.26.4        | 1.6.6            | 1.1.1           | DNS サフィックス検索リストに正しく入力されない問題を解決しました。                                                                  |
| 1.26-2023.06.14 | 1.26.4        | 1.6.6            | 1.1.1           | Kubernetes を 1.26.4 にアップグレードしました。CNI のホストポートマッピングのサポートが追加されました。プルリクエスト <a href="#">#93</a> をマージしました。 |

| AMI のバージョン      | kubelet バージョン | containerd バージョン | csi-proxy バージョン | リリースノート                                                                                                                               |
|-----------------|---------------|------------------|-----------------|---------------------------------------------------------------------------------------------------------------------------------------|
| 1.26-2023.05.09 | 1.26.2        | 1.6.6            | 1.1.1           | ノードの再起動後にポッドでネットワーク接続の問題 <a href="#">#1126</a> を引き起こすバグを修正しました。新しい <a href="#">ブートストラップスクリプトの設定パラメータ</a> (ExcludedSnatCIDs ) を導入しました。 |
| 1.26-2023.04.26 | 1.26.2        | 1.6.6            | 1.1.1           |                                                                                                                                       |
| 1.26-2023.04.11 | 1.26.2        | 1.6.6            | 1.1.1           | サービスクラッシュ時の kubelet および kube-proxy の回復メカニズムを追加しました。                                                                                   |
| 1.26-2023.03.24 | 1.26.2        | 1.6.6            | 1.1.1           |                                                                                                                                       |

## 推奨 Microsoft Windows AMI ID を取得する

ノードをデプロイする際に、事前構築済みの Amazon EKS 最適化 Amazon マシンイメージ (AMI) の ID を指定できます。希望する設定に合った AMI ID を取得するには、AWS Systems Manager Parameter Store API をクエリします。この API を使用すると、Amazon EKS 最適化 AMI ID を手動で検索する必要がなくなります。詳細については、「[GetParameter](#)」を参照してください



い。使用する [IAM プリンシパル](#) には、Amazon EKS 最適化 AMI メタデータを取得するための `ssm:GetParameter` IAM アクセス許可が必要です。

サブパラメータ `image_id` を使用する次のコマンドで、推奨される最新の Amazon EKS 最適化 Windows AMI のイメージ ID を取得できます。必要に応じてコマンドに次の変更を加え、変更したコマンドを実行してください：

- `release` は、以下のいずれかのオプションに置き換えます。
  - Windows Server 2022 には `2022` を使用します。
  - Windows Server 2019 には `2019` を使用します。
- `installation-option` は、以下のいずれかのオプションに置き換えます。詳細については、「[What is the Server Core installation option in Windows Server](#)」を参照してください。
  - アタックサーフェスを小さくした最小限のインストールには `Core` を使用します。
  - Windows デスクトップエクスペリエンスを含めるには `Full` を使用します。
- `kubernetes-version` をサポートされる [Amazon EKS バージョン](#) に置き換えます。
- `region-code` を、AMI ID が必要な [Amazon EKS がサポートしている AWS リージョン](#) に置き換えます。

```
aws ssm get-parameter --name /aws/service/ami-windows-latest/Windows_Server-release-English-installation-option-EKS_Optimized-kubernetes-version/image_id \
--region region-code --query "Parameter.Value" --output text
```

プレースホルダーの置換が行われた後のコマンドの例を以下に示します。

```
aws ssm get-parameter --name /aws/service/ami-windows-latest/Windows_Server-2022-English-Core-EKS_Optimized-k8s-n-2/image_id \
--region us-west-2 --query "Parameter.Value" --output text
```

出力例は次のとおりです。

```
ami-1234567890abcdef0
```

## Image Builder を使用してカスタム Windows AMI を構築する

EC2 Image Builder を使用し、次のオプションのいずれかを使用して、カスタムの Amazon EKS 最適化 Windows AMI を作成できます。

- [Amazon EKS 最適化 Windows AMI をベースとして使用する](#)
- [Amazon 管理ビルドコンポーネントを使用する](#)

どちらの方法でも、独自の Image Builder レシピを作成する必要があります。詳細については、「Image Builder ユーザーガイド」の「[イメージレシピの新しいバージョンの作成](#)」を参照してください。

**⚠ Important**

以下の eks の [Amazon 管理] コンポーネントには、CVE-2024-5321 のパッチが含まれません。

- 1.26.4 以上
- 1.27.2 以上
- 1.28.2 以上
- 1.29.2 以上
- 1.30.1 以上

## Amazon EKS 最適化 Windows AMI をベースとして使用する

このオプションは、カスタム Windows AMI を構築するのに推奨される方法です。当社が提供する Amazon EKS 最適化 Windows AMI は、Amazon 管理のビルドコンポーネントよりも頻繁に更新されます。

### 1. 新しい Image Builder レシピを始める

- a. で EC2 Image Builder コンソールを開きます。 <https://console.aws.amazon.com/imagebuilder>。
- b. 左側のナビゲーションペインで、[イメージレシピ] を選択します。
- c. [イメージレシピの作成] を選択します。

### 2. [レシピの詳細] セクションで、[名前] と [バージョン] を入力します。

### 3. [ベースイメージ] セクションで Amazon EKS 最適化 Windows AMI の ID を指定します。

- a. [カスタム AMI ID を入力] を選択します。
- b. 必要な Windows OS バージョンの AMI ID を取得します。詳細については、「[the section called “最新の ID を取得する”](#)」を参照してください。

- c. カスタム [AMI ID] を入力します。AMI ID が見つからない場合は、AMI ID の AWS リージョンがコンソールの右上に表示されている AWS リージョンと一致しているか確認してください。
4. (オプション) 最新のセキュリティアップデートを入手するには、update-windows コンポーネントを [ビルドコンポーネント -] セクションに追加します。
  - a. [名前でコンポーネントを検索する] 検索ボックスの右側にあるドロップダウンリストから、[Amazon 管理] を選択します。
  - b. [名前でコンポーネントを検索する] 検索ボックスに update-windows と入力します。
  - c. **update-windows** の検索結果のチェックボックスを選択します。このコンポーネントは、オペレーティングシステムの最新 Windows パッチを含みます。
5. 必要な設定で残りのイメージレシピの入力を完了します。詳細については、「Image Builder ユーザーガイド」の「[イメージレシピの新しいバージョンの作成 \(コンソール\)](#)」を参照してください。
6. [レシピを作成する] を選択します。
7. 新しいイメージレシピを新規または既存のイメージパイプラインで使います。イメージパイプラインが正常に実行されると、カスタム AMI が出力イメージとして一覧表示され、使用できるようになります。詳細については、「[EC2 Image Builder コンソールウィザードを使用してイメージパイプラインを作成する](#)」を参照してください。

## Amazon 管理ビルドコンポーネントを使用する

Amazon EKS 最適化 Windows AMI をベースとして使用できない場合は、代わりに Amazon 管理ビルドコンポーネントを使用できます。このオプションは、サポートされている最新の Kubernetes バージョンよりも遅れる可能性があります。

1. 新しい Image Builder レシピを始める
  - a. で EC2 Image Builder コンソールを開きます。 <https://console.aws.amazon.com/imagebuilder>。
  - b. 左側のナビゲーションペインで、[イメージレシピ] を選択します。
  - c. [イメージレシピの作成] を選択します。
2. [レシピの詳細] セクションで、[名前] と [バージョン] を入力します。
3. [ベースイメージ] のセクションで、カスタム AMI の作成に使用するオプションを決定します。
  - [管理イメージの選択] — [イメージオペレーティングシステム (OS)] で [Windows] を選択します。次に、[イメージのオリジン] について、以下のいずれかのオプションを選択します。

- [クイックスタート (Amazon 管理)] – [イメージ名] ドロップダウンから、Amazon EKS でサポートされている Windows Server のバージョンを選択します。詳細については、「[the section called “Windows”](#)」を参照してください。
  - [ユーザー所有のイメージ] – [イメージ名] で、独自ライセンスを持つ独自のイメージの ARN を選択します。Amazon EKS コンポーネントをインストール済みのイメージを指定することはできません。
  - [カスタム AMI ID を入力] – [AMI ID] に、独自ライセンスを持つ AMI の ID を入力します。Amazon EKS コンポーネントをインストール済みのイメージを指定することはできません。
4. [ビルドコンポーネント - Windows] セクションで、次の操作を行います。
- a. [名前でコンポーネントを検索する] 検索ボックスの右側にあるドロップダウンリストから、[Amazon 管理] を選択します。
  - b. [名前でコンポーネントを検索する] 検索ボックスに eks と入力します。
  - c. たとえ目的のバージョンではない場合でも、**eks-optimized-ami-windows** 検索結果のチェックボックスを選択します。
  - d. [名前でコンポーネントを検索する] 検索ボックスに update-windows と入力します。
  - e. update-windows の検索結果のチェックボックスを選択します。このコンポーネントは、オペレーティングシステムの最新 Windows パッチを含みます。
5. [選択したコンポーネント] セクションで、次の手順を実行します。
- a. [**eks-optimized-ami-windows**] の [バージョンニングオプション] を選択してください。
  - b. [コンポーネントバージョンを指定する] を選択します。
  - c. [コンポーネントバージョン] フィールドに **version.x** と入力し、##### をサポートされている Kubernetes バージョンに置き換えます。バージョン番号の一部に **x** を入力すると、明示的に定義したバージョンの一部と一致する最新のコンポーネントバージョンが使用されます。コンソールの出力に注目してください。目的のバージョンが管理コンポーネントとして利用可能かどうかを示されます。ビルドコンポーネントでは、最新の Kubernetes バージョンを使用できない場合があります。使用できるバージョンの詳細については、「[the section called “eks-optimized-ami-windows コンポーネントのバージョンに関する情報を取得する”](#)」を参照してください。
6. 必要な設定で残りのイメージレシピの入力を完了します。詳細については、「Image Builder ユーザーガイド」の「[イメージレシピの新しいバージョンの作成 \(コンソール\)](#)」を参照してください。
7. [レシピを作成する] を選択します。

- 新しいイメージレシピを新規または既存のイメージパイプラインで使します。イメージパイプラインが正常に実行されると、カスタム AMI が出力イメージとして一覧表示され、使用できるようになります。詳細については、「[EC2 Image Builder コンソールウィザードを使用してイメージパイプラインを作成する](#)」を参照してください。

## eks-optimized-ami-windows コンポーネントのバージョンに関する情報を取得する

各コンポーネントにインストールされているものに関する特定の情報を取得できます。例えば、どの kubelet バージョンがインストールされているかを確認できます。コンポーネントには、Amazon EKS がサポートする Windows オペレーティングシステムのバージョンで機能テストが実施されます。詳細については、「[the section called “リリースカレンダー”](#)」を参照してください。その他の Windows OS バージョンで、サポートされていない、またはサポート終了としてリストにあるものは、このコンポーネントと互換性がない可能性があります。

- で EC2 Image Builder コンソールを開きます。<https://console.aws.amazon.com/imagebuilder>。
- 左にあるナビゲーションペインで、[コンポーネント] をクリックします。
- [名前でコンポーネントを検索する] 検索ボックスの右側にあるドロップダウンリストで、[自己所有] を [クイックスタート (Amazon 管理)] に変更します。
- [名前でコンポーネントを検索] ボックスに eks と入力します。
- (オプション) 最新バージョンを使用している場合は、[バージョン] 列を 2 回選択して降順にソートします。
- 目的のバージョンの [eks-optimized-ami-windows] リンクを選択します。

結果のページの [説明] には、特定の情報が表示されます。

## ノードの自動修復を有効にし、ノードのヘルス問題を調査する

ノードのヘルスとは、ワークロードを効果的に実行するためのノードの運用ステータスと機能を指します。正常なノードは、予想される接続を維持し、十分なリソースを持ち、中断することなくポッドを正常に実行できます。お使いのノードに関する詳細を取得する方法については、「[the section called “ノードのヘルスを表示する”](#)」および「[the section called “ノードログの取得”](#)」を参照してください。

ノードを正常に維持するために、Amazon EKS にはノードモニタリングエージェントとノード自動修復の機能があります。

**⚠ Important**

ノードモニタリングエージェントとノード自動修復を使用できるのは、Linux のみです。Windows では使用できません。

## ノードモニタリングエージェント

ノードモニタリングエージェントはノードログを自動的に読み取り、特定のヘルス問題を検出します。ノードログを解析して障害を検出し、ワーカーノードに関するさまざまなステータス情報を表示します。ストレージやネットワーク形成の問題など、検出された問題のカテゴリごとに、ワーカーノードに専用の NodeCondition が適用されます。検出されたヘルスの問題の説明は、オブザーバビリティダッシュボードで確認できます。詳細については、「[the section called “ノードのヘルスの問題”](#)」を参照してください。

ノードモニタリングエージェントは、すべての Amazon EKS Auto Mode クラスターの機能として用意されています。他のクラスタータイプでは、モニタリングエージェントを Amazon EKS アドオンとして追加できます。詳細については、「[the section called “アドオンを作成する”](#)」を参照してください。

## ノードの自動修復

ノードの自動修復は、ノードのヘルスを継続的にモニタリングし、検出された問題に自動的に対応して、可能な場合はノードを置き換えることのできる追加機能です。これにより、最小限の手動操作でクラスターの全体的な可用性が向上します。ヘルスチェックが失敗すると、ノードは自動的に遮断され、ノードに新しいポッドがスケジュールされません。

ノードの自動修復自体は、kubelet および手動で削除されたノードオブジェクトの Ready 状態に対応できます。ノードモニタリングエージェントと組み合わせると、ノードの自動修復は、他の方法では検出されないより多くの状態に対応できます。これらの追加の状態には、KernelReady、NetworkingReady、および StorageReady があります。

この自動ノード修復は、クラスターへの結合の失敗、応答しない kubelets、アクセラレーター (デバイス) エラーの増加など、断続的なノードの問題に自動的に対処します。信頼性の向上により、アプリケーションのダウンタイムが短縮され、クラスターのオペレーションが向上します。ノードの自動修復では、DiskPressure、MemoryPressure、PIDPressure などの、報告された特定の問題を処理できません。Amazon EKS は AcceleratedHardwareReady NodeConditions に作用するまでに 10 分かかり、さらに他のすべての状態の場合は 30 分かかります。



また、マネージド型ノードグループは、2 つのシナリオで、安全上の理由からノードの修復を自動的に無効にします。以前から進行中の修復操作は、どちらの状態でも続行されます。

- クラスターのゾーンシフトが Application Recovery Controller (ARC) を介してトリガーされた場合、それ以降のすべての修復オペレーションが停止します。
- ノードグループに 5 つ以上のノードがあり、ノードグループ内のノードの 20% 以上が異常な状態である場合、修復オペレーションは停止します。

マネージド型ノードグループを作成または編集するときに、ノードの自動修復を有効にできます。

- Amazon EKS コンソールを使用する場合は、マネージド型ノードグループの [ノード自動修復を有効にする] チェックボックスをオンにします。詳細については、「[the section called “作成”](#)」を参照してください。
- AWS CLI を使用する場合は、`--node-repair-config enabled=true` を [eks create nodegroup](#) または [eks update-nodegroup-config](#) コマンドに追加します。
- ノードの自動修復でマネージド型ノードグループを使用する `eksctl ClusterConfig` の例については、GitHub の「[44-node-repair.yaml](#)」を参照してください。

## ノードのヘルスの問題

次の表は、ノードモニタリングエージェントが検出できるノードのヘルス問題を示しています。次の 2 種類の問題があります。

- 症状 – インスタンスの置き換えや再起動などの修復アクションを必要とするターミナルの問題。自動修復が有効になっている場合、Amazon EKS はノードの置換または再起動として修復アクションを実行します。詳細については、「[the section called “ノードの状態”](#)」を参照してください。
- イベント – 一時的な問題または最適ではないノード設定。自動修復アクションは行われません。詳細については、「[the section called “ノードイベント”](#)」を参照してください。

### カーネルノードのヘルスの問題

| 名前                 | 緊急度 | 説明                         |
|--------------------|-----|----------------------------|
| ForkFailedOutOfPID | 条件  | フォークまたは実行の呼び出しが失敗したのは、システム |

| 名前                      | 緊急度  | 説明                                                                                    |
|-------------------------|------|---------------------------------------------------------------------------------------|
|                         |      | がプロセス ID またはメモリ不足であるためです。これは、ゾンビプロセスまたは物理メモリの枯渇が原因である可能性があります。                        |
| AppBlocked              | イベント | タスクが長時間スケジュールからブロックされており、通常は入力または出力がブロックされていることが原因で発生します。                             |
| AppCrash                | イベント | ノード上のアプリケーションがクラッシュしました。                                                              |
| ApproachingKernelPidMax | イベント | プロセスの数が、現在の kernel.pid_max 設定で使用可能な PID の最大数に近づいています。最大数に到達すると、これ以上プロセスを起動することはできません。 |
| ApproachingMaxOpenFiles | イベント | 開いているファイルの数は、現在のカーネル設定で可能な開いているファイルの最大数に近づいています。最大数に到達すると、新しいファイルを開くことができなくなります。      |
| ConntrackExceededKernel | イベント | 接続追跡がカーネルの最大数を超え、新しい接続を確立できなかったため、パケットロスが発生する可能性があります。                                |



| 名前                       | 緊急度  | 説明                                                                                                            |
|--------------------------|------|---------------------------------------------------------------------------------------------------------------|
| ExcessiveZombieProcesses | イベント | 完全に再要求できないプロセスが大量に蓄積されています。これはアプリケーションの問題を示しており、システムプロセスの制限に達する可能性があります。                                      |
| KernelBug                | イベント | CPU またはメモリの使用率が高いノードが原因でイベント処理が遅れることがあります。カーネルのバグが Linux カーネル自体によって検出および報告されました。                              |
| LargeEnvironment         | イベント | このプロセスの環境変数の数は予想よりも多く、enableServiceLinks が true に設定されている多くのサービスが原因で発生する可能性があります。このため、パフォーマンスの問題が発生する可能性があります。 |
| RapidCron                | イベント | このノードでは、cron ジョブが 5 分間隔よりも速く実行されています。このため、ジョブが大量のリソースを消費すると、パフォーマンスに影響が出る可能性があります。                            |
| SoftLockup               | イベント | CPU が一定時間停止しました。                                                                                              |

## ネットワークノードのヘルスの問題

| 名前                       | 緊急度  | 説明                                                        |
|--------------------------|------|-----------------------------------------------------------|
| InterfaceNotRunning      | 条件   | このインターフェイスは実行されていないか、ネットワークの問題があります。                      |
| InterfaceNotUp           | 条件   | このインターフェイスは起動していないか、ネットワークの問題があります。                       |
| IPAMDNotReady            | 条件   | IPAMD が API サーバーに接続できません。                                 |
| IPAMDNotRunning          | 条件   | aws-k8s-agent プロセスは実行中であることが確認されませんでした。                   |
| MissingLoopbackInterface | 条件   | このインスタンスには、ループバックインターフェイスがないため、ローカル接続によってはサービスの不具合が発生します。 |
| BandwidthInExceeded      | イベント | インバウンド集計帯域幅がインスタンスの最大値を超えたため、パケットはキューに入れられたか、ドロップされました。   |
| BandwidthOutExceeded     | イベント | アウトバウンド集計帯域幅がインスタンスの最大値を超えたため、パケットはキューに入れられたか、ドロップされました。  |
| ConntrackExceeded        | イベント | 接続追跡がインスタンスの最大数を超え、新しい接続を確                                |

| 名前                             | 緊急度  | 説明                                                                  |
|--------------------------------|------|---------------------------------------------------------------------|
|                                |      | 立できなかったため、パケットロスが発生する可能性があります。                                      |
| IPAMDNoIPs                     | イベント | IPAM-D が IP アドレスから外れています。                                           |
| IPAMDRepeatedlyRestart         | イベント | IPAMD サービスで複数の再起動が発生しました。                                           |
| KubeProxyNotReady              | イベント | Kube-proxy がリソースの監視または一覧表示に失敗しました。                                  |
| LinkLocalExceeded              | イベント | ローカルプロキシサービスへのトラフィックの PPS がネットワークインターフェイスの最大値を超えたため、パケットがドロップされました。 |
| MissingDefaultRoutes           | イベント | デフォルトのルートルールが見つかりません。                                               |
| MissingIPRules、MissingIPRoutes | イベント | ルートテーブルに次のポッド IP のルートルールがありません。                                     |
| NetworkSysctl                  | イベント | このノードのネットワーク sysctl 設定が正しくない可能性があります。                               |

| 名前                   | 緊急度  | 説明                                                                                                              |
|----------------------|------|-----------------------------------------------------------------------------------------------------------------|
| PortConflict         | イベント | ポッドが hostPort を使用する場合、ホストの既にバインドされているポートを上書きするように iptables ルールを書き込みできるため、API サーバーが kubelet にアクセスできなくなる可能性があります。 |
| PPSExceeded          | イベント | 双方向 PPS がインスタンスの最大値を超えたため、パケットはキューに入れられたか、ドロップされました。                                                            |
| UnexpectedRejectRule | イベント | iptables で予期しない REJECT または DROP ルールが検出されたため、予想されたトラフィックがブロックされる可能性があります。                                        |

## Neuron ノードのヘルスの問題

| 名前                          | 緊急度 | 説明                                 |
|-----------------------------|-----|------------------------------------|
| NeuronDMAError              | 条件  | DMA エンジンで回復不可能なエラーが発生しました。         |
| NeuronHBMUncorrectableError | 条件  | HBM で修正不可能なエラーが発生し、誤った結果が生成されました。  |
| NeuronNCUncorrectableError  | 条件  | Neuron Core の修正不可能なメモリエラーが検出されました。 |

| 名前                           | 緊急度 | 説明                                     |
|------------------------------|-----|----------------------------------------|
| NeuronSRAMUncorrectableError | 条件  | オンチップ SRAM でパリティエラーが発生し、誤った結果が生成されました。 |

## NVIDIA ノードのヘルスの問題

自動修復が有効になっている場合、リストされている修復アクションは、問題が検出されてから 10 分後に開始されます。XID エラーの詳細については、「NVIDIA GPU のデプロイおよび管理ドキュメント」の「[Xid エラー](#)」を参照してください。個々の XID メッセージの詳細については、「NVIDIA GPU Deployment and Management Documentation」の「[Understanding Xid Messages](#)」を参照してください。

| 名前                   | 緊急度 | 説明                                | 修復アクション |
|----------------------|-----|-----------------------------------|---------|
| NvidiaDoubleBitError | 条件  | GPU ドライバーによってダブルビットエラーが生成されました。   | 置換      |
| NvidiaNVLinkError    | 条件  | NVLink エラーが GPU ドライバーによって報告されました。 | 置換      |
| NvidiaXID13Error     | 条件  | グラフィックエンジンの例外があります。               | 再起動     |
| NvidiaXID31Error     | 条件  | ハードウェアの問題が疑われます。                  | 再起動     |
| NvidiaXID48Error     | 条件  | ドライバーによってダブルビット ECC エラーが報告されます。   | 再起動     |

| 名前               | 緊急度 | 説明                                                                                                                    | 修復アクション |
|------------------|-----|-----------------------------------------------------------------------------------------------------------------------|---------|
| NvidiaXID63Error | 条件  | ページリタイアまたは行の再マッピングがあります。                                                                                              | 再起動     |
| NvidiaXID64Error | 条件  | ページの廃止またはノードの再マッピングの実行に失敗しました。                                                                                        | 再起動     |
| NvidiaXID74Error | 条件  | GPU から別の GPU または NVLink 経由の NVSwitch への接続に問題があります。これは、リンク自体のハードウェア障害を示しているか、リンクのリモートエンドにあるデバイスに問題があることを示している可能性があります。 | 置換      |
| NvidiaXID79Error | 条件  | GPU ドライバーが PCI Express 接続を介して GPU にアクセスしようとしたが、GPU にアクセスできないことがわかりました。                                                | 置換      |
| NvidiaXID94Error | 条件  | ECC メモリエラーがあります。                                                                                                      | 再起動     |
| NvidiaXID95Error | 条件  | ECC メモリエラーがあります。                                                                                                      | 再起動     |

| 名前                | 緊急度 | 説明                                                                                                                    | 修復アクション |
|-------------------|-----|-----------------------------------------------------------------------------------------------------------------------|---------|
| NvidiaXID119Error | 条件  | GSP は、ドライバーの他のビットからの RPC リクエストに応答しなくなりました。                                                                            | 置換      |
| NvidiaXID120Error | 条件  | GSP は時間内に応答しましたが、エラーが発生しています。                                                                                         | 置換      |
| NvidiaXID121Error | 条件  | C2C はチップ相互接続です。これにより、CPU、アクセラレーターなどの間でメモリを共有できます。                                                                     | 置換      |
| NvidiaXID140Error | 条件  | GPU ドライバーは、GPU メモリで修正不可能なエラーを観測した可能性があります。これは、動的ページをオフライン表示にしたり、行を再マッピングするためにページをマークする GPU ドライバーの機能を中断するような方法で行われました。 | 置換      |

| 名前                     | 緊急度  | 説明                                                                                                 | 修復アクション |
|------------------------|------|----------------------------------------------------------------------------------------------------|---------|
| NvidiaPageRetirement   | イベント | GPU ドライバーがメモリページを廃止としてマークしました。これは、1つのダブルビットエラーが起こった場合、または同じアドレスで2つのシングルビットエラーが起こった場合に発生する可能性があります。 | なし      |
| NvidiaXID[Code]Warning | イベント | このリストで定義されている以外の XID が発生すると、このイベントが発生します。                                                          | なし      |
| DCGMError              | 条件   | Data Center GPU Manager (DCGM) ホストプロセスへの接続を失ったか、確立できませんでした。                                        | なし      |
| DCGMDiagnosticError    | 条件   | DCGM アクティブ診断の実行中に問題が発生しました。                                                                        | なし      |
| DCGMDiagnosticFailure  | 条件   | DCGM アクティブ診断テストスイートからのテストケースが失敗しました。                                                               | なし      |



## ランタイムノードのヘルスの問題

| 名前                     | 緊急度  | 説明                                                                        |
|------------------------|------|---------------------------------------------------------------------------|
| PodStuckTerminating    | 条件   | ポッドが終了しているか、または長時間停止していました。これは、ポッドの状態の進行を妨げる CRI エラーが原因で発生する可能性があります。     |
| %sRepeatedRestart      | イベント | ノード上の任意の systemd サービスを再起動します (先頭文字を大文字指定したユニット名を使用してフォーマット)。              |
| ContainerRuntimeFailed | イベント | コンテナランタイムはコンテナの作成に失敗しました。繰り返し発生する場合は、報告された問題に関連している可能性があります。              |
| KubeletFailed          | イベント | kubelet が失敗状態になりました。                                                      |
| LivenessProbeFailures  | イベント | ライブネスプローブの障害が検出されました。アプリケーションコードの問題や、繰り返し発生する場合はタイムアウト値が不十分である可能性を示しています。 |
| ReadinessProbeFailures | イベント | レディネスプローブの障害が検出されました。アプリケーションコードの問題や、繰り返し発生する場合はタイムア                      |

| 名前                   | 緊急度  | 説明                      |
|----------------------|------|-------------------------|
|                      |      | ウト値が不十分である可能性を示しています。   |
| ServiceFailedToStart | イベント | systemd ユニットの起動に失敗しました。 |

## ストレージノードのヘルスの問題

| 名前                         | 緊急度  | 説明                                                                                                          |
|----------------------------|------|-------------------------------------------------------------------------------------------------------------|
| XFSSmallAverageClusterSize | 条件   | XFS Average Cluster サイズが小さいため、空き領域が過剰に断片化され、inode や空き領域が使用可能であってもファイルの作成が妨げられる可能性があります。                     |
| EtcHostsMountFailed        | イベント | kubelet-container オペレーション中にユーザーデータを /var/lib/kubelet/pods に再マウントしたため、kubelet が生成した /etc/hosts のマウントに失敗しました。 |
| IODelays                   | イベント | 入力または出力の遅延がプロセスで検出されました。過剰な場合は入力/出力プロビジョニングが不十分である可能性があります。                                                 |
| KubeletDiskUsageSlow       | イベント | Kubelet は、ファイルシステムにアクセスしようとしている時にディスクの使用量が遅いと報告しています。ディスク                                                   |

| 名前 | 緊急度 | 説明                                  |
|----|-----|-------------------------------------|
|    |     | の入力/出力不足やファイルシステムの問題を示している可能性があります。 |

## ノードのヘルスステータスを表示する

このトピックでは、Amazon EKS クラスターのノードのヘルスステータスをモニタリングするために使用可能なツールと方法について説明します。この情報には、ノードレベルの問題の特定と診断に役立つノードの状態、イベント、検出ケースが取り上げられます。ここで説明するコマンドとパターンを使用して、ノードのヘルスリソースを検査し、ステータス条件を解明し、ノードイベントを分析して運用上のトラブルシューティングを行います。

すべてのノードに対して Kubernetes コマンドを使用して、ノードのヘルス情報を取得できます。また、Amazon EKS Auto Mode または Amazon EKS マネージドアドオンを介してノードモニタリングエージェントを使用すると、トラブルシューティングに役立つさまざまなノードシグナルが得られます。ノードモニタリングエージェントによって検出されたヘルス問題の説明は、オブザーバビリティダッシュボードでも確認できます。詳細については「[the section called “ノードのヘルス”](#)」を参照してください。

### ノードの状態

ノードの状態は、インスタンスの置き換えや再起動などの修復アクションを必要とするターミナルの問題を表します。

すべてのノードの状態を取得するには:

```
kubectl get nodes -o 'custom-
columns=NAME:.metadata.name,CONDITIONS:.status.conditions[*].type,STATUS:.status.conditions[*].
```

特定のノードの詳細な状態を取得するには

```
kubectl describe node node-name
```

正常なノードの状態の出力の例:

```
- lastHeartbeatTime: "2024-11-21T19:07:40Z"
 lastTransitionTime: "2024-11-08T03:57:40Z"
 message: Monitoring for the Networking system is active
 reason: NetworkingIsReady
 status: "True"
 type: NetworkingReady
```

ネットワーク形成の問題がある異常なノードの状態の例:

```
- lastHeartbeatTime: "2024-11-21T19:12:29Z"
 lastTransitionTime: "2024-11-08T17:04:17Z"
 message: IPAM-D has failed to connect to API Server which could be an issue with
 IPTable rules or any other network configuration.
 reason: IPAMDNotReady
 status: "False"
 type: NetworkingReady
```

## ノードイベント

ノードイベントは、一時的な問題または最適ではない設定を示します。

ノードモニタリングエージェントによって報告されたすべてのイベントを取得するには

ノードモニタリングエージェントが使用可能になったら、次のコマンドを実行できます。

```
kubectl get events --field-selector=reportingComponent=eks-node-monitoring-agent
```

サンプル出力:

| LAST SEEN         | TYPE    | REASON     | OBJECT                                            |
|-------------------|---------|------------|---------------------------------------------------|
| MESSAGE           |         |            |                                                   |
| 4s                | Warning | SoftLockup | node/ip-192-168-71-251.us-west-2.compute.internal |
| CPU stuck for 23s |         |            |                                                   |

すべてのノードのイベントを取得するには

```
kubectl get events --field-selector involvedObject.kind=Node
```

特定のノードのイベントを取得するには

```
kubectl get events --field-selector involvedObject.kind=Node,involvedObject.name=node-name
```

イベントをリアルタイムで監視するには

```
kubectl get events -w --field-selector involvedObject.kind=Node
```

イベント出力の例:

| LAST SEEN | TYPE    | REASON         | OBJECT      | MESSAGE                           |
|-----------|---------|----------------|-------------|-----------------------------------|
| 2m        | Warning | MemoryPressure | Node/node-1 | Node experiencing memory pressure |
| 5m        | Normal  | NodeReady      | Node/node-1 | Node became ready                 |

一般的なトラブルシューティングのコマンド

```
Get comprehensive node status
kubectl get node node-name -o yaml

Watch node status changes
kubectl get nodes -w

Get node metrics
kubectl top node
```

## kubectl と S3 を使用してマネージドノードのノードログを取得する

ノードモニタリングエージェントを備えた Amazon EKS マネージドノードのノードログを取得する方法について説明します。

### 前提条件

以下があることを確認します。

- ノードモニタリングエージェントを備えた既存の Amazon EKS クラスター。詳細については、[「the section called “ノードのヘルス”」](#)を参照してください。
- クラスターと通信するようにインストールおよび設定された kubectl コマンドラインツール。
- S3 バケットとオブジェクトを作成するのに十分なアクセス許可が付与された状態でインストールおよびログインした AWS CLI。

- インストール済みの Python 3 最新バージョン
- インストール済みの AWS SDK for Python 3、Boto 3。

## ステップ 1: S3 バケットの宛先を作成する (任意)

ログを格納する S3 バケットがまだない場合は、作成します。次の AWS CLI コマンドを使用します。バケットのデフォルトは `private` アクセスコントロールリストです。`bucket-name` はユーザーが選択した一意のバケット名に置き換えます。

```
aws s3api create-bucket --bucket bucket-name
```

## ステップ 2: HTTP Put の事前署名された S3 URL を作成する

Amazon EKS は、指定した URL に対して HTTP PUT オペレーションを実行してノードログを返します。このチュートリアルでは、事前署名された S3 HTTP PUT URL を生成します。

ログは gzip tarball として `.tar.gz` 拡張子で返されます。

### Note

AWS API または SDK を使用して、EKS がログファイルをアップロードするための事前署名された S3 アップロード URL を作成する必要があります。AWS CLI を使用して事前署名された S3 アップロード URL を作成することはできません。

1. ログを格納するバケット内の場所を決定します。例えば、キーとして `2024-11-12/logs1.tar.gz` を使用できます。
2. ファイル `presign-upload.py` に次の Python コードを保存します。`<bucket-name>` と `<key>` を置き換えます。キーは `.tar.gz` で終わる必要があります。

```
import boto3; print(boto3.client('s3').generate_presigned_url(
 ClientMethod='put_object',
 Params={'Bucket': '<bucket-name>', 'Key': '<key>'},
 ExpiresIn=1000
))
```

3. 次を使用してスクリプトを実行します

```
python presign-upload.py
```

4. URL 出力をメモします。この値は次のステップで `http-put-destination` として使用します。

詳細については、AWS Boto3 SDK for Python ドキュメントの「[Generate a presigned URL to upload a file](#)」を参照してください。

### ステップ 3: NodeDiagnostic リソースを作成する

ログを収集するノードの名前を特定します。

リソースの名前としてノードの名前を使用する NodeDiagnostic マニフェストを作成し、HTTP PUT URL の宛先を指定します。

```
apiVersion: eks.amazonaws.com/v1alpha1
kind: NodeDiagnostic
metadata:
 name: node-name
spec:
 logCapture:
 destination: http-put-destination
```

マニフェストをクラスターに適用します。

```
kubectl apply -f nodediagnostic.yaml
```

NodeDiagnostic リソースを記述することで、コレクションのステータスを確認できます。

- ステータスが `Success` または `SuccessWithErrors` の場合は、タスクが完了し、ログが指定された宛先にアップロードされたことを示します (`SuccessWithErrors` は、一部のログが欠落している可能性があることを示します)。
- ステータスが `Failure` の場合は、アップロード URL が正しい形式であり、有効期限が切れていないことを確認してください。

```
kubectl describe nodediagnostics.eks.amazonaws.com/node-name
```

## ステップ 4: S3 からログをダウンロードする

ログをダウンロードする前に約 1 分待ちます。そして、S3 CLI を使用してログをダウンロードします。

```
Once NodeDiagnostic shows Success status, download the logs
aws s3 cp s3://bucket-name/key ./node-logs.tar.gz
```

## ステップ 5: NodeDiagnostic リソースをクリーンアップする

- NodeDiagnostic リソースは自動的に削除されません。ログアーティファクトを取得したら、自分でクリーンアップする必要があります。

```
Delete the NodeDiagnostic resource
kubectl delete nodediagnostics.eks.amazonaws.com/node-name
```

# Amazon EKS Hybrid Nodes の概要

Amazon EKS Hybrid Nodes では、オンプレミスおよびエッジインフラストラクチャを Amazon EKS クラスターのノードとして使用できます。AWS は、Amazon EKS クラスターの AWS ホスト型 Kubernetes コントロールプレーンを管理し、オンプレミス環境またはエッジ環境で実行されるハイブリッドノードを管理します。これにより、Kubernetes 管理が環境全体で統合され、Kubernetes コントロールプレーン管理がオンプレミスおよびエッジアプリケーションの AWS にオフロードされます。

Amazon EKS Hybrid Nodes は、オンプレミスのハードウェアまたは仮想マシンと連携し、アプリケーションを実行する必要がある場所に Amazon EKS の効率、スケーラビリティ、可用性をもたらします。Amazon EKS Hybrid Nodes では、Amazon EKS アドオン、Amazon EKS Pod Identity、クラスターアクセスエントリ、クラスターインサイト、Kubernetes バージョンの延長サポートといったさまざまな Amazon EKS 機能を使用できます。Amazon EKS Hybrid Nodes は、AWS Systems Manager、AWS IAM Roles Anywhere、Amazon Managed Service for Prometheus、Amazon CloudWatch、Amazon GuardDuty などの AWS サービスとネイティブに統合され、モニタリング、ログ記録、ID 管理を一元管理できます。

Amazon EKS Hybrid Nodes では、前払いの義務や最低料金はなく、ハイブリッドノードの vCPU リソースが Amazon EKS クラスターにアタッチされると、1 時間ごとに課金されます。料金の詳細については、「[Amazon EKS の料金表](#)」を参照してください。



## 機能

EKS Hybrid Nodes には、次の高度な機能があります。

- マネージド型 Kubernetes のコントロールプレーン: AWS がホストする、EKS クラスターの Kubernetes コントロールプレーンは AWS が管理し、オンプレミス環境またはエッジ環境で実行されるハイブリッドノードはユーザーが管理します。これにより、Kubernetes 管理が環境全体で統合され、Kubernetes コントロールプレーン管理がオンプレミスおよびエッジアプリケーションの AWS にオフロードされます。Kubernetes コントロールプレーンを AWS に移動させることで、ユーザーはアプリケーションのオンプレミス容量を節約し、ワークロードのスケーリングは Kubernetes コントロールプレーンに任せることができます。
- 一貫性のある EKS エクスペリエンス: ほとんどの EKS 機能は EKS Hybrid Nodes でサポートされており、EKS アドオン、EKS Pod Identity、クラスターアクセスエントリ、クラスターインサイト、Kubernetes バージョンの拡張サポートその他の、オンプレミス環境とクラウド環境を横断する、一貫性のある EKS エクスペリエンスを実現しています。EKS Hybrid Nodes でサポートされている EKS アドオンの詳細については、「[the section called “CNI アドオンの設定”](#)」を参照してください。
- オブザーバビリティと ID 管理を一元化: EKS Hybrid Nodes は、AWS Systems Manager、AWS IAM Roles Anywhere、Amazon Managed Service for Prometheus、Amazon CloudWatch などの AWS サービスとネイティブに統合されており、モニタリング、ログ記録、ID 管理を一元的に行えます。
- Burst-to-cloud またはオンプレミス容量の追加: EKS クラスターを 1 つ使うことで、AWS リージョン、AWS Local Zones、AWS Outposts のいずれかでハイブリッドノードおよびノードを実行し、EKS クラスターに Burst-to-cloud する、またはオンプレミス容量を追加することができます。詳細については、「[Considerations for mixed mode clusters](#)」を参照してください。
- 柔軟なインフラストラクチャ: EKS Hybrid Nodes は Bring Your Own Infrastructure のアプローチに従い、ユーザーがハイブリッドノードに使用しているインフラストラクチャに依存しません。ハイブリッドノードを物理マシンまたは仮想マシン、および x86 と ARM のアーキテクチャで実行できるため、ハイブリッドノードで実行されているオンプレミスワークロードを、さまざまなインフラストラクチャタイプに移行することができます。
- 柔軟なネットワーキング: EKS Hybrid Nodes を使用すると、EKS コントロールプレーンとハイブリッドノード間の通信は、クラスターの作成時にユーザーが渡す VPC とサブネットを介してルーティングされます。これは、コントロールプレーンからノードへのネットワーキング向けに[既存のメカニズム](#)の上に構築されます。これにより、ユーザーが希望する、オンプレミスネットワー

クを AWS の VPC に接続する方法に、柔軟に対応できます。AWS Site-to-Site VPN、AWS Direct Connect、あるいはユーザー独自の VPN ソリューションなど、[文書化されたオプション](#)を複数利用でき、ユースケースに最適な方法を選択できます。

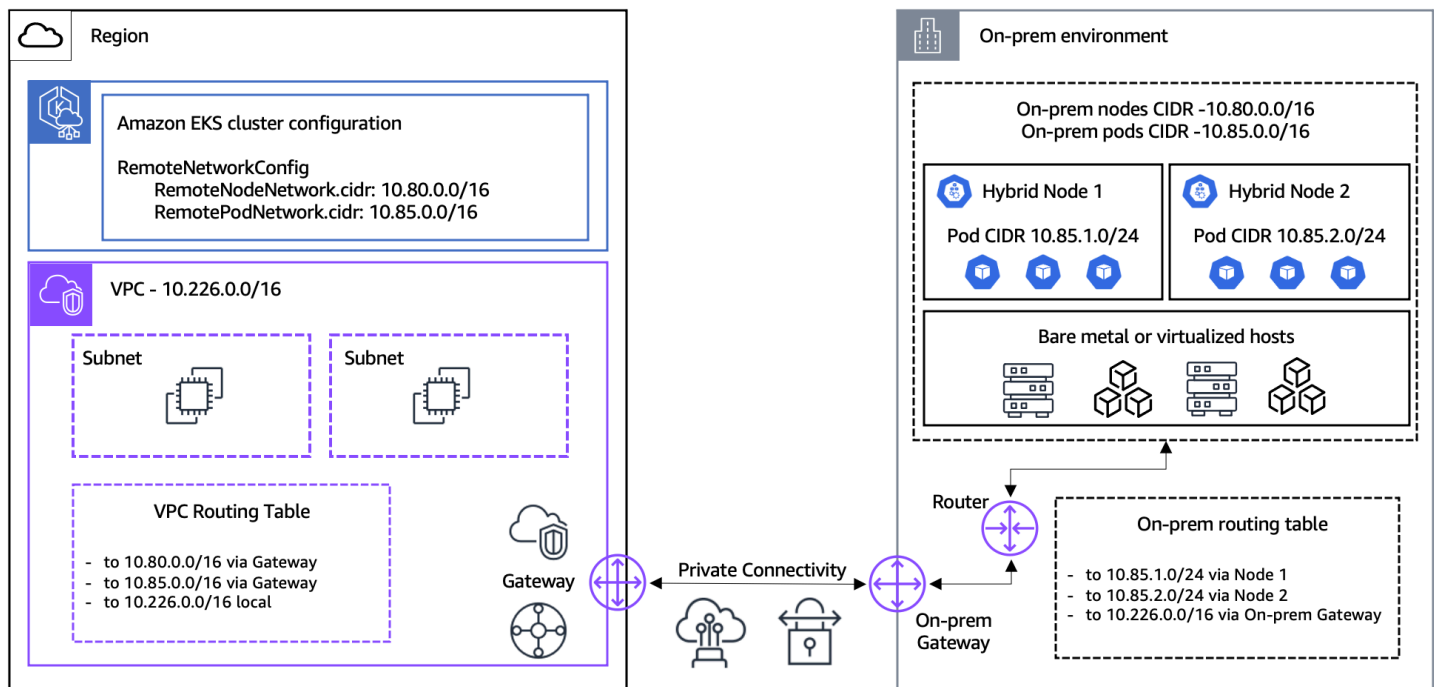
## 一般的な考慮事項

- EKS Hybrid Nodes は、新規の EKS クラスターでも既存の EKS クラスターでも使用できます。
- EKS Hybrid Nodes は、AWS GovCloud (米国) リージョンと AWS 中国リージョンを除くすべての AWS リージョンで使用できます。
- EKS Hybrid Nodes には、オンプレミス環境と AWS の間に信頼性の高い接続が必要です。EKS Hybrid Nodes は、切断された環境、中断しているまたは一時的に停止している環境、制限 (DDIL) されている環境には適していません。DDIL 環境で実行している場合は、[Amazon EKS Anywhere](#)を検討してください。
- EKS Hybrid Nodes をクラウドインフラストラクチャ (AWS リージョン、AWS Local Zones、AWS Outposts) やその他のクラウドで実行することは、サポートされていません。Amazon EC2 インスタンスでハイブリッドノードを実行すると、ハイブリッドノード料金が課金されます。
- ハイブリッドノードの請求は、ノードが EKS クラスターに追加されたとに開始し、ノードがクラスターから削除されたときに停止します。ハイブリッドノードを使用しないときは、必ず EKS クラスターからこれを削除してください。

## ハイブリッドノードの前提条件のセットアップ

Amazon EKS Hybrid Nodes を使用するには、オンプレミス環境と AWS 間のプライベート接続、サポートされているオペレーティングシステムを持つベアメタルサーバーまたは仮想マシン、AWS IAM Roles Anywhere または AWS Systems Manager (SSM) ハイブリッドアクティベーションを設定する必要があります。ハイブリッドノードのライフサイクル全体を通じてこれらの前提条件を管理するのはお客様の責任となります。

- オンプレミス環境と AWS 間のハイブリッドネットワーク接続
- 物理マシンまたは仮想マシンの形式のインフラストラクチャ
- ハイブリッドノードと互換性のあるオペレーティングシステム
- オンプレミスの IAM 認証情報プロバイダーが設定されました



## ハイブリッドネットワーク接続

Amazon EKS コントロールプレーンとハイブリッドノード間の通信は、クラスターの作成時に渡される VPC とサブネットを介してルーティングされます。これは、コントロールプレーンからノードへのネットワーク形成のために Amazon EKS の[既存のメカニズム](#)上に構築されます。オンプレミス環境と VPC の接続には、AWS Site-to-Site VPN、AWS Direct Connect、独自の VPN 接続など、[文書化されたオプション](#)がいくつか利用可能です。ハイブリッドネットワーク接続にこれらのソリューションを使用する方法の詳細については、「[AWS Site-to-Site VPN ユーザーガイド](#)」および「[AWS Direct Connect ユーザーガイド](#)」を参照してください。

最適なエクスペリエンスを実現するために、AWS では、AWS リージョンへのハイブリッドノード接続に、少なくとも 100 Mbps で、最大 200 ミリ秒のラウンドトリップレイテンシーの信頼性の高いネットワーク接続を推奨しています。帯域幅とレイテンシーの要件は、アプリケーションのイメージサイズ、アプリケーションの伸縮性、モニタリングとログ記録の設定、他の AWS サービスに保存されているデータへのアクセスに関するアプリケーションの依存関係など、ハイブリッドノードの数とワークロードの特性によって異なります。本番環境にデプロイする前に、独自のアプリケーションと環境でテストして、ネットワーク設定がワークロードの要件を満たしているか確認することをお勧めします。

## オンプレミスのネットワーク設定

Amazon EKS コントロールプレーンからオンプレミス環境へのインバウンドネットワークアクセスを有効にして、Amazon EKS コントロールプレーンが、ハイブリッドノードで実行されている kubelet と通信できるようにし、オプションでハイブリッドノードで実行されているウェブフックと通信できるようにする必要があります。さらに、Amazon EKS コントロールプレーンと通信するには、ハイブリッドノードとハイブリッドノードで実行されているコンポーネントのアウトバウンドネットワークアクセスを有効にする必要があります。この通信は、AWS Direct Connect、AWS Site-to-Site VPN、または独自の VPN 接続に対して完全にプライベートなままになるように設定できます。ファイアウォールおよびオンプレミス環境で有効にする必要があるポートとプロトコルの完全なリストについては、「[the section called “ネットワークを準備する”](#)」を参照してください。

オンプレミスノードおよびポッドネットワークに使用する Classless Inter-Domain Routing (CIDR) 範囲は、IPv4 RFC1918 アドレスの範囲を使用する必要があります。ハイブリッドノード対応の Amazon EKS クラスターを作成するときは、オンプレミスノードとオプションでポッド CIDR を渡して、Amazon EKS コントロールプレーンからハイブリッドノードへの通信、およびハイブリッドノードで実行されているリソースへの通信を有効にします。オンプレミスルーターは、オンプレミスノードとオプションでポッドへのルートで設定する必要があります。ボーダーゲートウェイプロトコル (BGP) または静的設定を使用して、ポッド IP を自分のルーターにアドバタイズできます。

## EKS クラスター設定

レイテンシーを最小限に抑えるには、オンプレミス環境またはエッジ環境に最も近い AWS リージョンに Amazon EKS クラスターを作成することをお勧めします。Amazon EKS クラスターの作成時に、2 つの API フィールド、RemoteNodeNetwork および RemotePodNetwork を介してオンプレミスのノードとポッドの CIDR を渡します。オンプレミスのノードとポッドの CIDR を特定するために、オンプレミスのネットワークチームに相談する必要がある場合があります。ノード CIDR はオンプレミスネットワークから割り当てられ、ポッド CIDR は、CNI のオーバーレイネットワークを使用している場合に使用する Container Network Interface (CNI) から割り当てられます。

オンプレミスノードとポッド CIDR は、VPC 経由でハイブリッドノードで実行されている kubelet およびポッドにトラフィックをルーティングするように Amazon EKS コントロールプレーンを設定するために使用されます。クラスター作成時に渡す VPC CIDR、または Amazon EKS クラスターのサービス IPv4 構成と互いに重複することはできません。ポッド CIDR はオプションです。CNI がネットワークアドレス変換 (NAT) を使用しない場合、またはポッドトラフィックがオンプレミスホストから出るときにポッド IP アドレスをマスカレードする場合は、ポッド CIDR を設定する必要があります。さらに、ハイブリッドノードで Kubernetes ウェブフックを実行している場合は、ポッド

CIDR を設定する必要があります。例えば、AWS Distro for Open Telemetry (ADOT) はウェブフックを使用します。

Amazon EKS Kubernetes API サーバーエンドポイントには、パブリックまたはプライベートのエンドポイントアクセスを使用することをお勧めします。「Public and Private (パブリックとプライベート)」を選択した場合、Amazon EKS Kubernetes API サーバーエンドポイントは、常に VPC の外部で稼働しているハイブリッドノードのパブリック IP に解決されるため、ハイブリッドノードがクラスターに参加できなくなる可能性があります。Amazon EKS Kubernetes API サーバーエンドポイントには、パブリックまたはプライベートのエンドポイントアクセスを使用できます。「Public and Private (パブリックとプライベート)」を選択することはできません。パブリックエンドポイントアクセスを使用すると、Kubernetes API サーバーエンドポイントはパブリック IP に解決され、ハイブリッドノードから Amazon EKS コントロールプレーンへの通信はインターネット経由でルーティングされます。プライベートエンドポイントアクセスを選択すると、Kubernetes API サーバーエンドポイントはプライベート IP に解決され、ハイブリッドノードから Amazon EKS コントロールプレーンへの通信は、プライベート接続リンク (ほとんどの場合 AWS Direct Connect または AWS Site-to-Site VPN) を介してルーティングされます。

## VPC の構成

Amazon EKS クラスター作成時に渡す VPC のルーティングテーブルに、オンプレミスノード用のルートを設定し、オプションで仮想プライベートゲートウェイ (VGW) またはトランジットゲートウェイ (TGW) をターゲットとするポッドネットワークを設定する必要があります。次に例を示します。REMOTE\_NODE\_CIDR と REMOTE\_POD\_CIDR をオンプレミスのネットワークの値に置き換えます。

| デスティネーション        | ターゲット            | 説明                                   |
|------------------|------------------|--------------------------------------|
| 10.226.0.0/16    | ローカル             | VPC 内の VPC ルートへのローカルトラフィック           |
| REMOTE_NODE_CIDR | tgw-abcdef123456 | オンプレミスノード CIDR、トラフィックを TGW にルーティングする |
| REMOTE_POD_CIDR  | tgw-abcdef123456 | オンプレミスポッド CIDR、トラフィックを TGW にルーティングする |

## セキュリティグループの構成

クラスターを作成すると、Amazon EKS により `eks-cluster-sg-<cluster-name>-<uniqueID>` という名前のセキュリティグループが作成されます。このクラスターセキュリティグループのインバウンドルールは変更できませんが、アウトバウンドルールは制限できます。クラスターに追加のセキュリティグループを追加して、kubelet と、オプションでハイブリッドノードで実行されているウェブフックが Amazon EKS コントロールプレーンに接続できるようにする必要があります。この追加のセキュリティグループに必要なインバウンドルールを以下に示します。REMOTE\_NODE\_CIDR と REMOTE\_POD\_CIDR をオンプレミスのネットワークの値に置き換えます。

| 名前               | セキュリティグループのルール ID | IP バージョン | タイプ   | プロトコル | ポート範囲 | ソース              |
|------------------|-------------------|----------|-------|-------|-------|------------------|
| オンプレミスノードのインバウンド | sgr-abcdef123456  | IPv4     | HTTPS | TCP   | 443   | REMOTE_NODE_CIDR |
| オンプレミスポッドのインバウンド | sgr-abcdef654321  | IPv4     | HTTPS | TCP   | 443   | REMOTE_POD_CIDR  |

## インフラストラクチャ

ハイブリッドノードとして使用するには、ベアメタルサーバーまたは仮想マシンが必要です。ハイブリッドノードは基盤となるインフラストラクチャに依存せず、x86 および ARM アーキテクチャに対応しています。Amazon EKS Hybrid Nodes は「独自のインフラストラクチャを導入する」アプローチに従います。ここでは、ハイブリッドノードに使用するベアメタルサーバーまたは仮想マシンのプロビジョニングと管理はお客様の責任となります。厳密な最小リソース要件はありませんが、ハイブリッドノードには少なくとも 1 つの vCPU と 1GiB RAM を持つホストを使用することをお勧めします。



## オペレーティングシステム

Amazon Linux 2023 (AL2023)、Ubuntu、および RHEL は、ハイブリッドノードのノードオペレーティングシステムとして継続的に検証されます。AWS は、これらのオペレーティングシステムとのハイブリッドノード統合をサポートしていますが、オペレーティングシステム自体にはサポートは提供していません。AL2023 は、Amazon EC2 の外部で実行される場合、AWS サポートプランの対象外です。AL2023 はオンプレミスの仮想化環境でのみ使用できます。詳細については、「[Amazon Linux 2023 ユーザーガイド](#)」を参照してください。

オペレーティングシステムのプロビジョニングと管理はお客様の責任となります。ハイブリッドノードを初めてテストする場合、プロビジョニング済みのホストで Amazon EKS Hybrid Nodes CLI (nodeadm) を実行するのが最も簡単です。本稼働デプロイでは、ホストのスタートアップ時にホストを Amazon EKS クラスターに自動的に結合する systemd サービスとして実行するように設定された nodeadm をゴールデンオペレーティングシステムイメージに含めることをお勧めします。

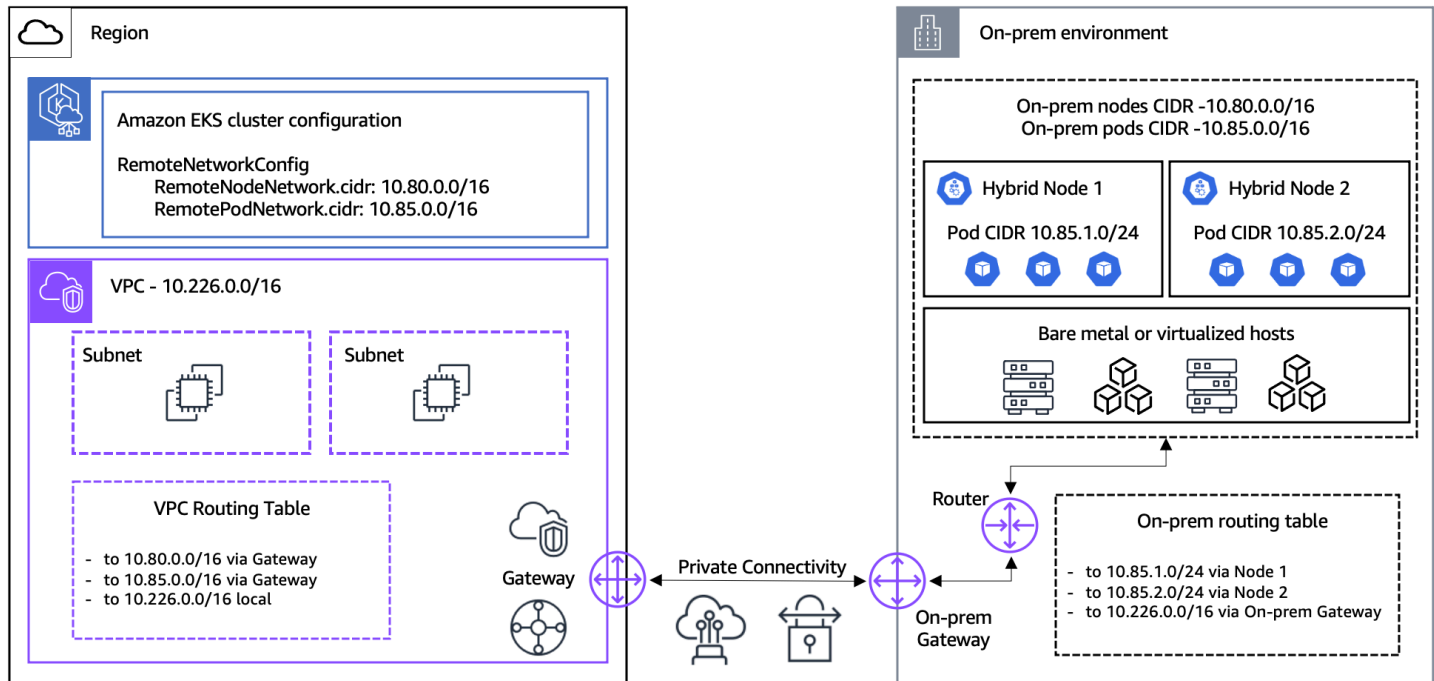
## オンプレミスの IAM 認証情報プロバイダー

Amazon EKS Hybrid Nodes は、AWS SSM ハイブリッドアクティベーションまたは AWS IAM Roles Anywhere によってプロビジョニングされた一時的な IAM 認証情報を使用して、Amazon EKS クラスターで認証します。Amazon EKS Hybrid Nodes CLI (nodeadm) を用いて、AWS SSM ハイブリッドアクティベーションまたは AWS IAM Roles Anywhere を使用する必要があります。認証局 (CA) とオンプレミス環境の証明書を持つ既存の公開鍵基盤 (PKI) がない場合は、AWS SSM ハイブリッドアクティベーションを使用することをお勧めします。既存の PKI と証明書がオンプレミスにある場合は、AWS IAM Roles Anywhere を使用します。

Amazon EC2 で実行されているノードの [the section called “ノードの IAM ロール”](#) と同様に、ハイブリッドノードを Amazon EKS クラスターに結合するために必要なアクセス許可を持つハイブリッドノード IAM ロールを作成します。AWS IAM Roles Anywhere を使用している場合は、AWS IAM Roles Anywhere が Hybrid Nodes IAM Role を引き受けることを許可する信頼ポリシーを設定し、ハイブリッドノード IAM ロールを引き受け可能なロールとして AWS IAM Roles Anywhere プロファイルを設定します。AWS SSM を使用している場合は、AWS SSM がハイブリッドノード IAM ロールを引き受け、ハイブリッドノード IAM ロールを使用してハイブリッドアクティベーションを作成できるようにする信頼ポリシーを設定します。必要なアクセス許可を持つハイブリッドノード IAM ロールを作成する方法については、「[the section called “認証情報を準備する”](#)」を参照してください。

## ハイブリッドノード用のネットワークを準備する

このトピックでは、Amazon EKS クラスターを作成してハイブリッドノードをアタッチする前に設定する必要があるネットワーク設定の概要を説明します。このガイドでは、[AWS Site-to-Site VPN](#)、[AWS Direct Connect](#)、または独自の VPN ソリューションを使用したハイブリッドネットワーク接続の前提条件の要件を満たしていることを前提としています。



### オンプレミスネットワーク設定

#### 最小ネットワーク要件

最適なエクスペリエンスを実現するために、AWS では、AWS リージョンへのハイブリッドノード接続に、少なくとも 100 Mbps で、最大 200 ミリ秒のラウンドトリップレイテンシーの信頼性の高いネットワーク接続を推奨しています。帯域幅とレイテンシーの要件は、アプリケーションのイメージサイズ、アプリケーションの伸縮性、モニタリングとログ記録の設定、他の AWS サービスに保存されているデータへのアクセスに関するアプリケーションの依存関係など、ハイブリッドノードの数とワークロードの特性によって異なります。

#### オンプレミスノードとポッド CIDR

ハイブリッドノードに使用するノードとポッドの CIDR と、それらで実行されているワークロードを特定します。CNI にオーバーレイネットワークを使用している場合、ノード CIDR はオンプレミスネットワークから割り当てられ、ポッド CIDR は Container Network Interface (CNI) から割り当てられます。RemoteNodeNetwork および RemotePodNetwork フィールドを使用して EKS クラス



ターを作成するときは、オンプレミスノード CIDR とオプションでポッド CIDR を入力として渡します。

オンプレミスノードとポッド CIDR ブロックは、以下の要件を満たしている必要があります。

1. 次の IPv4 RFC-1918 のいずれかの範囲内にあること: 10.0.0.0/8、172.16.0.0/12、または 192.168.0.0/16。
2. EKS クラスターの VPC CIDR や Kubernetes サービスの IPv4 CIDR と相互に重複しないこと。

CNI がオンプレミスホストを離れるときにポッドトラフィックのネットワークアドレス変換 (NAT) を実行する場合、ハイブリッドノードのワークロードに対する準備を整えるために、ポッド CIDR をオンプレミスネットワークでルーティング可能にしたり、リモートポッドネットワークで EKS クラスターを設定したりする必要はありません。CNI がオンプレミスホストを離れるときにポッドトラフィックに NAT を使用しない場合は、ハイブリッドノードのワークロードに対する準備を整えるために、オンプレミスネットワークでポッド CIDR をルーティング可能にし、リモートポッドネットワークで EKS クラスターを設定する必要があります。

オンプレミスネットワークでポッド CIDR をルーティング可能にするために使用できる手法がいくつかあります。具体的には、ボーダーゲートウェイプロトコル (BGP)、静的ルート、その他のカスタムルーティングソリューションなどです。BGP は、ルートを独自にまたは手動で設定する必要がある他のソリューションよりもスケーラブルで管理しやすいのでお勧めのソリューションです。AWS は、ハイブリッドノードのポッド CIDR をアダプタイズする Cilium と Calico の BGP 機能をサポートしています。詳細については、「[Configure CNI for hybrid nodes](#)」を参照してください。

ハイブリッドノードでウェブフックを実行している場合は、オンプレミスネットワークでポッド CIDR をルーティング可能にする必要があります。また、EKS コントロールプレーンがハイブリッドノードで実行されているウェブフックと直接通信できるように、リモートポッドネットワークで EKS クラスターを設定する必要があります。ポッド CIDR をオンプレミスネットワークでルーティングできないものの、ウェブフックを実行する必要がある場合は、同じ EKS クラスター内のクラウドノードでウェブフックを実行することをお勧めします。クラウドノードでウェブフックを実行する方法の詳細については、「[Configure webhooks for hybrid nodes](#)」を参照してください。

ハイブリッドノードのインストールとアップグレードに必要なアクセス

ホストにハイブリッドノードの依存関係をインストールするインストールプロセス中は、以下のドメインにアクセスできる必要があります。このプロセスは、オペレーティングシステムイメージを構築するときに 1 回実行すること、ランタイム時に各ホストで実行することでもできます。これには、初期インストールと、ハイブリッドノードの Kubernetes バージョンをアップグレードするが含まれます。

| コンポーネント                                               | [URL]                                                                                       | プロトコル | ポート |
|-------------------------------------------------------|---------------------------------------------------------------------------------------------|-------|-----|
| EKS ノードアーティファクト (S3)                                  | https://hybrid-assets.eks.amazonaws.com                                                     | HTTPS | 443 |
| <a href="#">VPC サービスエンドポイント</a>                       | https://eks. <i>region</i> .amazonaws.com                                                   | HTTPS | 443 |
| <a href="#">ECR サービスエンドポイント</a>                       | https://api.ecr. <i>region</i> .amazonaws.com                                               | HTTPS | 443 |
| EKS ECR エンドポイント                                       | リージョンのエンドポイントについては、「 <a href="#">the section called “Amazon イメージレジストリを表示する”</a> 」を参照してください。 | HTTPS | 443 |
| SSM バイナリエンドポイント <sup>1</sup>                          | https://amazon-ssm- <i>region</i> .s3. <i>region</i> .amazonaws.com                         | HTTPS | 443 |
| <a href="#">SSM サービスエンドポイント</a> <sup>1</sup>          | https://ssm. <i>region</i> .amazonaws.com                                                   | HTTPS | 443 |
| IAM Anywhere バイナリエンドポイント <sup>2</sup>                 | https://rolesanywhere.amazonaws.com                                                         | HTTPS | 443 |
| <a href="#">IAM Anywhere サービスエンドポイント</a> <sup>2</sup> | https://rolesanywhere. <i>region</i> .amazonaws.com                                         | HTTPS | 443 |

**Note**

<sup>1</sup> AWS SSM エンドポイントへのアクセスは、オンプレミスの IAM 認証情報プロバイダーに AWS SSM ハイブリッドアクティベーションを使用している場合のみ必要です。

<sup>2</sup> AWS IAM エンドポイントへのアクセスは、オンプレミスの IAM 認証情報プロバイダーに AWS IAM Roles Anywhere を使用している場合のみ必要です。

**継続的なクラスター運用に必要なアクセス**

クラスターの継続的な運用には、オンプレミスのファイアウォールに以下のネットワークアクセスが必要です。

**Important**

CNI の選択に応じて、CNI ポートに追加のネットワークアクセスルールを設定する必要があります。詳細については、「[Cilium のドキュメント](#)」および「[Calico のドキュメント](#)」を参照してください。

| タイプ   | プロトコル | 方向      | ポート | 送信元                 | デスティネーション                                 | 使用方法                                          |
|-------|-------|---------|-----|---------------------|-------------------------------------------|-----------------------------------------------|
| HTTPS | TCP   | アウトバウンド | 443 | リモート<br>ノード<br>CIDR | EKS クラ<br>スター IP <sup>1</sup>             | Kubelet<br>から<br>Kubernete<br>s API サー<br>バーへ |
| HTTPS | TCP   | アウトバウンド | 443 | リモート<br>ポッド<br>CIDR | EKS クラ<br>スター IP <sup>1</sup>             | Pod から<br>Kubernete<br>s API サー<br>バーへ        |
| HTTPS | TCP   | アウトバウンド | 443 | リモート<br>ノード<br>CIDR | <a href="#">SSM サー<br/>ビスエンド<br/>ポイント</a> | SSM ハイ<br>ブリッドア<br>クティベー<br>ション認証             |

| タイプ   | プロトコル | 方向      | ポート   | 送信元                       | デスティネーション                                   | 使用方法                                                     |
|-------|-------|---------|-------|---------------------------|---------------------------------------------|----------------------------------------------------------|
|       |       |         |       |                           |                                             | 情報の更新と 5 分ごとの SSM ハートビート                                 |
| HTTPS | TCP   | アウトバウンド | 443   | リモートノード CIDR              | <a href="#">IAM Anywhere サービスエンドポイント</a>    | IAM Roles Anywhere 認証情報の更新                               |
| HTTPS | TCP   | アウトバウンド | 443   | リモートポッド CIDR              | <a href="#">STS リージョナルエンドポイント</a>           | Pod から STS エンドポイントへ、IRSA のみ必要                            |
| HTTPS | TCP   | アウトバウンド | 443   | リモートノード CIDR              | <a href="#">Amazon EKS Auth サービスエンドポイント</a> | Amazon EKS Pod Identity のみに必要な Amazon EKS 認証エンドポイントへのノード |
| HTTPS | TCP   | インバウンド  | 10250 | EKS クラスター IP <sup>1</sup> | リモートノード CIDR                                | Kubernetes API サーバーから kubelet へ                          |

| タイプ    | プロトコル   | 方向             | ポート       | 送信元                       | デステイネーション    | 使用方法                                                                                                  |
|--------|---------|----------------|-----------|---------------------------|--------------|-------------------------------------------------------------------------------------------------------|
| HTTPS  | TCP     | インバウンド         | ウェブフックポート | EKS クラスター IP <sup>1</sup> | リモートポッド CIDR | Kubernetes API サーバーからウェブフックへ                                                                          |
| HTTPS  | TCP、UDP | インバウンド、アウトバウンド | 53        | リモートポッド CIDR              | リモートポッド CIDR | Pod から CoreDNS へ。クラウドで CoreDNS のレプリカを少なくとも 1 つ実行する場合は、CoreDNS が実行されている VPC への DNS トラフィックを許可する必要があります。 |
| ユーザー定義 | ユーザー定義  | インバウンド、アウトバウンド | アプリポート    | リモートポッド CIDR              | リモートポッド CIDR | Pod から Pod へ                                                                                          |

**Note**

<sup>1</sup> EKS クラスターの IP。Amazon EKS Elastic Network Interface については、次のセクションを参照してください。

## Amazon EKS ネットワークインターフェイス

Amazon EKS は、クラスターの作成時に渡す VPC 内のサブネットにネットワークインターフェイスをアタッチして、EKS コントロールプレーンと VPC 間の通信を有効にします。Amazon EKS が作成するネットワークインターフェイスは、クラスターの作成後に、Amazon EC2 コンソールまたは AWS CLI で確認できます。Kubernetes バージョンのアップグレードなど、EKS クラスターに変更が適用されると、元のネットワークインターフェイスが削除され、新しいネットワークインターフェイスが作成されます。クラスターの作成時に渡すサブネットに制約付きサブネットサイズを使用することで、Amazon EKS ネットワークインターフェイスの IP 範囲を制限できます。これにより、この既知の制約付き IP セットへのインバウンド/アウトバウンド接続を許可するようにオンプレミスファイアウォールを設定することが容易になります。ネットワークインターフェイスが作成されるサブネットを制御するためには、クラスター作成時に指定するサブネットの数を制限するか、クラスター作成後にサブネットを更新することができます。

Amazon EKS によってプロビジョニングされるネットワークインターフェイスには、Amazon EKS *your-cluster-name* の形式の説明が付けられています。Amazon EKS がプロビジョニングするネットワークインターフェイスの IP アドレスを見つけるために使用できる AWS CLI コマンドについては、以下の例を参照してください。VPC\_ID を、クラスターの作成時に渡す VPC の ID に置き換えます。

```
aws ec2 describe-network-interfaces \
--query 'NetworkInterfaces[?(VpcId == VPC_ID && contains(Description,Amazon
EKS))].PrivateIpAddress'
```

## AWS VPC とサブネットの設定

ハイブリッドノードを持つクラスターには、Amazon EKS の既存の [VPC とサブネットの要件](#)が適用されます。さらに、VPC CIDR はオンプレミスノードおよびポッド CIDR と重複することはできません。オンプレミスノードの VPC ルートテーブルにルートを設定し、オプションでポッド CIDR を設定する必要があります。これらのルートは、ハイブリッドネットワーク接続に使用しているゲートウェイにトラフィックをルーティングするように設定する必要があります。通常、これは仮想プライベートゲートウェイ (VGW) またはトランジットゲートウェイ (TGW) です。TGW または VGW を使用して VPC をオンプレミス環境に接続する場合は、VPC の TGW または VGW アタッチメントを

作成する必要があります。VPC は、DNS ホスト名と DNS 解決がサポートされている必要があります。

AWS CLI を使用する手順は以下のとおりです。これらのリソースは AWS Management Console で、または AWS CloudFormation、AWS CDK、Terraform などの他のインターフェイスでも作成することができます。

### ステップ 1: VPC を作成する

1. 次のコマンドを実行して VPC を作成します。VPC\_CIDR を、IPv4 RFC-1918 (プライベート) または non-RFC-1918 (パブリック) CIDR 範囲 (10.0.0.0/16 など) に置き換えます。注: EKS の要件である DNS 解決は、VPC に対しデフォルトで有効になっています。

```
aws ec2 create-vpc --cidr-block VPC_CIDR
```

2. VPC の DNS ホスト名を有効にします。DNS 解決は、VPC に対しデフォルトで有効になっています。VPC\_ID を、前のステップで作成した VPC の ID に置き換えます。

```
aws ec2 modify-vpc-attribute --vpc-id VPC_ID --enable-dns-hostnames
```

### ステップ 2: サブネットを作成する

少なくとも 2 つのサブネットを作成します。Amazon EKS は、これらのサブネットをクラスターのネットワークインターフェイスに使用します。詳細については、「[サブネットの要件と考慮事項](#)」を参照してください。

1. AWS リージョンのアベイラビリティゾーンは、以下のコマンドで確認できます。us-west-2 を実際のリージョンに置き換えます。

```
aws ec2 describe-availability-zones \
 --query 'AvailabilityZones[?(RegionName == us-west-2)].ZoneName'
```

2. サブネットを作成します。VPC\_ID を VPC の ID に置き換えます。SUBNET\_CIDR を、サブネットの CIDR ブロック (10.0.1.0/24 など) に置き換えます。AZ を、サブネットが作成されるアベイラビリティゾーン (us-west-2a など) に置き換えます。作成するサブネットは、少なくとも 2 つの異なるアベイラビリティゾーンに存在する必要があります。

```
aws ec2 create-subnet \
 --vpc-id VPC_ID --cidr-block SUBNET_CIDR --availability-zone AZ
```

```
--vpc-id VPC_ID \
--cidr-block SUBNET_CIDR \
--availability-zone AZ
```

(オプション) ステップ 3: Amazon VPC Transit Gateway (TGW) または AWS Direct Connect 仮想プライベートゲートウェイ (VGW) を使用して VPC をアタッチする

TGW または VGW を使用している場合は、VPC を TGW または VGW にアタッチします。詳細については、「[Amazon VPC attachments in Amazon VPC Transit Gateways](#)」または「[AWS Direct Connect virtual private gateway associations](#)」を参照してください。

### Transit Gateway

以下のコマンドを実行して、トランジットゲートウェイをアタッチします。VPC\_ID を VPC の ID に置き換えます。SUBNET\_ID1 と SUBNET\_ID2 を、前のステップで作成したサブネットの ID に置き換えます。TGW\_ID を TGW の ID に置き換えます。

```
aws ec2 create-transit-gateway-vpc-attachment \
--vpc-id VPC_ID \
--subnet-ids SUBNET_ID1 SUBNET_ID2 \
--transit-gateway-id TGW_ID
```

### 仮想プライベートゲートウェイ

以下のコマンドを実行して、トランジットゲートウェイをアタッチします。VPN\_ID を VGW の ID に置き換えます。VPC\_ID を VPC の ID に置き換えます。

```
aws ec2 attach-vpn-gateway \
--vpn-gateway-id VPN_ID \
--vpc-id VPC_ID
```

### (オプション) ステップ 4: ルートテーブルを作成する

VPC のメインルートテーブルを変更するか、カスタムルートテーブルを作成できます。以下の手順では、オンプレミスノードとポッド CIDR へのルートを含むカスタムルートテーブルを作成します。詳細については、「[サブネットルートテーブル](#)」を参照してください。VPC\_ID を VPC の ID に置き換えます。



```
aws ec2 create-route-table --vpc-id VPC_ID
```

## ステップ 5: オンプレミスノードとポッドのルートを作成する

オンプレミスの各リモートノードのルートテーブルにルートを作成します。VPC のメインルートテーブルを変更するか、前のステップで作成したカスタムルートテーブルを使用できます。

以下の例は、オンプレミスノードとポッド CIDR のルートを作成する方法を示します。この例では、トランジットゲートウェイ (TGW) を使用して VPC をオンプレミス環境に接続します。複数のオンプレミスノードとポッド CIDR がある場合は、CIDR ごとにステップを繰り返します。

- インターネットゲートウェイまたは仮想プライベートゲートウェイ (VGW) を使用している場合は、`--transit-gateway-id` を `--gateway-id` に置き換えます。
- `RT_ID` を、前のステップで作成したルートテーブルの ID に置き換えます。
- `REMOTE_NODE_CIDR` を、ハイブリッドノードに使用する CIDR 範囲に置き換えます。
- `REMOTE_POD_CIDR` を、ハイブリッドノードで稼働しているポッドに使用する CIDR 範囲に置き換えます。ポッド CIDR 範囲は、オンプレミスのオーバーレイネットワークを最も一般的に使用するコンテナネットワークインターフェイス (CNI) 設定に対応しています。詳細については、「[the section called “CNI を設定する”](#)」を参照してください。
- `TGW_ID` を TGW の ID に置き換えます。

## リモートノードネットワーク

```
aws ec2 create-route \
 --route-table-id RT_ID \
 --destination-cidr-block REMOTE_NODE_CIDR \
 --transit-gateway-id TGW_ID
```

## リモートポッドネットワーク

```
aws ec2 create-route \
 --route-table-id RT_ID \
 --destination-cidr-block REMOTE_POD_CIDR \
 --transit-gateway-id TGW_ID
```

## (オプション) ステップ 6: サブネットをルートテーブルに関連付ける

前のステップでカスタムルートテーブルを作成した場合は、前のステップで作成した各サブネットをカスタムルートテーブルに関連付けます。VPC メインルートテーブルを変更する場合、サブネットは VPC のメインルートテーブルに自動的に関連付けられるため、このステップをスキップできます。

前のステップで作成したサブネットごとに、以下のコマンドを実行します。RT\_ID を、前のステップで作成したルートテーブルに置き換えます。SUBNET\_ID をサブネットの ID に置き換えます。

```
aws ec2 associate-route-table --route-table-id RT_ID --subnet-id SUBNET_ID
```

## クラスターセキュリティグループの設定

クラスターの継続的な運用には、EKS クラスターセキュリティグループの以下のアクセスが必要です。

| タイプ   | プロトコル | 方向     | ポート | 送信元                 | デスティネーション | 使用方法                                                                                       |
|-------|-------|--------|-----|---------------------|-----------|--------------------------------------------------------------------------------------------|
| HTTPS | TCP   | インバウンド | 443 | リモート<br>ノード<br>CIDR | 該当なし      | Kubelet<br>から<br>Kubernetes API サー<br>バーへ                                                  |
| HTTPS | TCP   | インバウンド | 443 | リモート<br>ポッド<br>CIDR | 該当なし      | CNI がポッ<br>ドトラ<br>フィック<br>に NAT を<br>使用して<br>いない場合<br>で、ポッド<br>が K8s API<br>サーバーへ<br>のアクセス |

| タイプ   | プロトコル | 方向      | ポート       | 送信元  | デスティネーション    | 使用方法                                                     |
|-------|-------|---------|-----------|------|--------------|----------------------------------------------------------|
|       |       |         |           |      |              | を必要とする場合。                                                |
| HTTPS | TCP   | アウトバウンド | 10250     | 該当なし | リモートノード CIDR | Kubernetes API サーバーから Kubelet へ                          |
| HTTPS | TCP   | アウトバウンド | ウェブフックポート | 該当なし | リモートポッド CIDR | Kubernetes API サーバーからウェブフックへ (ハイブリッドノードでウェブフックを実行している場合) |

インバウンドアクセスルールを使用してセキュリティグループを作成するには、以下のコマンドを実行します。このセキュリティグループは、Amazon EKS クラスターを作成する際に渡す必要があります。デフォルトでは、以下のコマンドは、すべてのアウトバウンドアクセスを許可するセキュリティグループを作成します。アウトバウンドアクセスを上記のルールだけに制限することができます。アウトバウンドルールの制限を検討している場合、変更したルールを本番稼働用のクラスターに適用する前に、すべてのアプリケーションとポッドの接続を徹底的にテストすることをお勧めします。

- 最初のコマンドで、SG\_NAME をセキュリティグループの名前に置き換えます
- 最初のコマンドで、VPC\_ID を前のステップで作成した VPC の ID に置き換えます
- 2 番目のコマンドで、SG\_ID を最初のコマンドで作成したセキュリティグループの ID に置き換えます

- 2 番目のコマンドで、REMOTE\_NODE\_CIDR と REMOTE\_POD\_CIDR をハイブリッドノードとオンプレミスネットワークの値にそれぞれ置き換えます。

```
aws ec2 create-security-group \
 --group-name SG_NAME \
 --description "security group for hybrid nodes" \
 --vpc-id VPC_ID
```

```
aws ec2 authorize-security-group-ingress \
 --group-id SG_ID \
 --ip-permissions '[{"IpProtocol": "tcp", "FromPort": 443, "ToPort": 443,
"IpRanges": [{"CidrIp": "REMOTE_NODE_CIDR"}, {"CidrIp": "REMOTE_POD_CIDR"}]]'
```

## ハイブリッドノード用のオペレーティングシステムを準備する

Bottlerocket、Ubuntu、Red Hat Enterprise Linux (RHEL)、Amazon Linux 2023 (AL2023) は、ハイブリッドノードのノードオペレーティングシステムとして継続的に検証されます。AWS はこれらのオペレーティングシステムとのハイブリッドノード統合をサポートしていますが、Bottlerocket を除き、オペレーティングシステム自体のサポートは提供していません。AL2023 は、Amazon EC2 の外部で実行される場合、AWS サポートプランの対象外です。AL2023 はオンプレミスの仮想化環境でのみ使用できます。詳細については、「[Amazon Linux 2023 ユーザーガイド](#)」を参照してください。

オペレーティングシステムのプロビジョニングと管理はお客様の責任となります。ハイブリッドノードを初めてテストする場合、プロビジョニング済みのホストで Amazon EKS Hybrid Nodes CLI (nodeadm) を実行するのが最も簡単です。本稼働デプロイでは、ホスト起動時にホストを Amazon EKS クラスターに自動的に結合する systemd サービスとして実行するように設定された nodeadm を、オペレーティングシステムイメージに含めることが推奨されます。Bottlerocket を vSphere のノードオペレーティングシステムとして使用している場合は nodeadm を使用する必要はありません。Bottlerocket にはハイブリッドノードに必要な依存関係が既に含まれ、ホストのスタートアップ時に設定するクラスターに自動的に接続されるからです。

### バージョン互換性

以下の表は、ハイブリッドノードのノードオペレーティングシステムとしての使用に互換性があり、検証済みとなっているオペレーティングシステムのバージョンを示しています。オペレーティングシステムのこの表に記載されていないバリエーションやバージョンを使用している場合、そのオペレーティ

ングシステムのバリエーションまたはバージョンとのハイブリッドノードの互換性は、AWS サポートの対象外となります。ハイブリッドノードは基盤となるインフラストラクチャに依存せず、x86 および ARM アーキテクチャに対応しています。

| オペレーティングシステム             | バージョン                                                 |
|--------------------------|-------------------------------------------------------|
| Amazon Linux             | Amazon Linux 2023 (AL2023)                            |
| Bottlerocket             | v1.37.0 以降、Kubernetes v1.28 以降を実行している VMware のバリエーション |
| Ubuntu                   | Ubuntu 20.04、Ubuntu 22.04、Ubuntu 24.04                |
| Red Hat Enterprise Linux | RHEL 8、RHEL 9                                         |

## オペレーティングシステムに関する考慮事項

### 全般

- Amazon EKS Hybrid Nodes CLI (nodeadm) を使用すると、ハイブリッドノードのコンポーネントおよび依存関係のインストールと設定を簡素化できます。nodeadm install プロセスは、オペレーティングシステムイメージのビルドパイプライン中、または各オンプレミスホストの実行時に実行できます。nodeadm がインストールするコンポーネントの詳細については、「[the section called “ハイブリッドノード用 nodeadm”](#)」を参照してください。
- オンプレミス環境でプロキシを使用してインターネットにアクセスしている場合、インストールおよびアップグレードのプロセスでパッケージマネージャーがプロキシを使用するように設定するために、追加のオペレーティングシステム設定が必要です。手順については「[the section called “プロキシを設定する”](#)」を参照してください。

### Bottlerocket

- Bottlerocket ノードを接続するときのステップとツールは他のオペレーティングシステムのステップとは異なるため、「[the section called “ハイブリッドノードを接続する”](#)」ではなく「[the section called “Bottlerocket を実行しているハイブリッドノードの接続”](#)」に記載しています。
- Bottlerocket のステップでは、ハイブリッドノード CLI のツールである nodeadm は使用しません。

- EKS Hybrid Nodes でサポートされているのは、Bottlerocket のバージョンが v1.37.0 以降である VMware のバリエーションのみです。Bottlerocket の VMware のバリエーションは、Kubernetes のバージョン v1.28 以降で使用できます。[その他の Bottlerocket のバリエーション](#)は、ハイブリッドノードのオペレーティングシステムとしてサポートされていません。注: Bottlerocket の VMware のバリエーションを使用できるのは、x86\_64 のアーキテクチャのみです。

## Containerd

- Containerd は標準の Kubernetes コンテナランタイムであり、ハイブリッドノードのみならず、あらゆるコンピューティングタイプの Amazon EKS ノードの依存関係の 1 つです。Amazon EKS Hybrid Nodes CLI (nodeadm) は、nodeadm install プロセス中に containerd のインストールを試みます。--containerd-source コマンドラインオプションを使用して、nodeadm install の実行時に containerd のインストールを設定できます。有効なオプションは、none、distro、docker です。RHEL を使用している場合、distro は有効なオプションではなく、Docker のリポジトリから containerd ビルドをインストールするよう nodeadm を設定するか、containerd を手動でインストールできます。AL2023 または Ubuntu を使用する場合、nodeadm はオペレーティングシステムのディストリビューションから containerd をインストールするのがデフォルトになります。nodeadm で containerd をインストールしない場合は、--containerd-source none オプションを使用します。

## Ubuntu

- Ubuntu 20.04 を使用している場合は、認証情報プロバイダーとして AWS Systems Manager のハイブリッドアクティベーションを使用する必要があります。AWS IAM Roles Anywhere は、Ubuntu 20.04 ではサポートされていません。
- Ubuntu 24.04 を使用している場合、ポッドを適切に終了させるための修正を適用するために、containerd のバージョンを更新するか、AppArmor の設定を変更する必要がある場合があります。「[Ubuntu #2065423](#)」を参照してください。AppArmor プロファイルに変更を適用するには、再起動が必要です。Ubuntu 24.04 の最新バージョンには、修正が含まれた更新版の containerd バージョン (containerd バージョン 1.7.19 以降) がパッケージマネージャーに含まれています。

## RHEL

- RHEL 8 を使用している場合は、認証情報プロバイダーとして AWS Systems Manager のハイブリッドアクティベーションを使用する必要があります。AWS IAM Roles Anywhere は RHEL 8 ではサポートされていません。

## ARM

- ARM ハードウェアを使用している場合、EKS kube-proxy アドオンのバージョン 1.31 以降を実行するためには、Cryptography Extension (ARMv8.2+crypto) を搭載した ARMv8.2 準拠のプロセッサが必要です。Cortex-A72 ベースのプロセッサのほか、Raspberry Pi 5 より前のすべての Raspberry Pi システムもこの要件を満たしていません。回避策として、2026 年 7 月に延長サポートが終了するまで、EKS kube-proxy アドオンのバージョン 1.30 をこれまで通り使用できます。「[the section called “Amazon EKS Kubernetes リリースカレンダー”](#)」を参照するか、アップストリームからカスタム kube-proxy イメージを使用してください。
- kube-proxy ログに以下のエラーメッセージが記録されていれば、この非互換性が生じていることになります。

```
Fatal glibc error: This version of Amazon Linux requires a newer ARM64 processor
compliant with at least ARM architecture 8.2-a with Cryptographic extensions. On EC2
this is Graviton 2 or later.
```

## オペレーティングシステムイメージの構築

Amazon EKS は、nodeadm を含み、ホスト起動時に実行するように設定されたオペレーティングシステムイメージを作成するために使用できる [Packer テンプレートの例](#)を提供しています。このプロセスは、各ホストでハイブリッドノードの依存関係を個別にプルすることを避け、ハイブリッドノードのブートストラッププロセスを自動化するために推奨されています。Packer テンプレートの例は、Ubuntu 22.04、Ubuntu 24.04、RHEL 8、または RHEL 9 ISO イメージで使用でき、イメージの出力は OVA、Qcow2、または raw の形式で行えます。

## 前提条件

Packer テンプレートの例を使用する前に、Packer を実行しているマシンに以下がインストールされている必要があります。

- Packer バージョン 1.11.0 以降。Packer のインストール手順については、「[Packer documentation](#)」の「[install Packer](#)」を参照してください。
- OVA を構築する場合、VMware vSphere プラグイン 1.4.0 以降
- Qcow2 または raw イメージを構築する場合、QEMU プラグインバージョン 1.x

## 環境変数の設定

Packer ビルドを実行する前に、Packer を実行しているマシンで次の環境変数を設定します。

## 全般

いずれのオペレーティングシステムおよび出力形式においても、イメージを構築する際には、以下の環境変数を設定する必要があります。

| 環境変数                | タイプ    | 説明                                                                                                                                                                                                                                                                          |
|---------------------|--------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| PKR_SSH_PASSWORD    | String | Packer は、プロビジョニング時に作成されたマシンに SSH 接続する際は、ssh_username および ssh_password 変数を使用します。これは、各 OS のキックスタートファイルまたは user-data ファイル内に初期ユーザーを作成する際に使用されるパスワードと一致する必要があります。デフォルトは、OS に応じて「builder」または「ubuntu」に設定されます。パスワードを設定する際は、必ず対応する ks.cfg または user-data ファイル内のパスワードも一致するように変更してください。 |
| ISO_URL             | String | 使用する ISO の URL。サーバーからダウンロードするウェブリンクでも、ローカルファイルへの絶対パスでもかまいません                                                                                                                                                                                                                |
| ISO_CHECKSUM        | String | 指定された ISO に関連するチェックサム。                                                                                                                                                                                                                                                      |
| CREDENTIAL_PROVIDER | String | ハイブリッドノードの認証情報プロバイダー。有効な値は ssm (SSM ハイブリッド                                                                                                                                                                                                                                  |



| 環境変数         | タイプ    | 説明                                                                                                                                         |
|--------------|--------|--------------------------------------------------------------------------------------------------------------------------------------------|
|              |        | アクティベーション用、デフォルト) と iam (IAM Roles Anywhere 用) です                                                                                          |
| K8S_VERSION  | String | ハイブリッドノードの Kubernetes バージョン (1.31 など)。サポートされている Kubernetes のバージョンについては、「 <a href="#">the section called “Kubernetes バージョン”</a> 」を参照してください。 |
| NODEADM_ARCH | String | nodeadm install 用アーキテクチャ。amd または arm を選択します。                                                                                               |

## RHEL

RHEL を使用している場合は、以下の環境変数を設定する必要があります。

| 環境変数         | タイプ    | 説明                                       |
|--------------|--------|------------------------------------------|
| RH_USERNAME  | String | RHEL サブスクリプションマネージャーのユーザー名               |
| RH_PASSWORD  | String | RHEL サブスクリプションマネージャーのパスワード               |
| RHEL_VERSION | String | 使用する RHEL の ISO のバージョン。有効な値は 8 または 9 です。 |

## Ubuntu

Ubuntu 固有の環境変数は必要ありません。

## vSphere

VMware vSphere OVA を構築する場合は、以下の環境変数を設定する必要があります。

| 環境変数                  | タイプ    | 説明                       |
|-----------------------|--------|--------------------------|
| VSPHERE_SERVER        | String | vSphere のサーバーアドレス        |
| VSPHERE_USER          | String | vSphere のユーザー名           |
| VSPHERE_PASSWORD      | String | vSphere のパスワード           |
| VSPHERE_DATACENTER    | String | vSphere のデータセンター名        |
| VSPHERE_CLUSTER       | String | vSphere のクラスター名          |
| VSPHERE_DATASTORE     | String | vSphere のデータストア名         |
| VSPHERE_NETWORK       | String | vSphere のネットワーク名         |
| VSPHERE_OUTPUT_FOLDER | String | テンプレート用の vSphere の出力フォルダ |

## QEMU

| 環境変数                 | タイプ    | 説明                                         |
|----------------------|--------|--------------------------------------------|
| PACKER_OUTPUT_FORMAT | String | QEMU ビルダーの出力形式。<br>有効な値は、qcow2 および raw です。 |

## テンプレートを検証する

環境変数を設定した後は、ビルドを実行する前に以下のコマンドでテンプレートを検証します。テンプレートに異なる名前を使用している場合は、`template.pkr.hcl` を置き換えてください。

```
packer validate template.pkr.hcl
```

## イメージを構築する

以下のコマンドを使用してイメージを構築し、`-only` フラグを使用してイメージのターゲットとオペレーティングシステムを指定します。テンプレートに異なる名前を使用している場合は、`template.pkr.hcl` を置き換えてください。

### vSphere OVA

#### Note

vSphere で RHEL を使用している場合は、キックスタータファイルを OEMDRV イメージに変換し、起動元の ISO として渡す必要があります。詳細については、EKS ハイブリッドノードの GitHub リポジトリにある「[Packer Readme](#)」を参照してください。

### Ubuntu 22.04 OVA

```
packer build -only=general-build.vsphere-iso.ubuntu22 template.pkr.hcl
```

### Ubuntu 24.04 OVA

```
packer build -only=general-build.vsphere-iso.ubuntu24 template.pkr.hcl
```

### RHEL 8 OVA

```
packer build -only=general-build.vsphere-iso.rhel8 template.pkr.hcl
```

### RHEL 9 OVA

```
packer build -only=general-build.vsphere-iso.rhel9 template.pkr.hcl
```

### QEMU

#### Note

ビルダーのホストと一致しない特定のホストの CPU 用にイメージを構築する場合は、ホストの CPU に対応する名前を「[QEMU](#)」のドキュメントで確認し、以下のコマンドを実行する際にホストの CPU 名を指定した `-cpu` フラグを使用します。

## Ubuntu 22.04 Qcow2/Raw

```
packer build -only=general-build.qemu.ubuntu22 template.pkr.hcl
```

## Ubuntu 24.04 Qcow2/Raw

```
packer build -only=general-build.qemu.ubuntu24 template.pkr.hcl
```

## RHEL 8 Qcow2/Raw

```
packer build -only=general-build.qemu.rhel8 template.pkr.hcl
```

## RHEL 9 Qcow2/Raw

```
packer build -only=general-build.qemu.rhel9 template.pkr.hcl
```

## nodeadm 設定を user-data を通して渡す

cloud-init を介して nodeadm の設定を user-data で渡すことで、ホストの起動時にハイブリッドノードを設定し、自動的に EKS クラスターに接続することができます。以下は、ハイブリッドノードのインフラストラクチャとして VMware vSphere を使用する場合の実装例です。

1. GitHub の [govc readme](#) の手順に従って govc CLI をインストールします。
2. 前のセクションで Packer ビルドを実行してテンプレートをプロビジョニングした後は、以下を使用してテンプレートをクローンし、複数の異なるノードを作成することができます。ハイブリッドノードに使用する新規 VM を作成するたびに、テンプレートをクローンする必要があります。以下のコマンド内の変数は、お使いの環境に応じた値に置き換えます。以下のコマンドの VM\_NAME は、metadata.yaml ファイルを介して VM の名前を注入する際に NODE\_NAME として使用されます。

```
govc vm.clone -vm "/PATH/TO/TEMPLATE" -ds="YOUR_DATASTORE" \
-on=false -template=false -folder=/FOLDER/TO/SAVE/VM "VM_NAME"
```

3. 各新規 VM のテンプレートをクローンした後は、VM の metadata.yaml と userdata.yaml を作成します。VM は同じ userdata.yaml と metadata.yaml を共有でき、以下のステップで VM ごとにこれらを入力していきます。nodeadm の設定は、userdata.yaml の write\_files セクションで作成および定義されます。以下の例では、ハイブリッドノードのオンプレミス認証情報プロバイダーとして AWS SSM ハイブリッドアクティベーションを使用していま

す。nodeadm 設定の詳細については、「[the section called “ハイブリッドノード用 nodeadm”](#)」を参照してください。

userdata.yaml:

```
#cloud-config
users:
 - name: # username for login. Use 'builder' for RHEL or 'ubuntu' for Ubuntu.
 passwd: # password to login. Default is 'builder' for RHEL.
 groups: [adm, cdrom, dip, plugdev, lxd, sudo]
 lock-passwd: false
 sudo: ALL=(ALL) NOPASSWD:ALL
 shell: /bin/bash

write_files:
 - path: /usr/local/bin/nodeConfig.yaml
 permissions: '0644'
 content: |
 apiVersion: node.eks.aws/v1alpha1
 kind: NodeConfig
 spec:
 cluster:
 name: # Cluster Name
 region: # AWS region
 hybrid:
 ssm:
 activationCode: # Your ssm activation code
 activationId: # Your ssm activation id

runcmd:
 - /usr/local/bin/nodeadm init -c file:///usr/local/bin/nodeConfig.yaml >> /var/log/
 nodeadm-init.log 2>&1
```

metadata.yaml:

お使いの環境に応じた metadata.yaml を作成します。"\$NODE\_NAME" 変数の形式は、後続のステップで値が入力されるため、ファイル内にそのまま保持してください。

```
instance-id: "$NODE_NAME"
local-hostname: "$NODE_NAME"
network:
 version: 2
```

```
ethernets:
 nics:
 match:
 name: ens*
 dhcp4: yes
```

4. 以下のコマンドを使用して、`userdata.yaml` および `metadata.yaml` ファイルを `gzip +base64` 文字列として追加します。以下のコマンドは、作成する VM ごとに実行する必要があります。VM\_NAME は、更新する VM の名前に置き換えてください。

```
export NODE_NAME="VM_NAME"
export USER_DATA=$(gzip -c9 <userdata.yaml | base64)

govc vm.change -dc="YOUR_DATASTORE" -vm "$NODE_NAME" -e
 guestinfo.userdata="${USER_DATA}"
govc vm.change -dc="YOUR_DATASTORE" -vm "$NODE_NAME" -e
 guestinfo.userdata.encoding=gzip+base64

envsubst '$NODE_NAME' < metadata.yaml > metadata.yaml.tmp
export METADATA=$(gzip -c9 <metadata.yaml.tmp | base64)

govc vm.change -dc="YOUR_DATASTORE" -vm "$NODE_NAME" -e
 guestinfo.metadata="${METADATA}"
govc vm.change -dc="YOUR_DATASTORE" -vm "$NODE_NAME" -e
 guestinfo.metadata.encoding=gzip+base64
```

5. 新しい VM の電源をオンにします。これにより、設定した EKS クラスターに自動的に接続されます。

```
govc vm.power -on "${NODE_NAME}"
```

## ハイブリッドノードの認証情報を準備する

Amazon EKS Hybrid Nodes は、AWS SSM ハイブリッドアクティベーションまたは AWS IAM Roles Anywhere によってプロビジョニングされた一時的な IAM 認証情報を使用して、Amazon EKS クラスターで認証します。Amazon EKS Hybrid Nodes CLI (nodeadm) を用いて、AWS SSM ハイブリッドアクティベーションまたは AWS IAM Roles Anywhere を使用する必要があります。AWS SSM ハイブリッドアクティベーションと AWS IAM Roles Anywhere の両方を使用しないでください。認証局 (CA) とオンプレミス環境の証明書を持つ既存の公開鍵基盤 (PKI) がない場合は、AWS SSM ハイ

ブリッドアクティベーションを使用することをお勧めします。既存の PKI と証明書がオンプレミスにある場合は、AWS IAM Roles Anywhere を使用します。

## ハイブリッドノードの IAM ロール

ハイブリッドノードを Amazon EKS クラスターに接続する前に、SSM ハイブリッドアクティベーションで使用する AWS IAM ロールまたはハイブリッドノードの認証情報用の AWS IAM Roles Anywhere を作成する必要があります。クラスターの作成後、Amazon EKS アクセスエントリまたは aws-auth ConfigMap エントリでこのロールを使用して、IAM ロールを Kubernetes ロールベースのアクセスコントロール (RBAC) にマッピングします。ハイブリッドノードの IAM ロールと Kubernetes RBAC の関連付けの詳細については、「[the section called “クラスターアクセスを準備する”](#)」を参照してください。

ハイブリッドノードの IAM ロールには、以下のアクセス許可が必要です。

- ハイブリッドノードをクラスターに接続するために使用するクラスターに関する情報を収集するための eks:DescribeCluster アクションを、nodeadm が使用するためのアクセス許可。eks:DescribeCluster アクションを有効にしない場合は、nodeadm init の実行時に nodeadm に渡すノード設定で、Kubernetes API エンドポイント、クラスター CA バンドル、およびサービス IPv4 CIDR を渡す必要があります。
- kubelet が「[AmazonEC2ContainerRegistryPullOnly](#)」ポリシーで定義されているように、Amazon Elastic Container Registry (Amazon ECR) のコンテナイメージを使用するためのアクセス許可。
- AWS SSM を使用している場合、[AmazonSSMManagedInstanceCore](#) ポリシーで定義されているように、nodeadm init が AWS SSM ハイブリッドアクティベーションを使用するためのアクセス許可。
- AWS SSM を使用している場合、nodeadm uninstall がインスタンスを登録解除するために ssm:DeregisterManagedInstance アクションと ssm:DescribeInstanceInformation アクションを使用するためのアクセス許可。
- (オプション) Amazon EKS Pod Identity エージェントが eks-auth:AssumeRoleForPodIdentity アクションを使用してポッドの認証情報を取得するためのアクセス許可。

## AWS SSM ハイブリッドアクティベーションのセットアップ

AWS SSM ハイブリッドアクティベーションを設定する前に、ハイブリッドノードの IAM ロールを作成して設定する必要があります。詳細については、「[the section called “ハイブリッドノード”](#)」

[の IAM ロールを作成する](#)」を参照してください。「AWS Systems Manager ユーザーガイド」の「[Systems Manager にハイブリッドノードを登録するためのハイブリッドアクティベーションの作成](#)」の指示に従って、ハイブリッドノード用の AWS SSM ハイブリッドアクティベーションを作成します。受け取ったアクティベーションコードと ID は、ホストを Amazon EKS クラスターのハイブリッドノードとして登録する際に、`nodeadm` で使用されます。ハイブリッドノード用の Amazon EKS クラスターを作成し準備した後は、このステップに戻ることができます。

**⚠ Important**

アクティベーションの作成方法に応じて、Systems Manager はすぐにアクティベーションコードと ID をコンソールまたはコマンドウィンドウに返します。この情報をコピーして、安全な場所に保管します。コンソールから離れたり、コマンドウィンドウを閉じたりすると、この情報は失われる可能性があります。紛失した場合は、新しいアクティベーションを作成する必要があります。

デフォルトでは、AWS SSM ハイブリッドアクティベーションは 24 時間アクティブになります。または、ハイブリッドアクティベーションを作成する際に、`2024-08-01T00:00:00` のようなタイムスタンプ形式で `--expiration-date` を指定することもできます。認証情報プロバイダーとして AWS SSM を使用する場合、ハイブリッドノードのノード名は設定できず、AWS SSM によって自動生成されます。AWS SSM マネージドインスタンスは、Fleet Manager の AWS Systems Manager コンソールで表示および管理できます。追加コストなしで、アカウントにつき AWS リージョンごとに最大 1,000 のスタンダード [ハイブリッドアクティベーションノード](#) を登録できます。ただし、1,000 を超えるハイブリッドノードを登録するには、アドバンスドインスタンス層のアクティブ化が必要です。これには、[Amazon EKS Hybrid Nodes の料金](#)に含まれていない、アドバンスドインスタンス層の利用料金が発生します。詳細については、「[AWS Systems Manager の料金](#)」を参照してください。

ハイブリッドノードの IAM ロールを使用して AWS SSM ハイブリッドアクティベーションを作成する方法については、以下の例を参照してください。ハイブリッドノードの認証情報に AWS SSM ハイブリッドアクティベーションを使用する場合、ハイブリッドノードの名前は `mi-012345678abcdefgh` 形式になり、AWS SSM によってプロビジョニングされた一時的な認証情報は 1 時間有効です。AWS SSM を認証情報プロバイダーとして使用する場合、ノード名または認証情報の期間を変更することはできません。一時的な認証情報は AWS SSM によって自動的にローテーションされ、ローテーションはノードやアプリケーションのステータスには影響しません。

ハイブリッドノードの IAM ロールの AWS SSM `ssm:DeregisterManagedInstance` アクセス許可の範囲が、お使いの AWS SSM ハイブリッドアクティベーションに関連付けられたインス



タンスの登録解除のみに制限されるよう、EKS クラスターごとに 1 つの AWS SSM ハイブリッドアクティベーションを使用することをおすすめします。このページの例では、EKS クラスター ARN を含むタグが使用され、AWS SSM ハイブリッドアクティベーションを EKS クラスターにマッピングするために使用できます。または、任意のタグと方法を使用して、アクセス許可の境界と要件に基づいて AWS SSM アクセス許可の範囲を絞り込むこともできます。以下のコマンドの REGISTRATION\_LIMIT オプションは、AWS SSM ハイブリッドアクティベーションを使用できるマシンの数を制限するために使用される整数です (10 など)。

```
aws ssm create-activation \
 --region AWS_REGION \
 --default-instance-name eks-hybrid-nodes \
 --description "Activation for EKS hybrid nodes" \
 --iam-role AmazonEKSHybridNodesRole \
 --tags Key=EKSClusterARN,Value=arn:aws:eks:AWS_REGION:AWS_ACCOUNT_ID:cluster/
CLUSTER_NAME \
 --registration-limit REGISTRATION_LIMIT
```

AWS SSM ハイブリッドアクティベーションで使用可能な構成設定の詳細については、「[ハイブリッドアクティベーションを作成して Systems Manager にノードを登録する](#)」の手順を確認してください。

## AWS IAM Roles Anywhere の設定

「IAM Roles Anywhere ユーザーガイド」の「[IAM Roles Anywhere の開始方法](#)」の手順に従って、ハイブリッドノードの IAM ロールの一時的な IAM 認証情報に使用するトラストアンカーとプロファイルを設定します。プロファイルを作成する際は、ロールを追加せずにプロファイルを作成できます。プロファイルを作成してから、ハイブリッドノードの IAM ロールを作成するステップに戻り、作成したプロファイルにロールを追加することもできます。または、このページの後半にある AWS CloudFormation ステップを使用して、ハイブリッドノードの IAM Roles Anywhere セットアップを完了することもできます。

ハイブリッドノードの IAM ロールをプロファイルに追加するときは、AWS IAM Roles Anywhere コンソールの [プロファイルの編集] ページの下部にある [カスタムロール] セッション名パネルで [カスタムロールセッション名を受け入れる] を選択します。これは CreateProfile API の [acceptRoleSessionName](#) フィールドに対応します。これにより、ブートストラッププロセス中に nodeadm に渡す設定内で、ハイブリッドノードのカスタムノード名を指定できます。nodeadm init プロセス中にカスタムノード名を渡す必要があります。プロファイルの作成後、カスタムロールセッション名を受け入れるようにプロファイルを更新できます。

AWS IAM Roles Anywhere では、AWS IAM Roles Anywhere プロファイルの [durationSeconds](#) フィールドを通して認証情報の有効期間を設定できます。デフォルトの期間は 1 時間で、最大は 12 時間です。ハイブリッドノードの IAM ロールでの MaxSessionDuration 設定は、AWS IAM Roles Anywhere プロファイルの durationSeconds 設定よりも大きくする必要があります。MaxSessionDuration の詳細については、「[UpdateRole API documentation](#)」を参照してください。

認証機関 (CA) から生成するマシンごとの証明書とキーは、ファイル名を証明書は `server.pem`、キーは `server.key` として、各ハイブリッドノードの `/etc/iam/pki` ディレクトリに配置する必要があります。

### ハイブリッドノードの IAM ロールを作成する

このセクションのステップを実行するには、AWS コンソールまたは AWS CLI を使用する IAM プリンシパルに以下のアクセス許可が必要です。

- `iam:CreatePolicy`
- `iam:CreateRole`
- `iam:AttachRolePolicy`
- AWS IAM Roles Anywhere を使用している場合
  - `rolesanywhere:CreateTrustAnchor`
  - `rolesanywhere:CreateProfile`
  - `iam:PassRole`

### AWS CloudFormation

まだである場合は、AWS CLI をインストールして設定します。「[AWS CLI の最新バージョンのインストールまたは更新](#)」を参照してください。

### AWS SSM ハイブリッドアクティベーションのステップ

CloudFormation スタックは、上記のアクセス許可を持つハイブリッドノードの IAM ロールを作成します。CloudFormation テンプレートは、AWS SSM ハイブリッドアクティベーションを作成しません。

1. ハイブリッドノード用の AWS SSM CloudFormation テンプレートをダウンロードします。

```
curl -OL 'https://raw.githubusercontent.com/aws/eks-hybrid/refs/heads/main/example/hybrid-ssm-cfn.yaml'
```

2. 以下のオプションを使用して `cfn-ssm-parameters.json` を作成します。

- a. `ROLE_NAME` をハイブリッドノードの IAM ロールの名前に置き換えます。デフォルトでは、名前を指定しない場合、CloudFormation テンプレートは作成するロールの名前として `AmazonEKSHybridNodesRole` を使用します。
- b. `TAG_KEY` を、AWS SSM ハイブリッドアクティベーションの作成時に使用した AWS SSM リソースのタグキーに置き換えます。タグキーとタグ値の組み合わせは、`ssm:DeregisterManagedInstance` の条件で使用されます。これにより、ハイブリッドノードの IAM ロールは、ユーザーの AWS SSM ハイブリッドアクティベーションに関連付けられている AWS SSM マネージドインスタンスの登録解除のみ許可されるようになります。CloudFormation テンプレートでは、`TAG_KEY` はデフォルトで `EKSClusterARN` になります。
- c. `TAG_VALUE` を、AWS SSM ハイブリッドアクティベーションの作成時に使用した SSM AWS リソースタグ値に置き換えます。タグキーとタグ値の組み合わせは、`ssm:DeregisterManagedInstance` の条件で使用されます。これにより、ハイブリッドノードの IAM ロールは、ユーザーの AWS SSM ハイブリッドアクティベーションに関連付けられている AWS SSM マネージドインスタンスの登録解除のみ許可されるようになります。EKSClusterARN のデフォルトの `TAG_KEY` を使用している場合は、EKS クラスター ARN を `TAG_VALUE` として渡します。EKS クラスター ARN の形式は `arn:aws:eks:AWS_REGION:AWS_ACCOUNT_ID:cluster/CLUSTER_NAME` になります。

```
{
 "Parameters": {
 "RoleName": "ROLE_NAME",
 "SSMDeregisterConditionTagKey": "TAG_KEY",
 "SSMDeregisterConditionTagValue": "TAG_VALUE"
 }
}
```

3. CloudFormation スタックをデプロイします。STACK\_NAME は、その CloudFormation スタックに付けた任意の名前に置き換えてください。

```
aws cloudformation deploy \
 --stack-name STACK_NAME \
 --template-file hybrid-ssm-cfn.yaml \
 --parameter-overrides file://cfn-ssm-parameters.json \
```

```
--capabilities CAPABILITY_NAMED_IAM
```

## AWS IAM Roles Anywhere のステップ

CloudFormation スタックは、設定した認証機関 (CA) を使用して AWS IAM Roles Anywhere のトラストアンカーを作成し、AWS IAM Roles Anywhere プロファイルを作成し、前述のアクセス許可を使用して、ハイブリッドノードの IAM ロールを作成します。

### 1. 認証機関 (CA) のセットアップ

- AWS Private CA リソースを使用するには、[AWS Private Certificate Authority コンソール](#)を開きます。「[AWS Private CA ユーザーガイド](#)」の指示に従ってください。
- 外部 CA を使用するには、CA が提供する手順に従ってください。証明書本文は後のステップで提出します。
- パブリック CA から発行された証明書をトラストアンカーとして使用することはできません。

### 2. ハイブリッドノード用の AWS IAM Roles Anywhere CloudFormation テンプレートをダウンロードします

```
curl -OL 'https://raw.githubusercontent.com/aws/eks-hybrid/refs/heads/main/example/hybrid-ira-cfn.yaml'
```

### 3. 以下のオプションを使用して cfn-iamra-parameters.json を作成します。

- ROLE\_NAME をハイブリッドノードの IAM ロールの名前に置き換えます。デフォルトでは、名前を指定しない場合、CloudFormation テンプレートは作成するロールの名前として AmazonEKSHybridNodesRole を使用します。
- CERT\_ATTRIBUTE を、ホストを一意に識別するマシンごとの証明書属性に置き換えます。使用する証明書属性は、ハイブリッドノードをクラスターに接続するときに nodeadm 設定に使用する nodeName と一致する必要があります。詳細については、「[the section called “ハイブリッドノード用 nodeadm”](#)」を参照してください。デフォルトでは、CloudFormation テンプレートは \${aws:PrincipalTag/x509Subject/CN} を CERT\_ATTRIBUTE として使用します。これは、マシンごとの証明書の CN フィールドに対応します。または、\${aws:PrincipalTag/x509SAN/Name/CN} を CERT\_ATTRIBUTE として渡すこともできます。
- CA\_CERT\_BODY を、CA の証明書本文から改行を取り除いたものに置き換えます。CA\_CERT\_BODY は、プライバシー強化メール (PEM) 形式である必要があります。PEM 形式の CA 証明書がある場合は、CA 証明書本文を cfn-iamra-parameters.json ファイルに配置する前に、改行と BEGIN CERTIFICATE 行と END CERTIFICATE 行を削除します。

```
{
 "Parameters": {
 "RoleName": "ROLE_NAME",
 "CertAttributeTrustPolicy": "CERT_ATTRIBUTE",
 "CABundleCert": "CA_CERT_BODY"
 }
}
```

4. CloudFormation テンプレートをデプロイする。STACK\_NAME は、その CloudFormation スタックに付けた任意の名前に置き換えてください。

```
aws cloudformation deploy \
 --stack-name STACK_NAME \
 --template-file hybrid-ira-cfn.yaml \
 --parameter-overrides file://cfn-iamra-parameters.json
 --capabilities CAPABILITY_NAMED_IAM
```

## AWS CLI

まだである場合は、AWS CLI をインストールして設定します。「[AWS CLI の最新バージョンのインストールまたは更新](#)」を参照してください。

## EKS DescribeCluster ポリシーを作成する

1. 次の内容で、eks-describe-cluster-policy.json という名前のファイルを作成します。

```
{
 "Version": "2012-10-17",
 "Statement": [
 {
 "Effect": "Allow",
 "Action": [
 "eks:DescribeCluster"
],
 "Resource": "*"
 }
]
}
```

2. 次のコマンドを使用してポリシーを作成します。

```
aws iam create-policy \
 --policy-name EKSDescribeClusterPolicy \
 --policy-document file://eks-describe-cluster-policy.json
```

## AWS SSM ハイブリッドアクティベーションのステップ

1. 次の内容で、eks-hybrid-ssm-policy.json という名前のファイルを作成します。このポリシーは、ssm:DescribeInstanceInformation と ssm:DeregisterManagedInstance の 2 つのアクションのアクセス許可を付与します。このポリシーは、ssm:DeregisterManagedInstance へのアクセス許可の範囲を、信頼ポリシーで指定したリソースタグに基づき、AWS SSM ハイブリッドアクティベーションに関連付けられた AWS SSM マネージドインスタンスに制限します。
  - a. AWS\_REGION を、AWS SSM ハイブリッドアクティベーションの AWS リージョンに置き換えます。
  - b. AWS\_ACCOUNT\_ID を AWS アカウント ID に置き換えます。
  - c. TAG\_KEY を、AWS SSM ハイブリッドアクティベーションの作成時に使用した AWS SSM リソースのタグキーに置き換えます。タグキーとタグ値の組み合わせは、ssm:DeregisterManagedInstance の条件で使用されます。これにより、ハイブリッドノードの IAM ロールは、ユーザーの AWS SSM ハイブリッドアクティベーションに関連付けられている AWS SSM マネージドインスタンスの登録解除のみ許可されるようになります。CloudFormation テンプレートでは、TAG\_KEY はデフォルトで EKSClusterARN になります。
  - d. TAG\_VALUE を、AWS SSM ハイブリッドアクティベーションの作成時に使用した SSM AWS リソースタグ値に置き換えます。タグキーとタグ値の組み合わせは、ssm:DeregisterManagedInstance の条件で使用されます。これにより、ハイブリッドノードの IAM ロールは、ユーザーの AWS SSM ハイブリッドアクティベーションに関連付けられている AWS SSM マネージドインスタンスの登録解除のみ許可されるようになります。EKSClusterARN のデフォルトの TAG\_KEY を使用している場合は、EKS クラスター ARN を TAG\_VALUE として渡します。EKS クラスター ARN の形式は arn:aws:eks:AWS\_REGION:AWS\_ACCOUNT\_ID:cluster/CLUSTER\_NAME になります。

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

```

 "Effect": "Allow",
 "Action": "ssm:DescribeInstanceInformation",
 "Resource": "*"
 },
 {
 "Effect": "Allow",
 "Action": "ssm:DeregisterManagedInstance",
 "Resource": "arn:aws:ssm:AWS_REGION:AWS_ACCOUNT_ID:managed-instance/
*",
 "Condition": {
 "StringEquals": {
 "ssm:resourceTag/TAG_KEY": "TAG_VALUE"
 }
 }
 }
]
}

```

2. 次のコマンドを使用してポリシーを作成します。

```

aws iam create-policy \
 --policy-name EKSHybridSSMPolicy \
 --policy-document file://eks-hybrid-ssm-policy.json

```

3. eks-hybrid-ssm-trust.json という名前のファイルを作成します。AWS\_REGION を AWS SSM ハイブリッドアクティベーションの AWS リージョンに、AWS\_ACCOUNT\_ID をお使いの AWS アカウント ID に置き換えます。

```

{
 "Version": "2012-10-17",
 "Statement": [
 {
 "Sid": "",
 "Effect": "Allow",
 "Principal": {
 "Service": "ssm.amazonaws.com"
 },
 "Action": "sts:AssumeRole",
 "Condition": {
 "StringEquals": {
 "aws:SourceAccount": "AWS_ACCOUNT_ID"
 },
 "ArnEquals": {

```

```
 "aws:SourceArn": "arn:aws:ssm:AWS_REGION:AWS_ACCOUNT_ID:*"
 }
}
}
]
```

4. 次のコマンドを使用してロールを作成します。

```
aws iam create-role \
 --role-name AmazonEKSHybridNodesRole \
 --assume-role-policy-document file://eks-hybrid-ssm-trust.json
```

5. 前のステップで作成した EKSDescribeClusterPolicy と EKSHybridSSMPolicy をアタッチします。AWS\_ACCOUNT\_ID を AWS アカウント ID に置き換えます。

```
aws iam attach-role-policy \
 --role-name AmazonEKSHybridNodesRole \
 --policy-arn arn:aws:iam::AWS_ACCOUNT_ID:policy/EKSDescribeClusterPolicy
```

```
aws iam attach-role-policy \
 --role-name AmazonEKSHybridNodesRole \
 --policy-arn arn:aws:iam::AWS_ACCOUNT_ID:policy/EKSHybridSSMPolicy
```

6. AmazonEC2ContainerRegistryPullOnly と AmazonSSMManagedInstanceCore AWS マネージドポリシーをアタッチします。

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

```
aws iam attach-role-policy \
 --role-name AmazonEKSHybridNodesRole \
 --policy-arn arn:aws:iam::aws:policy/AmazonSSMManagedInstanceCore
```

## AWS IAM Roles Anywhere のステップ

AWS IAM Roles Anywhere を使用するには、ハイブリッドノードの IAM ロールを作成する前に、AWS IAM Roles Anywhere のトラストアンカーを設定する必要があります。手順については「[the section called “AWS IAM Roles Anywhere の設定”](#)」を参照してください。



1. eks-hybrid-iamra-trust.json という名前のファイルを作成します。TRUST\_ANCHOR ARN を、[the section called “AWS IAM Roles Anywhere の設定”](#) のステップで作成したトラストアンカーの ARN に置き換えます。この信頼ポリシーの条件は、AWS IAM Roles Anywhere がハイブリッドノードの IAM ロールを引き受けて一時的な IAM 認証情報を交換する機能を、ロールセッション名がハイブリッドノードにインストールされている x509 証明書の CN と一致する場合のみに制限します。別の方法として、他の証明書属性を使用してノードを一意に識別することもできます。信頼ポリシーで使用する証明書属性は、nodeadm 設定で設定した nodeName に対応している必要があります。詳細については、「[the section called “ハイブリッドノード用 nodeadm”](#)」を参照してください。

```
{
 "Version": "2012-10-17",
 "Statement": [
 {
 "Effect": "Allow",
 "Principal": {
 "Service": "rolesanywhere.amazonaws.com"
 },
 "Action": [
 "sts:TagSession",
 "sts:SetSourceIdentity"
],
 "Condition": {
 "ArnEquals": {
 "aws:SourceArn": "TRUST_ANCHOR_ARN"
 }
 }
 },
 {
 "Effect": "Allow",
 "Principal": {
 "Service": "rolesanywhere.amazonaws.com"
 },
 "Action": "sts:AssumeRole",
 "Condition": {
 "StringEquals": {
 "sts:RoleSessionName": "${aws:PrincipalTag/x509Subject/CN}"
 },
 "ArnEquals": {
 "aws:SourceArn": "TRUST_ANCHOR_ARN"
 }
 }
 }
]
}
```

```
 }
]
}
```

2. 次のコマンドを使用してロールを作成します。

```
aws iam create-role \
 --role-name AmazonEKSHybridNodesRole \
 --assume-role-policy-document file://eks-hybrid-iamra-trust.json
```

3. 前のステップで作成した EKSDescribeClusterPolicy をアタッチします。AWS\_ACCOUNT\_ID を AWS アカウント ID に置き換えます。

```
aws iam attach-role-policy \
 --role-name AmazonEKSHybridNodesRole \
 --policy-arn arn:aws:iam::AWS_ACCOUNT_ID:policy/EKSDescribeClusterPolicy
```

4. AmazonEC2ContainerRegistryPullOnly AWS マネージドポリシーをアタッチします。

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

## AWS Management Console

### EKS DescribeCluster ポリシーを作成する

1. [Amazon IAM コンソール](#)を開きます。
2. 左のナビゲーションペインの [ポリシー] を選択します。
3. [ポリシー] ページで、[ポリシーの作成] を選択します。
4. [アクセス許可を指定] ページの [サービスの選択] パネルで、[EKS] を選択します。
  - a. DescribeCluster でアクションをフィルタリングし、DescribeCluster の Read アクションを選択します。
  - b. [次へ] を選択します。
5. [確認と作成] ページで以下を行います。
  - a. ポリシーのポリシー名を、たとえば EKSDescribeClusterPolicy のように入力します。
  - b. [Create policy] (ポリシーの作成) を選択します。

## AWS SSM ハイブリッドアクティベーションのステップ

1. [Amazon IAM コンソール](#)を開きます。
2. 左のナビゲーションペインの [ポリシー] を選択します。
3. [ポリシー] ページで、[ポリシーの作成] を選択します。
4. [アクセス許可を指定] ページの [ポリシーエディタ] の右上のナビゲーションで、[JSON] を選択します。以下のスニペットを貼り付けます。AWS\_REGION を AWS SSM ハイブリッドアクティベーションの AWS リージョンに置き換え、AWS\_ACCOUNT\_ID をお使いの AWS アカウント ID に置き換えます。TAG\_KEY と TAG\_VALUE を、AWS SSM ハイブリッドアクティベーションの作成時に使用した AWS SSM リソースのタグキーに置き換えます。

```
{
 "Version": "2012-10-17",
 "Statement": [
 {
 "Effect": "Allow",
 "Action": "ssm:DescribeInstanceInformation",
 "Resource": "*"
 },
 {
 "Effect": "Allow",
 "Action": "ssm:DeregisterManagedInstance",
 "Resource": "arn:aws:ssm:AWS_REGION:AWS_ACCOUNT_ID:managed-instance/*",
 "Condition": {
 "StringEquals": {
 "ssm:resourceTag/TAG_KEY": "TAG_VALUE"
 }
 }
 }
]
}
```

- a. [次へ] を選択します。
5. [確認と作成] ページで以下を行います。
    - a. ポリシーのポリシー名を、たとえば EKSHybridSSMPolicy のように入力します。
    - b. [ポリシーの作成] を選択します。
  6. 左のナビゲーションペインで、[ロール] を選択してください。
  7. [ロール] ページで、[ロールの作成] を選択してください。

8. [信頼されたエンティティを選択] ページで、以下の操作を実行してください：

- a. [信頼されたエンティティのタイプ] セクションで [カスタム信頼ポリシー] を選択します。以下をカスタム信頼ポリシーエディタに貼り付けます。AWS\_REGION を AWS SSM ハイブリッドアクティベーションの AWS リージョンに、AWS\_ACCOUNT\_ID をお使いの AWS アカウント ID に置き換えます。

```
{
 "Version": "2012-10-17",
 "Statement": [
 {
 "Sid": "",
 "Effect": "Allow",
 "Principal": {
 "Service": "ssm.amazonaws.com"
 },
 "Action": "sts:AssumeRole",
 "Condition": {
 "StringEquals": {
 "aws:SourceAccount": "AWS_ACCOUNT_ID"
 },
 "ArnEquals": {
 "aws:SourceArn": "arn:aws:ssm:AWS_REGION:AWS_ACCOUNT_ID:*"
 }
 }
 }
]
}
```

- b. [次へ] を選択します。

9. [アクセス許可を追加] ページで、カスタムポリシーをアタッチするか、以下の操作を行います。

- a. [フィルターポリシー] ボックスに、EKSDescribeClusterPolicy または上記で作成したポリシーの名前を入力します。検索結果でポリシー名の左にあるチェックボックスを選択します。
- b. [フィルターポリシー] ボックスに、EKSHybridSSMPolicy または上記で作成したポリシーの名前を入力します。検索結果でポリシー名の左にあるチェックボックスを選択します。
- c. [フィルタポリシー] ボックスに AmazonEC2ContainerRegistryPullOnly と入力します。検索結果の AmazonEC2ContainerRegistryPullOnly の左にあるチェックボックスを選択します。
- d. [フィルタポリシー] ボックスに AmazonSSMManagedInstanceCore と入力します。検索結果の AmazonSSMManagedInstanceCore の左にあるチェックボックスを選択します。

e. [次へ] を選択します。

10[名前を付けて、レビューし、作成する] ページで、以下の操作を実行します。

- a. [ロール名] に、AmazonEKSHybridNodesRole などのロールの一意の名前を入力します。
- b. [Description] (説明) では、現在のテキストを「Amazon EKS - Hybrid Nodes role」などの説明文に置き換えます。
- c. [ロールの作成] を選択してください。

## AWS IAM Roles Anywhere のステップ

AWS IAM Roles Anywhere を使用するには、ハイブリッドノードの IAM ロールを作成する前に、AWS IAM Roles Anywhere のトラストアンカーを設定する必要があります。手順については「[the section called “AWS IAM Roles Anywhere の設定”](#)」を参照してください。

1. [Amazon IAM コンソール](#)を開きます。
2. 左のナビゲーションペインで、[ロール] を選択してください。
3. [ロール] ページで、[ロールの作成] を選択してください。
4. [信頼されたエンティティを選択] ページで、以下の操作を実行してください：
  - a. [信頼されたエンティティのタイプ] セクションで [カスタム信頼ポリシー] を選択します。以下をカスタム信頼ポリシーエディタに貼り付けます。TRUST\_ANCHOR ARN を、[the section called “AWS IAM Roles Anywhere の設定”](#) のステップで作成したトラストアンカーの ARN に置き換えます。この信頼ポリシーの条件は、AWS IAM Roles Anywhere がハイブリッドノードの IAM ロールを引き受けて一時的な IAM 認証情報を交換する機能を、ロールセッション名がハイブリッドノードにインストールされている x509 証明書の CN と一致する場合のみに制限します。別の方法として、他の証明書属性を使用してノードを一意に識別することもできます。信頼ポリシーで使用する証明書属性は、nodeadm 設定で設定した nodeName に対応している必要があります。詳細については、「[the section called “ハイブリッドノード用 nodeadm”](#)」を参照してください。

```
{
 "Version": "2012-10-17",
 "Statement": [
 {
 "Effect": "Allow",
 "Principal": {
 "Service": "rolesanywhere.amazonaws.com"
 },
 "Action": [
```

```

 "sts:TagSession",
 "sts:SetSourceIdentity"
],
 "Condition": {
 "ArnEquals": {
 "aws:SourceArn": "TRUST_ANCHOR_ARN"
 }
 }
},
{
 "Effect": "Allow",
 "Principal": {
 "Service": "rolesanywhere.amazonaws.com"
 },
 "Action": "sts:AssumeRole",
 "Condition": {
 "StringEquals": {
 "sts:RoleSessionName": "${aws:PrincipalTag/x509Subject/CN}"
 },
 "ArnEquals": {
 "aws:SourceArn": "TRUST_ANCHOR_ARN"
 }
 }
}
]
}

```

- b. [次へ] を選択します。
5. [アクセス許可を追加] ページで、カスタムポリシーをアタッチするか、以下の操作を行います。
  - a. [フィルターポリシー] ボックスに、EKSDescribeClusterPolicy または上記で作成したポリシーの名前を入力します。検索結果でポリシー名の左にあるチェックボックスを選択します。
  - b. [フィルタポリシー] ボックスに AmazonEC2ContainerRegistryPullOnly と入力します。検索結果の AmazonEC2ContainerRegistryPullOnly の左にあるチェックボックスを選択します。
  - c. [次へ] を選択します。
6. [名前を付けて、レビューし、作成する] ページで、以下の操作を実行します。
  - a. [ロール名] に、AmazonEKSHybridNodesRole などのロールの一意の名前を入力します。
  - b. [Description] (説明) では、現在のテキストを「Amazon EKS - Hybrid Nodes role」などの説明文に置き換えます。
  - c. [ロールの作成] を選択してください。

## ハイブリッドノードを使用して Amazon EKS クラスターを作成する

このトピックでは、使用可能なオプションの概要と、ハイブリッドノード対応の Amazon EKS クラスターの作成時に考慮すべき点を説明します。EKS Hybrid Nodes には、クラウドノードを使用する Amazon EKS クラスターと同じ [Kubernetes バージョンのサポート](#)があります (標準サポートや拡張サポートなど)。

EKS Hybrid Nodes を使用する予定がない場合は、「[the section called “クラスターを作成する”](#)」にある Amazon EKS でのクラスター作成に関する基本ドキュメントを参照してください。

### 前提条件

- [the section called “前提条件”](#) が完了していること。ハイブリッドノード対応クラスターを作成する前に、オンプレミスノードとオプションでポッド CIDR を特定し、EKS の要件とハイブリッドノードの要件に従って VPC とサブネットを作成し、オンプレミスおよびオプションでポッド CIDR のインバウンドルールを持つセキュリティグループを指定する必要があります。前提条件の詳細については、[the section called “ネットワークを準備する”](#) を参照してください。
- AWS コマンドラインインターフェイス (AWS CLI) の最新バージョンがデバイスにインストールおよび設定されていること。現在のバージョンを確認するには「aws --version」を参照してください。yum、apt-get、macOS 用の Homebrew などのパッケージマネージャーは、多くの場合 AWS CLI の最新バージョンより数バージョン古くなっています。最新バージョンをインストールするには、「AWS コマンドラインインターフェイスユーザーガイド」の「[AWS CLI の最新バージョンのインストールまたは更新](#)」および「[AWS CLI の設定の構成](#)」を参照してください。
- IAM ロールを作成してポリシーをアタッチし、EKS クラスターを作成および記述するアクセス許可を持つ [IAM プリンシパル](#)があること

### 考慮事項

- クラスターは、クラスター認証モードに API または API\_AND\_CONFIG\_MAP を使用する必要があります。
- クラスターは IPv4 アドレスファミリーを使用する必要があります。
- クラスターは、パブリッククラスターエンドポイント接続またはプライベートクラスターエンドポイント接続を使用する必要があります。ハイブリッドノードが VPC 外で実行される場合、Amazon EKS Kubernetes API サーバーエンドポイントがパブリック IP に解決されるため、クラスターは「パブリックおよびプライベート」のクラスターエンドポイント接続を使用できません。
- OIDC 認証は、ハイブリッドノードが有効な EKS クラスターでサポートされています。

- 既存のクラスターのハイブリッドノード設定を追加、変更、削除できます。詳細については、「[the section called “既存のクラスター”](#)」を参照してください。

## ステップ 1: クラスター IAM ロールを作成する

既にクラスター IAM ロールがある場合や、eksctl または AWS CloudFormation を使用してクラスターを作成する場合は、このステップはスキップできます。デフォルトでは、eksctl および AWS CloudFormation テンプレートによってクラスター IAM ロールが作成されます。

1. IAM 信頼ポリシー用の JSON ファイルを作成するには次のコマンドを実行してください。

```
cat >eks-cluster-role-trust-policy.json <<EOF
{
 "Version": "2012-10-17",
 "Statement": [
 {
 "Effect": "Allow",
 "Principal": {
 "Service": "eks.amazonaws.com"
 },
 "Action": "sts:AssumeRole"
 }
]
}
EOF
```

2. Amazon EKS クラスターの IAM ロールを作成します。必要であれば、前のステップでファイルを書き込んだコンピュータ上のパスを eks-cluster-role-trust-policy.json の前につけます。このコマンドは前のステップで作成した信頼ポリシーをロールに関連付けます。IAM ロールを作成するにはロールを作成する [IAM プリンシパル](#)に iam:CreateRole アクション (許可) を割り当てる必要があります。

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

3. Amazon EKS 管理のポリシーを割り当てるか、独自のカスタムポリシーを作成できます。カスタムポリシーで使用する必要がある最小限の許可については、「[the section called “ノードの IAM ロール”](#)」を参照してください。このロールに、Amazon EKS 管理の IAM ポリシー (AmazonEKSClusterPolicy) をアタッチします。IAM ポリシーを [IAM プリンシパ](#)



[ル](#)にアタッチするには、ポリシーのアタッチを行っているプリンシパルに、次のいずれかの IAM アクション (許可) を割り当てる必要があります: `iam:AttachUserPolicy` または `iam:AttachRolePolicy`。

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

## ステップ 2: ハイブリッドノード対応クラスターを作成する

以下を使用してクラスターを作成できます：

- [eksctl](#)
- [AWS CloudFormation](#)
- [AWS CLI](#)
- [AWS Management Console](#)

### ハイブリッドノード対応クラスターの作成 - eksctl

eksctl コマンドラインツールの最新バージョンをインストールする必要があります。eksctl をインストールまたはアップグレードするにはeksctl ドキュメントの「[インストール](#)」を参照してください。

1. `cluster-config.yaml` を作成して、ハイブリッドノード対応の Amazon EKS IPv4 クラスターを定義します。`cluster-config.yaml` で以下の置換を実行します。設定の完全なリストについては、「[eksctl documentation](#)」を参照してください。
  - a. `CLUSTER_NAME` をクラスターの名前に置き換えます。この名前には英数字 (大文字と小文字が区別されます) とハイフンのみを使用できます。先頭の文字は英数字である必要があります。また、100 文字より長くすることはできません。名前はクラスターを作成する AWS リージョンおよび AWS アカウント内で一意である必要があります。
  - b. `AWS_REGION` を、クラスターを作成する AWS リージョンに置き換えます。
  - c. `K8S_VERSION` を [Amazon EKS がサポートする任意のバージョン](#) に置き換えます。
  - d. [the section called “認証情報を準備する”](#) のステップで設定した認証情報プロバイダーに基づいて、`CREDS_PROVIDER` を `ssm` または `ira` に置き換えます。
  - e. 認証情報プロバイダーが `ira` に設定されている場合 (AWS IAM Roles Anywhere を認証情報プロバイダーとして使用)、`CA_BUNDLE_CERT` を置き換えます。`CA_BUNDLE_CERT` は認証機

関 (CA) 証明書本文であり、CA の選択によって異なります。証明書はプライバシー強化メール (PEM) 形式である必要があります。

- f. GATEWAY\_ID を、VPC にアタッチする仮想プライベートゲートウェイまたはトランジットゲートウェイの ID に置き換えます。
- g. REMOTE\_NODE\_CIDRS を、ハイブリッドノードのオンプレミスノード CIDR に置き換えます。
- h. REMOTE\_POD\_CIDRS をハイブリッドノードで実行されているワークロードのオンプレミスポッド CIDR に置き換えるか、ハイブリッドノードでウェブフックを実行していない場合は設定から行を削除します。CNI がネットワークアドレス変換 (NAT) を使用しない場合や、ポッドトラフィックがオンプレミスホストから出るときにポッド IP アドレスをマスクする場合は、REMOTE\_POD\_CIDRS を設定する必要があります。ハイブリッドノードでウェブフックを実行している場合は、REMOTE\_POD\_CIDRS を設定する必要があります。詳細については、[「the section called “ウェブフックの設定”」](#)を参照してください。
- i. オンプレミスノードとポッド CIDR ブロックは、以下の要件を満たしている必要があります。
  - i. 次の IPv4 RFC-1918 のいずれかの範囲内にあること: 10.0.0.0/8、172.16.0.0/12、または 192.168.0.0/16。
  - ii. 相互に、クラスターの VPC CIDR、または Kubernetes サービスの IPv4 CIDR と重複しないこと

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

metadata:
 name: CLUSTER_NAME
 region: AWS_REGION
 version: "K8S_VERSION"

remoteNetworkConfig:
 iam:
 provider: CREDS_PROVIDER # default SSM, can also be set to IRA
 # caBundleCert: CA_BUNDLE_CERT
 vpcGatewayID: GATEWAY_ID
 remoteNodeNetworks:
 - cidrs: ["REMOTE_NODE_CIDRS"]
 remotePodNetworks:
 - cidrs: ["REMOTE_POD_CIDRS"]
```

2. 次のコマンドを実行してください：

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

クラスターのプロビジョニングには数分かかります。クラスターの作成中は数行の出力が表示されます。出力の最後の行は次のサンプル行のようになります。

```
[#] EKS cluster "CLUSTER_NAME" in "REGION" region is ready
```

3. 「[the section called “ステップ 3: kubeconfig を更新する”](#)」に進みます。

## ハイブリッドノード対応クラスターの作成 - AWS CloudFormation

CloudFormation スタックは、指定した RemotePodNetwork と RemoteNodeNetwork を使用して、EKS クラスター IAM ロールと EKS クラスターを作成します。CloudFormation テンプレートで公開されていない EKS クラスターの設定をカスタマイズする必要がある場合は、CloudFormation テンプレートを変更します。

1. CloudFormation テンプレートをダウンロードします。

```
curl -OL 'https://raw.githubusercontent.com/aws/eks-hybrid/refs/heads/main/example/hybrid-eks-cfn.yaml'
```

2. cfn-eks-parameters.json を作成し、各値の設定を指定します。

- a. CLUSTER\_NAME: 作成する EKS クラスターの名前。
- b. CLUSTER\_ROLE\_NAME: 作成する EKS クラスター IAM ロールの名前。テンプレートのデフォルトは「EKSClusterRole」です。
- c. SUBNET1\_ID: 前提条件のステップで作成した最初のサブネットの ID
- d. SUBNET2\_ID: 前提条件のステップで作成した 2 番目のサブネットの ID
- e. SG\_ID: 前提条件のステップで作成したセキュリティグループの ID
- f. REMOTE\_NODE\_CIDRS: ハイブリッドノードのオンプレミスノード CIDR
- g. REMOTE\_POD\_CIDRS: ハイブリッドノードで実行されているワークロードのオンプレミスポッド CIDR CNi がネットワークアドレス変換 (NAT) を使用しない場合や、ポッドトラフィックがオンプレミスホストから出るときにポッド IP アドレスをマスクする場合は、REMOTE\_POD\_CIDRS を設定する必要があります。ハイブリッドノードでウェブフックを実行している場合は、REMOTE\_POD\_CIDRS を設定する必要があります。詳細については、「[the section called “ウェブフックの設定”](#)」を参照してください。

- h. オンプレミスノードとポッド CIDR ブロックは、以下の要件を満たしている必要があります。
  - i. 次の IPv4 RFC-1918 のいずれかの範囲内にあること: 10.0.0.0/8、172.16.0.0/12、または 192.168.0.0/16。
  - ii. 相互に、クラスターの VPC CIDR、または Kubernetes サービスの IPv4 CIDR と重複しないこと。
- i. CLUSTER\_AUTH: クラスターのクラスター認証モード。有効な値は、API および API\_AND\_CONFIG\_MAP です。テンプレートのデフォルトは API\_AND\_CONFIG\_MAP です。
- j. CLUSTER\_ENDPOINT: クラスターのクラスターエンドポイント接続。有効な値は「Public」および「Private」です。テンプレートのデフォルトは Private です。つまり、Kubernetes API エンドポイントには VPC 内からしか接続できません。
- k. K8S\_VERSION: クラスターに使用する Kubernetes のバージョン。「[the section called “Kubernetes バージョン”](#)」を参照してください。

```
{
 "Parameters": {
 "ClusterName": "CLUSTER_NAME",
 "ClusterRoleName": "CLUSTER_ROLE_NAME",
 "SubnetId1": "SUBNET1_ID",
 "SubnetId2": "SUBNET2_ID",
 "SecurityGroupId": "SG_ID",
 "RemoteNodeCIDR": "REMOTE_NODE_CIDRS",
 "RemotePodCIDR": "REMOTE_POD_CIDRS",
 "ClusterAuthMode": "CLUSTER_AUTH",
 "ClusterEndpointConnectivity": "CLUSTER_ENDPOINT",
 "K8sVersion": "K8S_VERSION"
 }
}
```

- 3. CloudFormation スタックをデプロイします。STACK\_NAME を CloudFormation スタックの名前に置き換え、AWS\_REGION をクラスターを作成する AWS リージョンに置き換えます。

```
aws cloudformation deploy \
 --stack-name STACK_NAME \
 --region AWS_REGION \
 --template-file hybrid-eks-cfn.yaml \
 --parameter-overrides file://cfn-eks-parameters.json \
 --capabilities CAPABILITY_NAMED_IAM
```

クラスターのプロビジョニングには数分かかります。以下のコマンドを使用してスタックの状態を確認できます。STACK\_NAME を CloudFormation スタックの名前に置き換え、AWS\_REGION をクラスターを作成する AWS リージョンに置き換えます。

```
aws cloudformation describe-stacks \
 --stack-name STACK_NAME \
 --region AWS_REGION \
 --query 'Stacks[].StackStatus'
```

4. 「[the section called “ステップ 3: kubeconfig を更新する”](#)」に進みます。

## ハイブリッドノード対応クラスターの作成 - AWS CLI

1. 以下のコマンドを実行して、ハイブリッドノード対応の EKS クラスターを作成します。コマンドを実行する前に、以下を自分の設定に置き換えます。設定の完全なリストについては、[the section called “クラスターを作成する”](#) のドキュメントを参照してください。
  - a. CLUSTER\_NAME: 作成する EKS クラスターの名前。
  - b. AWS\_REGION: クラスターを作成する AWS リージョン。
  - c. K8S\_VERSION: クラスターに使用する Kubernetes のバージョン。「サポートされている Amazon EKS バージョン」を参照してください。
  - d. ROLE\_ARN: クラスター用に設定した Amazon EKS クラスターロール。詳細については、「Amazon EKS クラスターの IAM ロール」を参照してください。
  - e. SUBNET1\_ID: 前提条件のステップで作成した最初のサブネットの ID
  - f. SUBNET2\_ID: 前提条件のステップで作成した 2 番目のサブネットの ID
  - g. SG\_ID: 前提条件のステップで作成したセキュリティグループの ID
  - h. クラスターアクセス認証モードには API と API\_AND\_CONFIG\_MAP を使用できます。以下のコマンドでは、クラスターアクセス認証モードは API\_AND\_CONFIG\_MAP に設定されています。
  - i. endpointPrivateAccess パラメータ と endpointPublicAccess パラメータを使用して、クラスターの Kubernetes API サーバーエンドポイントへのパブリックとプライベートアクセスを有効または無効にすることができます。以下のコマンドでは、endpointPublicAccess は false に設定され、endpointPrivateAccess は true に設定されています。
  - j. REMOTE\_NODE\_CIDRS: ハイブリッドノードのオンプレミスノード CIDR。

- k. REMOTE\_POD\_CIDRS (オプション): ハイブリッドノードで実行されているワークロードのオンプレミスポッド CIDR
- l. オンプレミスノードとポッド CIDR ブロックは、以下の要件を満たしている必要があります。
  - i. 次の IPv4 RFC-1918 のいずれかの範囲内にあること: 10.0.0.0/8、172.16.0.0/12、または 192.168.0.0/16。
  - ii. 相互に、Amazon EKS クラスターの VPC CIDR、または Kubernetes サービスの IPv4 CIDR と重複しないこと。

```
aws eks create-cluster \
 --name CLUSTER_NAME \
 --region AWS_REGION \
 --kubernetes-version K8S_VERSION \
 --role-arn ROLE_ARN \
 --resources-vpc-config
subnetIds=SUBNET1_ID,SUBNET2_ID,securityGroupIds=SG_ID,endpointPrivateAccess=true,endpointPublicAccess=true \
 --access-config authenticationMode=API_AND_CONFIG_MAP \
 --remote-network-config '{"remoteNodeNetworks":[{"cidrs":
["REMOTE_NODE_CIDRS"]}], "remotePodNetworks":[{"cidrs":["REMOTE_POD_CIDRS"]}]}'
```

- 2. クラスターがプロビジョニングされるまでに数分かかります。クラスターのステータスのクエリを実行するには次のコマンドを使用します。CLUSTER\_NAME を、作成しているクラスターの名前に置き換え、AWS\_REGION を、クラスターを作成している AWS リージョンに置き換えます。出力が「ACTIVE」を返すまで、次のステップに進まないでください。

```
aws eks describe-cluster \
 --name CLUSTER_NAME \
 --region AWS_REGION \
 --query "cluster.status"
```

- 3. 「[the section called “ステップ 3: kubeconfig を更新する”](#)」に進みます。

## ハイブリッドノード対応クラスターの作成 - AWS Management Console

- 1. [Amazon EKS コンソール](#)で Amazon EKS コンソールを開きます。
- 2. [クラスターを追加]、[作成] の順にクリックします。
- 3. [クラスターの設定] ページで、次のフィールドに入力します。

- a. [名前] - クラスターの名前。この名前には英数字 (大文字と小文字が区別されます)、ハイフン、下線のみを使用できます。先頭の文字は英数字である必要があります。また、100 文字より長くすることはできません。名前はクラスターを作成する AWS リージョンおよび AWS アカウ  
ント内で一意である必要があります。
- b. [クラスター IAM ロール] - ユーザーに代わって AWS リソースを管理することを Kubernetes コ  
ントロールプレーンに許可するために作成した Amazon EKS クラスター IAM ロールを選択し  
ます。
- c. [Kubernetes バージョン] - クラスターで使用する Kubernetes のバージョン。以前のバージョ  
ンが必要でない限り、最新バージョンを選択することをお勧めします。
- d. [アップグレードポリシー] - 拡張または標準を選択します。
  - i. 拡張: このオプションは、リリース日から 26 か月間 Kubernetes バージョンをサポートしま  
す。延長サポート期間には、標準サポート期間終了後に開始される追加の時間単位のコスト  
がかかります。延長サポートが終了すると、クラスターは次のバージョンに自動的にアップ  
グレードされます。
  - ii. 標準: このオプションは、リリース日から 14 か月間 Kubernetes バージョンをサポートしま  
す。追加コストはかかりません。標準サポートが終了すると、クラスターは次のバージョン  
に自動的にアップグレードされます。
- e. クラスターアクセス - クラスター管理者アクセスを許可または禁止し、認証モードを選択しま  
す。ハイブリッドノードが有効なクラスターでは、以下の認証モードがサポートされます。
  - i. EKS API: クラスターは、認証された IAM プリンシパルを EKS アクセスエントリ API から  
のみ取得します。
  - ii. EKS API と ConfigMap: クラスターは、EKS アクセスエントリ API と aws-auth  
ConfigMap の両方から認証された IAM プリンシパルを取得します。
- f. [Secret encryption] (シークレット暗号化) - (オプション) KMS キーを使用して Kubernetes シー  
クレットのシークレット暗号化を有効にするよう選択します。クラスターを作成した後で、こ  
れを有効にすることもできます。この機能を有効にする前に、[the section called “シークレット  
暗号化を有効にする”](#) の情報をよく理解していることを確認してください。
- g. ARC ゾーンシフト - 有効にすると、EKS はクラスターを ARC ゾーンシフトに登録し、ゾーン  
シフトを使用してアプリケーショントラフィックを AZ から遠ざけることができるようになり  
ます。
- h. [タグ] - (オプション) クラスターに任意のタグを追加します。詳細については、「[the section  
called “リソースのタグ付け”](#)」を参照してください。
- i. このページを読み終えたら、[次へ] を選択してください。

#### 4. [ネットワーキングの指定] ページで、次のフィールドの値を選択してください：



- a. VPC – [the section called “VPC およびサブネットの要件”](#) および [Amazon EKS Hybrid Nodes](#) の要件を満たす既存の VPC を選択します。VPC を選択する前に、「VPC、サブネット、ハイブリッドノードの Amazon EKS ネットワーキング要件を表示する」の要件と考慮事項をすべて理解しておくことをお勧めします。クラスターの作成後は使用する VPC を変更できません。VPC が表示されていない場合はまず作成する必要があります。詳細については、「[the section called “VPC を作成する”](#)」および「[Amazon EKS Hybrid Nodes のネットワーク要件](#)」を参照してください。
- b. [サブネット] - デフォルトで、前のフィールドで指定した VPC 内の利用可能なすべてのサブネットがあらかじめ選択されています。少なくとも 2 つ選択する必要があります。
- c. [セキュリティグループ] - (オプション Amazon EKS が作成するネットワークインターフェイスに関連付ける 1 つまたは複数のセキュリティグループを指定します。指定するセキュリティグループの少なくとも 1 つには、オンプレミスのノードとオプションで ポッド CIDR に対するインバウンドルールが必要です。詳細については、「[Amazon EKS Hybrid Nodes のネットワーク要件](#)」を参照してください。セキュリティグループを選択するかどうかにかかわらず、Amazon EKS はクラスターと VPC 間の通信を可能にするセキュリティグループを作成します。Amazon EKS はこのセキュリティグループおよびユーザーが選択したセキュリティグループを、作成するネットワークインターフェイスに関連付けます。Amazon EKS が作成するクラスターセキュリティグループの詳細については、「[the section called “セキュリティグループの要件”](#)」を参照してください。Amazon EKS が作成するクラスターセキュリティグループのルールは変更できます。
- d. クラスター IP アドレスファミリーの選択 — ハイブリッドノード対応クラスターには IPv4 を選択する必要があります。
- e. (オプション) [Kubernetes サービス IP アドレスの範囲を設定する] を選択し、[サービスの IPv4 範囲] を指定します。
- f. [リモートネットワークを設定してハイブリッドノードを有効にする] を選択し、ハイブリッドノードのオンプレミスノードとポッド CIDR を指定します。
- g. ポッドトラフィックがオンプレミスホストを離れる際、CNI がポッド IP アドレスに対してネットワークアドレス変換 (NAT) またはマスカレードを使用しない場合は、リモートポッド CIDR を設定する必要があります。ハイブリッドノードでウェブフックを実行している場合は、リモートポッド CIDR を設定する必要があります。
- h. オンプレミスノードとポッド CIDR ブロックは、以下の要件を満たしている必要があります。
  - i. 次の IPv4 RFC-1918 のいずれかの範囲内にあること: 10.0.0.0/8、172.16.0.0/12、または 192.168.0.0/16。



- ii. 相互に、クラスターの VPC CIDR、または Kubernetes サービスの IPv4 CIDR と重複しないこと
  - i. [クラスターエンドポイントのアクセス] で、オプションを選択してください。クラスターを作成した後で、このオプションを変更できます。ハイブリッドノードが有効なクラスターの場合は、パブリックまたはプライベートのいずれかを選択する必要があります。デフォルト以外のオプションを選択する前に、オプションとその意味を理解しておいてください。詳細については「[the section called “クラスターエンドポイントのアクセス”](#)」を参照してください。
  - j. このページを読み終えたら、[次へ] を選択してください。
5. (オプション) [オブザーバビリティの設定] ページで、有効にする [メトリクス] と [コントロールプレーンのロギング] オプションを選択します。デフォルトではそれぞれのログタイプは無効化されています。
- a. Prometheus メトリクスの詳細については、「[the section called “Prometheus メトリクス”](#)」を参照してください。
  - b. EKS コントロールのログ記録のオプションの詳細については、「[the section called “コントロールプレーンのログ”](#)」を参照してください。
  - c. このページを読み終えたら、[次へ] を選択してください。
6. [アドオンの選択] ページで、クラスターに追加するアドオンを選択してください。
- a. [Amazon EKS アドオン] と [AWS マーケットプレイス アドオン] は必要な数だけ選択できます。ハイブリッドノードと互換性のない Amazon EKS アドオンは「ハイブリッドノードと互換性がない」とマークされ、これらのアドオンには、ハイブリッドノードで実行されないようにするためのアンチアフィニティルールが設定されます。詳細については、「ハイブリッドノードのアドオンの設定」を参照してください。インストールする [AWS Marketplace アドオン] が一覧にない場合は、検索ボックスにテキストを入力して、利用可能な [AWS Marketplace アドオン] を検索できます。[カテゴリ]、[ベンダー]、または [価格モデル] で検索し、検索結果からアドオンを選択することもできます。
  - b. CoreDNS や kube-proxy などの一部のアドオンは、デフォルトでインストールされます。デフォルトのアドオンのいずれかを無効にすると、Kubernetes アプリケーションを実行する機能に影響する場合があります。
  - c. このページを読み終えたら、[Next] を選択してください。
7. [選択したアドオン設定の構成] ページで、インストールするバージョンを選択し、[次へ] を選択してください。
- a. クラスターを作成した後はいつでも新しいバージョンに更新できます。クラスターの作成後に、各アドオンの設定を更新できます。アドオンの設定の詳細については「[the section called](#)

[「アドオンを更新する」](#)を参照してください。ハイブリッドノードと互換性のあるアドオンのバージョンについては、[「the section called “CNI アドオンの設定”](#)を参照してください。

b. このページを読み終えたら、[次へ] を選択してください。

8. [確認と作成] ページで、前のページで入力または選択した情報を確認します。変更する必要がある場合は[編集] を選択してください。そのままであれば、[作成] を選択してください。クラスターがプロビジョニングされている間、[ステータス] フィールドに [作成中] と表示されます。クラスターのプロビジョニングには数分かかります。

9. [「the section called “ステップ 3: kubeconfig を更新する”](#)」に進みます。

### ステップ 3: kubeconfig を更新する

eksctl を使用してクラスターを作成した場合、このステップはスキップできます。eksctl によってこのステップは既に完了しているからです。新しいコンテキストを kubectl 設定ファイルに追加して、kubectl がクラスターと通信できるようにします。ファイルを作成および更新する方法の詳細については [「the section called “kubectl を使用してクラスターにアクセスする”](#)」を参照してください。

```
aws eks update-kubeconfig --name CLUSTER_NAME --region AWS_REGION
```

出力例は次のとおりです。

```
Added new context arn:aws:eks:AWS_REGION:111122223333:cluster/CLUSTER_NAME to /home/username/.kube/config
```

次のコマンドを実行して、クラスターとの通信を確認します。

```
kubectl get svc
```

出力例は次のとおりです。

| NAME       | TYPE      | CLUSTER-IP | EXTERNAL-IP | PORT(S) | AGE |
|------------|-----------|------------|-------------|---------|-----|
| kubernetes | ClusterIP | 10.100.0.1 | <none>      | 443/TCP | 28h |

### ステップ 4: クラスターのセットアップ

次のステップとして、ハイブリッドノードをクラスターに参加させるためにアクセスを有効にする手順は、[「the section called “クラスターアクセスを準備する”](#)」を参照してください。

## 既存の Amazon EKS クラスターでハイブリッドノードを有効にするか、設定を変更する

このトピックでは、使用可能なオプションの概要と、Amazon EKS クラスターのハイブリッドノード設定を追加、変更、削除する際の考慮事項について説明します。

Amazon EKS クラスターでハイブリッドノードを使用できるようにするには、オンプレミスノードとポッドネットワーク (オプション) の IP アドレス CIDR 範囲を RemoteNetworkConfig 設定に追加します。EKS では、この CIDR リストを使用して、クラスターとオンプレミスネットワーク間の接続を有効にします。クラスター設定を更新する際のオプションについては、「Amazon EKS API リファレンス」の「[UpdateClusterConfig](#)」でリスト全体を確認できます。

クラスター内にある EKS Hybrid Nodes のネットワーク設定では、次のいずれかのアクションを実行できます。

- [リモートネットワーク設定を追加して、既存のクラスターで EKS Hybrid Nodes を有効にする。](#)
- [既存のクラスター内のリモートノードネットワークまたはリモートポッドネットワークを追加、変更、削除する。](#)
- [すべてのリモートノードネットワーク CIDR 範囲を削除して、既存のクラスターの EKS Hybrid Nodes を無効にする。](#)

### 前提条件

- ハイブリッドノードの Amazon EKS クラスターを有効にする前に、環境が次の要件を満たしていることを確認してください: 「[the section called “前提条件”](#)」で概説されている要件に加え、「[the section called “ネットワークを準備する”](#)」、「[the section called “オペレーティングシステムを準備する”](#)」、「[the section called “認証情報を準備する”](#)」で詳述されている要件
- クラスターは IPv4 アドレスファミリーを使用する必要があります。
- クラスターは、クラスター認証モードに API または API\_AND\_CONFIG\_MAP を使用する必要があります。クラスター認証モードを変更するプロセスについては、「[the section called “\[Authentication mode\] \(認証モード\)”](#)」を参照してください。
- Amazon EKS Kubernetes API サーバーエンドポイントには、パブリックまたはプライベートいずれか (両方ではなく) のエンドポイントアクセスを使用することが推奨されます。「Public and Private (パブリックとプライベート)」を選択した場合、Amazon EKS Kubernetes API サーバーエンドポイントは、常に VPC の外部で稼働しているハイブリッドノードのパブリック IP に解決されるため、ハイブリッドノードがクラスターに参加できなくなる可能性があります。クラスターへ

のネットワークアクセスを変更するプロセスについては、「[the section called “クラスターエンドポイントのアクセス”](#)」を参照してください。

- AWS コマンドラインインターフェイス (AWS CLI) の最新バージョンがデバイスにインストールおよび設定されていること。現在のバージョンを確認するには「aws --version」を参照してください。yum、apt-get、macOS 用の Homebrew などのパッケージマネージャーは、多くの場合 AWS CLI の最新バージョンより数バージョン古くなっています。最新バージョンをインストールするには、「AWS Command Line Interface ユーザーガイド」の「[AWS CLI の最新バージョンのインストールまたは更新](#)」および「[AWS CLI の設定を構成する](#)」を参照してください。
- Amazon EKS クラスターで [UpdateClusterConfig](#) を呼び出す権限を持つ [IAM プリンシパル](#)。
- ハイブリッドノードと互換性のあるバージョンにアドオンを更新します。ハイブリッドノードと互換性のあるアドオンのバージョンについては、「[the section called “CNI アドオンの設定”](#)」を参照してください。
- ハイブリッドノードと互換性のないアドオンを実行している場合は、ハイブリッドノードへのデプロイを防ぐために、アドオンの [DaemonSet](#) または [Deployment](#) に、次のアフィニティルールが設定されていることを確認します。次のアフィニティルールが存在しない場合は追加してください。

```
affinity:
 nodeAffinity:
 requiredDuringSchedulingIgnoredDuringExecution:
 nodeSelectorTerms:
 - matchExpressions:
 - key: eks.amazonaws.com/compute-type
 operator: NotIn
 values:
 - hybrid
```

## 考慮事項

remoteNetworkConfig JSON オブジェクトは、更新中に次のように動作します。

- 指定していない既存の設定部分は変更されません。remoteNodeNetworks または remotePodNetworks のいずれかを指定していない場合、該当箇所への変更は行われません。
- remoteNodeNetworks または remotePodNetworks の CIDR リストを変更する場合は、最終設定に必要な CIDR の完全なリストを指定しなければなりません。remoteNodeNetworks または remotePodNetworks の CIDR リストのいずれかに変更を指定すると、更新中に元のリストが置き換えられます。
- オンプレミスノードとポッド CIDR ブロックは、以下の要件を満たしている必要があります。

1. 次のいずれかの IPv4 RFC-1918 範囲内にあること: 10.0.0.0/8、172.16.0.0/12、192.168.0.0/16
2. Amazon EKS クラスターの VPC の全 CIDR、または Kubernetes サービスの IPv4 CIDR と相互に重複していないこと。

既存のクラスターでハイブリッドノードを有効にする

既存のクラスターで EKS Hybrid Nodes を有効にするには、以下を使用します。

- [AWS CloudFormation](#)
- [AWS CLI](#)
- [AWS Management Console](#)

既存のクラスターで EKS Hybrid Nodes を有効にする - AWS CloudFormation

1. クラスターで EKS Hybrid Nodes を有効にするには、RemoteNodeNetwork と RemotePodNetwork (オプション) を CloudFormation テンプレートに追加して、スタックを更新します。RemoteNodeNetwork は最大 1 つの Cidrs 項目が含まれるリストであり、Cidrs は複数の IP CIDR 範囲が含まれるリストであることに注意してください。

```
RemoteNetworkConfig:
 RemoteNodeNetworks:
 - Cidrs: [RemoteNodeCIDR]
 RemotePodNetworks:
 - Cidrs: [RemotePodCIDR]
```

2. 「[the section called “クラスターアクセスを準備する”](#)」に進みます。

既存のクラスターで EKS Hybrid Nodes を有効にする - AWS CLI

1. EKS クラスターの EKS Hybrid Nodes に使用する RemoteNetworkConfig を有効にするには、以下のコマンドを実行します。コマンドを実行する前に、以下を自分の設定に置き換えます。全設定のリストについては、「Amazon EKS API リファレンス」の「[UpdateClusterConfig](#)」を参照してください。
  - a. CLUSTER\_NAME: 更新する EKS クラスターの名前。
  - b. AWS\_REGION: EKS クラスターが実行されている AWS リージョン。
  - c. REMOTE\_NODE\_CIDRS: ハイブリッドノードのオンプレミスノード CIDR。

- d. `REMOTE_POD_CIDRS` (オプション): ハイブリッドノードで実行されているワークロードのオンプレミスポッド CIDR

```
aws eks update-cluster-config \
 --name CLUSTER_NAME \
 --region AWS_REGION \
 --remote-network-config '{"remoteNodeNetworks":[{"cidrs":
["REMOTE_NODE_CIDRS"]}], "remotePodNetworks":[{"cidrs":["REMOTE_POD_CIDRS"]}]}'
```

2. クラスターの更新には数分かかります。クラスターのステータスのクエリを実行するには次のコマンドを使用します。`CLUSTER_NAME` は変更するクラスターの名前に、`AWS_REGION` はクラスターが稼働している AWS リージョンに置き換えてください。出力が「ACTIVE」を返すまで、次のステップに進まないでください。

```
aws eks describe-cluster \
 --name CLUSTER_NAME \
 --region AWS_REGION \
 --query "cluster.status"
```

3. 「[the section called “クラスターアクセスを準備する”](#)」に進みます。

既存のクラスターで EKS Hybrid Nodes を有効にする - AWS Management Console

1. [Amazon EKS コンソール](#)で Amazon EKS コンソールを開きます。
2. クラスター名を選択すると、そのクラスターの情報を表示されます。
3. [ネットワーキング] タブを開き、[管理] 選択します。
4. ドロップダウンで、[リモートネットワーク] を選択します。
5. [リモートネットワークを設定してハイブリッドノードを有効にする] を選択し、ハイブリッドノードのオンプレミスノードとポッド CIDR を指定します。
6. [変更を保存] を選択して終了します。クラスターのステータスが [アクティブ] に戻るまで待ちます。
7. 「[the section called “クラスターアクセスを準備する”](#)」に進みます。

既存のクラスターでハイブリッドノード設定を更新する

既存のハイブリッドクラスターで `remoteNetworkConfig` を変更するには、以下のいずれかを使用します。

- [AWS CloudFormation](#)
- [AWS CLI](#)
- [AWS Management Console](#)

## 既存のクラスターでハイブリッド設定を更新する - AWS CloudFormation

1. CloudFormation テンプレートを新しいネットワーク CIDR 値で更新します。

```
RemoteNetworkConfig:
 RemoteNodeNetworks:
 - Cidrs: [NEW_REMOTE_NODE_CIDRS]
 RemotePodNetworks:
 - Cidrs: [NEW_REMOTE_POD_CIDRS]
```

### Note

RemoteNodeNetworks または RemotePodNetworks の CIDR リストを更新するときは、すべての CIDR (新規と既存) を含めます。更新中には、リスト全体が置き換えられます。更新リクエストでこれらのフィールドを省略すると、既存の設定が保持されます。

2. 変更したテンプレートで CloudFormation スタックを更新し、スタックの更新が完了するまで待ちます。

## 既存のクラスターでハイブリッド設定を更新する - AWS CLI

1. リモートネットワークの CIDR を変更するには、次のコマンドを実行します。値をご自身の設定に置き換えます。

```
aws eks update-cluster-config
--name CLUSTER_NAME
--region AWS_REGION
--remote-network-config '{"remoteNodeNetworks":[{"cidrs":
["NEW_REMOTE_NODE_CIDRS"]}],"remotePodNetworks":[{"cidrs":
["NEW_REMOTE_POD_CIDRS"]}]}'
```



 Note

remoteNodeNetworks または remotePodNetworks の CIDR リストを更新するときは、すべての CIDR (新規と既存) を含めます。更新中には、リスト全体が置き換えられます。更新リクエストでこれらのフィールドを省略すると、既存の設定が保持されます。

2. クラスターのステータスが [アクティブ] に戻るまで待ち、次に進みます。

## 既存のクラスターでハイブリッド設定を更新する - AWS Management Console

1. [Amazon EKS コンソール](#)で Amazon EKS コンソールを開きます。
2. クラスター名を選択すると、そのクラスターの情報を表示されます。
3. [ネットワーキング] タブを開き、[管理] 選択します。
4. ドロップダウンで、[リモートネットワーク] を選択します。
5. 必要に応じて、Remote node networks と Remote pod networks - Optional の CIDR を更新します。
6. [変更の保存] を選択し、クラスターのステータスが [アクティブ] に戻るのを待ちます。

## 既存のクラスターで Hybrid Nodes を無効にする

既存のクラスターで EKS Hybrid Nodes を無効にするには、以下を使用します。

- [AWS CloudFormation](#)
- [AWS CLI](#)
- [AWS Management Console](#)

## 既存のクラスターで EKS Hybrid Nodes を無効にする - AWS CloudFormation

1. クラスターで EKS Hybrid Nodes を無効にするには、RemoteNodeNetworks と RemotePodNetworks を CloudFormation テンプレートの空の配列に設定して、スタックを更新します。

```
RemoteNetworkConfig:
 RemoteNodeNetworks: []
 RemotePodNetworks: []
```



## 既存のクラスターで EKS Hybrid Nodes を無効にする - AWS CLI

1. クラスターから RemoteNetworkConfig を削除するには、以下のコマンドを実行します。コマンドを実行する前に、以下を自分の設定に置き換えます。全設定のリストについては、「Amazon EKS API リファレンス」の「[UpdateClusterConfig](#)」を参照してください。
  - a. CLUSTER\_NAME: 更新する EKS クラスターの名前。
  - b. AWS\_REGION: EKS クラスターが実行されている AWS リージョン。

```
aws eks update-cluster-config \
 --name CLUSTER_NAME \
 --region AWS_REGION \
 --remote-network-config '{"remoteNodeNetworks":[],"remotePodNetworks":[]}'
```

2. クラスターの更新には数分かかります。クラスターのステータスのクエリを実行するには次のコマンドを使用します。CLUSTER\_NAME は変更するクラスターの名前に、AWS\_REGION はクラスターが稼働している AWS リージョンに置き換えてください。出力が「ACTIVE」を返すまで、次のステップに進まないでください。

```
aws eks describe-cluster \
 --name CLUSTER_NAME \
 --region AWS_REGION \
 --query "cluster.status"
```

## 既存のクラスターで EKS Hybrid Nodes を無効にする - AWS Management Console

1. [Amazon EKS コンソール](#)で Amazon EKS コンソールを開きます。
2. クラスター名を選択すると、そのクラスターの情報を表示されます。
3. [ネットワーキング] タブを開き、[管理] 選択します。
4. ドロップダウンで、[リモートネットワーク] を選択します。
5. [リモートネットワークを設定してハイブリッドノードを有効にする] を選択して、Remote node networks と Remote pod networks - Optional のすべての CIDR を削除します。
6. [変更を保存] を選択して終了します。クラスターのステータスが [アクティブ] に戻るまで待ちます。

## ハイブリッドノードのクラスターアクセスを準備する

ハイブリッドノードを Amazon EKS クラスターに接続するには、クラスターに参加するために Kubernetes のアクセス許可を持つハイブリッドノードの IAM ロールを有効にする必要があります。ハイブリッドノードの IAM ロールの作成方法については、「[the section called “認証情報を準備する”](#)」を参照してください。Amazon EKS は、IAM プリンシパルを Kubernetes ロールベースアクセスコントロール (RBAC) に関連付ける方法として、Amazon EKS アクセスエントリと aws-auth ConfigMap の 2 つをサポートしています。Amazon EKS アクセス管理の詳細については、「[the section called “Kubernetes API アクセス”](#)」を参照してください。

以下の手順を使用して、ハイブリッドノードの IAM ロールを Kubernetes のアクセス許可に関連付けます。Amazon EKS のアクセスエントリを使用するには、クラスターが API または API\_AND\_CONFIG\_MAP 認証モードで作成されている必要があります。aws-auth ConfigMap を使用するには、クラスターが API\_AND\_CONFIG\_MAP 認証モードで作成されている必要があります。CONFIG\_MAP のみの認証モードは、ハイブリッドノードが有効な Amazon EKS クラスターではサポートされていません。

ハイブリッドノードの IAM ロールに Amazon EKS のアクセスエントリを使用する

IAM ロールと組み合わせて使用できる、HYBRID\_LINUX というハイブリッドノード用 Amazon EKS アクセスエントリタイプがあります。このアクセスエントリタイプでは、ユーザー名は自動的に system:node:{{SessionName}} に設定されます。アクセスエントリの作成の詳細については、「[the section called “アクセスエントリを作成する”](#)」を参照してください。

### AWS CLI

1. デバイスに最新バージョンの AWS CLI をインストールし、設定しておく必要があります。現在のバージョンを確認するには「aws --version」を参照してください。yum、apt-get、macOS 用の Homebrew などのパッケージマネージャーは、多くの場合 AWS CLI の最新バージョンより数バージョン古くなっています。最新バージョンをインストールするには、「AWS コマンドラインインターフェイスユーザーガイド」の「インストール」および「aws configure を使用したクイック設定」を参照してください。
2. 以下のコマンドを使用してアクセスエントリを作成します。CLUSTER\_NAME をお使いのクラスターの名前に、HYBRID\_NODES\_ROLE\_ARN を [the section called “認証情報を準備する”](#) のステップで作成したロールの ARN に置き換えます。

```
aws eks create-access-entry --cluster-name CLUSTER_NAME \
 --principal-arn HYBRID_NODES_ROLE_ARN \
 --type HYBRID_LINUX
```

## AWS Management Console

1. [Amazon EKS コンソール](#)で Amazon EKS コンソールを開きます。
2. ハイブリッドノードが有効なクラスターの名前を選択します。
3. [リモートアクセス] タブを選択してください。
4. [アクセスエントリの作成] を選択してください。
5. [IAM プリンシパル] には、「[the section called “認証情報を準備する”](#)」のステップで作成したハイブリッドノードの IAM ロールを選択します。
6. [タイプ] には Hybrid Linux を選択します。
7. (オプション) [タグ] ではアクセスエントリにラベルを割り当てます。例えば、同じタグを持つすべてのリソースを検索しやすくするためです。
8. [レビューと作成にスキップ] を選択します。Hybrid Linux のアクセスエントリにポリシーを追加することや、アクセス範囲を変更することはできません。
9. アクセスエントリの設定を確認してください。何かが正しくないと思われる場合は[戻る] を選択してステップに戻り、エラーを修正します。設定が正しければ、[作成] を選択してください。

### ハイブリッドノードの IAM ロールに aws-auth ConfigMap を使用する

以下のステップでは、[the section called “認証情報を準備する”](#) のステップで作成したハイブリッドノード IAM ロールの ARN を使用して、aws-auth ConfigMap を作成または更新します。

1. クラスターに既存の aws-auth ConfigMap があるかどうかを確認します。特定の kubeconfig ファイルを使用している場合は、--kubeconfig フラグを使用してください。

```
kubectl describe configmap -n kube-system aws-auth
```

2. aws-auth ConfigMap が表示されている場合は、必要に応じて更新してください。
  - a. ConfigMap を編集用を開きます。

```
kubectl edit -n kube-system configmap/aws-auth
```

- b. 必要に応じて新しい mapRoles エントリを追加します。HYBRID\_NODES\_ROLE\_ARN をハイブリッドノード IAM ロールの ARN に置き換えます。なお、{{SessionName}} は ConfigMap に保存するための正しいテンプレート形式です。他の値には置き換えないでください。

```
data:
```

```
mapRoles: |
- groups:
 - system:bootstrappers
 - system:nodes
rolearn: HYBRID_NODES_ROLE_ARN
username: system:node:{{SessionName}}
```

c. ファイルを保存し、テキストエディタを終了します。

3. クラスターに既存の aws-auth ConfigMap がない場合は、以下のコマンドを使用して作成します。HYBRID\_NODES\_ROLE\_ARN をハイブリッドノード IAM ロールの ARN に置き換えます。なお、{{SessionName}} は ConfigMap に保存するための正しいテンプレート形式です。他の値には置き換えしないでください。

```
kubectl apply -f=/dev/stdin <<-EOF
apiVersion: v1
kind: ConfigMap
metadata:
 name: aws-auth
 namespace: kube-system
data:
 mapRoles: |
 - groups:
 - system:bootstrappers
 - system:nodes
 rolearn: HYBRID_NODES_ROLE_ARN
 username: system:node:{{SessionName}}
EOF
```

## ハイブリッドノードでオンプレミスのワークロードを実行する

ハイブリッドノードが有効になっている EKS クラスターではAWS クラウドで使用するのと同じ Amazon EKS クラスター、機能、ツールを使用して、独自のインフラストラクチャでオンプレミスおよびエッジアプリケーションを実行できます。

以下のセクションではハイブリッドノードを使用するためのステップバイステップの手順について説明します。

### トピック

- [ハイブリッドノードを接続する](#)
- [Bottlerocket を実行しているハイブリッドノードの接続](#)

- [クラスターのハイブリッドノードをアップグレードする](#)
- [ハイブリッドノード用のセキュリティ更新プログラムにパッチを適用する](#)
- [ハイブリッドノードを削除する](#)

## ハイブリッドノードを接続する

### Note

次の手順は、互換性のあるオペレーティングシステム (Bottlerocket を除く) を実行しているハイブリッドノードに適用されます。Bottlerocket を実行するハイブリッドノードを接続する手順については、「[the section called “Bottlerocket を実行しているハイブリッドノードの接続”](#)」を参照してください。

このトピックでは、ハイブリッドノードを Amazon EKS クラスターに接続する方法について説明します。ハイブリッドノードがクラスターに参加すると、Amazon EKS コンソールと kubectl などの Kubernetes 互換ツールに準備中ステータスが表示されます。このページのステップを完了したら、「[the section called “CNI を設定する”](#)」に進み、ハイブリッドノードでアプリケーションを実行する準備をします。

### 前提条件

ハイブリッドノードを Amazon EKS クラスターに接続する前に、前提条件の手順が完了していることを確認してください。

- オンプレミス環境から Amazon EKS クラスターをホストする AWS リージョンへのネットワーク接続があること。詳細については「[the section called “ネットワークを準備する”](#)」を参照してください。
- ハイブリッドノード用の互換性のあるオペレーティングシステムがオンプレミスホストにインストールされていること。詳細については「[the section called “オペレーティングシステムを準備する”](#)」を参照してください。
- ハイブリッドノードの IAM ロールを作成し、オンプレミス認証情報プロバイダー (AWS Systems Manager ハイブリッドアクティベーションまたは AWS IAM Roles Anywhere) をセットアップしていること。詳細については「[the section called “認証情報を準備する”](#)」を参照してください。
- ハイブリッドノード対応の Amazon EKS クラスターを作成していること。詳細については「[the section called “クラスターを作成する”](#)」を参照してください。

- ハイブリッドノードの IAM ロールを、Kubernetes ロールベースのアクセスコントロール (RBAC) のアクセス許可に関連付けていること。詳細については「[the section called “クラスターアクセスを準備する”](#)」を参照してください。

## ステップ 1: ハイブリッドノード CLI (**nodeadm**) を各オンプレミスホストにインストールする

構築済みのオペレーティングシステムイメージに Amazon EKS Hybrid Nodes CLI (nodeadm) を含める場合は、このステップをスキップできます。nodeadm のハイブリッドノード版の詳細については、「[the section called “ハイブリッドノード用 nodeadm”](#)」を参照してください。

nodeadm のハイブリッドノード版は、Amazon CloudFront をフロントに置いた Amazon S3 でホストされます。各オンプレミスホストに nodeadm をインストールするにはオンプレミスホストから以下のコマンドを実行してください。

x86\_64 ホストの場合:

```
curl -OL 'https://hybrid-assets.eks.amazonaws.com/releases/latest/bin/linux/amd64/nodeadm'
```

ARM ホストの場合

```
curl -OL 'https://hybrid-assets.eks.amazonaws.com/releases/latest/bin/linux/arm64/nodeadm'
```

各ホストでダウンロードしたバイナリに実行可能ファイルのアクセス許可を追加します。

```
chmod +x nodeadm
```

## ステップ 2: **nodeadm** を使用してハイブリッドノードの依存関係をインストールする

構築済みのオペレーティングシステムイメージにハイブリッドノードの依存関係をインストールする場合は、このステップをスキップできます。nodeadm install コマンドを使用して、ハイブリッドノードに必要なすべての依存関係をインストールできます。ハイブリッドノードの依存関係には、containerd、kubelet、kubectrl、AWS SSM または AWS IAM Roles Anywhere コンポーネントが含まれます。nodeadm install によってインストールされるコンポーネントとファイルの場所の詳細については、「[the section called “ハイブリッドノード用 nodeadm”](#)」を参照してください。オンプレミスファイアウォールで nodeadm install プロセスのために許可する必要があるドメインの詳細については、「[the section called “ネットワークを準備する”](#)」を参照してください。

以下のコマンドを実行して、ハイブリッドノードの依存関係をオンプレミスホストにインストールします。以下のコマンドは、ホストで sudo/root アクセスを持つユーザーで実行する必要があります。

**⚠ Important**

ハイブリッドノードの CLI (nodeadm) は、ホストで sudo/root アクセスを持つユーザーで実行する必要があります。

- K8S\_VERSION を、1.31 などの Amazon EKS クラスターの Kubernetes マイナーバージョンに置き換えます。サポートされている Kubernetes バージョンのリストについては、「[the section called “Kubernetes バージョン”](#)」を参照してください。
- CRED\_PROVIDER を、使用しているオンプレミスの認証情報プロバイダーに置き換えます。有効な値は、AWS SSM の場合は ssm、AWS IAM Roles Anywhere の場合は iam-ra です。

```
nodeadm install K8S_VERSION --credential-provider CRED_PROVIDER
```

### ステップ 3: ハイブリッドノードをクラスターに接続する

ハイブリッドノードをクラスターに接続する前に、Amazon EKS コントロールプレーンとハイブリッドノード間の通信のために、オンプレミスファイアウォールとクラスターのセキュリティグループで必要なアクセスを許可していることを確認してください。このステップのほとんどの問題は、ファイアウォール設定、セキュリティグループ設定、またはハイブリッドノードの IAM ロール設定に関連しています。

**⚠ Important**

ハイブリッドノードの CLI (nodeadm) は、ホストで sudo/root アクセスを持つユーザーで実行する必要があります。

1. デプロイの値を使用して、各ホストに nodeConfig.yaml ファイルを作成します。使用可能な構成設定の詳細については、「[the section called “ハイブリッドノード用 nodeadm”](#)」を参照してください。ハイブリッドノードの IAM ロールに eks:DescribeCluster アクションのアクセス許可がない場合は、nodeConfig.yaml のクラスターセクションで Kubernetes API エンドポイント、クラスター CA バンドル、および Kubernetes サービス IPv4 CIDR を渡す必要があります。



- a. オンプレミス認証情報プロバイダーに AWS SSM ハイブリッドアクティベーションを使用している場合は、以下の `nodeConfig.yaml` の例を使用します。
  - i. `CLUSTER_NAME` を自分のクラスター名に置き換えます。
  - ii. `AWS_REGION` を、クラスターがホストされている AWS リージョンに置き換えます。例えば、`us-west-2`。
  - iii. `ACTIVATION_CODE` を、AWS SSM ハイブリッドアクティベーションの作成時に受け取ったアクティベーションコードに置き換えます。詳細については「[the section called “認証情報を準備する”](#)」を参照してください。
  - iv. `ACTIVATION_ID` を、AWS SSM ハイブリッドアクティベーションの作成時に受け取ったアクティベーション ID に置き換えます。この情報は、AWS Systems Manager コンソールまたは AWS CLI `aws ssm describe-activations` コマンドから取得できます。

```
apiVersion: node.eks.aws/v1alpha1
kind: NodeConfig
spec:
 cluster:
 name: CLUSTER_NAME
 region: AWS_REGION
 hybrid:
 ssm:
 activationCode: ACTIVATION_CODE
 activationId: ACTIVATION_ID
```

- b. オンプレミス認証情報プロバイダーに AWS IAM Roles Anywhere を使用している場合は、以下の `nodeConfig.yaml` の例を使用します。
  - i. `CLUSTER_NAME` を自分のクラスター名に置き換えます。
  - ii. `AWS_REGION` を、クラスターがホストされている AWS リージョンに置き換えます。例えば、`us-west-2`。
  - iii. `NODE_NAME` をノードの名前に置き換えます。ハイブリッドノードの IAM ロールの信頼ポリシーを `"sts:RoleSessionName": "${aws:PrincipalTag/x509Subject/CN}"` リソース条件で設定した場合、ノード名はホスト上の証明書の CN と一致する必要があります。使用する `nodeName` は 64 文字以下にする必要があります。
  - iv. `TRUST_ANCHOR_ARN` を、ハイブリッドノードの認証情報を準備する手順で設定したトラストアンカーの ARN に置き換えます。
  - v. `PROFILE_ARN` を、[the section called “認証情報を準備する”](#) のステップで設定したトラストアンカーの ARN に置き換えます。



- vi. `ROLE_ARN` をハイブリッドノード IAM ロールの ARN に置き換えます。
- vii. `CERTIFICATE_PATH` をノード証明書へのディスク内のパスに置き換えます。指定しなかった場合、デフォルトは `/etc/iam/pki/server.pem` です。
- viii. `KEY_PATH` を、証明書のプライベートキーへのディスク内のパスに置き換えます。指定しなかった場合、デフォルトは `/etc/iam/pki/server.key` です。

```
apiVersion: node.eks.aws/v1alpha1
kind: NodeConfig
spec:
 cluster:
 name: CLUSTER_NAME
 region: AWS_REGION
 hybrid:
 iamRolesAnywhere:
 nodeName: NODE_NAME
 trustAnchorArn: TRUST_ANCHOR_ARN
 profileArn: PROFILE_ARN
 roleArn: ROLE_ARN
 certificatePath: CERTIFICATE_PATH
 privateKeyPath: KEY_PATH
```

2. `nodeConfig.yaml` で `nodeadm init` コマンドを実行して、ハイブリッドノードを Amazon EKS クラスターに接続します。

```
nodeadm init -c file:///nodeConfig.yaml
```

上記のコマンドが正常に完了すれば、ハイブリッドノードは Amazon EKS クラスターに参加したことになります。これは、Amazon EKS コンソールでクラスターの [コンピューティング] タブに移動する ([IAM プリンシパルに表示権限があることを確認してください](#)) か、または `kubectl get nodes` を使用して確認できます。

#### Important

ノードのステータスは `Not Ready` になります。これは、ハイブリッドノードで実行されている CNI がないことが原因であり、予想通りのことです。ノードがクラスターに参加しなかった場合は、「[the section called “トラブルシューティング”](#)」を参照してください。

## ステップ 4: ハイブリッドノードの CNI を設定する

ハイブリッドノードでアプリケーションを実行する準備を整えるには、「[the section called “CNI を設定する”](#)」の手順に進みます。

### Bottlerocket を実行しているハイブリッドノードの接続

このトピックでは、Bottlerocket を実行しているハイブリッドノードを Amazon EKS クラスターに接続する方法について解説します。[Bottlerocket](#) は、AWS が支援およびサポートしているオープンソースの Linux ディストリビューションです。Bottlerocket は、コンテナワークロードをホストするために専用に構築されています。Bottlerocket を使用すると、コンテナインフラストラクチャの更新を自動化することで、コンテナ化されたデプロイの可用性を向上させ、運用コストを削減できます。Bottlerocket には、コンテナの実行に不可欠なソフトウェアのみが含まれており、リソース使用率の向上、セキュリティ上の脅威の軽減、管理オーバーヘッドの軽減が実現されます。

EKS Hybrid Nodes でサポートされているのは、Bottlerocket のバージョンが v1.37.0 以降である VMware のバリエーションのみです。Bottlerocket の VMware のバリエーションは、Kubernetes のバージョン v1.28 以降で使用できます。これらのバリエーション OS イメージには、kubelet、containerd、aws-iam-authenticator、ならびに EKS Hybrid Nodes のその他のソフトウェアの前提条件、が含まれています。これらのコンポーネントは、Bottlerocket のブートストラップと管理コンテナ向けの、base64 でエンコードされたユーザーデータを含む、Bottlerocket の[設定](#)ファイルを使用して設定できます。これらを設定することで、Bottlerocket は、ハイブリッドノードの認証情報プロバイダーを使用して、クラスターのハイブリッドノードを認証できるようになります。ハイブリッドノードをクラスターに追加すると、Amazon EKS コンソールや Kubernetes 互換ツール (kubectl など) にそれらのステータスが Not Ready と表示されます。このページのステップを完了したら、「[the section called “CNI を設定する”](#)」に進み、ハイブリッドノードでアプリケーションを実行する準備をします。

### 前提条件

ハイブリッドノードを Amazon EKS クラスターに接続する前に、前提条件の手順が完了していることを確認してください。

- オンプレミス環境から Amazon EKS クラスターをホストする AWS リージョンへのネットワーク接続があること。詳細については「[the section called “ネットワークを準備する”](#)」を参照してください。
- ハイブリッドノードの IAM ロールを作成し、オンプレミス認証情報プロバイダー (AWS Systems Manager ハイブリッドアクティベーションまたは AWS IAM Roles Anywhere) をセットアップしていること。詳細については「[the section called “認証情報を準備する”](#)」を参照してください。

- ハイブリッドノード対応の Amazon EKS クラスターを作成していること。詳細については「[the section called “クラスターを作成する”](#)」を参照してください。
- ハイブリッドノードの IAM ロールを、Kubernetes ロールベースのアクセスコントロール (RBAC) のアクセス許可に関連付けていること。詳細については「[the section called “クラスターアクセスを準備する”](#)」を参照してください。

## ステップ 1: Bottlerocket 設定の TOML ファイルを作成する

ハイブリッドノード用に Bottlerocket を設定するには、`settings.toml` ファイルを必要とする設定で作成する必要があります。TOML ファイルのコンテンツは、使用している認証情報プロバイダー (SSM または IAM Roles Anywhere) に応じて異なります。このファイルは、Bottlerocket インスタンスをプロビジョニングするときに、ユーザーデータとして渡されます。

### SSM

AWS Systems Manager を認証情報プロバイダーとして使用している場合は、コンテンツを次のようにして `settings.toml` ファイルを作成します。

```
[settings.kubernetes]
cluster-name = "<cluster-name>"
api-server = "<api-server-endpoint>"
cluster-certificate = "<cluster-certificate-authority>"
hostname-override = "<hostname>"
provider-id = "eks-hybrid:///<region>/<cluster-name>/<hostname>"
authentication-mode = "aws"

[settings.network]
hostname = "<hostname>"

[settings.aws]
region = "<region>"

[settings.kubernetes.node-labels]
"eks.amazonaws.com/compute-type" = "hybrid"
"eks.amazonaws.com/hybrid-credential-provider" = "ssm"

[settings.host-containers.admin]
enabled = true
user-data = "<base64-encoded-admin-container-userdata>"

[settings.bootstrap-containers.eks-hybrid-setup]
```

```
mode = "always"
user-data = "<base64-encoded-bootstrap-container-userdata>"

[settings.host-containers.control]
enabled = true
```

プレースホルダを次の値に置き換えます。

- `<cluster-name>`: Amazon EKS クラスターの名前
- `<api-server-endpoint>`: クラスターの API サーバーエンドポイント
- `<cluster-certificate-authority>`: クラスターの base64 エンコードされた CA バンドル
- `<region>`: クラスターをホストしている AWS リージョン。例: "us-east-1"
- `<hostname>`: Bottlerocket インスタンスのホスト名。これはノード名としても設定されます。こちらは任意の一意の値にすることができますが、[Kubernetes オブジェクトの命名規則](#)に従う必要があります。さらに、使用するホスト名は 64 文字以下にする必要があります。注: SSM プロバイダーを使用する場合、このホスト名とノード名は、インスタンスが SSM に登録された後、マネージドインスタンス ID (mi-\* ID など) に置き換えられます。
- `<base64-encoded-admin-container-userdata>`: Bottlerocket 管理コンテナの設定の、base64 でエンコードされたコンテンツ。管理コンテナを有効にすると、SSH を使用して Bottlerocket インスタンスに接続し、システムの探索やデバッグを行えるようになります。これは必須の設定ではありませんが、トラブルシューティングを容易にするため有効にしておくことが推奨されます。管理コンテナを使用した認証の詳細については [Bottlerocket 管理コンテナのドキュメント](#)を参照してください。管理者コンテナは、以下の例に示すように、SSH ユーザーとキーの入力を JSON 形式で受け取ります。

```
{
 "user": "<ssh-user>",
 "ssh": {
 "authorized-keys": [
 "<ssh-authorized-key>"
]
 }
}
```

- `<base64-encoded-bootstrap-container-userdata>`: Bottlerocket ブートストラップコンテナの設定の、base64 でエンコードされたコンテンツ。設定の詳細については [Bottlerocket ブートストラップコンテナのドキュメント](#)を参照してください。ブートストラップコンテナは、イン

スタンスを AWS SSM マネージドインスタンスとして登録し、これを Amazon EKS クラスターの Kubernetes ノードとして結合する役割を果たします。ブートストラップコンテナに渡されるユーザーデータは、以前に作成した SSM ハイブリッドアクティベーションコードと ID を入力として受け入れる、コマンド呼び出しの形式をとります。

```
eks-hybrid-ssm-setup --activation-id=<activation-id> --activation-code=<activation-code> --region=<region>
```

## IAM Roles Anywhere

AWS IAM Roles Anywhere を認証情報プロバイダーとして使用している場合は、コンテンツを次のようにして `settings.toml` ファイルを作成します。

```
[settings.kubernetes]
cluster-name = "<cluster-name>"
api-server = "<api-server-endpoint>"
cluster-certificate = "<cluster-certificate-authority>"
hostname-override = "<hostname>"
provider-id = "eks-hybrid:///<region>/<cluster-name>/<hostname>"
authentication-mode = "aws"

[settings.network]
hostname = "<hostname>"

[settings.aws]
region = "<region>"
config = "<base64-encoded-aws-config-file>"

[settings.kubernetes.node-labels]
"eks.amazonaws.com/compute-type" = "hybrid"
"eks.amazonaws.com/hybrid-credential-provider" = "iam-ra"

[settings.host-containers.admin]
enabled = true
user-data = "<base64-encoded-admin-container-userdata>"

[settings.bootstrap-containers.eks-hybrid-setup]
mode = "always"
user-data = "<base64-encoded-bootstrap-container-userdata>"
```

プレースホルダを次の値に置き換えます。

- <cluster-name>: Amazon EKS クラスターの名前
- <api-server-endpoint>: クラスターの API サーバーエンドポイント
- <cluster-certificate-authority>: クラスターの base64 エンコードされた CA バンドル
- <region>: クラスターをホストしている AWS リージョン。例: "us-east-1"
- <hostname>: Bottlerocket インスタンスのホスト名。これはノード名としても設定されます。こちらは任意の一意の値にすることができますが、[Kubernetes オブジェクトの命名規則](#)に従う必要があります。さらに、使用するホスト名は 64 文字以下にする必要があります。注: IAM-RA プロバイダーを使用しているときは、ノード名は、ハイブリッドノードの IAM ロールの信頼ポリシーを "sts:RoleSessionName": "\${aws:PrincipalTag/x509Subject/CN}" リソース条件で設定した場合にホストの証明書の CN と一致している必要があります。
- <base64-encoded-aws-config-file>: AWS 設定ファイルの base64 でエンコードされたコンテンツ。ファイルのコンテンツは以下のようになるはずです。

```
[default]
credential_process = aws_signing_helper credential-process --certificate /root/.aws/
node.crt --private-key /root/.aws/node.key --profile-arn <profile-arn> --role-arn
<role-arn> --trust-anchor-arn <trust-anchor-arn> --role-session-name <role-session-
name>
```

- <base64-encoded-admin-container-userdata>: Bottlerocket 管理コンテナの設定の、base64 でエンコードされたコンテンツ。管理コンテナを有効にすると、SSH を使用して Bottlerocket インスタンスに接続し、システムの探索やデバッグを行えるようになります。これは必須の設定ではありませんが、トラブルシューティングを容易にするため有効にしておくことが推奨されます。管理コンテナを使用した認証の詳細については [Bottlerocket 管理コンテナのドキュメント](#)を参照してください。管理者コンテナは、以下の例に示すように、SSH ユーザーとキーの入力を JSON 形式で受け取ります。

```
{
 "user": "<ssh-user>",
 "ssh": {
 "authorized-keys": [
 "<ssh-authorized-key>"
]
 }
}
```

- <base64-encoded-bootstrap-container-userdata>: Bottlerocket ブートストラップコンテナの設定の、base64 でエンコードされたコンテンツ。設定の詳細については [Bottlerocket ブートストラップコンテナのドキュメント](#) を参照してください。ブートストラップコンテナは、インスタンスに IAM Roles Anywhere ホスト証明書と証明書プライベートキーファイルを作成する役割を果たします。これらはその後、Amazon EKS クラスターで認証を行うときに、一時的な認証情報を取得するために aws\_signing\_helper が使用します。ブートストラップコンテナに渡されるユーザーデータは、以前に作成した証明書とプライベートキーのコンテンツを入力として受け入れる、コマンド呼び出しの形式をとります。

```
eks-hybrid-iam-ra-setup --certificate=<certificate> --key=<private-key>
```

## ステップ 2: ユーザーデータを使用して Bottlerocket vSphere VM をプロビジョニングする

TOML ファイルを作成したら、vSphere VM の作成時にこれをユーザーデータとして渡します。ユーザーデータは VM の電源を初めて入れる前に設定しておく必要があることを忘れないでください。そのため、インスタンスの作成時にこれを指定しておく必要があります。または、VM を事前に作成する場合は、ユーザーデータを設定するまで VM のステータスを poweredOff にしておく必要があります。govc CLI を使用する場合は以下に示します。

### VM を初めて作成する

```
govc vm.create \
 -on=true \
 -c=2 \
 -m=4096 \
 -net.adapter=<network-adapter> \
 -net=<network-name> \
 -e guestinfo.userdata.encoding="base64" \
 -e guestinfo.userdata="$(base64 -w0 settings.toml)" \
 -template=<template-name> \
 <vm-name>
```

### 既存の VM のユーザーデータを更新する

```
govc vm.create \
 -on=false \
 -c=2 \
 -m=4096 \
 -net.adapter=<network-adapter> \
 -net=<network-name> \
 <vm-name>
```

```
-template=<template-name> \
<vm-name>

govc vm.change
 -vm <vm-name> \
 -e guestinfo.userdata="$(base64 -w0 settings.toml)" \
 -e guestinfo.userdata.encoding="base64"

govc vm.power -on <vm-name>
```

上記のセクションでは、`-e guestinfo.userdata.encoding="base64"` オプションでユーザーデータが base64 でエンコードされていることを指定します。`-e guestinfo.userdata` オプションは、`settings.toml` ファイルの base64 でエンコードされたコンテンツを、ユーザーデータとして Bottlerocket インスタンスに渡します。プレースホルダーを Bottlerocket OVA テンプレートやネットワークの詳細など特定の値に置き換えます。

### ステップ 3: ハイブリッドノードの接続を検証する

Bottlerocket インスタンスは、起動すると、Amazon EKS クラスターに結合しようとしします。Amazon EKS コンソールでこの接続を検証するには、クラスターの [コンピューティング] タブに移動するか、次のコマンドを実行します。

```
kubectl get nodes
```

#### Important

ノードのステータスは `Not Ready` になります。これは、ハイブリッドノードで実行されている CNI がないことが原因であり、予想通りのことです。ノードがクラスターに参加しなかった場合は、[「the section called “トラブルシューティング”」](#) を参照してください。

### ステップ 4: ハイブリッドノードの CNI を設定する

ハイブリッドノードでアプリケーションを実行する準備を整えるには、[「the section called “CNI を設定する”」](#) の手順に進みます。

### クラスターのハイブリッドノードをアップグレードする

ハイブリッドノードのアップグレードに関するガイダンスは、Amazon EC2 で実行されるセルフマネージド Amazon EKS ノードと類似しています。ターゲットの Kubernetes バージョンで新しいハイブリッドノードを作成し、既存のアプリケーションを新しい Kubernetes バージョンのハイブリッ



ドノードに円滑に移行し、古い Kubernetes バージョンのハイブリッドノードはクラスターから削除することをお勧めします。アップグレードを開始する前に、アップグレードに関する「[Amazon EKS ベストプラクティス](#)」を確認してください。Amazon EKS Hybrid Nodes には、標準サポートや拡張サポートなどの、クラウドノードを持つ Amazon EKS クラスターに対して同じ [Kubernetes バージョン](#) のサポートがあります。

Amazon EKS Hybrid Nodes は、アップストリーム Kubernetes と同じノードの [バージョンスキューポリシー](#) に従います。Amazon EKS Hybrid Nodes は、Amazon EKS コントロールプレーンよりも新しいバージョンにすることはできません。また、ハイブリッドノードは、Amazon EKS のコントロールプレーンのマイナーバージョンより、最大 3 つ前まで古い Kubernetes のマイナーバージョンである可能性があります。

カットオーバー移行のアップグレード戦略のためにターゲットの Kubernetes バージョンに新しいハイブリッドノードを作成する空き容量がない場合は、代わりに Amazon EKS Hybrid Nodes CLI (nodeadm) を使用してハイブリッドノードの Kubernetes バージョンをインプレースでアップグレードできます。

#### Important

nodeadm を使用してインプレースでハイブリッドノードをアップグレードしている場合、古いバージョンの Kubernetes コンポーネントがシャットダウンされ、新しい Kubernetes バージョンコンポーネントがインストールされて起動されるプロセス中に、ノードのダウンタイムが発生します。

## 前提条件

アップグレードする前に、以下の前提条件を満たしていることを確認してください。

- ハイブリッドノードのアップグレードのターゲット Kubernetes バージョンは、Amazon EKS コントロールプレーンのバージョン以下である必要があります。
- カットオーバー移行のアップグレード戦略に従う場合、ターゲット Kubernetes バージョンにインストールする新しいハイブリッドノードは、[the section called “前提条件”](#) 要件を満たしている必要があります。この要件には、Amazon EKS クラスターの作成時に渡した Remote Node Network CIDR 内に IP アドレスがあることが含まれます。
- カットオーバー移行とインプレースアップグレードの両方について、ハイブリッドノードは、ハイブリッドノードの依存関係の新しいバージョンを取得するために [必要なドメイン](#) にアクセスできる必要があります。

- Amazon EKS Kubernetes API エンドポイントとのやり取りに使用するローカルマシンまたはインスタンスに kubectl をインストールしてする必要があります。
- CNI のバージョンは、アップグレード先の Kubernetes バージョンをサポートしている必要があります。そうでない場合は、ハイブリッドノードをアップグレードする前に CNI バージョンをアップグレードします。詳細については「[the section called “CNI を設定する”](#)」を参照してください。

## カットオーバー移行 (Blue/Green) アップグレード

カットオーバー移行のアップグレードとは、ターゲット Kubernetes バージョンを使用して新しいホストに新しいハイブリッドノードを作成し、既存のアプリケーションをターゲット Kubernetes バージョンの新しいハイブリッドノードに円滑に移行し、クラスターから古い Kubernetes のバージョンのハイブリッドノードを削除するプロセスを指します。この戦略は Blue/Green 移行とも呼ばれます。

1. [the section called “ハイブリッドノードを接続する”](#) 手順に従って、新しいホストをハイブリッドノードとして接続します。nodeadm install コマンドを実行するときは、ターゲット Kubernetes バージョンを使用します。
2. ターゲット Kubernetes バージョンの新しいハイブリッドノードと古い Kubernetes バージョンのハイブリッドノード間の通信を有効にします。この設定により、ターゲット Kubernetes バージョンのハイブリッドノードにワークロードを移行している間に、ポッドが相互に通信できるようになります。
3. ターゲット Kubernetes バージョンのハイブリッドノードがクラスターに正常に参加しており、ステータスが Ready (準備完了) であることを確認します。
4. 以下のコマンドを使用して、削除する各ノードをスケジュール不可としてマークします。これは、置き換えるノード上で新しいポッドがスケジュールまたは再スケジュールされないようにするためです。詳細については、Kubernetes のドキュメントの「[kubectl cordon](#)」を参照してください。NODE\_NAME を古い Kubernetes バージョンのハイブリッドノードの名前に置き換えます。

```
kubectl cordon NODE_NAME
```

次のコードスニペットを使用して、特定の Kubernetes バージョン (この場合は 1.28) のすべてのノードを識別して隔離できます。

```
K8S_VERSION=1.28
```

```
for node in $(kubectl get nodes -o json | jq --arg K8S_VERSION
"$K8S_VERSION" -r '.items[] | select(.status.nodeInfo.kubeletVersion |
match("\($K8S_VERSION)")).metadata.name')
do
 echo "Cordoning $node"
 kubectl cordon $node
done
```

5. 現在のデプロイメントで、ハイブリッドノードで実行している CoreDNS レプリカが 2 つ未満の場合は、そのデプロイメントを少なくとも 2 つのレプリカにスケールアウトします。通常のオペレーションでは、回復力を高めるために、ハイブリッドノードで少なくとも 2 つの CoreDNS レプリカを実行することをお勧めします。

```
kubectl scale deployments/coredns --replicas=2 -n kube-system
```

6. 次のコマンドを使用して、クラスターから削除する古い Kubernetes バージョンのハイブリッドノードをそれぞれドレインします。ノードドレインのドレイン処理に関する詳細については、Kubernetes ドキュメントの「[Safely Drain a Node](#)」を参照してください。NODE\_NAME を古い Kubernetes バージョンのハイブリッドノードの名前に置き換えます。

```
kubectl drain NODE_NAME --ignore-daemonsets --delete-emptydir-data
```

以下のコードスニペットを使用して、特定の Kubernetes バージョン (この場合は 1.28) のすべてのノードを識別してドレインすることができます。

```
K8S_VERSION=1.28
for node in $(kubectl get nodes -o json | jq --arg K8S_VERSION
"$K8S_VERSION" -r '.items[] | select(.status.nodeInfo.kubeletVersion |
match("\($K8S_VERSION)")).metadata.name')
do
 echo "Draining $node"
 kubectl drain $node --ignore-daemonsets --delete-emptydir-data
done
```

7. nodeadm を使用して、ホストからハイブリッドノードアーティファクトを停止および削除できます。root/sudo 権限を持つユーザーで nodeadm を実行する必要があります。デフォルトでは、ノードにポッドが残っている場合、nodeadm uninstall は続行されません。詳細については、「[the section called “ハイブリッドノード用 nodeadm”](#)」を参照してください。

```
nodeadm uninstall
```

8. ハイブリッドノードのアーティファクトを停止およびアンインストールしたら、クラスターからノードリソースを削除します。

```
kubectl delete node node-name
```

次のコードスニペットを使用して、特定の Kubernetes バージョン (この場合は 1.28) のすべてのノードを識別および削除できます。

```
K8S_VERSION=1.28
for node in $(kubectl get nodes -o json | jq --arg K8S_VERSION
"$K8S_VERSION" -r '.items[] | select(.status.nodeInfo.kubeletVersion |
match("\($K8S_VERSION)")).metadata.name')
do
 echo "Deleting $node"
 kubectl delete node $node
done
```

9. CNI の選択によっては、上記のステップを実行した後にハイブリッドノードにアーティファクトが残っている場合があります。詳細については「[the section called “CNI を設定する”](#)」を参照してください。

## インプレースアップグレード

インプレースアップグレードプロセスとは、`nodeadm upgrade` を使用して、新しい物理ホストまたは仮想ホストやカットオーバー移行戦略を使用せずに、ハイブリッドノードの Kubernetes バージョンをアップグレードすることを指します。この `nodeadm upgrade` プロセスでは、ハイブリッドノードで実行されている既存の古い Kubernetes コンポーネントのシャットダウン、既存の古い Kubernetes コンポーネントのアンインストール、新しいターゲット Kubernetes コンポーネントのインストールを行い、新しいターゲット Kubernetes コンポーネントを起動します。ハイブリッドノードで実行されているアプリケーションへの影響を最小限に抑えるために、一度にアップグレードするノードは 1 つにすることを強くお勧めします。このプロセスの期間は、ネットワーク帯域幅とレイテンシーによって異なります。

1. アップグレードするノードをスケジューラ不可としてマークするには、次のコマンドを使用します。これは、アップグレードするノード上で新しいポッドがスケジューラまたは再スケジューラされないようにするためです。詳細については、Kubernetes のドキュメントの「[kubectl cordon](#)」を参照してください。NODE\_NAME をアップグレードするハイブリッドノードの名前に置き換える

```
kubectl cordon NODE_NAME
```

2. 次のコマンドを使用して、アップグレードするノードをドレインします。ノードドレインのドレイン処理に関する詳細については、Kubernetes ドキュメントの「[Safely Drain a Node](#)」を参照してください。NODE\_NAME をアップグレードするハイブリッドノードの名前に置き換えます。

```
kubectl drain NODE_NAME --ignore-daemonsets --delete-emptydir-data
```

3. アップグレードするハイブリッドノードで `nodeadm upgrade` を実行します。root/sudo 権限を持つユーザーで `nodeadm` を実行する必要があります。ノードの名前は、AWS SSM と AWS IAM Roles Anywhere の両方の認証情報プロバイダーのアップグレードによって保持されます。アップグレードプロセス中に認証情報プロバイダーを変更することはできません。nodeConfig.yaml の設定値については、「[the section called “ハイブリッドノード用 nodeadm”](#)」を参照してください。K8S\_VERSION を、アップグレード先のターゲット Kubernetes バージョンに置き換えます。

```
nodeadm upgrade K8S_VERSION -c file://nodeConfig.yaml
```

4. アップグレード後にノード上で Pod をスケジュールできるように、以下を入力します。NODE\_NAME をノードの名前に置き換えます。

```
kubectl uncordon NODE_NAME
```

5. ハイブリッドノードのステータスを監視し、ノードがシャットダウンするのを待ち、Ready (準備完了) ステータスになったら新しい Kubernetes バージョンで再起動します。

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

## ハイブリッドノード用のセキュリティ更新プログラムにパッチを適用する

このトピックでは、ハイブリッドノードで実行されている特定のパッケージと依存関係を対象としたセキュリティ更新プログラムにインプレースでパッチを適用する手順について説明します。ベストプラクティスとして、ハイブリッドノードを定期的に更新して CVE とセキュリティパッチを受け取ることをお勧めします。

Kubernetes バージョンをアップグレードする手順については、「[the section called “ハイブリッドノードをアップグレードする”](#)」を参照してください。

セキュリティパッチの適用が必要になるソフトウェアの一例が `containerd` です。

## Containerd

containerd は、標準の Kubernetes コンテナランタイムであり、EKS ハイブリッドノードとはコアな依存関係にあります。その用途は、イメージのプルやコンテナ実行の管理などコンテナのライフサイクルを管理することです。ハイブリッドノードでは、[nodeadm CLI](#) を利用するか手動で containerd をインストールできます。ノードのオペレーティングシステムに応じて、nodeadm は OS 配布のパッケージまたは Docker パッケージから containerd をインストールします。

containerd の CVE が公開されたら、以下のオプションを利用して、ハイブリッドノードで containerd をパッチ適用済みのバージョンにアップグレードできます。

ステップ 1: パッチがパッケージマネージャーに公開されているかどうかを確認する

containerd CVE パッチがそれぞれの OS パッケージマネージャーに公開されているかどうか、対応するセキュリティ速報を参照することで確認できます。

- [Amazon Linux 2023](#)
- [RHEL](#)
- [Ubuntu 20.04](#)
- [Ubuntu 22.04](#)
- [Ubuntu 24.04](#)

Docker リポジトリを containerd のソースとして使用している場合は、[Docker セキュリティのお知らせ](#)をチェックして、Docker リポジトリからパッチ適用済みのバージョンを入手できるかどうかを確認できます。

ステップ 2: パッチのインストール方法を選択する

ノードにインプレースでパッチを適用し、セキュリティアップグレードをインストールする方法が 3 つあります。どの方法を使用できるかは、パッチをパッケージマネージャーでオペレーティングシステムから入手できるかどうかによって異なります。

1. nodeadm upgrade を使用してパッケージマネージャーに公開されているパッチをインストールします。「[ステップ 2 a](#)」を参照してください。
2. パッケージマネージャーで直接パッチをインストールします。「[ステップ 2 b](#)」を参照してください。

3. パッケージマネージャーに公開されていないカスタムパッチをインストールします。containerd に対するカスタムパッチについては特別な考慮事項があるので注意してください。「[ステップ 2 c](#)」を参照してください。

#### ステップ 2 a: **nodeadm upgrade** でパッチを適用する

containerd CVE パッチが OS や Docker リポジトリ (Apt または RPM) に公開されたことを確認したら、`nodeadm upgrade` コマンドを使用して containerd を最新バージョンにアップグレードできます。これは Kubernetes バージョンのアップグレードではないため、現在の Kubernetes バージョンを `nodeadm` アップグレードコマンドに渡す必要があります。

```
nodeadm upgrade K8S_VERSION --config-source file:///root/nodeConfig.yaml
```

#### ステップ 2 b: オペレーティングシステムのパッケージマネージャーでパッチを適用する

あるいは、それぞれのパッケージマネージャーから更新することも可能であり、それを使用して以下のように containerd パッケージをアップグレードできます。

##### Amazon Linux 2023

```
sudo yum update -y
sudo yum install -y containerd
```

##### RHEL

```
sudo yum install -y yum-utils
sudo yum-config-manager --add-repo https://download.docker.com/linux/rhel/docker-ce.repo
sudo yum update -y
sudo yum install -y containerd
```

##### Ubuntu

```
sudo mkdir -p /etc/apt/keyrings
sudo curl -fsSL https://download.docker.com/linux/ubuntu/gpg -o /etc/apt/keyrings/docker.asc
echo \
 "deb [arch=$(dpkg --print-architecture) signed-by=/etc/apt/keyrings/docker.asc]
 https://download.docker.com/linux/ubuntu \
```



```
$(. /etc/os-release && echo "${UBUNTU_CODENAME:-$VERSION_CODENAME}") stable" | \
sudo tee /etc/apt/sources.list.d/docker.list > /dev/null
sudo apt update -y
sudo apt install -y --only-upgrade containerd.io
```

ステップ 2 c: **Containerd** CVE パッチがパッケージマネージャーに公開されていない場合の対応

パッチ適用済みの containerd バージョンがパッケージマネージャー以外の手段でのみ使用できる場合、例えば GitHub リリースのみであれば GitHub の公式サイトから containerd をインストールできます。

1. マシンがハイブリッドノードとしてクラスターに既に参加している場合は、`nodeadm uninstall` コマンドを実行する必要があります。
2. containerd の公式のバイナリをインストールします。GitHub の[公式のインストール手順](#)を使用できます。
3. `--containerd-source` 引数を `none` に設定して `nodeadm install` コマンドを実行します。nodeadm から containerd をインストールする操作がスキップされます。ノードでどのオペレーティングシステムが実行されていても、containerd ソースで `none` の値を使用できます。

```
nodeadm install K8S_VERSION --credential-provider CREDS_PROVIDER --containerd-source
none
```

## ハイブリッドノードを削除する

このトピックではアマゾン EKS クラスターからハイブリッドノードを削除する方法について説明します。[kubecti](#) などの Kubernetes 互換ツールを選択して、ハイブリッドノードを削除する必要があります。ハイブリッドノードの課金はノードオブジェクトがアマゾン EKS クラスターから削除されると停止します。ハイブリッドノードの料金の詳細については「[アマゾン EKS 料金表](#)」を参照してください。

### Important

ノードを削除すると、ノードで実行されているワークロードが中断されます。ハイブリッドノードを削除する前に、まずノードをドレインして、ポッドを別のアクティブなノードに移動することをお勧めします。ノードドレインのドレイン処理に関する詳細については Kubernetes ドキュメントの「[ノードの安全な排水](#)」を参照してください。



アマゾン EKS クラスターの Kubernetes API エンドポイントとのやり取りに使用するローカルマシンまたはインスタンスから、以下の `kubectl` ステップを実行してください。特定の `kubeconfig` ファイルを使用している場合は `--kubeconfig` フラグを使用します。

#### ステップ 1: ノードを一覧表示する

```
kubectl get nodes
```

#### ステップ 2: ノードをドレインする

`kubectl drain` コマンドの詳細については Kubernetes ドキュメントの「[kubectl drain](#)」を参照してください。

```
kubectl drain --ignore-daemonsets <node-name>
```

#### ステップ 3: ハイブリッドノードアーティファクトを停止およびアンインストールする

アマゾン EKS ハイブリッドノード CLI (`nodeadm`) を使用して、ホストからハイブリッドノードアーティファクトを停止および削除できます。root/sudo 権限を持つユーザーで `nodeadm` を実行する必要があります。デフォルトではノードにポッドが残っている場合、`nodeadm uninstall` は続行されません。AWS システム・マネージャー (SSM) を認証情報プロバイダーとして使用している場合、`nodeadm uninstall` コマンドはホストの AWS SSM マネージドインスタンスとしての登録を解除します。詳細については「[the section called “ハイブリッドノード用 nodeadm”](#)」を参照してください。

```
nodeadm uninstall
```

#### ステップ 4: クラスターからノードを削除する

ハイブリッドノードのアーティファクトを停止およびアンインストールしたら、クラスターからノードリソースを削除します。

```
kubectl delete node <node-name>
```

#### ステップ 5: 残りのアーティファクトを確認する

CNI の選択によっては上記のステップを実行した後にハイブリッドノードにアーティファクトが残っている場合があります。詳細については「[the section called “CNI を設定する”](#)」を参照してください。

## ハイブリッドノードの CNI、アドオン、ウェブフックを設定する

ハイブリッドノードでは、クラスターを設定した後に、コンテナネットワークインターフェイス (CNI)、アドオン、ウェブフック、および可能であればプロキシの設定を行う必要があります。ハイブリッドノードと互換性のある EKS およびコミュニティアドオンの完全なリストは、「[the section called “CNI アドオンの設定”](#)」を参照してください。

EKS クラスターインサイト EKS には、クラスターまたはワークロードの機能を損なう可能性のある EKS Hybrid Nodes の設定ミスに関するインサイトチェックが含まれています。クラスターインサイトの詳細については、「[the section called “クラスターのインサイト”](#)」を参照してください。

次に、ハイブリッドノードに対して異なる設定を持つアドオンやその他のコンポーネントを一覧表示します。

- Container Networking Interface (CNI): [Cilium](#) および [Calico](#) のコア機能は、ハイブリッドノードでの使用がサポートされています。CNI は、ハイブリッドノードで、Helm などの任意のツールを使って管理できます。AWS VPC CNI は、ハイブリッドノードでは使用できません。詳細については、「[the section called “CNI を設定する”](#)」を参照してください。
- CoreDNS と **kube-proxy**: ハイブリッドノードが EKS クラスターと結合したときは、CoreDNS と kube-proxy が自動的にインストールされます。これらのアドオンは、クラスターの作成後、EKS アドオンとして管理できます。
- Ingress とロードバランシング: AWS Direct Connect または AWS Site-to-Site VPN に接続されたハイブリッドノード上のワークロードには、ターゲットタイプ ip で AWS Load Balancer Controller と Application Load Balancer (ALB) または Network Load Balancer (NLB) を使用できます。または、選択した Ingress コントローラーまたはロードバランサーは、オンプレミス環境にローカルなままのアプリケーショントラフィックを行うのに使用できます。
- メトリクス: Amazon Managed Service for Prometheus (AMP) エージェントレススクレイパー、AWS Distro for Open Telemetry (ADOT)、Amazon CloudWatch Observability Agent は、ハイブリッドノードで使用できます。ポッドメトリクス用の AMP エージェントレススクレイパーをハイブリッドノードで使用する場合は、Amazon EKS クラスターに使用する VPC からポッドにアクセスする必要があります。
- ログ: ハイブリッドノードが有効になっているクラスターに対して、EKS コントロールプレーンのログ記録を有効にすることができます。ADOT EKS アドオンと Amazon CloudWatch Observability Agent EKS アドオンを、ハイブリッドノードとポッドのログ記録に使用できます。
- Pod Identity と IRSA: EKS Pod Identity と IAM Roles for Service Accounts (IRSA) を、ハイブリッドノードで実行中のアプリケーションで使用すると、ハイブリッドノードで実行中の Pod に、他の AWS サービスを使ってきめ細かくアクセスすることが可能になります。

- ウェブフック: ウェブフックを実行している場合は、オンプレミスのポッドネットワークをルーティングできない場合に、クラウドノードでウェブフックを選択的に実行する際の考慮事項と手順について「[the section called “ウェブフックの設定”](#)」を参照してください。
- Proxy: データセンターまたはエッジ環境から送信されるトラフィックに、オンプレミス環境にあるプロキシサーバーを使用している場合は、そのプロキシサーバーを使用するようにノードとクラスターを設定することができます。詳細については、「[the section called “プロキシを設定する”](#)」を参照してください。

## トピック

- [ハイブリッドノードの CNI を設定する](#)
- [ハイブリッドノードのアドオンを構成する](#)
- [ハイブリッドノード用のウェブフックを設定する](#)
- [ハイブリッドノードのプロキシを設定する](#)

## ハイブリッドノードの CNI を設定する

Cilium と Calico は Amazon EKS Hybrid Nodes のコンテナネットワークインターフェイス (CNI としてサポートされています。ハイブリッドノードがワークロードを処理できるようにするには CNI をインストールする必要があります。CNI が稼働するまではハイブリッドノードは Not Ready のステータスで表示されます。これらの CNI は Helm などの任意のツールで管理できます。Amazon VPC CNI はハイブリッドノードと互換性がなく、VPC CNI は `eks.amazonaws.com/compute-type: hybrid` ラベルに対しアンチアフィニティで設定されています。

### CNI のバージョン互換性

Cilium バージョン v1.16.x は、Amazon EKS でサポートされているすべての Kubernetes バージョンの EKS ハイブリッドノードでサポートおよび推奨されています。

Calico バージョン v3.29.x は、Amazon EKS でサポートされているすべての Kubernetes バージョンの EKS ハイブリッドノードでサポートされ、推奨されます。

Amazon EKS でサポートされている Kubernetes のバージョンについては、「[Kubernetes version support](#)」を参照してください。EKS Hybrid Nodes には、クラウドノードを使用する Amazon EKS クラスターと同じ Kubernetes バージョンのサポートがあります。

## サポートされる機能

AWS は、ハイブリッドノードで使用する Cilium および Calico の以下の機能についてテクニカルサポートを提供しています。AWS のサポート範囲外の機能を使用する予定がある場合は当該プラグインの商用サポートを受けるか、社内のエキスパートによるトラブルシューティングを行い、解決策を CNI プラグインプロジェクトに提供することをお勧めします。

| 機能                   | Cilium                  | Calico            |
|----------------------|-------------------------|-------------------|
| Kubernetes ネットワーク適合性 | はい                      | はい                |
| コントロールプレーンからノードへの接続性 | はい                      | はい                |
| コントロールプレーンからポッドへの接続性 | はい                      | はい                |
| ライフサイクル管理            | インストール、アップグレード、削除       | インストール、アップグレード、削除 |
| ネットワークモード            | VXLAN                   | VXLAN             |
| IP アドレス管理 (IPAM)     | クラスタースコープ (Cilium IPAM) | Calico IPAM       |
| IP ファミリー             | IPv4                    | IPv4              |
| BGP                  | はい (Cilium コントロールプレーン)  | はい                |

## Cilium に関する考慮事項

- デフォルトでは、Cilium は VXLAN を [カプセル化方法](#) とするオーバーレイ/トンネルモードで動作するように設定されています。このモードでは、基盤となる物理ネットワークに関する要件が最少になります。
- デフォルトでは、Cilium はクラスターから出ていくすべてのポッドトラフィックの送信元 IP アドレスをノードの IP アドレスに [マスカレード](#) します。そのため、クラスターにリモートポッドネットワークが設定されているかどうかにかかわらず、ハイブリッドノードで Cilium を実行でき

ます。マスカレードを無効にする場合は、オンプレミスネットワークでポッド CIDR をルーティングし、リモートポッドネットワークで Amazon EKS クラスターを設定する必要があります。

- ハイブリッドノードでウェブフックを実行している場合は、オンプレミスネットワークでポッド CIDR をルーティングし、リモートポッドネットワークで Amazon EKS クラスターを設定する必要があります。オンプレミスネットワークでポッド CIDR をルーティングできない場合は、同じクラスター内のクラウドノードでウェブフックを実行することが推奨されます。詳細については、「[Configure webhooks for hybrid nodes](#)」を参照してください。
- オンプレミスネットワークでポッド CIDR をルーティング可能にするために、BGP でポッドアドレスをアドバタイズするという方法がよくとられます。Cilium で BGP を使用するには、Helm の設定に `bgpControlPlane.enabled: true` を設定する必要があります。Cilium の BGP に対するサポート機能の詳細については「[織研新聞](#)」の「[Cilium BGP コントロールプレーン](#)」を参照してください。
- Cilium のデフォルトの IP Address Manager (IPAM) は [クラスタースコープ](#) と呼ばれ、Cilium オペレーターがユーザー設定のポッド CIDR に基づいてノードごとに IP アドレスを割り当てます。ポッド CIDR は `clusterPoolIPv4PodCIDRList` という Helm 値で設定されており、この値は Amazon EKS クラスターに対して設定したリモートポッドネットワーク CIDR と一致する必要があります。Cilium は、`clusterPoolIPv4PodCIDRList` から各ノードにセグメントを割り当てます。ノードセグメント 1 つあたりのサイズは、`clusterPoolIPv4MaskSize` という Helm 値で設定されます。`clusterPoolIPv4PodCIDRList` と `clusterPoolIPv4MaskSize` の詳細については、Cilium ドキュメントの「[Expanding the cluster pool](#)」を参照してください。

## ハイブリッドノードに Cilium をインストールする

1. コマンドライン環境に Helm CLI がインストールされていることを確認します。インストール手順については、[Helm の設定](#)を参照してください。
2. Cilium Helm リポジトリをインストールします。

```
helm repo add cilium https://helm.cilium.io/
```

3. `cilium-values.yaml` という名前の YAML ファイルを作成します。次の例では、`eks.amazonaws.com/compute-type: hybrid` ラベルにアフィニティを設定することでハイブリッドノードでのみ動作するように Cilium を設定しています。
  - リモートポッドネットワークで Amazon EKS クラスターを設定した場合は、`clusterPoolIPv4PodCIDRList` に同じポッド CIDR を設定してください。例えば、`10.100.0.0/24`。オーバーレイ/トンネルモードで CNi を実行するときは、オンプレミスポッド CIDR がオンプレミスノード CIDR と重ならないようにする必要があります。

- ノードあたりに必要なポッドに基づいて `clusterPoolIPv4MaskSize` を設定します。例えば、ノードあたり 128 個のポッドが必要で、そのサイズが /25 セグメントの場合は 25 とします。
- クラスターに Cilium をデプロイした後で、`clusterPoolIPv4PodCIDRList` または `clusterPoolIPv4MaskSize` を変更しないでください。詳細については、Cilium のドキュメントで「[Expanding the cluster pool](#)」を参照してください。
- Cilium の Helm 値の完全なリストについては、Cilium ドキュメントの「[Helm reference](#)」を参照してください。

```
affinity:
 nodeAffinity:
 requiredDuringSchedulingIgnoredDuringExecution:
 nodeSelectorTerms:
 - matchExpressions:
 - key: eks.amazonaws.com/compute-type
 operator: In
 values:
 - hybrid
ipam:
 mode: cluster-pool
 operator:
 clusterPoolIPv4MaskSize: 25
 clusterPoolIPv4PodCIDRList:
 - POD_CIDR
loadBalancer:
 serviceTopology: true
operator:
 affinity:
 nodeAffinity:
 requiredDuringSchedulingIgnoredDuringExecution:
 nodeSelectorTerms:
 - matchExpressions:
 - key: eks.amazonaws.com/compute-type
 operator: In
 values:
 - hybrid
 unmanagedPodWatcher:
 restart: false
envoy:
 enabled: false
```

#### 4. クラスターに Cilium をインストールします。

- CILIUM\_VERSION を Cilium バージョン (v1.17.0 など) に置き換えます。Cilium のマイナーバージョンに対して、最新のパッチバージョンを実行することをお勧めします。特定の Cilium マイナーリリースの最新のパッチリリースはCilium ドキュメントの「[安定リリースのセクション](#)」で確認できます。
- デプロイで BGP を有効にする場合は以下のコマンドに `--set bgpControlPlane.enabled=true` フラグを追加します。
- 特定の kubeconfig ファイルを使用している場合はHelm install コマンドで `--kubeconfig` フラグを使用します。

```
helm install cilium cilium/cilium \
 --version CILIUM_VERSION \
 --namespace kube-system \
 --values cilium-values.yaml
```

5. 以下のコマンドを使用して、Cilium のインストールが成功したことを確認できます。cilium-operator デプロイと、各ハイブリッドノードで実行されている cilium-agent が確認できるはずです。また、ハイブリッドノードのステータスは Ready になっているはずです。Cilium 用に BGP を設定する方法については次のステップに進んでください。

```
kubectl get pods -n kube-system
```

| NAME                            | READY | STATUS  | RESTARTS | AGE |
|---------------------------------|-------|---------|----------|-----|
| cilium-jjjn8                    | 1/1   | Running | 0        | 11m |
| cilium-operator-d4f4d7fcb-sc5xn | 1/1   | Running | 0        | 11m |

```
kubectl get nodes
```

| NAME                 | STATUS | ROLES  | AGE | VERSION             |
|----------------------|--------|--------|-----|---------------------|
| mi-04a2cf999b7112233 | Ready  | <none> | 19m | v1.31.0-eks-a737599 |

6. Cilium で BGP を使用してオンプレミスネットワークでポッドアドレスをアドバタイズするには `bgpControlPlane.enabled: true` を使用して Cilium をインストールする必要があります。既存の Cilium のデプロイで BGP を有効にする場合は、Cilium 演算子を再起動して BGP 設定を適用する必要があります (BGP が以前に有効になっていない場合)。Helm のインストール/アップグレードプロセスの一環として Cilium 演算子を再起動するには、Helm 値の `operator.rollOutPods` を `true` に設定します。



Cilium で BGP を設定するには、まず CiliumBGPClusterConfig 定義を使用して cilium-bgp-cluster.yaml という名前のファイルを作成します。ネットワーク管理者から次の情報を取得する必要がある場合があります。

- Cilium を実行しているノードの ASN を使用して、localASN を設定します。
- オンプレミスルーターの ASN を使用して、peerASN を設定します。
- Cilium を実行している各ノードがピアリングするオンプレミスルーター IP を使用して、peerAddress を設定します。

```
apiVersion: cilium.io/v2alpha1
kind: CiliumBGPClusterConfig
metadata:
 name: cilium-bgp
spec:
 nodeSelector:
 matchExpressions:
 - key: eks.amazonaws.com/compute-type
 operator: In
 values:
 - hybrid
 bgpInstances:
 - name: "rack0"
 localASN: NODES_ASN
 peers:
 - name: "onprem-router"
 peerASN: ONPREM_ROUTER_ASN
 peerAddress: ONPREM_ROUTER_IP
 peerConfigRef:
 name: "cilium-peer"
```

7. Cilium BGP クラスター設定をクラスターに適用します。

```
kubectl apply -f cilium-bgp-cluster.yaml
```

8. CiliumBGPPeerConfig リソースは BGP ピア設定を定義します。複数のピアが同じ設定を共有し、共通の CiliumBGPPeerConfig リソースを参照することができます。オンプレミスネットワークのピア構成を設定するために、cilium-bgp-peer.yaml という名前のファイルを作成します。設定オプションの完全なリストについてはCilium ドキュメントの「[BGP ピア設定](#)」を参照してください。



```
apiVersion: cilium.io/v2alpha1
kind: CiliumBGPPeerConfig
metadata:
 name: cilium-peer
spec:
 timers:
 holdTimeSeconds: 30
 keepAliveTimeSeconds: 10
 gracefulRestart:
 enabled: true
 restartTimeSeconds: 120
 families:
 - afi: ipv4
 safi: unicast
 advertisements:
 matchLabels:
 advertise: "bgp"
```

## 9. Cilium BGP ピア設定をクラスターに適用します。

```
kubectl apply -f cilium-bgp-peer.yaml
```

10. CiliumBGPAdvertisement リソースはそれらに関連付けられたさまざまなアドバタイズタイプと属性を定義するために使用されます。cilium-bgp-advertisement.yaml という名前のファイルを作成し、CiliumBGPAdvertisement リソースに必要な設定を行います。

```
apiVersion: cilium.io/v2alpha1
kind: CiliumBGPAdvertisement
metadata:
 name: bgp-advertisements
 labels:
 advertise: bgp
spec:
 advertisements:
 - advertisementType: "PodCIDR"
 - advertisementType: "Service"
 service:
 addresses:
 - ClusterIP
 - ExternalIP
 - LoadBalancerIP
```

## 11. Cilium BGP アドバタイズ設定をクラスターに適用します。

```
kubectl apply -f cilium-bgp-advertisement.yaml
```

`cilium bgp peers` コマンドを使用して、BGP ピアリングが [Cilium CLI](#) で機能していることを確認できます。出力には環境に応じた正しい値が表示され、セッションの状態が `established` と表示されているはずです。トラブルシューティングの詳細については「Cilium 資料」の「[トラブルシューティングおよび操作ガイド](#)」を参照してください。

### ハイブリッドノードで Cilium をアップグレードする

Cilium デプロイをアップグレードする前に、[Cilium アップグレードのドキュメント](#)とアップグレードに関する注意事項をよく確認し、対象となる Cilium バージョンの変更について理解してください。

1. コマンドライン環境に `helm` CLI がインストールされていることを確認します。インストール手順については「[Helm のドキュメント](#)」を参照してください。
2. Cilium Helm リポジトリをインストールします。

```
helm repo add cilium https://helm.cilium.io/
```

3. Cilium アップグレードのプリフライトチェックを実行してください。CILIUM\_VERSION を対象となる Cilium バージョンに置き換えます。Cilium のマイナーバージョンに対して、最新のパッチバージョンを実行することをお勧めします。特定の Cilium マイナーリリースの最新のパッチリリースはCilium ドキュメントの「[安定リリースのセクション](#)」で確認できます。

```
helm install cilium-preflight cilium/cilium --version CILIUM_VERSION \
 --namespace=kube-system \
 --set preflight.enabled=true \
 --set agent=false \
 --set operator.enabled=false
```

4. `cilium-preflight.yaml` の適用後は READY Pod の数が実行中の Cilium Pod の数と同じであることを確認します。

```
kubectl get ds -n kube-system | sed -n '1p;/cilium/p'
```

| NAME                    | DESIRED | CURRENT | READY | UP-TO-DATE | AVAILABLE | NODE   |
|-------------------------|---------|---------|-------|------------|-----------|--------|
| SELECTOR                |         |         |       |            |           |        |
| AGE                     |         |         |       |            |           |        |
| cilium                  | 2       | 2       | 2     | 2          | 2         | <none> |
| 1h20m                   |         |         |       |            |           |        |
| cilium-pre-flight-check | 2       | 2       | 2     | 2          | 2         | <none> |
| 7m15s                   |         |         |       |            |           |        |

5. READY ポッドの数が同じになったら、Cilium プリフライトデプロイも READY 1/1 とマークされていることを確認してください。READY 0/1 と表示された場合はアップグレードを続行する前に、「[CNP の検証](#)」セクションを参照しながらデプロイの問題を解決してください。

```
kubectl get deployment -n kube-system cilium-pre-flight-check -w
```

| NAME                    | READY | UP-TO-DATE | AVAILABLE | AGE |
|-------------------------|-------|------------|-----------|-----|
| cilium-pre-flight-check | 1/1   | 1          | 0         | 12s |

## 6. プリフライトを削除する

```
helm uninstall cilium-preflight --namespace kube-system
```

7. 通常のクラスターオペレーションではすべての Cilium コンポーネントが同じバージョンを実行している必要があります。以下のステップではある安定版リリースからより新しい安定版リリースへと、すべてのコンポーネントをアップグレードする方法について説明します。あるマイナーリリースから別のマイナーリリースにアップグレードする場合はまず既存の Cilium マイナーバージョンの最新のパッチリリースにアップグレードすることが推奨されます。中断を最小限に抑えるには upgradeCompatibility オプションを、このクラスターに最初にインストールされた Cilium のバージョンに設定します。

helm upgrade コマンドを実行する前に、デプロイの値を cilium-values.yaml に保存するか、設定に対し --set コマンドラインオプションを使用してください。アップグレード操作は Cilium ConfigMap を上書きするため、アップグレード時に設定値を渡すことが極めて重要です。BGP を使用している場合は値ファイルにこの情報を入力するのではなく、--set bgpControlPlane=true コマンドラインオプションを使用することが推奨されます。

```
helm upgrade cilium cilium/cilium --version CILIUM_VERSION \
 --namespace kube-system \
 --set upgradeCompatibility=1.X \
 -f cilium-values.yaml
```

8. (オプション) 問題があってアップグレードをロールバックする必要がある場合は次のコマンドを実行してください。

```
helm history cilium --namespace kube-system
helm rollback cilium [REVISION] --namespace kube-system
```

## ハイブリッドノードから Cilium を削除する

1. 以下のコマンドを実行して、クラスターからすべての Cilium コンポーネントをアンインストールします。なお、CNI をアンインストールすると、ノードと Pod の正常性に影響する可能性があるため、本番稼働用クラスターでは実行しないでください。

```
helm uninstall cilium --namespace kube-system
```

CNI がクラスターから削除されても、Cilium によって設定されたインターフェイスとルートはデフォルトでは削除されません。詳細については「[GitHub の問題](#)」を参照してください。

2. ディスク上の設定ファイルとリソースをクリーンアップするには標準的な設定ディレクトリを使用している場合は GitHub の Cilium リポジトリの [cni-uninstall.sh スクリプト](#) に示されているようにファイルを削除できます。
3. Cilium カスタムリソース定義 (CRD クラスターから削除するには以下のコマンドを実行してください)。

```
kubectl get crds -oname | grep "cilium" | xargs kubectl delete
```

## Calico の考慮事項

- VXLAN を [カプセル化方法](#) とするオーバーレイ/トンネルモードで Calico を実行することが推奨されます。このモードでは、基盤となる物理ネットワークに関する要件が最も少なくなります。Calico のさまざまなネットワークモードの詳細については「Calico 資料」の「[最適なネットワークオプションの決定](#)」を参照してください。
- natOutgoing を true に設定して Calico を実行することが推奨されます。natOutgoing を true に設定すると、クラスターから出ていくすべてのポッドトラフィックの送信元 IP アドレスがノードの IP アドレスに変換されます。そのため、クラスターにリモートポッドネットワークが設定されているかどうかにかかわらず、Amazon EKS クラスターで Calico を実行できま

す。natOutgoing を無効にする場合は、オンプレミスネットワークでポッド CIDR をルーティングし、リモートポッドネットワークで Amazon EKS クラスターを設定する必要があります。

- ハイブリッドノードでウェブフックを実行している場合は、オンプレミスネットワークでポッド CIDR をルーティングし、リモートポッドネットワークで Amazon EKS クラスターを設定する必要があります。オンプレミスネットワークでポッド CIDR をルーティングできない場合は、同じクラスター内のクラウドノードでウェブフックを実行することが推奨されます。詳細については、「[Configure webhooks for hybrid nodes](#)」を参照してください。
- オンプレミスネットワークでポッド CIDR をルーティング可能にするために、BGP でポッドアドレスをアドバタイズするという方法がよくとられます。Calico で BGP を使用するには、Helm の設定に `installation.calicoNetwork.bgp: Enabled` を設定する必要があります。Calico の BGP サポートの詳細については、Calico のドキュメントで「[Configure BGP peering](#)」を参照してください。
- Calico のデフォルトの IP Address Manager (IPAM) は [Calico IPAM](#) と呼ばれ、calico-ipam プラグインがユーザー設定のポッド CIDR に基づいてノードごとに IP アドレスを割り当てます。ポッド CIDR は `installation.calicoNetwork.ipPools.cidr` という Helm 値で設定されており、この値は Amazon EKS クラスターに対して設定したリモートポッドネットワーク CIDR と一致する必要があります。Calico は、`ipPools.cidr` から各ノードにセグメントを割り当てます。ノードセグメント 1 つあたりのサイズは、`ipPools.blockSize` という Helm 値で設定されます。Calico での IPAM の詳細については、Calico のドキュメントで「[Get started with IP address management](#)」を参照してください。

## ハイブリッドノードに Calico をインストールする

1. コマンドライン環境に helm CLI がインストールされていることを確認します。インストール手順については「[Helm のドキュメント](#)」を参照してください。
2. Cilium Helm リポジトリをインストールします。

```
helm repo add projectcalico https://docs.tigera.io/calico/charts
```

3. `calico-values.yaml` という名前の YAML ファイルを作成します。次の例では、`eks.amazonaws.com/compute-type: hybrid` ラベルにアフィニティを設定することでハイブリッドノードでのみ動作するようにすべての Calico コンポーネントを設定しています。
  - `POD_CIDR` をポッドの CIDR 範囲に置き換えます。Amazon EKS クラスターをリモートポッドネットワークで設定した場合、Calico に指定する `POD_CIDR` はリモートポッドネットワークと同じである必要があります。例えば、`10.100.0.0/24`。オーバーレイ/トンネルモードで CNI

を実行するときは、オンプレミスポッド CIDR がオンプレミスノード CIDR と重ならないようにする必要があります。

- CIDR\_SIZE を、各ノードに割り当てる CIDR セグメントのサイズに置き換えます。例えば、ノードあたり 128 個のポッドアドレスが必要で、そのサイズが /25 セグメントの場合は 25 とします。CIDR blockSize および blockSize の変更の詳細については「Calico 資料」の「[IP プールのブロックサイズを変更する](#)」を参照してください。
- 以下の例では natOutgoing は有効で、bgp は無効になっています。これらの値をターゲット設定に基づいて変更します。

```
installation:
 enabled: true
 cni:
 type: Calico
 ipam:
 type: Calico
 calicoNetwork:
 bgp: Disabled
 ipPools:
 - cidr: POD_CIDR
 blockSize: CIDR_SIZE
 encapsulation: VXLAN
 natOutgoing: Enabled
 nodeSelector: eks.amazonaws.com/compute-type == "hybrid"
 controlPlaneReplicas: 1
 controlPlaneNodeSelector:
 eks.amazonaws.com/compute-type: hybrid
 calicoNodeDaemonSet:
 spec:
 template:
 spec:
 nodeSelector:
 eks.amazonaws.com/compute-type: hybrid
 csiNodeDriverDaemonSet:
 spec:
 template:
 spec:
 nodeSelector:
 eks.amazonaws.com/compute-type: hybrid
 calicoKubeControllersDeployment:
 spec:
 template:
 spec:
```

```

 nodeSelector:
 eks.amazonaws.com/compute-type: hybrid
 typhaDeployment:
 spec:
 template:
 spec:
 nodeSelector:
 eks.amazonaws.com/compute-type: hybrid

```

#### 4. クラスターに Calico をインストールします。

- CALICO\_VERSION を Calico バージョン (v3.29.0 など) に置き換えます。Calico マイナーバージョンの最新のパッチリリースについては「[Calico releases](#)」を参照してください。Cilium のマイナーバージョンには最新のパッチバージョンを実行することが推奨されます。
- 特定の kubeconfig ファイルを使用している場合は --kubeconfig フラグを使用します。

```

helm install calico projectcalico/tigera-operator \
 --version CALICO_VERSION \
 --namespace kube-system \
 -f calico-values.yaml

```

5. 以下のコマンドを使用すると、Calico のインストールが正常に終了したことを確認できます。tigera-operator デプロイ、各ハイブリッドノードで実行されている calico-node エージェント、およびデプロイされた calico-apiserver、csi-node-driver、calico-kube-controllers が確認できるはずです。また、ハイブリッドノードのステータスは Ready になっているはずです。natOutgoing: Disabled を使用している場合、オンプレミスネットワークでポッドアドレスをアドバタイズするまで、すべての Calico コンポーネントを正常に起動することはできません。Calico 用に BGP を設定する方法については次のステップに進んでください。

```
kubectl get pods -A
```

| NAMESPACE        | NAME                                     | READY | STATUS  |
|------------------|------------------------------------------|-------|---------|
| calico-apiserver | calico-apiserver-6c77bb6d46-2n8mq        | 1/1   | Running |
| calico-system    | calico-kube-controllers-7c5f8556b5-7h267 | 1/1   | Running |
| calico-system    | calico-node-s5nnk                        | 1/1   | Running |
| calico-system    | calico-typha-6487cc9d8c-wc5jm            | 1/1   | Running |

|                      |                                 |     |         |   |
|----------------------|---------------------------------|-----|---------|---|
| calico-system<br>68s | csi-node-driver-cv42d           | 2/2 | Running | 0 |
| kube-system<br>6m27s | coredns-7bb495d866-2lc9v        | 1/1 | Running | 0 |
| kube-system<br>157m  | coredns-7bb495d866-2t8ln        | 1/1 | Running | 0 |
| kube-system<br>18m   | kube-proxy-lxzxh                | 1/1 | Running | 0 |
| kube-system<br>90s   | tigera-operator-f8bc97d4c-28b4d | 1/1 | Running | 0 |

```
kubectl get nodes
```

```
NAME STATUS ROLES AGE VERSION
mi-0c6ec2f6f79176565 Ready <none> 5h13m v1.31.0-eks-a737599
```

6. BGP なしで Calico をインストールした場合はこのステップをスキップしてください。BGP を設定するには BGPPeer 設定と BGPConfiguration を含む calico-bgp.yaml というファイルを作成してください。BGPPeer と BGPConfiguration を区別することが重要です。BGPPeer は Calico クラスター内のノードがピアリング接続する BGP 対応ルーターまたはリモートリソースです。BGPPeer 設定の asNumber は Cilium の設定 peerASN と類似しています。BGPConfiguration は各 Calico ノードに適用され、BGPConfiguration の asNumber は Cilium 設定 localASN と同等です。以下の例では ONPREM\_ROUTER\_IP、ONPREM\_ROUTER\_ASN、LOCAL\_ASN をオンプレミス環境の値に置き換えています。場合によってはネットワーク管理者から入手する必要があります。keepOriginalNextHop: true 設定は各ノードがそれ自体が所有するポッドネットワーク CIDR のみをアドバタイズすることを保証するために使用されます。

```
apiVersion: projectcalico.org/v3
kind: BGPPeer
metadata:
 name: calico-hybrid-nodes
spec:
 peerIP: ONPREM_ROUTER_IP
 asNumber: ONPREM_ROUTER_ASN
 keepOriginalNextHop: true

apiVersion: projectcalico.org/v3
kind: BGPConfiguration
metadata:
```



```
name: default
spec:
 nodeToNodeMeshEnabled: false
 asNumber: LOCAL_ASN
```

7. ファイルをクラスターに適用します。

```
kubectl apply -f calico-bgp.yaml
```

8. 以下のコマンドを使用して、Calico ポッドが実行されていることを確認します。

```
kubectl get pods -n calico-system -w
```

| NAMESPACE        | NAME                                     | READY | STATUS  |    |
|------------------|------------------------------------------|-------|---------|----|
| RESTARTS         | AGE                                      |       |         |    |
| calico-apiserver | calico-apiserver-598bf99b6c-2vltk        | 1/1   | Running | 0  |
|                  | 3h24m                                    |       |         |    |
| calico-system    | calico-kube-controllers-75f84bbfd6-zwmnx | 1/1   | Running | 31 |
|                  | (59m ago) 3h20m                          |       |         |    |
| calico-system    | calico-node-9b2pg                        | 1/1   | Running | 0  |
|                  | 5h17m                                    |       |         |    |
| calico-system    | calico-typha-7d55c76584-kxtnq            | 1/1   | Running | 0  |
|                  | 5h18m                                    |       |         |    |
| calico-system    | csi-node-driver-dmnmm                    | 2/2   | Running | 0  |
|                  | 5h18m                                    |       |         |    |
| kube-system      | coredns-7bb495d866-dtn4z                 | 1/1   | Running | 0  |
|                  | 6h23m                                    |       |         |    |
| kube-system      | coredns-7bb495d866-mk7j4                 | 1/1   | Running | 0  |
|                  | 6h19m                                    |       |         |    |
| kube-system      | kube-proxy-vms28                         | 1/1   | Running | 0  |
|                  | 6h12m                                    |       |         |    |
| kube-system      | tigera-operator-55f9d9d565-jj9bg         | 1/1   | Running | 0  |
|                  | 73m                                      |       |         |    |

これらのステップ中に問題が発生した場合はCalico ドキュメントの「[トラブルシューティングガイド](#)」を参照してください。

## ハイブリッドノードで Calico をアップグレードする

Calico デプロイをアップグレードする前に、[Calico アップグレードドキュメント](#)と[リリースノート](#)をよく確認し、対象となる Calico バージョンの変更について理解してください。アップグレードステップはHelm、Calico 演算子、およびデータストアのタイプによって異なります。以下のステップではHelm の使用を想定しています。

1. アップグレードする Calico のバージョン用のオペレータマニフェストをダウンロードします。CALICO\_VERSION をアップグレード先のバージョン、たとえば v3.29.0 に置き換えます。major.minor.patch の前には必ず v を付けてください。

```
kubectl apply --server-side --force-conflicts \
 -f https://raw.githubusercontent.com/projectcalico/calico/CALICO_VERSION/
 manifests/operator-crds.yaml
```

2. helm upgrade を実行して Calico デプロイをアップグレードします。CALICO\_VERSION をアップグレード先のバージョン、たとえば v3.29.0 に置き換えます。Calico のインストールに使用した設定値から calico-values.yaml ファイルを作成します。

```
helm upgrade calico projectcalico/tigera-operator \
 --version CALICO_VERSION \
 --namespace kube-system \
 -f calico-values.yaml
```

## ハイブリッドノードから Calico を削除する

1. 次のコマンドを実行して、Calico コンポーネントをクラスターからアンインストールします。なお、CNI をアンインストールすると、ノードと Pod の正常性に影響する可能性があるため、本番稼働用クラスターでは実行しないでください。Calico を kube-system 以外の名前空間にインストールした場合、以下のコマンドで名前空間を変更します。

```
helm uninstall calico --namespace kube-system
```

Calico によって設定されたインターフェイスとルートは CNI がクラスターから削除されても、デフォルトでは削除されないことに注意してください。

2. ディスク上の設定ファイルとリソースをクリーンアップするには/opt/cni および /etc/cni ディレクトリから Calico ファイルを削除します。
3. Calico CRD をクラスターから削除するには以下のコマンドを実行してください。

```
kubectl get crds -oname | grep "calico" | xargs kubectl delete
```

```
kubectl get crds -oname | grep "tigera" | xargs kubectl delete
```

## ハイブリッドノードのアドオンを構成する

このページでは、Amazon EKS Hybrid Nodes で AWS アドオンとコミュニティアドオンを実行する際の考慮事項について説明します。Amazon EKS アドオンと、クラスターからアドオンを作成、アップグレード、削除するプロセスの詳細については、「[the section called “Amazon EKS アドオン”](#)」を参照してください。このページに特に明記されていない限り、Amazon EKS アドオンを作成、アップグレード、削除するプロセスは、ハイブリッドノードを備える Amazon EKS クラスターでも、AWS クラウドでノードが実行される Amazon EKS クラスターでも同じです。このページに記載のアドオンについてのみ、Amazon EKS Hybrid Nodes との互換性が検証されています。

以下の AWS アドオンは、Amazon EKS Hybrid Nodes と互換性があります。

| AWS アドオン                                 | 互換性のあるアドオンバージョン                                                                                                                       |
|------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------|
| kube-proxy                               | v1.25.14-eksbuild.2 以降                                                                                                                |
| CoreDNS                                  | v1.9.3-eksbuild.7 以降                                                                                                                  |
| AWS オープンテレメトリー用ディストリビューター (ADOT)         | v0.102.1-eksbuild.2 以降                                                                                                                |
| CloudWatch Observability エージェント          | v2.2.1-eksbuild.1 以降                                                                                                                  |
| EKS Pod Identity エージェント                  | <ul style="list-style-type: none"> <li>v1.3.3-eksbuild.1 以降 (Bottlerocket 以外)</li> <li>v1.3.7-eksbuild.2 以降 (Bottlerocket)</li> </ul> |
| ノードモニタリングエージェント                          | v1.2.0-eksbuild.1 以降                                                                                                                  |
| CSI スナップショットコントローラー                      | v8.1.0-eksbuild.1 以降                                                                                                                  |
| AWS プライベート CA の Connector for Kubernetes | v1.6.0-eksbuild.1 以降                                                                                                                  |

以下のコミュニティアドオンは、Amazon EKS Hybrid Nodes と互換性があります。コミュニティアドオンの詳細については、「[the section called “コミュニティアドオン”](#)」を参照してください。

| コミュニティアドオン               | 互換性のあるアドオンバージョン       |
|--------------------------|-----------------------|
| Kubernetes メトリクスサーバー     | v0.7.2-eksbuild.1 以降  |
| cert-manager             | v1.17.2-eksbuild.1 以降 |
| Prometheus Node Exporter | v1.9.1-eksbuild.2 以降  |
| kube-state-metrics       | v2.15.0-eksbuild.4 以降 |

上記の表の Amazon EKS アドオンに加えて、[Amazon Managed Service for Prometheus コレクター](#)、および[アプリケーションイングレス](#) (HTTP) および[ロードバランシング](#) (TCP/UDP) 用の [AWS ロードバランサーコントローラー](#) はハイブリッドノードと互換性があります。

AWS アドオンとコミュニティアドオンには、Amazon EKS Hybrid Nodes と互換性のないものがあります。こうしたアドオンの最新バージョンには、デフォルトの `eks.amazonaws.com/compute-type: hybrid` ラベルをハイブリッドノードに適用するにあたってアンチアフィニティルールがあります。これにより、クラスターにデプロイされたときにハイブリッドノードで実行されるのが防止されます。ハイブリッドノードと AWS クラウドで実行されているノードの両方を備えたクラスターがある場合、クラスター内のこれらのアドオンを AWS クラウドで実行されているノードにデプロイできます。Amazon VPC CNI はハイブリッドノードと互換性がなく、Cilium と Calico は Amazon EKS Hybrid Nodes のコンテナネットワークインターフェイス (CNI としてサポートされています。詳細については「[the section called “CNI を設定する”](#)」を参照してください。

## AWS アドオン

以降のセクションでは、ハイブリッドノード上の互換性のある AWS アドオンを実行する場合と、他の Amazon EKS コンピューティングタイプで同じアドオンを実行する場合の違いについて説明します。

### kube-proxy と CoreDNS

AWS CLI によるものも含め AWS API と AWS SDK で EKS クラスターを作成すると、EKS はデフォルトで kube-proxy と CoreDNS をセルフマネージドアドオンとしてインストールします。こうしたアドオンは、クラスターの作成後に Amazon EKS アドオンで上書きできます。[the section called “kube-proxy”](#) および [the section called “CoreDNS”](#) の詳細については EKS ドキュメントを参

照してください。ハイブリッドノードと AWS クラウド内のノードの両方を備えた混合モードのクラスターを実行している場合は、ハイブリッドノードに少なくとも 1 つの CoreDNS レプリカがあり、AWS クラウド内のノードに少なくとも 1 つの CoreDNS レプリカがあるようにすることをお勧めします。設定手順については、「[the section called “CoreDNS レプリカを設定する”](#)」を参照してください。

## CloudWatch Observability エージェント

CloudWatch Observability エージェントオペレーターは、[ウェブフック](#)を使用します。ハイブリッドノードでオペレーターを実行する場合は、オンプレミスネットワークでオンプレミスポッド CIDR をルーティングし、リモートポッドネットワークで EKS クラスターを設定する必要があります。詳細については、「[Configure webhooks for hybrid nodes](#)」を参照してください。

ノードレベルのメトリクスは[クラウドコンテナインサイトを見る](#)がノードレベルのメトリクスに関して[インスタンスメタデータサービス](#) (IMDS の可用性に依存するため、ハイブリッドノードでは利用できません。クラスター、ワークロード、ポッド、コンテナレベルのメトリクスはハイブリッドノードで利用できます。

「[Amazon CloudWatch オペサビリティ を使用して Amazon CloudWatch エージェントをインストールする](#)」で説明されているステップに従ってアドオンをインストールした後、エージェントがハイブリッドノードで正常に実行できるようにするにはアドオンマニフェストを更新する必要があります。以下に示すように、クラスターの amazoncloudwatchagents リソースを編集して RUN\_WITH\_IRSA 環境変数を追加します。

```
kubectl edit amazoncloudwatchagents -n amazon-cloudwatch cloudwatch-agent
```

```
apiVersion: v1
items:
- apiVersion: cloudwatch.aws.amazon.com/v1alpha1
 kind: AmazonCloudWatchAgent
 metadata:
 ...
 name: cloudwatch-agent
 namespace: amazon-cloudwatch
 ...
 spec:
 ...
 env:
 - name: RUN_WITH_IRSA # <-- Add this
 value: "True" # <-- Add this
```

```
- name: K8S_NODE_NAME
 valueFrom:
 fieldRef:
 fieldPath: spec.nodeName
 ...
```

## Amazon Managed Prometheus ハイブリッドノード用マネージドコレクター

Amazon Managed Service for Prometheus (AMP) マネージドコレクターは Amazon EKS クラスター内のリソースからメトリクスを検出して収集するスクレイパーで構成されています。AMP はスクレイパーを自動的に管理するため、インスタンス、エージェント、スクレイパーをユーザーが管理する必要はありません。

AMP マネージドコレクターはハイブリッドノードに固有の追加設定を行わなくても使用できます。ただし、VPC からリモートポッドネットワーク CIDR へのルートや、オンプレミスファイアウォールで開いているポートなど、ハイブリッドノード上のアプリケーションのメトリクスエンドポイントが VPC から到達可能である必要があります。さらに、クラスターには[プライベートクラスターエンドポイントアクセス](#)が必要です。

「Amazon Managed Service for Prometheus ユーザーガイド」の「[AWS マネージドコレクターの使用](#)」のステップに従います。

## AWS オープンテレメトリー用ディストリビューター (ADOT)

AWS Distro for OpenTelemetry (ADOT) アドオンを使用して、ハイブリッドノードで実行されているアプリケーションからメトリクス、ログ、トレースデータを収集できます。ADOT は、Collector Custom Resource リクエストの変更と検証にアドミッション[ウェブフック](#)を使用します。ハイブリッドノードで ADOT オペレーターを実行する場合は、オンプレミスネットワークでオンプレミスポッド CIDR をルーティングし、リモートポッドネットワークで EKS クラスターを設定する必要があります。詳細については、「[Configure webhooks for hybrid nodes](#)」を参照してください。

AWS オープンテレメトリー用ディストリビューター ドキュメントの「[EKS アドオンを使用した AWS オープンテレメトリー用ディストリビューターの開始方法](#)」のステップに従ってください。

## AWS ロードバランサーコントローラー

AWS Direct Connect または AWS Site-to-Site VPN で接続されたハイブリッドノード上のワークロードに対し、[AWS ロードバランサーコントローラー](#) と Application Load Balancer (ALB または Network Load Balancer (NLB ターゲットタイプ ip) で使用できます。ALB または NLB で使用される IP ターゲットは、AWS からルーティング可能である必要があります。AWS Load Balancer

Controller も、[ウェブフック](#)を使用します。ハイブリッドノードで AWS Load Balancer Controller オペレーターを実行する場合は、オンプレミスネットワークでオンプレミスポッド CIDR をルーティングし、リモートポッドネットワークで EKS クラスターを設定する必要があります。詳細については、「[Configure webhooks for hybrid nodes](#)」を参照してください。

AWS ロードバランサーコントローラー をインストールするには「[the section called “Helm を使用してインストールする”](#)」または「[the section called “マニフェストを使用して インストールする”](#)」のステップに従います。

ALB でインGRESSを使用する場合は以下の注釈を指定する必要があります。手順については「[the section called “アプリケーション負荷分散”](#)」を参照してください。

```
alb.ingress.kubernetes.io/target-type: ip
```

NLB で負荷分散を使用する場合は以下の注釈を指定する必要があります。手順については「[the section called “ネットワーク負荷分散”](#)」を参照してください。

```
service.beta.kubernetes.io/aws-load-balancer-type: "external"
service.beta.kubernetes.io/aws-load-balancer-nlb-target-type: "ip"
```

## EKS Pod Identity エージェント

### Note

Bottlerocket を実行しているハイブリッドノードに EKS Pod Identity Agent アドオンを正常にデプロイするには、Bottlerocket のバージョンが v1.39.0 以上であることを確認します。Pod Identity エージェントのハイブリッドノード環境では、以前の Bottlerocket バージョンがサポートされていません。

元の Amazon EKS Pod Identity エージェント DaemonSet は必要な AWS 認証情報を取得するために、ノードの EC2 IMDS を利用します。IMDS はハイブリッドノードでは使用できないため、バージョン 1.3.3-eksbuild.1 以降では必要に応じて Pod Identity Agent アドオンに必要な認証情報をマウントする DaemonSet がデプロイされます。Bottlerocket を実行するハイブリッドノードでは、認証情報をマウントするために別の方法が必要です。バージョン 1.3.7-eksbuild.2 以降では、必要に応じて Pod Identity Agent アドオンに Bottlerocket ハイブリッドノードを対象とする専用の DaemonSet がデプロイされます。以下のセクションでは、オプションの DaemonSet を有効にするプロセスについて説明します。

## Ubuntu/RHEL/AL2023

1. Ubuntu/RHEL/AL2023 ハイブリッドノードで Pod Identity エージェントを使用するには、以下に示すように nodeadm 設定のハイブリッドセクションで `enableCredentialsFile: true` を設定します。

```
apiVersion: node.eks.aws/v1alpha1
kind: NodeConfig
spec:
 hybrid:
 enableCredentialsFile: true # <-- Add this
```

これにより、eks-pod-identity-agent ポッドで使用される認証情報ファイルをノード上の /eks-hybrid/.aws/credentials に作成するように nodeadm が設定されます。この認証情報ファイルには定期的に更新される一時的な AWS 認証情報が含まれます。

2. 各ノードの nodeadm 設定を更新したら、nodeConfig.yaml で以下の nodeadm init コマンドを実行して、ハイブリッドノードを Amazon EKS クラスターに結合します。ノードが以前にクラスターに参加していた場合でも、init コマンドを再度実行してください。

```
nodeadm init -c file:///nodeConfig.yaml
```

3. AWS CLI または AWS Management Console のいずれかを使用し、ハイブリッドノードのサポートを有効にして eks-pod-identity-agent をインストールします。
  - a. AWS CLI: クラスターの管理に使用しているマシンから以下のコマンドを実行して、ハイブリッドノードのサポートを有効にして eks-pod-identity-agent をインストールします。my-cluster を自分のクラスター名に置き換えます。

```
aws eks create-addon \
 --cluster-name my-cluster \
 --addon-name eks-pod-identity-agent \
 --configuration-values '{"daemonsets":{"hybrid":{"create": true}}}'
```

- b. AWS Management Console: AWS コンソールから Pod Identity Agent アドオンをインストールする場合は、オプションの設定に以下を追加して、ハイブリッドノードを対象とする DaemonSet をデプロイします。

```
{"daemonsets":{"hybrid":{"create": true}}}
```



## Bottlerocket

1. Bottlerocket ハイブリッドノードで Pod Identity エージェントを使用するには、「[the section called “Bottlerocket を実行しているハイブリッドノードの接続”](#)」で説明されているように、Bottlerocket ブートストラップコンテナのユーザーデータに使用されるコマンドに `--enable-credentials-file=true` フラグを追加します。

- a. SSM 認証情報プロバイダーを使用している場合、コマンドは次のようになります。

```
eks-hybrid-ssm-setup --activation-id=<activation-id> --activation-code=<activation-code> --region=<region> --enable-credentials-file=true
```

- b. IAM Roles Anywhere 認証情報プロバイダーを使用している場合、コマンドは次のようになります。

```
eks-hybrid-iam-ra-setup --certificate=<certificate> --key=<private-key> --enable-credentials-file=true
```

これによりブートストラップスクリプトが設定され、`/var/eks-hybrid/.aws/credentials` で `eks-pod-identity-agent` ポッドで使用される認証情報ファイルがノード上に作成されます。この認証情報ファイルには定期的に更新される一時的な AWS 認証情報が含まれます。

2. AWS CLI または AWS Management Console のいずれかを使用し、Bottlerocket ハイブリッドノードのサポートを有効にして `eks-pod-identity-agent` をインストールします。

- a. AWS CLI: クラスターの管理に使用しているマシンから、以下のコマンドを実行して `eks-pod-identity-agent` をインストールし、Bottlerocket ハイブリッドノードのサポートを有効にします。my-cluster を自分のクラスター名に置き換えます。

```
aws eks create-addon \
 --cluster-name my-cluster \
 --addon-name eks-pod-identity-agent \
 --configuration-values '{"daemonsets":{"hybrid-bottlerocket":{"create":true}}}'
```

- b. AWS Management Console: AWS コンソールから Pod Identity Agent アドオンをインストールする場合は、オプションの設定に以下を追加して、Bottlerocket ハイブリッドノードを対象とする DaemonSet をデプロイします。

```
{"daemonsets":{"hybrid-bottlerocket":{"create": true}}}
```

## CSI スナップショットコントローラー

バージョン v8.1.0-eksbuild.2 以降、[CSI スナップショットコントローラーアドオン](#)はハイブリッドノードにソフトアンチアフィニティルールを適用し、コントローラーの deployment を Amazon EKS コントロールプレーンと同じ AWS リージョンの EC2 で実行することを優先します。deployment を Amazon EKS コントロールプレーンと同じ AWS リージョンに配置することで、レイテンシーが改善されます。

## コミュニティアドオン

以降のセクションでは、ハイブリッドノード上の互換性のあるコミュニティアドオンを実行する場合と、他の Amazon EKS コンピューティングタイプで同じアドオンを実行する場合の違いについて説明します。

## Kubernetes メトリクスサーバー

コントロールプレーンは、Metrics Server のポッド IP (あるいは hostNetwork が有効になっている場合にはノード IP) に到達しなければなりません。このため、hostNetwork モードで Metrics Server を実行しない限り、Amazon EKS クラスターを作成するときにリモートポッドネットワークを設定し、ポッド IP アドレスをルーティング可能にする必要があります。ポッド IP アドレスをルーティング可能にする場合によく使用される方法の 1 つに、CNI でボーダーゲートウェイプロトコル (BGP) を実装することがあります。

## cert-manager

cert-manager は[ウェブフック](#)を使用します。ハイブリッドノードで cert-manager を実行する場合は、オンプレミスネットワークでオンプレミスポッド CIDR をルーティングし、リモートポッドネットワークで EKS クラスターを設定する必要があります。詳細については、「[Configure webhooks for hybrid nodes](#)」を参照してください。

## ハイブリッドノード用のウェブフックを設定する

このページでは、ハイブリッドノードでウェブフックを実行する場合の考慮事項について詳しく説明します。ウェブフックは、AWS Load Balancer Controller や CloudWatch Observability エージェントといった Kubernetes アプリケーションとオープンソースプロジェクトで実行時にミューテーション機能と検証機能を利用する目的で使用されます。

ハイブリッドノードでウェブフックを実行している場合は、オンプレミスネットワークでオンプレミスポッドの CIDR をルーティングできる必要があります。また、リモートポッドネットワークを使って EKS クラスターを設定し、EKS コントロールプレーンがハイブリッドノードで実行されているウェブフックと通信できるようにする必要があります。

オンプレミスネットワークでオンプレミスポッド CIDR をルーティング可能にするために使用できる手法がいくつかあります。具体的には、ボーダーゲートウェイプロトコル (BGP)、静的ルート、その他のカスタムルーティングソリューションなどです。BGP は、ルートを独自にまたは手動で設定する必要がある他のソリューションよりもスケーラブルで管理しやすいのでお勧めのソリューションです。AWS は、ハイブリッドノードのポッド CIDR をアドバタイズする Cilium と Calico の BGP 機能をサポートしています。詳細については、「[Configure CNI for hybrid nodes](#)」を参照してください。

オンプレミスポッドの CIDR をオンプレミスネットワークでルーティングすることができず、ウェブフックを実行する必要がある場合は、すべてのウェブフックを AWS クラウドで実行することが推奨されます。機能させるには、ウェブフックをハイブリッドノードと同じ EKS クラスターで実行する必要があります。

### 混合モードクラスターの考慮事項

混合モードクラスターは、ハイブリッドノードと AWS クラウドで実行中のノードの両方を備えた EKS クラスターであると定義されています。混合モードクラスターを実行する場合は、以下の推奨事項を検討してください。

- AWS Cloud 内のノードでは VPC CNI を実行し、ハイブリッドノードでは Cilium または Calico を実行します。AWS では、AWS Cloud 内のノードで Cilium と Calico を実行する操作はサポートされていません。
- アプリケーションの要件により AWS Cloud 内のノードで実行中のポッドがハイブリッドノードで実行中のポッドと直接通信しなければならない (「東西通信」)、そのうえ AWS Cloud 内のノードでは VPC CNI を使用し、ハイブリッドノードでは Cilium または Calico をオーバーレイ/トンネルモードで使用している場合は、オンプレミスネットワークでオンプレミスポッド CIDR をルーティングできる必要があります。
- AWS Cloud 内のノードで CoreDNS のレプリカを少なくとも 1 つ実行し、ハイブリッドノードで CoreDNS のレプリカを少なくとも 1 つ実行します。設定手順については、「[Configure add-ons and webhooks for mixed mode clusters](#)」を参照してください。
- AWS Cloud 内のノードで動作するようにウェブフックを設定します。混合モードクラスターを実行しているときに AWS およびコミュニティアドオンによって使用されるウェブフックを設定する方法については、「[Configuring webhooks for add-ons](#)」を参照してください。
- ハイブリッドノードで実行されているワークロードトラフィックに Application Load Balancer (ALB) または Network Load Balancer (NLB) を使用している場合は、ALB または NLB で使用される IP ターゲットを AWS からルーティングできる必要があります。

- Metrics Server アドオンでは、EKS コントロールプレーンから Metrics Server ポッド IP アドレスへの接続が必要になります。ハイブリッドノードで Metrics Server アドオンを実行している場合は、オンプレミスネットワークでオンプレミスポッド CIDR をルーティングできる必要があります。
- Amazon Managed Service for Prometheus (AMP) マネージドコレクターを使用してハイブリッドノードに関するメトリクスを収集するには、オンプレミスネットワークでオンプレミスポッド CIDR をルーティングできる必要があります。また、EKS コントロールプレーンメトリクスと AWS クラウドで実行されているノードには AMP マネージドコレクターを使用し、ハイブリッドノードには AWS Distro for OpenTelemetry (ADOT) アドオンを使用してメトリクスを収集することもできます。

## 混合モードクラスター用のアドオンとウェブフックを設定する

クラスターで実行中のミューテーション用と検証用のウェブフックを表示するには、クラスターの EKS コンソールの [リソース] パネルで [拡張機能] リソースタイプを表示するか、以下のコマンドを使用します。EKS は、クラスターオブザーバビリティダッシュボードにウェブフックメトリクスもレポートします。詳細については、「[the section called “オブザーバビリティダッシュボード”](#)」を参照してください。

```
kubectl get mutatingwebhookconfigurations
```

```
kubectl get validatingwebhookconfigurations
```

## CoreDNS レプリカを設定する

ハイブリッドノードと AWS クラウド内のノードの両方を備える混合モードのクラスターを実行している場合は、ハイブリッドノードと AWS クラウド内のノードのそれぞれに CoreDNS レプリカを 1 つ以上作成しておくことが推奨されます。混合モードクラスターの設定でレイテンシーとネットワークの問題を回避するには、[Service Traffic Distribution](#) を使って最も近くにある CoreDNS レプリカを優先するように CoreDNS サービスを設定します。

Service Traffic Distribution (EKS で Kubernetes バージョン 1.31 以降で使用可能) は、[Topology Aware Routing](#) よりも予測性に優れているため、推奨される方法です。Service Traffic Distribution では、ゾーン内の正常なエンドポイントは、そのゾーンのすべてのトラフィックを受信します。Topology Aware Routing の場合、カスタムルーティングを適用するには各サービスがそのゾーンの複数の条件を満たしている必要があります。さもないと、トラフィックはすべてのエンドポイント

トに均等にルーティングされます。以下のステップで、Service Traffic Distribution を設定していきます。

Cilium を CNI として使用している場合は、enable-service-topology を true に設定して CNI を実行し、Service Traffic Distribution を有効にする必要があります。この設定は Helm インストールフラグ `--set loadBalancer.serviceTopology=true` で渡すことができます。あるいは、Cilium CLI コマンド `cilium config set enable-service-topology true` を使用して既存のインストールを更新することもできます。既存のインストールの設定を更新した場合は、その後各ノードで実行されている Cilium エージェントを再起動する必要があります。

#### 1. ハイブリッドノードごとにトポロジゾーンラベルを追加します。例え

ば、`topology.kubernetes.io/zone: onprem` とします。または、nodeadm 設定でラベルを指定することで、nodeadm init フェーズのラベルを設定できます。「[the section called “kubelet をカスタマイズするためのノード設定 \(オプション\)”](#)」を参照してください。なお、AWS Cloud で実行されているノードは、自身のアベイラビリティーゾーン (AZ) に対応するトポロジゾーンラベルを自動的に適用します。

```
kubectl label node hybrid-node-name topology.kubernetes.io/zone=zone
```

#### 2. トポロジゾーンキーを使用して podAntiAffinity を CoreDNS デプロイに追加します。または、EKS アドオンを使用してインストール時に CoreDNS デプロイを設定することもできます。

```
kubectl edit deployment coredns -n kube-system
```

```
spec:
 template:
 spec:
 affinity:
 ...
 podAntiAffinity:
 preferredDuringSchedulingIgnoredDuringExecution:
 - podAffinityTerm:
 labelSelector:
 matchExpressions:
 - key: k8s-app
 operator: In
 values:
 - kube-dns
 topologyKey: kubernetes.io/hostname
```

```

 weight: 100
 - podAffinityTerm:
 labelSelector:
 matchExpressions:
 - key: k8s-app
 operator: In
 values:
 - kube-dns
 topologyKey: topology.kubernetes.io/zone
 weight: 50
 ...

```

3. kube-dns サービス設定に `trafficDistribution: PreferClose` 設定を追加して、Topology Aware Routing を有効にします。

```

kubectl patch svc kube-dns -n kube-system --type=merge -p '{
 "spec": {
 "trafficDistribution": "PreferClose"
 }
}'

```

4. kube-dns サービスのエンドポイントスライスを表示することで、Service Traffic Distribution が有効になっていることを確認できます。エンドポイントスライスにはトポロジゾーンのラベルの hints が表示されているはずです。これで、Service Traffic Distribution が有効になっていることが確認できます。各エンドポイントアドレスの hints が表示されなければ、Service Traffic Distribution は有効になっていません。

```

kubectl get endpointslice -A | grep "kube-dns"

```

```

kubectl get endpointslice [replaceable]`kube-dns-` -n kube-system -o yaml

```

```

addressType: IPv4
apiVersion: discovery.k8s.io/v1
endpoints:
- addresses:
 - <your-hybrid-node-pod-ip>
 hints:
 forZones:
 - name: onprem
 nodeName: <your-hybrid-node-name>
 zone: onprem

```

```
- addresses:
 - <your-cloud-node-pod-ip>
hints:
 forZones:
 - name: us-west-2a
nodeName: <your-cloud-node-name>
zone: us-west-2a
```

## アドオン用のウェブフックを設定する

以下のアドオンは、ウェブフックを使用しており、ハイブリッドノードでの使用がサポートされています。

- AWS ロードバランサーコントローラー
- Amazon CloudWatch オペサビリティ エージェント
- AWS オープンテレメトリー用ディストリビューター (ADOT)
- cert-manager

これらのアドオンで使用するウェブフックを AWS クラウド内のノードで実行するように設定する方法については、以下のセクションを参照してください。

### AWS ロードバランサーコントローラー

AWS Load Balancer Controller を混合モードクラスターの設定で使用するには、このコントローラーを AWS クラウドのノードで実行する必要があります。それには、Helm 値の設定に以下を追加するか、EKS アドオン設定を使用して値を指定します。

```
affinity:
 nodeAffinity:
 requiredDuringSchedulingIgnoredDuringExecution:
 nodeSelectorTerms:
 - matchExpressions:
 - key: eks.amazonaws.com/compute-type
 operator: NotIn
 values:
 - hybrid
```

## Amazon CloudWatch オペサビリティ エージェント

CloudWatch Observability Agent アドオンには、ウェブフックを使用する Kubernetes Operator があります。混合モードクラスター設定の AWS Cloud 内のノードでオペレーターを実行するには、CloudWatch Observability エージェントオペレーター設定を編集します。インストール時に Helm と EKS アドオンを使用してオペレーターアフィニティを設定することはできません (「[containers-roadmap issue #2431](#)」を参照のこと)。

```
kubectrl edit -n amazon-cloudwatch deployment amazon-cloudwatch-observability-controller-manager
```

```
spec:
 ...
 template:
 ...
 spec:
 affinity:
 nodeAffinity:
 requiredDuringSchedulingIgnoredDuringExecution:
 nodeSelectorTerms:
 - matchExpressions:
 - key: eks.amazonaws.com/compute-type
 operator: NotIn
 values:
 - hybrid
```

## AWS オープンテレメトリ用ディストリビューター (ADOT)

AWS Distro for OpenTelemetry (ADOT) アドオンには、ウェブフックを使用する Kubernetes Operator があります。混合モードクラスター設定の AWS クラウド内のノードでオペレーターを実行するには、Helm 値設定に以下を追加するか、EKS アドオン設定を使用して値を指定します。

```
affinity:
 nodeAffinity:
 requiredDuringSchedulingIgnoredDuringExecution:
 nodeSelectorTerms:
 - matchExpressions:
 - key: eks.amazonaws.com/compute-type
 operator: NotIn
 values:
 - hybrid
```



ポッド CIDR をオンプレミスネットワークでルーティングできない場合は、ADOT コレクターをハイブリッドノードで実行して、ハイブリッドノードとそこで実行されているワークロードからメトリクスをスクレイピングする必要があります。これを行うには、カスタムリソース定義 (CRD) を編集します。

```
kubectl -n opentelemetry-operator-system edit opentelemetrycollectors.opentelemetry.io
adot-col-prom-metrics
```

```
spec:
 affinity:
 nodeAffinity:
 requiredDuringSchedulingIgnoredDuringExecution:
 nodeSelectorTerms:
 - matchExpressions:
 - key: eks.amazonaws.com/compute-type
 operator: In
 values:
 - hybrid
```

ADOT コレクター CRD 設定で各 `scrape_configs` に以下の `relabel_configs` を追加することで、ハイブリッドノードとそこで実行されているリソースからのみメトリクスをスクレイピングするように ADOT コレクターを設定できます。

```
relabel_configs:
 - action: keep
 regex: hybrid
 source_labels:
 - __meta_kubernetes_node_label_eks_aws_com_compute_type
```

ADOT アドオンには、ADOT オペレーターウェブフックで使用する TLS 証明書に `cert-manager` をインストールするための前提条件があります。`cert-manager` はウェブフックも実行し、次の Helm 値設定を使用して AWS クラウドのノードで実行するように設定することができます。

```
affinity:
 nodeAffinity:
 requiredDuringSchedulingIgnoredDuringExecution:
 nodeSelectorTerms:
 - matchExpressions:
 - key: eks.amazonaws.com/compute-type
 operator: NotIn
 values:
```

```
 - hybrid
webhook:
 affinity:
 nodeAffinity:
 requiredDuringSchedulingIgnoredDuringExecution:
 nodeSelectorTerms:
 - matchExpressions:
 - key: eks.amazonaws.com/compute-type
 operator: NotIn
 values:
 - hybrid
cainjector:
 affinity:
 nodeAffinity:
 requiredDuringSchedulingIgnoredDuringExecution:
 nodeSelectorTerms:
 - matchExpressions:
 - key: eks.amazonaws.com/compute-type
 operator: NotIn
 values:
 - hybrid
startupapicheck:
 affinity:
 nodeAffinity:
 requiredDuringSchedulingIgnoredDuringExecution:
 nodeSelectorTerms:
 - matchExpressions:
 - key: eks.amazonaws.com/compute-type
 operator: NotIn
 values:
 - hybrid
```

## cert-manager

cert-manager アドオンは、ウェブフックを実行し、以下の Helm 値設定を使用して AWS クラウド内のノードで実行するように設定することができます。

```
affinity:
 nodeAffinity:
 requiredDuringSchedulingIgnoredDuringExecution:
 nodeSelectorTerms:
 - matchExpressions:
 - key: eks.amazonaws.com/compute-type
```

```
 operator: NotIn
 values:
 - hybrid
webhook:
 affinity:
 nodeAffinity:
 requiredDuringSchedulingIgnoredDuringExecution:
 nodeSelectorTerms:
 - matchExpressions:
 - key: eks.amazonaws.com/compute-type
 operator: NotIn
 values:
 - hybrid
cainjector:
 affinity:
 nodeAffinity:
 requiredDuringSchedulingIgnoredDuringExecution:
 nodeSelectorTerms:
 - matchExpressions:
 - key: eks.amazonaws.com/compute-type
 operator: NotIn
 values:
 - hybrid
startupapicheck:
 affinity:
 nodeAffinity:
 requiredDuringSchedulingIgnoredDuringExecution:
 nodeSelectorTerms:
 - matchExpressions:
 - key: eks.amazonaws.com/compute-type
 operator: NotIn
 values:
 - hybrid
```

## ハイブリッドノードのプロキシを設定する

オンプレミス環境でデータセンターまたはエッジ環境から出ていくトラフィックにプロキシサーバーを使用している場合は、プロキシサーバーを使用するように個別にノードとクラスターを設定する必要があります。

## クラスター

クラスターでは、プロキシサーバーを使用するように kube-proxy を設定する必要があります。Amazon EKS クラスターの作成後に kube-proxy を設定する必要があります。

## ノード

ノードでは、プロキシサーバーを使用するようにオペレーティングシステム、containerd、kubelet、Amazon SSM エージェントを設定する必要があります。こうした変更は、オペレーティングシステムイメージのビルドプロセス中に加えることも、各ハイブリッドノードで nodeadm init を実行する前に加えることもできます。

## ノードレベルの設定

以下の設定は、オペレーティングシステムイメージに適用するか、各ハイブリッドノードで nodeadm init を実行する前に適用する必要があります。

### containerd プロキシ設定

containerd は、Kubernetes のデフォルトのコンテナ管理ランタイムです。インターネットアクセスにプロキシを使用している場合は、Kubernetes と Amazon EKS に必要なコンテナイメージをプルできるように containerd を設定する必要があります。

次の内容で、各ハイブリッドノード上の /etc/systemd/system/containerd.service.d ディレクトリに「http-proxy.conf」という名前のファイルを作成します。proxy-domain および port を環境の値で置き換えます。

```
[Service]
Environment="HTTP_PROXY=http://proxy-domain:port"
Environment="HTTPS_PROXY=http://proxy-domain:port"
Environment="NO_PROXY=localhost"
```

### ユーザーデータからの containerd 設定

このファイル用に containerd.service.d ディレクトリを作成する必要があります。systemd を再ロードして設定ファイルを選択すると、再起動しなくても済むようになります。AL2023 では、スクリプトを実行した時点で既にサービスが実行されている可能性があります。その場合は再起動も必要です。

```
mkdir -p /etc/systemd/system/containerd.service.d
echo '[Service]' > /etc/systemd/system/containerd.service.d/http-proxy.conf
```

```
echo 'Environment="HTTP_PROXY=http://proxy-domain:port"' >> /etc/systemd/system/
containerd.service.d/http-proxy.conf
echo 'Environment="HTTPS_PROXY=http://proxy-domain:port"' >> /etc/systemd/system/
containerd.service.d/http-proxy.conf
echo 'Environment="NO_PROXY=localhost"' >> /etc/systemd/system/containerd.service.d/
http-proxy.conf
systemctl daemon-reload
systemctl restart containerd
```

## kubelet プロキシ設定

kubelet は、各 Kubernetes ノードで実行される Kubernetes ノードエージェントであり、そのノードで実行されているノードとポッドの管理を担当します。オンプレミス環境でプロキシを使用している場合は、Amazon EKS クラスターのパブリックエンドポイントまたはプライベートエンドポイントと通信できるように kubelet を設定する必要があります。

次の内容で、各ハイブリッドノードの `/etc/systemd/system/kubelet.service.d/` ディレクトリに「`http-proxy.conf`」という名前のファイルを作成します。`proxy-domain` および `port` を環境の値で置き換えます。

```
[Service]
Environment="HTTP_PROXY=http://proxy-domain:port"
Environment="HTTPS_PROXY=http://proxy-domain:port"
Environment="NO_PROXY=localhost"
```

## ユーザーデータからの kubelet 設定

このファイル用に `kubelet.service.d` ディレクトリを作成する必要があります。`systemd` を再ロードして設定ファイルを選択すると、再起動しなくても済むようになります。AL2023 では、スクリプトを実行した時点で既にサービスが実行されている可能性があります。その場合は再起動も必要です。

```
mkdir -p /etc/systemd/system/kubelet.service.d
echo '[Service]' > /etc/systemd/system/kubelet.service.d/http-proxy.conf
echo 'Environment="HTTP_PROXY=http://proxy-domain:port"' >> /etc/systemd/system/
kubelet.service.d/http-proxy.conf
echo 'Environment="HTTPS_PROXY=http://proxy-domain:port"' >> /etc/systemd/system/
kubelet.service.d/http-proxy.conf
echo 'Environment="NO_PROXY=localhost"' >> /etc/systemd/system/kubelet.service.d/http-
proxy.conf
systemctl daemon-reload
```

```
systemctl restart kubelet
```

## ssm プロキシ設定

ssm は認証情報プロバイダーの 1 つであり、これを使用するとハイブリッドノードを初期化できます。ssm は、AWS での認証を実施し、kubelet によって使用される一時的な認証情報を生成します。オンプレミス環境でプロキシを使用し、ノードで ssm を認証情報プロバイダーとして使用している場合は、Amazon SSM サービスエンドポイントと通信できるように ssm を設定する必要があります。

オペレーティングシステムに応じて、各ハイブリッドノードに http-proxy.conf という名前でファイルを以下のパスに作成します。

- Ubuntu - /etc/systemd/system/snap.amazon-ssm-agent.amazon-ssm-agent.service.d/http-proxy.conf
- Amazon Linux 2023 および Red Hat Enterprise Linux - /etc/systemd/system/amazon-ssm-agent.service.d/http-proxy.conf

ファイルには次の内容を入力します。proxy-domain および port を環境の値で置き換えます。

```
[Service]
Environment="HTTP_PROXY=http://proxy-domain:port"
Environment="HTTPS_PROXY=http://proxy-domain:port"
Environment="NO_PROXY=localhost"
```

## ユーザーデータからの ssm 設定

このファイル用に ssm システムサービスファイルディレクトリを作成する必要があります。ディレクトリパスは、ノードで使用されているオペレーティングシステムによって異なります。

- Ubuntu - /etc/systemd/system/snap.amazon-ssm-agent.amazon-ssm-agent.service.d
- Amazon Linux 2023 および Red Hat Enterprise Linux - /etc/systemd/system/amazon-ssm-agent.service.d

ノードで使用されているオペレーティングシステムに応じて、以下の restart コマンドの systemd サービス名を置き換えます。

- Ubuntu - snap.amazon-ssm-agent.amazon-ssm-agent

- Amazon Linux 2023 および Red Hat Enterprise Linux - amazon-ssm-agent

```
mkdir -p systemd-service-file-directory
echo '[Service]' > [.replaceable]#systemd-service-file-directory/http-proxy.conf
echo 'Environment="HTTP_PROXY=http://[.replaceable]#proxy-domain:port"' >> systemd-
service-file-directory/http-proxy.conf
echo 'Environment="HTTPS_PROXY=http://[.replaceable]#proxy-domain:port"' >>
[.replaceable]#systemd-service-file-directory/http-proxy.conf
echo 'Environment="NO_PROXY=localhost"' >> [.replaceable]#systemd-service-file-
directory/http-proxy.conf
systemctl daemon-reload
systemctl restart [.replaceable]#systemd-service-name
```

## オペレーティングシステムのプロキシ設定

インターネットアクセスにプロキシを使用している場合は、オペレーティングシステムのパッケージマネージャーからハイブリッドノードの依存関係をプルできるようにオペレーティングシステムを設定する必要があります。

### Ubuntu

1. 次のコマンドでプロキシを使用するように snap を設定します。

```
sudo snap set system proxy.https=http://proxy-domain:port
sudo snap set system proxy.http=http://proxy-domain:port
```

2. apt のプロキシを有効にするには、/etc/apt/ ディレクトリに apt.conf という名前のファイルを作成します。proxy-domain と port を環境の値に置き換えます。

```
Acquire::http::Proxy "http://proxy-domain:port";
Acquire::https::Proxy "http://proxy-domain:port";
```

### Amazon Linux 2023

1. プロキシを使用するように dnf を設定します。環境のプロキシドメイン値とポート値を使用してファイル /etc/dnf/dnf.conf を作成します。

```
proxy=http://proxy-domain:port
```

## Red Hat Enterprise Linux

1. プロキシを使用するように yum を設定します。環境のプロキシドメイン値とポート値を使用してファイル `/etc/yum.conf` を作成します。

```
proxy=http://proxy-domain:port
```

## IAM Roles Anywhere のプロキシ設定

IAM Roles Anywhere 認証情報プロバイダーサービスは、`enableCredentialsFile` フラグと共に IAM Roles Anywhere と使用する際に認証情報を更新する役割があります ([the section called “EKS Pod Identity エージェント”](#) を参照)。オンプレミス環境でプロキシを使用している場合は、IAM Roles Anywhere エンドポイントと通信できるようにサービスを設定する必要があります。

次の内容で、`/etc/systemd/system/aws_signing_helper_update.service.d/` ディレクトリに「`http-proxy.conf`」という名前のファイルを作成します。`proxy-domain` および `port` を環境の値で置き換えます。

```
[Service]
Environment="HTTP_PROXY=http://proxy-domain:port"
Environment="HTTPS_PROXY=http://proxy-domain:port"
Environment="NO_PROXY=localhost"
```

## クラスター全体の設定

このセクションの設定は、Amazon EKS クラスターを作成した後、および各ハイブリッドノードで `nodeadm init` を実行する前に適用する必要があります。

## kube-proxy のプロキシ設定

Amazon EKS は、ハイブリッドノードがクラスターに参加すると、各ハイブリッドノードに DaemonSet として自動的に kube-proxy をインストールします。kube-proxy は、Amazon EKS クラスターのポッドによってバックアップされるサービス間のルーティングを有効にします。各ホストを設定するために、kube-proxy には Amazon EKS クラスターエンドポイントの DNS 解決が必要です。



## 1. 次のコマンドを使用して kube-proxy DaemonSet を編集する

```
kubectl -n kube-system edit ds kube-proxy
```

これにより、設定されたエディタで kube-proxy DaemonSet 定義が開きます。

## 2. HTTP\_PROXY および HTTPS\_PROXY の環境変数を追加します。NODE\_NAME 環境変数は設定に既に存在している必要があります。proxy-domain と port を環境の値に置き換えます。

```
containers:
 - command:
 - kube-proxy
 - --v=2
 - --config=/var/lib/kube-proxy-config/config - --hostname-override=$(NODE_NAME)
 env:
 - name: HTTP_PROXY
 value: http://proxy-domain:port
 - name: HTTPS_PROXY
 value: http://proxy-domain:port
 - name: NODE_NAME
 valueFrom:
 fieldRef:
 apiVersion: v1
 fieldPath: spec.nodeName
```

## ハイブリッドノードの概念

Amazon EKS Hybrid Nodes を使用すると、オンプレミス環境またはエッジ環境で実行されている物理マシンまたは仮想マシンを、AWS クラウドで実行されている Amazon EKS クラスターに接続できます。このアプローチには多くの利点がありますが、単一のネットワーク環境で Kubernetes クラスターを運用することに慣れているユーザーにとっては、新しいネットワークの概念やアーキテクチャが導入されることにもなります。

以下のセクションでは、EKS Hybrid Nodes での Kubernetes とネットワーキングの概念、およびハイブリッドアーキテクチャにおけるトラフィックフローについて詳しく説明します。これらのセクションを理解するには、ポッド、ノード、サービス、Kubernetes コントロールプレーン、kubelet、kube-proxy の概念など、Kubernetes ネットワークに関する基本的な知識に精通している必要があります。

これらのページは、「[the section called “ネットワーキングの概念”](#)」、「[the section called “Kubernetes の概念”](#)」、「[the section called “トラフィックフロー”](#)」の順に読むことをお勧めします。

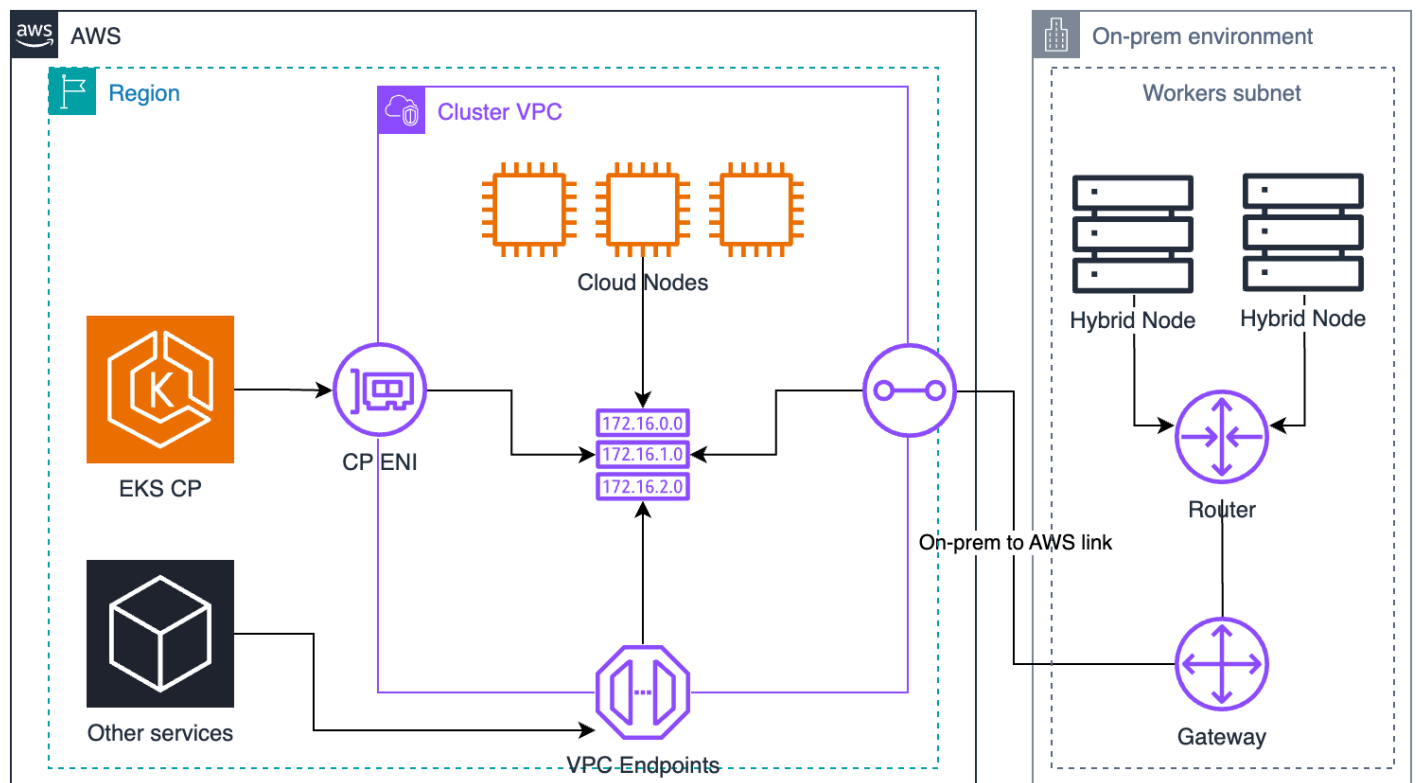
## トピック

- [ハイブリッドノードにおけるネットワーキングの概念](#)
- [ハイブリッドノードにおける Kubernetes の概念](#)
- [ハイブリッドノード向けネットワークトラフィックフロー](#)

## ハイブリッドノードにおけるネットワーキングの概念

このセクションでは、EKS Hybrid Nodes のネットワークポロジ設計の際に考慮すべき、コアネットワーキングの概念と制約について詳しく説明します。

### EKS Hybrid Nodes におけるネットワーキングの概念



### ネットワークハブとしての VPC

クラウドの境界を越えるすべてのトラフィックは VPC を経由しますが、これには、AWS で稼働する EKS コントロールプレーンまたはポッドや、ハイブリッドノードまたはそうしたノードで稼

働しているポッド間のトラフィックも含まれます。クラスターの VPC はハイブリッドノードと残りのクラスター部分のネットワークハブとして機能している、と考えるとよいでしょう。このアーキテクチャによってトラフィックとそのルーティングを完全に制御できますが、これを導入するには、VPC のルート、セキュリティグループ、ファイアウォールを適切に設定する責任を負うことにもなります。

## EKS コントロールプレーンから VPC に向かうトラフィック

EKS コントロールプレーンにより、Elastic Network Interface (ENI) が VPC にアタッチされ、これらの ENI によって、EKS API サーバーとの間でトラフィックが処理されます。EKS コントロールプレーンの ENI の配置は、クラスターを設定する際に制御します。なぜなら、クラスターの作成中に渡すサブネットに ENI がアタッチされるからです。

EKS では、セキュリティグループを、サブネットにアタッチする ENI に関連付けます。こうしたセキュリティグループにより、EKS コントロールプレーンとの間のトラフィックを、ENI を介して許可します。この点が EKS Hybrid Nodes では重要となります。ハイブリッドノードとそこで稼働しているポッドから EKS コントロールプレーン ENI へのトラフィックを許可する必要があるからです。

## リモートノードネットワーク

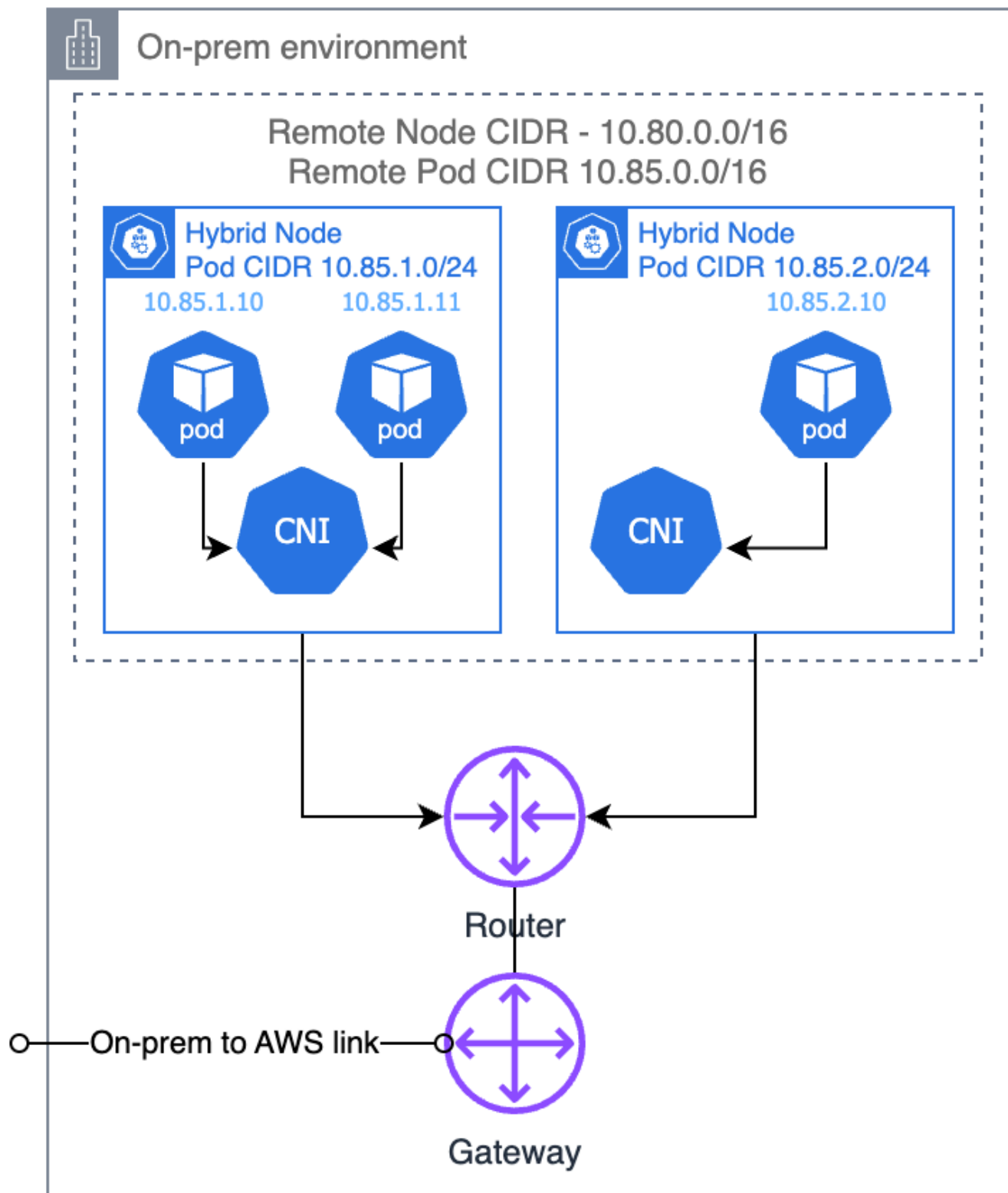
リモートノードネットワーク、具体的にはリモートノード CIDR は、ハイブリッドノードとして使用するマシンに割り当てる IP アドレスの範囲を意味します。ハイブリッドノードをプロビジョニングすると、それらはオンプレミスのデータセンターやエッジロケーションに配置されます。これらのネットワークドメインは、EKS コントロールプレーンおよび VPC のドメインとは異なります。また、各ハイブリッドノードには、IP アドレスが 1 つまたは複数あり、VPC 内のサブネットとは異なるリモートノード CIDR に基づいて設定されています。

こうしたリモートノード CIDR を使用して EKS クラスターを構成すると、EKS でそれが認識され、ハイブリッドノードの IP アドレスを宛先としたすべてのトラフィック (kubelet API へのリクエストなど) をクラスターの VPC 経由でルーティングできるようになります。

## リモートポッドネットワーク

ここでのリモートポッドネットワークとは、ハイブリッドノードで稼働するポッドに割り当てる IP アドレスの範囲を意味します。一般的には、これらの範囲で CNI を設定すると、CNI の IP Address Manager (IPAM) 機能によって、その範囲の一部が各ハイブリッドノードに割り当てられます。また、新規作成したポッドには、ポッドをスケジューリングしたノードに割り当てた IP アドレスからアドレスが割り当てられます。

こうしたリモートポッド CIDR を使用して EKS クラスターを設定すると、EKS コントロールプレーンでそれが認識され、ハイブリッドノードで稼働しているポッドを宛先とするすべてのトラフィック (ウェブフックとの通信など) をクラスターの VPC 経由でルーティングできます。



オンプレミスから VPC に向かうトラフィック

ハイブリッドノードに使用するオンプレミスネットワークのトラフィックは、EKS クラスターに使用する VPC にルーティングする必要があります。オンプレミスネットワークを VPC に接続する場合、複数用意されている [Network-to-Amazon VPC 接続オプション](#)を利用することも、独自の VPN ソリューションを使用することもできます。

ここで重要なのは、AWS クラウド側の VPC とオンプレミスネットワーク間のルーティングを正しく設定することで、両方のネットワークの接続点を介して、適切なトラフィックが相互にルーティングされるようにすることです。

VPC では、リモートノードおよびリモートポッドネットワークに向かうすべてのトラフィックを、オンプレミスネットワークとの接続点 (ゲートウェイ) 経由でルーティングする必要があります。サブネットの一部に異なるルートテーブルがある場合は、ハイブリッドノードとそこで稼働しているポッドのルートを各ルートテーブルに設定する必要があります。これは、EKS コントロールプレーンの ENI がアタッチされるサブネットや、ハイブリッドノードと通信する必要のある EC2 のノードまたはポッドが含まれるサブネットにも当てはまります。

オンプレミスネットワークでは、ネットワークの設定によって、EKS クラスターの VPC や、ハイブリッドノードに必要なその他の AWS サービスとの間で生じるトラフィックを許可する必要があります。EKS クラスターのトラフィックは、ゲートウェイを双方向に通過することになります。

## ネットワーク上の制約

### 完全にルーティングされたネットワーク

ここでの主な制約は、EKS コントロールプレーンとすべてのノード (クラウドまたはハイブリッドのノード) で生じるトラフィックが完全にルーティングされるネットワークを形成する必要があることです。つまり、すべてのノードが IP アドレスを使用してレイヤー 3 で互いに到達可能でなければなりません。

EKS コントロールプレーンとクラウドノードはフラットなネットワーク (VPC) 内にあるため、相互に到達可能な状態にあります。ハイブリッドノードは異なるネットワークドメインに存在しています。そのため、VPC とオンプレミスネットワークでルーティング設定を追加して、ハイブリッドノードと他のクラスター部分との間で生じるトラフィックをルーティングする必要があります。ハイブリッドノードが互いに到達可能で、VPC から到達可能な場合、ハイブリッドノードは、単一のフラットネットワークに配置することも、セグメント化した複数のネットワークに配置することもできます。

### ルーティング可能なリモートポッド CIDR

EKS コントロールプレーンがハイブリッドノードで稼働中のポッド (ウェブフックやメトリクスサーバーなど) と通信したり、クラウドノードで稼働中のポッドがハイブリッドノードで稼働中のポッド

と通信 (ワークロードの East-West 通信) したりするには、リモートポッドの CIDR へのトラフィックが VPC からルーティング可能でなければなりません。つまり、VPC では、ポッドの CIDR に向かうトラフィックをゲートウェイ経由でオンプレミスネットワークにルーティングでき、オンプレミスネットワークでは、ポッドのトラフィックを適切なノードにルーティングできる必要があります。

ここで重要なのは、VPC とオンプレミスでポッドのルーティング要件に違いがあることです。VPC には、リモートポッドへのトラフィックがゲートウェイを通過する必要があることを認識させるだけで済み、リモートポッドの CIDR が 1 つしかない場合は、必要なルートも 1 つのみです。

この要件は、オンプレミスネットワーク内の、ハイブリッドノードと同じサブネットでローカルルーターに至るまでに生じるすべてのホップに当てはまります。このルーターは、各ノードに割り当てられたポッド CIDR スライスを認識する必要がある唯一のルーターです。これにより、特定のポッドに向かうトラフィックを、そのポッドがスケジュールされているノードに配信できます。

オンプレミスのポッド CIDR へのこうしたルートを、ローカルのオンプレミスルーターから VPC ルートテーブルに伝達することもできますが、これは必須ではありません。オンプレミスのポッド CIDR を頻繁に変更しているため、VPC ルートテーブルを更新してそれらの変更を反映する必要がある場合は、オンプレミスのポッド CIDR を VPC ルートテーブルに伝達することが推奨されます。しかし、こうしたケースはまれです。

オンプレミスのポッド CIDR へのルーティングは、必ずしも設定する必要はありません。ハイブリッドノードでウェブフックを実行する必要がない場合、またはクラウドノード上のポッドがハイブリッドノード上のポッドと通信する必要がない場合は、オンプレミスネットワークのポッド CIDR へのルーティングを設定しなくても済みます。

では、ハイブリッドノードを使用するときに、オンプレミスのポッド CIDR 宛てトラフィックをルーティング可能にする理由について説明します。

VPC CNI を備えた EKS をクラウドノードに使用している場合、VPC CNI によって VPC からポッドに直接 IP が割り当てられます。つまり、クラウドポッドも EKS コントロールプレーンもポッド IP に直接到達できるため、特別なルーティングは必要ありません。

オンプレミスでポッドが稼働している場合 (クラウドで他の CNI を使用)、ポッドは、通常、分離されたオーバーレイネットワークで実行され、CNI によってポッド間のトラフィック配信が処理されます。これは通常、カプセル化によって行われます。つまり、ポッド間のトラフィックをノード間のトラフィックに変換し、両端でカプセル化とカプセル化解除を行います。そのため、ノードとルーターに設定をさらに追加する必要はありません。

ハイブリッドノードを使用したネットワークは、両方のトポロジが組み合わさっているという点で独特なものです。つまり、EKS コントロールプレーンとクラウドノード (VPC CNI を使用) は、ノー

ドとポッドが含まれるフラットなネットワークを想定して動作し、一方、ハイブリッドノードで稼働するポッドは、オーバーレイネットワーク内に存在し、カプセル化には VXLAN (Cilium のデフォルト) が使用されます。ハイブリッドノードで稼働するポッドは、オンプレミスネットワークと VPC 間のルーティングが可能であることを前提に、EKS コントロールプレーンとクラウドノードで稼働中のポッドに到達できます。ただし、オンプレミスネットワークのポッド CIDR 宛てトラフィックをルーティングしておらず、オーバーレイネットワークに到達したトラフィックを適切なノードに転送できない場合、オンプレミスのポッド IP に戻るトラフィックは、最終的にドロップされます。

## ハイブリッドノードにおける Kubernetes の概念

このページでは、EKS Hybrid Nodes システムアーキテクチャを支える Kubernetes の主要な概念について詳しく説明します。

### VPC の EKS コントロールプレーン

EKS コントロールプレーン ENI の IP は、default デフォルトの名前空間にある `kubernetes Endpoints` オブジェクトに保存されます。EKS では、ENI の新規作成や削除が行われると、このオブジェクトが更新され、IP リストが常に最新状態に保たれます。

これらのエンドポイントは、`kubernetes` サービスを通じて使用できます。このサービスも default の名前空間に存在します。この `ClusterIP` タイプのサービスには、クラスターのサービス CIDR に含まれる最初の IP が割り当てられます。例えば、サービス CIDR が `172.16.0.0/16` の場合、サービス IP は `172.16.0.1` になります。

通常、ポッドは (稼働がクラウドノード上かハイブリッドノード上かに関係なく) そのように EKS Kubernetes API サーバーにアクセスします。サービス IP を宛先 IP として使用し、これをいずれかの EKS コントロールプレーン ENI に設定されている実際の IP に変換します。こうした動作の主な例外として `kube-proxy` があります。これによって変換を設定するからです。

### EKS API サーバーエンドポイント

EKS API サーバーには `kubernetes` サービス IP 以外の方法でもアクセスできます。EKS では、クラスター作成時に Route53 DNS 名も作成されます。これが、EKS の `endpoint API` アクションを呼び出す際に、EKS クラスターの `DescribeCluster` フィールド値になります。

```
{
 "cluster": {
 "endpoint": "https://xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx.gr7.us-
west-2.eks.amazonaws.com",
 "name": "my-cluster",
```



```
 "status": "ACTIVE"
 }
}
```

パブリックエンドポイントアクセスクラスター、もしくはパブリックおよびプライベートエンドポイントアクセスクラスターでは、ハイブリッドノードで、この DNS 名がインターネット経由でルーティング可能なパブリック IP に解決されます (デフォルトの場合)。プライベートエンドポイントアクセスクラスターの場合、DNS 名は、EKS コントロールプレーン ENI のプライベート IP に解決されます。

kubelet と kube-proxy は、そのように Kubernetes API サーバーにアクセスします。すべての Kubernetes クラスタートラフィックを VPC 経由にする場合、クラスターをプライベートアクセスモードで設定するか、オンプレミスの DNS サーバーを変更して EKS クラスタートエンドポイントを EKS コントロールプレーン ENI のプライベート IP に解決する必要があります。

## kubelet エンドポイント

kubelet では複数の REST エンドポイントが公開されており、これによってシステムの他の要素は、kubelet と通信し、各ノードの情報を収集しています。ほとんどのクラスターにおいて、kubelet サーバーに向かうトラフィックの大部分は、コントロールプレーンで生じたものですが、特定のモニタリングエージェントも kubelet と通信する場合があります。

このインターフェイスを通じて、kubelet では、ログの取得 (`kubect1 logs`)、コンテナ内でのコマンド実行 (`kubect1 exec`)、トラフィックのポート転送 (`kubect1 port-forward`) といった各種リクエストが処理されます。こうしたリクエストでは、基盤のコンテナランタイム環境との通信が kubelet を介して行われるため、クラスター管理者や開発者にとってシームレスなユーザー体験が実現します。

この API で最も一般的なコンシューマーとなるのは Kubernetes API サーバーです。上記の `kubect1` コマンドのいずれかを使用すると、`kubect1` が API サーバーに API リクエストを送信し、その後 API サーバーが、ポッドが稼働しているノードの kubelet API を呼び出します。そのため、多くの場合、ノード IP が EKS コントロールプレーンから到達可能である必要があります。また、ノードルートを誤って設定すると、ポッドが稼働中であっても、ログや `exec` コマンドを利用できません。

## ノード IP

EKS コントロールプレーンは、ノードと通信する際に、Node オブジェクトのステータス (`status.addresses`) で報告されるアドレスの 1 つを使用します。

EKS クラウドノードでは、一般的に、ノード登録時に kubelet が EC2 インスタンスのプライベート IP を InternalIP として報告します。その後、Cloud Controller Manager (CCM) がこの IP を検証し、それが EC2 インスタンスに属していることを確認します。さらに、CCM は通常、インスタンスのパブリック IP (ExternalIP) と DNS 名 (InternalDNS と ExternalDNS) をノードステータスに追加します。

ただし、ハイブリッドノードには CCM はありません。EKS Hybrid Nodes CLI (nodeadm) を使用してハイブリッドノードを登録すると、CCM を介さずにノードのステータスでマシンの IP が直接報告されるように kubelet が設定されます。

```
apiVersion: v1
kind: Node
metadata:
 name: my-node-1
spec:
 providerID: eks-hybrid:///us-west-2/my-cluster/my-node-1
status:
 addresses:
 - address: 10.1.1.236
 type: InternalIP
 - address: my-node-1
 type: Hostname
```

マシンに複数の IP がある場合、kubelet は独自のロジックに従ってそれらの 1 つを選択します。選択される IP アドレスは `--node-ip` フラグで制御でき、このフラグは、`nodeadm config` の `spec.kubelet.flags` に渡すことが可能です。Node オブジェクトで報告された IP アドレスにのみ VPC からのルートが必要であり、マシンには、クラウドからアクセスできない他の IP アドレスを割り当てることができます。

## kube-proxy

kube-proxy は、各ノードのネットワークレイヤーでサービス抽象化を実装する役割を果たし、Kubernetes サービスに向かうトラフィックのネットワークプロキシおよびロードバランサーとして機能します。また、kube-proxy は、Kubernetes API サーバーを監視し、サービスとエンドポイントに関連した変更を継続的に確認することで、基盤となるホストのネットワークルールを動的に更新し、適切なトラフィック転送を維持します。

iptables モードの kube-proxy は、複数の netfilter チェーンをプログラミングして、サービストラフィックを処理します。こうしたルールによって、以下の階層が形成されます。

1. KUBE-SERVICES チェーン: すべてのサービストラフィックのエントリポイント。各サービスの ClusterIP とポートに一致するルールが定義されています。
2. KUBE-SVC-XXX チェーン: サービス固有のチェーンに、各サービスの負荷分散ルールが定義されています。
3. KUBE-SEP-XXX チェーン: エンドポイント固有のチェーンに、実際の DNAT ルールが定義されています。

次に示す default 名前空間内のサービス test-server の状況を確認してみましょう。\* サービス ClusterIP: 172.16.31.14 \* サービスポート: 80 \* バックアップポッド: 10.2.0.110, 10.2.1.39, 10.2.2.254

iptables ルールを検査するとき (iptables-save | grep -A10 KUBE-SERVICES を使用):

1. KUBE-SERVICES チェーンには、以下のように、サービスに一致するルールがあります。

```
-A KUBE-SERVICES -d 172.16.31.14/32 -p tcp -m comment --comment "default/test-server cluster IP" -m tcp --dport 80 -j KUBE-SVC-XYZABC123456
```

- このルールは、172.16.31.14:80 宛てのパケットに一致させる
- コメントは、このルールの目的を示している: default/test-server cluster IP
- 一致するパケットは KUBE-SVC-XYZABC123456 チェーンにジャンプさせる

2. KUBE-SVC-XYZABC123456 チェーンには、確率ベースの負荷分散ルールが定義されています。

```
-A KUBE-SVC-XYZABC123456 -m statistic --mode random --probability 0.33333333349 -j KUBE-SEP-POD1XYZABC
-A KUBE-SVC-XYZABC123456 -m statistic --mode random --probability 0.500000000000 -j KUBE-SEP-POD2XYZABC
-A KUBE-SVC-XYZABC123456 -j KUBE-SEP-POD3XYZABC
```

- 最初のルール: 33.3% の確率で KUBE-SEP-POD1XYZABC にジャンプさせる
- 2 番目のルール: 50% の確率で残りのトラフィック (全体の 33.3%) を KUBE-SEP-POD2XYZABC にジャンプさせる
- 最後のルール: 残りのすべてのトラフィック (合計の 33.3%) を KUBE-SEP-POD3XYZABC にジャンプさせる

3. 個々の KUBE-SEP-XXX チェーンで、DNAT (宛先 NAT) を実行します。

```
-A KUBE-SEP-POD1XYZABC -p tcp -m tcp -j DNAT --to-destination 10.2.0.110:80
```

```
-A KUBE-SEP-POD2XYZABC -p tcp -m tcp -j DNAT --to-destination 10.2.1.39:80
-A KUBE-SEP-POD3XYZABC -p tcp -m tcp -j DNAT --to-destination 10.2.2.254:80
```

- これらの DNAT ルールでは、宛先 IP とポートを書き換えて、トラフィックを特定のポッドに転送します。
- 各ルールによって、トラフィックの約 33.3% を処理し、10.2.0.110、10.2.1.39、10.2.2.254 の間で均等な負荷分散を行います。

このマルチレベルチェーン構造により、kube-proxy では、データパスにプロキシプロセスを必要とせずに、カーネルレベルのパケット操作によってサービスの負荷分散およびリダイレクトを効率的に実装できます。

## Kubernetes 運用への影響

ノードで kube-proxy の機能が失われると、ノードはサービストラフィックを適切にルーティングできなくなり、クラスターサービスに依存するポッドのタイムアウトや接続の失敗が発生します。この問題が特に深刻になりうるのは、ノードが最初に登録されたときです。CNI では、ポッドネットワークを設定する前に、Kubernetes API サーバーと通信して、ノードのポッド CIDR などの情報を取得する必要があります。それを行うには、kubernetes サービス IP を使用します。ただし、kube-proxy が起動できなかつたり、適切な iptables ルールを設定できなかつたりした場合、kubernetes サービス IP へのリクエストは EKS コントロールプレーン ENI の実際の IP に変換されません。その結果、CNI がクラッシュループに陥り、どのポッドも正しく稼働しなくなります。

前述したように、ポッドは Kubernetes API サーバーとの通信に kubernetes のサービス IP を使用しますが、そのように動作させるには、最初に kube-proxy に iptables ルールを設定する必要があります。

kube-proxy は、どのように API サーバーと通信するのでしょうか？

kube-proxy は、Kubernetes API サーバーの実際の IP アドレス、またはそれらに解決される DNS 名を使用するように設定する必要があります。EKS の場合、デフォルトの kube-proxy が、クラスター作成時に EKS によって作成される Route53 DNS 名を指すように設定されます。この値は、kube-system 名前空間の kube-proxy ConfigMap で確認できます。この ConfigMap の内容は、kube-proxy ポッドに注入される kubeconfig であるため、clusters0.cluster.server フィールドを確認してください。この値は、EKS DescribeCluster API を呼び出した結果にある、EKS クラスターの endpoint フィールドと一致しています。

```
apiVersion: v1
```

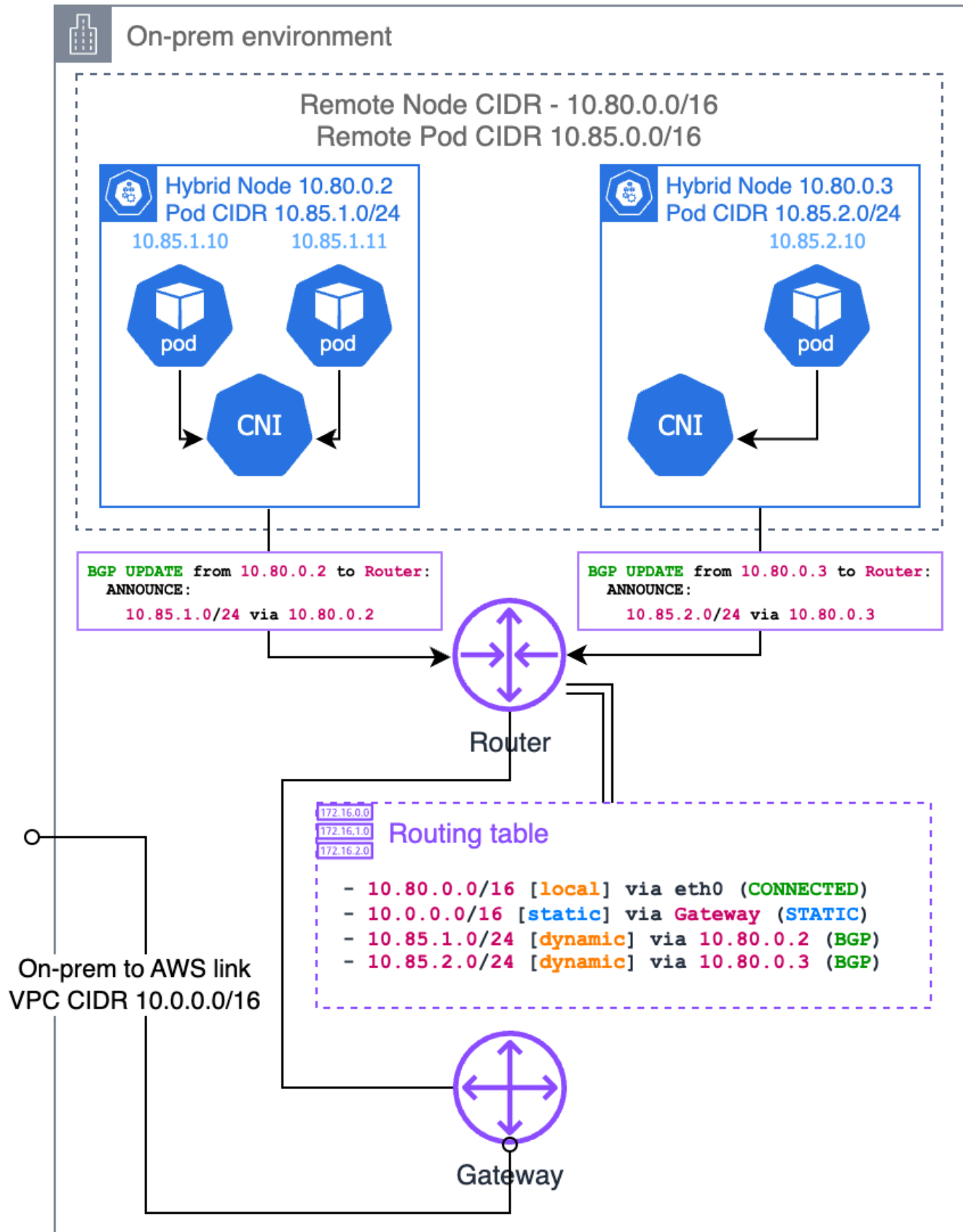
```
data:
 kubeconfig: |-
 kind: Config
 apiVersion: v1
 clusters:
 - cluster:
 certificate-authority: /var/run/secrets/kubernetes.io/serviceaccount/ca.crt
 server: https://xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx.gr7.us-
west-2.eks.amazonaws.com
 name: default
 contexts:
 - context:
 cluster: default
 namespace: default
 user: default
 name: default
 current-context: default
 users:
 - name: default
 user:
 tokenFile: /var/run/secrets/kubernetes.io/serviceaccount/token
kind: ConfigMap
metadata:
 name: kube-proxy
 namespace: kube-system
```

## ルーティング可能なリモートポッド CIDR

この [the section called “ネットワークングの概念”](#) のページでは、ハイブリッドノードでウェブフックを実行したり、クラウドノードで稼働しているポッドがハイブリッドノードで稼働しているポッドと通信したりするための要件について詳しく説明します。ここでは重要な要件があります。特定のポッド IP を担当するノードを、オンプレミスルーターに認識させる必要があることです。これを実現する方法として、ボーダーゲートウェイプロトコル (BGP)、静的ルート、アドレス解決プロトコル (ARP) プロキシなどが挙げられます。以下のセクションでは、これらの方法について説明します。

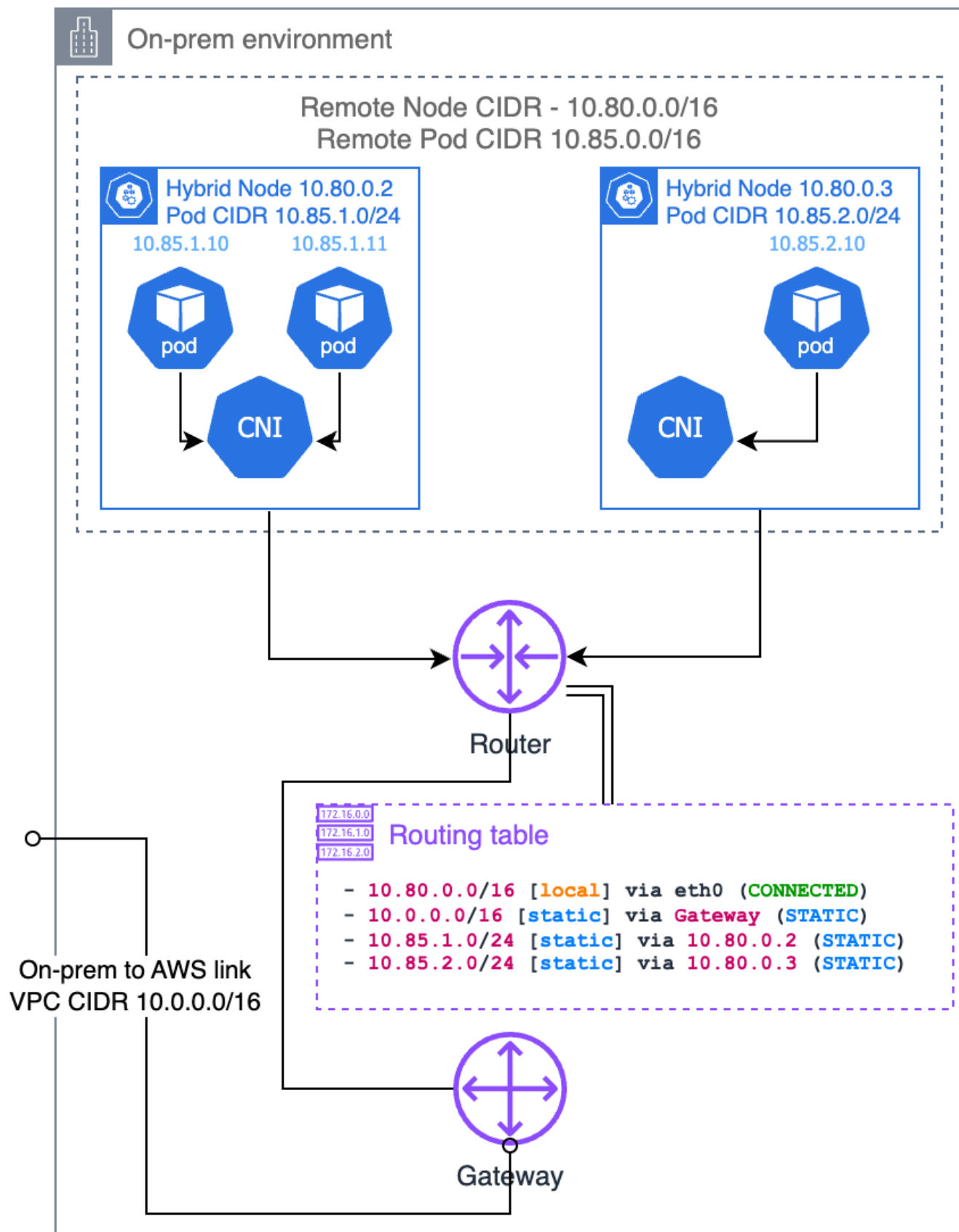
### ボーダーゲートウェイプロトコル (BGP)

CNI でサポートされている場合 (Cilium や Calico など)、CNI の BGP モードを使用して、ノードごとのポッド CIDR へのルートをノードからローカルルーターに伝達できます。CNI の BGP モードを使用する場合、CNI は仮想ルーターとして機能するため、ローカルルーターは、ポッド CIDR が別のサブネットに属し、ノードはそのサブネットへのゲートウェイであると認識します。



## 静的ルート

または、ローカルルーターで静的ルートを設定することもできます。この方法を取ると、オンプレミスポッド CIDR を VPC に非常に簡単にルーティングできますが、最もエラーが発生しやすく、保守も困難になります。ルートが既存のノードと割り当てられたポッド CIDR で、常に最新の状態であることを確認しなければなりません。ノードの数が少なく、インフラストラクチャが静的である場合、これは実行可能なオプションであり、ルーターでの BGP 対応も不要になります。この方法を取る場合は、アドレスを IPAM によって決定せず、各ノードに割り当てるポッド CIDR スライスを使用して CNI を設定することをお勧めします。





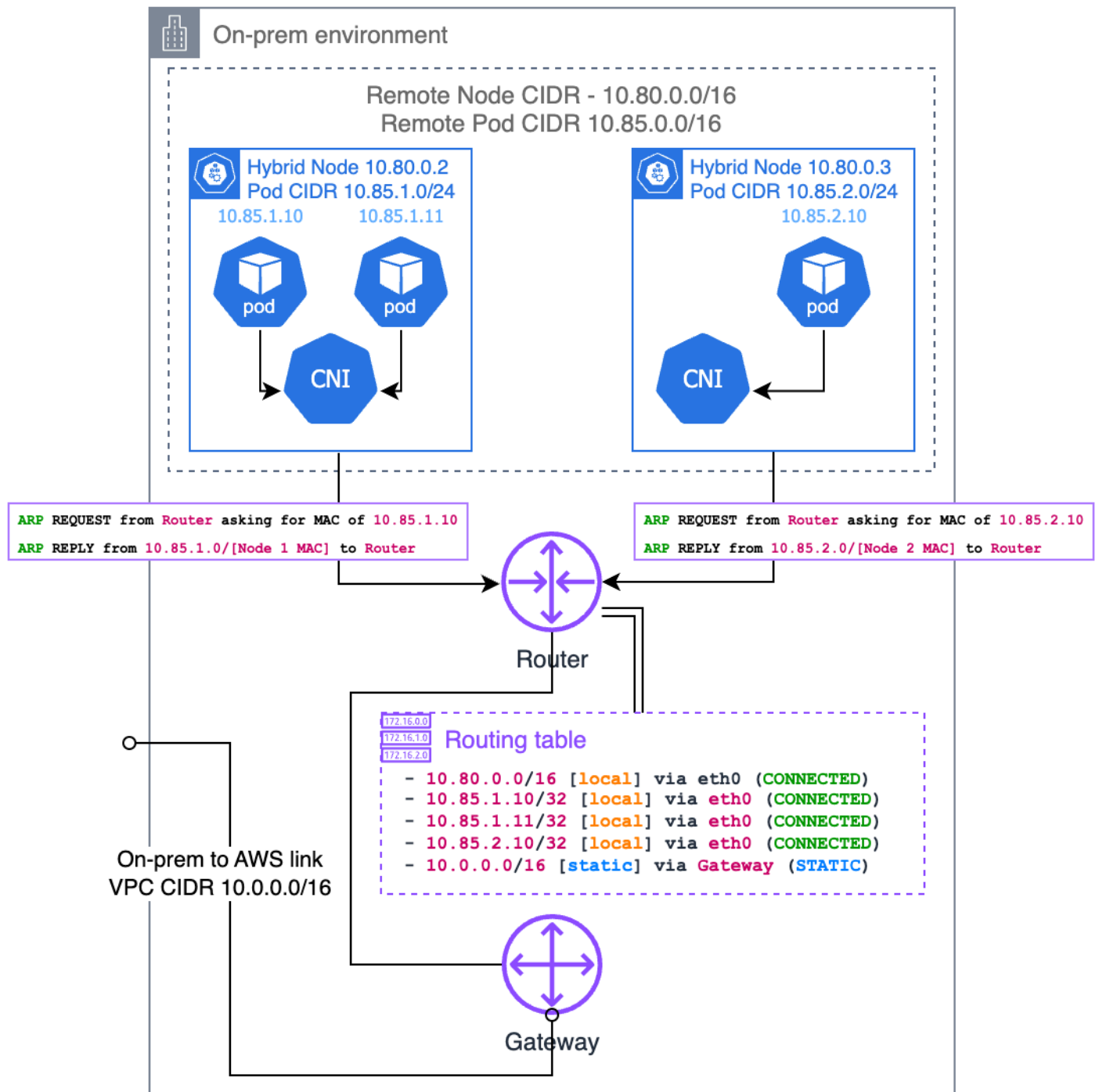
## アドレス解決プロトコル (ARP) プロキシ

ARP プロキシは、オンプレミスのポッド IP をルーティング可能にするもう 1 つのアプローチであり、特に、ハイブリッドノードがローカルルーターと同じレイヤー 2 ネットワークにある場合に便利です。ARP プロキシを有効にすると、ノードは、その IP が別のサブネットに属している場合でも、ホストするポッド IP の ARP リクエストに応答します。

ローカルネットワーク上のデバイスがポッド IP に到達を試みる場合、最初に「この IP を持っているなら応答せよ」という ARP リクエストを送信します。該当するポッドをホストしているハイブリッドノードは、自分の MAC アドレスで応答し、「その IP のトラフィックを処理できます」と伝えます。これによって、ルーター設定を必要とせずに、ローカルネットワーク上のデバイスとポッドの間に直接のパスが作成されます。

これを機能させるには、CNI がプロキシ ARP 機能に対応している必要があります。Cilium は、設定によってプロキシ ARP を有効化できるように設計されています。ここで特に考慮すべき点は、ポッド CIDR が環境内の他のネットワークと重複しないようにすることです。重複すると、ルーティングの競合が生じる可能性があるからです。

このアプローチには、次のような利点があります。\* BGP でルーターを設定する必要はなく、静的ルートを維持する必要もない。\* ルーター設定を制御できない環境でも効果的に機能する。



## ポッド間でのカプセル化

オンプレミス環境の CNI では、通常、カプセル化プロトコルによって、物理ネットワーク上で動作するオーバーレイネットワークが作成されます。この場合、再設定が不要になります。このセクションでは、このカプセル化の仕組みについて説明します。使用している CNI によっては、一部の詳細情報が異なる場合があります。

カプセル化では、元のポッドネットワークパケットを、基盤の物理ネットワークを介してルーティング可能な別のネットワークパケット内にラップします。これによって、ポッドの CIDR のルーティング方法が物理ネットワークで認識されていなくても、同じ CNI を実行しているノード間でポッドが通信できます。

Kubernetes で使用される最も一般的なカプセル化プロトコルは Virtual Extensible LAN (VXLAN) ですが、CNI に応じて、他のプロトコル (Geneve など) も利用できます。

## VXLAN カプセル化

VXLAN では、UDP パケット内にレイヤー 2 イーサネットフレームをカプセル化します。ポッドが別のノードにある別のポッドにトラフィックを送信すると、CNI では、以下が実行されます。

1. ポッド A からのパケットをインターセプトする
2. 元のパケットを VXLAN ヘッダーにラップする
3. ラップしたパケットを、ノードの通常のネットワークスタックを介して宛先ノードに送信する
4. 宛先ノードの CNI が、パケットをラップ解除し、ポッド B に配信する

VXLAN カプセル化中のパケット構造の変化を次に示します。

### 元のポッド間パケット

```
+-----+-----+-----+-----+
Ethernet Header	IP Header	TCP/UDP	Payload
Src: Pod A MAC	Src: Pod A IP	Src Port	
Dst: Pod B MAC	Dst: Pod B IP	Dst Port	
+-----+-----+-----+-----+
```

### VXLAN カプセル化後

```
+-----+-----+-----+-----+
+-----+
| Outer Ethernet | Outer IP | Outer UDP | VXLAN | Original Pod-to-Pod
| | | | |
| Src: Node A MAC | Src: Node A | Src: Random | VNI: xx | Packet (unchanged
| | | | |
| Dst: Node B MAC | Dst: Node B | Dst: 4789 | | from above)
| | | | |
+-----+-----+-----+-----+
+-----+
```

この VXLAN ネットワーク識別子 (VNI) によって、異なるオーバーレイネットワークが区別されます。

## ポッド通信のシナリオ

### 同じハイブリッドノード上のポッド

同じハイブリッドノード上のポッドが通信する場合、通常はカプセル化は必要ありません。CNI では、ローカルルートが設定されます。これにより、ノードの内部仮想インターフェイスを介してポッド間のトラフィックを転送します。

```
Pod A -> veth0 -> node's bridge/routing table -> veth1 -> Pod B
```

パケットはノードから出ることはなく、カプセル化を必要としません。

### 異なるハイブリッドノード上のポッド

異なるハイブリッドノード上のポッド間で通信するには、カプセル化が必要です。

```
Pod A -> CNI -> [VXLAN encapsulation] -> Node A network -> router or gateway -> Node B network -> [VXLAN decapsulation] -> CNI -> Pod B
```

これにより、物理ネットワークでポッド IP のルーティングが認識されていなくても、ポッドトラフィックは、物理ネットワークインフラストラクチャを通過できます。

## ハイブリッドノード向けネットワークトラフィックフロー

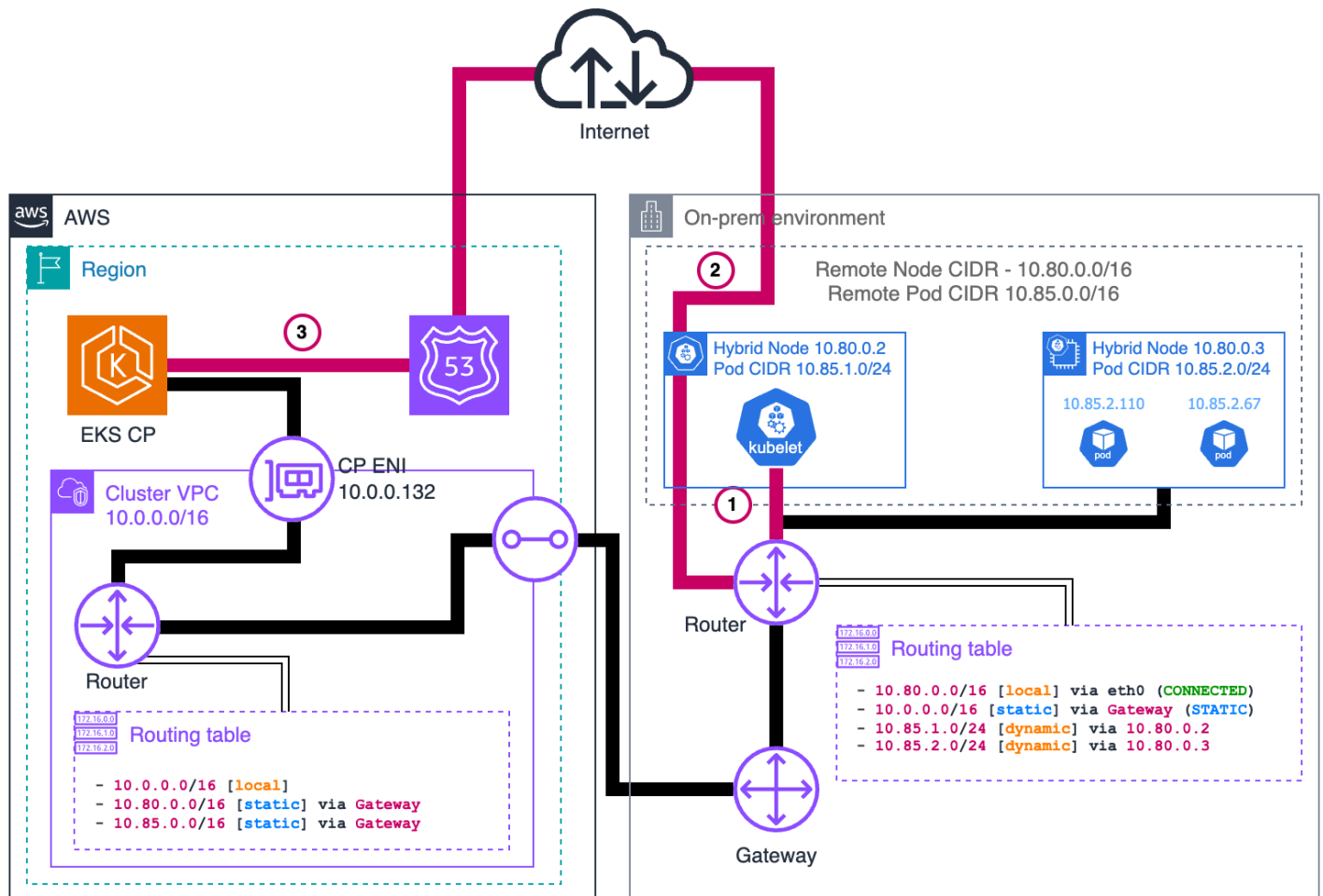
このページでは、EKS ハイブリッドノード向けネットワークトラフィックフローについて、いくつかのトラフィックタイプ別にエンドツーエンドでネットワークパスの図を示しながら詳しく説明します。

以下のトラフィックフローを取り上げます。

- [the section called “ハイブリッドノード kubelet から EKS コントロールプレーンまでのフロー”](#)
- [the section called “EKS コントロールプレーンからハイブリッドノード \(kubelet サーバー\) までのフロー”](#)
- [the section called “ハイブリッドノードで実行されているポッドから EKS コントロールプレーンまでのフロー”](#)

- [the section called “EKS コントロールプレーンからハイブリッドノードで実行されているポッドまでのフロー \(ウェブフック\)”](#)
- [the section called “ハイブリッドノードで実行されているポッド間”](#)
- [the section called “クラウドノード上のポッドからハイブリッドノード上のポッドまでのフロー \(東西トラフィック\)”](#)

ハイブリッドノード **kubelet** から EKS コントロールプレーンまでのフロー



## リクエスト

### 1 **kubelet**. がリクエストを開始する

ハイブリッドノード上の **kubelet** が EKS コントロールプレーンと通信する必要がある場合 (例えば、ノードステータスを報告するため、あるいはポッドの仕様を取得するため) には、ノード登録時に提供された **kubeconfig** ファイルが使用されます。この **kubeconfig** には、ダイレクト IP アドレスではなく、API サーバーエンドポイント URL (Route53 DNS 名) が記載されています。

kubelet がエンドポイントに対して DNS ルックアップを実行します (例: <https://xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx.gr7.us-west-2.eks.amazonaws.com>)。パブリックアクセスクラスターでは、これは AWS で実行中の EKS サービスに属するパブリック IP アドレス (例: 54.239.118.52) に解決されます。次に kubelet は、このエンドポイントへの安全な HTTPS リクエストを作成します。初期パケットは以下のようになります。

```
+-----+-----+-----+
IP Header	TCP Header	Payload
Src: 10.80.0.2	Src: 52390 (random)	
Dst: 54.239.118.52	Dst: 443	
+-----+-----+-----+
```

## 2. ローカルルーターのルーティング

宛先 IP はパブリック IP アドレスであり、ローカルネットワークの一部ではないため、kubelet はこのパケットをそのデフォルトゲートウェイ (ローカルオンプレミスルーター) に送信します。ルーターは、宛先 IP を確認して、パブリック IP アドレスであると判断します。

パブリックトラフィックの場合、ルーターは通常、パケットをインターネットゲートウェイ、つまりインターネットへのアウトバウンドトラフィックを処理するボーダールーターに転送します。図では、これを省略しています。実際の転送は、オンプレミスネットワークがどのように設定されているかによって異なります。パケットは、オンプレミスのネットワークインフラストラクチャを通過していき、最終的にインターネットサービスプロバイダーのネットワークに到達します。

## 3. EKS コントロールプレーンまでの配信

パケットは、パブリックインターネットとトランジットネットワークを横断していき、AWS のネットワークに到達します。AWS のネットワークは、パケットを適切なリージョンの EKS サービスエンドポイントにルーティングします。EKS サービスに到達したパケットは、クラスターの実際の EKS コントロールプレーンに転送されます。

これはパブリックインターネットを通るルーティングであり、他のトラフィックフローで見られるようなプライベート VPC をルーティングされていくパスとは異なります。その主な違いは、パブリックアクセスモードを使用した場合、ポッドからではなくオンプレミスの kubelet から EKS コントロールプレーンへのトラフィックが VPC を経由せず、代わりにグローバルインターネットインフラストラクチャを使用することです。

## レスポンス

EKS コントロールプレーンは、kubelet リクエストを処理してレスポンスを返します。

### 3. EKS コントロールプレーンがレスポンスを送信する

EKS コントロールプレーンはレスポンスパケットを作成します。このパケットには、ソースとしてパブリック IP、送信先としてハイブリッドノードの IP が含まれています。

```
+-----+-----+-----+
IP Header	TCP Header	Payload
Src: 54.239.118.52	Src: 443	
Dst: 10.80.0.2	Dst: 52390	
+-----+-----+-----+
```

## 2. インターネットでのルーティング

レスポンスパケットは、インターネットサービスプロバイダーが決定したルーティングパスをたどってインターネット経由で戻り、オンプレミスネットワークのエッジルーターに到達します。

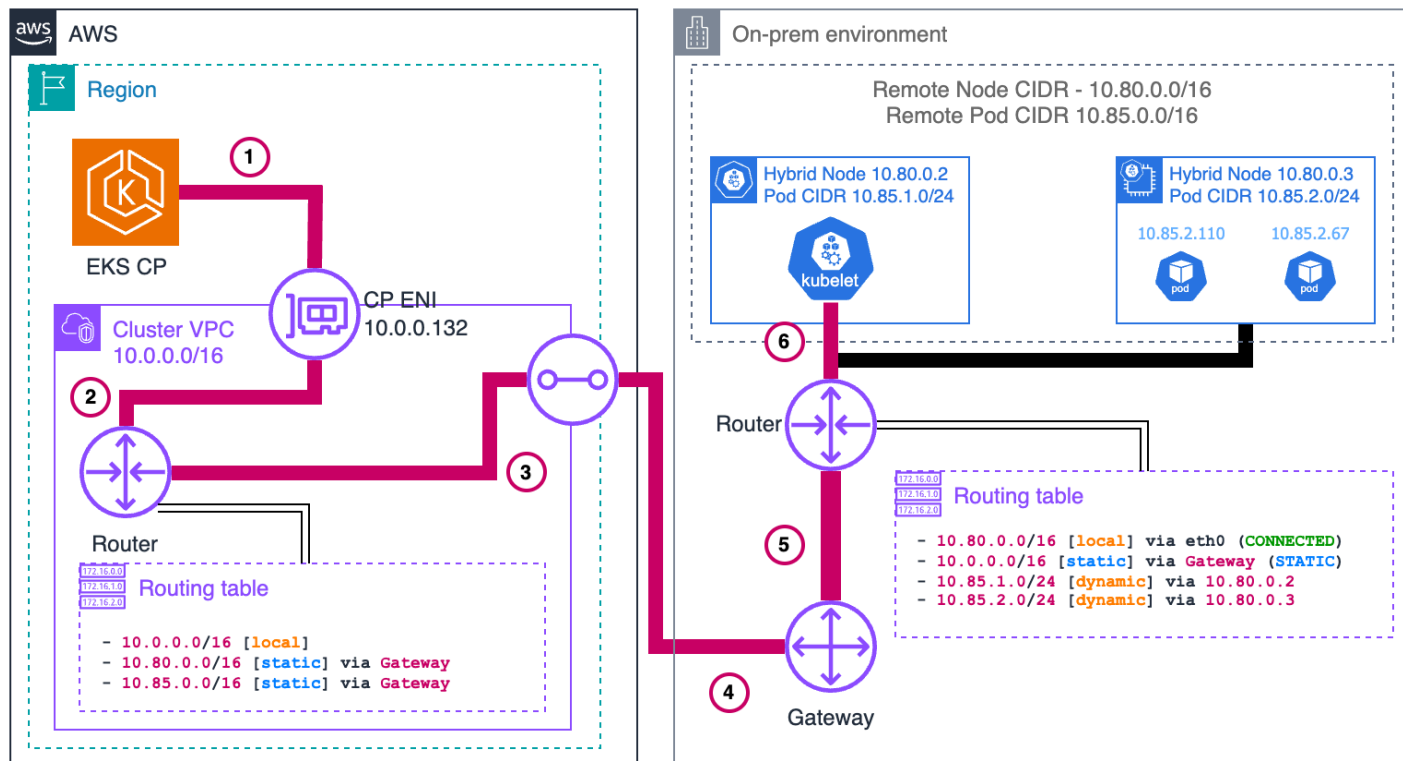
### 1. ローカルでの配信

オンプレミスのルーターは、パケットを受信し、宛先 IP (10.80.0.2) がローカルネットワークに属していると認識します。次に、受信したパケットをローカルネットワークインフラストラクチャ経由で転送します。パケットがターゲットのハイブリッドノードに到達すると、kubelet がレスポンスを受信して処理します。

ハイブリッドノード **kube-proxy** から EKS コントロールプレーンまでのフロー

クラスターのパブリックエンドポイントアクセスを有効にすると、戻りのトラフィックはパブリックインターネットを使用します。このトラフィックは、ハイブリッドノードの kube-proxy から EKS コントロールプレーンへと進み、kubelet から EKS コントロールプレーンへのトラフィックと同じパスを通ります。

## EKS コントロールプレーンからハイブリッドノード (kubelet サーバー) までのフロー



## リクエスト

## 1. EKS Kubernetes API サーバーがリクエストを開始する

EKS Kubernetes API サーバーは、ノードオブジェクトのステータスからノードの IP アドレス (10.80.0.2) を取得します。次に、送信先 IP が設定済みのリモートノード CIDR (10.80.0.0/16) に属しているため、このリクエストを VPC 内の ENI 経由でルーティングします。初期パケットは以下のようになります。

```

+-----+-----+-----+
IP Header	TCP Header	Payload
Src: 10.0.0.132	Src: 67493 (random)	
Dst: 10.80.0.2	Dst: 10250	
+-----+-----+-----+

```

## 2. VPC ネットワークでの処理

パケットは、ENI を出て VPC ネットワーキングレイヤーに入り、さらにルーティングするためにサブネットのゲートウェイ宛てに送信されます。

## 3. VPC ルートテーブルのルックアップ



EKS コントロールプレーン ENI が含まれているサブネット用の VPC ルートテーブルには、リモートノード CIDR に固有のルート (図の 2 番目のルート) が登録されています。このルーティングルールに基づいて、パケットが VPC-to-onprem ゲートウェイに向けて送信されます。

#### 4. 境界を越えて移動する

ゲートウェイは、確立済みの接続 (Direct Connect や VPN など) 経由でクラウド境界を越えてオンプレミスネットワークにパケットを転送します。

#### 5. オンプレミスネットワークでの受信

パケットは、ハイブリッドノードが配置されているサブネット宛てのトラフィックを処理するローカルオンプレミスルーターに到達します。

#### 6. 最後の配信

ローカルルーターは、宛先 IP (10.80.0.2) アドレスが自身に直接接続されているネットワークに属しているかどうかを識別して、ターゲットのハイブリッドノードに直接パケットを転送し、そこで kubelet がリクエストを受信して処理します。

#### レスポンス

ハイブリッドノードの kubelet は、リクエストを処理してレスポンスを返します。レスポンスは、同じパスを逆方向にたどります。

```
+-----+-----+-----+
IP Header	TCP Header	Payload
Src: 10.80.0.2	Src: 10250	
Dst: 10.0.0.132	Dst: 67493	
+-----+-----+-----+
```

#### 6. kubelet レスポンスを送信する

ハイブリッドノード (10.80.0.2) 上の kubelet は、送信元の IP を宛先としてレスポンスパケットを作成します。その宛先はローカルネットワークに属していないため、パケットはホストのデフォルトゲートウェイ、つまりローカルルーターに送信されます。

#### 5. ローカルルーターのルーティング

ルーターは、宛先 IP (10.0.0.132) が 10.0.0.0/16 に属していると判断します。ここには、AWS に接続しているゲートウェイを指すルートがあります。

## 4. 境界を越えて戻る

パケットは、同じオンプレミスを通して VPC 接続 (Direct Connect や VPN など) まで戻り、クラウド境界を逆方向に横断します。

## 3. VPC でのルーティング

パケットが VPC に到達すると、ルートテーブルは宛先 IP が VPC CIDR に属しているかどうかを識別します。パケットは VPC 内にルーティングされます。

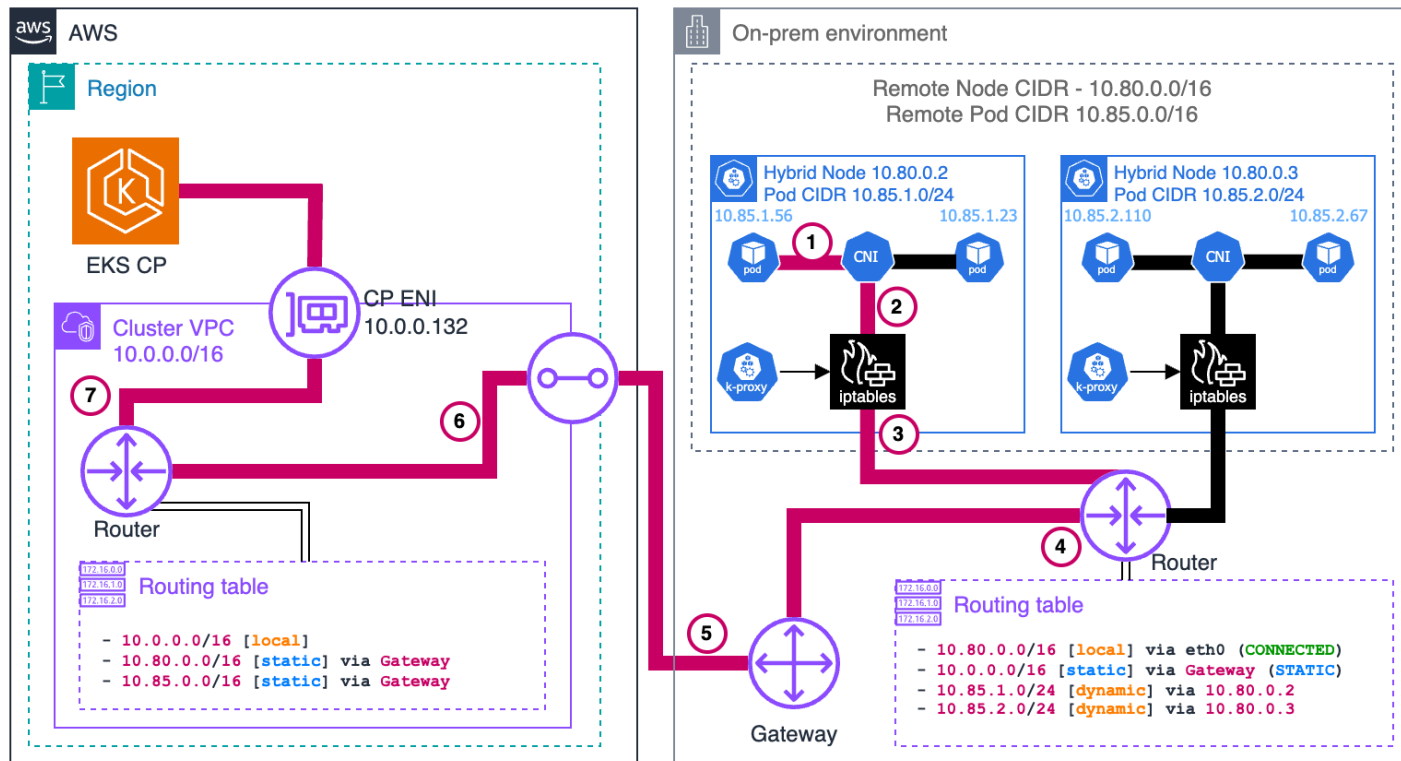
## 2. VPC ネットワークでの配信

VPC ネットワーキングレイヤーは、EKS コントロールプレーン ENI (10.0.0.132) を備えたサブネットにパケットを転送します。

## 1. ENI での受信

パケットは、Kubernetes API サーバーにアタッチされている EKS コントロールプレーン ENI に到達します。これでラウンドトリップは完了です。

ハイブリッドノードで実行されているポッドから EKS コントロールプレーンまでのフロー



## CNI NAT なし

### リクエスト

ポッドは一般に、kubernetes サービスを介して Kubernetes API サーバーと対話します。サービス IP がクラスターのサービス CIDR の最初の IP となります。この規約により、CoreDNS が使用可能になる前に稼働する必要があるポッドが API サーバー (例えば、CNI) に到達できるようになります。リクエストは、サービス IP を宛先としてポッドを出ていきます。例えば、サービス CIDR が 172.16.0.0/16 である場合、サービス IP は 172.16.0.1 になります。

#### 1. ポッドがリクエストを開始する

ポッドは、ランダムな送信元ポートから API サーバーポート (443) 上の kubernetes サービス IP (172.16.0.1) にリクエストを送信します。パケットは以下のようになります。

```
+-----+-----+-----+
IP Header	TCP Header	Payload
Src: 10.85.1.56	Src: 67493 (random)	
Dst: 172.16.0.1	Dst: 443	
+-----+-----+-----+
```

#### 2. CNI での処理

CNI は、宛先 IP が自身の管理するポッド CIDR に属していないことを検出します。送信 NAT が無効になっているため、CNI はパケットを変更せずにホストネットワークスタックに渡します。

#### 3. ノードネットワークでの処理

パケットはノードのネットワークスタックに入り、そこで netfilter フックが kube-proxy によって設定された iptables ルールをトリガーします。以下の順序でルールがいくつか適用されます。

1. パケットはまず KUBE-SERVICES チェーンをヒットします。ここには、各サービスの ClusterIP とポートに対応するルールが含まれています。
2. その対応するルールは、kubernetes サービスの KUBE-SVC-XXX チェーンにジャンプします (172.16.0.1:443 宛てのパケット)。ここには、ロードバランシングルールが含まれています。
3. ロードバランシングルールは、コントロールプレーン ENI IP (10.0.0.132 または 10.0.1.23) 用に KUBE-SEP-XXX チェーンのいずれかをランダムに選択します。
4. 選択した KUBE-SEP-XXX チェーンには実際のルールがあり、それにより宛先 IP はサービス IP から選択されている IP に変更されます。これは、Destination Network Address Translation (宛先ネットワークアドレス変換、DNAT) と呼ばれます。

選択されている EKS コントロールプレーン ENI の IP を 10.0.0.132 とした場合、これらのルールが適用されると、パケットは以下ようになります。

```
+-----+-----+-----+
IP Header	TCP Header	Payload
Src: 10.85.1.56	Src: 67493 (random)	
Dst: 10.0.0.132	Dst: 443	
+-----+-----+-----+
```

宛先 IP がローカルネットワーク内にないため、ノードはパケットをデフォルトゲートウェイに転送します。

#### 4. ローカルルーターのルーティング

ローカルルーターは、宛先 IP (10.0.0.132) が VPC CIDR (10.0.0.0/16) に属していると判断して、AWS に接続しているゲートウェイに転送します。

#### 5. 境界を越えて移動する

パケットは、確立済みの接続 (Direct Connect や VPN など) 経由でクラウド境界を越えて VPC に移動します。

#### 6. VPC ネットワークでの配信

VPC ネットワーキングレイヤーは、EKS コントロールプレーン ENI (10.0.0.132) が存在する適切なサブネットにパケットをルーティングします。

#### 7. ENI での受信

パケットは、Kubernetes API サーバーにアタッチされている EKS コントロールプレーン ENI に到達します。

#### レスポンス

EKS コントロールプレーンは、リクエストを処理してポッドにレスポンスを返します。

#### 7. API サーバーがレスポンスを送信する

EKS Kubernetes API サーバーは、送信元の IP を宛先としてレスポンスパケットを作成します。パケットは以下ようになります。

```
+-----+-----+-----+
| IP Header | TCP Header | Payload |
+-----+-----+-----+
```

|                     |            |  |  |
|---------------------|------------|--|--|
| Src: 10.0.0.132     | Src: 443   |  |  |
| Dst: 10.85.1.56     | Dst: 67493 |  |  |
| +-----+-----+-----+ |            |  |  |

宛先 IP が設定済みのリモートポッド CIDR (10.85.0.0/16) に属しているため、パケットはサブネットのルーターを次のホップとして VPC 内の ENI 経由で送信されます。

## 6. VPC でのルーティング

VPC ルートテーブルにはリモートポッド CIDR (10.85.0.0/16) のエントリがあるため、このトラフィックは VPC-to-onprem ゲートウェイ宛てに送信されます。

## 5. 境界を越えて移動する

ゲートウェイは、確立済みの接続 (Direct Connect や VPN など) 経由でクラウド境界を越えてオンプレミスネットワークにパケットを転送します。

## 4. オンプレミスネットワークでの受信

パケットは、ローカルオンプレミスルーターに到達します。

## 3. ノードへの配信

ルーターのテーブルには 10.80.0.2 を次のホップとした 10.85.1.0/24 のエントリがあるため、パケットはノードに配信されます。

## 2. ノードネットワークでの処理

パケットがノードのネットワークスタックによって処理されると、conntrack (netfilter の一部) はポッドが最初に確立した接続とパケットを照合します。DNAT が最初に適用されたため、conntrack はソース IP を EKS コントロールプレーン ENI の IP から kubernetes サービス IP に書き換えることで DNAT を元に戻します。

|                     |            |         |  |
|---------------------|------------|---------|--|
| +-----+-----+-----+ |            |         |  |
| IP Header           | TCP Header | Payload |  |
| Src: 172.16.0.1     | Src: 443   |         |  |
| Dst: 10.85.1.56     | Dst: 67493 |         |  |
| +-----+-----+-----+ |            |         |  |

## 1. CNI での処理

CNI は、宛先 IP が自身のネットワーク内のポッドに属しているかどうかを識別して、適切なポッドネットワーク名前空間にパケットを配信します。

このフローは、なぜリモートポッド CIDR を VPC から各ポッドをホストする特定のノードに至るまで正しくルーティングできなければならないかを示しています。戻りパス全体は、クラウドとオンプレミスの両方のネットワーク全体にわたってポッド IP が適切にルーティングされるかどうかによって異なります。

## CNI NAT あり

このフローは、CNI NAT がないものと非常によく似ていますが、重要な違いが 1 つあります。CNI は、パケットに送信元 NAT (SNAT) を適用してから、ノードのネットワークスタックにパケットを送信します。これにより、パケットの送信元 IP がノードの IP に変更されるため、追加でルーティングを設定することなく、ノードにパケットをルーティングできます。

## リクエスト

### 1. ポッドがリクエストを開始する

ポッドは、ランダムな送信元ポートから EKS Kubernetes API サーバーポート (443) 上の kubernetes サービス IP (172.16.0.1) にリクエストを送信します。パケットは以下のようになります。

```
+-----+-----+-----+
IP Header	TCP Header	Payload
Src: 10.85.1.56	Src: 67493 (random)	
Dst: 172.16.0.1	Dst: 443	
+-----+-----+-----+
```

### 2. CNI での処理

CNI は、宛先 IP が自身の管理するポッド CIDR に属していないことを検出します。送信 NAT が有効になっているため、CNI は SNAT をパケットに適用して送信元 IP をノードの IP に変更してから、パケットをノードのネットワークスタックに渡します。

```
+-----+-----+-----+
IP Header	TCP Header	Payload
Src: 10.80.0.2	Src: 67493 (random)	
Dst: 172.16.0.1	Dst: 443	
+-----+-----+-----+
```

注: この例ではわかりやすくするために CNI と iptables をそれぞれ別のブロックに分けていますが、実際には iptables を使用して NAT を適用する CNI もあります。

### 3. ノードネットワークでの処理

ここでは、kube-proxy によって設定された iptables ルールが前の例と同じように動作して、パケットを EKS コントロールプレーン ENI のいずれかに負荷分散します。これでパケットは以下のようになります。

```
+-----+-----+-----+
IP Header	TCP Header	Payload
Src: 10.80.0.2	Src: 67493 (random)	
Dst: 10.0.0.132	Dst: 443	
+-----+-----+-----+
```

宛先 IP がローカルネットワーク内にないため、ノードはパケットをデフォルトゲートウェイに転送します。

### 4. ローカルルーターのルーティング

ローカルルーターは、宛先 IP (10.0.0.132) が VPC CIDR (10.0.0.0/16) に属していると判断して、AWS に接続しているゲートウェイに転送します。

### 5. 境界を越えて移動する

パケットは、確立済みの接続 (Direct Connect や VPN など) 経由でクラウド境界を越えて VPC に移動します。

### 6. VPC ネットワークでの配信

VPC ネットワーキングレイヤーは、EKS コントロールプレーン ENI (10.0.0.132) が存在する適切なサブネットにパケットをルーティングします。

### 7. ENI での受信

パケットは、Kubernetes API サーバーにアタッチされている EKS コントロールプレーン ENI に到達します。

### レスポンス

EKS コントロールプレーンは、リクエストを処理してポッドにレスポンスを返します。

### 7. API サーバーがレスポンスを送信する

EKS Kubernetes API サーバーは、送信元の IP を宛先としてレスポンスパケットを作成します。パケットは以下のようになります。

```

+-----+-----+-----+
IP Header	TCP Header	Payload
Src: 10.0.0.132	Src: 443	
Dst: 10.80.0.2	Dst: 67493	
+-----+-----+-----+

```

宛先 IP が設定済みのリモートノード CIDR (10.80.0.0/16) に属しているため、パケットはサブネットのルーターを次のホップとして VPC 内の ENI 経由で送信されます。

## 6. VPC でのルーティング

VPC ルートテーブルにはリモートノード CIDR (10.80.0.0/16) のエントリがあるため、このトラフィックは VPC-to-onprem ゲートウェイ宛てに送信されます。

## 5. 境界を越えて移動する

ゲートウェイは、確立済みの接続 (Direct Connect や VPN など) 経由でクラウド境界を越えてオンプレミスネットワークにパケットを転送します。

## 4. オンプレミスネットワークでの受信

パケットは、ローカルオンプレミスルーターに到達します。

## 3. ノードへの配信

ローカルルーターは、宛先 IP (10.80.0.2) アドレスが自身に直接接続されているネットワークに属しているかどうかを識別して、ターゲットのハイブリッドノードに直接パケットを転送します。

## 2. ノードネットワークでの処理

パケットがノードのネットワークスタックによって処理されると、conntrack (netfilter の一部) はパケットをポッドが最初に確立した接続と照合します。元々 DNAT が適用されていたため、送信元 IP を EKS コントロールプレーン ENI の IP から kubernetes サービス IP に書き換えることで元に戻します。

```

+-----+-----+-----+
IP Header	TCP Header	Payload
Src: 172.16.0.1	Src: 443	
Dst: 10.80.0.2	Dst: 67493	
+-----+-----+-----+

```

## 1. CNI での処理



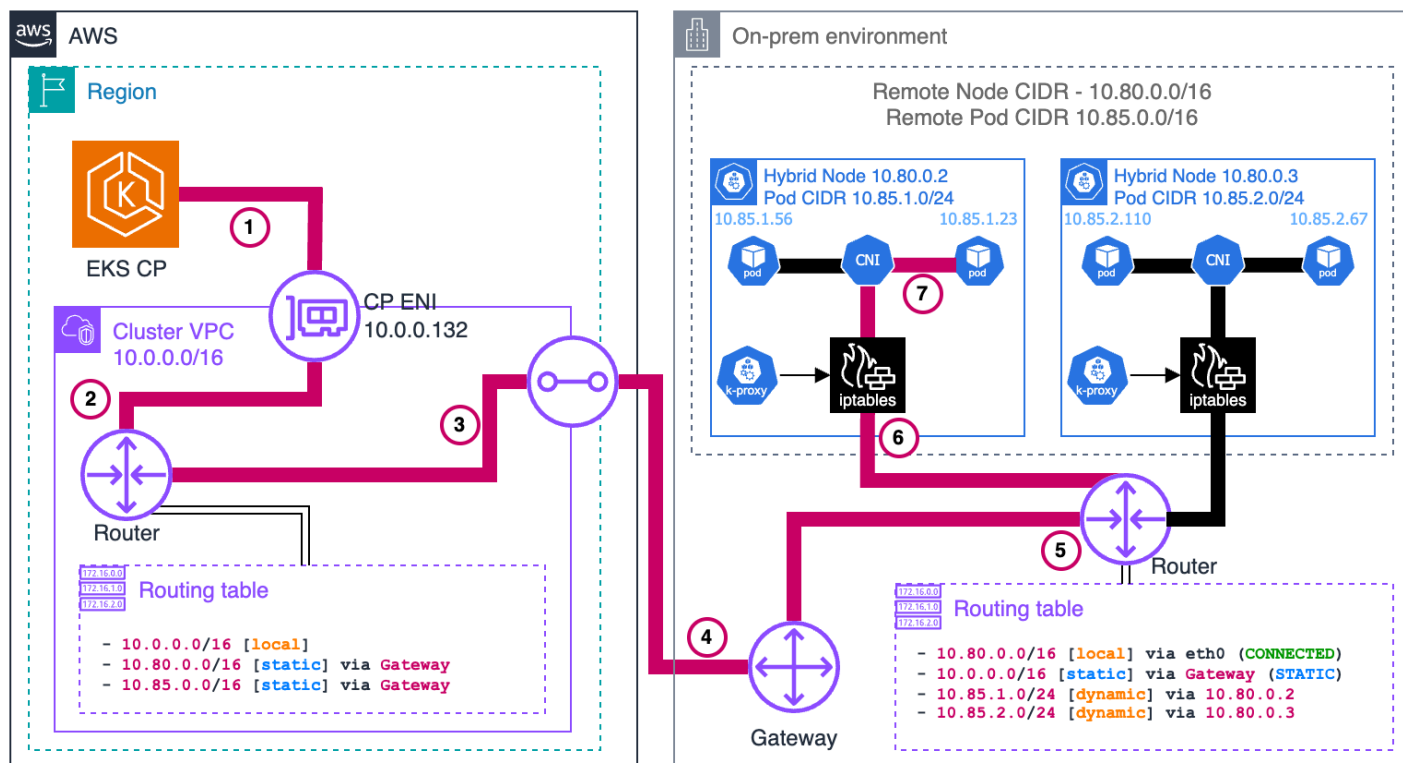
CNI は、このパケットが以前に SNAT が適用された接続に属しているかどうかを識別します。これにより、SNAT が逆方向に適用されるため、宛先 IP がポッドの IP に戻ります。

```
+-----+-----+-----+
IP Header	TCP Header	Payload
Src: 172.16.0.1	Src: 443	
Dst: 10.85.1.56	Dst: 67493	
+-----+-----+-----+
```

CNI は、宛先 IP が自身のネットワーク内のポッドに属していることを検出して、適切なポッドネットワーク名前空間にパケットを配信します。

このフローは、CNI NAT の適用により、追加でポッド CIDR のルーティングを行うことなくパケットをノードにルーティングできるようにすることで、設定をどのように簡素化できるかを示しています。

EKS コントロールプレーンからハイブリッドノードで実行されているポッドまでのフロー (ウェブフック)



このトラフィックパターンはウェブフックで最もよく見られるものであり、ウェブフックではハイブリッドノード上のポッドで実行されているウェブフックサーバーへの接続を EKS コントロールプ

レーンが直接開始する必要があります。例えば、アドミッションウェブフックを検証して変更するといった場合、このウェブフックはリソースの検証と変更のプロセス中に API サーバーによって呼び出されます。

## リクエスト

### 1. EKS Kubernetes API サーバーがリクエストを開始する

クラスターに設定されたウェブフックが関連する API オペレーションによってトリガーされた場合、EKS Kubernetes API サーバーはそのウェブフックサーバーポッドに直接接続する必要があります。API サーバーはまず、そのウェブフックに関連付けられているサービスリソースまたはエンドポイントリソースでポッドの IP アドレスを検索します。

IP が 10.85.1.23 のハイブリッドノード上でウェブフックポッドが実行されている場合、EKS Kubernetes API サーバーはウェブフックエンドポイントへの HTTPS リクエストを作成します。10.85.1.23 という宛先 IP は設定済みのリモートポッド CIDR (10.85.0.0/16) に属しているため、初期パケットは VPC の EKS コントロールプレーン ENI を介して送信されます。パケットは以下のようになります。

```
+-----+-----+-----+
IP Header	TCP Header	Payload
Src: 10.0.0.132	Src: 41892 (random)	
Dst: 10.85.1.23	Dst: 8443	
+-----+-----+-----+
```

### 2. VPC ネットワークでの処理

パケットは、EKS コントロールプレーン ENI を出て、サブネットのルーターを次のホップとする VPC ネットワーキングレイヤーに入ります。

### 3. VPC ルートテーブルのルックアップ

EKS コントロールプレーン ENI が含まれているサブネット用の VPC ルートテーブルには、リモートポッド CIDR (10.85.0.0/16) に固有のルートが登録されています。このルーティングルールにより、パケットは VPC-to-onprem ゲートウェイ (例えば、Direct Connect や VPN 接続の場合は仮想プライベートゲートウェイ) 宛てに送信されます。

| Destination  | Target                         |
|--------------|--------------------------------|
| 10.0.0.0/16  | local                          |
| 10.85.0.0/16 | vgw-id (VPC-to-onprem gateway) |

## 4. 境界を越えて移動する

ゲートウェイは、確立済みの接続 (Direct Connect や VPN など) 経由でクラウド境界を越えてオンプレミスネットワークにパケットを転送します。パケットがこの接続を通過するとき、パケットの送信元 IP アドレスと宛先 IP アドレスは元のまま維持されます。

## 5. オンプレミスネットワークでの受信

パケットは、ローカルオンプレミスルーターに到達します。ルーターは、自身のルーティングテーブルに問い合わせ、10.85.1.23 アドレスに到達する方法を決定します。そのためにオンプレミスネットワークには、適切なハイブリッドノード宛てにトラフィックを送信するポッド CIDR へのルートが必要です。

この例の場合、ルーターのルートテーブルには IP が 10.80.0.2 のハイブリッドノード経由で 10.85.1.0/24 サブネットに到達できることを示すエントリが登録されています。

| Destination  | Next Hop  |
|--------------|-----------|
| 10.85.1.0/24 | 10.80.0.2 |

## 6. ノードへの配信

ルーターは、ルーティングテーブルのエントリに基づいて、パケットをハイブリッドノード (10.80.0.2) に転送します。ノードに到達したとき、パケットは EKS Kubernetes API サーバーから送信されたときと同じままで、宛先 IP は依然としてポッドの IP です。

## 7. CNI での処理

ノードのネットワークスタックは、パケットを受信し、宛先 IP がノード自体の IP でないことを確認して、さらに処理するために CNI に渡します。CNI は、宛先 IP がこのノードでローカルに実行されているポッドに属しているかどうかを識別し、適切な仮想インターフェイス経由で適切なポッドにパケットを転送します。

```
Original packet -> node routing -> CNI -> Pod's network namespace
```

ポッド内のウェブフックサーバーは、リクエストを受信して処理します。

## レスポンス

ウェブフックポッドは、リクエストを処理してレスポンスを返します。レスポンスは、同じパスを逆方向にたどります。

## 7. ポッドがレスポンスを送信する

ウェブフックポッドは、自身の IP を送信元とし、元のリクエスト (EKS コントロールプレーン ENI) を宛先として、レスポンスパケットを作成します。

```
+-----+-----+-----+
IP Header	TCP Header	Payload
Src: 10.85.1.23	Src: 8443	
Dst: 10.0.0.132	Dst: 41892	
+-----+-----+-----+
```

CNI は、このパケットが外部ネットワーク (ローカルポッドではない) に送信され、パケットが、元のソース IP を保持したままノードのネットワークスタックに渡されたことを確認します。

## 6. ノードネットワークでの処理

ノードは、宛先 IP (10.0.0.132) がローカルネットワーク内ないと判断して、自身のデフォルトゲートウェイ (ローカルルーター) にパケットを転送します。

## 5. ローカルルーターのルーティング

ローカルルーターは、自身のルーティングテーブルに問い合わせ、宛先 IP (10.0.0.132) が VPC CIDR (10.0.0.0/16) に属していると判断します。次に、AWS に接続しているゲートウェイにパケットを転送します。

## 4. 境界を越えて移動する

パケットは、同じオンプレミスを通して VPC 接続まで戻り、クラウド境界を逆方向に横断します。

## 3. VPC でのルーティング

パケットが VPC に到達すると、ルートテーブルは宛先 IP が VPC 内のサブネットに属しているかどうかを識別します。それに応じて、パケットは VPC 内にルーティングされます。

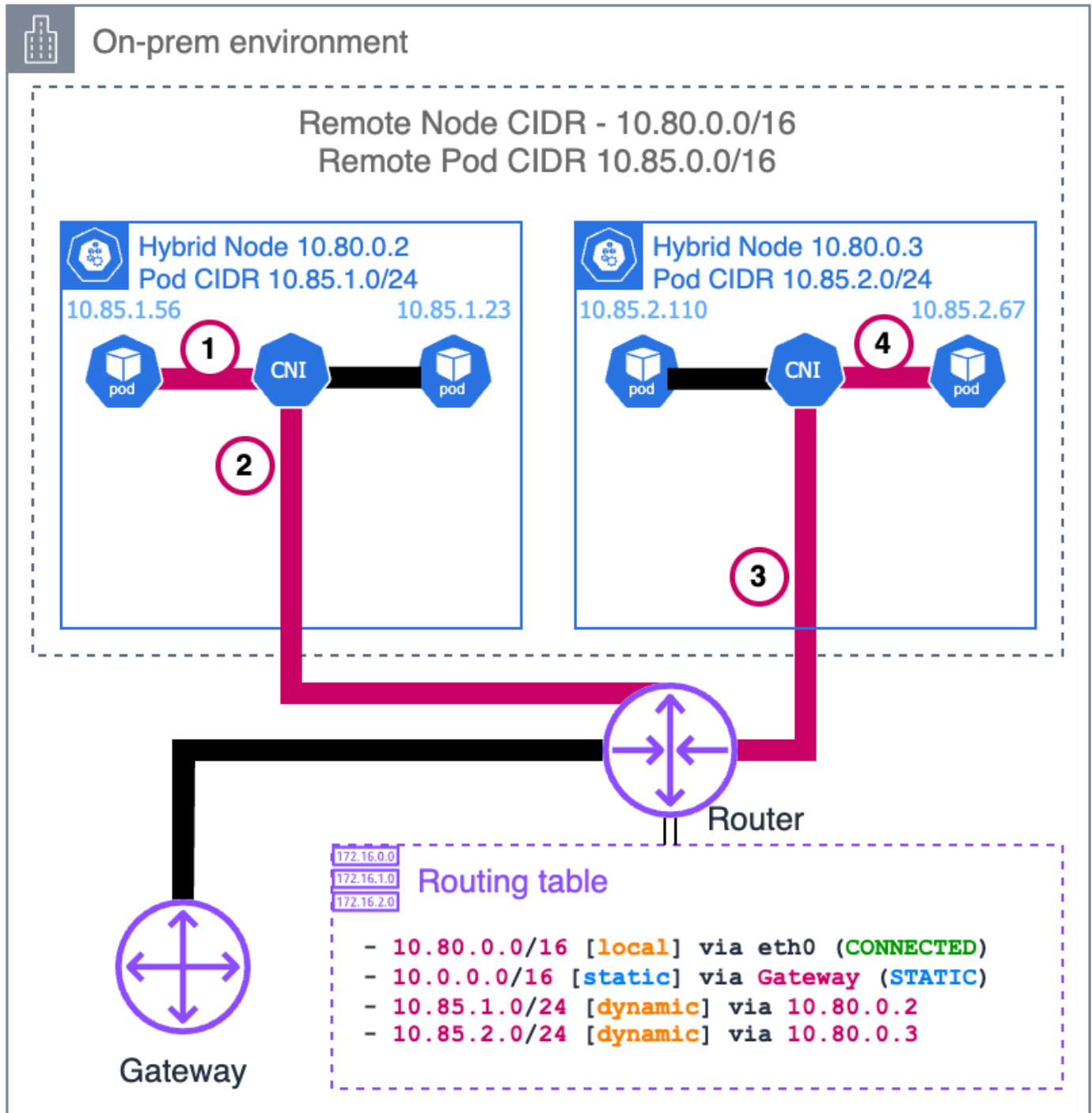
## 2. と 1. EKS コントロールプレーン ENI での受信

パケットは、Kubernetes API サーバーにアタッチされている ENI に到達します。これでラウンドトリップは完了です。API サーバーは、ウェブフックレスポンスを受信し、このレスポンスに基づいて元の API リクエストの処理を続行します。

このトラフィックフローは、なぜリモートポッド CIDR を正しく設定してルーティングする必要があるかを示しています。

- VPC には、オンプレミスゲートウェイを指すリモートポッド CIDR のルートが登録されている必要があります。
- オンプレミスネットワークには、そうしたポッドをホストしている特定のノード宛てにトラフィックを送信するポッド CIDR のルートが登録されている必要があります。
- このルーティング設定がないと、ハイブリッドノード上のポッドで実行されているウェブフックやその他の同じようなサービスに EKS コントロールプレーンから到達できなくなります。

## ハイブリッドノードで実行されているポッド間



このセクションでは、異なるハイブリッドノード上で実行されているポッド同士がどのように通信しているかについて説明します。この例では、CNI がカプセル化に VXLAN を使用しているものとしま

す。これは、Cilium や Calico などの CNI でよく見られることです。プロセス全体は、Geneve や IP-in-IP など、他のカプセル化プロトコルに似ています。

## リクエスト

### 1. ポッド A が通信を開始する

ノード 1 上のポッド A (10.85.1.56) が、ノード 2 上のポッド B (10.85.2.67) にトラフィックを送信しようとしています。初期パケットは以下のようになります。

```
+-----+-----+-----+-----+
Ethernet Header	IP Header	TCP/UDP	Payload
Src: Pod A MAC	Src: 10.85.1.56	Src: 43721	
Dst: Gateway MAC	Dst: 10.85.2.67	Dst: 8080	
+-----+-----+-----+-----+
```

### 2. CNI がパケットをインターセプトして処理する

ポッド A のパケットが自身のネットワーク名前空間を出ると、CNI がそのパケットをインターセプトします。CNI は自身のルーティングテーブルに問い合わせ、宛先 IP (10.85.2.67) はポッド CIDR に属している、この IP はローカルノードではなくノード 2 (10.80.0.3) に属している、パケットは VXLAN でカプセル化する必要がある、と判断します。

カプセル化するという判断は重要です。基盤となる物理ネットワークが、ノード IP 間のトラフィックをルーティングする方法は認識しているものの、ポッド CIDR を直接ルーティングする方法は認識していないためです。

CNI は、元のパケット全体を VXLAN フレーム内にカプセル化します。これにより、「パケット内のパケット」が次のような新しいヘッダー付きで効果的に作成されます。

```
+-----+-----+-----+-----+
+-----+
| Outer Ethernet | Outer IP | Outer UDP | VXLAN | Original Pod-to-Pod
| | | | |
| Src: Node1 MAC | Src: 10.80.0.2 | Src: Random | VNI: 42 | Packet (unchanged
| | | | |
| Dst: Router MAC | Dst: 10.80.0.3 | Dst: 8472 | | from above)
| |
+-----+-----+-----+-----+
+-----+
```

このカプセル化の重要なポイントは、外側のパケットにはノード 1 (10.80.0.2) からノード 2 (10.80.0.3) までのアドレスが指定されること、UDP ポート 8472 は Cilium がデフォルトで使用する VXLAN ポートであること、VXLAN Network Identifier (VNI) はこのパケットが属するオーバーレイネットワークを識別すること、元のパケット (ポッド A の IP を送信元とし、ポッド B の IP を宛先とする) 全体が内側でもそのまま保持されることです。

これで、カプセル化されたパケットは、ノード 1 の通常のネットワークスタックに入って、他のパケットと同じ方法で処理されます。

1. ノードネットワークでの処理: ノード 1 のネットワークスタックは、宛先 (10.80.0.3) に基づいてパケットをルーティングします。
2. ローカルネットワークでの配信:
  - 両方のノードが同じレイヤー 2 ネットワークにある場合、パケットはノード 2 に直接送信されます。
  - それぞれ別のサブネットにある場合、パケットはまずローカルルーターに転送されます。
3. ルーターでの処理: ルーターは、自身のルーティングテーブルに基づいてパケットを転送して、ノード 2 に配信します。

### 3. 受信側のノードでの処理

カプセル化されたパケットがノード 2 (10.80.0.3) に到達すると、次のようになります。

1. ノードのネットワークスタックは、パケットを受信し、VXLAN パケット (UDP ポート 4789) であると識別します。
2. パケットは、さらに処理するために CNI の VXLAN インターフェイスに渡されます。

### 4. VXLAN カプセル化解除

ノード 2 上の CNI は、次のように VXLAN パケットを処理します。

1. 外側のヘッダー (イーサネット、IP、UDP、VXLAN) を取り除きます。
2. 内側の元のパケットを抽出します。
3. これで、パケットは元の形式に戻ります。

```
+-----+-----+-----+-----+
| Ethernet Header | IP Header | TCP/UDP | Payload |
```



|                  |                 |            |         |
|------------------|-----------------|------------|---------|
| Src: Pod A MAC   | Src: 10.85.1.56 | Src: 43721 |         |
| Dst: Gateway MAC | Dst: 10.85.2.67 | Dst: 8080  |         |
| +-----+          | +-----+         | +-----+    | +-----+ |

ノード 2 上の CNI は、宛先 IP (10.85.2.67) を確認して、次の処理を行います。

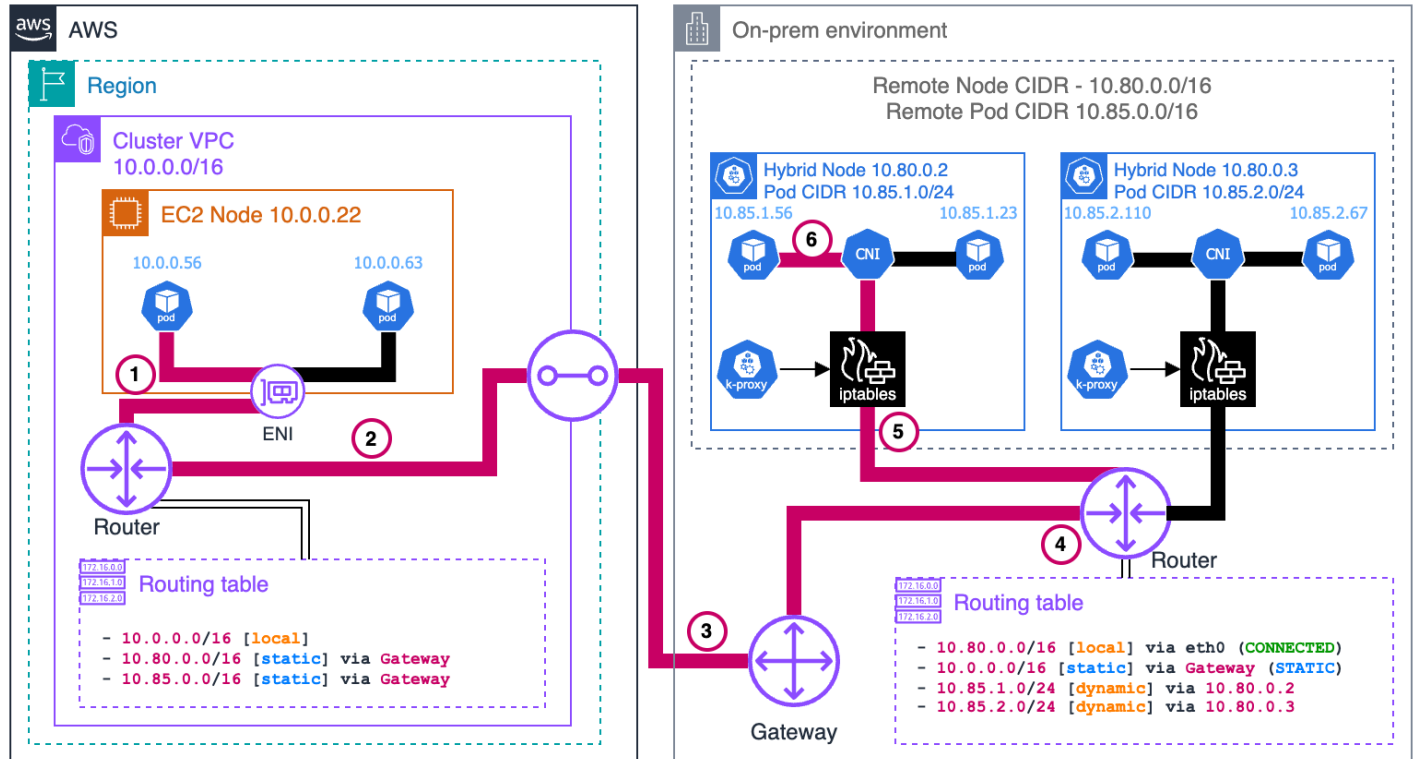
1. この IP がローカルポッドに属しているかどうかを識別します。
2. 適切な仮想インターフェイス経由でパケットをルーティングします。
3. ポッド B のネットワーク名前空間にパケットを配信します。

## レスポンス

ポッド B がポッド A に応答するときは、プロセス全体が逆方向になります。

4. ポッド B がポッド A (10.85.1.56) にパケットを送信します。
5. ノード 2 の CNI は、そのパケットを VXLAN でカプセル化し、宛先をノード 1 (10.80.0.2) に設定します。
6. カプセル化されたパケットがノード 1 に配信されます。
7. ノード 1 の CNI は、パケットをカプセル化解除し、元のレスポンスをポッド A に配信します。

## クラウドノード上のポッドからハイブリッドノード上のポッドまでのフロー (東西トラフィック)



## リクエスト

## 1. ポッド A が通信を開始する

EC2 ノード上のポッド A (10.0.0.56) がハイブリッドノード上のポッド B (10.85.1.56) にトラフィックを送信しようとしています。初期パケットは以下のようになります。

```

+-----+-----+-----+
IP Header	TCP Header	Payload
Src: 10.0.0.56	Src: 52390 (random)	
Dst: 10.85.1.56	Dst: 8080	
+-----+-----+-----+

```

VPC CNI では、ポッド A に VPC CIDR からの IP があるため、ポッド A は EC2 インスタンス上の ENI に直接アタッチされます。ポッドのネットワーク名前空間は VPC ネットワークに接続されているため、パケットは VPC ルーティングインフラストラクチャに直接入ります。

## 2. VPC でのルーティング

VPC ルートテーブルにはリモートポッド CIDR (10.85.0.0/16) に固有のルートが登録されているため、このトラフィックは VPC-to-onprem ゲートウェイ宛てに送信されます。

| Destination  | Target                         |
|--------------|--------------------------------|
| 10.0.0.0/16  | local                          |
| 10.85.0.0/16 | vgw-id (VPC-to-onprem gateway) |

このルーティングルールに基づいて、パケットはオンプレミスネットワークに接続しているゲートウェイに向けて送信されます。

### 3. 境界を越えて移動する

ゲートウェイは、確立済みの接続 (Direct Connect や VPN など) 経由でクラウド境界を越えてオンプレミスネットワークにパケットを転送します。このトランジットを全体を通して、パケットの送信元 IP アドレスと宛先 IP アドレスは元のまま維持されます。

### 4. オンプレミスネットワークでの受信

パケットは、ローカルオンプレミスルーターに到達します。ルーターは、自身のルーティングテーブルに問い合わせ、10.85.1.56 アドレスに到達するための次のホップを決定します。オンプレミスルーターには、適切なハイブリッドノード宛てにトラフィックを送信するポッド CIDR へのルートが必要です。

ルーターのルートテーブルには IP 10.80.0.2 のハイブリッドノード経由で 10.85.1.0/24 サブネットに到達できることを示すエントリが登録されています。

| Destination  | Next Hop  |
|--------------|-----------|
| 10.85.1.0/24 | 10.80.0.2 |

### 5. ノードネットワークでの処理

ルーターは、パケットをハイブリッドノード (10.80.0.2) に転送します。パケットがノードに到達したとき、パケットのポッド A の IP は送信元のまま、ポッド B の IP は宛先のままです。

### 6. CNI での処理

ノードのネットワークスタックは、パケットを受信し、宛先 IP が自身のものでないことを確認して、さらに処理するために CNI に渡します。CNI は、宛先 IP がこのノードでローカルに実行されているポッドに属しているかどうかを識別し、適切な仮想インターフェイス経由で適切なポッドにパケットを転送します。

```
Original packet -> node routing -> CNI -> Pod B's network namespace
```

ポッド B は、パケットを受信し、必要に応じて処理します。

## レスポンス

### 6. ポッド B がレスポンスを送信する

ポッド B は、自身の IP を送信元、ポッド A の IP を宛先として、レスポンスパケットを作成します。

```
+-----+-----+-----+
IP Header	TCP Header	Payload
Src: 10.85.1.56	Src: 8080	
Dst: 10.0.0.56	Dst: 52390	
+-----+-----+-----+
```

CNI は、このパケットが外部ネットワーク宛てかどうかを識別して、ノードのネットワークスタックに渡します。

### 5. ノードネットワークでの処理

ノードは、宛先 IP (10.0.0.56) がローカルネットワークに属していないと判断し、自身のデフォルトゲートウェイ (ローカルルーター) にパケットを転送します。

### 4. ローカルルーターのルーティング

ローカルルーターは、自身のルーティングテーブルに問い合わせ、宛先 IP (10.0.0.56) が VPC CIDR (10.0.0.0/16) に属していると判断します。次に、AWS に接続しているゲートウェイにパケットを転送します。

### 3. 境界を越えて移動する

パケットは、同じオンプレミスを通して VPC 接続まで戻り、クラウド境界を逆方向に横断します。

### 2. VPC でのルーティング

パケットが VPC に到達すると、ルーティングシステムは宛先 IP が VPC 内のサブネットに属しているかどうかを識別します。パケットは、ポッド A をホストしている EC2 インスタンスに向けて、VPC ネットワーク経由でルーティングされます。

## 1. ポッド A がレスポンスを受信する

パケットは、EC2 インスタンスに到達し、そこにアタッチされている ENI 経由でポッド A に直接配信されます。VPC CNI は VPC 内のポッドにオーバーレイネットワークを使用しないため、別途カプセル化解除は必要なく、パケットは元のヘッダーのまま届きます。

この東西トラフィックフローは、なぜリモートポッド CIDR を正しく設定して、両方向からルーティング可能にする必要があるかを示しています。

- VPC には、オンプレミスゲートウェイを指すリモートポッド CIDR のルートが登録されている必要があります。
- オンプレミスネットワークには、そうしたポッドをホストしている特定のノード宛てにトラフィックを送信するルートがポッド CIDR で登録されている必要があります。

## ハイブリッドノード **nodeadm** 参照

Amazon EKS Hybrid Nodes CLI (nodeadm) は、ハイブリッドノードコンポーネントのインストール、設定、登録、アンインストールを簡素化します。オペレーティングシステムイメージに nodeadm を含めると、ハイブリッドノードのブートストラップを自動化できます。詳細については、「[the section called “オペレーティングシステムを準備する”](#)」を参照してください。

ハイブリッドノードの nodeadm バージョンは、Amazon EKS クラスター内のノードとして Amazon EC2 インスタンスをブートストラップするために使用される nodeadm バージョンとは異なります。適切な nodeadm バージョンのドキュメントとリファレンスに従ってください。このドキュメントページは、ハイブリッドノードの nodeadm バージョンを対象としています。

ハイブリッドノード nodeadm のソースコードは、<https://github.com/aws/eks-hybrid> GitHub リポジトリで公開されています。

### Important

root/sudo 権限を持つユーザーで nodeadm を実行する必要があります。

## nodeadm のダウンロード

nodeadm のハイブリッドノード版は Amazon CloudFront をフロントに置いた Amazon S3 でホストされます。各オンプレミスホストに nodeadm をインストールするにはオンプレミスホストから以下のコマンドを実行してください。

## x86\_64 ホストの場合

```
curl -OL 'https://hybrid-assets.eks.amazonaws.com/releases/latest/bin/linux/amd64/nodeadm'
```

## ARM ホストの場合

```
curl -OL 'https://hybrid-assets.eks.amazonaws.com/releases/latest/bin/linux/arm64/nodeadm'
```

各ホストでダウンロードしたバイナリに実行可能ファイルのアクセス許可を追加します。

```
chmod +x nodeadm
```

## nodeadm install

`nodeadm install` コマンドはハイブリッドノードを実行して Amazon EKS クラスターに結合するために必要なアーティファクトと依存関係をインストールするために使用されます。`nodeadm install` コマンドは各ハイブリッドノードで個別に実行することも、イメージビルドパイプライン中に実行して、オペレーティングシステムイメージにハイブリッドノードの依存関係をプリインストールすることもできます。

### 使用方法

```
nodeadm install [KUBERNETES_VERSION] [flags]
```

### 位置引数

(必須 `KUBERNETES_VERSION` インストールする EKS Kubernetes のメジャーバージョンとマイナーバージョン (例: 1.32))

### Flags

| 名前                                                     | 必要 | 説明                                                                                                      |
|--------------------------------------------------------|----|---------------------------------------------------------------------------------------------------------|
| <code>-p,</code><br><code>--credential-provider</code> | 正  | インストールする認証情報プロバイダー。サポートされている値は <code>iam-ra</code> および <code>ssm</code> です。詳細については「 <a href="#">the</a> |

| 名前 | 必要 | 説明                                                     |
|----|----|--------------------------------------------------------|
|    |    | <a href="#">section called “認証情報を準備する”</a> 」を参照してください。 |

| 名前                                                   | 必要 | 説明                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
|------------------------------------------------------|----|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>-s,</code><br><code>--containerd-source</code> | 誤  | <p>containerd のソース。nodeadm はOS ディストリビューション、Docker パッケージからの containerd のインストール、および containerd のインストールのスキップをサポートしています。</p> <p>[値]</p> <p>distro - これはデフォルト値です。nodeadm はノード OS によって配布される containerd パッケージをインストールします。distro は Red Hat Enterprise Linux (RHEL オペレーティングシステムでサポートされる値ではありません)。</p> <p>docker — nodeadm は Docker によって構築および配布された containerd パッケージをインストールします。docker は Amazon Linux 2023 でサポートされる値ではありません</p> <p>none — nodeadm は containerd パッケージをインストールしません。nodeadm init を実行する前に、containerd を手</p> |



| 名前               | 必要 | 説明                                                                                          |
|------------------|----|---------------------------------------------------------------------------------------------|
|                  |    | 動でインストールする必要があります。                                                                          |
| -r,<br>--region  | 誤  | SSM エージェントといったアーティファクトをダウンロードするための AWS リージョンを指定します。デフォルトは us-west-2 です。                     |
| -t,<br>--timeout | 誤  | install コマンドの最大実行時間。入力 は 時間形式に従います。例: 1h23m。install コマンドのデフォルトのダウンロードタイムアウトは 20 分に設定されています。 |
| -h, --help       | 誤  | 使用可能なフラグ、サブコマンド、位置値パラメータを含むヘルプメッセージを表示します。                                                  |

## 例

認証情報プロバイダーとして AWS システムマネージャー (SSM 使用して Kubernetes バージョン 1.32 をインストールする

```
nodeadm install 1.32 --credential-provider ssm
```

AWS システムマネージャー (SSM 認証情報プロバイダーとして、Docker を containerd のソースとして、ダウンロードタイムアウトを 20 分にして Kubernetes バージョン 1.32 をインストールします。

```
nodeadm install 1.32 --credential-provider ssm --containerd-source docker --timeout 20m
```

認証情報プロバイダーとして AWS IAM Roles Anywhere を使用して Kubernetes バージョン 1.32 をインストールする

```
nodeadm install 1.32 --credential-provider iam-ra
```

## nodeadm config check

nodeadm config check コマンドは提供されたノード設定にエラーがないか確認します。このコマンドはハイブリッドノード設定ファイルの正確性を検証するために使用できます。

### 使用方法

```
nodeadm config check [flags]
```

### Flags

| 名前                     | 必要 | 説明                                                             |
|------------------------|----|----------------------------------------------------------------|
| -c,<br>--config-source | 正  | nodeadm 設定のソース。ハイブリッドノードの場合、入力は file スキームを使用する URI に従う必要があります。 |
| -h, --help             | 誤  | 使用可能なフラグ、サブコマンド、位置値パラメータを含むヘルプメッセージを表示します。                     |

### 例

```
nodeadm config check -c file:///nodeConfig.yaml
```

## nodeadm init

nodeadm init コマンドはハイブリッドノードを起動し、設定された Amazon EKS クラスターに接続します。nodeConfig.yaml ファイルの設定方法の詳細については、「[the section called “SSM ハイブリッドアクティベーション用のノード設定”](#)」または「[the section called “IAM Roles Anywhere 用のノード設定”](#)」を参照してください。

### 使用方法

```
nodeadm init [flags]
```

## Flags

| 名前                                               | 必要 | 説明                                                                                                                                                                                                                                                                                                                                                                                                 |
|--------------------------------------------------|----|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>-c,</code><br><code>--config-source</code> | 正  | nodeadm 設定のソース。ハイブリッドノードの場合、入力は file スキームを使用する URI に従う必要があります。                                                                                                                                                                                                                                                                                                                                     |
| <code>-s,</code><br><code>--skip</code>          | 誤  | <p>スキップする init のフェーズ。問題の修正に役立つ場合を除き、どのフェーズもスキップすることはお勧めしません。</p> <p>[値]</p> <p><code>install-validation</code> は、先行する <code>install</code> コマンドが正常に実行されたかどうかの確認をスキップします。</p> <p>ノードでファイアウォールが有効になっている場合、<code>cni-validation</code> は Cilium または Calico CNI の VXLAN ポートが開かれているかどうかの確認をスキップします。</p> <p><code>node-ip-validation</code> は、ノード IP がリモートノードネットワークの CIDR 内に含まれているかどうかのチェックをスキップします。</p> |

| 名前         | 必要 | 説明                                         |
|------------|----|--------------------------------------------|
| -h, --help | 誤  | 使用可能なフラグ、サブコマンド、位置値パラメータを含むヘルプメッセージを表示します。 |

## 例

```
nodeadm init -c file://nodeConfig.yaml
```

## nodeadm upgrade

nodeadm upgrade コマンドはインストールされているすべてのアーティファクトを最新バージョンにアップグレードし、ノードをブートストラップしてアップグレードされたアーティファクトを設定し、AWS の EKS クラスタに参加させます。アップグレードはノードで実行されているワークロードに対する破壊的なコマンドです。アップグレードを実行する前にはワークロードを別のノードに移動してください。

## 使用方法

```
nodeadm upgrade [KUBERNETES_VERSION] [flags]
```

## 位置引数

(必須 KUBERNETES\_VERSION インストールする EKS Kubernetes のメジャーバージョンとマイナーバージョン (例: 1.32))

## Flags

| 名前                     | 必要 | 説明                                                             |
|------------------------|----|----------------------------------------------------------------|
| -c,<br>--config-source | 正  | nodeadm 設定のソース。ハイブリッドノードの場合、入力は file スキームを使用する URI に従う必要があります。 |

| 名前                                         | 必要 | 説明                                                                                                                                                                                                                                                                              |
|--------------------------------------------|----|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>-t,</code><br><code>--timeout</code> | 誤  | アーティファクトのダウンロードのタイムアウト。入力は時間形式に従います。たとえば、1h23m です。upgrade コマンドのデフォルトのダウンロードタイムアウトは 10 分に設定されています。                                                                                                                                                                               |
| <code>-s,</code><br><code>--skip</code>    | 誤  | <p>スキップするアップグレードフェーズ。問題の修正に役立つ場合を除き、どのフェーズもスキップすることはお勧めしません。</p> <p>[値]</p> <p>pod-validation はデモンセツトと静的ポッドを除くすべてのポッドがノード上で実行されていないことのチェックをスキップします。</p> <p>node-validation はノードが隔離されているかどうかの確認をスキップします。</p> <p>init-validation はアップグレードを実行する前にノードが正常に初期化されたかどうかの確認をスキップします。</p> |
| <code>-h, --help</code>                    | 誤  | 使用可能なフラグ、サブコマンド、位置値パラメータを含むヘルプメッセージを表示します。                                                                                                                                                                                                                                      |

## 例

```
nodeadm upgrade 1.32 -c file:///nodeConfig.yaml
```

```
nodeadm upgrade 1.32 -c file:///nodeConfig.yaml --timeout 20m
```

## nodeadm uninstall

nodeadm uninstall コマンドは kubelet や containerd など、nodeadm install の実行中に nodeadm がインストールしたアーティファクトを停止および削除してください。uninstall コマンドはクラスターからハイブリッドノードをドレインまたは削除しないことに注意してください。ドレインと削除の操作は個別に実行する必要があります。詳細については「[the section called “ハイブリッドノードを削除する”](#)」を参照してください。デフォルトではノードにポッドが残っている場合、nodeadm uninstall は続行されません。同様に、nodeadm uninstall は CNI 依存関係や、クラスターで実行している他の Kubernetes アドオンの依存関係を削除しません。ホストから CNI インストールを完全に削除するには「[the section called “CNI を設定する”](#)」の手順を参照してください。オンプレミス認証情報プロバイダーとして AWS SSM ハイブリッドアクティベーションを使用している場合、nodeadm uninstall コマンドはホストを AWS SSM マネージドインスタンスから登録解除します。

## 使用方法

```
nodeadm uninstall [flags]
```

## Flags

| 名前            | 必要 | 説明                                                                                                        |
|---------------|----|-----------------------------------------------------------------------------------------------------------|
| -s,<br>--skip | 誤  | スキップするアップグレードフェーズ。問題の修正に役立つ場合を除き、どのフェーズもスキップすることはお勧めしません。<br><br>[値]<br><br>pod-validation はデモンセットと静的ポッドを |

| 名前            | 必要 | 説明                                                                                                                                                               |
|---------------|----|------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|               |    | 除くすべてのポッドがノード上で実行されていないことのチェックをスキップします。<br><br>node-validation はノードが隔離されているかどうかの確認をスキップします。<br><br>init-validation はアンインストールを実行する前にノードが正常に初期化されたかどうかの確認をスキップします。 |
| -h,<br>--help | 誤  | 使用可能なフラグ、サブコマンド、位置値パラメータを含むヘルプメッセージを表示します。                                                                                                                       |

## 例

```
nodeadm uninstall
```

```
nodeadm uninstall --skip node-validation,pod-validation
```

## nodeadm debug

nodeadm debug コマンドは異常または誤設定のハイブリッドノードのトラブルシューティングに使用できます。以下の要件が満たされていることを検証します。

- ノードが認証情報を取得するために必要な AWS API へのネットワークアクセスがあること
- ノードが設定されたハイブリッドノードの IAM ロールに対し AWS 認証情報を取得できること
- ノードに EKS Kubernetes API エンドポイントへのネットワークアクセスがあり、EKS Kubernetes API エンドポイント証明書が有効であること

- ノードが EKS クラスターに対し認証でき、クラスター内での ID が有効で、EKS クラスター用に設定された VPC を介して EKS クラスターにアクセスできること。

エラーが見つかった場合、コマンドの出力はトラブルシューティングのステップを提示します。特定の検証ステップは子プロセスを示しています。これらが失敗した場合、出力は検証エラーの下で stderr セクションに表示されます。

## 使用方法

```
nodeadm debug [flags]
```

## Flags

| 名前                               | 必要 | 説明                                                             |
|----------------------------------|----|----------------------------------------------------------------|
| <code>-c, --config-source</code> | 正  | nodeadm 設定のソース。ハイブリッドノードの場合、入力は file スキームを使用する URI に従う必要があります。 |
| <code>--no-color</code>          | 誤  | 色の出力を無効にします。自動化に役立ちます。                                         |
| <code>-h, --help</code>          | 誤  | 使用可能なフラグ、サブコマンド、位置値パラメータを含むヘルプメッセージを表示します。                     |

## 例

```
nodeadm debug -c file://nodeConfig.yaml
```

## Nodeadm ファイルの場所

### nodeadm のインストール

`nodeadm install` を実行すると、以下のファイルおよびファイルの場所が設定されます。



| アーティファクト                 | パス                                                                                                                  |
|--------------------------|---------------------------------------------------------------------------------------------------------------------|
| IAM Roles Anywhere       | /usr/local/bin/aws_signing_helper                                                                                   |
| Kubelet binary           | /usr/bin/kubelet                                                                                                    |
| Kubectrl binary          | usr/local/bin/kubectrl                                                                                              |
| ECR Credentials Provider | /etc/eks/image-credential-provider/ecr-credential-provider                                                          |
| AWS IAM オーセンティケーター       | /usr/local/bin/aws-iam-authenticator                                                                                |
| SSM Setup CLI            | /opt/ssm/ssm-setup-cli                                                                                              |
| SSM Agent                | Ubuntu の場合 - /snap/amazon-ssm-agent/current/amazon-ssm-agent<br><br>RHEL および AL2023 の場合 - /usr/bin/amazon-ssm-agent |
| Containerd               | Ubuntu および AL2023 の場合 - /usr/bin/containerd<br><br>RHEL の場合 - /bin/containerd                                       |
| Iptables                 | Ubuntu および AL2023 の場合 - /usr/sbin/iptables<br><br>RHEL の場合 - /sbin/iptables                                         |
| CNI プラグイン                | /opt/cni/bin                                                                                                        |
| インストール済みアーティファクトトラッカー    | /opt/nodeadm/tracker                                                                                                |

nodeadm init

nodeadm init を実行すると、以下のファイルおよびファイルの場所が設定されます。

| 名前                              | パス                                                    |
|---------------------------------|-------------------------------------------------------|
| Kubelet kubeconfig              | /var/lib/kubelet/kubeconfig                           |
| Kubelet 設定                      | /etc/kubernetes/kubelet/config.json                   |
| Kubelet systemd ユニット            | /etc/systemd/system/kubelet.service                   |
| イメージ認証情報プロバイダー設定                | /etc/eks/image-credential-provider/config.json        |
| Kubelet 環境ファイル                  | /etc/eks/kubelet/environment                          |
| Kubelet 証明書                     | /etc/kubernetes/pki/ca.crt                            |
| Containerd 設定                   | /etc/containerd/config.toml                           |
| Containerd カーネルモジュール設定          | /etc/modules-load.d/containerd.conf                   |
| AWS 設定ファイル                      | /etc/aws/hybrid/config                                |
| AWS 認証情報ファイル (認証情報ファイルを有効にする場合) | /eks-hybrid/.aws/credentials                          |
| AWS 署名ヘルパーシステムユニット              | /etc/systemd/system/aws_signing_helper_update.service |
| Sysctl 設定ファイル                   | /etc/sysctl.d/99-nodeadm.conf                         |
| Ca-certificates                 | /etc/ssl/certs/ca-certificates.crt                    |
| Gpg キーファイル                      | /etc/apt/keyrings/docker.asc                          |
| Docker リポジトリソースファイル             | /etc/apt/sources.list.d/docker.list                   |

## SSM ハイブリッドアクティベーション用のノード設定

ハイブリッドノードの認証情報に AWS SSM ハイブリッドアクティベーションを使用する場合の `nodeConfig.yaml` の例を以下に示します。

```
apiVersion: node.eks.aws/v1alpha1
```

```
kind: NodeConfig
spec:
 cluster:
 name: # Name of the EKS cluster
 region: # AWS Region where the EKS cluster resides
 hybrid:
 ssm:
 activationCode: # SSM hybrid activation code
 activationId: # SSM hybrid activation id
```

## IAM Roles Anywhere 用のノード設定

ハイブリッドノードの認証情報用の AWS IAM Roles Anywhere の nodeConfig.yaml の例を以下に示します。

AWS IAM Roles Anywhere をオンプレミス認証情報プロバイダーとして使用する場合、nodeadm 設定で使用する nodeName はハイブリッドノードの IAM ロールを対象としたアクセス許可と一致する必要があります。たとえば、ハイブリッドノードの IAM ロールのアクセス許可ではロールセッション名がホスト証明書の CN と一致する場合しか AWS IAM Roles Anywhere がロールを引き受けることは許可されない場合、nodeadm 設定の nodeName は証明書の CN と同じである必要があります。使用する nodeName は 64 文字以下にする必要があります。詳細については、「[the section called “認証情報を準備する”](#)」を参照してください。

```
apiVersion: node.eks.aws/v1alpha1
kind: NodeConfig
spec:
 cluster:
 name: # Name of the EKS cluster
 region: # AWS Region where the EKS cluster resides
 hybrid:
 iamRolesAnywhere:
 nodeName: # Name of the node
 trustAnchorArn: # ARN of the IAM Roles Anywhere trust anchor
 profileArn: # ARN of the IAM Roles Anywhere profile
 roleArn: # ARN of the Hybrid Nodes IAM role
 certificatePath: # Path to the certificate file to authenticate with the IAM
 Roles Anywhere trust anchor
 privateKeyPath: # Path to the private key file for the certificate
```

## kubelet をカスタマイズするためのノード設定 (オプション)

nodeadm 設定で kubelet 設定とフラグを渡すことができます。追加のノードラベル `abc.amazonaws.com/test-label` を追加する方法と、`shutdownGracePeriod` を 30 秒に設定するための設定については以下の例を参照してください。

```
apiVersion: node.eks.aws/v1alpha1
kind: NodeConfig
spec:
 cluster:
 name: # Name of the EKS cluster
 region: # AWS Region where the EKS cluster resides
 kubelet:
 config: # Map of kubelet config and values
 shutdownGracePeriod: 30s
 flags: # List of kubelet flags
 - --node-labels=abc.company.com/test-label=true
 hybrid:
 ssm:
 activationCode: # SSM hybrid activation code
 activationId: # SSM hybrid activation id
```

## containerd をカスタマイズするためのノード設定 (オプション)

nodeadm 設定でカスタム containerd 設定を渡すことができます。nodeadm の containerd 設定はインライン TOML を受け入れます。containerd コンテンツストアで解凍されたイメージレイヤーの削除を無効にするための containerd の設定方法については以下の例を参照してください。

```
apiVersion: node.eks.aws/v1alpha1
kind: NodeConfig
spec:
 cluster:
 name: # Name of the EKS cluster
 region: # AWS Region where the EKS cluster resides
 containerd:
 config: | # Inline TOML containerd additional configuration
 [plugins."io.containerd.grpc.v1.cri".containerd]
 discard_unpacked_layers = false
 hybrid:
 ssm:
 activationCode: # SSM hybrid activation code
 activationId: # SSM hybrid activation id
```

containerd 設定を使用して SELinux サポートを有効にすることもできます。containerd で SELinux が有効になっている場合はノードでスケジュールされたポッドで適切な securityContext と selinuxOptions が有効になっていることを確認してください。セキュリティコンテキストの設定の詳細については[Kubernetes ドキュメント](#)を参照してください。

#### Note

Red Hat Enterprise Linux (RHEL 8 および RHEL 9 では SELinux がデフォルトで有効になっており、ホスト上で strict に設定されています。Amazon Linux 2023 はデフォルトで SELinux が有効になっており、許可モードに設定されています。SELinux がホストで許可モードに設定されている場合、containerd で有効にしてもリクエストはブロックされませんが、ホストの SELinux 設定に従ってログに記録されます。

```
apiVersion: node.eks.aws/v1alpha1
kind: NodeConfig
spec:
 cluster:
 name: # Name of the EKS cluster
 region: # AWS Region where the EKS cluster resides
 containerd:
 config: | # Inline TOML containerd additional configuration
 [plugins."io.containerd.grpc.v1.cri"]
 enable_selinux = true
 hybrid:
 ssm:
 activationCode: # SSM hybrid activation code
 activationId: # SSM hybrid activation id
```

## ハイブリッドノードのトラブルシューティング

このトピックでは、Amazon EKS Hybrid Nodes の使用中に表示される一般的なエラーとその修復方法について説明します。その他のトラブルシューティング情報については、「[トラブルシューティング](#)」および AWS re:Post の「[Amazon EKS のナレッジセンタータグ](#)」を参照してください。問題を解決できない場合は、AWS サポートにお問い合わせください。

**nodeadm debug** を使用したノードのトラブルシューティング ハイブリッドノードから nodeadm debug コマンドを実行して、ネットワークと認証情報の要件が満たされていることを検証できます。nodeadm debug コマンドの詳細については、「[the section called “ハイブリッドノード用 nodeadm”](#)」を参照してください。

クラスターインサイトを使用してハイブリッドノードの問題を検出する Amazon EKS クラスターインサイトには、クラスター内の EKS Hybrid Nodes の設定に関する一般的な問題を検出するインサイトチェックが含まれます。すべてのインサイトチェックの結果は AWS Management Console、AWS CLI、および AWS SDK から表示できます。クラスターインサイトの詳細については、「[the section called “クラスターのインサイト”](#)」を参照してください。

## ハイブリッドノードのインストールに関するトラブルシューティング

以下のトラブルシューティングのトピックは、`nodeadm install` コマンドを使用してホストにハイブリッドノードの依存関係をインストールすることに関連しています。

### `nodeadm` コマンドが **must run as root** に失敗した

`nodeadm install` コマンドは、ホストに対して root または `sudo` 権限を持つユーザーに実行させる必要があります。root または `sudo` 権限を持たないユーザーに `nodeadm install` を実行させると、`nodeadm` 出力に次のエラーが表示されます。

```
"msg":"Command failed","error":"must run as root"
```

### 依存関係に接続できない

`nodeadm install` コマンドは、ハイブリッドノードに必要な依存関係をインストールします。ハイブリッドノードの依存関係には、`containerd`、`kubelet`、`kubectl`、および AWS SSM または AWS IAM Roles Anywhere コンポーネントがあります。これらの依存関係をダウンロードするには、`nodeadm install` を実行している場所からアクセスする必要があります。アクセスする必要があるロケーションのリストの詳細については、「[the section called “ネットワークを準備する”](#)」を参照してください。アクセス権限がない場合は、次のようなエラーが `nodeadm install` 出力に表示されます。

```
"msg":"Command failed","error":"failed reading file from url: ...: max retries achieved for http request"
```

### パッケージマネージャーの更新に失敗しました

`nodeadm install` コマンドは、ハイブリッドノードの依存関係をインストールする前に、`apt update`、`yum update`、`dnf update` のいずれかを実行します。このステップが失敗すると、次のようなエラーが表示される可能性があります。これを修正するには、`apt update`、`yum update`、`dnf update` のいずれかを実行してから `nodeadm install` を実行するか、`nodeadm install` を再実行します。

```
failed to run update using package manager
```

タイムアウトまたはコンテキストの期限を超過しました

`nodeadm install` を実行するときに、インストールプロセスのさまざまな段階でタイムアウトまたはコンテキストの期限を超過したことを示すエラーの問題が発生した場合、接続が遅くなり、タイムアウトが完了する前にハイブリッドノードの依存関係がインストールされない可能性があります。これらの問題を回避するには、`nodeadm` の `--timeout` フラグを使用して、依存関係のダウンロードのタイムアウト期間を延長できます。

```
nodeadm install K8S_VERSION --credential-provider CRED_PROVIDER --timeout 20m0s
```

## ハイブリッドノードの接続のトラブルシューティング

このセクションのトラブルシューティングトピックは、`nodeadm init` コマンドを使用してハイブリッドノードを EKS クラスターに接続するプロセスに関連しています。

オペレーションエラーまたはサポートされていないスキーム

`nodeadm init` を実行するときに、`operation error` または `unsupported scheme` に関連するエラーが表示された場合は、`nodeConfig.yaml` が正しくフォーマットされ、`nodeadm` に渡されているか確認します。`nodeConfig.yaml` のフォーマットのオプションの詳細については、「[the section called “ハイブリッドノード用 nodeadm”](#)」を参照してください。

```
"msg":"Command failed","error":"operation error ec2imds: GetRegion, request canceled, context deadline exceeded"
```

ハイブリッドノード IAM ロールに **eks:DescribeCluster** アクションのアクセス許可がありません

`nodeadm init` を実行すると、`nodeadm` は `Describe Cluster` を呼び出して EKS クラスターに関する情報の収集を試みます。ハイブリッドノード IAM ロールに `eks:DescribeCluster` アクションのアクセス許可がない場合。ハイブリッドノード IAM ロールに必須のアクセス許可の詳細については、「[the section called “認証情報を準備する”](#)」を参照してください。

```
"msg":"Command failed","error":"operation error EKS: DescribeCluster, https response error StatusCode: 403 ... AccessDeniedException"
```

リモートノードネットワークの CIDR がないノード IP

nodeadm init を実行するときに、指定されたりモートノードネットワークの CIDR の中にノードの IP アドレスがない場合、エラーが発生することがあります。エラーは次の例のような見た目になります。

```
node IP 10.18.0.1 is not in any of the remote network CIDR blocks [10.0.0.0/16
192.168.0.0/16]
```

こちらの例では、IP 10.18.0.1 のノードが、リモートネットワークの CIDR 10.0.0.0/16 と 192.168.0.0/16 でクラスターを結合しようとしています。10.18.0.1 がいずれの範囲にもないため、エラーが発生しています。

RemoteNodeNetworks が正しく設定され、すべてのノード IP アドレスが含まれていることを確認します。ネットワーク設定の詳細については、「[the section called “ネットワークを準備する”](#)」を参照してください。

- お使いのクラスターが配置されているリージョンで次のコマンドを実行し、RemoteNodeNetwork 設定をチェックします。出力に表示された CIDR ブロックに、ご自身のノードの IP 範囲が含まれ、エラーメッセージに表示された CIDR ブロックと同一であることを確認します。同じでない場合は、クラスター名と nodeConfig.yaml のリージョンが、対象となるクラスターと一致していることを確認します。

```
aws eks describe-cluster --name CLUSTER_NAME --region REGION_NAME --query
cluster.remoteNetworkConfig.remoteNodeNetworks
```

- 対象のノードで作業していることを確認します。
  - ホスト名と IP アドレスが、クラスターに登録しようとしているものと一致することを確認して、正しいノードで作業していることを確認します。
  - このノードが正しいオンプレミスネットワーク (クラスターのセットアップ中に CIDR 範囲が RemoteNodeNetwork として登録されたネットワーク) にあることを確認します。

それでもやはりノード IP が想定されるものと異なる場合は、以下を確認します。

- IAM Roles Anywhere を使用している場合、kubelet が DNS ルックアップを IAM Roles Anywhere で実行し、nodeName がノード名 (使用できる場合) に関連付けられた IP を使用している。ノードの DNS エントリを維持している場合、これらのエントリがリモートノードネットワークの CIDR 内の IP を指していることを確認する。



- ノードに複数のネットワークインターフェイスがある場合、kubelet は、リモートノードネットワークの CIDR の、範囲外の IP アドレスを持つインターフェイスをデフォルトとして選択することがあります。別のインターフェイスを使用するには、ご自身の nodeConfig.yaml の --node-ip kubelet フラグを使用して、その IP アドレスを指定します。詳細については、「[the section called “ハイブリッドノード用 nodeadm”](#)」を参照してください。ノードで次のコマンドを実行すると、ノードのネットワークインターフェイスとその IP アドレスを確認できます。

```
ip addr show
```

### ハイブリッドノードが EKS クラスターに表示されない

nodeadm init を実行して完了しても、ハイブリッドノードがクラスターに表示されない場合、ハイブリッドノードと EKS コントロールプレーン間のネットワーク接続に問題があるか、必要なセキュリティグループのアクセス許可が設定されていないか、Kubernetes ロールベースアクセスコントロール (RBAC) へのハイブリッドノード IAM ロールの必要なマッピングがない可能性があります。次のコマンドを使用して kubelet と kubelet ログのステータスを確認することで、デバッグプロセスを開始できます。クラスターに参加できなかったハイブリッドノードから次のコマンドを実行します。

```
systemctl status kubelet
```

```
journalctl -u kubelet -f
```

### クラスターと通信できません

ハイブリッドノードがクラスターコントロールプレーンと通信できなかった場合は、次のようなログが表示されることがあります。

```
"Failed to ensure lease exists, will retry" err="Get ..."
```

```
"Unable to register node with API server" err="Post ..."
```

```
Failed to contact API server when waiting for CSINode publishing ... dial tcp <ip address>: i/o timeout
```

これらのメッセージが表示された場合は、以下をチェックして、[the section called “ネットワークを準備する”](#) で説明されているハイブリッドノード要件を満たしていることを確認します。

- EKS クラスターに渡される VPC に、オンプレミスノードとオプションでポッド CIDR の Transit Gateway (TGW) または仮想プライベートゲートウェイ (VGW) へのルートがあることを確認します。
- EKS クラスターに追加のセキュリティグループがあり、オンプレミスノード CIDR とオプションでポッド CIDR のインバウンドルールがあることを確認します。
- EKS コントロールプレーンとの間のトラフィックを許可するようにオンプレミスルーターが設定されていることを確認します。

## Unauthorized

ハイブリッドノードが EKS コントロールプレーンと通信できたが、登録できなかった場合、次のようなログが表示されることがあります。以下のログメッセージの主な違いは Unauthorized エラーであることに注意してください。これは、必要なアクセス許可がないため、ノードがタスクを実行できなかったことを示します。

```
"Failed to ensure lease exists, will retry" err="Unauthorized"
```

```
"Unable to register node with API server" err="Unauthorized"
```

```
Failed to contact API server when waiting for CSINode publishing: Unauthorized
```

これらのメッセージが表示された場合は、以下をチェックして、[the section called “認証情報を準備する”](#) および [the section called “クラスターアクセスを準備する”](#) のハイブリッドノードの要件の詳細を満たしていることを確認します。

- ハイブリッドノードのアイデンティティが、想定の高ブリッドノードの IAM ロールと一致することを確認します。これを行うには、ハイブリッドノードから `sudo aws sts get-caller-identity` を実行します。
- ハイブリッドノード IAM ロールに必要なアクセス許可があることを確認します。
- クラスターにハイブリッドノード IAM ロールの EKS アクセスエントリがあることを確認するか、aws-auth ConfigMap にハイブリッドノード IAM ロールのエントリがあることを確認します。EKS アクセスエントリを使用している場合は、ハイブリッドノード IAM ロールのアクセスエントリが HYBRID\_LINUX アクセスタイプであることを確認します。aws-auth ConfigMap を使用している場合は、ハイブリッドノード IAM ロールのエントリが、「[the section called “クラスターアクセスを準備する”](#)」で説明されている要件と形式を満たしていることを確認します。

ハイブリッドノードが EKS クラスターに登録されているのにステータス **Not Ready** が表示される

ハイブリッドノードが EKS クラスターに正常に登録されたが、ハイブリッドノードのステータスが Not Ready である場合、最初に確認すべきことは、コンテナネットワークインターフェイス (CNI) のステータスです。CNI をインストールしていない場合は、ハイブリッドノードのステータスが Not Ready であることが予想されます。CNI が正常にインストールされ実行されると、ノードのステータスは Ready に移行します。CNI をインストールしようとしたが、正常に実行されていない場合は、このページの「[the section called “ハイブリッドノード CNI のトラブルシューティング”](#)」を参照してください。

証明書署名リクエスト (CSR) が保留中のままである

ハイブリッドノードを EKS クラスターに接続した後、ハイブリッドノードの保留中の CSR がある場合、ハイブリッドノードは自動承認の要件を満たしていません。ハイブリッドノードの CSR は、ハイブリッドノードの CSR が eks.amazonaws.com/compute-type: hybrid ラベル付きノードによって作成され、CSR に次のサブジェクト代替名 (SAN) がある場合、自動的に承認および発行されます: 少なくとも 1 つの DNS SAN はノード名と等しく、IP SAN はリモートノードネットワークの CIDR に属するものです。

ハイブリッドプロファイルは既に存在します

nodeadm 設定を変更し、ノードを新しい設定に再登録しようとする、ハイブリッドプロファイルが既に存在するが、その内容が変更されたことを示すエラーが表示されることがあります。設定変更の合間に nodeadm init を実行する代わりに、nodeadm uninstall を実行し、その後に nodeadm install と nodeadm init を実行します。これにより、設定の変更に伴う適切なクリーンアップが保証されます。

```
"msg":"Command failed","error":"hybrid profile already exists at /etc/aws/hybrid/config but its contents do not align with the expected configuration"
```

ハイブリッドノードがプライベート API の解決に失敗しました

nodeadm init 実行後、no such host が存在するために EKS Kubernetes API サーバーへの接続に失敗したことを示すエラーが kubelet ログに表示された場合は、オンプレミスネットワークまたはホストレベルで EKS Kubernetes API エンドポイントの DNS エントリを変更する必要がある場合があります。「AWS Route53 documentation」の「[Forwarding inbound DNS queries to your VPC](#)」を参照してください。

```
Failed to contact API server when waiting for CSINode publishing: Get ... no such host
```

## EKS コンソールでハイブリッドノードを表示できない

ハイブリッドノードが登録されているのに、EKS コンソールで表示できない場合は、コンソールの表示に使用している IAM プリンシパルのアクセス許可を確認してください。使用している IAM プリンシパルには、コンソールでリソースを表示するための特定の最小の IAM および Kubernetes アクセス許可が必要です。詳細については、「[the section called “クラスターリソースにアクセスする”](#)」を参照してください。

## ハイブリッドノードのトラブルシューティングの実行

EKS クラスターに登録されたハイブリッドノードのステータスが Ready で、その後ステータスが Not Ready に移行した場合、ノードに CPU、メモリ、使用可能なディスク容量の十分なリソースがない、ノードが EKS コントロールプレーンから切断されているなど、さまざまな問題が異常なステータスの原因となった可能性があります。以下のステップを使用してノードのトラブルシューティングを行うことができます。問題を解決できない場合は、AWS サポートにお問い合わせください。

ハイブリッドノードから `nodeadm debug` を実行して、ネットワークと認証情報の要件が満たされていることを確認します。`nodeadm debug` コマンドの詳細については、「[the section called “ハイブリッドノード用 nodeadm”](#)」を参照してください。

### ノードのステータスを取得する

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

### ノードの条件とイベントを確認する

```
kubectl describe node NODE_NAME
```

### ポッドのステータスを取得する

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

### ポッドの条件とイベントを確認する

```
kubectl describe pod POD_NAME
```

### ポッドログを確認する

```
kubectl logs POD_NAME
```

## kubectl ログを確認する

```
systemctl status kubelet
```

```
journalctl -u kubelet -f
```

ポッドのライブネスプローブが失敗しているか、ウェブフックが機能していない

ハイブリッドノードで動作しているアプリケーション、アドオン、またはウェブフックが正しく起動しない場合、ポッドへの通信をブロックするネットワークの問題が発生する可能性があります。EKS コントロールプレーンがハイブリッドノードで動作しているウェブフックに接続するには、リモートポッドネットワークで EKS クラスターを設定し、VPC ルーティングテーブルにオンプレミスポッド CIDR のルートを設定し、ターゲットを Transit Gateway (TGW)、仮想プライベートゲートウェイ (VPW)、または VPC をオンプレミスネットワークに接続するために使用するその他のゲートウェイとして設定する必要があります。ハイブリッドノードのネットワーク要件の詳細については、「[the section called “ネットワークを準備する”](#)」を参照してください。さらに、オンプレミスのファイアウォールでこのトラフィックを許可し、ルーターがポッドに適切にルーティングできることを確認する必要があります。ハイブリッドノードでウェブフックを実行するための要件の詳細については、「[the section called “ウェブフックの設定”](#)」を参照してください。

このシナリオの一般的なポッドログメッセージを以下に示します。ここでは、ip-address は Kubernetes サービスのクラスター IP です。

```
dial tcp <ip-address>:443: connect: no route to host
```

## kubectl logs または kubectl exec コマンドが機能しない

他の kubectl コマンドは正常に実行されているのに kubectl logs または kubectl exec コマンドがタイムアウトした場合は、リモートネットワークの設定に問題がある可能性があります。これらのコマンドはクラスターを介してノードの kubelet エンドポイントに接続しています。詳細については、「[the section called “kubelet エンドポイント”](#)」を参照してください。

ノード IP とポッド IP が、ご自身のクラスター向けに設定したリモートノードネットワークおよびリモートポッドネットワーク CIDR の範囲に含まれていることを確認します。以下のコマンドを使用して IP の割り当てを調べます。

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

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

これらの IP を、設定したリモートネットワークの CIDR と比較して、適切にルーティングされていることを確認します。ネットワークの設定要件については、「[the section called “ネットワークを準備する”](#)」を参照してください。

## ハイブリッドノード CNI のトラブルシューティング

ハイブリッドノードで最初に Cilium または Calico を起動する際に問題が発生した場合は、多くの場合、ハイブリッドノードまたはハイブリッドノードで稼働する CNI ポッドと EKS コントロールプレーン間のネットワークの問題が原因です。環境が「ハイブリッドノードのネットワークを準備する」の要件を満たしていることを確認します。問題を分割すると分かりやすくなります。

### EKS クラスター設定

RemoteNodeNetwork と RemotePodNetwork の設定は正しいですか？

### VPC の構成

VPC ルーティングテーブルに、Transit Gateway または Virtual Private Gateway のターゲットを持つ RemoteNodeNetwork と RemotePodNetwork のルートはありますか？

### セキュリティグループの構成

RemoteNodeNetwork および RemotePodNetwork のインバウンドルールとアウトバウンドルールはありますか？

### オンプレミスのネットワーク

EKS コントロールプレーンとハイブリッドノード、およびハイブリッドノードで実行されているポッドとの間のルートとアクセスはありますか？

### CNI の設定

オーバーレイネットワークを使用している場合、CNI の IP プール設定は、ウェブフックを使用している場合に EKS クラスター用に設定された RemotePodNetwork と一致していますか？

### CNI がインストールされていないハイブリッドノードのステータス **Ready**

ハイブリッドノードのステータスが Ready と表示されているが、クラスターに CNI をインストールしていない場合は、ハイブリッドノードに古い CNI アーティファクトが存在する可能性があります。デフォルトでは、Helm などのツールを使用して Cilium と Calico をアンインストールしても、ディスク上のリソースは物理マシンや仮想マシンからは削除されません。さらに、これらの CNI のカスタムリソース定義 (CRD) は、古いインストールからクラスターにまだ存在する場合があります。詳細については、「[the section called “CNI を設定する”](#)」の「Cilium と Calico の削除」セクションを参照してください。

## Cilium のトラブルシューティング

ハイブリッドノードで Cilium の実行に問題がある場合は、Cilium ドキュメントの「[トラブルシューティング手順](#)」を参照してください。以下のセクションでは、ハイブリッドノードへの Cilium のデプロイに固有の問題について説明します。

### Cilium が起動しない

各ハイブリッドノードで実行されている Cilium エージェントが開始されていない場合は、Cilium エージェントポッドのログにエラーがないか確認します。Cilium エージェントを起動するには、EKS Kubernetes API エンドポイントへの接続が必要です。この接続が正しく設定されていない場合、Cilium エージェントの起動は失敗します。この場合、Cilium エージェントポッドログに次のようなログメッセージが表示されます。

```
msg="Unable to contact k8s api-server"
level=fatal msg="failed to start: Get \"https://<k8s-cluster-ip>:443/api/v1/namespaces/kube-system\": dial tcp <k8s-cluster-ip>:443: i/o timeout"
```

Cilium エージェントはホストネットワークで実行されます。EKS クラスターは、Cilium 接続用に RemoteNodeNetwork で設定する必要があります。EKS クラスターに RemoteNodeNetwork のインバウンドルールを持つ追加のセキュリティグループがあり、VPC に RemoteNodeNetwork のルートがあり、EKS コントロールプレーンへの接続を許可するようにオンプレミスネットワークが正しく設定されていることを確認します。

Cilium オペレータが実行されていて、一部の Cilium エージェントが動作しているが、すべてが動作していない場合は、クラスター内のすべてのノードに割り当てるポッド IP があることを確認します。Cilium 設定で clusterPoolIPv4PodCIDRList でクラスタープール IPAM を使用するとき、割り当て可能なポッド CIDR のサイズを設定します。ノードごとの CIDR サイズは、Cilium 設定の clusterPoolIPv4MaskSize 設定で設定されます。詳細については、Cilium ドキュメントの「[Expanding the cluster pool](#)」を参照してください。

### Cilium BGP が機能しない



Cilium BGP コントロールプレーンを使用して、ポッドまたはサービスアドレスをオンプレミスネットワークにアドバタイズする場合は、次の Cilium CLI コマンドを使用して、BGP がリソースへのルートをアドバタイズしているかどうかを確認できます。Cilium CLI をインストールする手順については、Cilium ドキュメントの「[Cilium CLI のインストール](#)」を参照してください。

BGP が正しく動作している場合は、ハイブリッドノードの出力に Session State established が含まれている必要があります。場合によっては、ネットワークチームと協力して、環境のローカル AS、ピア AS、ピアアドレスの正しい値を特定する必要があります。

```
cilium bgp peers
```

```
cilium bgp routes
```

Cilium BGP を使用してタイプ LoadBalancer のサービスの IP をアドバタイズする場合は、CiliumLoadBalancerIPPool と Service の両方に同じラベルが必要です。ラベルは、CiliumBGPAdvertisement のセレクトラで使用する必要があります。次に例を示します。Cilium BGP を使用して LoadBalancer タイプのサービスの IP をアドバタイズしている場合、Cilium エージェントの再起動中に BGP ルートが中断される可能性があります。詳細については、「Cilium documentation」の「[Failure Scenarios](#)」を参照してください。

## サービス

```
kind: Service
apiVersion: v1
metadata:
 name: guestbook
 labels:
 app: guestbook
spec:
 ports:
 - port: 3000
 targetPort: http-server
 selector:
 app: guestbook
 type: LoadBalancer
```

## CiliumLoadBalancerIPPool

```
apiVersion: cilium.io/v2alpha1
```



```
kind: CiliumLoadBalancerIPPool
metadata:
 name: guestbook-pool
 labels:
 app: guestbook
spec:
 blocks:
 - cidr: <CIDR to advertise>
 serviceSelector:
 matchExpressions:
 - { key: app, operator: In, values: [guestbook] }
```

## CiliumBGPAdvertisement

```
apiVersion: cilium.io/v2alpha1
kind: CiliumBGPAdvertisement
metadata:
 name: bgp-advertisements-guestbook
 labels:
 advertise: bgp
spec:
 advertisements:
 - advertisementType: "Service"
 service:
 addresses:
 - ExternalIP
 - LoadBalancerIP
 selector:
 matchExpressions:
 - { key: app, operator: In, values: [guestbook] }
```

## Calico のトラブルシューティング

ハイブリッドノードで Calico の実行に問題がある場合は、Calico ドキュメントの「[トラブルシューティング手順](#)」を参照してください。以下のセクションでは、Calico をハイブリッドノードにデプロイする際に特有の問題について説明します。

次の表は、Calico コンポーネントと、それらがデフォルトでノードネットワークまたはポッドネットワークで実行されているかどうかをまとめたものです。ポッドの送信トラフィックに NAT を使用するように Calico を設定した場合、オンプレミスネットワークはオンプレミスノード CIDR にトラフィックをルーティングするように設定する必要があり、VPC ルーティングテーブルはトランジットゲートウェイ (TGW) または仮想プライベートゲートウェイ (VGW) をターゲットとするオンプレ

ミスノード CIDR のルートで設定する必要があります。ポッドの送信トラフィックに NAT を使用するように Calico を設定していない場合は、オンプレミスネットワークがオンプレミスポッド CIDR にトラフィックをルーティングするように設定し、VPC ルーティングテーブルをトランジットゲートウェイ (TGW) または仮想プライベートゲートウェイ (VGW) をターゲットとするオンプレミスポッド CIDR のルートで設定する必要があります。

| コンポーネント                      | ネットワーク |
|------------------------------|--------|
| Calico API サーバー              | ノード    |
| Kubernetes 用の Calico コントローラー | ポッド    |
| Calico ノードエージェント             | ノード    |
| Calico typha                 | ノード    |
| Calico CSI ノードドライバ           | ポッド    |
| Calico 演算子                   | ノード    |

Calico リソースは、遮断されたノードでスケジュールまたは実行されています

DaemonSet として実行されない Calico リソースには、デフォルトで柔軟な許容範囲があり、ポッドのスケジュールや実行の準備が整っていない遮断されたノードでスケジュールできます。以下を含めるようにオペレータのインストールを変更することで、DaemonSet Calico 以外のリソースの許容範囲を厳しくすることができます。

```
installation:
 ...
 controlPlaneTolerations:
 - effect: NoExecute
 key: node.kubernetes.io/unreachable
 operator: Exists
 tolerationSeconds: 300
 - effect: NoExecute
 key: node.kubernetes.io/not-ready
 operator: Exists
 tolerationSeconds: 300
 calicoKubeControllersDeployment:
 spec:
```

```

template:
 spec:
 tolerations:
 - effect: NoExecute
 key: node.kubernetes.io/unreachable
 operator: Exists
 tolerationSeconds: 300
 - effect: NoExecute
 key: node.kubernetes.io/not-ready
 operator: Exists
 tolerationSeconds: 300
 typhaDeployment:
 spec:
 template:
 spec:
 tolerations:
 - effect: NoExecute
 key: node.kubernetes.io/unreachable
 operator: Exists
 tolerationSeconds: 300
 - effect: NoExecute
 key: node.kubernetes.io/not-ready
 operator: Exists
 tolerationSeconds: 300

```

## 認証情報のトラブルシューティング

AWS SSM ハイブリッドアクティベーションと AWS IAM Roles Anywhere の両方で、ハイブリッドノードから次のコマンドを実行して、ハイブリッドノード IAM ロールの認証情報がハイブリッドノードで正しく設定されていることを検証できます。ノード名とハイブリッドノード IAM ロール名が想定どおりであることを確認します。

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

```

{
 "UserId": "ABCDEFGHIJKLM12345678910:<node-name>",
 "Account": "<aws-account-id>",
 "Arn": "arn:aws:sts::<aws-account-id>:assumed-role/<hybrid-nodes-iam-role/<node-name>"
}

```

## AWS Systems Manager (SSM) のトラブルシューティング

ハイブリッドノードの認証情報に AWS SSM ハイブリッドアクティベーションを使用している場合は、nodeadm によってハイブリッドノードにインストールされている次の SSM ディレクトリとアーティファクトに注意してください。SSM エージェントの詳細については、「AWS Systems Manager ユーザーガイド」の「[SSM Agent の使用](#)」を参照してください。

| 説明            | 場所                                                                                                 |
|---------------|----------------------------------------------------------------------------------------------------|
| SSM エージェント    | Ubuntu - /snap/amazon-ssm-agent/current/amazon-ssm-agent RHEL & AL2023 - /usr/bin/amazon-ssm-agent |
| SSM Agent ログ  | /var/log/amazon/ssm                                                                                |
| AWS 認証情報      | /root/.aws/credentials                                                                             |
| SSM Setup CLI | /opt/ssm/ssm-setup-cli                                                                             |

## SSM エージェントを再起動する

SSM エージェントを再起動することで解決できる問題もあります。以下のコマンドを使用して再起動できます。

### AL2023 およびその他のオペレーティングシステム

```
systemctl restart amazon-ssm-agent
```

### Ubuntu

```
systemctl restart snap.amazon-ssm-agent.amazon-ssm-agent
```

## SSM エンドポイントへの接続を確認する

ハイブリッドノードから SSM エンドポイントに接続できることを確認します。SSM エンドポイントのリストについては、「[AWS Systems Manager endpoints and quotas](#)」を参照してください。以下のコマンドの us-west-2 を、AWS SSM ハイブリッドアクティベーションの AWS リージョンに置き換えます。

```
ping ssm.us-west-2.amazonaws.com
```

## 登録された SSM インスタンスの接続ステータスを表示する

次の AWS CLI コマンドを使用して、SSM ハイブリッドアクティベーションに登録されているインスタンスの接続ステータスを確認できます。マシン ID をインスタンスのマシン ID に置き換えます。

```
aws ssm get-connection-status --target mi-012345678abcdefgh
```

## SSM セットアップ CLI チェックサムの不一致

nodeadm install の実行時に、ssm-setup-cli チェックサムの不一致に問題がある場合は、ホストに古い既存の SSM インストールがないことを確認する必要があります。ホストに古い SSM インストールがある場合は、それらを削除して nodeadm install を再実行し、問題を解決します。

```
Failed to perform agent-installation/on-prem registration: error while verifying
installed ssm-setup-cli checksum: checksum mismatch with latest ssm-setup-cli.
```

## SSM InvalidActivation

インスタンスを AWS SSM に登録する際にエラーが表示された場合は、nodeConfig.yaml の region、activationCode、および activationId が正しいことを確認します。EKS クラスターの AWS リージョンは、SSM ハイブリッドアクティベーションのリージョンと一致する必要があります。これらの値が誤って設定されている場合は、次のようなエラーが表示されることがあります。

```
ERROR Registration failed due to error registering the instance with AWS SSM.
InvalidActivation
```

**SSM ExpiredTokenException:** リクエストに含まれているセキュリティトークンが有効期限切れです

SSM エージェントが認証情報を更新できない場合は、「ExpiredTokenException」が表示されることがあります。このシナリオでは、ハイブリッドノードから SSM エンドポイントに接続できる場合、認証情報の更新を強制するために SSM エージェントを再起動する必要がある場合があります。

```
"msg":"Command failed","error":"operation error SSM: DescribeInstanceInformation, https
response error StatusCode: 400, RequestID: eee03a9e-f7cc-470a-9647-73d47e4cf0be, api
error ExpiredTokenException: The security token included in the request is expired"
```

## register machine コマンドの実行中の SSM エラー

マシンを SSM に登録する際にエラーが発生した場合は、`nodeadm install` を再実行して、すべての SSM 依存関係が正しくインストールされていることを確認する必要があります。

```
"error": "running register machine command: , error: fork/exec /opt/aws/ssm-setup-cli:
no such file or directory"
```

## SSM ActivationExpired

`nodeadm init` の実行時に、アクティベーションの有効期限が切れたために SSM へのインスタンスの登録がエラーになった場合は、新しい SSM ハイブリッドアクティベーションを作成し、新しい SSM ハイブリッドアクティベーションの `activationCode` と `activationId` で `nodeConfig.yaml` を更新して、`nodeadm init` を再実行する必要があります。

```
"msg": "Command failed", "error": "SSM activation expired. Please use a valid activation"
```

```
ERROR Registration failed due to error registering the instance with AWS SSM.
ActivationExpired
```

## SSM がキャッシュされた認証情報を更新できませんでした

キャッシュされた認証情報の更新に失敗した場合、`/root/.aws/credentials` ファイルはホストで削除されている可能性があります。まず、SSM ハイブリッドアクティベーションをチェックして、アクティブであること、およびアクティベーションを使用するようにハイブリッドノードが正しく設定されていることを確認します。`/var/log/amazon/ssm` で SSM エージェントログを確認し、SSM 側で問題を解決したら `nodeadm init` コマンドを再実行します。

```
"Command failed", "error": "operation error SSM: DescribeInstanceInformation, get
identity: get credentials: failed to refresh cached credentials"
```

## SSM のクリーンアップ

ホストから SSM エージェントを削除するには、次のコマンドを実行します。

```
dnf remove -y amazon-ssm-agent
sudo apt remove --purge amazon-ssm-agent
snap remove amazon-ssm-agent
```

```
rm -rf /var/lib/amazon/ssm/Vault/Store/RegistrationKey
```

## AWS IAM Roles Anywhere のトラブルシューティング

ハイブリッドノードの認証情報に AWS IAM Roles Anywhere を使用している場合は、nodeadm によってハイブリッドノードにインストールされている次のディレクトリとアーティファクトに注意してください。IAM Roles Anywhere のトラブルシューティングの詳細については、「AWS IAM Roles Anywhere User Guide」の「[Troubleshooting AWS IAM Roles Anywhere identity and access](#)」を参照してください。

| 説明                     | 場所                                |
|------------------------|-----------------------------------|
| IAM Roles Anywhere CLI | /usr/local/bin/aws_signing_helper |
| デフォルトの証明書の場所と名前        | /etc/iam/pki/server.pem           |
| デフォルトのキーの場所と名前         | /etc/iam/pki/server.key           |

### IAM Roles Anywhere がキャッシュされた認証情報の更新に失敗しました

キャッシュされた認証情報の更新に失敗した場合は、/etc/aws/hybrid/config の内容を確認し、nodeadm 設定で IAM Roles Anywhere が正しく設定されていることを確認します。/etc/iam/pki が存在することを確認します。各ノードには一意の証明書とキーが必要です。デフォルトでは、認証情報プロバイダーとして IAM Roles Anywhere を使用する場合、nodeadm は証明書の場所と名前に /etc/iam/pki/server.pem を使用し、プライベートキーに /etc/iam/pki/server.key を使用します。sudo mkdir -p /etc/iam/pki を使用してディレクトリに証明書とキーを配置する前に、ディレクトリの作成が必要になる場合があります。証明書の内容は、以下のコマンドで確認できます。

```
openssl x509 -text -noout -in server.pem
```

```
open /etc/iam/pki/server.pem: no such file or directory
could not parse PEM data
Command failed {"error": "... get identity: get credentials: failed to refresh cached
credentials, process provider error: error in credential_process: exit status 1"}
```

### sts:AssumeRole の実行が許可されていない IAM Roles Anywhere

kubelet ログで、IAM Roles Anywhere の使用時に `sts:AssumeRole` オペレーションに対するアクセス拒否の問題が表示された場合は、ハイブリッドノード IAM ロールの信頼ポリシーをチェックして、IAM Roles Anywhere サービスプリンシパルがハイブリッドノード IAM ロールを引き受けることができることを確認します。さらに、ハイブリッドノード IAM ロールの信頼ポリシーでトラストアンカー ARN が正しく設定されていること、およびハイブリッドノード IAM ロールが IAM Roles Anywhere プロファイルに追加されていることを確認します。

```
could not get token: AccessDenied: User: ... is not authorized to perform:
sts:AssumeRole on resource: ...
```

### **roleSessionName** を設定することが許可されていない IAM Roles Anywhere

kubelet ログで、`roleSessionName` の設定にアクセス拒否の問題が発生した場合は、IAM Roles Anywhere プロファイルで `acceptRoleSessionName` を `true` に設定していることを確認します。

```
AccessDeniedException: Not authorized to set roleSessionName
```

## オペレーティングシステムのトラブルシューティング

### RHEL

#### 使用権限管理者またはサブスクリプションマネージャーの登録失敗

`nodeadm install` を実行中で、使用権限登録の問題によりハイブリッドノードの依存関係のインストールに失敗した場合、ホストに Red Hat ユーザー名とパスワードを適切に設定していることを確認してください。

```
This system is not registered with an entitlement server
```

### Ubuntu

#### GLIBC が見つかりません

オペレーティングシステムに Ubuntu を使用し、ハイブリッドノードを持つ認証情報プロバイダーに IAM Roles Anywhere を使用していて、GLIBC が見つからない問題がある場合は、その依存関係を手動でインストールすることで問題を解決できます。

```
GLIBC_2.32 not found (required by /usr/local/bin/aws_signing_helper)
```



次のコマンドを実行して依存関係をインストールします。

```
ldd --version
sudo apt update && apt install libc6
sudo apt install glibc-source
```

## Bottlerocket

Bottlerocket 管理コンテナを有効にしている場合、SSH を使用してこれにアクセスし、昇格された権限を使って高度なデバッグとトラブルシューティングを行うことができます。以下のセクションでは、Bottlerocket ホストのコンテキストで実行する必要があるコマンドを記載しています。管理コンテナにアクセスすると、sheltie を実行すれば Bottlerocket ホストで完全なルートシェルを取得することができます。

```
sheltie
```

以下のセクションにあるコマンドは、各コマンドの前に `sudo chroot /.bottlerocket/rootfs` を付けることで管理者コンテナシェルから実行することもできます。

```
sudo chroot /.bottlerocket/rootfs <command>
```

## ログ収集に logdog を使用する

Bottlerocket には、トラブルシューティングを目的に、ログとシステム情報とを効率的に収集する logdog ユーティリティがあります。

```
logdog
```

logdog ユーティリティは、Bottlerocket ホストのさまざまな場所からログを収集し、それらを tarball と組み合わせます。デフォルトでは、tarball は `/var/log/support/bottlerocket-logs.tar.gz` で作成され、`/.bottlerocket/support/bottlerocket-logs.tar.gz` にあるホストコンテナからアクセスすることができます。

## journalctl を使用したシステムログへのアクセス

次のコマンドを使用することで、kubelet、containerd などのさまざまなシステムサービスのステータスをチェックしたりログを表示したりできます。`-f` フラグは、ログをリアルタイムでフォローします。

kubelet サービスのステータスを確認し kubelet ログを取得するときは、以下を実行します。

```
systemctl status kubelet
journalctl -u kubelet -f
```

containerd サービスのステータスを確認し、オーケストレーションされた containerd インスタンスのログを取得するときは、以下を実行します。

```
systemctl status containerd
journalctl -u containerd -f
```

host-containerd サービスのステータスを確認し、ホスト containerd インスタンスのログを取得するときは、以下を実行します。

```
systemctl status host-containerd
journalctl -u host-containerd -f
```

ブートストラップコンテナとホストコンテナのログを取得するときは、以下を実行します。

```
journalctl _COMM=host-ctr -f
```

# クラスターのアプリケーションデータストレージを使用する

アプリケーションのストレージニーズに応じて、Amazon EKS でさまざまな AWS ストレージサービスを使用できます。AWS でサポートされている幅広い Container Storage Interface (CSI) ドライバーを使用すると、Amazon EKS で実行されているアプリケーションのストレージニーズに応じて、Amazon EBS、Amazon S3、Amazon EFS、Amazon FSX、および Amazon File Cache を簡単に使用できます。

この章では、Amazon EKS クラスターのストレージオプションについて説明します。

## トピック

- [Amazon EBS で Kubernetes ボリュームストレージを使用する](#)
- [Amazon EFS で Elastic File System ストレージを使用する](#)
- [Amazon FSx for Lustre で高性能なアプリケーションストレージを使用する](#)
- [FSx for NetApp ONTAP で高性能なアプリケーションストレージを使用する](#)
- [Amazon FSx for OpenZFS でデータストレージを使用する](#)
- [Amazon File Cache を使用してレイテンシーを最小化する](#)
- [Mountpoint for Amazon S3 CSI ドライバーを使用して Amazon S3 オブジェクトにアクセスする](#)
- [CSI ボリュームのためにスナップショット機能を有効にする](#)

## Amazon EBS で Kubernetes ボリュームストレージを使用する

### Note

新規: Amazon EKS Auto Mode は、ブロックストレージのルーチンタスクを自動化します。  
「[the section called “ステートフルワークロードをデプロイする”](#)」ではその方法を説明しています。

[Amazon Elastic Block Store \(Amazon EBS\) コンテナストレージインターフェイス \(CSI\) ドライバー](#)は、Amazon EBS ボリュームのライフサイクルを、作成した Kubernetes ボリュームのストレージとして管理します。Amazon EBS CSI ドライバーは、ジェネリック「[エフェメラルボリューム](#)」と「[永続ボリューム](#)」という種類の Kubernetes ボリューム用の Amazon EBS ボリュームを作成します。

## 考慮事項

- EKS Auto Mode クラスターに Amazon EBS CSI コントローラーをインストールする必要はありません。
- Amazon EBS ボリュームを Fargate Pod にマウントすることはできません。
- Amazon EBS CSI コントローラーは Fargate ノードで実行できますが、Amazon EBS CSI ノード DaemonSet は Amazon EC2 インスタンスでのみ実行できます。
- Amazon EBS ボリュームと Amazon EBS CSI ドライバーは Amazon EKS Hybrid Nodes と互換性がありません。
- 最新のアドオンバージョンと 1 つの以前のバージョンがサポートされます。最新バージョンで見つかったバグや脆弱性は、新しいマイナーバージョンの以前のリリースにバックポートされます。
- EKS Auto Mode では、プロビジョナーとして `ebs.csi.eks.amazonaws.com` を使用するストレージクラスが必要です。標準の Amazon EBS CSI ドライバー (`ebs.csi.aws.com`) は、独自のボリュームを個別に管理します。EKS Auto Mode で既存のボリュームを使用するには、ボリュームスナップショットを使用して、Auto Mode プロビジョナーを使用するストレージクラスに移行します。

### Important

Amazon EBS CSI ドライバーのスナップショット機能を使用するには、まず CSI スナップショットコントローラーをインストールする必要があります。詳細については、「[the section called “CSI スナップショットコントローラー”](#)」を参照してください。

## 前提条件

- 既存のクラスター。必要なプラットフォームのバージョンを確認するには、次のコマンドを実行します。

```
aws eks describe-addon-versions --addon-name aws-ebs-csi-driver
```

- EBS CSI ドライバーには AWS IAM アクセス許可が必要です。
  - AWS では、EKS Pod Identity の使用を推奨しています。詳細については、「[the section called “EKS Pod Identity のセットアップの概要”](#)」を参照してください。

- サービスアカウントの IAM ロールの詳細については、「[the section called “IAM OIDC プロバイダー”](#)」を参照してください。

## ステップ 1: IAM ロールを作成する

Amazon EBS CSI プラグインでは、ユーザーに代わって AWS API の呼び出しを行うための IAM アクセス許可が必要です。これらのステップを実行しない場合、アドオンをインストールして `kubectl describe pvc` を実行しようとする、`could not create volume in EC2: UnauthorizedOperation` エラーとともに `failed to provision volume with StorageClass` が表示されます。詳細については、GitHub 上の [Set up driver permission](#) を参照してください。

### Note

Pod は、IMDS へのアクセスをブロックする場合を除き、IAM ロールに割り当てられたアクセス許可にアクセスできます。詳細については、「[the section called “ベストプラクティス”](#)」を参照してください。

以下の手順は、IAM ロールを作成し、それに AWS マネージドポリシーをアタッチする方法を示しています。この手順を実装するには、次のいずれかのツールを使用できます。

- [the section called “eksctl”](#)
- [the section called “AWS Management Console”](#)
- [the section called “AWS CLI”](#)

### Note

この手順には、ドライバーを Amazon EKS アドオンとして使用するための特定のステップが書かれています。ドライバーをセルフマネージドのアドオンとして使用するには、さまざまなステップが必要です。詳細については、GitHub の「[Set up driver permissions](#)」を参照してください。

## eksctl

1. IAM ロールを作成して、ポリシーをアタッチします。AWS が AWS マネージドポリシーを維持しますが、ユーザーが独自のカスタムポリシーを作成することもできます。次のコマンドを使用して、IAM ロールを作成し、それに AWS マネージドポリシーをアタッチできます。**#####** の部分は自分のクラスター名に置き換えます。このコマンドは IAM ロールを作成して IAM ポリシーをアタッチする AWS CloudFormation スタックをデプロイします。

```
eksctl create iamserviceaccount \
 --name ebs-csi-controller-sa \
 --namespace kube-system \
 --cluster my-cluster \
 --role-name AmazonEKS_EBS_CSI_DriverRole \
 --role-only \
 --attach-policy-arn arn:aws:iam::aws:policy/service-role/
AmazonEBSCSIDriverPolicy \
 --approve
```

2. カスタム [KMS キー](#) を使用しない場合は、このステップをスキップできます。このキーを Amazon EBS ボリュームでの暗号化に使用する場合、必要に応じて IAM ロールをカスタマイズします。例えば、以下を実行してください。
  - a. 次のコードをコピーし、新しい kms-key-for-encryption-on-ebs.json ファイルに貼り付けます。**custom-key-arn** をカスタム [KMS key ARN](#) に置き換えます。

```
{
 "Version": "2012-10-17",
 "Statement": [
 {
 "Effect": "Allow",
 "Action": [
 "kms:CreateGrant",
 "kms:ListGrants",
 "kms:RevokeGrant"
],
 "Resource": ["custom-key-arn"],
 "Condition": {
 "Bool": {
 "kms:GrantIsForAWSResource": "true"
 }
 }
 }
],
}
```

```
{
 "Effect": "Allow",
 "Action": [
 "kms:Encrypt",
 "kms:Decrypt",
 "kms:ReEncrypt*",
 "kms:GenerateDataKey*",
 "kms:DescribeKey"
],
 "Resource": ["custom-key-arn"]
}
```

- b. ポリシーを作成します。*KMS\_Key\_For\_Encryption\_On\_EBS\_Policy* を別の名前に変更できます。ただし、変更する場合は、必ず後の手順で変更してください。

```
aws iam create-policy \
 --policy-name KMS_Key_For_Encryption_On_EBS_Policy \
 --policy-document file://kms-key-for-encryption-on-ebs.json
```

- c. 次のコマンドを使用して、IAM ポリシーをロールに添付します。*111122223333* は、ご自分のアカウント ID に置き換えます。

```
aws iam attach-role-policy \
 --policy-arn arn:aws:iam::111122223333:policy/
KMS_Key_For_Encryption_On_EBS_Policy \
 --role-name AmazonEKS_EBS_CSI_DriverRole
```

## AWS Management Console

1. IAM コンソール (<https://console.aws.amazon.com/iam/>) を開きます。
2. 左のナビゲーションペインで、[ロール] を選択してください。
3. [ロール] ページで、[ロールの作成] を選択してください。
4. [信頼されたエンティティを選択] ページで、以下の操作を実行します。
  - a. [信頼されたエンティティの種類] セクションで、[ウェブ アイデンティティ] を選択します。
  - b. [Identity provider] (ID プロバイダー) で、(Amazon EKS の [Overview] (概要) に示されているように) クラスターに [OpenID Connect provider URL] (OpenID Connect プロバイダーの URL) を選択します。

- c. [対象者] で `sts.amazonaws.com` を選択します。
- d. [次へ] を選択します。
5. [アクセス許可を追加] ページで、以下を実行します。
  - a. [フィルタポリシー] ボックスに `AmazonEBSCSIDriverPolicy` と入力します。
  - b. 検索で返された `AmazonEBSCSIDriverPolicy` の左にあるチェックボックスを選択します。
  - c. [次へ] を選択します。
6. [名前を付けて、レビューし、作成する] ページで、以下の操作を実行します。
  - a. [ロール名] に、`AmazonEKS_EBS_CSI_DriverRole` などのロールの一意の名前を入力します。
  - b. [タグの追加 (オプション)] で、タグをキーバリューのペアとして添付して、メタデータをロールに追加します。IAM でのタグの使用に関する詳細については『IAM ユーザーガイド』の「[IAM リソースにタグを付ける](#)」を参照してください。
  - c. [ロールの作成] を選択してください。
7. ロールが作成されたら、コンソールでロールを選択して編集用を開きます。
8. [信頼関係] タブを選択し、続いて [信頼ポリシーの編集] を選択します。
9. 次の行と似ている行を探しています。

```
"oidc.eks.region-code.amazonaws.com/id/EXAMPLED539D4633E53DE1B71EXAMPLE:aud":
 "sts.amazonaws.com"
```

前の行の末尾にカンマを追加し、前の行の後に次の行を追加します。`region-code` を、クラスターのある AWS リージョンに置き換えます。`EXAMPLED539D4633E53DE1B71EXAMPLE` をクラスターの OIDC プロバイダー ID に置き換えます。

```
"oidc.eks.region-code.amazonaws.com/id/EXAMPLED539D4633E53DE1B71EXAMPLE:sub":
 "system:serviceaccount:kube-system:ebs-csi-controller-sa"
```

- 10[ポリシーの更新] を選択して終了します。
- 11Amazon EBS ボリュームでの暗号化にカスタム「[KMS キー](#)」を使用する場合、必要に応じて IAM ロールをカスタマイズします。例えば、以下を実行してください。
  - a. 左のナビゲーションペインの [ポリシー] を選択します。
  - b. [ポリシー] ページで、[ポリシーの作成] を選択します。
  - c. [ポリシーの作成] ページで、[JSON] タブを選択します。



- d. 次のコードをコピーしてエディタに貼り付け、*custom-key-arn* をカスタム [KMS key ARN](#) に置き換えます。

```
{
 "Version": "2012-10-17",
 "Statement": [
 {
 "Effect": "Allow",
 "Action": [
 "kms:CreateGrant",
 "kms:ListGrants",
 "kms:RevokeGrant"
],
 "Resource": ["custom-key-arn"],
 "Condition": {
 "Bool": {
 "kms:GrantIsForAWSResource": "true"
 }
 }
 },
 {
 "Effect": "Allow",
 "Action": [
 "kms:Encrypt",
 "kms:Decrypt",
 "kms:ReEncrypt*",
 "kms:GenerateDataKey*",
 "kms:DescribeKey"
],
 "Resource": ["custom-key-arn"]
 }
]
}
```

- e. [Next: Tags] (次へ: タグ) を選択します。
- f. [タグの追加 (オプション)] ページで、[次へ: 確認] を選択します。
- g. 名前 には、ポリシーの一意の名前を入力します (例:  
*KMS\_Key\_For\_Encryption\_On\_EBS\_Policy*)。
- h. [Create policy] (ポリシーの作成) を選択します。
- i. 左のナビゲーションペインで、[ロール] を選択してください。
- j. コンソールで ***AmazonEKS\_EBS\_CSI\_DriverRole*** を選択し、編集用を開きます。

- k. [アクセス許可を追加] ドロップダウンリストから [ポリシーをアタッチ] を選択します。
- l. [フィルターポリシー] ボックスに、*KMS\_Key\_For\_Encryption\_On\_EBS\_Policy* と入力します。
- m. 検索で返された *KMS\_Key\_For\_Encryption\_On\_EBS\_Policy* の左側にあるチェックボックスをオンにします。
- n. [ポリシーのアタッチ] を選択します。

## AWS CLI

1. クラスターの OIDC プロバイダーの URL を表示します。##### の部分は自分のクラスター名に置き換えます。コマンドの出力が None の場合は、「前提条件」を確認してください。

```
aws eks describe-cluster --name my-cluster --query "cluster.identity.oidc.issuer" --output text
```

出力例は次のとおりです。

```
https://oidc.eks.region-code.amazonaws.com/id/EXAMPLED539D4633E53DE1B71EXAMPLE
```

2. IAM ロールを作成して AssumeRoleWithWebIdentity アクションを付与します。
  - a. 次の内容を aws-ebs-csi-driver-trust-policy.json という名前のファイルにコピーします。*111122223333* は、ご自分のアカウント ID に置き換えます。*EXAMPLED539D4633E53DE1B71EXAMPLE* および *region-code* を、前のステップで返された値にそれぞれ置き換えます。

```
{
 "Version": "2012-10-17",
 "Statement": [
 {
 "Effect": "Allow",
 "Principal": {
 "Federated": "arn:aws:iam::111122223333:oidc-provider/oidc.eks.region-code.amazonaws.com/id/EXAMPLED539D4633E53DE1B71EXAMPLE"
 },
 "Action": "sts:AssumeRoleWithWebIdentity",
 "Condition": {
 "StringEquals": {
```

```

 "oidc.eks.region-code.amazonaws.com/id/
EXAMPLED539D4633E53DE1B71EXAMPLE:aud": "sts.amazonaws.com",
 "oidc.eks.region-code.amazonaws.com/id/
EXAMPLED539D4633E53DE1B71EXAMPLE:sub": "system:serviceaccount:kube-system:ebs-csi-
controller-sa"
 }
}
]
}

```

- b. ロールを作成します。*AmazonEKS\_EBS\_CSI\_DriverRole* を別の名前に変更できます。変更する場合は、必ず後の手順で変更してください。

```

aws iam create-role \
 --role-name AmazonEKS_EBS_CSI_DriverRole \
 --assume-role-policy-document file://aws-ebs-csi-driver-trust-policy.json

```

3. ポリシーをアタッチします。AWS が AWS マネージドポリシーを維持しますが、ユーザーが独自のカスタムポリシーを作成することもできます。次のコマンドを使用して、AWS マネージドポリシーをロールにアタッチします。

```

aws iam attach-role-policy \
 --policy-arn arn:aws:iam::aws:policy/service-role/AmazonEBSCSIDriverPolicy \
 --role-name AmazonEKS_EBS_CSI_DriverRole

```

4. Amazon EBS ボリュームでの暗号化にカスタム「[KMS キー](#)」を使用する場合、必要に応じて IAM ロールをカスタマイズします。例えば、以下を実行してください。

- a. 次のコードをコピーし、新しい kms-key-for-encryption-on-ebs.json ファイルに貼り付けます。*custom-key-arn* をカスタム [KMS key ARN](#) に置き換えます。

```

{
 "Version": "2012-10-17",
 "Statement": [
 {
 "Effect": "Allow",
 "Action": [
 "kms:CreateGrant",
 "kms:ListGrants",
 "kms:RevokeGrant"
],
 "Resource": ["custom-key-arn"],

```

```

 "Condition": {
 "Bool": {
 "kms:GrantIsForAWSResource": "true"
 }
 },
 {
 "Effect": "Allow",
 "Action": [
 "kms:Encrypt",
 "kms:Decrypt",
 "kms:ReEncrypt*",
 "kms:GenerateDataKey*",
 "kms:DescribeKey"
],
 "Resource": ["custom-key-arn"]
 }
]
}

```

- b. ポリシーを作成します。**KMS\_Key\_For\_Encryption\_On\_EBS\_Policy** を別の名前に変更できます。ただし、変更する場合は、必ず後の手順で変更してください。

```

aws iam create-policy \
 --policy-name KMS_Key_For_Encryption_On_EBS_Policy \
 --policy-document file://kms-key-for-encryption-on-ebs.json

```

- c. 次のコマンドを使用して、IAM ポリシーをロールに添付します。**111122223333** は、ご自分のアカウント ID に置き換えます。

```

aws iam attach-role-policy \
 --policy-arn arn:aws:iam::111122223333:policy/KMS_Key_For_Encryption_On_EBS_Policy \
 --role-name AmazonEKS_EBS_CSI_DriverRole

```

Amazon EBS CSI ドライバーの IAM ロールを作成したので、次のセクションに進むことができます。この IAM ロールを使用してアドオンをデプロイすると、`ebs-csi-controller-sa` という名前のサービスアカウントが作成され、それを使用されるように設定されます。サービスアカウントは `Kubernetes clusterrole` にバインドされます。これには、必要な Kubernetes アクセス許可が割り当てられています。

## ステップ 2: Amazon EBS CSI ドライバーを入手する

セキュリティを強化し、作業量を削減するために、Amazon EKS アドオンを通じて Amazon EBS CSI ドライバーをインストールすることをお勧めします。Amazon EKS アドオンをクラスターに追加するには、「[the section called “アドオンを作成する”](#)」を参照してください。アドオンの詳細については、「[the section called “Amazon EKS アドオン”](#)」を参照してください。

### Important

Amazon EBS ドライバーを Amazon EKS アドオンとして追加する前に、クラスターにセルフマネージドバージョンのドライバーがインストールされていないことを確認してください。インストールされている場合は、GitHub の「[Uninstalling a self-managed Amazon EBS CSI driver](#)」を参照してください。

または、Amazon EBS CSI ドライバーのセルフマネージドインストールが必要な場合は、GitHub の「[Installation](#)」を参照してください。

## ステップ 3: サンプルアプリケーションをデプロイする

さまざまなサンプルアプリケーションをデプロイし、必要に応じて変更できます。詳細については、GitHub の「[Kubernetes Examples](#)」を参照してください。

## Amazon EFS で Elastic File System ストレージを使用する

[Amazon Elastic File System](#) (Amazon EFS) は、サーバーレスで伸縮自在なファイルストレージを提供するため、ストレージ容量およびパフォーマンスのプロビジョニングや管理を行うことなくファイルデータを共有できます。[Amazon EFS Container Storage Interface \(CSI\) ドライバー](#)は、AWS で動作する Kubernetes クラスターが Amazon EFS ファイルシステムのライフサイクルを管理できるようにする CSI インターフェイスを提供します。このトピックでは、Amazon EFS CSI ドライバーを Amazon EKS クラスターにデプロイする方法を示します。

### 考慮事項

- Amazon EFS CSI ドライバーは、Windows ベースのコンテナイメージと互換性がありません。
- Fargate ノードでは永続ボリュームのために[動的プロビジョニング](#)を使用できませんが、[静的プロビジョニング](#)は使用できます。

- [動的プロビジョニング](#)には、[1.2](#)以降のドライバーが必要です。サポートされている Amazon EKS クラスターのバージョンのドライバーのバージョン 1.1 を使用して、永続ボリュームのために[静的プロビジョニング](#)を使用できます ([the section called “Kubernetes バージョン”](#) を参照)。
- このドライバーのバージョン [1.3.2](#) 以降では、Amazon EC2 Graviton ベースのインスタンスを含む Arm64 アーキテクチャがサポートされています。
- バージョン [1.4.2](#) 以降のドライバーでは、ファイルシステムのマウントに FIPS を使用できます。
- Amazon EFS のリソースクォータに注意してください。例えば、Amazon EFS ファイルシステムごとに作成できるアクセスポイントのクォータは 1,000 です。詳細については、「[変更できない Amazon EFS リソースクォータ](#)」を参照してください。
- バージョン [2.0.0](#) 以降、このドライバーは TLS 接続に stunnel から efs-proxy を使用するようになり切り替えました。efs-proxy を使用すると、実行中のノードのコア数に 1 を足した数のスレッドが開きます。
- Amazon EFS CSI ドライバーは Amazon EKS Hybrid Nodes と互換性がありません。

## 前提条件

- Amazon EFS CSI ドライバーには AWS Identity and Access Management (IAM) のアクセス権限が必要です。
  - AWS では、EKS Pod Identity の使用を推奨しています。詳細については、「[the section called “EKS Pod Identity のセットアップの概要”](#)」を参照してください。
  - サービスアカウントの IAM ロールと、クラスターの IAM OpenID Connect (OIDC) プロバイダーの設定については、「[the section called “IAM OIDC プロバイダー”](#)」で詳しく確認できます。
- ご使用のデバイスまたは AWS クラウドシェル で、バージョン 2.12.3 以降、または AWS コマンドラインインターフェイス (AWS CLI のバージョン 1.27.160 以降がインストールおよび設定されていること。現在のバージョンを確認するには「aws --version | cut -d / -f2 | cut -d ' ' -f1」を参照してください。yum、apt-get、macOS 用の Homebrew などのパッケージマネージャーは、多くの場合 AWS CLI の最新バージョンより数バージョン古くなっています。最新バージョンをインストールするには「AWS コマンドラインインターフェイスユーザーガイド」の「[インストール](#)」および「[aws configure を使用したクイック設定](#)」を参照してください。AWS クラウドシェルにインストールされている AWS CLI バージョンも最新バージョンより数バージョン遅れることがあります。更新するには「AWS クラウドシェル ユーザーガイド」の「[ホームディレクトリへの AWS CLI のインストール](#)」を参照してください。
- デバイスまたは AWS クラウドシェル に、kubectl コマンドラインツールがインストールされていること。バージョンはご使用のクラスターの Kubernetes バージョンと同じか、1 つ前のマイ

ナーバージョン以前、あるいはそれより新しいバージョンが使用できます。例えば、クラスターのバージョンが 1.29 である場合、kubectl のバージョン 1.28、1.29、または 1.30 が使用できます。kubectl をインストールまたはアップグレードする方法については「[the section called “kubectl と eksctl のセットアップ”](#)」を参照してください。

#### Note

Fargate 上で実行される Pod は、ドライバーの手動インストールステップなしで、Amazon EFS ファイルシステムを自動的にマウントします。

## ステップ 1: IAM ロールを作成する

Amazon EFS CSI ドライバーには、ファイルシステムと対話するための IAM アクセス許可が必要です。IAM ロールを作成して、必要な AWS 管理ポリシーをアタッチします。この手順を実装するには、次のいずれかのツールを使用できます。

- [the section called “eksctl”](#)
- [the section called “AWS Management Console”](#)
- [the section called “AWS CLI”](#)

#### Note

この手順には、ドライバーを Amazon EKS アドオンとして使用するための特定のステップが書かれています。セルフマネージド型インストールについての詳細は、GitHub の「[Set up driver permission](#)」を参照してください。

## eksctl

Pod Identity を使用する場合

以下のコマンドを実行します。ここでは、eksctl を使用して IAM ロールと、Pod Identity の関連付けを作成しています。my-cluster をクラスター名に置き換えます。AmazonEKS\_EFS\_CSI\_DriverRole を別の名前に置き換えることもできます。

```
export cluster_name=my-cluster
```

```
export role_name=AmazonEKS_EFS_CSI_DriverRole
eksctl create podidentityassociation \
 --service-account-name efs-csi-controller-sa \
 --namespace kube-system \
 --cluster $cluster_name \
 --role-name $role_name \
 --permission-policy-arns arn:aws:iam::aws:policy/service-role/
AmazonEFSCSIDriverPolicy \
 --approve
```

## サービスアカウントに IAM ロールを使用している場合

次のコマンドを実行して、eksctl を使用して IAM ロールを作成します。my-cluster をクラスター名に置き換えます。AmazonEKS\_EFS\_CSI\_DriverRole を別の名前に置き換えることもできます。

```
export cluster_name=my-cluster
export role_name=AmazonEKS_EFS_CSI_DriverRole
eksctl create iamserviceaccount \
 --name efs-csi-controller-sa \
 --namespace kube-system \
 --cluster $cluster_name \
 --role-name $role_name \
 --role-only \
 --attach-policy-arn arn:aws:iam::aws:policy/service-role/AmazonEFSCSIDriverPolicy \
 --approve
TRUST_POLICY=$(aws iam get-role --output json --role-name $role_name --query
 'Role.AssumeRolePolicyDocument' | \
 sed -e 's/efs-csi-controller-sa/efs-csi-*/' -e 's/StringEquals/StringLike/')
aws iam update-assume-role-policy --role-name $role_name --policy-document
"$TRUST_POLICY"
```

## AWS Management Console

次のコマンドを実行して、AWS Management Console を使用して IAM ロールを作成します。

1. IAM コンソール (<https://console.aws.amazon.com/iam/>) を開きます。
2. 左のナビゲーションペインで、[ロール] を選択してください。
3. [ロール] ページで、[ロールの作成] を選択してください。
4. [信頼されたエンティティを選択] ページで、以下の操作を実行してください：
  - a. EKS Pod Identity を使用する場合



- i. [信頼するエンティティのタイプ] で [AWS サービス] を選択してください。
    - ii. [サービスまたはユースケース] のドロップダウンで、EKS を選択します。
    - iii. ユースケースセクションで、[EKS - Pod Identity] を選択します。
    - iv. [次へ] を選択します。
  - b. サービスアカウントに IAM ロールを使用している場合
    - i. [信頼されたエンティティの種類] セクションで、[ウェブ アイデンティティ] を選択します。
    - ii. [Identity provider] (ID プロバイダー) で、(Amazon EKS の [Overview] (概要) に示されているように) クラスターに [OpenID Connect provider URL] (OpenID Connect プロバイダーの URL) を選択します。
    - iii. [対象者] で [sts.amazonaws.com] を選択します。
    - iv. [次へ] を選択します。
5. [アクセス許可を追加] ページで、以下を実行します。
- a. [フィルタポリシー] ボックスに AmazonEFSCSIDriverPolicy と入力します。
  - b. 検索で返された AmazonEFSCSIDriverPolicy の左にあるチェックボックスを選択します。
  - c. [次へ] を選択します。
6. [名前を付けて、レビューし、作成する] ページで、以下の操作を実行します。
- a. [ロール名] に、AmazonEKS\_EFS\_CSI\_DriverRole などのロールの一意の名前を入力します。
  - b. [タグの追加 (オプション)] で、タグをキーバリューのペアとして添付して、メタデータをロールに追加します。IAM でのタグの使用に関する詳細については『IAM ユーザーガイド』の「[IAM リソースにタグを付ける](#)」を参照してください。
  - c. [ロールの作成] を選択してください。
7. ロールの作成後
- a. EKS Pod Identity を使用する場合
    - i. [Amazon EKS コンソール](#)を開きます。
    - ii. 左のナビゲーションペインで、[クラスター] を選択して、EKS Pod Identity の関連付けを設定するクラスターの名前を選択します。
    - iii. [リモートアクセス] タブを選択してください。
    - iv. [Pod Identity の関連付け] で [作成] を選択します。
    - v. [IAM ロール] ドロップダウンを選択して、新しく作成したロールを選択します。
  - vi. [Kubernetes 名前空間] フィールドを選択して、kube-system を入力します。

- vii.[Kubernetes サービスアカウント] フィールドを選択して、efs-csi-controller-sa を入力します。
  - viii[作成] を選択します。
  - ix. Pod Identity の関連付けの作成については、「[the section called “Pod Identity の関連付けを作成する \(AWS コンソール\)”](#)」で詳しく確認できます。
- b. サービスアカウントに IAM ロールを使用している場合
- i. 編集するロールを選択します。
  - ii. [信頼関係] タブを選択し、続いて [信頼ポリシーの編集] を選択します。
  - iii. 次の行と似ている行を探します。

```
"oidc.eks.region-code.amazonaws.com/id/<EXAMPLED539D4633E53DE1B71EXAMPLE>:aud":
"sts.amazonaws.com"
```

前の行の上に、次の行を追加します。<region-code> を、クラスターのある AWS リージョンに置き換えます。<EXAMPLED539D4633E53DE1B71EXAMPLE> をクラスターの OIDC プロバイダー ID に置き換えます。

```
"oidc.eks.<region-code>.amazonaws.com/id/
<EXAMPLED539D4633E53DE1B71EXAMPLE>:sub": "system:serviceaccount:kube-system:efs-
csi-*",
```

- iv. Condition オペレーターを "StringEquals" から "StringLike" に修正します。
- v. [ポリシーの更新] を選択して終了します。

## AWS CLI

次のコマンドを実行して、AWS CLI を使用して IAM ロールを作成します。

Pod Identity を使用する場合

1. pods.eks.amazonaws.com サービスに AssumeRole および TagSession アクションを許可する IAM ロールを作成します。
  - a. 次の内容を aws-efs-csi-driver-trust-policy-pod-identity.json という名前のファイルにコピーします。

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

```
"Statement": [
 {
 "Sid": "AllowEksAuthToAssumeRoleForPodIdentity",
 "Effect": "Allow",
 "Principal": {
 "Service": "pods.eks.amazonaws.com"
 },
 "Action": [
 "sts:AssumeRole",
 "sts:TagSession"
]
 }
]
```

- b. ロールを作成します。my-cluster をクラスター名に置き換えます。AmazonEKS\_EFS\_CSI\_DriverRole を別の名前に置き換えることもできます。

```
export cluster_name=my-cluster
export role_name=AmazonEKS_EFS_CSI_DriverRole
aws iam create-role \
 --role-name $role_name \
 --assume-role-policy-document file://"aws-efs-csi-driver-trust-policy-pod-identity.json"
```

2. 次のマンドを実行して、必要な AWS マネージドポリシーをロールに添付します。

```
aws iam attach-role-policy \
 --policy-arn arn:aws:iam::aws:policy/service-role/AmazonEFSCSIDriverPolicy \
 --role-name $role_name
```

3. 次のコマンドを実行して、Pod Identity の関連付けを作成します。  
arn:aws:iam::<111122223333>:role/my-role を前のステップで作成したロールに置き換えます。

```
aws eks create-pod-identity-association --cluster-name $cluster_name --role-arn {arn-aws}iam::<111122223333>:role/my-role --namespace kube-system --service-account efs-csi-controller-sa
```

4. Pod Identity の関連付けの作成については、「[the section called “Pod Identity の関連付けを作成する \(AWS コンソール\)”](#)」で詳しく確認できます。

## サービスアカウントに IAM ロールを使用している場合

1. クラスターの OIDC プロバイダーの URL を表示します。my-cluster をクラスター名に置き換えます。AmazonEKS\_EFS\_CSI\_DriverRole を別の名前に置き換えることもできます。

```
export cluster_name=my-cluster
export role_name=AmazonEKS_EFS_CSI_DriverRole
aws eks describe-cluster --name $cluster_name --query "cluster.identity.oidc.issuer"
--output text
```

出力例は次のとおりです。

```
https://oidc.eks.<region-code>.amazonaws.com/id/<EXAMPLED539D4633E53DE1B71EXAMPLE>
```

コマンドの出力が None の場合は、「前提条件」を確認してください。

2. IAM ロールを作成して AssumeRoleWithWebIdentity アクションを付与します。
  - a. 次の内容を aws-efs-csi-driver-trust-policy.json という名前のファイルにコピーします。<111122223333> はご使用のアカウントの ID に置き換えます。<EXAMPLED539D4633E53DE1B71EXAMPLE> と <region-code> を、前のステップで返された値に置き換えます。

```
{
 "Version": "2012-10-17",
 "Statement": [
 {
 "Effect": "Allow",
 "Principal": {
 "Federated": "arn:aws:iam::<111122223333>:oidc-provider/oidc.eks.<region-code>.amazonaws.com/id/<EXAMPLED539D4633E53DE1B71EXAMPLE>"
 },
 "Action": "sts:AssumeRoleWithWebIdentity",
 "Condition": {
 "StringLike": {
 "oidc.eks.region-code.amazonaws.com/id/<EXAMPLED539D4633E53DE1B71EXAMPLE>:sub": "system:serviceaccount:kube-system:efs-csi-*",
 "oidc.eks.region-code.amazonaws.com/id/<EXAMPLED539D4633E53DE1B71EXAMPLE>:aud": "sts.amazonaws.com"
 }
 }
 }
]
}
```

```
}
]
}
```

b. ロールを作成します。

```
aws iam create-role \
 --role-name $role_name \
 --assume-role-policy-document file:///aws-efs-csi-driver-trust-policy.json
```

3. 次のマンドを実行して、必要な AWS マネージドポリシーをロールに添付します。

```
aws iam attach-role-policy \
 --policy-arn arn:aws:iam::aws:policy/service-role/AmazonEFSCSIDriverPolicy \
 --role-name $role_name
```

## ステップ 2: Amazon EFS CSI ドライバーを取得する

Amazon EFS CSI ドライバーは Amazon EKS アドオンを使用してインストールすることをお勧めします。Amazon EKS アドオンをクラスターに追加するには、「[the section called “アドオンを作成する”](#)」を参照してください。アドオンの詳細については、「[the section called “Amazon EKS アドオン”](#)」を参照してください。Amazon EKS アドオンを使用できない場合は、その理由に関する問題を[コンテナロードマップ GitHub リポジトリ](#)に送信することをお勧めします。

### Important

Amazon EFS ドライバーを Amazon EKS アドオンとして追加する前に、クラスターにセルフマネージドバージョンのドライバーがインストールされていないことを確認してください。インストールされている場合は、GitHub の「[Uninstalling the Amazon EFS CSI Driver](#)」を参照してください。

または、Amazon EFS CSI ドライバーのセルフマネージド型インストールが必要な場合は、GitHub の「[Installation](#)」を参照してください。

## ステップ 3: Amazon EFS ファイルシステムを作成する

Amazon EFS ファイルシステムを作成するには、「GitHub」の「[Amazon EKS 用の Amazon EFS ファイルシステムを作成](#)」を参照してください。

## ステップ 4: サンプルアプリケーションをデプロイする

さまざまなサンプルアプリケーションをデプロイし、必要に応じて変更できます。詳細については、「GitHub」の「[例](#)」を参照してください。

## Amazon FSx for Lustre で高性能なアプリケーションストレージを使用する

[Amazon FSx for Lustre コンテナストレージインターフェイス \(CSI\) ドライバー](#)は、Amazon EKS クラスターが Amazon FSx for Lustre ファイルシステムのライフサイクルを管理できるようにする CSI インターフェイスを提供します。詳細については、「[Amazon FSx for Lustre ユーザーガイド](#)」を参照してください。

Amazon FSx for Lustre CSI ドライバーを Amazon EKS クラスターにデプロイし、動作することを確認する方法の詳細は、「[the section called “ドライバーをデプロイする”](#)」を参照してください。

### FSx for Lustre ドライバーをデプロイする

このトピックでは、[FSx for Lustre CSI ドライバー](#)を Amazon EKS クラスターにデプロイし、動作することを確認する方法について説明します。最新バージョンのドライバーを使用することをお勧めします。利用可能なバージョンについては、GitHub の「[CSI Specification Compatibility Matrix](#)」(CSI 仕様互換性マトリックス) を参照してください。

#### Note

ドライバーは、Fargate または Amazon EKS Hybrid Nodes ではサポートされていません。

使用可能なパラメータの詳細と、ドライバーの機能を示す完全な例については、GitHub の「[FSx for Lustre Container Storage Interface \(CSI\) driver](#)」プロジェクトを参照してください。

### 前提条件

- 既存のクラスター。
- Amazon FSx CSI ドライバー EKS アドオンの認証には、EKS Pod Identity エージェントが必要です。このコンポーネントがないと、アドオンは「Amazon EKS Pod Identity agent is not installed in the cluster」というエラーで失敗し、ボリュームオペレーションが回避されます。FSx CSI ドライバーアドオンをデプロイする前後に Pod Identity エージェントをインストー

ルします。詳細については、「[the section called “エージェントのセットアップ”](#)」を参照してください。

- ご使用のデバイスまたは AWS クラウドシェル で、バージョン 2.12.3 以降、または AWS コマンドラインインターフェイス (AWS CLI のバージョン 1.27.160 以降がインストールおよび設定されていること。現在のバージョンを確認するには「`aws --version | cut -d / -f2 | cut -d ' ' -f1`」を参照してください。yum、apt-get、macOS 用の Homebrew などのパッケージマネージャーは、多くの場合 AWS CLI の最新バージョンより数バージョン古くなっています。最新バージョンをインストールするには「AWS コマンドラインインターフェイスユーザーガイド」の「[インストール](#)」および「[aws configure を使用したクイック設定](#)」を参照してください。AWS クラウドシェル にインストールされている AWS CLI バージョンも最新バージョンより数バージョン遅れることがあります。更新するには、「AWS CloudShell ユーザーガイド」の「[ホームディレクトリへの AWS CLI のインストール](#)」を参照してください。
- デバイスまたは AWS CloudShell にインストールされている eksctl コマンドラインツールのバージョン 0.210.0 以降。eksctl をインストールまたはアップグレードするにはeksctl ドキュメントの「[インストール](#)」を参照してください。
- デバイスまたは AWS クラウドシェル に、kubectl コマンドラインツールがインストールされていること。バージョンはご使用のクラスターの Kubernetes バージョンと同じか、1 つ前のマイナーバージョン以前、あるいはそれより新しいバージョンが使用できます。例えば、クラスターのバージョンが 1.29 である場合、kubectl のバージョン 1.28、1.29、または 1.30 が使用できます。kubectl をインストールまたはアップグレードする方法については「[the section called “kubectl と eksctl のセットアップ”](#)」を参照してください。

## ステップ 1: IAM ロールを作成する

Amazon FSx CSI プラグインでは、ユーザーに代わって AWS API の呼び出しを行うための IAM アクセス許可が必要です。

### Note

Pod は、IMDS へのアクセスをブロックする場合を除き、IAM ロールに割り当てられたアクセス許可にアクセスできます。詳細については、「[the section called “ベストプラクティス”](#)」を参照してください。

以下の手順は、IAM ロールを作成し、それに AWS マネージドポリシーをアタッチする方法を示しています。

1. IAM ロールを作成し、次のコマンドで AWS マネージドポリシーをアタッチします。my-cluster を、使用するクラスターの名前に置き換えます。このコマンドは IAM ロールを作成して IAM ポリシーをアタッチする AWS CloudFormation スタックをデプロイします。

```
eksctl create iamserviceaccount \
 --name fsx-csi-controller-sa \
 --namespace kube-system \
 --cluster my-cluster \
 --role-name AmazonEKS_FSx_CSI_DriverRole \
 --role-only \
 --attach-policy-arn arn:aws:iam::aws:policy/AmazonFSxFullAccess \
 --approve
```

サービスアカウントが作成されると、数行の出力が表示されます。出力の最後の数行は、次のようになります。

```
[#] 1 task: {
 2 sequential sub-tasks: {
 create IAM role for serviceaccount "kube-system/fsx-csi-controller-sa",
 create serviceaccount "kube-system/fsx-csi-controller-sa",
 } }
[#] building iamserviceaccount stack "eksctl-my-cluster-addon-iamserviceaccount-kube-system-fsx-csi-controller-sa"
[#] deploying stack "eksctl-my-cluster-addon-iamserviceaccount-kube-system-fsx-csi-controller-sa"
[#] waiting for CloudFormation stack "eksctl-my-cluster-addon-iamserviceaccount-kube-system-fsx-csi-controller-sa"
[#] created serviceaccount "kube-system/fsx-csi-controller-sa"
```

デプロイされた AWS CloudFormation スタックの名前を書き留めます。前述の出力例では、スタックの名前は eksctl-my-cluster-addon-iamserviceaccount-kube-system-fsx-csi-controller-sa です。

Amazon FSx CSI ドライバーの IAM ロールを作成したので、次のセクションに進むことができます。この IAM ロールを使用してアドオンをデプロイすると、fsx-csi-controller-sa という名前のサービスアカウントが作成され、それを使用されるように設定されます。サービスアカウントは Kubernetes clusterrole にバインドされます。これには、必要な Kubernetes アクセス許可が割り当てられています。



## ステップ 2: Amazon FSx CSI ドライバーをインストールする

セキュリティを強化し、作業量を削減するために、Amazon EKS アドオンを通じて Amazon FSx CSI ドライバーをインストールすることをお勧めします。Amazon EKS アドオンをクラスターに追加するには、「[the section called “アドオンを作成する”](#)」を参照してください。アドオンの詳細については、「[the section called “Amazon EKS アドオン”](#)」を参照してください。

### Important

クラスターに既存の Amazon FSx CSI ドライバーをインストールすると、アドオンのインストールに失敗する可能性があります。EKS 以外の FSx CSI ドライバーが存在するところに Amazon EKS のアドオンバージョンをインストールしようとする、リソースの競合によりインストールが失敗します。この問題を解消するには、インストール時に OVERWRITE フラグを使用します。

```
aws eks create-addon --addon-name aws-fsx-csi-driver --cluster-name my-cluster
--resolve-conflicts OVERWRITE
```

または、Amazon FSx CSI ドライバーのセルフマネージド型インストールが必要な場合は、GitHub の「[Installation](#)」を参照してください。

## ステップ 3: ストレージクラス、永続的なボリューム要求、サンプルアプリケーションをデプロイする

この手順では、「[FSx for Lustre Container Storage Interface \(CSI\) driver](#)」(FSx for Lustre コンテナストレージインターフェイス (CSI) ドライバー) GitHub リポジトリを使用して、動的にプロビジョニングされた FSx for Lustre ボリュームを使用します。

1. クラスターのセキュリティグループを書き留めます。AWS Management Console の [ネットワーク] セクションまたは次の AWS CLI コマンドを使用して確認できます。my-cluster を、使用するクラスターの名前に置き換えます。

```
aws eks describe-cluster --name my-cluster --query
cluster.resourcesVpcConfig.clusterSecurityGroupId
```

2. 「Amazon FSx for Lustre ユーザーガイド」の「[Amazon VPC セキュリティグループ](#)」に示す基準に従って、Amazon FSx ファイルシステムのセキュリティグループを作成します。[VPC]

で、[ネットワーク] セクションに示されているようにクラスターの VPC を選択します。「Lustre クライアントに関連付けられているセキュリティグループ」には、クラスターセキュリティグループを使用します。アウトバウンドルールをそのままにして、[すべてのトラフィック] を許可することができます。

3. 次のコマンドを使用して、ストレージクラスマニフェストをダウンロードします。

```
curl -O https://raw.githubusercontent.com/kubernetes-sigs/aws-fsx-csi-driver/master/examples/kubernetes/dynamic_provisioning/specs/storageclass.yaml
```

4. `storageclass.yaml` ファイルの `parameters` セクションを編集します。各サンプル値は独自の値に置き換えます。

```
parameters:
 subnetId: subnet-0eabfaa81fb22bcaf
 securityGroupIds: sg-068000ccf82dfba88
 deploymentType: PERSISTENT_1
 automaticBackupRetentionDays: "1"
 dailyAutomaticBackupStartTime: "00:00"
 copyTagsToBackups: "true"
 perUnitStorageThroughput: "200"
 dataCompressionType: "NONE"
 weeklyMaintenanceStartTime: "7:09:00"
 fileSystemTypeVersion: "2.12"
```

- **subnetId** – Amazon FSx for Lustre ファイルシステムが作成されるサブネット ID。Amazon FSx for Lustre は、すべてのアベイラビリティゾーンでサポートされているわけではありません。<https://console.aws.amazon.com/fsx/> で Amazon FSx for Lustre コンソールを開き、使用するサブネットが、サポートされているアベイラビリティゾーンにあることを確認します。次のように、サブネットにノードを含めることも、別のサブネットまたは VPC にすることもできます。
- AWS Management Console で [コンピューティング] セクションのノードグループを選択すると、ノードサブネットを確認できます。
- 指定するサブネットが、ノードがあるサブネットと同じでない場合は、VPC が [接続](#) されている必要があり、セキュリティグループで必要なポートが開いていることを確認する必要があります。
- **securityGroupIds** – ファイルシステム用に作成したセキュリティグループの ID。
- **deploymentType** (オプション) – ファイルシステムのデプロイのタイプ。有効な値は、SCRATCH\_1、SCRATCH\_2、PERSISTENT\_1、および PERSISTENT\_2 です。デプロイのタ

イプの詳細については、「[Amazon FSx for Lustre ファイルシステムを作成する](#)」を参照してください。

- 他のパラメータ (オプション) – 他のパラメータについては、GitHub の「[Edit StorageClass](#)」(StorageClass の編集) を参照してください。

## 5. ストレージクラスマニフェストを作成します。

```
kubectl apply -f storageclass.yaml
```

出力例は次のとおりです。

```
storageclass.storage.k8s.io/fsx-sc created
```

## 6. 永続的なボリューム要求マニフェストをダウンロードします。

```
curl -O https://raw.githubusercontent.com/kubernetes-sigs/aws-fsx-csi-driver/master/examples/kubernetes/dynamic_provisioning/specs/claim.yaml
```

## 7. (オプション) claim.yaml ファイルを編集します。ストレージ要件と前のステップで選択した deploymentType に基づいて、1200Gi を次のいずれかの増分値に変更します。

```
storage: 1200Gi
```

- SCRATCH\_2 および PERSISTENT – 1.2 TiB、2.4 TiB、または 2.4 TiB を超えると 2.4 TiB の増分。
- SCRATCH\_1 – 1.2 TiB、2.4 TiB、3.6 TiB、または 3.6 TiB を超えると 3.6 TiB の増分。

## 8. 永続的なボリューム要求を作成します。

```
kubectl apply -f claim.yaml
```

出力例は次のとおりです。

```
persistentvolumeclaim/fsx-claim created
```

## 9. ファイルシステムがプロビジョニングされていることを確認します。

```
kubectl describe pvc
```

出力例は次のとおりです。

```
Name: fsx-claim
Namespace: default
StorageClass: fsx-sc
Status: Bound
[...]
```

### Note

Status は、Pending になる前に 5～10 分間 Bound と表示されることがあります。Status が Bound になるまで次のステップに進まないでください。Status が 10 分を超えて Pending になっている場合は、Events 内の警告メッセージを問題に対処するための参考として使用します。

## 10. サンプルアプリケーションをデプロイします。

```
kubectl apply -f https://raw.githubusercontent.com/kubernetes-sigs/aws-fsx-csi-driver/master/examples/kubernetes/dynamic_provisioning/specs/pod.yaml
```

## 11. サンプルアプリケーションが実行中であることを確認します。

```
kubectl get pods
```

出力例は次のとおりです。

| NAME    | READY | STATUS  | RESTARTS | AGE |
|---------|-------|---------|----------|-----|
| fsx-app | 1/1   | Running | 0        | 8s  |

## 12. ファイルシステムがアプリケーションによって正しくマウントされていることを確認します。

```
kubectl exec -ti fsx-app -- df -h
```

出力例は次のとおりです。

| Filesystem              | Size | Used | Avail | Use% | Mounted on     |
|-------------------------|------|------|-------|------|----------------|
| overlay                 | 80G  | 4.0G | 77G   | 5%   | /              |
| tmpfs                   | 64M  | 0    | 64M   | 0%   | /dev           |
| tmpfs                   | 3.8G | 0    | 3.8G  | 0%   | /sys/fs/cgroup |
| 192.0.2.0@tcp:/abcdef01 | 1.1T | 7.8M | 1.1T  | 1%   | /data          |
| /dev/nvme0n1p1          | 80G  | 4.0G | 77G   | 5%   | /etc/hosts     |

|                |      |     |      |    |                             |
|----------------|------|-----|------|----|-----------------------------|
| shm            | 64M  | 0   | 64M  | 0% | /dev/shm                    |
| tmpfs          | 6.9G | 12K | 6.9G | 1% | /run/secrets/kubernetes.io/ |
| serviceaccount |      |     |      |    |                             |
| tmpfs          | 3.8G | 0   | 3.8G | 0% | /proc/acpi                  |
| tmpfs          | 3.8G | 0   | 3.8G | 0% | /sys/firmware               |

13.データがサンプルアプリケーションによって FSx for Lustre ファイルシステムに書き込まれたことを確認します。

```
kubectl exec -it fsx-app -- ls /data
```

出力例は次のとおりです。

```
out.txt
```

この出力例は、サンプルアプリケーションが out.txt ファイルをファイルシステムに正常に書き込んだことを示しています。

#### Note

クラスターを削除する前に FSx for Lustre ファイルシステムを必ず削除してください。詳細については、「FSx for Lustre ユーザーガイド」の「[リソースをクリーンアップする](#)」を参照してください。

## FSx for Lustre のパフォーマンスチューニング

Amazon EKS で FSx for Lustre を使用する場合、ノードの初期化中に Lustre のチューニングを適用するとパフォーマンスを最適化できます。推奨されるアプローチは、起動テンプレートのユーザーデータを使用して、すべてのノードで一貫した設定を確保することです。

チューニングの内容は次のとおりです。

- ネットワークと RPC の最適化
- Lustre モジュールの管理
- LRU (ロックリソースユニット) の調整
- クライアントキャッシュコントロールの設定
- OST および MDC の RPC コントロール

これらのパフォーマンスチューニングを実装する詳細な手順については、以下を参照してください。

- 標準ノード (EFA 以外) のパフォーマンスを最適化するには、「[the section called “最適化 \(EFA 以外\)”](#)」で起動テンプレートのユーザーデータに追加できる完全なスクリプトを参照してください。
- EFA 対応ノードのパフォーマンスを最適化するには、「[the section called “最適化 \(EFA\)”](#)」を参照してください。

## ノードでの Amazon FSx for Lustre のパフォーマンスを最適化する (EFA)

このトピックでは、Amazon EKS と Amazon FSx for Lustre を使用して Elastic Fabric Adapter (EFA) のチューニングを設定する方法について説明します。

### Note

- FSx for Lustre CSI ドライバーの作成とデプロイの詳細については、「[the section called “ドライバーをデプロイする”](#)」を参照してください。
- EFA を使用しない標準ノードの最適化については、「[the section called “最適化 \(EFA 以外\)”](#)」を参照してください。

## ステップ 1. EKS クラスターを作成する

所定の設定ファイルを使用してクラスターを作成します。

```
Create cluster using efa-cluster.yaml
eksctl create cluster -f efa-cluster.yaml
```

例 efa-cluster.yaml:

```
#efa-cluster.yaml

apiVersion: eksctl.io/v1alpha5
kind: ClusterConfig

metadata:
 name: csi-fsx
 region: us-east-1
 version: "1.30"
```

```

iam:
 withOIDC: true

availabilityZones: ["us-east-1a", "us-east-1d"]

managedNodeGroups:
 - name: my-efa-ng
 instanceType: c6gn.16xlarge
 minSize: 1
 desiredCapacity: 1
 maxSize: 1
 availabilityZones: ["us-east-1b"]
 volumeSize: 300
 privateNetworking: true
 amiFamily: Ubuntu2204
 efaEnabled: true
 preBootstrapCommands:
 - |
 #!/bin/bash
 eth_intf="$(/sbin/ip -br -4 a sh | grep $(hostname -i)/ | awk '{print $1}')"
 efa_version=$(modinfo efa | awk '/^version:/ {print $2}' | sed 's/[^0-9.]/g')
 min_efa_version="2.12.1"

 if [["$(printf '%s\n' "$min_efa_version" "$efa_version" | sort -V | head -
n1)" != "$min_efa_version"]]; then
 sudo curl -O https://efa-installer.amazonaws.com/aws-efa-
installer-1.37.0.tar.gz
 tar -xf aws-efa-installer-1.37.0.tar.gz && cd aws-efa-installer
 echo "Installing EFA driver"
 sudo apt-get update && apt-get upgrade -y
 sudo apt install -y pciutils environment-modules libnl-3-dev libnl-
route-3-200 libnl-route-3-dev dkms
 sudo ./efa_installer.sh -y
 modinfo efa
 else
 echo "Using EFA driver version $efa_version"
 fi

 echo "Installing Lustre client"
 sudo wget -O - https://fsx-lustre-client-repo-public-keys.s3.amazonaws.com/fsx-
ubuntu-public-key.asc | gpg --dearmor | sudo tee /usr/share/keyrings/fsx-ubuntu-public-
key.gpg > /dev/null

```

```
sudo echo "deb [signed-by=/usr/share/keyrings/fsx-ubuntu-public-key.gpg]
https://fsx-lustre-client-repo.s3.amazonaws.com/ubuntu jammy main" > /etc/apt/
sources.list.d/fsxlustreclientrepo.list
sudo apt update | tail
sudo apt install -y lustre-client-modules-$(uname -r) amazon-ec2-utils | tail
modinfo lustre

echo "Loading Lustre/EFA modules..."
sudo /sbin/modprobe lnet
sudo /sbin/modprobe kefalnd ipif_name="$eth_intf"
sudo /sbin/modprobe ksocklnd
sudo lnetctl lnet configure

echo "Configuring TCP interface..."
sudo lnetctl net del --net tcp 2> /dev/null
sudo lnetctl net add --net tcp --if $eth_intf

For P5 instance type which supports 32 network cards,
by default add 8 EFA interfaces selecting every 4th device (1 per PCI bus)
echo "Configuring EFA interface(s)..."
instance_type="$(ec2-metadata --instance-type | awk '{ print $2 }')"
num_efa_devices="$(ls -1 /sys/class/infiniband | wc -l)"
echo "Found $num_efa_devices available EFA device(s)"

if [["$instance_type" == "p5.48xlarge" || "$instance_type" ==
"p5e.48xlarge"]]; then
 for intf in $(ls -1 /sys/class/infiniband | awk 'NR % 4 == 1'); do
 sudo lnetctl net add --net efa --if $intf --peer-credits 32
 done
else
 # Other instances: Configure 2 EFA interfaces by default if the instance
supports multiple network cards,
 # or 1 interface for single network card instances
 # Can be modified to add more interfaces if instance type supports it
 sudo lnetctl net add --net efa --if $(ls -1 /sys/class/infiniband | head -
n1) --peer-credits 32
 if [[$num_efa_devices -gt 1]]; then
 sudo lnetctl net add --net efa --if $(ls -1 /sys/class/infiniband | tail
-n1) --peer-credits 32
 fi
fi

echo "Setting discovery and UDSP rule"
sudo lnetctl set discovery 1
```



```
sudo lnctlctl udsp add --src efa --priority 0
sudo /sbin/modprobe lustre

sudo lnctlctl net show
echo "Added $(sudo lnctlctl net show | grep -c '@efa') EFA interface(s)"
```

## ステップ 2. ノードグループの作成

EFA 対応ノードグループを作成します。

```
Create node group using efa-ng.yaml
eksctl create nodegroup -f efa-ng.yaml
```

### Important

=== これらの値をセクション # 5. Mount FSx filesystem の環境向けに調整します。

```
FSX_DNS="<your-fsx-filesystem-dns>" # Needs to be adjusted.
MOUNT_NAME="<your-mount-name>" # Needs to be adjusted.
MOUNT_POINT="</your/mount/point>" # Needs to be adjusted.
```

===

例 efa-ng.yaml:

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

metadata:
 name: final-efa
 region: us-east-1

managedNodeGroups:
- name: ng-1
 instanceType: c6gn.16xlarge
 minSize: 1
 desiredCapacity: 1
 maxSize: 1
 availabilityZones: ["us-east-1a"]
 volumeSize: 300
```

```
privateNetworking: true
amiFamily: Ubuntu2204
efaEnabled: true
preBootstrapCommands:
- |
 #!/bin/bash
 exec 1> >(logger -s -t $(basename $0)) 2>&1
```

```
#####
Configuration Section
#
#####

File System Configuration
FSX_DNS="" # Needs to be adjusted.
MOUNT_NAME="" # Needs to be adjusted.
MOUNT_POINT="" # Needs to be adjusted.

Lustre Tuning Parameters
LUSTRE_LRU_MAX_AGE=600000
LUSTRE_MAX_CACHED_MB=64
LUSTRE_OST_MAX_RPC=32
LUSTRE_MDC_MAX_RPC=64
LUSTRE_MDC_MOD_RPC=50

File paths
FUNCTIONS_SCRIPT="/usr/local/bin/lustre_functions.sh"
TUNINGS_SCRIPT="/usr/local/bin/apply_lustre_tunings.sh"
SERVICE_FILE="/etc/systemd/system/lustre-tunings.service"

#EFA
MIN_EFA_VERSION="2.12.1"

Function to check if a command was successful
check_success() {
 if [$? -eq 0]; then
 echo "SUCCESS: $1"
 else
 echo "FAILED: $1"
 return 1
 fi
}
}
```

```
echo "*****Starting FSx for Lustre configuration*****"

1. Install Lustre client
if grep -q '^ID=ubuntu' /etc/os-release; then
 echo "Detected Ubuntu, proceeding with Lustre setup..."
 # Add Lustre repository
 sudo wget -O - https://fsx-lustre-client-repo-public-keys.s3.amazonaws.com/
fsx-ubuntu-public-key.asc | sudo gpg --dearmor | sudo tee /usr/share/keyrings/fsx-
ubuntu-public-key.gpg > /dev/null

 echo "deb [signed-by=/usr/share/keyrings/fsx-ubuntu-public-key.gpg]
https://fsx-lustre-client-repo.s3.amazonaws.com/ubuntu jammy main" | sudo tee /etc/
apt/sources.list.d/fsxlustreclientrepo.list

 sudo apt-get update
 sudo apt-get install -y lustre-client-modules-$(uname -r)
 sudo apt-get install -y lustre-client
else
 echo "Not Ubuntu, exiting"
 exit 1
fi

check_success "Install Lustre client"

Ensure Lustre tools are in the PATH
export PATH=$PATH:/usr/sbin

2. Apply network and RPC tunings
echo "*****Applying network and RPC tunings*****"
if ! grep -q "options ptlrpc ptlrpcd_per_cpt_max" /etc/modprobe.d/
modprobe.conf; then
 echo "options ptlrpc ptlrpcd_per_cpt_max=64" | sudo tee -a /etc/modprobe.d/
modprobe.conf
 check_success "Set ptlrpcd_per_cpt_max"
else
 echo "ptlrpcd_per_cpt_max already set in modprobe.conf"
fi

if ! grep -q "options ksocklnd credits" /etc/modprobe.d/modprobe.conf; then
 echo "options ksocklnd credits=2560" | sudo tee -a /etc/modprobe.d/
modprobe.conf
 check_success "Set ksocklnd credits"
else
```

```
 echo "ksocklnd credits already set in modprobe.conf"
 fi

 # 3. Load Lustre modules
 manage_lustre_modules() {
 echo "Checking for existing Lustre modules..."
 if lsmod | grep -q lustre; then
 echo "Existing Lustre modules found."

 # Check for mounted Lustre filesystems
 echo "Checking for mounted Lustre filesystems..."
 if mount | grep -q "type lustre"; then
 echo "Found mounted Lustre filesystems. Attempting to unmount..."
 mounted_fs=$(mount | grep "type lustre" | awk '{print $3}')
 for fs in $mounted_fs; do
 echo "Unmounting $fs"
 sudo umount $fs
 check_success "Unmount filesystem $fs"
 done
 else
 echo "No Lustre filesystems mounted."
 fi

 # After unmounting, try to remove modules
 echo "Attempting to remove Lustre modules..."
 sudo lustre_rmmod
 if [$? -eq 0]; then
 echo "SUCCESS: Removed existing Lustre modules"
 else
 echo "WARNING: Could not remove Lustre modules. They may still be
in use."

 echo "Please check for any remaining Lustre processes or mounts."
 return 1
 fi
 else
 echo "No existing Lustre modules found."
 fi

 echo "Loading Lustre modules..."
 sudo modprobe lustre
 check_success "Load Lustre modules" || exit 1

 echo "Checking loaded Lustre modules:"
 lsmod | grep lustre
 }
}
```

```

}

Managing Lustre kernel modules
echo "*****Managing Lustre kernel modules*****"
manage_lustre_modules

4. Initializing Lustre networking
echo "*****Initializing Lustre networking*****"
sudo lctl network up
check_success "Initialize Lustre networking" || exit 1

4.5 EFA Setup and Configuration
setup_efa() {
 echo "*****Starting EFA Setup*****"

 # Get interface and version information
 eth_intf="$(/sbin/ip -br -4 a sh | grep $(hostname -i)/ | awk '{print
$1}')"
 efa_version=$(modinfo efa | awk '/^version:/ {print $2}' | sed 's/[^0-9.]//
g')
 min_efa_version=$MIN_EFA_VERSION

 # Install or verify EFA driver
 if [["$(printf '%s\n' "$min_efa_version" "$efa_version" | sort -V | head -
n1)" != "$min_efa_version"]]; then
 echo "Installing EFA driver..."
 sudo curl -O https://efa-installer.amazonaws.com/aws-efa-
installer-1.37.0.tar.gz
 tar -xf aws-efa-installer-1.37.0.tar.gz && cd aws-efa-installer

 # Install dependencies
 sudo apt-get update && apt-get upgrade -y
 sudo apt install -y pciutils environment-modules libnl-3-dev libnl-
route-3-200 libnl-route-3-dev dkms

 # Install EFA
 sudo ./efa_installer.sh -y
 modinfo efa
 else
 echo "Using existing EFA driver version $efa_version"
 fi
}

configure_efa_network() {

```

```

echo "*****Configuring EFA Network*****"

Load required modules
echo "Loading network modules..."
sudo /sbin/modprobe lnet
sudo /sbin/modprobe kefalnd ipif_name="$eth_intf"
sudo /sbin/modprobe ksocklnd

Initialize LNet
echo "Initializing LNet..."
sudo lnetctl lnet configure

Configure TCP interface
echo "Configuring TCP interface..."
sudo lnetctl net del --net tcp 2> /dev/null
sudo lnetctl net add --net tcp --if $eth_intf

For P5 instance type which supports 32 network cards,
by default add 8 EFA interfaces selecting every 4th device (1 per PCI
bus)

echo "Configuring EFA interface(s)..."
instance_type="$(ec2-metadata --instance-type | awk '{ print $2 }')"
num_efa_devices="$(ls -1 /sys/class/infiniband | wc -l)"
echo "Found $num_efa_devices available EFA device(s)"

if [["$instance_type" == "p5.48xlarge" || "$instance_type" ==
"p5e.48xlarge"]]; then
 # P5 instance configuration
 for intf in $(ls -1 /sys/class/infiniband | awk 'NR % 4 == 1'); do
 sudo lnetctl net add --net efa --if $intf --peer-credits 32
 done
else
 # Standard configuration
 # Other instances: Configure 2 EFA interfaces by default if the
instance supports multiple network cards,
 # or 1 interface for single network card instances
 # Can be modified to add more interfaces if instance type supports it
 sudo lnetctl net add --net efa --if $(ls -1 /sys/class/infiniband |
head -n1) --peer-credits 32
 if [[$num_efa_devices -gt 1]]; then
 sudo lnetctl net add --net efa --if $(ls -1 /sys/class/infiniband |
tail -n1) --peer-credits 32
 fi
fi
fi

```

```
Configure discovery and UDSP
echo "Setting up discovery and UDSP rules..."
sudo lnctl set discovery 1
sudo lnctl udsp add --src efa --priority 0
sudo /sbin/modprobe lustre

Verify configuration
echo "Verifying EFA network configuration..."
sudo lnctl net show
echo "Added $(sudo lnctl net show | grep -c '@efa') EFA interface(s)"
}

Main execution
setup_efa
configure_efa_network

5. Mount FSx filesystem
if [! -z "$FSX_DNS"] && [! -z "$MOUNT_NAME"]; then
 echo "*****Creating mount point*****"
 sudo mkdir -p $MOUNT_POINT
 check_success "Create mount point"

 echo "Mounting FSx filesystem..."
 sudo mount -t lustre ${FSX_DNS}@tcp:${MOUNT_NAME} ${MOUNT_POINT}
 check_success "Mount FSx filesystem"
else
 echo "Skipping FSx mount as DNS or mount name is not provided"
fi

6. Applying Lustre performance tunings
echo "*****Applying Lustre performance tunings*****"

Get number of CPUs
NUM_CPUS=$(nproc)

Calculate LRU size (100 * number of CPUs)
LRU_SIZE=$((100 * NUM_CPUS))

#Apply LRU tunings
echo "Apply LRU tunings"
sudo lctl set_param ldlm.namespaces.*.lru_max_age=${LUSTRE_LRU_MAX_AGE}
check_success "Set lru_max_age"
sudo lctl set_param ldlm.namespaces.*.lru_size=$LRU_SIZE
```

```
check_success "Set lru_size"

Client Cache Control
sudo lctl set_param llite.*.max_cached_mb=${LUSTRE_MAX_CACHED_MB}
check_success "Set max_cached_mb"

RPC Controls
sudo lctl set_param osc.*OST*.max_rpcs_in_flight=${LUSTRE_OST_MAX_RPC}
check_success "Set OST max_rpcs_in_flight"

sudo lctl set_param mdc.*.max_rpcs_in_flight=${LUSTRE_MDC_MAX_RPC}
check_success "Set MDC max_rpcs_in_flight"

sudo lctl set_param mdc.*.max_mod_rpcs_in_flight=${LUSTRE_MDC_MOD_RPC}
check_success "Set MDC max_mod_rpcs_in_flight"

7. Verify all tunings
echo "*****Verifying all tunings*****"

Function to verify parameter value
verify_param() {
 local param=$1
 local expected=$2
 local actual=$3

 if ["$actual" == "$expected"]; then
 echo "SUCCESS: $param is correctly set to $expected"
 else
 echo "WARNING: $param is set to $actual (expected $expected)"
 fi
}

echo "Verifying all parameters:"

LRU tunings
actual_lru_max_age=$(lctl get_param -n ldlm.namespaces.*.lru_max_age | head -1)
verify_param "lru_max_age" "600000" "$actual_lru_max_age"

actual_lru_size=$(lctl get_param -n ldlm.namespaces.*.lru_size | head -1)
verify_param "lru_size" "$LRU_SIZE" "$actual_lru_size"

Client Cache
actual_max_cached_mb=$(lctl get_param -n llite.*.max_cached_mb | grep
"max_cached_mb:" | awk '{print $2}')
```



```

verify_param "max_cached_mb" "64" "$actual_max_cached_mb"

RPC Controls
actual_ost_rpcs=$(lctl get_param -n osc.*OST*.max_rpcs_in_flight | head -1)
verify_param "OST max_rpcs_in_flight" "32" "$actual_ost_rpcs"

actual_mdc_rpcs=$(lctl get_param -n mdc.*.max_rpcs_in_flight | head -1)
verify_param "MDC max_rpcs_in_flight" "64" "$actual_mdc_rpcs"

actual_mdc_mod_rpcs=$(lctl get_param -n mdc.*.max_mod_rpcs_in_flight | head -1)
verify_param "MDC max_mod_rpcs_in_flight" "50" "$actual_mdc_mod_rpcs"

Network and RPC configurations from modprobe.conf
actual_ptlrpc=$(grep "ptlrpc ptlrpcd_per_cpt_max" /etc/modprobe.d/modprobe.conf
| awk '{print $3}')
verify_param "ptlrpcd_per_cpt_max" "ptlrpcd_per_cpt_max=64" "$actual_ptlrpc"

actual_ksocklnd=$(grep "ksocklnd credits" /etc/modprobe.d/modprobe.conf | awk
'{print $3}')
verify_param "ksocklnd credits" "credits=2560" "$actual_ksocklnd"

8. Setup persistence
setup_persistence() {
 # Create functions file
 cat << EOF > $FUNCTIONS_SCRIPT
#!/bin/bash

apply_lustre_tunings() {
 local NUM_CPUS=\$(nproc)
 local LRU_SIZE=\$((100 * NUM_CPUS))

 echo "Applying Lustre performance tunings..."
 lctl set_param ldlm.namespaces.*.lru_max_age=$LUSTRE_LRU_MAX_AGE
 lctl set_param ldlm.namespaces.*.lru_size=\$LRU_SIZE
 lctl set_param llite.*.max_cached_mb=$LUSTRE_MAX_CACHED_MB
 lctl set_param osc.*OST*.max_rpcs_in_flight=$LUSTRE_OST_MAX_RPC
 lctl set_param mdc.*.max_rpcs_in_flight=$LUSTRE_MDC_MAX_RPC
 lctl set_param mdc.*.max_mod_rpcs_in_flight=$LUSTRE_MDC_MOD_RPC
}
EOF

 # Create tuning script
 cat << EOF > $TUNINGS_SCRIPT
#!/bin/bash

```

```
exec 1> >(logger -s -t \$(basename \$0)) 2>&1

source $FUNCTIONS_SCRIPT

Function to check if Lustre is mounted
is_lustre_mounted() {
 mount | grep -q "type lustre"
}

Function to mount Lustre
mount_lustre() {
 echo "Mounting Lustre filesystem..."
 mkdir -p $MOUNT_POINT
 mount -t lustre ${FSX_DNS}@tcp:/${MOUNT_NAME} $MOUNT_POINT
 return $?
}

Main execution
Try to mount if not already mounted
if ! is_lustre_mounted; then
 echo "Lustre filesystem not mounted, attempting to mount..."
 mount_lustre
fi

Wait for successful mount (up to 5 minutes)
for i in {1..30}; do
 if is_lustre_mounted; then
 echo "Lustre filesystem mounted, applying tunings..."
 apply_lustre_tunings
 exit 0
 fi
 echo "Waiting for Lustre filesystem to be mounted... (attempt $i/30)"
 sleep 10
done

echo "Timeout waiting for Lustre filesystem mount"
exit 1
EOF

Create systemd service

Create systemd directory if it doesn't exist
sudo mkdir -p /etc/systemd/system/
```

```
Create service file directly for Ubuntu
cat << EOF > $SERVICE_FILE
[Unit]
Description=Apply Lustre Performance Tunings
After=network.target remote-fs.target

[Service]
Type=oneshot
ExecStart=/bin/bash -c 'source $FUNCTIONS_SCRIPT && $TUNINGS_SCRIPT'
RemainAfterExit=yes

[Install]
WantedBy=multi-user.target
EOF

Make scripts executable and enable service
sudo chmod +x $FUNCTIONS_SCRIPT
sudo chmod +x $TUNINGS_SCRIPT
systemctl enable lustre-tunings.service
systemctl start lustre-tunings.service
}

echo "*****Setting up persistent tuning*****"
setup_persistence

echo "FSx for Lustre configuration completed."
```

## (オプション) ステップ 3. EFA の設定を検証する

ノードに SSH 接続する:

```
Get instance ID from EKS console or AWS CLI
ssh -i /path/to/your-key.pem ec2-user@<node-internal-ip>
```

EFA 設定を検証する:

```
sudo lnctl net show
```

セットアップログを確認する:

```
sudo cat /var/log/cloud-init-output.log
```

lnetctl net show の想定される出力例:

```
net:
 - net type: tcp
 ...
 - net type: efa
 local NI(s):
 - nid: xxx.xxx.xxx.xxx@efa
 status: up
```

## デプロイ例

### a. claim.yaml を作成する

```
#claim.yaml

apiVersion: v1
kind: PersistentVolumeClaim
metadata:
 name: fsx-claim-efa
spec:
 accessModes:
 - ReadWriteMany
 storageClassName: ""
 resources:
 requests:
 storage: 4800Gi
 volumeName: fsx-pv
```

claim を適用する:

```
kubectl apply -f claim.yaml
```

### b. pv.yaml を作成する

<replaceable-placeholders> を更新:

```
#pv.yaml

apiVersion: v1
kind: PersistentVolume
```

```
metadata:
 name: fsx-pv
spec:
 capacity:
 storage: 4800Gi
 volumeMode: Filesystem
 accessModes:
 - ReadWriteMany
 mountOptions:
 - flock
 persistentVolumeReclaimPolicy: Recycle
 csi:
 driver: fsx.csi.aws.com
 volumeHandle: fs-<1234567890abcdef0>
 volumeAttributes:
 dnsname: fs-<1234567890abcdef0>.fsx.us-east-1.amazonaws.com
 mountname: <abcdef01>
```

永続ボリュームを適用する:

```
kubectl apply -f pv.yaml
```

c. pod.yaml を作成する

```
#pod.yaml

apiVersion: v1
kind: Pod
metadata:
 name: fsx-efa-app
spec:
 containers:
 - name: app
 image: amazonlinux:2
 command: ["/bin/sh"]
 args: ["-c", "while true; do dd if=/dev/urandom bs=100M count=20 > data/test_file;
sleep 10; done"]
 resources:
 requests:
 vpc.amazonaws.com/efa: 1
 limits:
 vpc.amazonaws.com/efa: 1
 volumeMounts:
```

```
- name: persistent-storage
 mountPath: /data
volumes:
- name: persistent-storage
 persistentVolumeClaim:
 claimName: fsx-claim-efa
```

ポッドを適用する:

```
kubectl apply -f pod.yaml
```

## 追加の検証コマンド

ポッドがマウントされファイルシステムに書き込まれていることを確認します。

```
kubectl exec -ti fsx-efa-app -- df -h | grep data
Expected output:
<192.0.2.0>@tcp:/<abcdef01> 4.5T 1.2G 4.5T 1% /data

kubectl exec -ti fsx-efa-app -- ls /data
Expected output:
test_file
```

ノードに SSH 接続してトラフィックが EFA を経由していることを確認します。

```
sudo lnnetctl net show -v
```

想定される出力に、トラフィックの統計を含む EFA インターフェイスが表示されます。

## 関連情報

- [the section called “ドライバーをデプロイする”](#)
- [the section called “最適化 \(EFA 以外\)”](#)
- [Amazon FSx for Lustre のパフォーマンス](#)
- [Elastic Fabric Adapter](#)

# ノードでの Amazon FSx for Lustre のパフォーマンスを最適化する (EFA 以外)

起動テンプレートのユーザーデータを使用して、ノードの初期化中にチューニングパラメータを適用することで、Amazon FSx for Lustre のパフォーマンスを最適化することができます。

## Note

- FSx for Lustre CSI ドライバーの作成とデプロイの詳細については、「[the section called “ドライバーをデプロイする”](#)」を参照してください。EFA 対応ノードでパフォーマンスを最適化するには、「[the section called “最適化 \(EFA\)”](#)」を参照してください。

## なぜ起動テンプレートのユーザーデータを使用するのか

- ノードの初期化中にチューニングを自動的に適用するため。
- すべてのノードで一貫した設定を確保するため。
- 手動でのノード設定を不要にするため。

## スクリプトの概要の例

このトピックで定義されているスクリプト例では、以下の操作を実行します。

### # 1. Install Lustre client

- OS バージョン (Amazon Linux 2 (AL2) または Amazon Linux 2023 (AL2023)) を自動検出します。
- 適切な Lustre クライアントパッケージをインストールします。

### # 2. Apply network and RPC tunings

- 並列 RPC 処理用に `ptlrpcd_per_cpt_max=64` を設定します。
- `ksocklnd credits=2560` を設定してネットワークバッファを最適化します。

### # 3. Load Lustre modules

- 既存の Lustre モジュールがある場合は、安全に削除します。

- 既存のファイルシステムのアンマウントに対処します。
- 新しい Lustre モジュールをロードします。

#### # 4. Lustre Network Initialization

- Lustre のネットワーク設定を初期化します。
- 必要なネットワークパラメータを設定します。

#### # 5. Mount FSx filesystem

- このセクションでは、お使いの環境向けに値を調整する必要があります。

#### # 6. Apply tunings

- LRU (ロッキリソースユニット) の調整:
  - `lru_max_age=600000`
  - `lru_size` CPU 数に基づいて計算
- クライアントキャッシュコントロール: `max_cached_mb=64`
- RPC コントロール:
  - `OST max_rpcs_in_flight=32`
  - `MDC max_rpcs_in_flight=64`
  - `MDC max_mod_rpcs_in_flight=50`

#### # 7. Verify tunings

- 適用されたすべてのチューニングを検証します。
- 各パラメータの成功または警告を報告します。

#### # 8. Setup persistence

- このセクションでも同様にお使いの環境向けに値を調整する必要があります。
- OS のバージョン (AL2 または AL2023) を自動検出し、どの Systemd サービスを適用するか判断します。
- システムが起動します。



- Systemd が lustre-tunings サービスを開始します (WantedBy=multi-user.target により)。
- サービスが apply\_lustre\_tunings.sh 実行します。こちらは、
  - ファイルシステムがマウントされているかどうかを確認します。
  - マウントされていない場合、ファイルシステムをマウントします。
  - マウントが完了するまで待機します (最大 5 分)。
  - マウントが完了すると、チューニングパラメータを適用します。
- 設定は、再起動するまでアクティブな状態のままです。
- スクリプトが完成すると、サービスは終了します。
  - Systemd は、サービスを [アクティブ (終了済み)] にマークします。
- 次回の再起動時にプロセスが繰り返されます。

## 起動テンプレートの作成

1. Amazon EC2 コンソールの <https://console.aws.amazon.com/ec2/> を開いてください。
2. [起動テンプレート] を選択します。
3. [起動テンプレートの作成] を選択してください。
4. [高度な詳細] で、[ユーザーデータ] のセクションを見つけます。
5. 以下のスクリプトを貼り付け、必要に応じて更新します。

### Important

セクション # 5. Mount FSx filesystem と、セクション # 8. Setup persistence の apply\_lustre\_tunings.sh の setup\_persistence() 関数の両方で、お使いの環境のこれらの値を調整します。

```
FSX_DNS="<your-fsx-filesystem-dns>" # Needs to be adjusted.
MOUNT_NAME="<your-mount-name>" # Needs to be adjusted.
MOUNT_POINT="</your/mount/point>" # Needs to be adjusted.
```

```
MIME-Version: 1.0
Content-Type: multipart/mixed; boundary=="MYBOUNDARY=="
--==MYBOUNDARY==
```

```
Content-Type: text/x-shellscript; charset="us-ascii"
#!/bin/bash
exec 1> >(logger -s -t $(basename $0)) 2>&1
Function definitions
check_success() {
 if [$? -eq 0]; then
 echo "SUCCESS: $1"
 else
 echo "FAILED: $1"
 return 1
 fi
}
}
apply_tunings() {
 local NUM_CPUS=$(nproc)
 local LRU_SIZE=$((100 * NUM_CPUS))
 local params=(
 "ldlm.namespaces.*.lru_max_age=600000"
 "ldlm.namespaces.*.lru_size=$LRU_SIZE"
 "llite.*.max_cached_mb=64"
 "osc.*OST*.max_rpcs_in_flight=32"
 "mdc.*.max_rpcs_in_flight=64"
 "mdc.*.max_mod_rpcs_in_flight=50"
)
 for param in "${params[@]"; do
 lctl set_param $param
 check_success "Set ${param%%=*}"
 done
}
verify_param() {
 local param=$1
 local expected=$2
 local actual=$3

 if ["$actual" == "$expected"]; then
 echo "SUCCESS: $param is correctly set to $expected"
 else
 echo "WARNING: $param is set to $actual (expected $expected)"
 fi
}
}
verify_tunings() {
 local NUM_CPUS=$(nproc)
 local LRU_SIZE=$((100 * NUM_CPUS))
 local params=(
 "ldlm.namespaces.*.lru_max_age:600000"
```

```

 "ldlm.namespaces.*.lru_size:$LRU_SIZE"
 "llite.*.max_cached_mb:64"
 "osc.*OST*.max_rpcs_in_flight:32"
 "mdc.*.max_rpcs_in_flight:64"
 "mdc.*.max_mod_rpcs_in_flight:50"
)
echo "Verifying all parameters:"
for param in "${params[@]"; do
 name="${param%:*}"
 expected="${param#*:}"
 actual=$(lctl get_param -n $name | head -1)
 verify_param "${name##*}" "$expected" "$actual"
done
}

setup_persistence() {
 # Create functions file
 cat << 'EOF' > /usr/local/bin/lustre_functions.sh
#!/bin/bash
apply_lustre_tunings() {
 local NUM_CPUS=$(nproc)
 local LRU_SIZE=$((100 * NUM_CPUS))

 echo "Applying Lustre performance tunings..."
 lctl set_param ldlm.namespaces.*.lru_max_age=600000
 lctl set_param ldlm.namespaces.*.lru_size=$LRU_SIZE
 lctl set_param llite.*.max_cached_mb=64
 lctl set_param osc.*OST*.max_rpcs_in_flight=32
 lctl set_param mdc.*.max_rpcs_in_flight=64
 lctl set_param mdc.*.max_mod_rpcs_in_flight=50
}
EOF
 # Create tuning script
 cat << 'EOF' > /usr/local/bin/apply_lustre_tunings.sh
#!/bin/bash
exec 1> >(logger -s -t $(basename $0)) 2>&1
Source the functions
source /usr/local/bin/lustre_functions.sh
FSx details
FSX_DNS="<<your-fsx-filesystem-dns>" # Needs to be adjusted.
MOUNT_NAME="<<your-mount-name>" # Needs to be adjusted.
MOUNT_POINT="<</your/mount/point>" # Needs to be adjusted.
Function to check if Lustre is mounted
is_lustre_mounted() {
 mount | grep -q "type lustre"

```

```
}
Function to mount Lustre
mount_lustre() {
 echo "Mounting Lustre filesystem..."
 mkdir -p ${MOUNT_POINT}
 mount -t lustre ${FSX_DNS}@tcp:/${MOUNT_NAME} ${MOUNT_POINT}
 return $?
}
Main execution
Try to mount if not already mounted
if ! is_lustre_mounted; then
 echo "Lustre filesystem not mounted, attempting to mount..."
 mount_lustre
fi
Wait for successful mount (up to 5 minutes)
for i in {1..30}; do
 if is_lustre_mounted; then
 echo "Lustre filesystem mounted, applying tunings..."
 apply_lustre_tunings
 exit 0
 fi
 echo "Waiting for Lustre filesystem to be mounted... (attempt $i/30)"
 sleep 10
done
echo "Timeout waiting for Lustre filesystem mount"
exit 1
EOF
Create systemd service
cat << 'EOF' > /etc/systemd/system/lustre-tunings.service
[Unit]
Description=Apply Lustre Performance Tunings
After=network.target remote-fs.target
StartLimitIntervalSec=0
[Service]
Type=oneshot
ExecStart=/usr/local/bin/apply_lustre_tunings.sh
RemainAfterExit=yes
Restart=on-failure
RestartSec=30
[Install]
WantedBy=multi-user.target
EOF
 chmod +x /usr/local/bin/lustre_functions.sh
 chmod +x /usr/local/bin/apply_lustre_tunings.sh
```

```
systemctl enable lustre-tunings.service
systemctl start lustre-tunings.service
}
echo "Starting FSx for Lustre configuration..."
1. Install Lustre client
if grep -q 'VERSION="2"' /etc/os-release; then
 amazon-linux-extras install -y lustre
elif grep -q 'VERSION="2023"' /etc/os-release; then
 dnf install -y lustre-client
fi
check_success "Install Lustre client"
2. Apply network and RPC tunings
export PATH=$PATH:/usr/sbin
echo "Applying network and RPC tunings..."
if ! grep -q "options ptlrpc ptlrpcd_per_cpt_max" /etc/modprobe.d/modprobe.conf; then
 echo "options ptlrpc ptlrpcd_per_cpt_max=64" | tee -a /etc/modprobe.d/
modprobe.conf
 echo "options ksocklnd credits=2560" | tee -a /etc/modprobe.d/modprobe.conf
fi
3. Load Lustre modules
modprobe lustre
check_success "Load Lustre modules" || exit 1
4. Lustre Network Initialization
lctl network up
check_success "Initialize Lustre networking" || exit 1
5. Mount FSx filesystem
FSX_DNS="" # Needs to be adjusted.
MOUNT_NAME="" # Needs to be adjusted.
MOUNT_POINT="" # Needs to be adjusted.
if [! -z "$FSX_DNS"] && [! -z "$MOUNT_NAME"]; then
 mkdir -p $MOUNT_POINT
 mount -t lustre ${FSX_DNS}@tcp:/${MOUNT_NAME} ${MOUNT_POINT}
 check_success "Mount FSx filesystem"
fi
6. Apply tunings
apply_tunings
7. Verify tunings
verify_tunings
8. Setup persistence
setup_persistence
echo "FSx for Lustre configuration completed."
---MYBOUNDARY---
```

6. Amazon EKS ノードグループを作成するときは、この起動テンプレートを選択します。詳細については、「[the section called “作成”](#)」を参照してください。

## 関連情報

- [the section called “ドライバーをデプロイする”](#)
- [the section called “最適化 \(EFA\)”](#)
- [Amazon FSx for Lustre のパフォーマンス](#)

## FSx for NetApp ONTAP で高性能なアプリケーションストレージを使用する

NetApp Trident は、コンテナストレージインターフェイス (CSI) 準拠のドライバーを使用して、動的なストレージオーケストレーションを提供します。これにより、Amazon EKS クラスターが Amazon FSx for NetApp ONTAP ファイルシステムによってバックアップされた永続的ボリューム (PV) のライフサイクルを管理できるようになります。Amazon FSx for NetApp ONTAP CSI ドライバーは Amazon EKS Hybrid Nodes と互換性がないことに注意してください。開始するには、NetApp Trident ドキュメントの「[Use Trident with Amazon FSx for NetApp ONTAP](#)」を参照してください。

Amazon FSx for NetApp ONTAP は、クラウドでフルマネージド型の ONTAP ファイルシステムを起動して実行できるストレージサービスです。ONTAP は、NetApp のファイルシステムテクノロジーであり、幅広く採用されているデータアクセス機能とデータ管理機能を提供します。FSx for ONTAP によって、オンプレミスの NetApp ファイルシステムの機能、パフォーマンス、および API に、フルマネージド型 AWS サービスの俊敏性、スケーラビリティが加わり、簡素化されます。詳細については、「[FSx for ONTAP ユーザーガイド](#)」を参照してください。

### Important

Amazon EBS CSI ドライバーと共に Amazon FSx for NetApp ONTAP を使用して EBS ボリュームをプロビジョニングしている場合は、multipath.conf ファイルに EBS デバイスを使用しないように指定する必要があります。サポートされている方法については、「[Configuration File Blacklist](#)」を参照してください。以下はその例です。

```
defaults {
 user_friendly_names yes
 find_multipaths no
```

```
}
blacklist {
 device {
 vendor "NVME"
 product "Amazon Elastic Block Store"
 }
}
```

## Amazon FSx for OpenZFS でデータストレージを使用する

Amazon FSx for OpenZFS は、オンプレミスの ZFS またはその他の Linux ベースのファイルサーバーから AWS にデータを簡単に移動できるようにするフルマネージド型のファイルストレージサービスです。データの移動は、アプリケーションコードやデータの管理方法を変更しなくても実行できます。Amazon FSx for OpenZFS は、オープンソースの OpenZFS ファイルシステム上に構築された、信頼性が高く、スケーラブルで効率的で機能豊富なファイルストレージを提供しています。これらの機能は、フルマネージド型の AWS サービスの俊敏性、スケーラビリティ、およびシンプルさを兼ね備えています。詳細については、「[Amazon FSx for OpenZFS ユーザーガイド](#)」を参照してください。

FSx for OpenZFS コンテナストレージインターフェイス (CSI) ドライバーは、Amazon EKS クラスターが FSx for OpenZFS ボリュームのライフサイクルを管理できるようにする CSI インターフェイスを提供します。Amazon FSx for OpenZFS CSI ドライバーは Amazon EKS Hybrid Nodes と互換性がないことに注意してください。FSx for OpenZFS CSI ドライバーを Amazon EKS クラスターにデプロイするには、GitHub の「[aws-fsx-openszfs-csi-driver](#)」を参照してください。

## Amazon File Cache を使用してレイテンシーを最小化する

Amazon File Cache は、データの保存場所にかかわらず、ファイルデータの処理に使用される AWS のフルマネージド型の高速キャッシュです。Amazon File Cache は、初回アクセス時にデータをキャッシュに自動的にロードし、使用されていないときにデータを解放します。詳細については、「[Amazon File Cache ユーザーガイド](#)」を参照してください。

Amazon File Cache Container Storage Interface (CSI) ドライバーは、Amazon EKS クラスターが Amazon ファイルキャッシュのライフサイクルを管理できるようにする CSI インターフェイスを提供します。Amazon File Cache CSI ドライバーは Amazon EKS Hybrid Nodes と互換性がないことに注意してください。Amazon File Cache CSI ドライバーを Amazon EKS クラスターにデプロイするには、GitHub の「[aws-file-cache-csi-driver](#)」を参照してください。

# Mountpoint for Amazon S3 CSI ドライバーを使用して Amazon S3 オブジェクトにアクセスする

[Mountpoint for Amazon S3 Container Storage Interface \(CSI\) ドライバー](#)を使用すると、Kubernetes アプリケーションはファイルシステムインターフェイスを介して Amazon S3 オブジェクトにアクセスできるため、アプリケーションコードを変更せずに高い総スループットを実現できます。[Mountpoint for Amazon S3](#) 上に構築された CSI ドライバーは、Amazon S3 バケットを Amazon EKS 内のコンテナやセルフマネージド型 Kubernetes クラスターからアクセスできるボリュームとして提供します。

## 考慮事項

- Mountpoint for Amazon S3 CSI ドライバーは、Windows ベースのコンテナイメージと現時点では互換性がありません。
- Mountpoint for Amazon S3 CSI ドライバーは現在、Amazon EKS Hybrid Nodes と互換性がありません。
- Mountpoint for Amazon S3 CSI ドライバーは、AWS Fargate をサポートしていません。ただし、Amazon EC2 で実行されているコンテナ (Amazon EKS またはカスタム Kubernetes インストール) はサポートされています。
- Mountpoint for Amazon S3 CSI ドライバーは、静的プロビジョニングのみをサポートします。動的プロビジョニング、つまり新しいバケットの作成はサポートされていません。

### Note

静的プロビジョニングとは、PersistentVolume オブジェクトの `volumeAttributes` で `bucketName` として指定されている既存の Amazon S3 バケットを使用することです。詳細については、GitHub の「[Static Provisioning](#)」を参照してください。

- Mountpoint for Amazon S3 CSI ドライバーでマウントされたボリュームは、すべての POSIX ファイルシステム機能をサポートしていません。ファイルシステムの動作の詳細については、GitHub の「[Mountpoint for Amazon S3 file system behavior](#)」を参照してください。

ドライバーのデプロイ方法の詳細は、「[the section called “ドライバーをデプロイする”](#)」を参照してください。ドライバーの削除方法の詳細は、「[the section called “ドライバーを削除する”](#)」を参照してください。



# Mountpoint for Amazon S3 ドライバーをデプロイする

[Mountpoint for Amazon S3 Container Storage Interface \(CSI\) ドライバー](#)を使用すると、Kubernetes アプリケーションはファイルシステムインターフェイスを介して Amazon S3 オブジェクトにアクセスできるため、アプリケーションコードを変更せずに高い総スループットを実現できます。

こちらの手順では、[Mountpoint for Amazon S3 CSI Amazon EKS ドライバー](#)をデプロイする方法を解説します。先に進む前に、[考慮事項](#)をご一読ください。

## 前提条件

- クラスターの既存の AWS Identity and Access Management (IAM) OpenID Connect (OIDC) プロバイダー。既に存在しているかどうかを確認する、または作成するには「[the section called “IAM OIDC プロバイダー”](#)」を参照してください。
- ご使用のデバイスまたは AWS CloudShell で、AWS CLI のバージョン 2.12.3 以降がインストールおよび設定されていること。
- デバイスまたは AWS クラウドシェル に、kubectl コマンドラインツールがインストールされていること。バージョンはご使用のクラスターの Kubernetes バージョンと同じか、1 つ前のマイナーバージョン以前、あるいはそれより新しいバージョンが使用できます。例えば、クラスターのバージョンが 1.29 である場合、kubectl のバージョン 1.28、1.29、または 1.30 が使用できます。kubectl をインストールまたはアップグレードする方法については「[the section called “kubectl と eksctl のセットアップ”](#)」を参照してください。

## ステップ 1: IAM ポリシーを作成する

Mountpoint for Amazon S3 CSI ドライバーには、ファイルシステムとやり取りするための Amazon S3 アクセス許可が必要です。このセクションでは、必要な権限を付与する IAM ポリシーを作成する方法を説明します。

以下のポリシー例は、Mountpoint の IAM アクセス許可に関する推奨事項に従っています。別の方法として、AWS マネージドポリシー [AmazonS3FullAccess](#) を使用することもできますが、このマネージドポリシーでは Mountpoint に必要以上のアクセス権限が付与されます。

Mountpoint の推奨アクセス許可の詳細については、GitHub の「[Mountpoint IAM permissions](#)」を参照してください。

1. IAM コンソール (<https://console.aws.amazon.com/iam/>) を開きます。
2. 左のナビゲーションペインの [ポリシー] を選択します。

3. [ポリシー] ページで、[ポリシーの作成] を選択します。
4. [ポリシーエディター] には [JSON] を選択します。
5. 以下の [ポリシー] をコピーして、エディタに貼り付けます。

 Important

amzn-s3-demo-bucket1 を Amazon S3 バケットの名前に置き換えます。

```
{
 "Version": "2012-10-17",
 "Statement": [
 {
 "Sid": "MountpointFullBucketAccess",
 "Effect": "Allow",
 "Action": [
 "s3:ListBucket"
],
 "Resource": [
 "arn:aws:s3:::amzn-s3-demo-bucket1"
]
 },
 {
 "Sid": "MountpointFullObjectAccess",
 "Effect": "Allow",
 "Action": [
 "s3:GetObject",
 "s3:PutObject",
 "s3:AbortMultipartUpload",
 "s3:DeleteObject"
],
 "Resource": [
 "arn:aws:s3:::amzn-s3-demo-bucket1/*"
]
 }
]
}
```

Amazon S3 Express One Zone ストレージクラスで導入されたディレクトリバケットは、汎用バケットとは異なる認証メカニズムを使用します。s3:\* アクションではな

く、s3express:CreateSession アクションを使用する必要があります。ディレクトリバケットの詳細については、Amazon S3 ユーザーガイドの「[ディレクトリバケット](#)」を参照してください。

以下は、ディレクトリバケットに使用する最小特権ポリシーの例です。

```
{
 "Version": "2012-10-17",
 "Statement": [
 {
 "Effect": "Allow",
 "Action": "s3express:CreateSession",
 "Resource": "arn:aws:s3express:us-west-2:111122223333:bucket/amzn-s3-demo-bucket1--usw2-az1--x-s3"
 }
]
}
```

6. [次へ] を選択します。
7. [確認と作成] ページで、ポリシーに名前を付けます。このウォークスルーでは、名前に AmazonS3CSIDriverPolicy を使用します。
8. [Create policy] (ポリシーの作成) を選択します。

## ステップ 2: IAM ロールを作成する

Mountpoint for Amazon S3 CSI ドライバーには、ファイルシステムとやり取りするための Amazon S3 アクセス許可が必要です。このセクションでは、IAM ロールを作成して、これらの権限を委任する方法を示します。このロールを作成するには、次のいずれかのツールを使用できます。

- [the section called “eksctl”](#)
- [the section called “AWS Management Console”](#)
- [the section called “AWS CLI”](#)

### Note

IAM ポリシー AmazonS3CSIDriverPolicy は、前のセクションで作成しました。

## eksctl

**eksctl** を使用して Mountpoint for Amazon S3 CSI ドライバーの IAM ロールを作成するには

IAM ロールと Kubernetes サービスアカウントを作成するには、以下のコマンドを実行します。これらのコマンドはまた、AmazonS3CSIDriverPolicy IAM ポリシーをロールにアタッチし、Kubernetes サービスアカウント (s3-csi-controller-sa) に IAM ロールの Amazon リソースネーム (ARN) の注釈を付け、Kubernetes サービスアカウント名を IAM ロールの信頼ポリシーに追加します。

```
CLUSTER_NAME=my-cluster
REGION=region-code
ROLE_NAME=AmazonEKS_S3_CSI_DriverRole
POLICY_ARN=AmazonEKS_S3_CSI_DriverRole_ARN
eksctl create iamserviceaccount \
 --name s3-csi-driver-sa \
 --namespace kube-system \
 --cluster $CLUSTER_NAME \
 --attach-policy-arn $POLICY_ARN \
 --approve \
 --role-name $ROLE_NAME \
 --region $REGION \
 --role-only
```

## AWS Management Console

1. IAM コンソール (<https://console.aws.amazon.com/iam/>) を開きます。
2. 左のナビゲーションペインで、[ロール] を選択してください。
3. [ロール] ページで、[ロールの作成] を選択してください。
4. [信頼されたエンティティを選択] ページで、以下の操作を実行します。
  - a. [信頼されたエンティティの種類] セクションで、[ウェブ アイデンティティ] を選択します。
  - b. [Identity provider] (ID プロバイダー) で、(Amazon EKS の [Overview] (概要) に示されているように) クラスターに [OpenID Connect provider URL] (OpenID Connect プロバイダーの URL) を選択します。  
  
URL が表示されない場合は、「[前提条件](#)」を確認してください。
  - c. [対象者] で [sts.amazonaws.com] を選択します。
  - d. [次へ] を選択します。
5. [アクセス許可を追加] ページで、以下を実行します。

- a. [フィルターポリシー] ボックスに AmazonS3CSIDriverPolicy と入力します。

 Note

このポリシーは、前のセクションで作成されました。

- b. 検索で返された AmazonS3CSIDriverPolicy 結果の左にあるチェックボックスを選択します。
  - c. [次へ] を選択します。
6. [名前を付けて、レビューし、作成する] ページで、以下の操作を実行します。
    - a. [ロール名] に、AmazonEKS\_S3\_CSI\_DriverRole などロールの一意の名前を入力します。
    - b. [タグの追加 (オプション)] で、タグをキーバリューのペアとして添付して、メタデータをロールに追加します。IAM でのタグの使用に関する詳細については『IAM ユーザーガイド』の「[IAM リソースにタグを付ける](#)」を参照してください。
    - c. [ロールの作成] を選択してください。
  7. ロールが作成されたら、コンソールでロールを選択して編集用を開きます。
  8. [信頼関係] タブを選択し、続いて [信頼ポリシーの編集] を選択します。
  9. 次のように表示されます。

```
"oidc.eks.region-code.amazonaws.com/id/EXAMPLED539D4633E53DE1B71EXAMPLE:aud":
 "sts.amazonaws.com"
```

前の行の末尾にカンマを追加し、前の行の後に次の行を追加します。*region-code* を、クラスターのある AWS リージョンに置き換えます。*EXAMPLED539D4633E53DE1B71EXAMPLE* をクラスターの OIDC プロバイダー ID に置き換えます。

```
"oidc.eks.region-code.amazonaws.com/id/EXAMPLED539D4633E53DE1B71EXAMPLE:sub":
 "system:serviceaccount:kube-system:s3-csi-driver-sa"
```

- 10.Condition 演算子が "StringEquals" に設定されていることを確認します。

- 11[ポリシーの更新] を選択して終了します。

## AWS CLI

1. クラスターの OIDC プロバイダーの URL を表示します。*#####* の部分は自分のクラスター名に置き換えます。コマンドの出力が None の場合は、「[前提条件](#)」を確認してください。

```
aws eks describe-cluster --name my-cluster --query "cluster.identity.oidc.issuer" --output text
```

出力例は次のとおりです。

```
https://oidc.eks.region-code.amazonaws.com/id/EXAMPLED539D4633E53DE1B71EXAMPLE
```

2. IAM ロールを作成し、Kubernetes サービスアカウントに AssumeRoleWithWebIdentity アクションを付与します。

- a. 次の内容を `aws-s3-csi-driver-trust-policy.json` という名前のファイルにコピーします。**111122223333** は、ご自分のアカウント ID に置き換えます。**EXAMPLED539D4633E53DE1B71EXAMPLE** および **region-code** を、前のステップで返された値にそれぞれ置き換えます。

```
{
 "Version": "2012-10-17",
 "Statement": [
 {
 "Effect": "Allow",
 "Principal": {
 "Federated": "arn:aws:iam::111122223333:oidc-provider/oidc.eks.region-code.amazonaws.com/id/EXAMPLED539D4633E53DE1B71EXAMPLE"
 },
 "Action": "sts:AssumeRoleWithWebIdentity",
 "Condition": {
 "StringEquals": {
 "oidc.eks.region-code.amazonaws.com/id/EXAMPLED539D4633E53DE1B71EXAMPLE:sub": "system:serviceaccount:kube-system:s3-csi-driver-sa",
 "oidc.eks.region-code.amazonaws.com/id/EXAMPLED539D4633E53DE1B71EXAMPLE:aud": "sts.amazonaws.com"
 }
 }
 }
]
}
```

- b. ロールを作成します。**AmazonEKS\_S3\_CSI\_DriverRole** を別の名前に変更できますが、変更する場合は、後の手順でも変更してください。

```
aws iam create-role \
 --role-name AmazonEKS_S3_CSI_DriverRole \
 --assume-role-policy-document file://"aws-s3-csi-driver-trust-policy.json"
```

3. 次のコマンドを使用して IAM ロールを作成し、そのロールに IAM ポリシーをアタッチします。

```
aws iam attach-role-policy \
 --policy-arn arn:aws:iam::aws:policy/AmazonS3CSIDriverPolicy \
 --role-name AmazonEKS_S3_CSI_DriverRole
```

 Note

IAM ポリシー AmazonS3CSIDriverPolicy は、前のセクションで作成しました。

4. ドライバーを Amazon EKS アドオンとしてインストールする場合は、この手順をスキップしてください。ドライバーのセルフマネージド型インストールでは、作成した IAM ロールの ARN の注釈が付けられた Kubernetes サービスアカウントを作成します。

a. 次の内容を mountpoint-s3-service-account.yaml という名前のファイルに保存します。**111122223333** は、ご自分のアカウント ID に置き換えます。

```

apiVersion: v1
kind: ServiceAccount
metadata:
 labels:
 app.kubernetes.io/name: aws-mountpoint-s3-csi-driver
 name: mountpoint-s3-csi-controller-sa
 namespace: kube-system
 annotations:
 eks.amazonaws.com/role-arn: arn:aws:iam::111122223333:role/
 AmazonEKS_S3_CSI_DriverRole
```

b. クラスター上で Kubernetes サービスアカウントを作成します。Kubernetes サービスアカウント (mountpoint-s3-csi-controller-sa) は、**AmazonEKS\_S3\_CSI\_DriverRole** という名前で作成した IAM ロールの注釈が付けられています。

```
kubectl apply -f mountpoint-s3-service-account.yaml
```

 Note

その手順でプラグインがデプロイされると、s3-csi-driver-sa という名前のサービスアカウントを作成して使用するよう設定されます。

### ステップ 3: Mountpoint for Amazon S3 CSI ドライバーをインストールする

Mountpoint for Amazon S3 CSI ドライバーは、Amazon EKS アドオンを使用してインストールできます。次のツールを使用して、アドオンをクラスターに追加できます。

- [the section called “eksctl”](#)
- [the section called “AWS Management Console”](#)
- [the section called “AWS CLI”](#)

または、セルフマネージド型インストールとして Mountpoint for Amazon S3 CSI ドライバーをインストールできます。セルフマネージド型インストールを実行する方法については、GitHub の「[Installation](#)」を参照してください。

v1.8.0 以降、CSI ドライバーの Pod を許容するようにテイントを設定できます。これを行うには、`node.tolerations` を使用して許容するカスタムのテイントセットを指定するか、`node.tolerateAllTaints` を使用してすべてのテイントを許容します。詳細については、Kubernetes ドキュメントの「[テイントと容認](#)」を参照してください。

eksctl

**eksctl** を使用して Amazon S3 CSI アドオンを追加するには

以下のコマンドを実行してください。*my-cluster* をクラスターの名前に、*111122223333* をアカウント ID に、*AmazonEKS\_S3\_CSI\_DriverRole* を [前に作成した IAM ロール](#) の名前に置き換えます。

```
eksctl create addon --name aws-mountpoint-s3-csi-driver --cluster my-cluster \
 --service-account-role-arn arn:aws:iam::111122223333:role/AmazonEKS_S3_CSI_DriverRole
--force
```

*--force* オプションを削除した状態で Amazon EKS アドオンを更新する際に、そのアドオン設定のいずれかが現状の設定と競合している場合、その更新は失敗します。この場合、競合を解決するため



のエラーメッセージが表示されます。このオプションを指定する前に、自分が管理する必要がある設定を Amazon EKS アドオンが管理していないことを確認してください。これらの設定は、このオプションの指定により上書きされます。この設定のその他のオプションの詳細については、eksctl ドキュメントの「[アドオン](#)」を参照してください。Amazon EKS Kubernetes フィールド管理の詳細については、「[the section called “カスタマイズ可能なフィールド”](#)」を参照してください。

設定ファイルをによって eksctl をカスタマイズできます。詳細については、eksctl ドキュメントの「[Working with configuration values](#)」を参照してください。次の例は、すべてのテイントを許容する方法を示しています。

```
config.yaml
...

addons:
- name: aws-mountpoint-s3-csi-driver
 serviceAccountRoleARN: arn:aws:iam::111122223333:role/AmazonEKS_S3_CSI_DriverRole
 configurationValues: |-
 node:
 tolerateAllTaints: true
```

## AWS Management Console

1. [Amazon EKS コンソール](#)を開きます。
2. 左のナビゲーションペインで [Clusters] (クラスター) を選択してください。
3. Mountpoint for Amazon S3 CSI アドオンを設定するクラスターの名前を選択します。
4. [アドオン] タブを選択してください。
5. [その他のアドオンを入手] を選択してください。
6. [アドオンを選択] ページで、次の操作を行います。
  - a. [Amazon EKS アドオン] セクションで、[Mountpoint for Amazon S3 CSI ドライバー] チェックボックスを選択します。
  - b. [次へ] を選択します。
7. [選択したアドオンセッティングの設定] ページで、次の操作を行います。
  - a. 使用する [バージョン] を選択します。
  - b. [IAM ロールを選択] で、Mountpoint for Amazon S3 CSI ドライバーの IAM ポリシーをアタッチした IAM ロールの名前を選択します。
  - c. (オプション) [オプション設定] を展開した後、[競合解決方法] を更新します。[上書きする] を選択すると、既存のアドオンでの 1 つ以上の設定が、Amazon EKS アドオンの設定で上書きさ

れます。このオプションが有効でない状態で既存の設定との競合が発生する場合は、オペレーションが失敗します。表示されたエラーメッセージを使用して、競合をトラブルシューティングできます。このオプションを選択する前に、自分で管理する必要のある設定が、Amazon EKS アドオンにより管理されていないことを確認してください。

- d. (オプション) [オプション設定] を展開した後、[設定値] フィールドで許容値を設定します。
  - e. [次へ] を選択します。
8. [確認と追加] ページで、[作成] を選択します。アドオンのインストールが完了した後、インストールしたアドオンが表示されます。

## AWS CLI

AWS CLI を使用して Mountpoint for Amazon S3 CSI アドオンを追加するには

以下のコマンドを実行してください。*my-cluster* をクラスターの名前に、*111122223333* をアカウント ID に、*AmazonEKS\_S3\_CSI\_DriverRole* を前に作成したロールの名前に置き換えます。

```
aws eks create-addon --cluster-name my-cluster --addon-name aws-mountpoint-s3-csi-driver \
 --service-account-role-arn arn:aws:iam::111122223333:role/AmazonEKS_S3_CSI_DriverRole
```

--configuration-values フラグを使用してコマンドをカスタマイズできます。次の代替例は、すべてのテイントを許容する方法を示しています。

```
aws eks create-addon --cluster-name my-cluster --addon-name aws-mountpoint-s3-csi-driver \
 --service-account-role-arn arn:aws:iam::111122223333:role/AmazonEKS_S3_CSI_DriverRole \
 --configuration-values '{"node":{"tolerateAllTaints":true}}'
```

## ステップ 4: Mountpoint for Amazon S3 を設定する

ほとんどの場合、Mountpoint for Amazon S3 はバケット名のみで設定できます。Mountpoint for Amazon S3 を設定する手順については、GitHub の「[Configuring Mountpoint for Amazon S3](#)」を参照してください。

## ステップ 5: サンプルアプリケーションをデプロイする

既存の Amazon S3 バケットのドライバーにデプロイできます。詳細については、GitHub の「[Static provisioning](#)」を参照してください。

# Mountpoint for Amazon S3 Amazon EKS アドオンを削除する

[Mountpoint for Amazon S3 CSI ドライバー](#)を削除する方法は 2 通りあります。

- クラスター上のアドオンソフトウェアを保持する — このオプションではすべての設定の Amazon EKS 管理が削除されます。また、Amazon EKS による更新の通知機能や、更新の開始後の Amazon EKS アドオンの自動更新機能も削除されます。ただし、クラスター上のアドオンソフトウェアは保持されます。このオプションを選択すると、アドオンは Amazon EKS アドオンではなく、セルフマネージド型インストールになります。このオプションを使用すると、アドオンのダウンタイムは発生しません。この手順のコマンドには、このオプションを使用します。
- クラスターからアドオンソフトウェアを完全に削除する — クラスターから Amazon EKS アドオンを削除するのはアドオンに依存するリソースがクラスター上にない場合のみとすることをお勧めします。このオプションを実行するには、この手順で使用するコマンドから `--preserve` を削除します。

アドオンに IAM アカウントが関連付けられている場合、その IAM アカウントは削除されません。

次のツールを使用して、Amazon S3 CSI アドオンを削除できます。

- [the section called “eksctl”](#)
- [the section called “AWS Management Console”](#)
- [the section called “AWS CLI”](#)

## eksctl

**eksctl** を使用して Amazon S3 CSI アドオンを削除するには

*my-cluster* を自分のクラスター名に置き換えた上で、次のコマンドを実行します。

```
eksctl delete addon --cluster my-cluster --name aws-mountpoint-s3-csi-driver --preserve
```

## AWS Management Console

1. [Amazon EKS コンソール](#)を開きます。
2. 左のナビゲーションペインで [Clusters] (クラスター) を選択してください。
3. Amazon EBS CSI アドオンを削除するクラスターの名前を選択します。

4. [アドオン] タブを選択してください。
5. [Mountpoint for Amazon S3 CSI ドライバー] を選択します。
6. [削除] を選択してください。
7. [削除: aws-mountpoint-s3-csi-driver] 確認ダイアログボックスで、以下の操作を行います。
  - a. Amazon EKS でのアドオンの設定の管理を停止する場合は[クラスターで保持する] を選択してください。クラスターでアドオンソフトウェアを保持する場合はこの操作を実行してください。これはアドオンのすべての設定を自分で管理できるようにするためです。
  - b. aws-mountpoint-s3-csi-driver と入力してください。
  - c. [削除] を選択します。

## AWS CLI

AWS CLI を使用して Amazon S3 CSI アドオンを削除するには

*my-cluster* を自分のクラスター名に置き換えた上で、次のコマンドを実行します。

```
aws eks delete-addon --cluster-name my-cluster --addon-name aws-mountpoint-s3-csi-driver --preserve
```

## CSI ボリュームのためにスナップショット機能を有効にする

スナップショット機能を使用すると、データのポイントインタイムコピーが可能になります。Kubernetes でこの機能を使用するには、スナップショットをサポートする CSI ドライバー (Amazon EBS CSI ドライバーなど) と CSI スナップショットコントローラーの両方が必要です。スナップショットコントローラーは、Amazon EKS マネージドアドオンまたはセルフマネージドインストールのいずれかとして使用できます。

CSI スナップショットコントローラーを使用する場合の考慮事項を以下に示します。

- スナップショットコントローラーは、スナップショット機能を備えた CSI ドライバーと一緒にインストールする必要があります。Amazon EBS CSI ドライバーのインストール手順については、「[the section called “Amazon EBS”](#)」を参照してください。
- Kubernetes では、プロビジョナー [kubernetes.io/aws-ebs](https://kubernetes.io/aws-ebs) で StorageClass を使用する Amazon EBS ボリュームなど、CSI 移行経由で提供されるボリュームのスナップショットはサポートされていません。ボリュームは CSI ドライバープロビジョナー、[ebs.csi.aws.com](https://ebs.csi.aws.com) を参照する StorageClass を使用して作成する必要があります。

- Amazon EKS Auto Mode にはスナップショットコントローラーは含まれません。EKS Auto Mode のストレージ機能はスナップショットコントローラーと互換性があります。

CSI スナップショットコントローラーは Amazon EKS マネージドアドオンを使用してインストールすることをお勧めします。このアドオンには、Amazon EKS でスナップショットを作成および管理するために必要なカスタムリソース定義 (CRD) が含まれています。Amazon EKS アドオンをクラスターに追加するには、「[the section called “アドオンを作成する”](#)」を参照してください。アドオンの詳細については、「[the section called “Amazon EKS アドオン”](#)」を参照してください。

または、CSI スナップショットコントローラーの自己管理型インストールを希望する場合は、GitHub でのアップストリーム Kubernetes external-snapshotter での「[Usage](#)」を参照してください。

# Amazon EKS クラスターのネットワーキングを設定する

Amazon EKS クラスターは VPC 内に作成されます。AWS インフラストラクチャ上で実行されるノードに対しては、ポッドネットワークは Amazon VPC Container Network Interface (CNI) プラグインによって提供されます。独自のインフラストラクチャでノードを実行している場合は、「[the section called “CNI を設定する”](#)」を参照してください。この章では、クラスターのネットワークの詳細に関する以下のトピックについて説明します。

## トピック

- [管理コンソールから既存の VPC サブネットを Amazon EKS クラスターに追加する](#)
- [VPC とサブネットの Amazon EKS ネットワーキング要件を表示する](#)
- [Amazon EKS クラスターの Amazon VPC を作成する](#)
- [クラスターの Amazon EKS セキュリティグループ要件を表示する](#)
- [Amazon EKS クラスターのネットワーキングアドオンを管理する](#)

## 管理コンソールから既存の VPC サブネットを Amazon EKS クラスターに追加する

1. 管理コンソールでクラスターに移動します。
2. [ネットワーキング] タブから [VPC リソースを管理] を選択します。
3. [サブネット] ドロップダウンから、クラスターの VPC からのサブネットを追加で選択します。

新しい VPC サブネットを作成するには:

- [EKS サブネットの要件を確認します。](#)
- 「Amazon Virtual Private Cloud ユーザーガイド」の「[Create a Subnet](#)」を参照してください。

## VPC とサブネットの Amazon EKS ネットワーキング要件を表示する

クラスターを作成する際には、[VPC](#) と、異なるアベイラビリティーゾーンに存在する 2 つ以上のサブネットを指定します。このトピックでは、クラスターで使用する VPC およびサブネットに関する Amazon EKS 固有の要件と考慮事項の概要について説明します。Amazon EKS で使用する VPC が

ない場合は、「[the section called “「VPC を作成する」”](#)」を参照してください。AWS Outposts でローカルクラスターまたは拡張クラスターを作成する場合は、このトピックの代わりに「[the section called “VPC とサブネットを作成する”](#)」を参照してください。このトピックのコンテンツは、ハイブリッドノードを持つ Amazon EKS クラスターに適用されます。ハイブリッドノードのその他のネットワーク要件については、「[the section called “ネットワークを準備する”](#)」を参照してください。

## VPC の要件と考慮事項

クラスターを作成する際には、指定する VPC が次の要件と考慮事項を満たす必要があります。

- VPC には、作成するクラスター、ノード、およびその他の Kubernetes リソースで利用できる十分な数の IP アドレスが必要です。使用する VPC に十分な数の IP アドレスがない場合は、使用可能な IP アドレスの数を増やしてみてください。

これを行うには、クラスター設定を更新して、クラスターが使用するサブネットとセキュリティグループを変更します。AWS Management Console、AWS CLI の最新バージョン、AWS CloudFormation、および eksctl のバージョン v0.164.0-rc.0 以降から更新できます。クラスターバージョンを正常にアップグレードするにはこれを実行して、サブネットに利用可能な IP アドレスを増やす必要がある場合があります。

### Important

追加するサブネットはすべて、クラスターの作成時に最初に提供したのと同じ一連の AZ 内にある必要があります。新しいサブネットは、その他のすべての要件 (例えば、十分な IP アドレスを持っている必要がある) を満たす必要があります。

例えば、1 つのクラスターを作成し、4 つのサブネットを指定したとします。指定した順序では、1 番目のサブネットは us-west-2a アベイラビリティーゾーンにあり、2 番目と 3 番目のサブネットは us-west-2b アベイラビリティーゾーンにあり、4 番目のサブネットは us-west-2c アベイラビリティーゾーンにあります。サブネットを変更する場合は、3 つのアベイラビリティーゾーンのそれぞれに少なくとも 1 つのサブネットを指定する必要があります。また、サブネットは元のサブネットと同じ VPC 内にある必要があります。

VPC 内の CIDR ブロックよりも多くの IP アドレスが必要な場合は、VPC に[追加の Classless Inter-Domain Routing \(CIDR\) ブロックを関連付ける](#)ことで CIDR ブロックを追加できます。クラスターの作成前または作成後に、プライベート (RFC 1918) とパブリック (非 RFC 1918) の CIDR ブロックを VPC に関連付けることができます。



新しい CIDR ブロックは、追加するとすぐに使用できるようになります。ただし、コントロールプレーンでは調整が完了した後にのみ新しい CIDR ブロックが認識されるため、VPC に関連付けられた CIDR ブロックがクラスターで認識されるまでに最大で 5 時間かかることがあります。認識後、`kubectl exec` および `kubectl logs` コマンドを実行できます。また、ウェブフックバックエンドとして機能するポッドがある場合は、コントロールプレーンの調整が完了するまで待機する必要があります。

共有サービス VPC でトランジットゲートウェイを使用することで、IP アドレスの利用率を節約できます。詳細については、「[共有サービスによる分離された VPC](#)」および「[Amazon EKS VPC routable IP address conservation patterns in a hybrid network \(ハイブリッドネットワークにおける Amazon EKS VPC ルーティング可能な IP アドレス保全パターン\)](#)」を参照してください。

- Kubernetes で IPv6 アドレスを Pod およびサービスに割り当てる場合は、IPv6 CIDR ブロックを VPC に関連付けます。詳細については、「Amazon VPC ユーザーガイド」の「[IPv6 CIDR ブロックと VPC の関連付け](#)」を参照してください。ハイブリッドノードで実行されているポッドやサービスで IPv6 アドレスを使用することはできません。また、IPv6 IP アドレスファミリーで設定されたクラスターでハイブリッドノードを使用することはできません。
- VPC は、DNS ホスト名と DNS 解決がサポートされている必要があります。そうではない場合、ノードはクラスターに登録されません。詳細については、「Amazon VPC ユーザーガイド」の「[DNS attributes for your VPC](#)」(VPC の DNS 属性) を参照してください。
- VPC では、AWS PrivateLink を使用する VPC エンドポイントが必要になる場合があります。詳細については、「[the section called “サブネットの要件と考慮事項”](#)」を参照してください。

Kubernetes 1.14 以前でクラスターを作成した場合、Amazon EKS は次のタグを VPC に追加しています。

| キー                                                    | 値                  |
|-------------------------------------------------------|--------------------|
| <code>kubernetes.io/cluster/ <i>my-cluster</i></code> | <code>owned</code> |

このタグは Amazon EKS でのみ使用されています。サービスに影響を与えずに、このタグを削除できます。このタグは、バージョン 1.15 以降のクラスターでは使用されません。



## サブネットの要件と考慮事項

クラスターを作成すると、Amazon EKS は、指定したサブネットに 2~4 つの [Elastic Network Interface](#) を作成します。これらのネットワークインターフェイスは、クラスターと VPC 間の通信を可能にします。これらのネットワークインターフェイスでは、Kubernetes の機能 (`kubectl exec` や `kubectl logs` など) も有効化されます。Amazon EKS が作成した各ネットワークインターフェイスの説明には、テキスト Amazon EKS *cluster-name* が書き込まれます。

Amazon EKS は、クラスターの作成時に指定した任意のサブネットにネットワークインターフェイスを作成することができます。クラスターの作成後に、Amazon EKS がネットワークインターフェイスを作成するサブネットを変更できます。クラスターの Kubernetes バージョンを更新すると、Amazon EKS は作成した元のネットワークインターフェイスを削除し、新しいネットワークインターフェイスを作成します。これらのネットワークインターフェイスは、元のネットワークインターフェイスと同じサブネット内に作成することも、元のネットワークインターフェイスとは異なるサブネット内に作成することもできます。ネットワークインターフェイスを作成するサブネットを制御する場合は、クラスターを作成するとき、またはクラスターの作成後にサブネットを更新するときに、指定するサブネットの数を 2 つだけに制限します。

### クラスターのサブネットの要件

クラスターの作成または更新時に指定する [サブネット](#) は、次の要件を満たす必要があります。

- サブネットには、Amazon EKS での使用のために、それぞれ 6 個以上の IP アドレスが必要です。ただし、IP アドレスは 16 個以上を推奨します。
- サブネットは、少なくとも 2 つの異なるアベイラビリティゾーンに存在している必要があります。
- サブネットは AWS Outposts または AWS Wavelength 内に存在できません。ただし、VPC 内にサブネットが存在する場合は、セルフマネージド型ノードおよび Kubernetes のリソースをこれらのタイプのサブネットにデプロイできます。セルフマネージド型ノードの詳細については、「[the section called “セルフマネージド型ノード”](#)」を参照してください。
- サブネットは、パブリックでもプライベートでもかまいません。ただし、可能であれば、プライベートサブネットを指定することをお勧めします。パブリックサブネットは、[インターネットゲートウェイ](#)へのルートが含まれているルートテーブルを含むサブネットです。一方、プライベートサブネットは、インターネットゲートウェイへのルートが含まれていないルートテーブルを含むサブネットです。
- サブネットは以下のアベイラビリティゾーンには配置できません。

| AWS リージョン    | リージョン名          | 拒否されたアベイラビリティゾーン ID |
|--------------|-----------------|---------------------|
| us-east-1    | 米国東部 (バージニア北部)  | use1-az3            |
| us-west-1    | 米国西部 (北カリフォルニア) | usw1-az2            |
| ca-central-1 | カナダ (中部)        | cac1-az3            |

## 各コンポーネントの IP アドレスファミリー使用状況

以下の表に、Amazon EKS の各コンポーネントで使用される IP アドレスファミリーを示します。ネットワークアドレス変換 (NAT) またはその他の互換システムを使用すると、表の値が「No」のファミリーに属する送信元 IP アドレスから、これらのコンポーネントに接続できます。

機能は、クラスターの IP ファミリー (ipFamily) 設定によって異なる場合があります。この設定は、Kubernetes がサービスに割り当てる CIDR ブロックに使用される IP アドレスのタイプを変更します。設定値が IPv4 のクラスターは IPv4 クラスターと呼ばれ、設定値が IPv6 のクラスターは IPv6 クラスターと呼ばれます。

| コンポーネント                                      | IPv4 アドレス          | IPv6 アドレス          | デュアルスタックアドレス       |
|----------------------------------------------|--------------------|--------------------|--------------------|
| EKS API パブリックエンドポイント                         | あり <sup>1, 3</sup> | あり <sup>1, 3</sup> | あり <sup>1, 3</sup> |
| EKS API VPC エンドポイント                          | はい                 | いいえ                | いいえ                |
| EKS Auth API パブリックエンドポイント (EKS Pod Identity) | あり <sup>1</sup>    | あり <sup>1</sup>    | あり <sup>1</sup>    |
| EKS Auth API VPC エンドポイント (EKS Pod Identity)  | あり <sup>1</sup>    | あり <sup>1</sup>    | あり <sup>1</sup>    |

| コンポーネント                                         | IPv4 アドレス         | IPv6 アドレス         | デュアルスタックアドレス      |
|-------------------------------------------------|-------------------|-------------------|-------------------|
| IPv4 Kubernetes クラスターパブリックエンドポイント <sup>2</sup>  | はい                | いいえ               | いいえ               |
| IPv4 Kubernetes クラスタープライベートエンドポイント <sup>2</sup> | はい                | いいえ               | いいえ               |
| IPv6 Kubernetes クラスターパブリックエンドポイント <sup>2</sup>  | はい <sup>1、4</sup> | はい <sup>1、4</sup> | はい <sup>4</sup>   |
| IPv6 Kubernetes クラスタープライベートエンドポイント <sup>2</sup> | はい <sup>1、4</sup> | はい <sup>1、4</sup> | はい <sup>4</sup>   |
| Kubernetes クラスターサブネット                           | あり <sup>2</sup>   | いいえ               | あり <sup>2</sup>   |
| ノードのプライマリ IP アドレス                               | あり <sup>2</sup>   | いいえ               | あり <sup>2</sup>   |
| サービス IP アドレスのクラスター CIDR 範囲                      | あり <sup>2</sup>   | あり <sup>2</sup>   | いいえ               |
| VPC CNI による Pod IP アドレス                         | あり <sup>2</sup>   | あり <sup>2</sup>   | いいえ               |
| IRSA OIDC 発行者 URL                               | あり <sup>1、3</sup> | あり <sup>1、3</sup> | あり <sup>1、3</sup> |

**Note**

<sup>1</sup> エンドポイントは、IPv4 アドレスと IPv6 アドレスの両方を持つデュアルスタックです。AWS 外部のアプリケーション、クラスターのノード、およびクラスター内のポッドは、IPv4 または IPv6 のいずれかによってこのエンドポイントに到達できます。

<sup>2</sup> クラスターの作成時に、クラスターの IP ファミリー (ipFamily) 設定で IPv4 クラスターと IPv6 クラスターのどちらかを選択します。これは後から変更できません。代わりに、他のクラスターを作成してワークロードを移行する際は、別の設定を選択する必要があります。

<sup>3</sup> デュアルスタックエンドポイントは、2024 年 8 月に導入されました。AWS CLI でデュアルスタックエンドポイントを使用するには、「AWS SDK およびツールリファレンスガイド」の「[Dual-stack and FIPS endpoints](#)」の設定を参照してください。新しいエンドポイントを次に示します:

EKS API パブリックエンドポイント

```
eks.region.api.aws
```

IRSA OIDC 発行者 URL

```
oidc-eks.region.api.aws
```

<sup>4</sup> デュアルスタッククラスターエンドポイントは、2024 年 10 月に導入されました。EKS は、この日より後に作成され、クラスターの IP ファミリー (ipFamily) 設定で IPv6 を選択した新しいクラスターに次のエンドポイントを作成します。

EKS クラスターパブリック / プライベートエンドポイント

```
eks-cluster.region.api.aws
```

## ノードのサブネットの要件

クラスターの作成時に指定するのと同じサブネットに、ノードと Kubernetes リソースをデプロイできます。ただし、これは必須ではありません。これは、クラスターの作成時に指定しなかったサブネットにもノードと Kubernetes リソースをデプロイできるためです。ノードを異なるサブネットにデプロイする場合、Amazon EKS は、それらのサブネットにクラスターネットワークインターフェイスを作成しません。ノードと Kubernetes リソースをデプロイするサブネットは、次の要件を満たす必要があります。

- サブネットには、すべてのノードと Kubernetes リソースをデプロイするのに十分な数の使用可能な IP アドレスが必要です。
- Kubernetes で IPv6 アドレスを Pod およびサービスに割り当てる場合は、サブネットに関連付けられている IPv6 CIDR ブロックと IPv4 CIDR ブロックをそれぞれ 1 つずつ指定する必要があります。詳細については、「Amazon VPC ユーザーガイド」の「[IPv6 CIDR ブロックをサブネットに関連付ける](#)」を参照してください。サブネットに関連付けられているルートテーブルには、IPv4 および IPv6 のアドレスへのルートを含める必要があります。詳細については、「Amazon VPC ユーザーガイド」の「[Routes](#)」(ルート)を参照してください。ポッドに割り当てられるのは IPv6 アドレスのみです。ただし、Amazon EKS がクラスターとノード用に作成するネットワークインターフェイスには、IPv4 および IPv6 のアドレスが割り当てられます。
- インターネットから Pod へのインバウンドアクセスが必要な場合は、ロードバランサーのデプロイおよびインバウンドトラフィック (イングレス) に十分な数の IP アドレスを利用できるパブリックサブネットが少なくとも 1 つあることを必ず確認してください。パブリックサブネットにロードバランサーをデプロイできます。ロードバランサーは、プライベートサブネットまたはパブリックサブネット内の Pod への負荷分散を行います。可能であれば、プライベートサブネットにノードをデプロイすることをお勧めします。
- ノードをパブリックサブネットにデプロイする場合は、サブネットが IPv4 パブリックアドレスまたは IPv6 アドレスを自動割り当てする必要があります。ノードを IPv6 CIDR ブロックが関連付けられているプライベートサブネットにデプロイする場合、プライベートサブネットも IPv6 アドレスを自動割り当てする必要があります。2020 年 3 月 26 日以降に Amazon EKS が提供する AWS CloudFormation テンプレートを使用して VPC をデプロイした場合、この設定は有効になっています。テンプレートを使用してこの日付より前に VPC をデプロイした場合、または独自の VPC を使用した場合は、この設定を手動で有効にする必要があります。テンプレートについては、「[the section called “「VPC を作成する」”](#)」を参照してください。詳細については、「[Amazon VPC ユーザーガイド](#)」の「[サブネットのパブリック IPv4 アドレス指定属性を変更する](#)」および「[サブネットの IPv6 アドレス指定属性を変更する](#)」を参照してください。
- ノードをデプロイするサブネットがプライベートサブネットであり、そのルートテーブルにネットワークアドレス変換 (NAT) デバイス (IPv4) または [Egress-Only ゲートウェイ](#) (IPv6) へのルートが含まれていない場合は、AWS PrivateLink を使用して VPC エンドポイントを VPC に追加します。VPC エンドポイントは、ノードと Pod が通信を行う必要があるすべての AWS サービスで必要になります。例としては、Amazon ECR、Elastic Load Balancing、Amazon CloudWatch、AWS Security Token Service、Amazon Simple Storage Service (Amazon S3) などがあります。エンドポイントには、ノードが存在するサブネットを含める必要があります。すべての AWS で VPC エンドポイントがサポートされているわけではありません。詳細については、「[AWS PrivateLink とは](#)」および「[AWS PrivateLink と統合する AWS サービス](#)」を参照してください。Amazon EKS の

その他の要件のリストについては、「[the section called “プライベートクラスター”](#)」を参照してください。

- ロードバランサーをサブネットにデプロイする場合は、サブネットに次のタグが必要です。
- プライベートサブネット

| キー                              | 値 |
|---------------------------------|---|
| kubernetes.io/role/internal-elb | 1 |

- パブリックサブネット

| キー                     | 値 |
|------------------------|---|
| kubernetes.io/role/elb | 1 |

バージョン 1.18 以前の Kubernetes クラスターが作成された場合、Amazon EKS は指定したすべてのサブネットに次のタグを追加します。

| キー                                       | 値      |
|------------------------------------------|--------|
| kubernetes.io/cluster/ <i>my-cluster</i> | shared |

ここで新しい Kubernetes クラスターを作成した場合、Amazon EKS はサブネットにタグを追加しません。1.19 より前のバージョンのクラスターが使用していたサブネットにタグがあった場合、クラスターが新しいバージョンに更新されても、タグはサブネットから自動的に削除されませんでした。AWS Load Balancer Controller のバージョン 2.1.1 以前では、このタグが必要です。新しいバージョンの Load Balancer Controller を使用している場合は、サービスを中断することなくタグを削除できます。コントローラーの詳細については、「[the section called “AWS ロードバランサーコントローラー”](#)」を参照してください。

eksctl または Amazon EKS AWS CloudFormation VPC テンプレートのいずれかを使用して VPC をデプロイした場合、次が適用されます。

- 2020 年 3 月 26 日以降 - パブリック IPv4 アドレスは、パブリックサブネットにより、パブリックサブネットにデプロイした新しいノードに自動的に割り当てられます。

- 2020 年 3 月 26 日より前 - パブリック IPv4 アドレスは、パブリックサブネットにより、パブリックサブネットにデプロイした新しいノードに自動的に割り当てられません。

この変更は、パブリックサブネットにデプロイされた新しいノードグループに、次のような影響を与えます。

- [マネージドノードグループ](#) - 2020 年 4 月 22 日以降にノードグループをパブリックサブネットにデプロイした場合は、パブリックサブネットのパブリック IP アドレスの自動割り当てを有効にする必要があります。詳細については「[サブネットの IPv4 アドレス指定属性の変更](#)」を参照してください。
- [Linux](#)、[Windows](#)、または [Arm](#) のセルフマネージドノードグループ - 2020 年 3 月 26 日以降にノードグループをパブリックサブネットにデプロイした場合は、パブリックサブネットのパブリック IP アドレスの自動割り当てを有効にする必要があります。それ以外の場合は、代わりにパブリック IP アドレスを使用してノードを起動する必要があります。詳細については、「[サブネットのパブリック IPv4 アドレス指定属性の変更](#)」または「[インスタンス起動時のパブリック IPv4 アドレスの割り当て](#)」を参照してください。

## 共有サブネットの要件と考慮事項

VPC 共有を使用して、同じ AWS 組織内でサブネットを他の AWS アカウントと共有できます。共有サブネットに Amazon EKS クラスターを作成できますが、以下の考慮事項に注意してください。

- VPC サブネットの所有者は、参加者アカウントで Amazon EKS クラスターを作成する前に、そのアカウントとサブネットを共有する必要があります。
- VPC のデフォルトセキュリティグループは所有者に属しているため、デフォルトのセキュリティグループを使用してリソースを起動することはできません。さらに、参加者は、他の参加者または所有者が所有するセキュリティグループを使用してリソースを起動することはできません。
- 共有サブネットでは、参加者と所有者がそれぞれのアカウント内のセキュリティグループを個別に管理します。サブネットの所有者は、参加者が作成したセキュリティグループを表示できますが、これらのグループに対してアクションを実行することはできません。サブネットの所有者がこれらのセキュリティグループの削除または変更を希望する場合は、セキュリティグループを作成した参加者がそのアクションを実行する必要があります。
- 参加者がクラスターを作成する場合、以下の考慮事項が適用されます。
  - クラスター IAM ロールとノード IAM ロールは、そのアカウントで作成する必要があります。詳細については、「[the section called “クラスター IAM ロール”](#)」および「[the section called “ノードの IAM ロール”](#)」を参照してください。



- 管理対象ノードグループを含め、すべてのノードは同じ参加者が作成する必要があります。
- 共有 VPC の所有者は、参加者が共有サブネット内に作成したクラスターを表示、更新、削除することはできません。これは、アカウントごとに異なるアクセス権を持つ VPC リソースに加えて適用されます。詳細については、「Amazon VPC ユーザーガイド」の「[所有者および参加者の責任と権限](#)」を参照してください。
- Amazon VPC CNI plugin for Kubernetes のカスタムネットワーク機能を使用する場合、所有者アカウントに記載されているアベイラビリティゾーン ID マッピングを使用して、それぞれの ENIConfig を作成する必要があります。詳細については、「[the section called “カスタムネットワークキング”](#)」を参照してください。

VPC サブネット共有の詳細については、「Amazon VPC ユーザーガイド」の「[VPC を他のアカウントと共有する](#)」を参照してください。

## Amazon EKS クラスターの Amazon VPC を作成する

Amazon Virtual Private Cloud (Amazon VPC) を使用すると、定義した仮想ネットワーク内で AWS リソースを起動できます。この仮想ネットワークは、お客様自身のデータセンターで運用されている従来のネットワークによく似ています。ただし、Amazon Web Services のスケーラブルなインフラストラクチャを使用できるというメリットがあります。本番用に使う Amazon EKS クラスターをデプロイする前に、Amazon VPC サービスを十分に理解しておくことをお勧めします。詳細については、[Amazon VPC ユーザーガイド](#)を参照してください。

Amazon EKS クラスター、ノード、および Kubernetes リソースが VPC にデプロイされます。Amazon EKS で既存の VPC を使用する場合は、その VPC が「[the section called “VPC およびサブネットの要件”](#)」に記載されている要件を満たしている必要があります。このトピックでは、Amazon EKS に用意されている AWS CloudFormation テンプレートを使用して Amazon EKS 要件を満たす VPC を作成する方法について説明します。テンプレートをデプロイすると、テンプレートによって作成されたリソースを表示して、作成されたリソースとそれらのリソースの設定を正確に把握できます。ハイブリッドノードを使用している場合、VPC のルートテーブルにオンプレミスネットワークへのルートが含まれている必要があります。ハイブリッドノードのネットワーク要件の詳細については、「[the section called “ネットワークを準備する”](#)」を参照してください。

### 前提条件

Amazon EKS 用の VPC を作成するには、Amazon VPC リソースを作成するのに必要な IAM アクセス許可が必要です。これらのリソースは、VPC、サブネット、セキュリティグループ、ルートテー



ブルとルート、およびインターネットと NAT ゲートウェイです。詳細については、「Amazon VPC ユーザーガイド」の「[パブリックサブネットを持つ VPC を作成するポリシー例](#)」を参照してください。完全なリストについては、「[サービス認可リファレンス](#)」の「[アクション](#)」を参照してください。

パブリックサブネットとプライベートサブネット、パブリックサブネットのみ、またはプライベートサブネットのみで、VPC を作成できます。

## パブリックサブネットおよびプライベートサブネット

この VPC には、2 つのパブリックサブネットと 2 つのプライベート サブネットがあります。パブリックサブネットの関連付けられているルートテーブルには、インターネットゲートウェイへのルートが含まれています。一方、プライベートサブネットのルートテーブルには、インターネットゲートウェイへのルートがありません。1 つのパブリックサブネットと 1 つのプライベートサブネットが同じ[アベイラビリティーゾーン](#)にデプロイされます。他のパブリックサブネットとプライベートサブネットは、同じ AWS リージョン内の 2 番目のアベイラビリティーゾーンにデプロイされます。ほとんどのデプロイにこのオプションをお勧めします。

このオプションを使用すると、プライベートサブネットにノードをデプロイできます。このオプションでは、Kubernetes がパブリックサブネットにロードバランサーをデプロイできるようにします。これにより、プライベートサブネットのノードで実行されている Pod へのトラフィックの負荷を分散できます。パブリック IPv4 アドレスは、パブリックサブネットにデプロイされたノードに自動的に割り当てられます。一方、プライベートサブネットにデプロイされたノードに対しては、パブリック IPv4 アドレスは割り当てられません。

また、パブリックサブネットおよびプライベートサブネットに置かれたノードに対し、IPv6 アドレスを割り当てることもできます。プライベートサブネット内のノードは、クラスターや他の AWS サービスとの通信が可能です。Pod は、各アベイラビリティーゾーンにデプロイされた NAT ゲートウェイ (IPv4 アドレスを使用) または送信専用インターネットゲートウェイ (IPv6 アドレスを使用) を介して、インターネットへの通信が可能です。クラスターまたはノード以外のソースからのすべてのインバウンドトラフィックを拒否するが、すべてのアウトバウンドトラフィックを許可するルールを持つセキュリティグループがデプロイされます。サブネットには、Kubernetes がロードバランサーをデプロイできるようにタグが付けられています。

- a. [AWS CloudFormation コンソール](#)を開きます。
- b. ナビゲーションバーで、Amazon EKS をサポートする AWS リージョンを選択します。
- c. [スタックの作成] を選択し、[新しいリソースの使用 (標準)] を選択します。

- d. [Prerequisite - Prepare template] (前提条件 – テンプレートの準備) の下で、[Template is ready] (テンプレートの準備完了) がオンになっていることを確認した上で、[Specify template] (テンプレートの指定) の下で、[Amazon S3 URL] を選択します。
- e. IPv4 のみをサポートする VPC、または IPv4 と IPv6 をサポートする VPC を作成できます。下記の URL の 1 つを [Amazon S3 URL] の下にあるテキスト領域に貼り付けて、[Next] (次へ) をクリックします。
- IPv4

```
https://s3.us-west-2.amazonaws.com/amazon-eks/cloudformation/2020-10-29/amazon-eks-vpc-private-subnets.yaml
```

- IPv4 および IPv6

```
https://s3.us-west-2.amazonaws.com/amazon-eks/cloudformation/2020-10-29/amazon-eks-ipv6-vpc-public-private-subnets.yaml
```

- a. [Specify stack details] (スタック詳細の指定) ページで、パラメータを入力し、[Next] (次へ) を選択します。
- スタック名: AWS CloudFormation スタックのスタック名を選択します。例えば、前のステップで使用されたテンプレート名を使用できます。この名前には英数字 (大文字と小文字が区別されます) とハイフンのみを使用できます。先頭の文字は英数字である必要があります。また、100 文字より長くすることはできません。名前は、クラスターを作成する AWS リージョンおよび AWS アカウント内で一意である必要があります。
  - [VpcBlock]: VPC の IPv4 CIDR 範囲を選択します。デプロイする各ノード、ポッド、およびロードバランサーには、このブロックの IPv4 アドレスが割り当てられます。デフォルトの IPv4 値で、ほとんどの実装において十分な IP アドレスを取得できますが、不足する場合はこの値を変更できます。詳細については、「Amazon VPC ユーザーガイド」の「[VPC とサブネットのサイズ設定](#)」を参照してください。VPC を作成したら、追加の CIDR ブロックを VPC に追加することもできます。IPv6 VPC を作成する場合、IPv6 CIDR 範囲は Amazon のグローバルユニキャストアドレススペースから自動的に割り当てられます。
  - [PublicSubnet01Block]: パブリックサブネット 1 の IPv4 CIDR ブロックを指定します。デフォルト値は、ほとんどの実装で十分な IP アドレスを提供しますが、そうでない場合は、変更することができます。IPv6 VPC を作成する場合、テンプレートで指定済みブロックを使用できます。

- [PublicSubnet02Block]: パブリックサブネット 2 の IPv4 CIDR ブロックを指定します。デフォルト値は、ほとんどの実装で十分な IP アドレスを提供しますが、そうでない場合は、変更することができます。IPv6 VPC を作成する場合、テンプレートで指定済みブロックを使用できます。
  - [PrivateSubnet01Block]: プライベートサブネット 1 の IPv4 CIDR ブロックを指定します。デフォルト値は、ほとんどの実装で十分な IP アドレスを提供しますが、そうでない場合は、変更することができます。IPv6 VPC を作成する場合、テンプレートで指定済みブロックを使用できます。
  - [PrivateSubnet02Block]: プライベートサブネット 2 の IPv4 CIDR ブロックを指定します。デフォルト値は、ほとんどの実装で十分な IP アドレスを提供しますが、そうでない場合は、変更することができます。IPv6 VPC を作成する場合、テンプレートで指定済みブロックを使用できます。
- b. (オプション) [Configure stack options] (スタックオプションの設定) ページで、スタックリソースにタグ付けを行った上で、[Next] (次へ) をクリックします。
- c. [確認] ページで [スタックの作成] を選択します。
- d. スタックが作成されたら、コンソールで選択し、[Outputs] (出力) を選択します。
- e. 作成された VPC の [VpcId] を記録します。これは、クラスターとノードを作成するときに必要です。
- f. 作成されたサブネットの SubnetIds と、それらをパブリックサブネットとプライベートサブネットのどちらとして作成したかを記録します。クラスターとノードを作成するときは、これらのうち少なくとも 2 つが必要です。
- g. IPv4 VPC を作成している場合、この手順をスキップしてください。IPv6 VPC を作成している場合は、テンプレートによって作成されたパブリックサブネットにおいて、IPv6 アドレスの自動割り当てオプションが有効化されている必要があります。この設定は、プライベートサブネットに対してすでに有効になっています。次の手順を完了して、設定を有効にします。
- i. Amazon VPC コンソール (<https://console.aws.amazon.com/vpc/>) を開きます。
  - ii. 左のナビゲーションペインで [Subnets] (サブネット) を選択します。
  - iii. パブリックサブネット ( **stack-name**/SubnetPublic01 または **stack-name**/SubnetPublic02 には、public という語が含まれています) の 1 つを選択し、[Actions] (アクション)、[Edit subnet settings] (サブネット設定の編集) の順に選択します。
  - iv. [Enable auto-assign IPv6 address] (IPv6 アドレスの自動割り当てを有効にする) チェックボックスをオンにし、[Save] (保存) をクリックします。
  - v. 他のパブリックサブネットに対しても、ここまでの手順を再度実行します。

## パブリックサブネットのみ

この VPC には、AWS リージョン内の異なるアベイラビリティーゾーンにデプロイされる 3 つのパブリックサブネットがあります。すべてのノードには自動的にパブリック IPv4 アドレスが割り当てられ、[インターネットゲートウェイ](#)を介してインターネットトラフィックを送受信できます。すべてのインバウンドトラフィックを拒否し、すべてのアウトバウンドトラフィックを許可する[セキュリティグループ](#)がデプロイされます。サブネットには、Kubernetes がロードバランサーをデプロイできるようにタグが付けられています。

- a. [AWS CloudFormation コンソール](#)を開きます。
- b. ナビゲーションバーで、Amazon EKS をサポートする AWS リージョンを選択します。
- c. [Create stack] (スタックの作成) を選択し、[With new resources (standard)] (新しいリソースの使用 (標準)) を選択します。
- d. [テンプレートの準備] の下で、[テンプレートの準備完了] が選択されていることを確認し、[テンプレートソース] の下で、[Amazon S3 URL] を選択します。
- e. 次の URL を [Amazon S3 URL] の下のテキスト領域に貼り付けて、[次へ] を選択します。

```
https://s3.us-west-2.amazonaws.com/amazon-eks/cloudformation/2020-10-29/amazon-eks-vpc-sample.yaml
```

- a. [Specify Details] (詳細の指定) ページで、パラメータを入力し、[Next] (次へ) を選択します。
  - スタック名: AWS CloudFormation スタックのスタック名を選択します。例えば、**amazon-eks-vpc-sample** という名前にすることができます。この名前には英数字 (大文字と小文字が区別されます) とハイフンのみを使用できます。先頭の文字は英数字である必要があります。また、100 文字より長くすることはできません。名前は、クラスターを作成する AWS リージョンおよび AWS アカウント内で一意である必要があります。
  - [VpcBlock]: VPC の CIDR ブロックを選択します。デプロイする各ノード、ポッド、およびロードバランサーには、このブロックの IPv4 アドレスが割り当てられます。デフォルトの IPv4 値で、ほとんどの実装において十分な IP アドレスを取得できますが、不足する場合はこの値を変更できます。詳細については、「Amazon VPC ユーザーガイド」の「[VPC とサブネットのサイズ設定](#)」を参照してください。VPC を作成したら、追加の CIDR ブロックを VPC に追加することもできます。
  - Subnet01Block: サブネット 1 の CIDR ブロックを指定します。デフォルト値は、ほとんどの実装で十分な IP アドレスを提供しますが、そうでない場合は、変更することができます。

- Subnet02Block: サブネット 2 の CIDR ブロックを指定します。デフォルト値は、ほとんどの実装で十分な IP アドレスを提供しますが、そうでない場合は、変更することができます。
  - Subnet03Block: サブネット 3 の CIDR ブロックを指定します。デフォルト値は、ほとんどの実装で十分な IP アドレスを提供しますが、そうでない場合は、変更することができます。
- b. (オプション) [オプション] ページで、スタックリソースをタグ付けします。[Next] を選択します。
- c. [Review] ページで、[Create] を選択します。
- d. スタックが作成されたら、コンソールで選択し、[Outputs] (出力) を選択します。
- e. 作成された VPC の [VpcId] を記録します。これは、クラスターとノードを作成するときに必要です。
- f. 作成されたサブネットの [SubnetIds] を記録します。クラスターとノードを作成するときは、これらのうち少なくとも 2 つが必要です。
- g. (オプション) この VPC にデプロイするクラスターは、Pod とサービスにプライベート IPv4 アドレスを割り当てることができます。この VPC にクラスターをデプロイし、Pod とサービスにプライベート IPv6 アドレスを割り当てる場合は、VPC、サブネット、ルートテーブル、およびセキュリティグループを更新します。詳細については、「Amazon VPC ユーザーガイド」の「[既存の VPC を IPv4 から IPv6 に移行する](#)」を参照してください。Amazon EKS では、サブネットにおいて、IPv6 アドレスの Auto-assign オプションが有効化されている必要があります。デフォルトでは、このオプションは無効となっています。

## プライベートサブネットのみ

この VPC には、AWS リージョン内の異なるアベイラビリティーゾーンにデプロイされる 3 つのプライベートサブネットがあります。サブネットにデプロイされたリソースはインターネットにアクセスできず、インターネットからサブネット内のリソースにアクセスすることもできません。ノードが通常アクセスする必要があるいくつかの AWS サービスの AWS PrivateLink を使用して、[VPC エンドポイント](#)がテンプレートによって作成されます。ノードにアウトバウンドインターネットアクセスが必要な場合は、VPC が作成された後で、各サブネットのアベイラビリティーゾーン内に、パブリック [NAT ゲートウェイ](#)を追加できます。[セキュリティグループ](#)は、サブネットにデプロイされたリソースからのトラフィックを除き、すべてのインバウンドトラフィックを拒否するように作成されます。また、セキュリティグループは、すべてのアウトバウンドトラフィックを許可します。サブネットには、Kubernetes が内部ロードバランサーをデプロイできるようにタグが付けられています。この設定で VPC を作成する場合は、「[the section called “プライベートクラスター”](#)」を参照してその他の要件と考慮事項を確認してください。

- a. [AWS CloudFormation コンソール](#)を開きます。



- b. ナビゲーションバーで、Amazon EKS をサポートする AWS リージョンを選択します。
- c. [Create stack] (スタックの作成) を選択し、[With new resources (standard)] (新しいリソースの使用 (標準)) を選択します。
- d. [テンプレートの準備] の下で、[テンプレートの準備完了] が選択されていることを確認し、[テンプレートソース] の下で、[Amazon S3 URL] を選択します。
- e. 次の URL を [Amazon S3 URL] の下のテキスト領域に貼り付けて、[次へ] を選択します。

```
https://s3.us-west-2.amazonaws.com/amazon-eks/cloudformation/2020-10-29/amazon-eks-fully-private-vpc.yaml
```

- a. [詳細の指定] ページでパラメータを入力し、[次へ] を選択します。
  - スタック名: AWS CloudFormation スタックのスタック名を選択します。例えば、*amazon-eks-fully-private-vpc* という名前にすることができます。この名前には英数字 (大文字と小文字が区別されます) とハイフンのみを使用できます。先頭の文字は英数字である必要があります。また、100 文字より長くすることはできません。名前は、クラスターを作成する AWS リージョンおよび AWS アカウント内で一意である必要があります。
  - [VpcBlock]: VPC の CIDR ブロックを選択します。デプロイする各ノード、ポッド、およびロードバランサーには、このブロックの IPv4 アドレスが割り当てられます。デフォルトの IPv4 値で、ほとんどの実装において十分な IP アドレスを取得できますが、不足する場合はこの値を変更できます。詳細については、「Amazon VPC ユーザーガイド」の「[VPC とサブネットのサイズ設定](#)」を参照してください。VPC を作成したら、追加の CIDR ブロックを VPC に追加することもできます。
  - PrivateSubnet01Block: サブネット 1 の CIDR ブロックを指定します。デフォルト値は、ほとんどの実装で十分な IP アドレスを提供しますが、そうでない場合は、変更することができます。
  - PrivateSubnet02Block: サブネット 2 の CIDR ブロックを指定します。デフォルト値は、ほとんどの実装で十分な IP アドレスを提供しますが、そうでない場合は、変更することができます。
  - PrivateSubnet03Block: サブネット 3 の CIDR ブロックを指定します。デフォルト値は、ほとんどの実装で十分な IP アドレスを提供しますが、そうでない場合は、変更することができます。
- b. (オプション) [オプション] ページで、スタックリソースをタグ付けします。[Next] を選択します。
- c. [Review] ページで、[Create] を選択します。
- d. スタックが作成されたら、コンソールで選択し、[Outputs] (出力) を選択します。
- e. 作成された VPC の [VpcId] を記録します。これは、クラスターとノードを作成するときに必要です。

- f. 作成されたサブネットの [SubnetIds] を記録します。クラスターとノードを作成するときは、これらのうち少なくとも 2 つが必要です。
- g. (オプション) この VPC にデプロイするクラスターは、Pod とサービスにプライベート IPv4 アドレスを割り当てることができます。この VPC にクラスターをデプロイし、Pod とサービスにプライベート IPv6 アドレスを割り当てる場合は、VPC、サブネット、ルートテーブル、およびセキュリティグループを更新します。詳細については、「Amazon VPC ユーザーガイド」の「[既存の VPC を IPv4 から IPv6 に移行する](#)」を参照してください。Amazon EKS は、サブネットで Auto-assign IPv6 アドレスオプションが有効になっている必要があります (デフォルトでは無効)。

## クラスターの Amazon EKS セキュリティグループ要件を表示する

このトピックでは、Amazon EKS クラスターのセキュリティグループの要件について説明します。

### デフォルトのクラスターセキュリティグループ

クラスターを作成すると、Amazon EKS により `eks-cluster-sg-my-cluster-uniqueID` という名前のセキュリティグループが作成されます。セキュリティグループには、デフォルトで次のルールがあります。

| ルールタイプ  | プロトコル | ポート | 送信元 | デスティネーション                           |
|---------|-------|-----|-----|-------------------------------------|
| インバウンド  | すべて   | すべて | 自分  |                                     |
| アウトバウンド | すべて   | すべて |     | 0.0.0.0/0 (IPv4)<br>または ::/0 (IPv6) |
| アウトバウンド | すべて   | すべて |     | 自分 (EFA トラフィック用)                    |

デフォルトのセキュリティグループには、同じセキュリティグループの宛先に対して Elastic Fabric Adapter (EFA) トラフィックを許可するアウトバウンドルールが含まれます。これにより、クラスター内の EFA トラフィックが有効になり、AI/ML およびハイパフォーマンスコンピューティング (HPC) ワークロードに役立てることができます。詳細については、「Amazon Elastic Compute

Cloud ユーザーガイド」の「[Amazon EC2 の AI/ML および HPC ワークロード用の Elastic Fabric Adapter](#)」を参照してください。

**⚠ Important**

クラスターにアウトバウンドルールが必要ない場合は、削除できます。削除する場合でも、「[クラスタートラフィックの制限](#)」に記載されている最低限のルールが適用されます。インバウンドルールを削除すると、Amazon EKS はクラスターが更新されるたびにそのルールを再作成します。

Amazon EKS は次のタグをセキュリティグループに追加します。タグを削除した場合、Amazon EKS はクラスターが更新されるたびにこれらのタグをセキュリティグループに追加します。

| キー                                       | 値                                          |
|------------------------------------------|--------------------------------------------|
| kubernetes.io/cluster/ <i>my-cluster</i> | owned                                      |
| aws:eks:cluster-name                     | <i>my-cluster</i>                          |
| Name                                     | eks-cluster-sg- <i>my-cluster-uniqueid</i> |

Amazon EKS は、同様に作成される次のリソースに、セキュリティグループを自動的に関連付けます。

- 2—4 エラスティックネットワークインターフェイス (これ以降、ネットワークインターフェイス) は、クラスターの作成時に作成されます。
- 作成したマネージドノードグループ内のノードのネットワークインターフェイス。

デフォルトのルールでは、すべてのトラフィックがクラスターとノード間で自由に行き来することができます。また、任意の送信先へのすべてのアウトバウンドトラフィックが許可されています。クラスターを作成すると、オプションで独自のセキュリティグループを指定できます。その場合、Amazon EKS は、指定したセキュリティグループをクラスター用に作成するネットワークインターフェイスにも関連付けます。ただし、作成したノードグループには関連付けられません。



クラスターのセキュリティグループの ID は、AWS Management Console のクラスターの [Networking] (ネットワーキング) セクションで確認できます。もしくは、次の AWS CLI コマンドを実行して確認できます。

```
aws eks describe-cluster --name my-cluster --query
cluster.resourcesVpcConfig.clusterSecurityGroupId
```

## クラスタートラフィックの制限

クラスターとノード間で開いているポートの数を制限する必要がある場合は、[デフォルトのアウトバウンドルール](#)を削除し、クラスターに必要な次の最低限のルールを追加できます。[デフォルトのインバウンドルール](#)を削除すると、Amazon EKS はクラスターが更新されるたびにそのルールを再作成します。

| ルールタイプ           | プロトコル     | ポート   | デスティネーション       |
|------------------|-----------|-------|-----------------|
| アウトバウンド          | TCP       | 443   | クラスターセキュリティグループ |
| アウトバウンド          | TCP       | 10250 | クラスターセキュリティグループ |
| アウトバウンド<br>(DNS) | TCP と UDP | 53    | クラスターセキュリティグループ |

次のトラフィックにもルールを追加する必要があります。

- ノード間通信にノードが使用するプロトコルおよびポート。
- アウトバウンドのインターネットアクセス。これによりノードが Amazon EKS API にアクセス可能になり、起動時のノードの登録やクラスターの詳細分析が行えます。ノードがインターネットにアクセスできない場合は、「[インターネットアクセスが制限されたプライベートクラスターをデプロイする](#)」でその他の考慮事項を参照してください。
- Amazon ECR や、イメージをプルする必要があるその他のコンテナレジストリ (DockerHub など) からコンテナイメージをプルするためのノードのアクセス。詳細については、「AWS 全般のリリースノート」の「[AWS IP アドレス範囲](#)」を参照してください。
- Amazon S3 へのノードのアクセス。

- IPv4 や IPv6 アドレスが必要な個別のルール。
- ハイブリッドノードを使用している場合は、クラスターに追加のセキュリティグループを追加して、オンプレミスのノードやポッドとの通信を許可する必要があります。詳細については、「[the section called “ネットワークを準備する”](#)」を参照してください。

ルールの制限を検討している場合は、変更したルールを本番稼働用のクラスターに適用する前に、すべての Pod を徹底的にテストすることをお勧めします。

最初に Kubernetes 1.14 およびプラットフォームバージョン eks.3 またはそれ以前でクラスターを最初にデプロイした場合、次の点を考慮してください。

- コントロールプレーンおよびノードセキュリティグループがある場合もあります。これらのグループの作成時、前の表に示した制限付きルールが含まれていました。これらのセキュリティグループは不要で、削除できます。ただし、それらのグループに含まれるルールがクラスターセキュリティグループに含まれていることを確認する必要があります。
- API を使用してクラスターを直接デプロイした場合、または AWS CLI や AWS CloudFormation などのツールを使用してクラスターを作成しており、クラスター作成時にセキュリティグループを指定しなかった場合、Amazon EKS が作成したクラスターネットワークインターフェイスに VPC のデフォルトのセキュリティグループが適用されます。

## 共有セキュリティグループ

Amazon EKS は、共有セキュリティグループをサポートしています。

- [セキュリティグループの VPC 関連付け] により、セキュリティグループは同じアカウントとリージョン内の複数の VPC に関連付けられます。
  - 「Amazon VPC ユーザーガイド」で、[セキュリティグループを複数の VPC に関連付ける](#)方法をご覧ください。
- [共有セキュリティグループ] を使用すると、セキュリティグループを他の AWS アカウントと共有できます。アカウントは同じ AWS 組織内にある必要があります。
  - 「Amazon VPC ユーザーガイド」で、[セキュリティグループを組織と共有する](#)方法をご覧ください。
- セキュリティグループは常に 1 つの AWS リージョンに制限されます。

## Amazon EKS の考慮事項

- EKS には、標準セキュリティグループと同じ共有セキュリティグループまたはマルチ VPC セキュリティグループの要件があります。

## Amazon EKS クラスターのネットワーキングアドオンを管理する

Amazon EKS クラスターで利用できるネットワーキングアドオンがいくつかあります。

### 組み込みアドオン

#### Note

コンソール以外の方法でクラスターを作成した場合、各クラスターにセルフマネージド版の組み込みアドオンが付属します。セルフマネージド型バージョンは、AWS Management Console、AWS コマンドラインインターフェイス、SDK のいずれからでも管理することはできません。セルフマネージド型アドオンの設定とアップグレードは、ユーザーが管理します。

セルフマネージド型のアドオンを使用する代わりに、アマゾン EKS タイプのアドオンをクラスターに追加することをお勧めします。コンソールを使用してクラスターを作成する場合、これらのアドオンの Amazon EKS タイプがインストールされます。

### Amazon VPC CNI plugin for Kubernetes

この CNI アドオンは、Elastic Network Interface を作成し、Amazon EC2 ノードにアタッチします。このアドオンは、VPC からプライベート IPv4 または IPv6 アドレスを各 Pod およびサービスにも割り当てます。このアドオンはデフォルトでクラスターにインストールされます。詳細については、「[the section called “Amazon VPC CNI”](#)」を参照してください。ハイブリッドノードを使用している場合、VPC CNI はデフォルトでインストールされますが、アンチアフィニティルールによりハイブリッドノード上での実行が防止されます。ハイブリッドノードの CNI オプションの詳細については、「[the section called “CNI を設定する”](#)」を参照してください。

### CoreDNS

CoreDNS は、Kubernetes クラスター DNS として機能する、フレキシブルで拡張可能な DNS サーバーです。CoreDNS により、クラスター内にあるすべての Pod の名前が解決されます。このアドオンはデフォルトでクラスターにインストールされます。詳細については、「[the section called “CoreDNS”](#)」を参照してください。

## kube-proxy

このアドオンは、Amazon EC2 ノード上のネットワークルールを維持し、Pod へのネットワーク通信を可能にします。このアドオンはデフォルトでクラスターにインストールされます。詳細については、「[the section called “kube-proxy”](#)」を参照してください。

## オプションの AWS ネットワークアドオン

### AWS Load Balancer Controller

loadbalancer タイプの Kubernetes サービスオブジェクトをデプロイするとき、コントローラーは AWS Network Load Balancer を作成します。Kubernetes 入力オブジェクトを作成するとき、コントローラーは AWS Application Load Balancer を作成します。Network Load Balancer のプロビジョニングには、Kubernetes に組み込まれている[従来のクラウドプロバイダー](#)コントローラーではなく、このコントローラーを使用することをお勧めします。詳細については、[AWS Load Balancer Controller](#) ドキュメントを参照してください。

### AWS ゲートウェイ API コントローラー

このコントローラーを使用すると、[Kubernetes ゲートウェイ API](#) を使用して複数の Kubernetes クラスターを通してサービスに接続できます。コントローラーは、[Amazon VPC Lattice](#) サービスを使用して Amazon EC2 インスタンス、コンテナ、サーバーレス機能で実行されている Kubernetes サービスを接続します。詳細については、「[AWS ゲートウェイ API コントローラー](#)」ドキュメントを参照してください。

アドオンの詳細については、「[the section called “Amazon EKS アドオン”](#)」を参照してください。

## Amazon VPC CNI を使用して Pod に IP を割り当てる

### Tip

Amazon EKS 自動モード ではネットワーク形成のアドオンをインストールまたはアップグレードする必要はありません。自動モード にはポッドのネットワーク形成とロードバランシング機能が含まれています。

詳細については、「[EKS Auto Mode](#)」を参照してください。

Amazon VPC CNI plugin for Kubernetes アドオンは、Amazon EKS クラスター内の各 Amazon EC2 ノードにデプロイされます。アドオンは [エラスティックネットワークインターフェイス](#) を作成

し、Amazon EC2 ノードにアタッチします。また、アドオンは VPC のプライベート IPv4 アドレスまたは IPv6 アドレスを各 Pod に割り当てます。

アドオンのバージョンはクラスター内の各 Fargate ノードにデプロイされますが、Fargate ノードでは更新しません。Amazon EKS クラスターではその他の互換性のある CNI プラグインも使用できますが、これは AWS インフラストラクチャで動作するノード用の Amazon EKS がサポートする唯一の CNI プラグインです。互換性のある他の CNI プラグインの詳細については「[the section called “代替 CNI プラグイン”](#)」を参照してください。VPC CNI はハイブリッドノードでの使用はサポートされていません。ハイブリッドノードの CNI オプションの詳細については「[the section called “CNI を設定する”](#)」を参照してください。

次の表は、Kubernetes バージョンごとに Amazon EKS アドオンタイプの最新バージョンを示しています。

### Amazon VPC CNI のバージョン

| Kubernetes バージョン | Amazon EKS タイプの VPC CNI バージョン |
|------------------|-------------------------------|
| 1.33             | v1.19.5-eksbuild.3            |
| 1.32             | v1.19.5-eksbuild.3            |
| 1.31             | v1.19.5-eksbuild.3            |
| 1.30             | v1.19.5-eksbuild.3            |
| 1.29             | v1.19.5-eksbuild.3            |
| 1.28             | v1.19.5-eksbuild.3            |
| 1.27             | v1.19.5-eksbuild.3            |
| 1.26             | v1.19.5-eksbuild.3            |

#### Important

このアドオンを自己管理している場合、表のバージョンは利用可能なセルフマネージドバージョンと同じではない可能性があります。このアドオンのセルフマネージドタイプの更新の詳細については「[the section called “更新 \( 自己管理 \)”](#)」を参照してください。

**⚠ Important**

VPC CNI v1.12.0 以降にアップグレードするにはまず VPC CNI v1.7.0 にアップグレードする必要があります。一度に 1 つのマイナーバージョンを更新することをお勧めします。

## 考慮事項

この機能を使用する際の考慮事項を次に示します。

- バージョンは `major-version.minor-version.patch-version-eksbuild.build-number` として指定されます。
- 各機能のバージョン互換性を確認します。Amazon VPC CNI plugin for Kubernetes の各リリースに含まれる機能によっては、特定の Kubernetes バージョンが必要になることがあります。異なる Amazon EKS の機能を使用する際、アドオンの特定のバージョンが必要な場合については機能に関するドキュメントに記載されています。以前のバージョンを実行する特定の理由がある場合を除いて、最新のバージョンを実行することをお勧めします。

## Amazon VPC CNI Amazon EKS アドオンの追加します。

次のステップを使用して、Amazon VPC CNI Plugin for Kubernetes の Amazon EKS アドオンを作成します。

開始する前に、考慮事項を確認してください。詳細については、「[the section called “考慮事項”](#)」を参照してください。

### 前提条件

以下は、Amazon VPC CNI Plugin for Kubernetes の Amazon EKS アドオンの前提条件です。

- 既存の Amazon EKS クラスター。デプロイするには「[はじめに](#)」を参照してください。
- クラスターの既存の AWS Identity and Access Management (IAM) OpenID Connect (OIDC) プロバイダー。既に存在しているかどうかを確認する、または作成するには「[the section called “IAM OIDC プロバイダー”](#)」を参照してください。
- [AmazonEKS\\_CNI\\_Policy](#) IAM ポリシー (クラスターが IPv4 ファミリーを使用している場合)、または IPv6 ポリシー (クラスターが IPv6 ファミリーを使用している場合) がアタッチされた IAM 役割です。VPC CNI 役割の詳細については、「[the section called “IRSA 用の設定”](#)」を参照してく

ださい。IPv6 ポリシーの詳細については、「[the section called “IPv6 ファミリーを使用するクラスター用に IAM ポリシーを作成します。”](#)」を参照してください。

**⚠ Important**

Amazon VPC CNI Plugin for Kubernetes バージョン v1.16.0 から v1.16.1 では、CNI 仕様バージョン v1.0.0 が実装されています。バージョン v1.0.0 の CNI 仕様の詳細については、GitHub の「[Container Network Interface \(CNI\) Specification](#)」を参照してください。

## 手順

前提条件を満たしたら、次のステップを使用してアドオンを作成します。

1. クラスターにインストールされているアドオンのバージョンを確認します。

```
kubectl describe daemonset aws-node --namespace kube-system | grep amazon-k8s-cni: |
cut -d : -f 3
```

出力例は次のとおりです。

```
v1.16.4-eksbuild.2
```

2. クラスターにインストールされているアドオンのタイプを確認します。クラスターを作成するために使用したツールによっては、現在クラスターに Amazon EKS アドオンタイプがインストールされていない場合があります。##### の部分は、自分のクラスター名に置き換えます。

```
aws eks describe-addon --cluster-name my-cluster --addon-name vpc-cni --query
addon.addonVersion --output text
```

バージョン番号が返された場合、Amazon EKS タイプのアドオンがクラスターにインストールされているため、このステップの残りのステップを完了する必要はありません。エラーが返された場合、クラスターに Amazon EKS タイプのアドオンがインストールされていません。インストールするには、このステップの残りのステップを完了します。

3. 現在インストールされているアドオンの設定を保存します。

```
kubectl get daemonset aws-node -n kube-system -o yaml > aws-k8s-cni-old.yaml
```



4. AWS CLI を使用してアドオンを作成します。AWS Management Console または `eksctl` を使用してアドオンを作成する場合は、「[the section called “アドオンを作成する”](#)」を参照して、アドオン名の `vpc-cni` を指定します。デバイスに沿ったコマンドをコピーします。必要に応じてコマンドに次の変更を加え、変更したコマンドを実行してください。

- `#####` の部分は、自分のクラスター名に置き換えます。
- `v1.19.2-eksbuild.1` を、使用しているクラスターバージョンに対して最新バージョンの表に記載されている最新バージョンに置き換えます。最新バージョンの表については、「[the section called “Amazon VPC CNI のバージョン”](#)」を参照してください。
- `111122223333` を、アカウント ID に置き換えて、`AmazonEKSVPCCNIRole` を作成した[既存の IAM 役割](#)の名前に置き換えます。ロールを指定するには、クラスター用に IAM OpenID Connect (OIDC) プロバイダーが必要です。クラスター用に持っているかどうかを確認、あるいは作成するには「[the section called “IAM OIDC プロバイダー”](#)」を参照してください。

```
aws eks create-addon --cluster-name my-cluster --addon-name vpc-cni --addon-version
v1.19.2-eksbuild.1 \
--service-account-role-arn arn:aws:iam::111122223333:role/AmazonEKSVPCCNIRole
```

Amazon EKS アドオンのデフォルト設定と競合するカスタム設定を現在のアドオンに適用した場合、作成が失敗する可能性があります。作成に失敗した場合、問題解決に役立つエラーが表示されます。または、前のコマンドに `--resolve-conflicts OVERWRITE` を追加することもできます。これにより、アドオンは既存のカスタム設定を上書きできます。アドオンを作成したら、カスタム設定で更新できます。

5. クラスターの Kubernetes バージョン用のアドオンの最新バージョンがクラスターに追加されたことを確認します。`#####` の部分は、自分のクラスター名に置き換えます。

```
aws eks describe-addon --cluster-name my-cluster --addon-name vpc-cni --query
addon.addonVersion --output text
```

アドオンの作成が完了するまでに数秒かかる場合があります。

出力例は次のとおりです。

```
v1.19.2-eksbuild.1
```

6. 元のアドオンに対してカスタム設定を行った場合は、Amazon EKS アドオンを作成する前に、前のステップで保存した設定を使用して、カスタム設定で EKS アドオンを更新します。「[the section called “更新 \(EKS アドオン\)”](#)」のステップを実行してください。



7. (オプション)cni-metrics-helper をクラスターにインストールします。メトリクスヘルパーは、ネットワークインターフェイスと IP アドレス情報を収集し、クラスターレベルでメトリクスを集計し、メトリクスを Amazon CloudWatch に発行するために使用できるツールです。詳細については[GitHub の CNI](#) を参照してください。

## Amazon VPC CNI を更新する (Amazon EKS アドオン)

Amazon EKS タイプの Amazon VPC CNI Plugin for Kubernetes アドオンを更新します。Amazon EKS タイプのアドオンをクラスターに追加していない場合は「[the section called “作成”](#)」に従ってインストールできます。または「[the section called “更新 \( 自己管理 \)”](#)」に従って他のタイプの VPC CNI インストールを更新します。

1. クラスターにインストールされているアドオンのバージョンを確認します。##### の部分は自分のクラスター名に置き換えます。

```
aws eks describe-addon --cluster-name my-cluster --addon-name vpc-cni --query
"addon.addonVersion" --output text
```

出力例は次のとおりです。

```
v1.16.4-eksbuild.2
```

バージョンを [the section called “Amazon VPC CNI のバージョン”](#) の最新バージョンの表と比較します。返されたバージョンが、最新バージョンの表にあるクラスターの Kubernetes バージョンのバージョンと同じである場合は既に最新バージョンがクラスターにインストールされているため、この手順の残りを完了する必要はありません。出力にバージョン番号ではなくエラーが表示される場合はアマゾン EKS タイプのアドオンがクラスターにインストールされていません。この手順でアドオンを更新する前に、アドオンを作成する必要があります。VPC CNI アドオンの Amazon EKS タイプを作成するには「[the section called “作成”](#)」に従います。

2. 現在インストールされているアドオンの設定を保存します。

```
kubectl get daemonset aws-node -n kube-system -o yaml > aws-k8s-cni-old.yaml
```

3. AWS CLI を使用してアドオンを更新します。AWS Management Console または eksctl を使用してアドオンを更新する場合は「[the section called “アドオンを更新する”](#)」を参照してください。デバイスに沿ったコマンドをコピーします。必要に応じてコマンドに次の変更を加え、変更したコマンドを実行してください。

- ##### の部分は、自分のクラスター名に置き換えます。
- `v1.19.2-eksbuild.1` を、使用しているクラスターバージョンに対して最新バージョンの表に記載されている最新バージョンに置き換えます。
- `111122223333` を、アカウント ID に置き換えます。また、`AmazonEKSVPCCNIRole` を、作成した既存の IAM 役割の名前に置き換えます。VPC CNI の IAM 役割を作成するには「[the section called “ステップ 1: Amazon VPC CNI plugin for Kubernetes の IAM ロールを作成する”](#)」を参照してください。ロールを指定するには、クラスター用に IAM OpenID Connect (OIDC) プロバイダーが必要です。クラスター用に持っているかどうかを確認、あるいは作成するには「[the section called “IAM OIDC プロバイダー”](#)」を参照してください。
- `--resolve-conflicts PRESERVE` オプションはアドオンの既存の設定値を保存します。アドオン設定にカスタム値を設定していて、このオプションを使用しない場合、アマゾン EKS は値をデフォルト値で上書きします。このオプションを使用する場合、実稼働クラスターのアドオンを更新する前に、非稼働クラスターのフィールドおよび値変更をテストすることをお勧めします。この値を `OVERWRITE` に変更する場合、すべての設定がアマゾン EKS のデフォルト値に変更されます。いずれかの設定にカスタム値を設定した場合、アマゾン EKS のデフォルト値で上書きされる可能性があります。この値を `none` に変更した場合、アマゾン EKS は設定の値を一切変更しませんが、更新が失敗する可能性があります。更新に失敗した場合、競合の解決に役立つエラーメッセージが返されます。
- 構成設定を更新しない場合はコマンドから `--configuration-values '{"env":{"AWS_VPC_K8S_CNI_EXTERNALSNAT":"true"}}'` を削除します。構成設定を更新する場合は `"env":{"AWS_VPC_K8S_CNI_EXTERNALSNAT":"true"}` を、希望する設定に置き換えます。この例では `AWS_VPC_K8S_CNI_EXTERNALSNAT` 環境変数は `true` に設定されています。指定する値は設定スキーマに対して有効である必要があります。設定スキーマが不明である場合は、`aws eks describe-addon-configuration --addon-name vpc-cni --addon-version v1.19.2-eksbuild.1` を実行して、`v1.19.2-eksbuild.1` を、設定を表示するアドオンのバージョン番号に置き換えます。出力でスキーマが返されます。既存のカスタム設定があり、それをすべて削除してすべての設定の値を Amazon EKS のデフォルトに戻したい場合はコマンドから `"env":{"AWS_VPC_K8S_CNI_EXTERNALSNAT":"true"}` を削除して、`{}` が空になるようにします。各設定の説明についてはGitHub の「[CNI 設定変数](#)」を参照してください。

```
aws eks update-addon --cluster-name my-cluster --addon-name vpc-cni --addon-version
v1.19.2-eksbuild.1 \
 --service-account-role-arn arn:aws:iam::111122223333:role/AmazonEKSVPCCNIRole \
 --resolve-conflicts PRESERVE --configuration-values '{"env":
{"AWS_VPC_K8S_CNI_EXTERNALSNAT":"true"}}'
```

更新が完了するまでに数秒かかる場合があります。

4. アドオンのバージョンが更新されたことを確認します。##### の部分は自分のクラスター名に置き換えます。

```
aws eks describe-addon --cluster-name my-cluster --addon-name vpc-cni
```

更新が完了するまでに数秒かかる場合があります。

出力例は次のとおりです。

```
{
 "addon": {
 "addonName": "vpc-cni",
 "clusterName": "my-cluster",
 "status": "ACTIVE",
 "addonVersion": "v1.19.2-eksbuild.1",
 "health": {
 "issues": []
 },
 "addonArn": "arn:aws:eks:region:111122223333:addon/my-cluster/vpc-cni/74c33d2f-b4dc-8718-56e7-9fdfa65d14a9",
 "createdAt": "2023-04-12T18:25:19.319000+00:00",
 "modifiedAt": "2023-04-12T18:40:28.683000+00:00",
 "serviceAccountRoleArn": "arn:aws:iam::111122223333:role/AmazonEKSVPCCNIRole",
 "tags": {},
 "configurationValues": "{\"env\":{\"AWS_VPC_K8S_CNI_EXTERNALSNAT\": \"true\"}}\"
 }
}
```

## Amazon VPC CNI の更新 (セルフマネージド型アドオン)

### Important

セルフマネージド型のアドオンを使用する代わりに、アマゾン EKS タイプのアドオンをクラスターに追加することをお勧めします。タイプの違いがよくわからない場合は「[the section called “Amazon EKS アドオン”](#)」を参照してください。アマゾン EKS アドオンをクラスターに追加する方法については「[the section called “アドオンを作成する”](#)」を参照して

ください。アマゾン EKS アドオンを使用できない場合はその理由に関する問題を[コンテナロードマップ GitHub リポジトリ](#)に送信することをお勧めします。

1. Amazon EKS タイプのアドオンがクラスターにインストールされていないことを確認します。##  
##### の部分は自分のクラスター名に置き換えます。

```
aws eks describe-addon --cluster-name my-cluster --addon-name vpc-cni --query
addon.addonVersion --output text
```

エラーメッセージが返された場合、クラスターに Amazon EKS タイプのアドオンがインストールされていません。アドオンを自己管理するにはこの手順の残りのステップを完了してアドオンを更新します。バージョン番号が返された場合、クラスターに アマゾン EKS タイプのアドオンがインストールされています。更新するにはこの手順を使用するのではなく、「[the section called “アドオンを更新する”](#)」の手順を使用してください。アドオンタイプの違いがよくわからない場合は「[the section called “Amazon EKS アドオン”](#)」を参照してください。

2. クラスターに現在インストールされているコンテナイメージのバージョンを確認します。

```
kubectl describe daemonset aws-node --namespace kube-system | grep amazon-k8s-cni: |
cut -d : -f 3
```

出力例は次のとおりです。

```
v1.19.2-eksbuild.1
```

出力にビルド番号が含まれていない場合があります。

3. 現在の設定をバックアップして、バージョンを更新した後も同じ設定を構成できるようにします。

```
kubectl get daemonset aws-node -n kube-system -o yaml > aws-k8s-cni-old.yaml
```

利用可能なバージョンを確認し、更新するバージョンの変更について理解するには、GitHub の「[Releases](#)」を参照してください。新しいバージョンが GitHub で利用できる場合でも、利用可能な最新バージョンの表に記載されている同じ major.minor.patch バージョンに更新することをお勧めします。利用可能な最新バージョンの表については「[the section called “Amazon VPC CNI のバージョン”](#)」を参照してください。表に記載されているビルドバージョンは GitHub に記載

されているセルフマネージドバージョンでは指定されていません。次のオプションのいずれかでタスクを完了して、バージョンを更新します。

- アドオンのカスタム設定がない場合は更新先の[リリース](#)の GitHub の To apply this release: の見出しの下にあるコマンドを実行してください。
- カスタム設定がある場合は次のコマンドを実行してマニフェストファイルをダウンロードします。<https://raw.githubusercontent.com/aws/amazon-vpc-cni-k8s/v1.19.2/config/master/aws-k8s-cni.yaml> を、更新先の GitHub 上のリリースの URL に変更します。

```
curl -O https://raw.githubusercontent.com/aws/amazon-vpc-cni-k8s/v1.19.2/config/master/aws-k8s-cni.yaml
```

必要に応じて、前のステップで作成したバックアップのカスタム設定でマニフェストを変更し、変更したマニフェストをクラスターに適用します。ノードがイメージの取得元のプライベート Amazon EKS Amazon ECR リポジトリにアクセスできない場合 (マニフェストの image: で始まる行を参照)、イメージをダウンロードして自分のリポジトリにコピーし、リポジトリからイメージを取得するようにマニフェストを変更する必要があります。詳細については「[the section called “リポジトリにイメージをコピーする”](#)」を参照してください。

```
kubectl apply -f aws-k8s-cni.yaml
```

4. 新しいバージョンがクラスターにインストールされたことを確認します。

```
kubectl describe daemonset aws-node --namespace kube-system | grep amazon-k8s-cni: |
cut -d : -f 3
```

出力例は次のとおりです。

```
v1.19.2
```

5. (オプション)cni-metrics-helper をクラスターにインストールします。メトリクスヘルパーは、ネットワークインターフェイスと IP アドレス情報を収集し、クラスターレベルでメトリクスを集計し、メトリクスを アマゾン CloudWatch に発行するために使用できるツールです。詳細については[GitHub の CNI](#) を参照してください。

## IRSA を使用するように Amazon VPC CNI プラグインを設定する

[Amazon VPC CNI plugin for Kubernetes](#) は、Amazon EKS クラスター内の Pod ネットワーキング用のネットワークプラグインです。このプラグインは、Kubernetes ノードに VPC IP アドレスを割り当て、各ノードの Pod に必要なネットワーキングを構成します。プラグイン:

- AWS アイデンティティとアクセス管理 (IAM のアクセス許可が必要です。クラスターが IPv4 ファミリーを使用する場合、このアクセス許可は、[AmazonEKS\\_CNI\\_Policy](#) AWS 管理ポリシーで指定されます。クラスターが IPv6 ファミリーを使用する場合には、作成した IAM ポリシーにアクセス許可を追加する必要があります。手順については、「[the section called “IPv6 ファミリーを使用するクラスター用に IAM ポリシーを作成します。”](#)」を参照してください。このポリシーは Amazon EKS ノード IAM ロール または 個別の IAM ロールにアタッチすることができます。Amazon EKS ノード IAM ロールにポリシーをアタッチする手順については「[the section called “ノードの IAM ロール”](#)」を参照してください。このトピックで詳細に説明するように、別のロールに割り当てることをお勧めします。
- デプロイ時に作成すると、aws-node という名前の Kubernetes サービスアカウントを使用するように設定されます。サービスアカウントは aws-node という名前の Kubernetes clusterrole にバインドされます。これには、必要な Kubernetes アクセス許可が割り当てられています。

### Note

IMDS へのアクセスをブロックする場合を除き、Amazon VPC CNI plugin for Kubernetes 用の Pod には、[Amazon EKS ノード IAM ロール](#)に割り当てられたパーミッションへのアクセス権があります。詳細については「[ワーカーノードに割り当てられたインスタンスプロファイルへのアクセスを制限する](#)」を参照してください。

- 既存の Amazon EKS クラスターが必要です。デプロイするには「[はじめに](#)」を参照してください。
- クラスターに、既存の AWS Identity and Access Management (IAM) OpenID Connect (OIDC) プロバイダーが必要です。既に存在しているかどうかを確認する、または作成するには「[the section called “IAM OIDC プロバイダー”](#)」を参照してください。

### ステップ 1: Amazon VPC CNI plugin for Kubernetes の IAM ロールを作成する

1. クラスターで使用する IP ファミリーを決定します。

```
aws eks describe-cluster --name my-cluster | grep ipFamily
```

出力例は次のとおりです。

```
"ipFamily": "ipv4"
```

この出力では代わりに ipv6 が返されることがあります。

2. IAM ロールを作成します。IAM ロールを作成するには `eksctl` または `kubectl` および AWS CLI を使用しています。

`eksctl`

- クラスターの IP ファミリーに適合するコマンドを使用して IAM ロールを作成し、そのロールに IAM ポリシーをアタッチします。このコマンドでは、IAM ロールを作成する AWS CloudFormation スタックを作成およびデプロイし、そのために指定したポリシーをアタッチします。さらに、既存の `aws-node` Kubernetes サービスアカウントに対し、作成された IAM ロールの ARN をアノテーションします。

- IPv4

**#####** を独自の値に置き換えます。

```
eksctl create iamserviceaccount \
 --name aws-node \
 --namespace kube-system \
 --cluster my-cluster \
 --role-name AmazonEKSVPCCNIRole \
 --attach-policy-arn arn:aws:iam::aws:policy/AmazonEKS_CNI_Policy \
 --override-existing-serviceaccounts \
 --approve
```

- IPv6

**#####** を独自の値に置き換えます。**111122223333** を、ご自身のアカウント ID に置き換えます。また、**AmazonEKS\_CNI\_IPv6\_Policy** を、IPv6 ポリシー名に置き換えます。IPv6 ポリシーがない場合は [the section called “IPv6 ファミリーを使用するクラスター用に IAM ポリシーを作成します。”](#) を参照して作成します。クラスターで IPv6 を使用するにはいくつかの要件を満たす必要があります。詳細については「[the section called “IPv6”](#)」を参照してください。



```
eksctl create iamserviceaccount \
 --name aws-node \
 --namespace kube-system \
 --cluster my-cluster \
 --role-name AmazonEKSVPCCNIRole \
 --attach-policy-arn arn:aws:iam::111122223333:policy/
AmazonEKS_CNI_IPv6_Policy \
 --override-existing-serviceaccounts \
 --approve
```

## kubectl と AWS CLI

- i. クラスターの OIDC プロバイダーの URL を表示します。

```
aws eks describe-cluster --name my-cluster --query
"cluster.identity.oidc.issuer" --output text
```

出力例は次のとおりです。

```
https://oidc.eks.region-code.amazonaws.com/id/EXAMPLED539D4633E53DE1B71EXAMPLE
```

出力が返されない場合は[クラスター用の IAM OIDC プロバイダーを作成する](#)必要があります。

- ii. 次の内容を *vpc-cni-trust-policy.json* という名前のファイルにコピーします。**111122223333** を、ご自身のアカウント ID および前のステップで返された出力 **EXAMPLED539D4633E53DE1B71EXAMPLE** に置き換えます。**#####** を、クラスターのある AWS リージョンに置き換えます。

```
{
 "Version": "2012-10-17",
 "Statement": [
 {
 "Effect": "Allow",
 "Principal": {
 "Federated": "arn:aws:iam::111122223333:oidc-provider/
oidc.eks.region-code.amazonaws.com/id/EXAMPLED539D4633E53DE1B71EXAMPLE"
 },
 "Action": "sts:AssumeRoleWithWebIdentity",
 "Condition": {
 "StringEquals": {
```



```

 "oidc.eks.region-code.amazonaws.com/id/
EXAMPLED539D4633E53DE1B71EXAMPLE:aud": "sts.amazonaws.com",
 "oidc.eks.region-code.amazonaws.com/id/
EXAMPLED539D4633E53DE1B71EXAMPLE:sub": "system:serviceaccount:kube-system:aws-
node"
 }
}
]
}

```

- iii. ロールを作成します。*AmazonEKSVPCCNIRole* は任意の名前に置き換えることができます。

```

aws iam create-role \
 --role-name AmazonEKSVPCCNIRole \
 --assume-role-policy-document file:///vpc-cni-trust-policy.json

```

- iv. 必要な IAM ポリシーをロールにアタッチします。クラスターの IP ファミリーに適合したコマンドを実行してください。

- IPv4

```

aws iam attach-role-policy \
 --policy-arn arn:aws:iam::aws:policy/AmazonEKS_CNI_Policy \
 --role-name AmazonEKSVPCCNIRole

```

- IPv6

*111122223333* はご自身のアカウント ID に置き換え、*AmazonEKS\_CNI\_IPv6\_Policy* はご使用の IPv6 ポリシーの名前に置き換えます。IPv6 ポリシーがない場合は[the section called “IPv6 ファミリーを使用するクラスター用に IAM ポリシーを作成します。”](#)を参照して作成します。クラスターで IPv6 を使用するにはいくつかの要件を満たす必要があります。詳細については「[the section called “IPv6”](#)」を参照してください。

```

aws iam attach-role-policy \
 --policy-arn arn:aws:iam::111122223333:policy/AmazonEKS_CNI_IPv6_Policy \
 --role-name AmazonEKSVPCCNIRole

```

- v. 次のコマンドを実行し、先に作成した IAM ロールの ARN で aws-node サービスアカウントをアノテーションします。*example* の値は独自の値に置き換えます。

```
kubectl annotate serviceaccount \
 -n kube-system aws-node \
 eks.amazonaws.com/role-arn=arn:aws:iam::111122223333:role/
AmazonEKSVPCCNIRole
```

3. (オプション) Kubernetes サービスアカウントで使用されている AWS Security Token Service エンドポイントタイプを設定します。詳細については、「[the section called “STS エンドポイント”](#)」を参照してください。

## ステップ 2: Amazon VPC CNI plugin for Kubernetes の Pod を再デプロイする

1. 認証情報環境変数を適用するために、サービスアカウントに関連付けられている既存の Pod を削除して再作成します。現在アノテーションなしで実行されている Pod には、アノテーションは適用されません。次のコマンドは、既存の aws-node DaemonSet Pod を削除し、サービスアカウントのアノテーションを使用してデプロイします。

```
kubectl delete Pods -n kube-system -l k8s-app=aws-node
```

2. Pod がすべて再起動したことを確認します。

```
kubectl get pods -n kube-system -l k8s-app=aws-node
```

3. Pod の 1 つについて説明し、AWS\_WEB\_IDENTITY\_TOKEN\_FILE および AWS\_ROLE\_ARN 環境変数が存在することを確認します。**cpjw7** を、前のステップの出力で返された、いずれかの Pod の名前に置き換えます。

```
kubectl describe pod -n kube-system aws-node-cpjw7 | grep 'AWS_ROLE_ARN:|
AWS_WEB_IDENTITY_TOKEN_FILE:'
```

出力例は次のとおりです。

```
AWS_ROLE_ARN: arn:aws:iam::111122223333:role/AmazonEKSVPCCNIRole
 AWS_WEB_IDENTITY_TOKEN_FILE: /var/run/secrets/eks.amazonaws.com/
serviceaccount/token
 AWS_ROLE_ARN: arn:aws:iam::111122223333:role/
AmazonEKSVPCCNIRole
 AWS_WEB_IDENTITY_TOKEN_FILE: /var/run/secrets/eks.amazonaws.com/
serviceaccount/token
```

Pod には 2 つのコンテナが含まれているため、重複した結果の 2 つのセットが返されます。両方のコンテナの値は同じです。

Pod で AWS リージョンエンドポイントを使用している場合、前の出力では下記の行も返されています。

```
AWS_STS_REGIONAL_ENDPOINTS=regional
```

### ステップ 3: ノードの IAM ロールから CNI ポリシーを削除する

現在、[Amazon EKS ノード IAM ロール](#)に AmazonEKS\_CNI\_Policy IAM (IPv4) ポリシーまたは [IPv6 ポリシー](#)がアタッチされており、さらに、別の IAM ロールを作成し、このロールに対し前出のポリシーをアタッチし、そのロールを aws-node Kubernetes サービスアカウントに割り当てている場合、クラスターの IP ファミリーに適合する AWS CLI コマンドを使用して、ノードのロールからポリシーを削除することをお勧めします。*AmazonEKSNodeRole* を、ノードのロールの名前に置き換えます。

- IPv4

```
aws iam detach-role-policy --role-name AmazonEKSNodeRole --policy-arn
arn:aws:iam::aws:policy/AmazonEKS_CNI_Policy
```

- IPv6

*111122223333* はご自身のアカウント ID に置き換え、*AmazonEKS\_CNI\_IPv6\_Policy* はご使用の IPv6 ポリシーの名前に置き換えます。

```
aws iam detach-role-policy --role-name AmazonEKSNodeRole --policy-arn
arn:aws:iam::111122223333:policy/AmazonEKS_CNI_IPv6_Policy
```

**IPv6** ファミリーを使用するクラスター用に IAM ポリシーを作成します。

IPv6 ファミリーを使用するクラスターを作成し、そのクラスターでバージョン 1.10.1 以降の Amazon VPC CNI plugin for Kubernetes アドオンが設定されている場合は、IAM ロールに割り当てることができる IAM ポリシーを作成する必要があります。作成時に IPv6 ファミリーの使用を設定していない、既存のクラスターにおいて、IPv6 を使用する場合には新しいクラスターを作成する必

必要があります。クラスターでの IPv6 使用の詳細については「[the section called “IPv6”](#)」を参照してください。

1. 次のテキストをコピーし、vpc-cni-ipv6-policy.json という名前のファイルに保存します。

```
{
 "Version": "2012-10-17",
 "Statement": [
 {
 "Effect": "Allow",
 "Action": [
 "ec2:AssignIpv6Addresses",
 "ec2:DescribeInstances",
 "ec2:DescribeTags",
 "ec2:DescribeNetworkInterfaces",
 "ec2:DescribeInstanceTypes"
],
 "Resource": "*"
 },
 {
 "Effect": "Allow",
 "Action": [
 "ec2:CreateTags"
],
 "Resource": [
 "arn:aws:ec2:*:*:network-interface/*"
]
 }
]
}
```

## 2. IAM ポリシーを作成する

```
aws iam create-policy --policy-name AmazonEKS_CNI_IPv6_Policy --policy-document
file://vpc-cni-ipv6-policy.json
```

VPC CNI モードと設定について説明します。

Amazon VPC CNI plugin for Kubernetes は、Pod のネットワーキングを提供します。使用可能なネットワーク機能の詳細については次の表を参照してください。

| ネットワーキング機能                                          | 詳細はこちら                                              |
|-----------------------------------------------------|-----------------------------------------------------|
| クラスター、Pod、サービスに IPv6 アドレスを割り当てるようにクラスターを設定する        | <a href="#">the section called “IPv6”</a>           |
| Pod に IPv4 Source Network Address Translation を使用する | <a href="#">the section called “アウトバウンドトラフィック”</a>  |
| Pod との間のネットワークトラフィックを制限する                           | <a href="#">the section called “トラフィックを制限する”</a>    |
| ノードでセカンダリネットワークインターフェイスをカスタマイズする                    | <a href="#">the section called “カスタムネットワーキング”</a>   |
| ノードの IP アドレスを増やす                                    | <a href="#">the section called “IP アドレスを増やす”</a>    |
| Pod ネットワークトラフィックにセキュリティグループを使用する                    | <a href="#">the section called “ポッドのセキュリティグループ”</a> |
| Pod に複数のネットワークインターフェイスを使用する                         | <a href="#">the section called “複数のインターフェイス”</a>    |

## クラスター、Pod、サービスに対する IPv6 アドレスの説明

適用対象: Amazon EC2 インスタンスを持つ Pod と Fargate Pod

Kubernetes のデフォルトでは、IPv4 アドレスが Pod とサービスに割り当てられます。Pod とサービスに IPv4 アドレスを割り当てる代わりに、IPv6 アドレスを割り当てるようにクラスターを設定できます。Amazon EKS は、Kubernetes がサポートしていても、デュアルスタックのポッドまたはサービスをサポートしません。つまり、Pod とサービスに IPv4 アドレスと IPv6 アドレスの両方を割り当てることはできません。

そのクラスターに使用する IP ファミリーは、クラスターの作成時に選択します。クラスターの作成後は、ファミリーを変更できません。

Amazon EKS IPv6 クラスターをデプロイするチュートリアルについては、「[the section called “デプロイ”](#)」を参照してください。

この機能を使用する際の考慮事項を次に示します。

## IPv6 の機能のサポート

- Windows サポートなし: Windows の Pod とサービスはサポートされていません。
- Nitro ベースの EC2 ノードが必要: IPv6 は、AWS Nitro ベースの Amazon EC2 または Fargate ノードでのみ使用が可能です。
- EC2 ノードおよび Fargate ノードがサポートされる: IPv6 は、Amazon EC2 ノードと Fargate ノードで [the section called “ポッドのセキュリティグループ”](#) とともに使用できます。
- Outposts はサポートされない: IPv6 は [Amazon EKS on AWS Outposts](#) とともに使用できません。
- FSx for Lustre はサポートされない: [the section called “Amazon FSx for Lustre”](#) はサポートされていません。
- インスタンスメタデータサービスはサポートされない: Amazon EC2 [インスタンスメタデータサービスの](#) IPv6 エンドポイントの使用は、Amazon EKS ではサポートされていません。
- カスタムネットワーキングはサポートされない: IP アドレスの枯渇を緩和するために、これまで使用していた [the section called “カスタムネットワーキング”](#) の代わりとして、IPv6 を使用することができます。IPv6 では、カスタムネットワーキングを使用することはできません。クラスターにおいて、ネットワークの分離用としてカスタムネットワーキングを使用する場合は、引き続き、IPv4 ファミリーによるカスタムネットワーキングを使用する必要があります。

## IP アドレスの割り当て

- Kubernetes サービス: Kubernetes サービスには IPv6 アドレスのみが割り当てられます。これらには、IPv4 アドレスは割り当てられません。
- ポッド: ポッドには IPv6 アドレスとホストローカル IPv4 アドレスが割り当てられます。ホストローカル IPv4 アドレスは、VPC CNI と連鎖されたホストローカル CNI プラグインを使用して割り当てられ、そのアドレスは Kubernetes コントロールプレーンに報告されません。これは、ポッドが別の Amazon VPC またはインターネット内の外部 IPv4 リソースと通信する必要がある場合にのみ使用されます。ホストローカル IPv4 アドレスは、ワーカーノードのプライマリ ENI のプライマリ IPv4 アドレスに (VPC CNI によって) SNAT されます。
- Pod とサービス: Pod とサービスには IPv6 アドレスのみが割り当てられます。これらには、IPv4 アドレスは割り当てられません。Pod は、インスタンス自体の NAT を介して IPv4 エンドポイントとの通信が可能なため、[DNS64 および NAT64](#) は必要となりません。トラフィックがパブリック IP アドレスを必要とする場合、そのトラフィックは、送信元ネットワークアドレスとしてパブリック IP に変換されます。

- ルーティングアドレス: VPC の外部と通信する際の、Pod の送信元 IPv6 アドレスは、送信元のネットワークアドレスがノードの IPv6 アドレスに変換されたものではありません。このルーティングは、インターネットゲートウェイ、または送信専用インターネットゲートウェイを使用して行われます。
- ノード: すべてのノードには、IPv4 と IPv6 のアドレスが割り当てられています。
- Fargate ポッド: 各 Fargate Pod は、デプロイ先のサブネット用に指定された CIDR から、IPv6 アドレスを受け取ります。Fargate Pod を実行するための基盤ハードウェアユニットは、そのユニットがデプロイされているサブネットに割り当てられている CIDR から、一意の IPv4 および IPv6 アドレスを取得します。

## EKS で IPv6 を使用方法

- 新しいクラスターを作成する: 新しいクラスターを作成し、そのクラスターで IPv6 ファミリーの使用を指定する必要があります。これより前のバージョンからクラスターを更新して、IPv6 ファミリーを有効化することはできません。新しいクラスターを作成する手順については、「[考慮事項](#)」を参照してください。
- 最新の VPC CNI を使用する: Amazon VPC CNI バージョン 1.10.1 以降をデプロイします。このバージョン以降がデフォルトでデプロイされます。アドオンのデプロイ後は、クラスター内のすべてのノードグループ内にあるすべてのノードを削除しない限り、Amazon VPC CNI アドオンを 1.10.1 より前のバージョンにダウングレードすることはできません。
- **IPv6 用の VPC CNI を設定する:** Amazon EC2 ノードを使用する場合は、IP プレフィックスの委任と IPv6 を使用して Amazon VPC CNI アドオンを設定する必要があります。クラスターの作成時に IPv6 ファミリーを選択した場合は、バージョン 1.10.1 のアドオンが、この設定のデフォルトになります。これは、セルフマネージド型のアドオン、および Amazon EKS アドオンの両方に当てはまります。IP プレフィックス委任の詳細については、「[the section called “IP アドレスを増やす”](#)」を参照してください。
- **IPv4 アドレスと IPv6 アドレスを設定する:** クラスターを作成する際に指定する VPC とサブネットには、それらに割り当てられた IPv6 CIDR ブロックが必要です。同時に、IPv4 CIDR ブロックも割り当てられている必要があります。これは、IPv6 のみを使用する場合でも、VPC が機能するには IPv4 CIDR ブロックが必要になるためです。詳細については、「[Amazon VPC ユーザーガイド](#)」の「[IPv6 CIDR ブロックと VPC の関連付け](#)」を参照してください。
- **IPv6 アドレスをノードに自動割り当てする:** ノードを作成する際は、IPv6 アドレスを自動割り当てするように設定されたサブネットを指定する必要があります。指定していない場合、ノードをデプロイできません。自動割り当ての設定はデフォルトで無効になっています。詳細については、



「Amazon VPC ユーザーガイド」の「[サブネットのパブリック IPv6 アドレス属性を変更する](#)」を参照してください。

- ルートテーブルを設定して **IPv6** を使用する: サブネットに割り当てられるルートテーブルには、IPv6 アドレス用のルートが必要です。詳細については、「Amazon VPC ユーザーガイド」の「[既存の VPC を IPv4 から IPv6 に移行する](#)」を参照してください。
- **IPv6** のセキュリティグループを設定する: セキュリティグループは IPv6 アドレスを許可する必要があります。詳細については、「Amazon VPC ユーザーガイド」の「[既存の VPC を IPv4 から IPv6 に移行する](#)」を参照してください。
- ロードバランサーを設定する: AWS Load Balancer Controller のバージョン 2.3.1 以降を使用して、「[the section called “アプリケーション負荷分散”](#)」を使用した HTTP アプリケーションのロードバランシング、または、「[the section called “ネットワーク負荷分散”](#)」を使用したネットワークトラフィックの IPv6 Pod へのロードバランシングを行います。いずれの場合もインスタンスモードではなく IP モードを使用してください。詳細については、「[the section called “AWS ロードバランサーコントローラー”](#)」を参照してください。
- **IPv6 IAM** ポリシーを追加する: ノード IAM または CNI IAM ロールには、IPv6 の IAM ポリシーをアタッチする必要があります。上記 2 つの中では、CNI IAM ロールへのアタッチが推奨されます。詳細については、「[the section called “IPv6 ファミリーを使用するクラスター用に IAM ポリシーを作成します。”](#)」および「[the section called “ステップ 1: Amazon VPC CNI plugin for Kubernetes の IAM ロールを作成する”](#)」を参照してください。
- すべてのコンポーネントを評価する: アプリケーション、Amazon EKS アドオン、および、IPv6 クラスターのデプロイ前に統合した AWS サービスに関しては、包括的な評価を実施します。これにより、IPv6 を使用した場合も、すべてが想定どおりに動作することを保証できます。
- **BootstrapArguments** セルフマネージドノードグループを追加する: IPv6 ファミリーを使用するクラスターでセルフマネージドノードグループを作成する場合、ユーザーデータには、ノードの起動時に実行される [bootstrap.sh](#) ファイルの次の BootstrapArguments が含まれている必要があります。*your-cidr* をクラスターの VPC の IPv6 CIDR 範囲に置き換えます。

```
--ip-family ipv6 --service-ipv6-cidr your-cidr
```

クラスターの IPv6 CIDR 範囲がわからない場合は、次のコマンドで確認できます (AWS CLI バージョン 2.4.9 以降が必要です)。

```
aws eks describe-cluster --name my-cluster --query
cluster.kubernetesNetworkConfig.serviceIpv6Cidr --output text
```



## Amazon EKS IPv6 クラスターとマネージド Amazon Linux ノードをデプロイする

このチュートリアルでは、IPv6 ファミリーを使用する IPv6 Amazon VPC と Amazon EKS クラスターのデプロイ、および Amazon EC2 Amazon Linux ノードを使用するマネージド型ノードグループのデプロイ方法を説明します。IPv6 クラスター内では、Amazon EC2 Windows のノードをデプロイすることはできません。Fargate ノードをクラスターにデプロイすることもできますが、理解しやすくするため、これらの手順はこのトピックでは説明していません。

### 前提条件

このチュートリアルを開始する前に以下を完了してください。

Amazon EKS クラスターの作成と管理に必要な次のツールおよびリソースをインストールおよび設定します。

- すべての設定を習熟し、要件を満たす設定でクラスターをデプロイすることをお勧めします。詳細については、「[the section called “クラスターを作成する”](#)」、「[the section called “マネージド型ノードグループ”](#)」、ならびにこのトピックの「[考慮事項](#)」を参照してください。一部の設定の有効化は、クラスターの作成時にのみ行えます。
- デバイスまたは AWS クラウドシェル に、kubectl コマンドラインツールがインストールされていること。バージョンはご使用のクラスターの Kubernetes バージョンと同じか、1 つ前のマイナーバージョン以前、あるいはそれより新しいバージョンが使用できます。例えば、クラスターのバージョンが 1.29 である場合、kubectl のバージョン 1.28、1.29、または 1.30 が使用できます。kubectl をインストールまたはアップグレードする方法については「[the section called “kubectl と eksctl のセットアップ”](#)」を参照してください。
- 使用している IAM セキュリティプリシパルは、Amazon EKS の IAM ロール、サービスにリンクされたロール、AWS CloudFormation、VPC、関連リソースを使用するために許可が必要です。詳細については、「IAM ユーザーガイド」の「[アクション](#)」および「[サービスにリンクされたロールの使用](#)」を参照してください。
- eksctl を使用する場合は、コンピュータにバージョン 0.210.0 以降をインストールします。インストールまたはアップグレードするには、eksctl ドキュメントの「[インストール](#)」を参照してください。
- ご使用のデバイスまたは AWS CloudShell で、バージョン 2.12.3 以降、または AWS コマンドラインインターフェイス (AWS CLI) のバージョン 1.27.160 以降がインストールおよび設定されていること。現在のバージョンを確認するには「aws --version | cut -d / -f2 | cut -d ' ' -f1」を参照してください。yum、apt-get、macOS 用の Homebrew などのパッケージマネージャーは、多くの場合 AWS CLI の最新バージョンより数バージョン古くなっています。最新バージョンをインストールするには「AWS コマンドラインインターフェイスユーザーガイド」

の「[インストール](#)」および「[aws configure を使用したクイック設定](#)」を参照してください。AWS クラウドシェル にインストールされている AWS CLI バージョンも最新バージョンより数バージョン遅れることがあります。更新するには、「AWS CloudShell ユーザーガイド」の「[ホームディレクトリへの AWS CLI のインストール](#)」を参照してください。AWS CloudShell を使用する場合は、[バージョン 2.12.3 以降または 1.27.160 以降の AWS CLI をインストール](#)する必要があります。AWS CloudShell には、デフォルトとして AWS CLI の古いバージョンがインストールされている可能性があります。

eksctl または CLI を使用して IPv6 クラスターをデプロイできます。

eksctl を使用して IPv6 クラスターをデプロイする

a. `ipv6-cluster.yaml` ファイルを作成します。デバイスに沿ったコマンドをコピーします。必要に応じてコマンドに次の変更を加え、変更したコマンドを実行してください：

- `my-cluster` の部分は、自分のクラスター名に置き換えます。この名前には英数字 (大文字と小文字が区別されます) とハイフンのみを使用できます。先頭の文字は英数字である必要があります。また、100 文字より長くすることはできません。名前は、クラスターを作成する AWS リージョンおよび AWS アカウント内で一意である必要があります。
- `region-code` は、Amazon EKS がサポートする任意の AWS リージョンに置き換えます。AWS リージョンの一覧については、AWS の全般的なリファレンスガイドの [Amazon EKS エンドポイントとクォータ](#) を参照してください。
- `version` の値には、クラスターのバージョンを設定する必要があります。詳細については、「[the section called “Kubernetes バージョン”](#)」を参照してください。
- `my-nodegroup` を、ノードグループの名前に置き換えます。ノードグループ名は 63 文字以下である必要があります。先頭は文字または数字でなければなりませんが、残りの文字にはハイフンおよびアンダースコアを含めることもできます。
- `t3.medium` の部分は、任意の [AWS Nitro System インスタンスタイプ](#) に置き換えます。

```
cat >ipv6-cluster.yaml <<EOF

apiVersion: eksctl.io/v1alpha5
kind: ClusterConfig

metadata:
 name: my-cluster
 region: region-code
 version: "X.XX"
```

```
kubernetesNetworkConfig:
 ipFamily: IPv6

addons:
- name: vpc-cni
 version: latest
- name: coredns
 version: latest
- name: kube-proxy
 version: latest

iam:
 withOIDC: true

managedNodeGroups:
- name: my-nodegroup
 instanceType: t3.medium
EOF
```

## b. クラスターを作成する

```
eksctl create cluster -f ipv6-cluster.yaml
```

クラスターの作成には数分かかります。出力の最後の行が表示されるまで、次の手順には進まないでください。この行は、下記のような出力になります。

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

## c. デフォルトの Pod に、IPv6 アドレスが割り当てられていることを確認します。

```
kubectl get pods -n kube-system -o wide
```

出力例は次のとおりです。

| NAME                                   | READY | STATUS  | RESTARTS | AGE   | IP                                            | NOMINATED NODE |
|----------------------------------------|-------|---------|----------|-------|-----------------------------------------------|----------------|
| aws-node-rslts                         | 1/1   | Running | 1        | 5m36s |                                               |                |
| 2600:1f13:b66:8200:11a5:ade0:c590:6ac8 |       |         |          |       | ip-192-168-34-75.region-code.compute.internal |                |
|                                        |       |         |          |       |                                               | <none>         |

```
aws-node-t74jh 1/1 Running 0 5m32s
 2600:1f13:b66:8203:4516:2080:8ced:1ca9 ip-192-168-253-70.region-
code.compute.internal <none> <none>
coredns-85d5b4454c-cw7w2 1/1 Running 0 56m
 2600:1f13:b66:8203:34e5:: ip-192-168-253-70.region-
code.compute.internal <none> <none>
coredns-85d5b4454c-tx6n8 1/1 Running 0 56m
 2600:1f13:b66:8203:34e5::1 ip-192-168-253-70.region-
code.compute.internal <none> <none>
kube-proxy-btpbk 1/1 Running 0 5m36s
 2600:1f13:b66:8200:11a5:ade0:c590:6ac8 ip-192-168-34-75.region-
code.compute.internal <none> <none>
kube-proxy-jjk2g 1/1 Running 0 5m33s
 2600:1f13:b66:8203:4516:2080:8ced:1ca9 ip-192-168-253-70.region-
code.compute.internal <none> <none>
```

- d. デフォルトのサービスに、IPv6 アドレスが割り当てられていることを確認します。

```
kubectl get services -n kube-system -o wide
```

出力例は次のとおりです。

| NAME         | TYPE      | CLUSTER-IP        | EXTERNAL-IP | PORT(S)       | AGE  |
|--------------|-----------|-------------------|-------------|---------------|------|
| SELECTOR     |           |                   |             |               |      |
| kube-dns     | ClusterIP | fd30:3087:b6c2::a | <none>      | 53/UDP,53/TCP | 57m  |
| app=kube-dns |           |                   |             |               | k8s- |

- e. (オプション) [サンプルアプリケーションをデプロイする](#)か、[AWS Load Balancer コントローラーとサンプルアプリケーションをデプロイ](#)して、「[the section called “アプリケーション負荷分散”](#)」に従い HTTP アプリケーションをロードバランシングするか、「[the section called “ネットワーク負荷分散”](#)」に従いネットワークトラフィックを IPv6 Pod にロードバランシングします。
- f. このチュートリアルのために作成したクラスターとノードの使用が終了したら、それらのリソースは、次のコマンドにより削除しておきます。

```
eksctl delete cluster my-cluster
```

## AWS CLI を使用して IPv6 クラスターをデプロイする

 Important

- この手順のすべてのステップは、同一のユーザーとして実行する必要があります。現在のユーザーを確認するには、次のコマンドを実行します。

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

- この手順のすべてのステップは、同一のシェル内で実行する必要があります。いくつかのステップには、これより前のステップで設定した変数を使用します。変数の値が異なるシェル内で設定されていると、その変数を使用するステップは正しく機能しません。[AWS CloudShell](#) を使用して次の手順を実行する際、約 20～30 分の間キーボードまたはポインタによる操作がないと、シェルセッションが終了することを銘記しておいてください。実行中のプロセスは、操作数としてカウントされません。
- 各命令は Bash シェル用に記述されているので、他のシェルでは修正が必要な場合があります。

この手順の各ステップにおいて、すべての *example values* は、ご自分が使用する値に置き換える必要があります。

- 以下のコマンドを実行して、後のステップで使用するいくつかの変数を設定します。*region-code* は、リソースをデプロイする AWS リージョンに置き換えます。この値には、Amazon EKS でサポートされている任意の AWS リージョンが設定できます。AWS リージョンの一覧については、AWS の全般的なリファレンスガイドの [Amazon EKS エンドポイントとクォータ](#) を参照してください。*my-cluster* の部分は、自分のクラスター名に置き換えます。この名前には英数字 (大文字と小文字が区別されます) とハイフンのみを使用できます。先頭の文字は英数字である必要があります。また、100 文字より長くすることはできません。名前は、クラスターを作成する AWS リージョンおよび AWS アカウント内で一意である必要があります。*my-nodegroup* を、ノードグループの名前に置き換えます。ノードグループ名は 63 文字以下である必要があります。先頭は文字または数字でなければなりませんが、残りの文字にはハイフンおよびアンダースコアを含めることもできます。*111122223333* は、ご自分のアカウント ID に置き換えます。

```
export region_code=region-code
export cluster_name=my-cluster
export nodegroup_name=my-nodegroup
```

```
export account_id=111122223333
```

b. パブリックサブネットとプライベートサブネットを持ち、Amazon EKS と IPv6 の要件を満たす Amazon VPC を作成します。

i. 次のコマンドを実行して、AWS CloudFormation スタック名のための変数を設定します。*my-eks-ipv6-vpc* は、任意の名前に置き換えることができます。

```
export vpc_stack_name=my-eks-ipv6-vpc
```

ii. AWS CloudFormation のテンプレートを使用して IPv6 VPC を作成します。

```
aws cloudformation create-stack --region $region_code --stack-name $vpc_stack_name \
 --template-url https://s3.us-west-2.amazonaws.com/amazon-eks/
cloudformation/2020-10-29/amazon-eks-ipv6-vpc-public-private-subnets.yaml
```

スタックが作成されるまで数分かかります。以下のコマンドを実行してください。コマンドの出力が「CREATE\_COMPLETE」になるまで、次のステップに進まないでください。

```
aws cloudformation describe-stacks --region $region_code --stack-name
$vpc_stack_name --query Stacks[].StackStatus --output text
```

iii. 作成されたパブリックサブネットの ID を取得します。

```
aws cloudformation describe-stacks --region $region_code --stack-name
$vpc_stack_name \
 --query='Stacks[].Outputs[?OutputKey==`SubnetsPublic`].OutputValue' --output
text
```

出力例は次のとおりです。

```
subnet-0a1a56c486EXAMPLE,subnet-099e6ca77aEXAMPLE
```

iv. 作成されたパブリックサブネットで、IPv6 アドレスの自動割り当てオプションを有効にします。

```
aws ec2 modify-subnet-attribute --region $region_code --subnet-id
subnet-0a1a56c486EXAMPLE --assign-ipv6-address-on-creation
aws ec2 modify-subnet-attribute --region $region_code --subnet-id
subnet-099e6ca77aEXAMPLE --assign-ipv6-address-on-creation
```

- v. テンプレートによって作成されたサブネットおよびセキュリティグループの名前を、デプロイされた AWS CloudFormation スタックから取得します。これらの名前は、後のステップで使用するために変数に格納しておきます。

```
security_groups=$(aws cloudformation describe-stacks --region $region_code --
stack-name $vpc_stack_name \
 --query='Stacks[].Outputs[?OutputKey==`SecurityGroups`].OutputValue' --output
text)

public_subnets=$(aws cloudformation describe-stacks --region $region_code --stack-
name $vpc_stack_name \
 --query='Stacks[].Outputs[?OutputKey==`SubnetsPublic`].OutputValue' --output
text)

private_subnets=$(aws cloudformation describe-stacks --region $region_code --
stack-name $vpc_stack_name \
 --query='Stacks[].Outputs[?OutputKey==`SubnetsPrivate`].OutputValue' --output
text)

subnets=${public_subnets},${private_subnets}
```

- c. クラスターの IAM ロールを作成して、そのロールに必要な Amazon EKS IAM 管理ポリシーをアタッチします。Amazon EKS によって管理される Kubernetes クラスターは、そのサービスで使用するリソースを管理するために、ユーザーに代わって他の AWS のサービスを呼び出します。
- i. 次のコマンドを実行して、eks-cluster-role-trust-policy.json ファイルを作成します。

```
cat >eks-cluster-role-trust-policy.json <<EOF
{
 "Version": "2012-10-17",
 "Statement": [
 {
 "Effect": "Allow",
 "Principal": {
 "Service": "eks.amazonaws.com"
 },
 "Action": "sts:AssumeRole"
 }
]
}
EOF
```

- ii. 次のコマンドを実行して、ロール名の変数を設定します。`myAmazonEKSClusterRole` は、任意の名前に置き換えることができます。

```
export cluster_role_name=myAmazonEKSClusterRole
```

- iii. ロールを作成します。

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

- iv. IAM ロールの ARN を取得し、後のステップで使用するために変数に格納します。

```
CLUSTER_IAM_ROLE=$(aws iam get-role --role-name $cluster_role_name --
query="Role.Arn" --output text)
```

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

```
aws iam attach-role-policy --policy-arn arn:aws:iam::aws:policy/
AmazonEKSClusterPolicy --role-name $cluster_role_name
```

#### d. クラスターを作成する

```
aws eks create-cluster --region $region_code --name $cluster_name --kubernetes-
version 1.XX \
 --role-arn $CLUSTER_IAM_ROLE --resources-vpc-config subnetIds=
$subnets,securityGroupIds=$security_groups \
 --kubernetes-network-config ipFamily=ipv6
```

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

クラスターが作成されるまでに数分かかります。以下のコマンドを実行してください。コマンドの出力が「ACTIVE」になるまで、次のステップに進まないでください。

```
aws eks describe-cluster --region $region_code --name $cluster_name --query
cluster.status
```



- e. クラスターの kubeconfig ファイルを作成または更新し、そのクラスターと通信できるようにします。

```
aws eks update-kubeconfig --region $region_code --name $cluster_name
```

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

- f. ノードの IAM ロールを作成します。
- i. 次のコマンドを実行して、vpc-cni-ipv6-policy.json ファイルを作成します。

```
cat >vpc-cni-ipv6-policy.json <<EOF
{
 "Version": "2012-10-17",
 "Statement": [
 {
 "Effect": "Allow",
 "Action": [
 "ec2:AssignIpv6Addresses",
 "ec2:DescribeInstances",
 "ec2:DescribeTags",
 "ec2:DescribeNetworkInterfaces",
 "ec2:DescribeInstanceTypes"
],
 "Resource": "*"
 },
 {
 "Effect": "Allow",
 "Action": [
 "ec2:CreateTags"
],
 "Resource": [
 "arn:aws:ec2:*:*:network-interface/*"
]
 }
]
}
EOF
```

- ii. IAM ポリシーを作成します。

```
aws iam create-policy --policy-name AmazonEKS_CNI_IPv6_Policy --policy-document
file://vpc-cni-ipv6-policy.json
```

- iii. 次のコマンドを実行して、`node-role-trust-relationship.json` ファイルを作成します。

```
cat >node-role-trust-relationship.json <<EOF
{
 "Version": "2012-10-17",
 "Statement": [
 {
 "Effect": "Allow",
 "Principal": {
 "Service": "ec2.amazonaws.com"
 },
 "Action": "sts:AssumeRole"
 }
]
}
EOF
```

- iv. 次のコマンドを実行して、ロール名の変数を設定します。`AmazonEKSNodeRole` は、任意の名前に置き換えることができます。

```
export node_role_name=AmazonEKSNodeRole
```

- v. IAM ロールを作成します。

```
aws iam create-role --role-name $node_role_name --assume-role-policy-document
file://node-role-trust-relationship.json
```

- vi. IAM ロールに IAM ポリシーをアタッチします。

```
aws iam attach-role-policy --policy-arn arn:aws:iam::$account_id:policy/
AmazonEKS_CNI_IPv6_Policy \
 --role-name $node_role_name
```

 Important

簡単にするため、このチュートリアルでは、ポリシーが以下の IAM ロールにアタッチされています。ただし、本番向けクラスターの場合は、異なる IAM ロールにポリシーをアタッチすることをお勧めします。詳細については、「[the section called “IRSA 用の設定”](#)」を参照してください。

vii. IAM ロールに、2 つの必須なマネージド IAM ポリシーをアタッチします。

```
aws iam attach-role-policy --policy-arn arn:aws:iam::aws:policy/
AmazonEKSWorkerNodePolicy \
 --role-name $node_role_name
aws iam attach-role-policy --policy-arn arn:aws:iam::aws:policy/
AmazonEC2ContainerRegistryReadOnly \
 --role-name $node_role_name
```

viii. IAM ロールの ARN を取得し、後のステップで使用するために変数に格納します。

```
node_iam_role=$(aws iam get-role --role-name $node_role_name --query="Role.Arn" --
output text)
```

g. マネージド型ノードグループを作成する

i. 前のステップで作成したサブネットの ID を表示します。

```
echo $subnets
```

出力例は次のとおりです。

```
subnet-0a1a56c486EXAMPLE, subnet-099e6ca77aEXAMPLE, subnet-0377963d69EXAMPLE, subnet-0c05f819
```

ii. ノードグループを作成しま

す。 **0a1a56c486EXAMPLE**、**099e6ca77aEXAMPLE**、**0377963d69EXAMPLE**、および **0c05f819d5EXAMPLE** は、前のステップの出力で返された値に置き換えます。次のコマンドでは、前の出力の中でサブネット ID の間にあるカンマを必ず削除して使用します。**t3.medium** の部分は、任意の [AWS Nitro System インスタンスタイプ](#) に置き換えることができます。

```
aws eks create-nodegroup --region $region_code --cluster-name $cluster_name --
nodegroup-name $nodegroup_name \
```

```
--subnets subnet-0a1a56c486EXAMPLE subnet-099e6ca77aEXAMPLE
subnet-0377963d69EXAMPLE subnet-0c05f819d5EXAMPLE \
--instance-types t3.medium --node-role $node_iam_role
```

ノードグループの作成には数分かかります。以下のコマンドを実行してください。出力が「ACTIVE」を返すまで、次のステップに進まないでください。

```
aws eks describe-nodegroup --region $region_code --cluster-name $cluster_name --
nodegroup-name $nodegroup_name \
 --query nodegroup.status --output text
```

#### h. デフォルトの Pod に IPv6 アドレスが割り当てられていることを、IP 列で確認します。

```
kubectl get pods -n kube-system -o wide
```

出力例は次のとおりです。

| NAME                                   | READY | STATUS  | RESTARTS | AGE   | IP                                             | NOMINATED NODE |
|----------------------------------------|-------|---------|----------|-------|------------------------------------------------|----------------|
| READINESS GATES                        |       |         |          |       |                                                |                |
| aws-node-rslts                         | 1/1   | Running | 1        | 5m36s |                                                |                |
| 2600:1f13:b66:8200:11a5:ade0:c590:6ac8 |       |         |          |       | ip-192-168-34-75.region-code.compute.internal  | <none>         |
| aws-node-t74jh                         | 1/1   | Running | 0        | 5m32s |                                                |                |
| 2600:1f13:b66:8203:4516:2080:8ced:1ca9 |       |         |          |       | ip-192-168-253-70.region-code.compute.internal | <none>         |
| coredns-85d5b4454c-cw7w2               | 1/1   | Running | 0        | 56m   |                                                |                |
| 2600:1f13:b66:8203:34e5::              |       |         |          |       | ip-192-168-253-70.region-code.compute.internal | <none>         |
| coredns-85d5b4454c-tx6n8               | 1/1   | Running | 0        | 56m   |                                                |                |
| 2600:1f13:b66:8203:34e5::1             |       |         |          |       | ip-192-168-253-70.region-code.compute.internal | <none>         |
| kube-proxy-btpbk                       | 1/1   | Running | 0        | 5m36s |                                                |                |
| 2600:1f13:b66:8200:11a5:ade0:c590:6ac8 |       |         |          |       | ip-192-168-34-75.region-code.compute.internal  | <none>         |
| kube-proxy-jjk2g                       | 1/1   | Running | 0        | 5m33s |                                                |                |
| 2600:1f13:b66:8203:4516:2080:8ced:1ca9 |       |         |          |       | ip-192-168-253-70.region-code.compute.internal | <none>         |

i. デフォルトのサービスに IPv6 アドレスが割り当てられていることを、IP 列で確認します。

```
kubectl get services -n kube-system -o wide
```

出力例は次のとおりです。

| NAME         | TYPE      | CLUSTER-IP        | EXTERNAL-IP | PORT(S)       | AGE |      |
|--------------|-----------|-------------------|-------------|---------------|-----|------|
| SELECTOR     |           |                   |             |               |     |      |
| kube-dns     | ClusterIP | fd30:3087:b6c2::a | <none>      | 53/UDP,53/TCP | 57m | k8s- |
| app=kube-dns |           |                   |             |               |     |      |

- j. (オプション) [サンプルアプリケーションをデプロイする](#)か、[AWS Load Balancer コントローラーとサンプルアプリケーションをデプロイ](#)して、「[the section called “アプリケーション負荷分散”](#)」に従い HTTP アプリケーションをロードバランシングするか、「[the section called “ネットワーク負荷分散”](#)」に従いネットワークトラフィックを IPv6 Pod にロードバランシングします。
- k. このチュートリアルのために作成したクラスターとノードの使用が終了したら、以下のコマンドにより、作成したリソースのクリーンアップを行う必要があります。削除を行う前に、これらのリソースが、このチュートリアルの外部で使用されていないことを確認します。
- i. ここまでの手順を実行したのとは異なるシェルで、このステップを完了する場合は、#### を前の手順を実行した際に指定した値に置き換えながら、すべての変数を前の手順で使用した値に設定します。前の手順を実行したのと同じシェルでこのステップを完了する場合は、このまま次のステップに進みます。

```
export region_code=region-code
export vpc_stack_name=my-eks-ipv6-vpc
export cluster_name=my-cluster
export nodegroup_name=my-nodegroup
export account_id=111122223333
export node_role_name=AmazonEKSNodeRole
export cluster_role_name=myAmazonEKSClusterRole
```

- ii. ノードグループを削除します。

```
aws eks delete-nodegroup --region $region_code --cluster-name $cluster_name --
nodegroup-name $nodegroup_name
```

削除には数分かかります。以下のコマンドを実行してください。何らかの出力が返された場合は、次のステップに進まないでください。

```
aws eks list-nodegroups --region $region_code --cluster-name $cluster_name --query
nodegroups --output text
```

iii. クラスターを削除します。

```
aws eks delete-cluster --region $region_code --name $cluster_name
```

クラスターの削除には数分かかります。次に進む前に、下記のコマンドを使用してクラスターが削除されていることを確認します。

```
aws eks describe-cluster --region $region_code --name $cluster_name
```

下記のような出力が返されるまで、次のステップに進まないでください。

```
An error occurred (ResourceNotFoundException) when calling the DescribeCluster
operation: No cluster found for name: my-cluster.
```

iv. 作成した IAM リソースを削除します。前の手順で使った名前とは異なる名前を選択した場合は、その名で *AmazonEKS\_CNI\_IPv6\_Policy* を置き換えます。

```
aws iam detach-role-policy --role-name $cluster_role_name --policy-arn
arn:aws:iam::aws:policy/AmazonEKSClusterPolicy
aws iam detach-role-policy --role-name $node_role_name --policy-arn
arn:aws:iam::aws:policy/AmazonEKSWorkerNodePolicy
aws iam detach-role-policy --role-name $node_role_name --policy-arn
arn:aws:iam::aws:policy/AmazonEC2ContainerRegistryReadOnly
aws iam detach-role-policy --role-name $node_role_name --policy-arn arn:aws:iam::
$account_id:policy/AmazonEKS_CNI_IPv6_Policy
aws iam delete-policy --policy-arn arn:aws:iam::$account_id:policy/
AmazonEKS_CNI_IPv6_Policy
aws iam delete-role --role-name $cluster_role_name
aws iam delete-role --role-name $node_role_name
```

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

```
aws cloudformation delete-stack --region $region_code --stack-name $vpc_stack_name
```

## Pod のアウトバウンドインターネットアクセスを有効にする

適用対象: Linux IPv4 Fargate ノード、Amazon EC2 インスタンスを持つ Linux ノード

IPv6 ファミリーを使用してクラスターをデプロイした場合、このトピックの情報はクラスターに適用されません。IPv6 アドレスがネットワーク変換されないからです。クラスターでの IPv6 使用の詳細については「[the section called “IPv6”](#)」を参照してください。

デフォルトでは、クラスター内の各 Pod に、Pod がデプロイされた VPC に関連付けられた Classless Inter-Domain Routing (CIDR) ブロックから、[プライベート](#) IPv4 アドレスが割り当てられます。同じ VPC 内の Pod は、これらのプライベート IP アドレスをエンドポイントとして使用して相互に通信します。VPC に関連付けられている CIDR ブロック外の IPv4 アドレスと Pod が通信する場合、([Linux](#) と [Windows](#) 両用の) Amazon VPC CNI プラグインが Pod の IPv4 アドレスを、Pod が実行されているノードのプライマリ [Elastic Network Interface](#) のプライマリプライベート IPv4 アドレスに変換します (デフォルトでは [\\*](#))。

### Note

Windows ノードについては、他にも考慮すべき事項があります。デフォルトでは、[Windows の VPC CNI プラグイン](#)は、同じ VPC 内の宛先へのトラフィックを SNAT から除外するネットワーク設定で定義されます。つまり、内部 VPC 通信では SNAT が無効になっており、Pod に割り当てられた IP アドレスは VPC 内でルーティング可能です。ただし、VPC 外の宛先へのトラフィックでは、送信元 Pod IP がインスタンスの ENI のプライマリ IP アドレスに SNAT されます。Windows のこのデフォルト設定により、Pod はホストインスタンスと同じ方法で VPC 外部のネットワークにアクセスできます。

この動作によって、次の現象が起こります。

- Pod は、それらが実行されているノードに[パブリック](#) IP アドレスまたは [Elastic](#) IP アドレスが割り当てられており、[パブリックサブネット](#)内にある場合にのみ、インターネットリソースと通信できます。パブリックサブネットに関連付けられている[ルートテーブル](#)には、インターネットゲートウェイへのルートが含まれています。可能であれば、プライベートサブネットにノードをデプロイすることをお勧めします。
- 1.8.0 よりも古いバージョンのプラグインの場合、[VPC ピアリング](#)、[トランジット VPC](#)、または [AWS Direct Connect](#) を使用してクラスター VPC に接続されているネットワークまたは VPC 内にあるリソースは、セカンダリ Elastic Network Interface の背後にある Pod との通信を開始できません。ただし、Pod はこれらのリソースとの通信を開始し、リソースから応答を受け取ることができます。

ご使用の環境で次のいずれかが当てはまる場合は、次のコマンドを使用してデフォルト設定を変更してください。

- [VPC ピアリング](#)、[トランジット VPC](#)、または [AWS Direct Connect](#) を使用してクラスター VPC に接続されているネットワークまたは VPC 内に、IPv4 アドレスを使用して Pod との通信を開始する必要があるリソースがあり、プラグインのバージョンが 1.8.0 よりも前である。
- Pod が [プライベートサブネット](#) にあり、インターネットへのアウトバウンド通信を行う必要がある。サブネットには、[NAT ゲートウェイ](#) へのルートがある。

```
kubectl set env daemonset -n kube-system aws-node AWS_VPC_K8S_CNI_EXTERNALSNAT=true
```

#### Note

AWS\_VPC\_K8S\_CNI\_EXTERNALSNAT および AWS\_VPC\_K8S\_CNI\_EXCLUDE\_SNAT\_CIDRS CNI の設定変数は Windows ノードに適用されません。SNAT の無効化は Windows でサポートされていません。IPv4 CIDR のリストを SNAT から除外する場合は、Windows ブートストラップスクリプトで ExcludedSnatCIDRs パラメータを指定して定義できます。このパラメータの使用に関する詳細については、「[the section called “ブートストラップスクリプトの設定パラメータ”](#)」を参照してください。

## ホストネットワークキング

\* Pod の仕様に `hostNetwork=true` が含まれている場合 (デフォルトは `false`)、その IP アドレスは別のアドレスに変換されません。これは、デフォルトでクラスターで実行される kube-proxy および Amazon VPC CNI plugin for Kubernetes Pod の場合に当てはまります。これらの Pod の場合、IP アドレスはノードのプライマリ IP アドレスと同じであるため、Pod の IP アドレスは変換されません。Pod の `hostNetwork` 設定の詳細については、Kubernetes API リファレンスの「[PodSpec v1 core](#)」を参照してください。

## Kubernetes ネットワークポリシーにより Pod トラフィックを制限する

デフォルトでは、Kubernetes の IP アドレス、ポート、クラスター内の Pod 間の接続、またはポッドと他のネットワークのリソースの接続に制限はありません。Kubernetes ネットワークポリシーを使用して、自分の Pod 間で送受信されるネットワークトラフィックを制限できます。詳細については、Kubernetes ドキュメントの「[ネットワークポリシー](#)」を参照してください。



クラスターの Amazon VPC CNI plugin for Kubernetes のバージョンが 1.13 以前の場合、Kubernetes ネットワークポリシーをクラスターに適用するには、サードパーティソリューションを実装する必要があります。バージョン 1.14 以降のプラグインではネットワークポリシーを実装できるため、サードパーティソリューションを使用する必要はありません。このトピックでは、サードパーティのアドオンを使用せずにクラスターの Kubernetes ネットワークポリシーを使用するようにクラスターを構成する方法を説明します。

Amazon VPC CNI plugin for Kubernetes のネットワークポリシーは、次の構成でサポートされています。

- クラスター上の Amazon VPC CNI plugin for Kubernetes のバージョン 1.14 以降。
- IPv4 または IPv6 アドレス用に設定されたクラスター。
- [ポッド用のセキュリティグループ](#)でネットワークポリシーを使用できます。ネットワークポリシーを使用すると、クラスター内の通信をすべて制御できます。Pod 用のセキュリティグループを使用すると、Pod 内のアプリケーションから AWS サービスへのアクセスを制御できます。
- カスタムネットワークおよびプレフィクス委任でネットワークポリシーを使用できます。

## 考慮事項

### アーキテクチャ

- Amazon VPC CNI plugin for Kubernetes を含むクラスターに Amazon VPC CNI plugin for Kubernetes ネットワークポリシーを適用する場合、Amazon EC2 Linux ノードにのみポリシーを適用できます。Fargate または Windows ノードにはポリシーを適用できません。
- ネットワークポリシーはIPv4 または IPv6 アドレスのいずれかのみを適用しますが、両方は適用しません。IPv4 クラスターではVPC CNI は IPv4 アドレスをポッドに割り当て、IPv4 ポリシーを適用します。IPv6 クラスターではVPC CNI は IPv6 アドレスをポッドに割り当て、IPv6 ポリシーを適用します。IPv6 クラスターに適用された IPv4 ネットワークポリシールールは無視されます。IPv4 クラスターに適用された IPv6 ネットワークポリシールールは無視されます。

### ネットワークポリシー

- ネットワークポリシーはデプロイの一部である Pod にのみ適用されます。metadata.ownerReferences セットを持たないスタンドアロンの Pod ではネットワークポリシーを適用できません。
- 同じ Pod に複数のネットワークポリシーを適用できます。同じ Pod を選択するポリシーが 2 つ以上設定されている場合、すべてのポリシーが Pod に適用されます。

- ネットワークポリシーの各 ingress: または egress: セレクターの各プロトコルにおけるポートの一意の組み合わせの最大数は 24 です。
- どの Kubernetes サービスでも、サービスポートはコンテナポートと同じでなければなりません。名前付きポートを使用している場合はサービス仕様でも同じ名前を使用してください。

## 移行

- クラスターが現在サードパーティーソリューションを使用して Kubernetes ネットワークポリシーを管理している場合、同じポリシーを Amazon VPC CNI plugin for Kubernetes で使用できます。ただし、同じポリシーを管理しないように、既存のソリューションを削除する必要があります。

## インストール

- ネットワークポリシー機能では `policyendpoints.networking.k8s.aws` と呼ばれる PolicyEndpoint カスタムリソース定義 (CRD が作成され、必要になります。カスタムリソースの PolicyEndpoint オブジェクトは Amazon EKS によって管理されます。これらのリソースを変更または削除しないでください。
- インスタンスロールの IAM 認証情報を使用するポッドを実行するか、EC2 IMDS に接続するポッドを実行する場合は EC2 IMDS へのアクセスをブロックするネットワークポリシーがないか慎重に確認してください。EC2 IMDS へのアクセスを許可するネットワークポリシーを追加する必要がある場合があります。詳細については「Amazon EC2 ユーザーガイド」の「[インスタンスメタデータとユーザーデータ](#)」を参照してください。

サービスアカウントの IAM ロールまたは EKS Pod Identity を使用するポッドは EC2 IMDS にアクセスしません。

- Amazon VPC CNI plugin for Kubernetes は、各ポッドの追加のネットワークインターフェイスにはネットワークポリシーを適用せず、各ポッドのプライマリインターフェイス (eth0) のみにネットワークポリシーを適用します。これは以下のアーキテクチャに影響します：
  - `ENABLE_V4_EGRESS` 変数が `true` に設定された IPv6 ポッド。この変数により、IPv4 エグレス機能が IPv6 ポッドをクラスター外のエンドポイントなどの IPv4 エンドポイントに接続できるようになります。IPv4 エグレス機能は、ローカルループバック IPv4 アドレスを持つ追加のネットワークインターフェイスを作成することで機能します。
  - Multus などのチェーンネットワークプラグインを使用する場合。これらのプラグインは各ポッドにネットワークインターフェイスを追加するため、ネットワークポリシーはチェーンネットワークプラグインには適用されません。

## Kubernetes ネットワークポリシーを使用して Pod ネットワークトラフィックを制限する

Kubernetes ネットワークポリシーを使用して、自分の Pod 間で送受信されるネットワークトラフィックを制限できます。詳細については、Kubernetes ドキュメントの「[ネットワークポリシー](#)」を参照してください。

この機能を使用するには以下を設定する必要があります：

1. Pod 起動時のポリシーの適用を設定します。これは VPC CNI DaemonSet の aws-node コンテナでできます。
2. アドオンのネットワークポリシーパラメータを有効にします。
3. Kubernetes ネットワークポリシーを使用するようにクラスターを設定します。

開始する前に、考慮事項を確認してください。詳細については「[the section called “考慮事項”](#)」を参照してください。

### 前提条件

この機能の前提条件は次のとおりです。

### 最小クラスターバージョン

既存の Amazon EKS クラスター。デプロイするには「[はじめに](#)」を参照してください。クラスターは、次の表に示す Kubernetes バージョンとプラットフォームバージョンのいずれかを実行している必要があります。一覧にあるバージョンより後の Kubernetes とプラットフォームのバージョンもサポートされることにご注意ください。現在の Kubernetes バージョンを確認するには、次のコマンドの *my-cluster* をクラスターの名前に置き換えて、変更したコマンドを実行します。

```
aws eks describe-cluster
 --name my-cluster --query cluster.version --output
 text
```

| Kubernetes バージョン | プラットフォームバージョン |
|------------------|---------------|
| 1.27.4           | eks.5         |
| 1.26.7           | eks.6         |

## 最小 VPC CNI バージョン

クラスター上の Amazon VPC CNI plugin for Kubernetes のバージョン 1.14 以降。現在のバージョンは、次のコマンドで確認できます。

```
kubectl describe daemonset aws-node --namespace kube-system | grep amazon-k8s-cni: |
cut -d : -f 3
```

バージョンが 1.14 より前の場合は、[the section called “更新 \(EKS アドオン\)”](#) を確認してバージョン 1.14 以降にアップグレードしてください。

## 最小 Linux カーネルバージョン

ノードにはバージョン 5.10 以降の Linux カーネルが必要です。カーネルのバージョンは `uname -r` で確認できます。Amazon EKS 最適化 Amazon Linux、Amazon EKS 最適化高速 Amazon Linux AMI、と Bottlerocket AMI の最新バージョンを使用している場合、既に必要なカーネルバージョンがインストールされています。

Amazon EKS 最適化高速 Amazon Linux AMI v20231116 バージョン以降には、カーネルバージョン 5.10 があります。

### ステップ 1: Pod 起動時のポリシーの適用を設定する

Amazon VPC CNI plugin for Kubernetes はポッドのプロビジョニングと並行して、ポッドのネットワークポリシーを設定します。新しいポッドにすべてのポリシーが設定されるまで、新しいポッドのコンテナはデフォルトの許可ポリシーで起動します。これは標準モードと呼ばれます。デフォルトの許可ポリシーはすべての ingress トラフィックと egress トラフィックが新しいポッドとの間で許可されることを意味します。例えば、新しいポッドがアクティブなポリシーで更新されるまで、ポッドにはファイアウォールルールが適用されません (すべてのトラフィックが許可されます)。

変数 `NETWORK_POLICY_ENFORCING_MODE` を `strict` に設定すると、VPC CNI を使用するポッドはデフォルトの拒否ポリシーで起動し、その後ポリシーが設定されます。これは Strict モードと呼ばれます。Strict モードではポッドがクラスター内でアクセスする必要があるすべてのエンドポイントにネットワークポリシーが必要です。この要件は CoreDNS ポッドに適用されることに注意してください。デフォルトの拒否ポリシーはホストネットワークを使用するポッドには設定されていません。

このデフォルトのネットワークポリシーは VPC CNI DaemonSet の `aws-node` コンテナで環境変数 `NETWORK_POLICY_ENFORCING_MODE` を `strict` に設定することで変更できます。

```
env:
```

```
- name: NETWORK_POLICY_ENFORCING_MODE
 value: "strict"
```

## ステップ 2: アドオンのネットワークポリシーパラメータを有効にする

ネットワークポリシー機能はデフォルトでノード上のポート 8162 をメトリクスに使用します。また、この機能はヘルスプローブにポート 8163 を使用していました。そのノード上、またはこれらのポートを使用する必要があるポッド内で他のアプリケーションを実行すると、そのアプリは実行できません。VPC CNI バージョン v1.14.1 以降ではこれらのポートを変更できます。

アドオンのネットワークポリシーパラメータを有効にするには次の手順に従います。

### AWS Management Console

1. [Amazon EKS コンソール](#)を開きます。
2. 左のナビゲーションペインで、[クラスター] を選択し、Amazon VPC CNI アドオンを設定するクラスターの名前を選択してください。
3. [アドオン] タブを選択してください。
4. アドオンボックスの右上にあるボックスを選択し、次に [編集] を選択してください。
5. [Amazon VPC CNI を設定] ページで、次の操作を行います：
  - a. [バージョン] リストで v1.14.0-eksbuild.3 以降のバージョンを選択してください。
  - b. [オプションの構成設定] を展開します。
  - c. [設定値] に JSON キー "enableNetworkPolicy": と値 "true" を入力します。結果のテキストは有効な JSON オブジェクトでなければなりません。このキーと値だけがテキストボックス内のデータである場合はキーと値を中括弧 { } で囲みます。

次の例ではネットワークポリシー機能が有効になっており、メトリクスとヘルスプローブがデフォルトのポート番号に設定されています：

```
{
 "enableNetworkPolicy": "true",
 "nodeAgent": {
 "healthProbeBindAddr": "8163",
 "metricsBindAddr": "8162"
 }
}
```

## Helm

helm を通して Amazon VPC CNI plugin for Kubernetes をインストールしている場合、設定を更新してポートを変更できます。

1. 次に、以下のコマンドを実行して、ポートを変更します。キー `nodeAgent.metricsBindAddr` または キー `nodeAgent.healthProbeBindAddr` の値にそれぞれポート番号を設定します。

```
helm upgrade --set nodeAgent.metricsBindAddr=8162 --set
nodeAgent.healthProbeBindAddr=8163 aws-vpc-cni --namespace kube-system eks/aws-vpc-
cni
```

## kubectl

1. エディターで `aws-nodeDaemonSet` を開きます。

```
kubectrl edit daemonset -n kube-system aws-node
```

2. VPC CNI `aws-node` デーモンセットマニフェストの `aws-network-policy-agent` コンテナで、以下の `args:` のコマンド引数でポート番号を置き換えます。

```
- args:
 - --metrics-bind-addr=:8162
 - --health-probe-bind-addr=:8163
```

ステップ 3: バークレーパケットフィルター (BPF) ファイルシステムを各ノードにマウントする

Berkeley Packet Filter (BPF ファイルシステムを各ノードにマウントする必要があります。

### Note

クラスターがバージョン 1.27 以降の場合、Amazon EKS 最適化された 1.27 以降の Amazon Linux AMI と Bottlerocket AMI にはすべてこの機能が既に実装されているため、このステップをスキップできます。

その他のすべてのクラスターバージョンでは Amazon EKS 最適化された Amazon Linux をバージョン v20230703 以降にアップグレードした場合、または Bottlerocket AMI をバージョン v1.0.2 にアップグレードした場合に。このステップをスキップできます。

1. バックレーパケットフィルタ (BPF) ファイルシステムを各ノードにマウントします。

```
sudo mount -t bpf bpffs /sys/fs/bpf
```

2. 次に、Amazon EC2 Auto Scaling グループの起動テンプレートのユーザーデータに同じコマンドを追加します。

ステップ 4: Kubernetes ネットワークポリシーを使用するようにクラスターを設定する

これはAmazon EKS アドオンまたはセルフマネージドアドオンに対して設定できます。

### Amazon EKS アドオン

AWS CLI を使用すると、次のコマンドを実行して、Kubernetes ネットワークポリシーを使用するようにクラスターを設定できます。my-cluster をクラスターの名前に置き換え、IAM ロール ARN を使用するロールに置き換えます。

```
aws eks update-addon --cluster-name my-cluster --addon-name vpc-cni --addon-version
v1.14.0-eksbuild.3 \
 --service-account-role-arn arn:aws:iam::123456789012:role/AmazonEKSVPCCNIRole \
 --resolve-conflicts PRESERVE --configuration-values '{"enableNetworkPolicy":
"true"}'
```

AWS マネジメントコンソールを使用してこれを設定するには、以下の手順に従います。

1. [Amazon EKS コンソール](#)を開きます。
2. 左のナビゲーションペインで、[クラスター] を選択し、Amazon VPC CNI アドオンを設定するクラスターの名前を選択してください。
3. [アドオン] タブを選択してください。
4. アドオンボックスの右上にあるボックスを選択し、次に [編集] を選択してください。
5. [Amazon VPC CNI を設定] ページで、次の操作を行います：
  - a. [バージョン] リストで v1.14.0-eksbuild.3 以降のバージョンを選択してください。
  - b. [オプションの構成設定] を展開します。
  - c. [設定値] に JSON キー "enableNetworkPolicy": と値 "true" を入力します。結果のテキストは有効な JSON オブジェクトでなければなりません。このキーと値だけがテキストボックス内のデータである場合はキーと値を中括弧 { } で囲みます。次の例はネットワークポリシーが有効になっていることを示しています：

```
{ "enableNetworkPolicy": "true" }
```

次のスクリーンショットはこのシナリオの例を示しています。



[EKS](#) > [Clusters](#) > > [Add-on](#) > [vpc-cni](#) > Edit add-on

## Configure Amazon VPC CNI

### Amazon VPC CNI [Info](#)

Listed by



Category

networking

Status

Active

#### Version

Select the version for this add-on.

v1.17.1-eksbuild.1

#### Select IAM role

Select an IAM role to use with this add-on. To create a new role, follow the instructions in the [Amazon EKS User Guide](#).

#### Optional configuration settings

##### Add-on configuration schema

Refer to the JSON schema below. The configuration values entered in the code editor will be validated against this schema.

```
{
 "$ref": "#/definitions/VpcCni",
 "$schema": "http://json-schema.org/draft-06/schema#",
 "definitions": {
 "Affinity": {
 "type": [
 "object",
 "null"
]
 },
 "EniConfig": {
 "additionalProperties": false,

```

#### Configuration values [Info](#)

Specify any additional JSON or YAML configurations that should be applied to the add-on.

|   |                                   |
|---|-----------------------------------|
| 1 | { "enableNetworkPolicy": "true" } |
|---|-----------------------------------|

## セルフマネージド型アドオン

### Helm

helm を通して Amazon VPC CNI plugin for Kubernetes をインストールしている場合、設定を更新してネットワークポリシーを有効にできます。

1. 次のコマンドを実行してネットワークポリシーを有効にします。

```
helm upgrade --set enableNetworkPolicy=true aws-vpc-cni --namespace kube-system eks/
aws-vpc-cni
```

### kubectl

1. エディターで amazon-vpc-cniConfigMap を開きます。

```
kubectl edit configmap -n kube-system amazon-vpc-cni -o yaml
```

2. 次の行を ConfigMap のdataに追加します。

```
enable-network-policy-controller: "true"
```

行を追加すると、ConfigMap は次の例のようになります。

```
apiVersion: v1
kind: ConfigMap
metadata:
 name: amazon-vpc-cni
 namespace: kube-system
data:
 enable-network-policy-controller: "true"
```

3. エディターで aws-nodeDaemonSet を開きます。

```
kubectl edit daemonset -n kube-system aws-node
```

- a. VPC CNI aws-node デーモンセットマニフェストの aws-network-policy-agent コンテナで、args: のコマンド引数 --enable-network-policy=false の false を true に置き換えます。

```
- args:
 - --enable-network-policy=true
```

## ステップ 5. 次のステップ

設定が完了したら、aws-node ポッドがクラスターで実行されていることを確認します。

```
kubectl get pods -n kube-system | grep 'aws-node\|amazon'
```

出力例は次のとおりです。

|                       |     |         |             |
|-----------------------|-----|---------|-------------|
| aws-node-gmqp7<br>24h | 2/2 | Running | 1 (24h ago) |
| aws-node-prnsh<br>24h | 2/2 | Running | 1 (24h ago) |

バージョン 1.14 以降の aws-node ポッドには 2 つのコンテナがあります。以前のバージョンではネットワークポリシーが無効になっている場合、aws-node ポッドにコンテナは 1 つしか存在しません。

これで、Kubernetes ネットワークポリシーをクラスターにデプロイできます。

Kubernetes ネットワークポリシーを実装するには、Kubernetes NetworkPolicy オブジェクトを作成して、クラスターにデプロイします。NetworkPolicy オブジェクトは名前空間に限定されます。ラベルセレクター、名前空間、および IP アドレス範囲に基づいて、Pod 間のトラフィックを許可または拒否するポリシーを実装します。NetworkPolicy オブジェクトの作成の詳細については Kubernetes ドキュメントの「[ネットワークポリシー](#)」を参照してください。

Kubernetes NetworkPolicy オブジェクトの適用は、Extended Berkeley Packet Filter (eBPF) を使用して実装されます。iptables をベースにした実装に関連して、CPU 使用率の低下やシーケンシャルルックアップの回避など、低レイテンシーを実現し、パフォーマンス特性を発揮します。さらに、eBPF プローブは、複雑なカーネルレベルの問題のデバッグやオブザーバビリティの向上に役立つ、コンテキスト豊富なデータへのアクセスを提供します。Amazon EKS は、プロブを利用して各ノードのポリシー結果をログに記録し、そのデータを外部のログコレクターにエクスポートしてデバッグに役立てる eBPF ベースのエクスポーターをサポートしています。詳細については「[eBPF ドキュメント](#)」を参照してください。

## Amazon EKS Pod ネットワークトラフィックの Kubernetes ネットワークポリシーを無効にする

Kubernetes ネットワークポリシーを無効にして Amazon EKS Pod ネットワークトラフィックの制限を停止する

1. すべての Kubernetes ネットワークポリシーを一覧表示します。

```
kubectl get netpol -A
```

2. 各 Kubernetes ネットワークポリシーを削除します。ネットワークポリシーを無効にする前に、すべてのネットワークポリシーを削除する必要があります。

```
kubectl delete netpol <policy-name>
```

3. エディタで aws-node DaemonSet を開きます。

```
kubectl edit daemonset -n kube-system aws-node
```

4. VPC CNI aws-node デーモンセットマニフェストの aws-network-policy-agent コンテナで、args: のコマンド引数 --enable-network-policy=true の true を false に置き換えます。

```
- args:
 - --enable-network-policy=false
```

## Amazon EKS の Kubernetes ネットワークポリシーのトラブルシューティング

[ネットワークポリシーのログ](#)を読み、[eBPF SDK](#) のツールを実行することにより、ネットワークポリシーを使用するネットワーク接続をトラブルシューティングおよび調査できます。

### ネットワークポリシーのログ

ネットワークポリシーによって接続が許可されているか拒否されているかはフローログに記録されています。各ノードのネットワークポリシーログにはネットワークポリシーが設定されているすべてのポッドのフローログが含まれます。ネットワークポリシーログは /var/log/aws-routed-eni/network-policy-agent.log に保存されます。次の例は network-policy-agent.log ファイルからのものです。

```
{"level":"info","timestamp":"2023-05-30T16:05:32.573Z","logger":"ebpf-client","msg":"Flow Info: ","Src
```

```
IP":"192.168.87.155","Src Port":38971,"Dest IP":"64.6.160","Dest Port":53,"Proto":"UDP","Verdict":"ACCEPT"}
```

ネットワークポリシーログはデフォルトで無効になっています。ネットワークポリシーログを有効にするには次の手順に従います。

 Note

ネットワークポリシーログにはVPC CNI aws-node デーモンセットマニフェストの aws-network-policy-agent コンテナ用に 1 つの vCPU を追加する必要があります。

## Amazon EKS アドオン

### AWS Management Console

- a. [Amazon EKS コンソール](#)を開きます。
- b. 左のナビゲーションペインで、[クラスター] を選択し、Amazon VPC CNI アドオンを設定するクラスターの名前を選択してください。
- c. [アドオン] タブを選択してください。
- d. アドオンボックスの右上にあるボックスを選択し、次に [編集] を選択してください。
- e. **Amazon VPC CNI** の設定ページで、次の手順に従います。
  - i. [バージョン] ドロップダウンリストで v1.14.0-eksbuild.3 以降のバージョンを選択してください。
  - ii. [オプションの構成設定] を展開します。
  - iii. 最上位の JSON キー "nodeAgent": を入力してください。値は [設定値] に  
キー "enablePolicyEventLogs": と "true" の値を持つオブジェクトです。結果のテキストは有効な JSON オブジェクトでなければなりません。次の例はネットワークポリシーとネットワークポリシーログが有効になっており、ネットワークポリシーログが CloudWatch Logs に送信されていることを示しています。

```
{
 "enableNetworkPolicy": "true",
 "nodeAgent": {
 "enablePolicyEventLogs": "true"
 }
}
```

次のスクリーンショットはこのシナリオの例を示しています。



## Configure Amazon VPC CNI

## Amazon VPC CNI [Info](#)



networking

🟢 Active

Select the version for this add-on.

v1.17.1-eksbuild.1

Select an IAM role to use with this add-on. To create a new role, follow the instructions in the [Amazon EKS User Guide](#).

### ▼ Optional configuration settings

### Add-on configuration schema

Refer to the JSON schema below. The configuration values entered in the code editor will be validated against this schema.

```
{
 "$ref": "#/definitions/VpcCni",
 "$schema": "http://json-schema.org/draft-06/schema#",
 "definitions": {
 "Affinity": {
 "type": [
 "object",
 "null"
]
 },
 "EniConfig": {
 "additionalProperties": false,

```

Configuration values [Info](#)

Specify any additional JSON or YAML configurations that should be applied to the add-on.

```
1 {
2 "enableNetworkPolicy": "true",
3 "nodeAgent": {
4 "enablePolicyEventLogs": "true"
5 }
6 }
```

## AWS CLI

- a. 次の AWS CLI コマンドを実行してください。my-cluster をクラスターの名前に置き換え、IAM 役割 ARN を使用する役割に置き換えます。

```
aws eks update-addon --cluster-name my-cluster --addon-name vpc-cni --addon-version v1.14.0-eksbuild.3 \
 --service-account-role-arn arn:aws:iam::123456789012:role/AmazonEKSVPCCNIRole \
 --resolve-conflicts PRESERVE --configuration-values '{"nodeAgent": {"enablePolicyEventLogs": "true"}}'
```

## セルフマネージド型アドオン

### Helm

helm を通して Amazon VPC CNI plugin for Kubernetes をインストールしている場合、設定を更新してネットワークポリシーログを記述できます。

- a. 次のコマンドを実行してネットワークポリシーを有効にします。

```
helm upgrade --set nodeAgent.enablePolicyEventLogs=true aws-vpc-cni --namespace kube-system eks/aws-vpc-cni
```

### kubectl

kubectl を通して Amazon VPC CNI plugin for Kubernetes をインストールしている場合、設定を更新してネットワークポリシーログを記述できます。

- a. エディターで aws-nodeDaemonSet を開きます。

```
kubectl edit daemonset -n kube-system aws-node
```

- b. VPC CNI aws-node デーモンセットマニフェストの aws-network-policy-agent コンテナで、args: のコマンド引数 --enable-policy-event-logs=false の false を true に置き換えます。

```
- args:
 - --enable-policy-event-logs=true
```



## ネットワークポリシーログを Amazon CloudWatch Logs に送信する

Amazon CloudWatch Logs などのサービスを使用して、ネットワークポリシーログをモニタリングできます。次の方法を使用して、ネットワークポリシーログを CloudWatch Logs に送信できます。

EKS クラスターの場合、ポリシー ログは `/aws/eks/cluster-name/cluster/` の下に配置され、セルフマネージド K8S クラスターの場合、ログは `/aws/k8s-cluster/cluster/` の下に配置されます。

### Amazon VPC CNI plugin for Kubernetes によるネットワークポリシーログの送信

ネットワークポリシーを有効にすると、2 つ目のコンテナがノードエージェントの `aws-node` ポッドに追加されます。このノードエージェントはネットワークポリシーログを CloudWatch Logs に送信できます。

#### Note

ノードエージェントはネットワークポリシーログのみを送信します。VPC CNI によって作成された他のログは含まれません。

### 前提条件

- VPC CNI に使用している IAM 役割に、次の権限をスタンザまたは個別のポリシーとして追加します。

```
{
 "Version": "2012-10-17",
 "Statement": [
 {
 "Sid": "VisualEditor0",
 "Effect": "Allow",
 "Action": [
 "logs:DescribeLogGroups",
 "logs:CreateLogGroup",
 "logs:CreateLogStream",
 "logs:PutLogEvents"
],
 "Resource": "*"
 }
]
}
```

```
}
```

## Amazon EKS アドオン

### AWS Management Console

- a. [Amazon EKS コンソール](#)を開きます。
- b. 左のナビゲーションペインで、[クラスター] を選択し、Amazon VPC CNI アドオンを設定するクラスターの名前を選択してください。
- c. [アドオン] タブを選択してください。
- d. アドオンボックスの右上にあるボックスを選択し、次に [編集] を選択してください。
- e. **Amazon VPC CNI** の設定ページで、次の手順に従います。
  - i. [バージョン] ドロップダウンリストで v1.14.0-eksbuild.3 以降のバージョンを選択してください。
  - ii. [オプションの構成設定] を展開します。
  - iii. 最上位の JSON キー "nodeAgent": を入力してください。値は [設定値] にキー "enableCloudWatchLogs": と "true" の値を持つオブジェクトです。結果のテキストは有効な JSON オブジェクトでなければなりません。次の例はネットワークポリシーとネットワークポリシーログが有効になっており、ログが CloudWatch Logs に送信されていることを示しています。

```
{
 "enableNetworkPolicy": "true",
 "nodeAgent": {
 "enablePolicyEventLogs": "true",
 "enableCloudWatchLogs": "true",
 }
}
```

次のスクリーンショットはこのシナリオの例を示しています。

## Amazon VPC CNI [Info](#)



networking

Active

Select the version for this add-on.

v1.17.1-eksbuild.1

Select an IAM role to use with this add-on. To create a new role, follow the instructions in the [Amazon EKS User Guide](#).

### Add-on configuration schema

Refer to the JSON schema below. The configuration values entered in the code editor will be validated against this schema.

```
{
 "$ref": "#/definitions/VpcCni",
 "$schema": "http://json-schema.org/draft-06/schema#",
 "definitions": {
 "Affinity": {
 "type": [
 "object",
 "null"
]
 },
 "EniConfig": {
 "additionalProperties": false,

```

Specify any additional JSON or YAML configurations that should be applied to the add-on.

```
1 {
2 "enableNetworkPolicy": "true",
3 "nodeAgent": {
4 "enablePolicyEventLogs": "true",
5 "enableCloudWatchLogs": "true"
6 }
7 }
```

## AWS CLI

- a. 次の AWS CLI コマンドを実行してください。my-cluster をクラスターの名前に置き換え、IAM 役割 ARN を使用する役割に置き換えます。

```
aws eks update-addon --cluster-name my-cluster --addon-name vpc-cni --addon-version v1.14.0-eksbuild.3 \
 --service-account-role-arn arn:aws:iam::123456789012:role/AmazonEKSVPCCNIRole \
 --resolve-conflicts PRESERVE --configuration-values '{"nodeAgent": {"enablePolicyEventLogs": "true", "enableCloudWatchLogs": "true"}}'
```

## セルフマネージド型アドオン

### Helm

helm を通して Amazon VPC CNI plugin for Kubernetes をインストールしている場合、設定を更新してネットワークポリシーログを CloudWatch Logs に送信できます。

- a. 次のコマンドを実行してネットワークポリシーログを有効にし、CloudWatch Logs に送信します。

```
helm upgrade --set nodeAgent.enablePolicyEventLogs=true --set nodeAgent.enableCloudWatchLogs=true aws-vpc-cni --namespace kube-system eks/aws-vpc-cni
```

### kubectl

- a. エディターで aws-nodeDaemonSet を開きます。

```
kubectl edit daemonset -n kube-system aws-node
```

- b. VPC CNI aws-node デーモンセットマニフェストの aws-network-policy-agent コンテナで、args: の 2 つのコマンド引数 --enable-policy-event-logs=false と --enable-cloudwatch-logs=false の false を true に置き換えます。

```
- args:
 - --enable-policy-event-logs=true
 - --enable-cloudwatch-logs=true
```

## ネットワークポリシーログを Fluent Bit デモンセットと共に送信する

デモンセット内で Fluent Bit を使用してノードからログを送信する場合は、ネットワークポリシーのネットワークポリシーログを含めるように設定を追加できます。次の設定例を使用できます。

```
[INPUT]
 Name tail
 Tag eksnp.*
 Path /var/log/aws-routed-eni/network-policy-agent*.log
 Parser json
 DB /var/log/aws-routed-eni/flb_npagent.db
 Mem_Buf_Limit 5MB
 Skip_Long_Lines On
 Refresh_Interval 10
```

## eBPF SDK の内容

Amazon VPC CNI plugin for Kubernetes は、複数のツールをまとめた eBPF SDK コレクションをノードにインストールします。eBPF SDK ツールを使用して、ネットワークポリシーの問題を特定できます。例えば、次のコマンドはノードで実行されているプログラムを一覧表示します。

```
sudo /opt/cni/bin/aws-eks-na-cli ebpf progs
```

このコマンドを実行するために、任意の方法を使用してノードに接続できます。

## Amazon EKS のネットワークポリシーの星型デモ

このデモでは、Amazon EKS クラスターにフロントエンド、バックエンド、およびクライアントサービスを作成します。また、このデモでは、各サービス間で利用可能な入力および出力のパスを示す管理グラフィカルユーザーインターフェースが作成されます。実稼働ワークロードを実行しないクラスターでデモを完了することをお勧めします。

ネットワークポリシーを作成する前に、サービスはすべて、双方向に通信できます。ネットワークポリシーを適用すると、クライアントはフロントエンドサービスとのみ通信でき、バックエンドはフロントエンドからのみトラフィックを受け付けます。

1. フロントエンド、バックエンド、クライアント、および管理ユーザーインターフェースサービスを適用します。

```
kubectl apply -f https://raw.githubusercontent.com/aws-samples/eks-workshop/2f9d29ed3f82ed6b083649e975a0e574fb8a4058/content/beginner/120_network-policies/calico/stars_policy_demo/create_resources.files/namespace.yaml
kubectl apply -f https://raw.githubusercontent.com/aws-samples/eks-workshop/2f9d29ed3f82ed6b083649e975a0e574fb8a4058/content/beginner/120_network-policies/calico/stars_policy_demo/create_resources.files/management-ui.yaml
kubectl apply -f https://raw.githubusercontent.com/aws-samples/eks-workshop/2f9d29ed3f82ed6b083649e975a0e574fb8a4058/content/beginner/120_network-policies/calico/stars_policy_demo/create_resources.files/backend.yaml
kubectl apply -f https://raw.githubusercontent.com/aws-samples/eks-workshop/2f9d29ed3f82ed6b083649e975a0e574fb8a4058/content/beginner/120_network-policies/calico/stars_policy_demo/create_resources.files/frontend.yaml
kubectl apply -f https://raw.githubusercontent.com/aws-samples/eks-workshop/2f9d29ed3f82ed6b083649e975a0e574fb8a4058/content/beginner/120_network-policies/calico/stars_policy_demo/create_resources.files/client.yaml
```

## 2. クラスター上の Pod をすべて表示します。

```
kubectl get pods -A
```

出力例は次のとおりです。

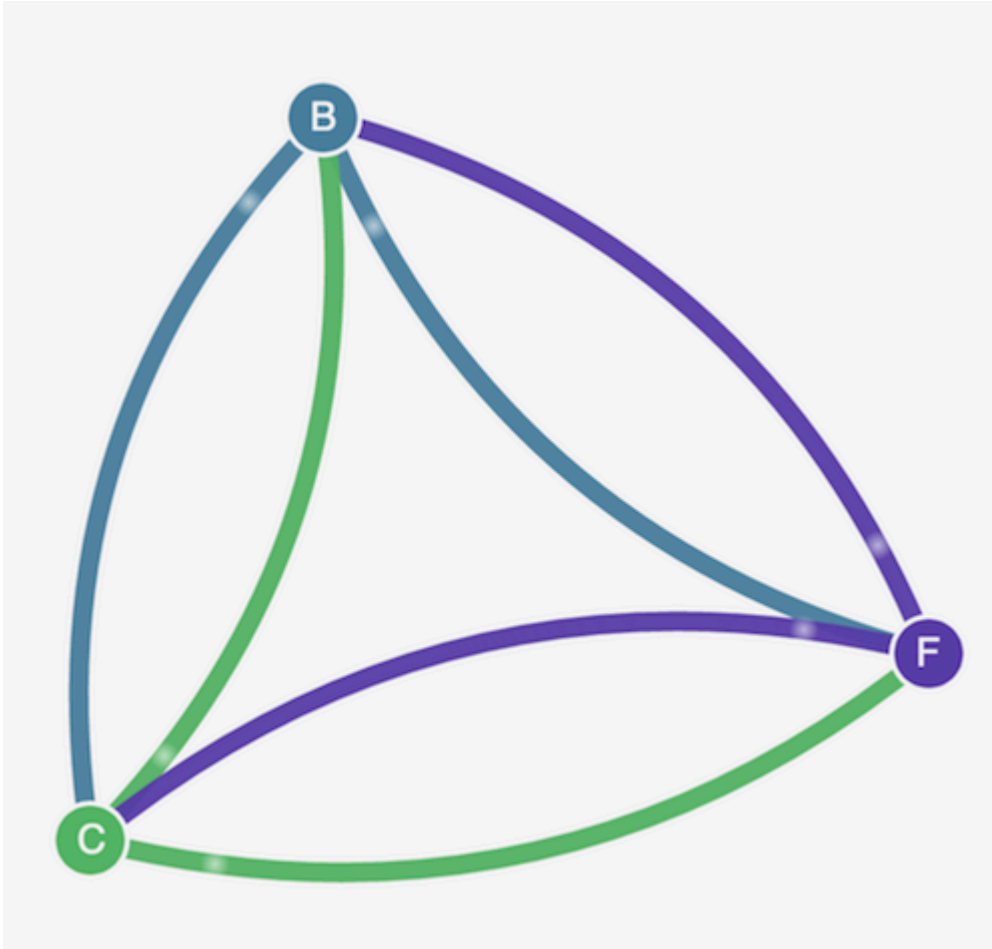
出力では、次の出力に示す名前空間にポッドが表示されます。READY 列内のポッドの **NAMES** とポッド数は、次の出力とは異なります。似たような名前のポッドが見えて、それらがすべて STATUS 列内に Running を持つまで続けないでください。

| NAMESPACE     | NAME                | READY | STATUS    |
|---------------|---------------------|-------|-----------|
| RESTARTS      | AGE                 |       |           |
| [...]         |                     |       |           |
| client        | client-xlffc        | 1/1   | Running 0 |
| 5m19s         |                     |       |           |
| [...]         |                     |       |           |
| management-ui | management-ui-qrb2g | 1/1   | Running 0 |
| 5m24s         |                     |       |           |
| stars         | backend-sz87q       | 1/1   | Running 0 |
| 5m23s         |                     |       |           |
| stars         | frontend-cscnf      | 1/1   | Running 0 |
| 5m21s         |                     |       |           |
| [...]         |                     |       |           |

3. 管理ユーザーインターフェイスに接続するには、クラスターで実行されているサービスの EXTERNAL-IP に接続します。

```
kubectl get service/management-ui -n management-ui
```

4. ブラウザーを開いて、前のステップの場所に移動します。管理ユーザーインターフェイスが表示されます。[C] ノードはクライアントサービス、[F] ノードはフロントエンドサービス、[B] ノードはバックエンドサービスを表します。各ノードには、太字の色付きの行で示されるように、他のすべてのノードへの完全な通信アクセスが含まれます。



5. 相互にサービスを分離するには、次のネットワークポリシーを `stars` および `client` の名前空間の両方に適用します。

```
kind: NetworkPolicy
apiVersion: networking.k8s.io/v1
metadata:
 name: default-deny
spec:
```

```
podSelector:
 matchLabels: {}
```

次のコマンドを使用して、ポリシーを両方の名前空間に適用できます。

```
kubectl apply -n stars -f https://raw.githubusercontent.com/aws-samples/eks-workshop/2f9d29ed3f82ed6b083649e975a0e574fb8a4058/content/beginner/120_network-policies/calico/stars_policy_demo/apply_network_policies.files/default-deny.yaml
kubectl apply -n client -f https://raw.githubusercontent.com/aws-samples/eks-workshop/2f9d29ed3f82ed6b083649e975a0e574fb8a4058/content/beginner/120_network-policies/calico/stars_policy_demo/apply_network_policies.files/default-deny.yaml
```

6. ブラウザを更新します。管理ユーザーインターフェイスはノードに到達できなくなるため、ユーザーインターフェイスに表示されないことがわかります。
7. 管理ユーザーインターフェイスがサービスにアクセスできるように、次の別のネットワークポリシーを適用します。このポリシーを適用して UI を許可します。

```
kind: NetworkPolicy
apiVersion: networking.k8s.io/v1
metadata:
 namespace: stars
 name: allow-ui
spec:
 podSelector:
 matchLabels: {}
 ingress:
 - from:
 - namespaceSelector:
 matchLabels:
 role: management-ui
```

このポリシーを適用してクライアントを許可します。

```
kind: NetworkPolicy
apiVersion: networking.k8s.io/v1
metadata:
 namespace: client
 name: allow-ui
spec:
 podSelector:
 matchLabels: {}
```

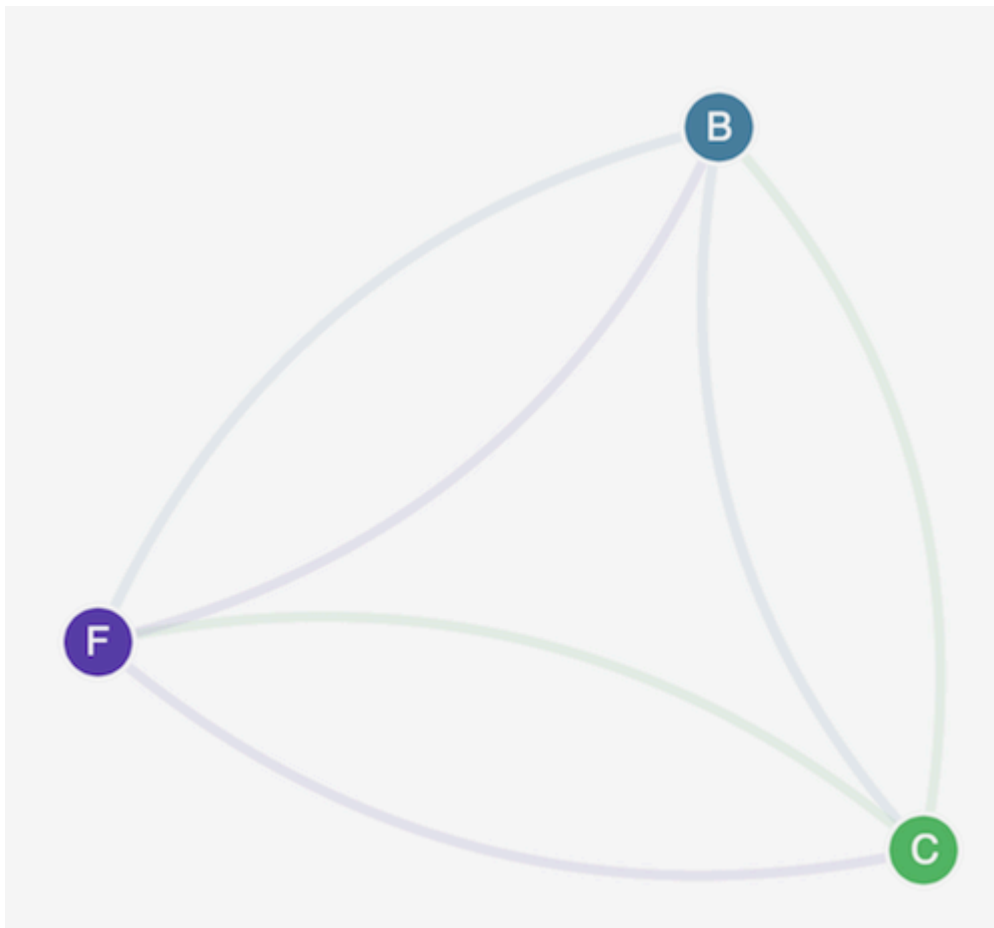


```
ingress:
 - from:
 - namespaceSelector:
 matchLabels:
 role: management-ui
```

次のコマンドを使用して両方のポリシーを適用できます。

```
kubectl apply -f https://raw.githubusercontent.com/aws-samples/eks-workshop/2f9d29ed3f82ed6b083649e975a0e574fb8a4058/content/beginner/120_network-policies/calico/stars_policy_demo/apply_network_policies.files/allow-ui.yaml
kubectl apply -f https://raw.githubusercontent.com/aws-samples/eks-workshop/2f9d29ed3f82ed6b083649e975a0e574fb8a4058/content/beginner/120_network-policies/calico/stars_policy_demo/apply_network_policies.files/allow-ui-client.yaml
```

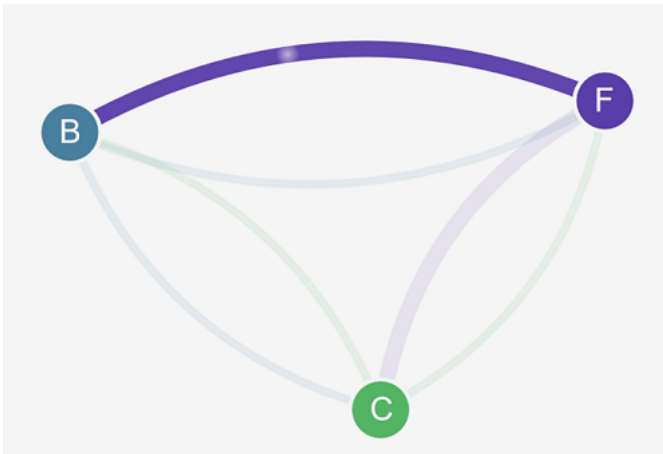
8. ブラウザを更新します。管理ユーザーインターフェイスは再びノードにアクセスできますが、ノードは相互に通信できないことがわかります。



9. フロントエンドサービスからバックエンドサービスへのトラフィックを許可するには、次のネットワークポリシーを適用します。

```
kind: NetworkPolicy
apiVersion: networking.k8s.io/v1
metadata:
 namespace: stars
 name: backend-policy
spec:
 podSelector:
 matchLabels:
 role: backend
 ingress:
 - from:
 - podSelector:
 matchLabels:
 role: frontend
 ports:
 - protocol: TCP
 port: 6379
```

10. ブラウザを更新します。フロントエンドがバックエンドと通信できることがわかります。

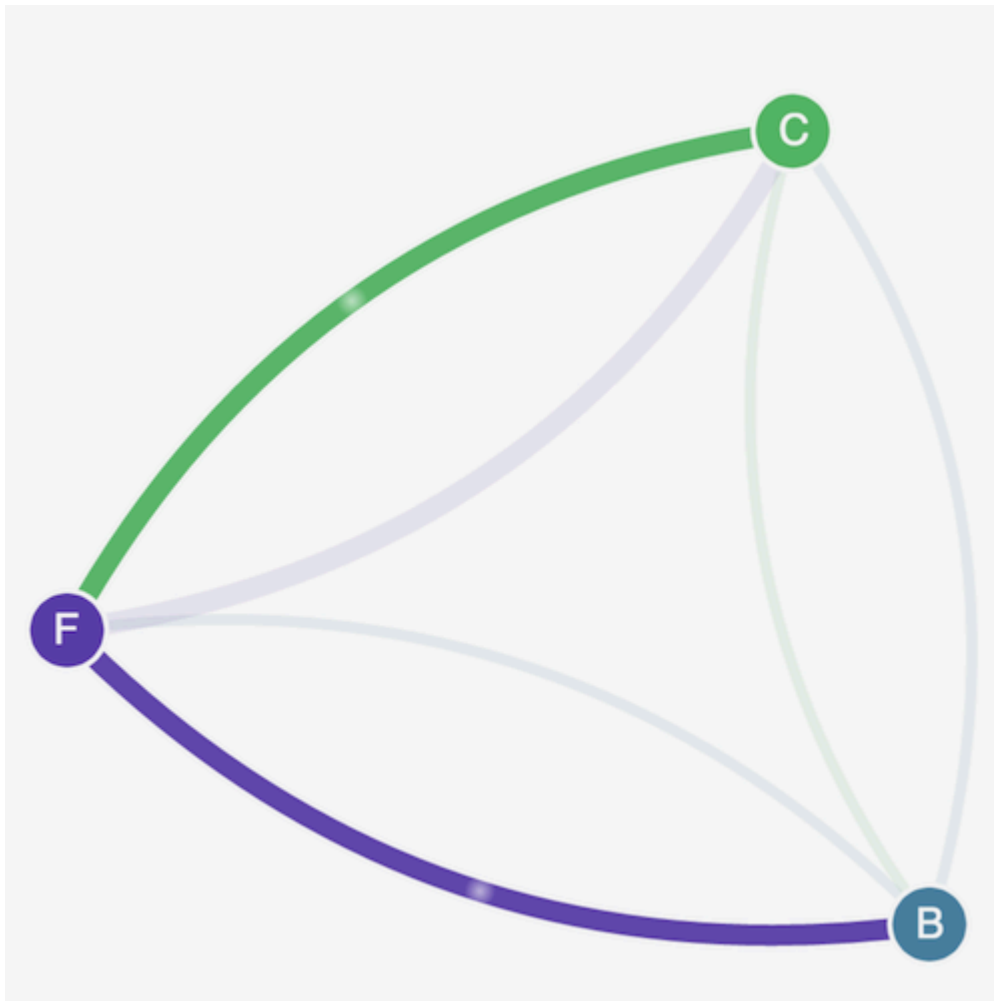


11. クライアントからフロントエンドサービスへのトラフィックを許可するには、次のネットワークポリシーを適用します。

```
kind: NetworkPolicy
apiVersion: networking.k8s.io/v1
metadata:
 namespace: stars
 name: frontend-policy
```

```
spec:
 podSelector:
 matchLabels:
 role: frontend
 ingress:
 - from:
 - namespaceSelector:
 matchLabels:
 role: client
 ports:
 - protocol: TCP
 port: 80
```

12. ブラウザを更新します。クライアントがフロントエンドサービスと通信できることがわかります。フロントエンドサービスは、バックエンドサービスと引き続き通信できます。



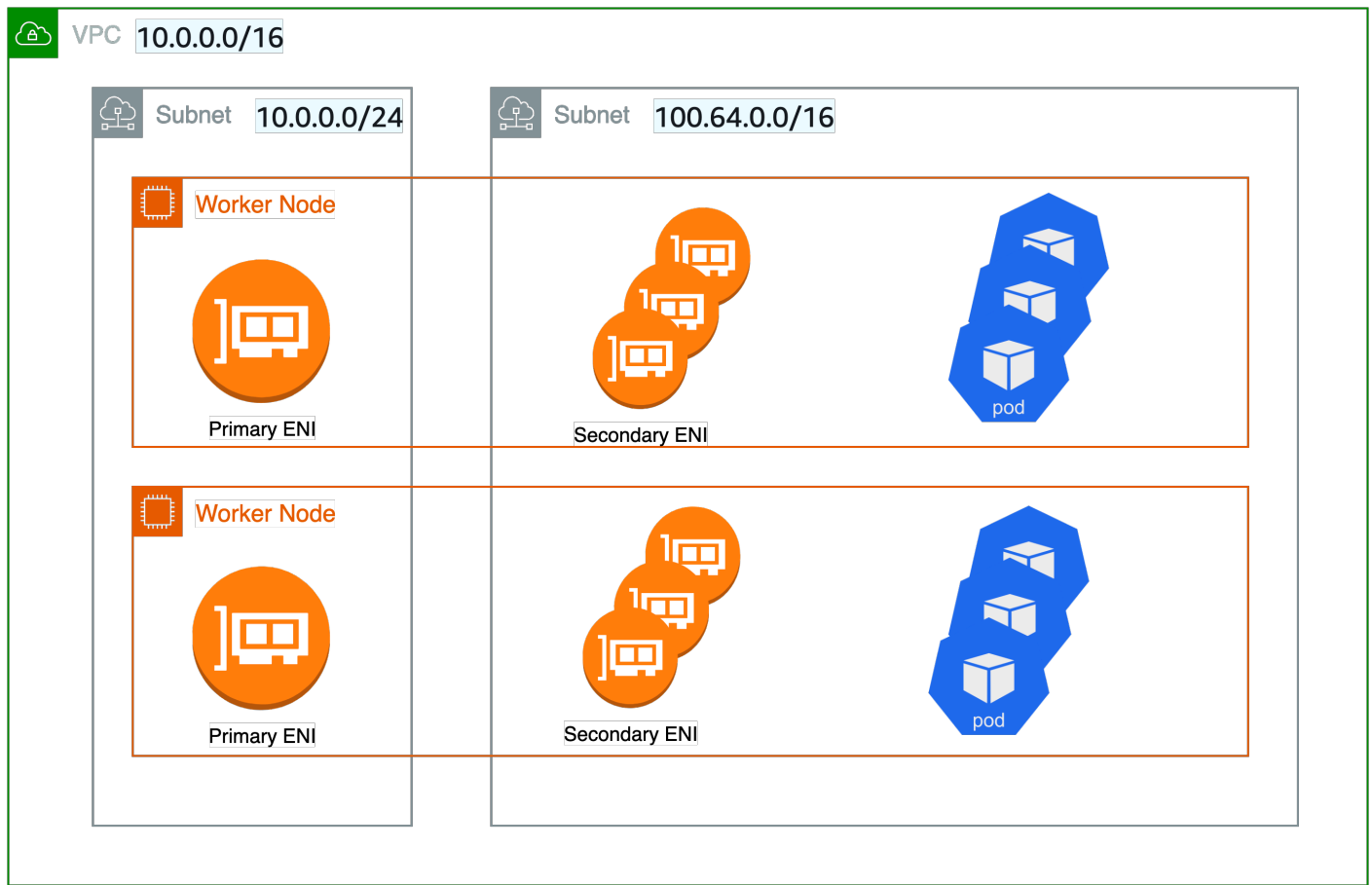
13(オプション) デモが完了したら、そのリソースを削除することができます。

```
kubectl delete -f https://raw.githubusercontent.com/aws-samples/eks-workshop/2f9d29ed3f82ed6b083649e975a0e574fb8a4058/content/beginner/120_network-policies/calico/stars_policy_demo/create_resources.files/client.yaml
kubectl delete -f https://raw.githubusercontent.com/aws-samples/eks-workshop/2f9d29ed3f82ed6b083649e975a0e574fb8a4058/content/beginner/120_network-policies/calico/stars_policy_demo/create_resources.files/frontend.yaml
kubectl delete -f https://raw.githubusercontent.com/aws-samples/eks-workshop/2f9d29ed3f82ed6b083649e975a0e574fb8a4058/content/beginner/120_network-policies/calico/stars_policy_demo/create_resources.files/backend.yaml
kubectl delete -f https://raw.githubusercontent.com/aws-samples/eks-workshop/2f9d29ed3f82ed6b083649e975a0e574fb8a4058/content/beginner/120_network-policies/calico/stars_policy_demo/create_resources.files/management-ui.yaml
kubectl delete -f https://raw.githubusercontent.com/aws-samples/eks-workshop/2f9d29ed3f82ed6b083649e975a0e574fb8a4058/content/beginner/120_network-policies/calico/stars_policy_demo/create_resources.files/namespace.yaml
```

リソースを削除した後でも、ノード上にネットワークポリシーエンドポイントが残っている場合があります、それがクラスター内のネットワークを予期しない方法で妨げることがあります。これらのルールを削除する唯一の確実な方法は、ノードを再起動するか、すべてのノードを終了して、リサイクルすることです。すべてのノードを終了するには、Auto Scaling グループの必要数を 0 に設定してから目的数にバックアップするか、単にノードを終了します。

## カスタムネットワーキングを使用して代替サブネットに Pod をデプロイする

適用対象: Linux IPv4 Fargate ノード、Amazon EC2 インスタンスを持つ Linux ノード



Amazon VPC CNI plugin for Kubernetes が、Amazon EC2 ノード用にセカンダリ [Elastic Network Interface](#) (ネットワークインターフェイス) を作成する際には、デフォルトで、ノードのプライマリネットワークインターフェイスと同じサブネットがその格納先になります。また、プライマリネットワークインターフェイスに関連付けられているものと同じセキュリティグループが、セカンダリネットワークインターフェイスにも関連付けられます。以下の 1 つ以上の理由により、プラグインを使用して異なるサブネットにセカンダリネットワークインターフェイスを作成させたり、セカンダリネットワークインターフェイスに異なるセキュリティグループを関連付けたり、また、その両方を行ったりする必要性が生じることがあります。

- プライマリネットワークインターフェイスが存在するサブネットで使用可能な IPv4 アドレスには、数の上で制限があります。これにより、サブネット内に作成できる Pod の数に制限が生じる可能性があります。セカンダリネットワークインターフェイス用に別のサブネットを指定すると、Pod のために使用可能な IPv4 アドレスの数を増やすことができます。
- セキュリティ上の理由からも、ノードのプライマリネットワークインターフェイスで使用するものとは異なるセキュリティグループまたはサブネットが、Pod に必要となる場合があります。

- ノードをパブリックサブネット内に構成した場合、Pod はプライベートサブネット内に配置します。パブリックサブネットに関連付けられているルートテーブルには、インターネットゲートウェイへのルートが含まれています。プライベートサブネットに関連付けられているルートテーブルには、インターネットゲートウェイへのルートは含まれません。

**i** Tip

また、カスタムネットワーキングを使用することなく Amazon EKS クラスターに直接新規または既存のサブネットを追加することもできます。詳細については、「[the section called “管理コンソールから既存の VPC サブネットを Amazon EKS クラスターに追加する”](#)」を参照してください。

**考慮事項**

この機能を使用する際の考慮事項を次に示します。

- カスタムネットワーキングを有効にすると、プライマリネットワークインターフェイスに割り当てられた IP アドレスが、Pod に割り当てられることはなくなります。Pod には、セカンダリネットワークインターフェイスからの IP アドレスのみが割り当てられます。
- クラスターで IPv6 ファミリーを使用している場合は、カスタムネットワーキングを使用することはできません。
- IPv4 アドレスの枯渇を軽減するために、カスタムネットワーキングを使用する予定の場合は、IPv6 ファミリーを使用してクラスターを作成することもできます。詳細については、「[the section called “IPv6”](#)」を参照してください。
- サブネットにデプロイされたセカンダリネットワークインターフェイス用の Pod が、ノードのプライマリネットワークインターフェイスとは異なるサブネットおよびセキュリティグループを使用できるように設定されていても、そのサブネットとセキュリティグループは、ノードと同じ VPC 内に配置される必要があります。
- Fargate の場合、サブネットは Fargate プロファイルを介して制御されます。詳細については、「[the section called “プロファイルを定義する”](#)」を参照してください。

**Amazon EKS ノードのセカンダリネットワークインターフェイスをカスタマイズする**

このチュートリアルを開始する前に以下を完了してください：

- 考慮事項を確認する

- Amazon VPC CNI plugin for Kubernetes によるセカンダリネットワークインターフェイスの作成と、Pod への IP アドレス割り当て方法に関する知識。詳細については、GitHub の「[ENI Allocation](#)」(ENI 割り当て) を参照してください。
- ご使用のデバイスまたは AWS クラウドシェル で、バージョン 2.12.3 以降、または AWS コマンドラインインターフェイス (AWS CLI のバージョン 1.27.160 以降がインストールおよび設定されていること。現在のバージョンを確認するには「`aws --version | cut -d / -f2 | cut -d ' ' -f1`」を参照してください。yum、apt-get、macOS 用の Homebrew などのパッケージマネージャーは、多くの場合 AWS CLI の最新バージョンより数バージョン古くなっています。最新バージョンをインストールするには「AWS コマンドラインインターフェイスユーザーガイド」の「[インストール](#)」および「[aws configure を使用したクイック設定](#)」を参照してください。AWS クラウドシェル にインストールされている AWS CLI バージョンも最新バージョンより数バージョン遅れることがあります。更新するには「AWS クラウドシェル ユーザーガイド」の「[ホームディレクトリへの AWS CLI のインストール](#)」を参照してください。
- デバイスまたは AWS クラウドシェル に、kubectl コマンドラインツールがインストールされていること。kubectl をインストールまたはアップグレードする方法については「[the section called “kubectl と eksctl のセットアップ”](#)」を参照してください。
- このトピック内のステップは Bash シェル内で実行することが推奨されます。Bash シェルを使用していない場合、行継続文字や、変数の設定と使用に関する方法など、一部のスクリプトコマンドのためにシェルの調整が必要となります。さらに、シェルの引用規則とエスケープ規則は異なる場合があります。詳細については「AWS コマンドラインインターフェイスのユーザーガイド」の「[AWS CLI での文字列への引用符の使用](#)」を参照してください。

このチュートリアルでは置き換えが必要だと明記されている場合を除き、サンプル値をそのまま使用することをお勧めします。本番向けクラスター向けにステップを完了する際には例値 を任意の値に置き換えることができます。すべてのステップは同一のターミナルで実行することをお勧めします。設定された変数はステップ全体で使用され、また、これらは異なるターミナルには存在しないためです。

このトピックのコマンドは「[AWS CLI 使用例を使用する](#)」に記載されている規則に従ってフォーマットされています。使用している AWS CLI [プロファイル](#) で定義されているデフォルトの AWS リージョンとは異なる AWS リージョンにあるリソースに対してコマンドラインからコマンドを実行する場合はコマンドに `--region us-west-2` を追加する必要があります。us-west-2 は自分の AWS リージョンに置き換えます。

カスタムネットワーキングを本番用クラスターにデプロイする場合は、[the section called “ステップ 2: VPC を設定する”](#) にスキップします。

## ステップ 1: テスト VPC およびクラスターを作成する

次の手順により、テスト VPC とクラスターを作成し、そのクラスターのためにカスタムネットワークを構成できます。本番向けのクラスターで必要になるいくつかの機能で、このトピックに関連のないものについては説明していません。そのため、本番ワークロード向けに、このテストクラスターを使用することはお勧めしません。詳細については、「[the section called “クラスターを作成する”](#)」を参照してください。

1. 以下のコマンドを実行して、`account_id` 変数を定義します。

```
account_id=$(aws sts get-caller-identity --query Account --output text)
```

2. VPC を作成します。

- a. テストシステムにデプロイする場合は Amazon EKS AWS クラウドフォーメーション テンプレートを使用して VPC を作成します。

```
aws cloudformation create-stack --stack-name my-eks-custom-networking-vpc \
 --template-url https://s3.us-west-2.amazonaws.com/amazon-eks/
cloudformation/2020-10-29/amazon-eks-vpc-private-subnets.yaml \
 --parameters ParameterKey=VpcBlock,ParameterValue=192.168.0.0/24 \
 ParameterKey=PrivateSubnet01Block,ParameterValue=192.168.0.64/27 \
 ParameterKey=PrivateSubnet02Block,ParameterValue=192.168.0.96/27 \
 ParameterKey=PublicSubnet01Block,ParameterValue=192.168.0.0/27 \
 ParameterKey=PublicSubnet02Block,ParameterValue=192.168.0.32/27
```

- b. AWS CloudFormation スタックが作成されるまで数分かかります。次のコマンドを実行して、スタックのデプロイステータスを確認します。

```
aws cloudformation describe-stacks --stack-name my-eks-custom-networking-vpc --
query Stacks\[\\].StackStatus --output text
```

コマンドの出力が「CREATE\_COMPLETE」になるまで、次のステップに進まないでください。

- c. テンプレートによって作成された、プライベートサブネット ID の値により変数を定義します。

```
subnet_id_1=$(aws cloudformation describe-stack-resources --stack-name my-eks-
custom-networking-vpc \
 --query "StackResources[?
LogicalResourceId=='PrivateSubnet01'].PhysicalResourceId" --output text)
subnet_id_2=$(aws cloudformation describe-stack-resources --stack-name my-eks-
custom-networking-vpc \
```



```
--query "StackResources[?
LogicalResourceId=='PrivateSubnet02'].PhysicalResourceId" --output text)
```

- d. 前のステップで取得した、サブネットのアベイラビリティゾーンを使用して変数を定義します。

```
az_1=$(aws ec2 describe-subnets --subnet-ids $subnet_id_1 --query
'Subnets[*].AvailabilityZone' --output text)
az_2=$(aws ec2 describe-subnets --subnet-ids $subnet_id_2 --query
'Subnets[*].AvailabilityZone' --output text)
```

### 3. クラスターの IAM ロールを作成します。

- a. IAM 信頼ポリシー用の JSON ファイルを作成するには次のコマンドを実行してください。

```
cat >eks-cluster-role-trust-policy.json <<EOF
{
 "Version": "2012-10-17",
 "Statement": [
 {
 "Effect": "Allow",
 "Principal": {
 "Service": "eks.amazonaws.com"
 },
 "Action": "sts:AssumeRole"
 }
]
}
EOF
```

- b. Amazon EKS クラスターの IAM ロールを作成します。必要に応じて、eks-cluster-role-trust-policy.json の前に、ステップでファイルの書き込み先となったコンピュータ上のパスを追加します。このコマンドは前のステップで作成した信頼ポリシーをロールに関連付けます。IAM ロールを作成するにはロールを作成する [IAM プリンシパル](#)に iam:CreateRole アクション (許可) を割り当てる必要があります。

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

- c. [AmazonEKSClusterPolicy](#) という名前の Amazon EKS マネージド型ポリシーをロールにアタッチします。IAM ポリシーを [IAM プリンシパル](#)にアタッチするにはポリシーのアタッチを行っ

ているプリンシパルに、次のいずれかの IAM アクション (許可) を割り当てる必要があります:  
iam:AttachUserPolicy または iam:AttachRolePolicy

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

#### 4. Amazon EKS クラスターを作成し、このクラスターとデバイスの間の通信を設定します。

##### a. クラスターを作成する。

```
aws eks create-cluster --name my-custom-networking-cluster \
 --role-arn arn:aws:iam::$account_id:role/myCustomNetworkingAmazonEKSClusterRole \
 --resources-vpc-config subnetIds=$subnet_id_1,"$subnet_id_2
```

##### Note

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

##### b. クラスターが作成されるまでに数分かかります。次のコマンドを実行して、クラスターのデプロイステータスを確認します。

```
aws eks describe-cluster --name my-custom-networking-cluster --query
cluster.status
```

コマンドの出力が「"ACTIVE"」になるまで、次のステップに進まないでください。

##### c. クラスターと通信するために kubectl を設定します。

```
aws eks update-kubeconfig --name my-custom-networking-cluster
```

## ステップ 2: VPC を設定する

このチュートリアルでは、[the section called “ステップ 1: テスト VPC およびクラスターを作成する”](#)で作成した VPC が必要です。本番向けクラスターの場合は、example values をすべて実際の値に置き換えて、VPC に合うように各ステップを調整します。

1. 現在インストールされている Amazon VPC CNI plugin for Kubernetes が最新バージョンであることを確認します。Amazon EKS アドオンタイプの最新バージョンを確認し、そのバージョンに更新するには「[the section called “アドオンを更新する”](#)」を参照してください。セルフマネージドアドオンタイプの最新バージョンを確認し、そのバージョンに更新するには「[the section called “Amazon VPC CNI”](#)」を参照してください。
2. クラスター VPC の ID を取得し、後のステップで使用するために変数に格納します。

```
vpc_id=$(aws eks describe-cluster --name my-custom-networking-cluster --query
"cluster.resourcesVpcConfig.vpcId" --output text)
```

3. 追加の Classless Inter-Domain Routing (CIDR ブロックを、クラスターの VPC に関連付けます。CIDR ブロックは既存の関連付けられた CIDR ブロックと重複することはできません。
  - a. 現在 VPC に関連付けられている CIDR ブロックを表示します。

```
aws ec2 describe-vpcs --vpc-ids $vpc_id \
 --query 'Vpcs[*].CidrBlockAssociationSet[*].{CIDRBlock: CidrBlock, State:
 CidrBlockState.State}' --out table
```

出力例は次のとおりです。

```

| DescribeVpcs |
+-----+-----+
| CIDRBlock | State |
+-----+-----+
| 192.168.0.0/24 | associated |
+-----+-----+
```

- b. 追加の CIDR ブロックを VPC に関連付けます。次のコマンドで CIDR ブロック値を置き換えます。詳細については「Amazon VPC ユーザーガイド」の「[追加の IPv4 CIDR ブロックと VPC の関連付け](#)」を参照してください。

```
aws ec2 associate-vpc-cidr-block --vpc-id $vpc_id --cidr-block 192.168.1.0/24
```

- c. 新しいブロックが関連付けられていることを確認します。

```
aws ec2 describe-vpcs --vpc-ids $vpc_id --query
'Vpcs[*].CidrBlockAssociationSet[*].{CIDRBlock: CidrBlock, State:
CidrBlockState.State}' --out table
```

出力例は次のとおりです。

```

| DescribeVpcs |
+-----+-----+
| CIDRBlock | State |
+-----+-----+
| 192.168.0.0/24 | associated |
| 192.168.1.0/24 | associated |
+-----+-----+
```

新しい CIDR ブロックの State が associated に遷移するまで、次のステップには進まないでください。

4. 既存のサブネットが存在する各アベイラビリティゾーン内で、使用するサブネットを必要な数だけ作成します。前のステップで VPC に関連付けた、CIDR ブロック内にある CIDR ブロックを指定します。
  - a. 新しいサブネットを作成します。次のコマンドで CIDR ブロック値を置き換えます。サブネットは既存のサブネットが置かれているものとは異なる VPC CIDR ブロックの中に作成する必要があります。作成するアベイラビリティゾーンは既存のサブネットと同じにする必要があります。この例では現在のプライベートサブネットが存在する各アベイラビリティゾーンにある新しい CIDR ブロックの中に、1 つのサブネットが作成されます。作成されたサブネットの ID は後のステップで使用するために変数に格納されます。Name の値は前のステップで Amazon EKS VPC テンプレートを使用して作成されたサブネットに、割り当てられている値と一致します。名前は必須ではありません。名前には異なる値を使用できます。

```
new_subnet_id_1=$(aws ec2 create-subnet --vpc-id $vpc_id --availability-zone $az_1
--cidr-block 192.168.1.0/27 \
--tag-specifications 'ResourceType=subnet,Tags=[{Key=Name,Value=my-eks-custom-
networking-vpc-PrivateSubnet01},{Key=kubernetes.io/role/internal-elb,Value=1}]' \
--query Subnet.SubnetId --output text)
new_subnet_id_2=$(aws ec2 create-subnet --vpc-id $vpc_id --availability-zone $az_2
--cidr-block 192.168.1.32/27 \
```

```
--tag-specifications 'ResourceType=subnet,Tags=[{Key=Name,Value=my-eks-custom-networking-vpc-PrivateSubnet02},{Key=kubernetes.io/role/internal-elb,Value=1}]' \
--query Subnet.SubnetId --output text)
```

### Important

デフォルトで新しいサブネットはVPC の[メインルートテーブル](#)に暗黙的に関連付けられます。このルートテーブルにより、その VPC にデプロイされているすべてのリソース間の通信が可能になります。ただし、VPC に関連付けられた CIDR ブロックの外部にある IP アドレスが割り当てられたリソースとの通信は許可されません。この動作を変更するには独自のルートテーブルをサブネットに関連付けます。詳細については「Amazon VPC ユーザーガイド」の「[サブネットルートテーブル](#)」を参照してください。

b. 使用している VPC の現在のサブネットを表示します。

```
aws ec2 describe-subnets --filters "Name=vpc-id,Values=$vpc_id" \
--query 'Subnets[*].{SubnetId: SubnetId,AvailabilityZone:
AvailabilityZone,CidrBlock: CidrBlock}' \
--output table
```

出力例は次のとおりです。

```

| DescribeSubnets |
+-----+-----+-----+
| AvailabilityZone | CidrBlock | SubnetId |
+-----+-----+-----+
us-west-2d	192.168.0.0/27	subnet-example1
us-west-2a	192.168.0.32/27	subnet-example2
us-west-2a	192.168.0.64/27	subnet-example3
us-west-2d	192.168.0.96/27	subnet-example4
us-west-2a	192.168.1.0/27	subnet-example5
us-west-2d	192.168.1.32/27	subnet-example6
+-----+-----+-----+
```

作成した 192.168.1.0 CIDR ブロック内のサブネットが、192.168.0.0 CIDR ブロック内のサブネットと同じアベイラビリティゾーンに置かれていることが確認できます。

## ステップ 3: Kubernetes リソースを設定する

1. aws-node DaemonSet で AWS\_VPC\_K8S\_CNI\_CUSTOM\_NETWORK\_CFG 環境変数を true に設定します。

```
kubectl set env daemonset aws-node -n kube-system
AWS_VPC_K8S_CNI_CUSTOM_NETWORK_CFG=true
```

2. [クラスターのセキュリティグループ](#) の ID を取得し、次のステップで使用するために変数に格納します。このセキュリティグループはクラスターの作成時に Amazon EKS により自動的に作成されます。

```
cluster_security_group_id=$(aws eks describe-cluster --name my-custom-networking-cluster --query cluster.resourcesVpcConfig.clusterSecurityGroupId --output text)
```

3. Pod をデプロイするサブネットごとに ENIConfig カスタムリソースを作成します。

- a. 各ネットワークインターフェイス設定に固有のファイルを作成します。

次のコマンドでは、前のステップで作成した 2 つのサブネットのそれぞれに、個別の ENIConfig ファイルが作成されます。name の値は一意である必要があります。これらの名前はサブネットが存在するアベイラビリティゾーンと同じものが付けられます。クラスターセキュリティグループは、ENIConfig に割り当てられます。

```
cat >$az_1.yaml <<EOF
apiVersion: crd.k8s.amazonaws.com/v1alpha1
kind: ENIConfig
metadata:
 name: $az_1
spec:
 securityGroups:
 - $cluster_security_group_id
 subnet: $new_subnet_id_1
EOF
```

```
cat >$az_2.yaml <<EOF
apiVersion: crd.k8s.amazonaws.com/v1alpha1
kind: ENIConfig
metadata:
 name: $az_2
spec:
 securityGroups:
```

```
- $cluster_security_group_id
 subnet: $new_subnet_id_2
EOF
```

本番向けのクラスターでは、前のコマンドに次の変更を行います。

- `$cluster_security_group_id` は各 ENIConfig で使用する既存の[セキュリティグループ](#)の ID に置き換えます。
- 可能な限り、ENIConfig が使用されるアベイラビリティーゾーンと同じ名前を、ENIConfigs に付けることをお勧めします。いくつかの理由から、ENIConfigs に対して、アベイラビリティーゾーンとは異なる名前を使用する必要が生じることがあります。例えば、同じアベイラビリティーゾーンに 3 つ以上のサブネットがあり、そのすべてでカスタムネットワーキングを使用したい場合は同一のアベイラビリティーゾーンに複数の ENIConfigs が必要になります。各 ENIConfig には一意の名前が必要であるため、複数の ENIConfigs に対しアベイラビリティーゾーンの名前を使用することはできません。

ENIConfig の中で、アベイラビリティーゾーンと異なる名前を持つものについては前のコマンドで `$az_1` と `$az_2` を独自の名前に置き換えます。さらに、この後のチュートリアルで[ENIConfig でノードをアノテーション](#)します。

 Note

本番向けクラスターで使用するために有効なセキュリティグループを指定しておらず、かつ:

- Amazon VPC CNI plugin for Kubernetes のバージョン 1.8.0 以降を使用する場合は、ノードのプライマリ Elastic Network Interface に関連付けられたセキュリティグループが使用されます。
- 1.8.0 より前のバージョンの Amazon VPC CNI plugin for Kubernetes を使用している場合は、VPC のデフォルトのセキュリティグループが、セカンダリネットワークインターフェイスに割り当てられます。

 Important

- `AWS_VPC_K8S_CNI_EXTERNALSNAT=false` は、Kubernetes 用 Amazon VPC CNI プラグインのデフォルトの設定です。デフォルト設定を使用している場合、VPC に関連付けられた CIDR ブロックのいずれにもない IP アドレスに向けて送信されたトラフィックにはノードのプライマリネットワークインターフェイスでの、セキュリ

ティグループとサブネットが使用されます。セカンダリネットワークインターフェイスを作成するために使用された、ENIConfigs で定義されているサブネットとセキュリティグループはこのトラフィックでは使用されません。この設定の詳細については「[the section called “アウトバウンドトラフィック”](#)」を参照してください。

- また、Pod でもセキュリティグループを使用する場合、そのセキュリティグループは SecurityGroupPolicy で指定されたものとなり、ENIConfigs で指定されたものは使用されません。詳細については、「[the section called “ポッドのセキュリティグループ”](#)」を参照してください。

- b. 以下のコマンドを使用して、作成したカスタムリソースファイルをそれぞれクラスターに適用します。

```
kubectl apply -f $az_1.yaml
kubectl apply -f $az_2.yaml
```

4. ENIConfigs が作成されたことを確認します。

```
kubectl get ENIConfigs
```

出力例は次のとおりです。

| NAME       | AGE  |
|------------|------|
| us-west-2a | 117s |
| us-west-2d | 105s |

5. 本番向けのクラスターで、カスタムネットワーキングを有効にしており、ENIConfigs の名前を使用されているアベイラビリティゾーンとは異なるものにした場合は[次のステップ](#)に移り、Amazon EC2 ノードをデプロイします。

クラスターで作成された任意の新しい Amazon EC2 ノードに対し、Kubernetes がアベイラビリティゾーン用の ENIConfig を自動的に適用することを許可します。

- a. このチュートリアルでのテスト用クラスターの場合は、このまま[次のステップ](#)に移動します。

実稼働クラスターの場合は、[ENI\\_CONFIG\\_ANNOTATION\\_DEF](#) 環境変数用のキー `aws-node` を使用するアノテーションが、`k8s.amazonaws.com/eniConfig DaemonSet` のためのコンテナ仕様内に存在することを確認します。



```
kubectl describe daemonset aws-node -n kube-system | grep
ENI_CONFIG_ANNOTATION_DEF
```

出力が返される場合はアノテーションが存在します。出力が返されない場合、その変数は設定されていません。本番向けクラスターの場合にはここでの設定と、以降のステップで示す設定のどちらも使用可能です。ここでの設定を使用すると、以降のステップの設定は上書きされます。このチュートリアルではこの後のステップでの設定を使用します。

- b. クラスター内に作成された任意の新しい Amazon EC2 ノードに対し、自動的にアベイラビリティゾーンの ENIConfig を適用するように、aws-node DaemonSet を更新します。

```
kubectl set env daemonset aws-node -n kube-system
ENI_CONFIG_LABEL_DEF=topology.kubernetes.io/zone
```

## ステップ 4: Amazon EC2 ノードをデプロイする

### 1. ノードの IAM ロールを作成します。

- a. IAM 信頼ポリシー用の JSON ファイルを作成するには次のコマンドを実行してください。

```
cat >node-role-trust-relationship.json <<EOF
{
 "Version": "2012-10-17",
 "Statement": [
 {
 "Effect": "Allow",
 "Principal": {
 "Service": "ec2.amazonaws.com"
 },
 "Action": "sts:AssumeRole"
 }
]
}
EOF
```

- b. IAM ロールを作成し、返された Amazon リソースネーム (ARN) を、後のステップで使用するために変数に格納します。

```
node_role_arn=$(aws iam create-role --role-name myCustomNetworkingNodeRole --
assume-role-policy-document file://"/node-role-trust-relationship.json" \
```

```
--query Role.Arn --output text)
```

c. IAM ロールに、3 つの必須な IAM マネージドポリシーをアタッチします。

```
aws iam attach-role-policy \
 --policy-arn arn:aws:iam::aws:policy/AmazonEKSEKSWorkerNodePolicy \
 --role-name myCustomNetworkingNodeRole
aws iam attach-role-policy \
 --policy-arn arn:aws:iam::aws:policy/AmazonEC2ContainerRegistryReadOnly \
 --role-name myCustomNetworkingNodeRole
aws iam attach-role-policy \
 --policy-arn arn:aws:iam::aws:policy/AmazonEKS_CNI_Policy \
 --role-name myCustomNetworkingNodeRole
```

#### Important

分かりやすくするため、このチュートリアルでは[AmazonEKS\\_CNI\\_Policy](#) ポリシーはノードの IAM ロールにアタッチされています。ただし、本番向けクラスターの場合は、Amazon VPC CNI plugin for Kubernetes のみで使用される IAM ロールに、個別にポリシーをアタッチすることをお勧めします。詳細については、「[the section called “IRSA 用の設定”](#)」を参照してください。

2. 次のいずれかのタイプのノードグループを作成します。デプロイするインスタンスタイプを確認するには「[the section called “Amazon EC2 インスタンスタイプ”](#)」を参照してください。このチュートリアルのために、マネージド型、および起動テンプレートを使用しない、または AMI ID が指定されていない起動テンプレートを使用するオプションを完了します。本番ワークロードでノードグループを使用する場合はそのグループをデプロイする前に、[マネージドノードグループ](#)および[セルフマネージドノードグループ](#)のすべてのオプションについて理解しておくことをお勧めします。

- マネージド型 — 次のいずれかのオプションを使用して、ノードグループをデプロイします：
  - 起動テンプレートを使用しない、または AMI ID が指定されていない起動テンプレートを使用する — 次のコマンドを実行してください。このチュートリアルではサンプルの値を使用します。本番向けのノードグループの場合はすべてのサンプルの値を、実際の値に置き換えてください。ノードグループ名は 63 文字以下である必要があります。先頭は文字または数字でなければなりません、残りの文字にはハイフンおよびアンダースコアを含めることもできます。

```
aws eks create-nodegroup --cluster-name my-custom-networking-cluster --nodegroup-name my-nodegroup \
 --subnets $subnet_id_1 $subnet_id_2 --instance-types t3.medium --node-role $node_role_arn
```

- 指定された AMI ID を持つ起動テンプレートを使用

- A. ノードでの Pod について、Amazon EKS で推奨される最大数を決定します。「[各 Amazon EC2 インスタンスタイプの Amazon EKS の推奨最大ポッド数](#)」の指示に従い、そのトピックのステップ 3 で `--cni-custom-networking-enabled` を追加します。後のステップで使用するために、この出力を書き留めます。
- B. 起動テンプレートで、Amazon EKS 最適化 AMI ID を指定するか、Amazon EKS 最適化 AMI から構築されたカスタム AMI を指定します。その後、[起動テンプレートを使用してノードグループをデプロイ](#)し、さらに、起動テンプレートにある次のユーザーデータを指定します。このユーザーデータは引数を `bootstrap.sh` ファイルに渡します。ブートストラップファイルの詳細については、GitHub の「[bootstrap.sh](#)」を参照してください。20 は、前のステップで使用した値 (推奨) か、または独自の値に置き換えることができます。

```
/etc/eks/bootstrap.sh my-custom-networking-cluster --use-max-pods false --
kubelet-extra-args '--max-pods=20'
```

Amazon EKS 最適化 AMI から構築されていないカスタム AMI を作成した場合は自分で設定をカスタム作成する必要があります。

- セルフマネージド型

- i. ノードでの Pod について、Amazon EKS で推奨される最大数を決定します。[the section called “各 Amazon EC2 インスタンスタイプの Amazon EKS の推奨最大ポッド数”](#) の手順に従います (`--cni-custom-networking-enabled` をステップ 3 に追加してください)。後のステップで使用するために、この出力を書き留めます。
- ii. [the section called “Amazon Linux”](#) の手順に従ってノードグループをデプロイします。ブートストラップ引数 パラメータに、以下のテキストを指定します。20 は、前のステップで使用した値 (推奨) か、または独自の値に置き換えることができます。

```
--use-max-pods false --kubelet-extra-args '--max-pods=20'
```

 Note

本番クラスター内のノードで、非常に大量の Pod をサポートさせる場合には、「[the section called “各 Amazon EC2 インスタンスタイプの Amazon EKS の推奨最大ポッド数”](#)」のスクリプトを再度実行します。また、このコマンドには `--cni-prefix-delegation-enabled` オプションを追加します。例えば、`m5.large` インスタンスタイプの場合は 110 が返されます。この機能を有効化する方法については「[the section called “IP アドレスを増やす”](#)」を参照してください。この機能はカスタムネットワーキングで使用できます。

3. ノードグループの作成には数分かかります。次のコマンドを使用すると、マネージド型のノードグループ作成のステータスを確認できます。

```
aws eks describe-nodegroup --cluster-name my-custom-networking-cluster --nodegroup-name my-nodegroup --query nodegroup.status --output text
```

出力が「ACTIVE」を返すまで、次のステップに進まないでください。

4. このチュートリアルではこのステップをスキップできます。

本番向けのクラスターにおいて、ENIConfigs の名前を、それが使用されているアベイラビリティゾーンと同じにしなかった場合にはノードに対し、そのノードで使用するべき ENIConfig の名前をアノテーションします。各アベイラビリティゾーンにサブネットが 1 つしかなく、ENIConfigs にアベイラビリティゾーンと同じ名前を付けている場合にはこのステップは必要ありません。[前のステップ](#)でその設定を有効にしている場合、Amazon VPC CNI plugin for Kubernetes が適切な ENIConfig をノードに対し自動的に関連付けるためです。

- a. クラスター内のノードのリストを取得します。

```
kubectl get nodes
```

出力例は次のとおりです。

| NAME                                                       | STATUS | ROLES  | AGE   | VERSION  |
|------------------------------------------------------------|--------|--------|-------|----------|
| ip-192-168-0-126.us-west-2.compute.internal<br>eks-810597c | Ready  | <none> | 8m49s | v1.22.9- |
| ip-192-168-0-92.us-west-2.compute.internal<br>eks-810597c  | Ready  | <none> | 8m34s | v1.22.9- |

- b. 各ノードが属するアベイラビリティーゾーンを決定します。前の出力に基づいて IP アドレスを置き換えて、前のステップで返されたノードごとに次のコマンドを実行します。

```
aws ec2 describe-instances --filters Name=network-interface.private-dns-name,Values=ip-192-168-0-126.us-west-2.compute.internal \
--query 'Reservations[].Instances[].{AvailabilityZone: Placement.AvailabilityZone, SubnetId: SubnetId}'
```

出力例は次のとおりです。

```
[
 {
 "AvailabilityZone": "us-west-2d",
 "SubnetId": "subnet-Example5"
 }
]
```

- c. 各ノードを、サブネット ID とアベイラビリティーゾーンのために作成した ENIConfig でアノテーションします。各ノードのアノテーションに使用できる ENIConfig は 1 つだけです。ただし、複数のノードに対し同じ ENIConfig でアノテーションすることは可能です。値の例は実際に使用する値に置き換えます。

```
kubectl annotate node ip-192-168-0-126.us-west-2.compute.internal
k8s.amazonaws.com/eniConfig=EniConfigName1
kubectl annotate node ip-192-168-0-92.us-west-2.compute.internal
k8s.amazonaws.com/eniConfig=EniConfigName2
```

5. カスタムネットワーキング機能の使用に切り替える前に、Pod を実行している本稼働クラスターにノードがあった場合は、以下のタスクを完了してください。
  - a. カスタムネットワーク機能を使用しているノードが、使用可能であることを確認します。
  - b. ノードを遮断およびドレインして、Pod をスムーズにシャットダウンします。詳細については、Kubernetes ドキュメントの「[Safely Drain a Node](#)」(ノードを安全にドレインする)を参照してください。
  - c. ノードを終了します。ノードが既存のマネージド型ノードグループ内にある場合はそのノードグループを削除できます。以下のコマンドを実行してください。

```
aws eks delete-nodegroup --cluster-name my-custom-networking-cluster --nodegroup-name my-nodegroup
```

カスタムネットワーク機能は `k8s.amazonaws.com/eniConfig` ラベルで登録されている新しいノードでのみ使用されます。

- Pod に、前の手順で作成したサブネットの 1 つに関連付けられた CIDR ブロックから、IP アドレスが割り当てられていることを確認します。

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

出力例は次のとおりです。

| NAMESPACE   | NAME                     | READY | STATUS                                      | RESTARTS  | AGE   | IP        |
|-------------|--------------------------|-------|---------------------------------------------|-----------|-------|-----------|
|             | NODE                     |       |                                             | NOMINATED | NODE  | READINESS |
| GATES       |                          |       |                                             |           |       |           |
| kube-system | aws-node-2rk4            | 1/1   | Running                                     | 0         | 7m19s |           |
|             | 192.168.0.92             |       | ip-192-168-0-92.us-west-2.compute.internal  | <none>    |       |           |
|             | <none>                   |       |                                             |           |       |           |
| kube-system | aws-node-k96wp           | 1/1   | Running                                     | 0         | 7m15s |           |
|             | 192.168.0.126            |       | ip-192-168-0-126.us-west-2.compute.internal | <none>    |       |           |
|             | <none>                   |       |                                             |           |       |           |
| kube-system | coredns-657694c6f4-smcgr | 1/1   | Running                                     | 0         | 56m   |           |
|             | 192.168.1.23             |       | ip-192-168-0-92.us-west-2.compute.internal  | <none>    |       |           |
|             | <none>                   |       |                                             |           |       |           |
| kube-system | coredns-657694c6f4-stwv9 | 1/1   | Running                                     | 0         | 56m   |           |
|             | 192.168.1.28             |       | ip-192-168-0-92.us-west-2.compute.internal  | <none>    |       |           |
|             | <none>                   |       |                                             |           |       |           |
| kube-system | kube-proxy-jgshq         | 1/1   | Running                                     | 0         | 7m19s |           |
|             | 192.168.0.92             |       | ip-192-168-0-92.us-west-2.compute.internal  | <none>    |       |           |
|             | <none>                   |       |                                             |           |       |           |
| kube-system | kube-proxy-wx9vk         | 1/1   | Running                                     | 0         | 7m15s |           |
|             | 192.168.0.126            |       | ip-192-168-0-126.us-west-2.compute.internal | <none>    |       |           |
|             | <none>                   |       |                                             |           |       |           |

VPC に追加した 192.168.1.0 CIDR ブロックからの IP アドレスが、coredns Pod に割り当てられていることが確認できます。カスタムネットワーキングを使用していない場合はここに 192.168.0.0 CIDR ブロックからのアドレスが割り当てられています。この CIDR ブロックが、元々 VPC に関連付けられている唯一のブロックであるためです。

Pod の spec が `hostNetwork=true` を含む場合には、ノードのプライマリ IP アドレスが割り当てられます。追加したサブネットのアドレスは割り当てられません。デフォルトではこの値は `false` に設定されます。クラスターで実行される kube-proxy および Amazon VPC CNI plugin

for Kubernetes (aws-node) の Pod に対して、この値は `true` に設定されます。これが、kube-proxy とプラグインの aws-node の Pod に、前の出力にある 192.168.1.x アドレスが割り当てられない理由です。Pod の hostNetwork 設定の詳細については、Kubernetes API リファレンスの「[PodSpec v1 core](#)」を参照してください。

## ステップ 5: チュートリアルでのリソースを削除する

このチュートリアル完了後は作成したリソースを削除することをお勧めします。その後、ここでのステップを調整して、本番向けクラスターのカスタムネットワーキングを有効化することができます。

1. 作成したノードグループが完全にテスト向けである場合はそれを削除してください。

```
aws eks delete-nodegroup --cluster-name my-custom-networking-cluster --nodegroup-name my-nodegroup
```

2. AWS CLI から、クラスターが削除済みだと出力された場合でも、実際にはその削除プロセスが完了していない場合があります。削除のプロセスには数分かかります。次のコマンドを実行して、処理の完了を確認します。

```
aws eks describe-nodegroup --cluster-name my-custom-networking-cluster --nodegroup-name my-nodegroup --query nodegroup.status --output text
```

次のように出力されるまで、次に進まないでください。

```
An error occurred (ResourceNotFoundException) when calling the DescribeNodegroup operation: No node group found for name: my-nodegroup.
```

3. 作成したノードグループが完全にテスト向けである場合はノードの IAM ロールを削除します。
  - a. ロールからポリシーをデタッチします。

```
aws iam detach-role-policy --role-name myCustomNetworkingNodeRole --policy-arn arn:aws:iam::aws:policy/AmazonEKSWorkerNodePolicy
aws iam detach-role-policy --role-name myCustomNetworkingNodeRole --policy-arn arn:aws:iam::aws:policy/AmazonEC2ContainerRegistryReadOnly
aws iam detach-role-policy --role-name myCustomNetworkingNodeRole --policy-arn arn:aws:iam::aws:policy/AmazonEKS_CNI_Policy
```

- b. ロールを削除します。

```
aws iam delete-role --role-name myCustomNetworkingNodeRole
```

#### 4. クラスターを削除します。

```
aws eks delete-cluster --name my-custom-networking-cluster
```

以下のコマンドを使用して、クラスターが削除されたことを確認します。

```
aws eks describe-cluster --name my-custom-networking-cluster --query cluster.status
--output text
```

次のような出力が返された場合はクラスターが正常に削除されています。

```
An error occurred (ResourceNotFoundException) when calling the DescribeCluster
operation: No cluster found for name: my-custom-networking-cluster.
```

#### 5. クラスターの IAM ロールを削除します。

##### a. ロールからポリシーをデタッチします。

```
aws iam detach-role-policy --role-name myCustomNetworkingAmazonEKSClusterRole --
policy-arn arn:aws:iam::aws:policy/AmazonEKSClusterPolicy
```

##### b. ロールを削除します。

```
aws iam delete-role --role-name myCustomNetworkingAmazonEKSClusterRole
```

#### 6. 前のステップで作成したサブネットを削除します。

```
aws ec2 delete-subnet --subnet-id $new_subnet_id_1
aws ec2 delete-subnet --subnet-id $new_subnet_id_2
```

#### 7. 作成した VPC を削除します。

```
aws cloudformation delete-stack --stack-name my-eks-custom-networking-vpc
```

プレフィックスを使用して Amazon EKS ノードに割り当てる IP アドレスを増やす

適用対象: Amazon EC2 インスタンスを持つ Linux および Windows ノード



## 適用対象: パブリックサブネットおよびプライベートサブネット

各 Amazon EC2 インスタンスではエラスティックネットワークインターフェイスの最大数と、各ネットワークインターフェイスに割り当て可能な IP アドレスの最大数がサポートされています。各ノードにはネットワークインターフェイスごとに 1 つの IP アドレスが必要です。その他の使用可能な IP アドレスはすべて Pods に割り当てることができます。Pod それぞれに固有の IP アドレスが必要です。その結果、使用可能なコンピューティングリソースとメモリリソースはあるが、Pods に割り当てる IP アドレスが不足しているために追加の Pods に対応できないノードが存在する可能性があります。

個別のセカンダリ IP アドレスではなく、IP プレフィックスをノードに割り当てて、ノードで Pods に割り当て可能な IP アドレスの数を増やす方法について説明します。各プレフィックスには複数の IP アドレスが含まれます。IP プレフィックスが割り当てられるようにクラスターを設定しない場合、クラスターは Pod 接続に必要なネットワークインターフェイスと IP アドレスを設定するために Amazon EC2 アプリケーションプログラミングインターフェイス (API) の呼び出しをさらに実行する必要があります。クラスターのサイズが大きくなるにつれて、これらの API コールの頻度が高くなるため、Pod とインスタンスの起動時間が長くなる場合があります。これにより、大規模でスパイク的なワークロードの需要を満たすためにスケーリングの遅延が発生します。また、スケーリング要件を満たすためには追加のクラスターと VPC をプロビジョニングする必要があるため、コストと管理オーバーヘッドも増加します。詳細については「GitHub」の「[Kubernetes スケーラビリティ閾値](#)」を参照してください。

## Amazon VPC CNI plugin for Kubernetes 機能との互換性

次の機能で IP プレフィックスを使用できます：

- IPv4 ソースネットワークアドレス変換 - 詳細については「[the section called “アウトバウンドトラフィック”](#)」を参照してください。
- クラスター、ポッド、サービスへの IPv6 アドレス - 詳細については「[the section called “IPv6”](#)」を参照してください。
- Kubernetes ネットワークポリシーを使用したトラフィックの制限 - 詳細については、「[the section called “Kubernetes ポリシー”](#)」を参照してください。

次のリストは適用される Amazon VPC CNI プラグイン設定に関する情報を示しています。各設定の詳細については、GitHub の「[amazon-vpc-cni-k8s](#)」を参照してください。

- WARM\_IP\_TARGET
- MINIMUM\_IP\_TARGET

- WARM\_PREFIX\_TARGET

## 考慮事項

この機能を使用する場合は以下を考慮してください：

- 各 Amazon EC2 インスタンスタイプでは、Pod の最大数がサポートされています。マネージド ノードグループが複数のインスタンスタイプで構成されている場合、クラスターのインスタンスで指定する Pod の最大数の内、最小の値がクラスター内のすべてのノードに適用されます。
- デフォルトでは1つのノードで実行できる Pods の最大数は 110 ですが、この値は変更できます。数値を変更し、既存のマネージド型ノードグループがある場合、ノードグループの AMI または起動テンプレートを次に更新するときに、新しいノードで変更済みの値が適用されるようになります。
- IP アドレスの割り当てから IP プレフィックスの割り当てに移行する場合は既存のノードにローリング置換を実行するのではなく、新しいノードグループを作成して使用可能な IP アドレスの数を増やすことをお勧めします。IP アドレスと IP プレフィックスの両方が割り当てられているノードで Pod を実行すると、アドバタイズされた IP アドレスの容量に不一致が生じ、ノード上の将来のワークロードに影響する可能性があります。推奨される移行方法については「Amazon EKS ベストプラクティスガイド」の「[セカンダリ IP モードからプレフィックス委任モード、またはその逆への移行中にすべてのノードを置換する](#)」を参照してください。
- セキュリティグループのスコープはノードレベル - 詳細については「[セキュリティグループ](#)」を参照してください。
- ネットワークインターフェイスに割り当てられた IP プレフィックスはノードあたりの高い Pod 密度をサポートし、最適な起動時間になっています。
- IP プレフィックスと IP アドレスは標準の Amazon EC2 Elastic ネットワークインターフェイスに関連付けられています。特定のセキュリティグループを必要とする Pod には、ブランチネットワークインターフェイスのプライマリ IP アドレスが割り当てられます。IP アドレスを取得するポッド、または IP プレフィックスからの IP アドレスを、同じノード上のブランチネットワークインターフェイスを取得する Pod と混在させることができます。
- Linux ノードのみを含むクラスターの場合。
  - ネットワークインターフェイスにプレフィックスを割り当てるようにアドオンを設定すると、クラスター内のすべてのノードグループにあるすべてのノードを削除しない限り、Amazon VPC CNI plugin for Kubernetes アドオンを 1.9.0 (または 1.10.1) より前のバージョンにダウングレードすることはできません。

- Pod のセキュリティグループ (POD\_SECURITY\_GROUP\_ENFORCING\_MODE=standard および AWS\_VPC\_K8S\_CNI\_EXTERNALSNAT=false に設定) も使用している場合、Pod が VPC の外部のエンドポイントと通信するときに、Pod に割り当てた任意のセキュリティグループではなく、ノードのセキュリティグループが使用されます。

セキュリティグループ (POD\_SECURITY\_GROUP\_ENFORCING\_MODE=strict に設定) も使用している場合、Pods が VPC の外部のエンドポイントと通信するときに、Pod's セキュリティグループが使用されます。

## Amazon EKS ノードで使用可能な IP アドレスを増やす

個別のセカンダリ IP アドレスではなく、IP プレフィックスをノードに割り当てて、ノードで Pod に割り当て可能な IP アドレスの数を増やす方法について説明します。

### 前提条件

- 既存のクラスターが必要です。デプロイするには「[the section called “クラスターを作成する”](#)」を参照してください。
- Amazon EKS ノードが配置されているサブネットでは、/28 個の連続ブロック (IPv4 クラスターの場合)、または /80 個の Classless Inter-Domain Routing (CIDR) ブロック (IPv6 クラスターの場合) が十分な数として必要になります。IPv6 クラスターに含めることができるのは Linux ノードだけです。IP アドレスがサブネット CIDR 全体に分散している場合、IP プレフィックスを使用すると失敗する可能性があります。次の構成を推奨します。
- サブネット CIDR 予約を使用すると、予約された範囲内の IP アドレスがまだ使用されている場合でも、解放時に IP アドレスが再割り当てされません。これにより、セグメンテーションなしでプレフィックスを割り当てることができます。
- IP プレフィックスが割り当てられているワークロードの実行に特に使用される新しいサブネットを使用してください。IP プレフィックスを割り当てると、Windows と Linux 両方のワークロードを同じサブネットで実行できます。
- ノードに IP プレフィックスを割り当てるには、ノードが AWS Nitro ベースである必要があります。Nitro ベースではないインスタンスでは、引き続き個別のセカンダリ IP アドレスが割り当てられますが、Pod に割り当てることのできる IP アドレスの数は Nitro ベースのインスタンスよりも大幅に少なくなります。
- Linux ノードのみのクラスターの場合 – クラスターが IPv4 ファミリーで設定されている場合は、バージョン 1.9.0 以降の Amazon VPC CNI plugin for Kubernetes アドオンがインストールされている必要があります。現在のバージョンは、次のコマンドで確認できます。

```
kubectl describe daemonset aws-node --namespace kube-system | grep Image | cut -d "/"
-f 2
```

クラスターが IPv6 ファミリーで設定されている場合は、バージョン 1.10.1 のアドオンがインストールされている必要があります。プラグインバージョンが必要なバージョンよりも古い場合は、更新する必要があります。詳細については、「[Amazon VPC CNI を使用してポッドに IP を割り当てる](#)」の更新セクションを参照してください。

- Windows ノードのみのクラスターの場合
- クラスターとそのプラットフォームバージョンは、次の表のバージョン以降である必要があります。クラスターバージョンをアップグレードするには、「[the section called “Kubernetes バージョンを更新する”](#)」を参照してください。クラスターがプラットフォームの最小バージョンでない場合、Amazon EKS がプラットフォームバージョンを更新するまで、ノードに IP プレフィックスを割り当てることはできません。

| Kubernetes バージョン | プラットフォームバージョン |
|------------------|---------------|
| 1.27             | eks.3         |
| 1.26             | eks.4         |

現在の Kubernetes バージョンとプラットフォームバージョンを確認するには、次のコマンドの *my-cluster* をクラスターの名前に置き換えて、変更したコマンドを実行します: `aws eks describe-cluster --name my-cluster --query 'cluster.{ "Kubernetes Version": version, "Platform Version": platformVersion }'`。

- クラスターで Windows サポートを有効にする必要があります。詳細については、「[the section called “Windows サポートを有効にする”](#)」を参照してください。

## ノードに IP アドレスプレフィックスを割り当てる

ノードに IP アドレスプレフィックスを割り当てるようにクラスターを設定します。ノードのオペレーティングシステムに対応する手順を完了します。

## リナックス

1. パラメータを有効にして、Amazon VPC CNI DaemonSet のネットワークインターフェイスにプレフィックスを割り当てます。クラスターをデプロイすると、バージョン 1.10.1 以降の Amazon VPC CNI plugin for Kubernetes アドオンも同時にデプロイされます。IPv6 ファミリーを使用してクラスターを作成している場合、この設定はデフォルトで `true` に設定されています。IPv4 ファミリーを使用してクラスターを作成している場合、この設定はデフォルトで `false` に設定されています。

```
kubectl set env daemonset aws-node -n kube-system ENABLE_PREFIX_DELEGATION=true
```

### Important

サブネットに使用可能な IP アドレスがある場合でも、サブネットに使用可能な /28 個の連続したブロックがない場合には、Amazon VPC CNI plugin for Kubernetes ログに次のエラーが記述されます。

```
InsufficientCidrBlocks: The specified subnet does not have enough free cidr blocks to satisfy the request
```

これは、サブネット全体に広がる既存のセカンダリ IP アドレスの断片化が原因で発生する可能性があります。このエラーを解決するには、新しいサブネットを作成してそこに Pod を起動するか、Amazon EC2 サブネットの CIDR 予約を使用して、プレフィックス割り当てで使用するサブネット内の領域を予約します。詳細については、Amazon VPC ユーザーガイドの「[サブネット CIDR の予約](#)」を参照してください。

2. 前提条件のリストに記載されたバージョン以降の Amazon VPC CNI plugin for Kubernetes を使用しながら、起動テンプレートなしでマネージドノードグループをデプロイする場合、または AMI ID を指定していない起動テンプレートを使用してデプロイする場合には、このまま次のステップに進みます。マネージド型ノードグループは、Pod の最大数を自動的に計算します。

AMI ID を指定した起動テンプレートを使用して、セルフマネージド型ノードグループまたはマネージド型ノードグループをデプロイする場合は、Amazon EKS でノードの推奨最大 Pod 数を決定する必要があります。「[各 Amazon EC2 インスタンスタイプの Amazon EKS の推奨最大ポッド数](#)」の指示に従い、ステップ 3 に `--cni-prefix-delegation-enabled` を追加します。後のステップで使用するために、出力に注意してください。

**⚠ Important**

マネージド型ノードグループは、maxPods の値に最大数を適用します。vCPUs が 30 未満のインスタンスの場合、最大数は 110 で、その他すべてのインスタンスの最大数は 250 です。この最大数は、プレフィックス委任が有効かどうかにかかわらず適用されます。

3. IPv6 用に設定されたクラスターを使用している場合は、このまま次のステップに進みます。

以下のオプションの内の 1 つでパラメータを指定します。どのオプションが適切で、どの値を提供するかを決定するには、GitHub の「[WARM\\_PREFIX\\_TARGET, WARM\\_IP\\_TARGET, and MINIMUM\\_IP\\_TARGET](#)」を参照してください。

*example values* は、ゼロより大きい値で置き換えます。

- WARM\_PREFIX\_TARGET

```
kubectl set env ds aws-node -n kube-system WARM_PREFIX_TARGET=1
```

- WARM\_IP\_TARGET または MINIMUM\_IP\_TARGET – この値を設定した場合、WARM\_PREFIX\_TARGET に対し設定されている値はすべて上書きされます。

```
kubectl set env ds aws-node -n kube-system WARM_IP_TARGET=5
```

```
kubectl set env ds aws-node -n kube-system MINIMUM_IP_TARGET=2
```

4. 少なくとも 1 つの Amazon EC2 Nitro Amazon Linux 2 インスタンスタイプを使用して、次のいずれかのタイプのノードグループを作成します。Nitro インスタンスタイプのリストについては、「Amazon EC2 ユーザーガイド」の「[Instances built on the Nitro System](#)」を参照してください。この機能は Windows ではサポートされていません。**110** が含まれているオプションの場合は、ステップ 3 の値 (推奨) または独自の値に置き換えてください。

- セルフマネージド – 「[セルフマネージド Amazon Linux ノードを作成する](#)」の手順に従い、ノードグループをデプロイします。ブートストラップ引数 パラメータに、以下のテキストを指定します。

```
--use-max-pods false --kubelet-extra-args '--max-pods=110'
```

eksctl を使用してノードグループを作成する場合は、次のコマンドを使用できます。

```
eksctl create nodegroup --cluster my-cluster --managed=false --max-pods-per-node 110
```

- マネージド型 — 次のいずれかのオプションを使用して、ノードグループをデプロイします：
  - 起動テンプレートがない場合、または AMI ID が指定されていない起動テンプレートを使用する場合 – 「[クラスターのマネージドノードグループを作成する](#)」の手順を完了します。マネージド型ノードグループは、Amazon EKS で推奨される max-pods の値を自動的に計算します。
  - 指定した AMI ID を持つ起動テンプレート — 起動テンプレートで Amazon EKS 最適化 AMI ID を指定するか、Amazon EKS 最適化 AMI から構築されたカスタム AMI を指定してから、[起動テンプレートを使用してノードグループをデプロイ](#) 起動テンプレートでは、次のユーザーデータを指定します。このユーザーデータは、引数を bootstrap.sh ファイルに渡します。ブートストラップファイルの詳細については、GitHub の「[bootstrap.sh](#)」を参照してください。

```
/etc/eks/bootstrap.sh my-cluster \
--use-max-pods false \
--kubelet-extra-args '--max-pods=110'
```

eksctl を使用してノードグループを作成する場合は、次のコマンドを使用できます。

```
eksctl create nodegroup --cluster my-cluster --max-pods-per-node 110
```

Amazon EKS 最適化 AMI から構築されていないカスタム AMI を作成した場合は、自分で設定をカスタム作成する必要があります。

#### Note

また、インスタンスとは異なるサブネットの Pod に IP アドレスを割り当てる場合は、このステップで機能を有効化する必要があります。詳細については、「[the section called “カスタムネットワーキング”](#)」を参照してください。

## Windows

1. IP プレフィックスの割り当てを有効にします。
  - a. 編集する amazon-vpc-cni ConfigMap を開きます。



```
kubectl edit configmap -n kube-system amazon-vpc-cni -o yaml
```

- b. data セクションに次の行を追加します。

```
enable-windows-prefix-delegation: "true"
```

- c. ファイルを保存し、エディタを閉じます。
- d. ConfigMap に行が追加されたことを確認します。

```
kubectl get configmap -n kube-system amazon-vpc-cni -o "jsonpath={.data.enable-windows-prefix-delegation}"
```

返された出力が true でない場合は、エラーが発生した可能性があります。ステップをもう一度実行してみてください。

 Important

サブネットに使用可能な IP アドレスがある場合でも、サブネットに使用可能な /28 個の連続したブロックがない場合には、Amazon VPC CNI plugin for Kubernetes ログに次のエラーが記述されます。

```
InsufficientCidrBlocks: The specified subnet does not have enough free cidr blocks to satisfy the request
```

これは、サブネット全体に広がる既存のセカンダリ IP アドレスの断片化が原因で発生する可能性があります。このエラーを解決するには、新しいサブネットを作成してそこに Pod を起動するか、Amazon EC2 サブネットの CIDR 予約を使用して、プレフィックス割り当てで使用するサブネット内の領域を予約します。詳細については、Amazon VPC ユーザーガイドの「[サブネット CIDR の予約](#)」を参照してください。

2. (オプション) クラスターの事前スケーリング動作と動的スケーリング動作を制御するための追加設定を指定します。詳細については、「GitHub」の「[Windows でのプレフィックス委任モードの設定オプション](#)」を参照してください。

- a. 編集する amazon-vpc-cni ConfigMap を開きます。

```
kubectl edit configmap -n kube-system amazon-vpc-cni -o yaml
```



- b. ##### を 0 より大きい値に置き換え、必要なエントリを ConfigMap の data セクションに追加します。warm-ip-target または minimum-ip-target のいずれかに値を設定した場合、その値は warm-prefix-target に設定された値をオーバーライドします。

```
warm-prefix-target: "1"
warm-ip-target: "5"
minimum-ip-target: "2"
```

- c. ファイルを保存し、エディタを閉じます。

3. 少なくとも 1 つの Amazon EC2 Nitro インスタンスタイプで Windows ノードグループを作成します。Nitro インスタンスタイプのリストについては、「Amazon EC2 ユーザーガイド」の「[Instances built on the Nitro System](#)」を参照してください。デフォルトでは、1 つのノードにデプロイできる Pod の最大数は 110 です。この数値を増減させる場合は、ブートストラップ設定のユーザーデータに以下を指定します。*max-pods-quantity* を最大ポッド値に置き換えます。

```
-KubeletExtraArgs '--max-pods=max-pods-quantity'
```

マネージド型ノードグループをデプロイする場合は、この設定を起動テンプレートに追加する必要があります。詳細については、「[the section called “起動テンプレート”](#)」を参照してください。Windows ブートストラップスクリプトの設定パラメータの詳細については、「[the section called “ブートストラップスクリプトの設定パラメータ”](#)」を参照してください。

## 最大 Pod 数と使用可能な IP アドレスを決定する

1. ノードがデプロイされると、クラスター内のノードの表示が可能になります。

```
kubectl get nodes
```

出力例は次のとおりです。

| NAME                                                         | STATUS | ROLES  | AGE | VERSION  |
|--------------------------------------------------------------|--------|--------|-----|----------|
| ip-192-168-22-103.region-code.compute.internal<br>eks-6b7464 | Ready  | <none> | 19m | v1.XX.X- |
| ip-192-168-97-94.region-code.compute.internal<br>eks-6b7464  | Ready  | <none> | 19m | v1.XX.X- |

2. いずれかのノードを記述して、そのノードの max-pods 値と使用可能な IP アドレスの数を決定します。**192.168.30.193** を、前の出力で返されたいずれかのノードの名前の IPv4 アドレスに置き換えます。

```
kubectl describe node ip-192-168-30-193.region-code.compute.internal | grep 'pods\|PrivateIPv4Address'
```

出力例は次のとおりです。

```
pods: 110
vpc.amazonaws.com/PrivateIPv4Address: 144
```

前の出力にある 110 は、**144** 個の IP アドレスが使用可能であるにもかかわらず、Kubernetes がノードにデプロイする Pod の最大数です。

## セキュリティグループを個別の Pod に割り当てる

適用対象: Amazon EC2 インスタンスを持つ Linux ノード

適用対象: プライベートサブネット

Pod のセキュリティグループは、Amazon EC2 セキュリティグループを Kubernetes Pod と統合します。Amazon EC2 セキュリティグループを使用して、多くの Amazon EC2 インスタンスタイプと Fargate で実行されているノードにデプロイする Pod との間で送受信されるインバウンドおよびアウトバウンドネットワークトラフィックを許可するルールを定義できます。この機能の詳細な説明については「[ポッドのセキュリティグループの紹介](#)」の投稿を参照してください。

Amazon VPC CNI plugin for Kubernetes 機能との互換性

Pod のセキュリティグループは、以下の機能で使用できます。

- IPv4 ソースネットワークアドレス変換 - 詳細については「[the section called “アウトバウンドトラフィック”](#)」を参照してください。
- クラスター、ポッド、サービスへの IPv6 アドレス - 詳細については「[the section called “IPv6”](#)」を参照してください。
- Kubernetes ネットワークポリシーを使用したトラフィックの制限 - 詳細については、「[the section called “Kubernetes ポリシー”](#)」を参照してください。

## 考慮事項

Pod のセキュリティグループをデプロイする前に、次の制限事項と条件を考慮してください。

- Pod のセキュリティグループは、Windows ノードでは使用できません。
- Pod のセキュリティグループは、バージョン 1.16.0 以降の Amazon VPC CNI プラグインを使用して、Amazon EC2 ノードを含む IPv6 ファミリー用に設定されたクラスターで使用できます。バージョン 1.7.7 以降の Amazon VPC CNI プラグインを使用すると、Fargate ノードのみを含むクラスター設定の IPv6 ファミリーで Pod のセキュリティグループを使用できます。詳細については、[the section called “IPv6”](#)を参照してください。
- Pod のセキュリティグループはほとんどの [Nitro ベース](#)の Amazon EC2 インスタンスファミリーでサポートされていますが、ファミリーの全世代がサポートしているわけではありません。例えば、m5、c5、r5、m6g、c6g、r6g インスタンスファミリーと世代ではサポートされています。t ファミリーのインスタンスタイプは一切サポートされていません。サポートされているインスタンスタイプの詳細なリストについては、GitHub で [limits.go](#) ファイルを参照してください。ノードはそのファイルで一覧表示されているインスタンスタイプのうち、IsTrunkingCompatible: true を含むものである必要があります。
- カスタムネットワークと Pod のセキュリティグループを組み合わせて使用している場合、ENIConfig で指定されたセキュリティグループではなく、Pod のセキュリティグループによって指定されたセキュリティグループが使用されます。
- Amazon VPC CNI プラグインのバージョン 1.10.2 以前を使用していて、Pod の仕様に terminationGracePeriodSeconds 設定を含める場合は、設定の値を「0」にすることはできません。
- Amazon VPC CNI プラグインのバージョン 1.10 以前、またはデフォルト設定が `POD_SECURITY_GROUP_ENFORCING_MODE=strict` のバージョン 1.11 を使用している場合、externalTrafficPolicy が Local に設定されたインスタンスターゲットを使用するタイプ NodePort および LoadBalancer の Kubernetes サービスは、セキュリティグループを割り当てる Pod ではサポートされていません。インスタンスターゲットでのロードバランサーの使用の詳細については「[the section called “ネットワーク負荷分散”](#)」を参照してください。
- Amazon VPC CNI プラグインのバージョン 1.10 以前、またはデフォルト設定が `POD_SECURITY_GROUP_ENFORCING_MODE=strict` のバージョン 1.11 を使用している場合、セキュリティグループが割り当てられた Pod からのアウトバウンドトラフィックに対してソース NAT が無効になり、その結果アウトバウンドセキュリティグループルールが適用されます。インターネットにアクセスするには、セキュリティグループが割り当てられた Pod を、NAT ゲートウェイまたはインスタンスで構成されたプライベートサブネットにデプロイされたノードで起動す

する必要があります。パブリックサブネットにデプロイされたセキュリティグループが割り当てられたポッドは、インターネットにアクセスできません。

POD\_SECURITY\_GROUP\_ENFORCING\_MODE=standard に設定されたプラグインのバージョン 1.11 以降を使用している場合、VPC の外部を送信先とする Pod トラフィックはインスタンスのプライマリネットワークインターフェイスの IP アドレスに変換されます。このトラフィックには Pod のセキュリティグループ内のルールではなく、プライマリネットワークインターフェイスのセキュリティグループ内のルールが使用されます。

- セキュリティグループが関連付けられている Pod で Calico ネットワークポリシーを使用するには、Amazon VPC CNI プラグインのバージョン 1.11.0 以降を使用し、かつ POD\_SECURITY\_GROUP\_ENFORCING\_MODE=standard に設定することが必要になります。それ以外の場合は、セキュリティグループが関連付けられている Pod との間のトラフィックフローに Calico ネットワークポリシーは適用されず、Amazon EC2 セキュリティグループの適用のみに限定されます。Amazon VPC CNI バージョンを更新するには「[the section called “Amazon VPC CNI”](#)」を参照してください
- [NodeLocal DNSCache](#) を使用するクラスターのセキュリティグループを使用する Amazon EC2 ノードで実行されている Pod は、Amazon VPC CNI プラグインのバージョンが 1.11.0 以降で POD\_SECURITY\_GROUP\_ENFORCING\_MODE=standard に設定されている場合にのみサポートされます。Amazon VPC CNI プラグインのバージョンを更新するには「[the section called “Amazon VPC CNI”](#)」を参照してください
- Pod のセキュリティグループは、解約率が高い Pod の場合、Pod の起動レイテンシーがより高くなる可能性があります。これはリソースコントローラーのレート制限によるものです。
- EC2 セキュリティグループのスコープは Pod レベル - 詳細については「[セキュリティグループ](#)」を参照してください。

POD\_SECURITY\_GROUP\_ENFORCING\_MODE=standard および AWS\_VPC\_K8S\_CNI\_EXTERNALSNAT=false を設定した場合、VPC 外部のエンドポイント宛てのトラフィックは、Pod のセキュリティグループではなくノードのセキュリティグループを使用します。

Amazon EKS Pod のセキュリティグループ用に Amazon VPC CNI Plugin for Kubernetes を設定する

Amazon EC2 インスタンスで Pod を使用する場合は、セキュリティグループ用に Amazon VPC CNI Plugin for Kubernetes を設定する必要があります。

Fargate Pod のみを使用し、クラスターに Amazon EC2 ノードがない場合は、「[the section called “SecurityGroupPolicy”](#)」を参照してください。

1. 次のコマンドを使用して、現在の Amazon VPC CNI Plugin for Kubernetes のバージョンをチェックします。

```
kubectl describe daemonset aws-node --namespace kube-system | grep amazon-k8s-cni: |
cut -d : -f 3
```

出力例は次のとおりです。

```
v1.7.6
```

Amazon VPC CNI Plugin for Kubernetes のバージョンが 1.7.7 より前の場合は、プラグインをバージョン 1.7.7 以降に更新してください。詳細については、「[the section called “Amazon VPC CNI”](#)」を参照してください。

2. [AmazonEKSVPCResourceController](#) マネージド IAM ポリシーを Amazon EKS クラスターに関連付けられている[クラスターロール](#)に追加します。ポリシーにより、ロールはネットワークインターフェイス、プライベート IP アドレス、およびネットワークインスタンスへのアタッチとデタッチを管理できます。
  - a. クラスター IAM ロールの名前を取得し、変数に格納します。##### の部分は自分のクラスター名に置き換えます。

```
cluster_role=$(aws eks describe-cluster --name my-cluster --query cluster.roleArn
--output text | cut -d / -f 2)
```

- b. ロールへのポリシーの付与

```
aws iam attach-role-policy --policy-arn arn:aws:iam::aws:policy/
AmazonEKSVPCResourceController --role-name $cluster_role
```

3. Amazon VPC CNI アドオンを有効にして Pod のネットワークインターフェイスを管理するには、aws-node DaemonSet で ENABLE\_POD\_ENI 変数を true に設定します。この設定が true になると、クラスター内の各ノードについて、アドオンにより cninode カスタムリソースが作成されます。VPC リソースコントローラーは、1 つの特別なネットワークインターフェイスを作成してアタッチします。これは、トランクネットワークインターフェイスと呼ばれ、説明は aws-k8s-trunk-eni です。

```
kubectl set env daemonset aws-node -n kube-system ENABLE_POD_ENI=true
```

 Note

トランクネットワークインターフェイスは、インスタンスタイプでサポートされているネットワークインターフェイスの最大数に含まれます。各インスタンスタイプによりサポートされるネットワークインターフェイスの最大数のリストについては、「Amazon EC2 ユーザーガイド」の「[各インスタンスタイプのネットワークインターフェイスあたりの IP アドレス数](#)」を参照してください。ノードにすでに最大数の標準ネットワークインターフェイスがアタッチされている場合、VPC リソースコントローラーはスペースを予約します。コントローラーが標準ネットワークインターフェイスをデタッチして削除し、トランクネットワークインターフェイスを作成し、インスタンスにアタッチできるように、実行中の Pod をスケールダウンする必要があります。

4. CNINode カスタムリソースがどのノードにあるか、次のコマンドで確認できます。[No resources found] が返された場合、数秒待ってから、もう一度試してください。前のステップで、Amazon VPC CNI Plugin for Kubernetes Pod を再起動する必要があります。再起動には数秒かかります。

```
kubectl get cninode -A
NAME FEATURES
ip-192-168-64-141.us-west-2.compute.internal [{"name":"SecurityGroupsForPods"}]
ip-192-168-7-203.us-west-2.compute.internal [{"name":"SecurityGroupsForPods"}]
```

1.15 より古いバージョンの VPC CNI バージョンを使用している場合は、CNINode カスタムリソースの代わりにノードラベルが使用されていました。true に設定されているノードラベル `aws-k8s-trunk-eni` がどのノードにあるか、次のコマンドで確認できます。[No resources found] が返された場合、数秒待ってから、もう一度試してください。前のステップで、Amazon VPC CNI Plugin for Kubernetes Pod を再起動する必要があります。再起動には数秒かかります。

```
kubectl get nodes -o wide -l vpc.amazonaws.com/has-trunk-attached=true
-
```

トランクネットワークインターフェイスが作成されると、Pod にはトランクネットワークインターフェイスまたは標準ネットワークインターフェイスのセカンダリ IP アドレスが割り当てられます。ノードが削除されると、トランクインターフェイスは自動的に削除されます。

後のステップで Pod のセキュリティグループをデプロイすると、VPC リソースコントローラーはブランチネットワークインターフェイスと呼ばれる特別なインターフェイスを `aws-k8s-`

branch-eni の説明と共に作成し、セキュリティグループを関連付けます。ノードにアタッチされた標準ネットワークインターフェイスとトランクネットワークインターフェイスに加えて、ブランチネットワークインターフェイスが作成されます。

Liveness プローブまたは Readiness プローブを使用している場合は、TCP Early Demux も無効にする必要があります。これにより、kubelet は TCP を使用してブランチネットワークインターフェイス上の Pod に接続できます。TCP Early Demux を無効にするには、次のコマンドを実行します。

```
kubectl patch daemonset aws-node -n kube-system \
 -p '{"spec": {"template": {"spec": {"initContainers": [{"env": [{"name": "DISABLE_TCP_EARLY_DEMUX", "value": "true"}], "name": "aws-vpc-cni-init"}]}}}'
```

#### Note

Amazon VPC CNI Plugin for Kubernetes アドオンのバージョン 1.11.0 以降を使用し、かつ `POD_SECURITY_GROUP_ENFORCING_MODE=standard` に設定している場合、次のステップで説明されているように、前のコマンドを実行する必要はありません。

5. クラスターで NodeLocal DNSCache が使用されている場合、独自のセキュリティグループを持つ Pod で Calico ネットワークポリシーを使用する場合、またはセキュリティグループを割り当てる Pod に対して externalTrafficPolicy を Local に設定したインスタンスターゲットを使用するタイプ NodePort および LoadBalancer の Kubernetes サービスがある場合、バージョン 1.11.0 以降の Amazon VPC CNI Plugin for Kubernetes アドオンを使用して次の設定を有効にする必要があります。

```
kubectl set env daemonset aws-node -n kube-system
 POD_SECURITY_GROUP_ENFORCING_MODE=standard
```

**重要:** Pod のセキュリティグループルールは、**kubelet** や **nodeLocalDNS** など、同じノードにある Pod とサービスの間のトラフィックまたは Pod 間のトラフィックには適用されません。同じノードで異なるセキュリティグループを使用するポッドは、異なるサブネットに設定されているため通信できません。また、これらのサブネット間のルーティングは無効になっています。Pod から VPC の外部のアドレスへのアウトバウンドトラフィックは、インスタンスのプライマリネットワークインターフェイスの IP アドレスに変換されたネットワークアドレスです (AWS\_VPC\_K8S\_CNI\_EXTERNALSNAT=true に設定していない場合)。このトラフィックには Pod のセキュリティグループ内のルールではなく、プライマリネットワークインターフェイスのセ



セキュリティグループ内のルールが使用されます。\*\* この設定を既存の Pod に適用するには、Pod または Pod が実行されているノードを再起動する必要があります。

6. Pod のセキュリティグループポリシーの使用方法については、「[the section called “SecurityGroupPolicy”](#)」を参照してください。

## Amazon EKS Pod のセキュリティグループポリシーを使用する

Pod のセキュリティグループを使用するには、既存のセキュリティグループが必要です。次のステップは、Pod に対してセキュリティグループポリシーを使用する方法を示しています。次のステップではターミナル間で保持されない変数が使用されるため、特に明記されていない限り、同じターミナルからすべてのステップを完了してください。

Amazon EC2 インスタンスを持つ Pod がある場合は、この手順を使用する前にプラグインを設定する必要があります。詳細については、「[the section called “構成する”](#)」を参照してください。

1. リソースをデプロイする Kubernetes 名前空間を作成します。*my-namespace* は使用する名前空間の名前に置き換えることができます。

```
kubectl create namespace my-namespace
```

2. Amazon EKS SecurityGroupPolicy をクラスターにデプロイします。
  - a. 次のコンテンツをデバイスにコピーします。サービスアカウントラベルに基づいて Pod を選択したい場合は、*podSelector* を *serviceAccountSelector* で置き換えることができます。セレクターをどちらか 1 つ指定する必要があります。podSelector が空 (例: podSelector: {}) であると、名前空間内のすべての Pod が選択されます。*my-role* は自分の役割名に変更できます。serviceAccountSelector が空であると、名前空間内のすべてのサービスアカウントが選択されます。*my-security-group-policy* を自分の SecurityGroupPolicy の名前に置き換え、*my-namespace* は SecurityGroupPolicy を作成する名前空間に置き換えることができます。

*my\_pod\_security\_group\_id* を既存のセキュリティグループの ID に置き換える必要があります。既存のセキュリティグループがない場合は作成する必要があります。詳細については「[Amazon EC2 ユーザーガイド](#)」の「[Amazon EC2 security groups for Linux instances](#)」(Linux インスタンス用の Amazon EC2 セキュリティグループ) を参照してください。1~5 個のセキュリティグループ ID を指定できます。複数の ID を指定した場合、すべてのセキュリティグループ内のすべてのルールの組み合わせが、選択した Pod に対して有効になります。

```
cat >my-security-group-policy.yaml <<EOF
```



```
apiVersion: vpcresources.k8s.aws/v1beta1
kind: SecurityGroupPolicy
metadata:
 name: my-security-group-policy
 namespace: my-namespace
spec:
 podSelector:
 matchLabels:
 role: my-role
 securityGroups:
 groupIds:
 - my_pod_security_group_id
EOF
```

### Important

Pod に対して指定した 1 つまたは複数のセキュリティグループは次の基準を満たす必要があります。

- 存在している必要があります。存在しない場合、セレクトターに一致する Pod をデプロイすると、Pod は作成プロセスでスタックしたままになります。Pod の説明を書き込むと、次のようなエラーメッセージが表示されます: An error occurred (InvalidSecurityGroupID.NotFound) when calling the CreateNetworkInterface operation: The securityGroup ID '*sg-05b1d815d1EXAMPLE*' does not exist.
- プローブを設定した任意のポートを介して、ノードに適用されたセキュリティグループ (kubelet の場合) からのインバウンド通信を許可する必要があります。
- TCP および UDP ポート 53 番経由のアウトバウンド通信を、CoreDNS を実行している Pod (または Pod が実行されるノード) に割り当てられたセキュリティグループへ許可する必要があります。CoreDNS Pod のセキュリティグループは、指定したセキュリティグループからのインバウンド TCP および UDP ポート 53 番トラフィックを許可する必要があります。
- セキュリティグループには、通信を行う必要がある他の Pod との通信に必要なインバウンドルールとアウトバウンドルールを含める必要があります。
- Fargate でセキュリティグループを使用している場合は、セキュリティグループに、Pod と Kubernetes コントロールプレーンとの通信を許可するルールを含める必要があります。最も簡単な方法はクラスターセキュリティグループをセキュリティグループの 1 つとして指定することです。

セキュリティグループポリシーは、新しくスケジュールされた Pod にのみ適用されます。実行中の Pod には影響しません。

- b. ポリシーをデプロイします。

```
kubectl apply -f my-security-group-policy.yaml
```

3. 前のステップで指定した *podSelector* の *my-role* の値に一致するラベルを持つサンプルアプリケーションをデプロイします。

- a. 次のコンテンツをデバイスにコピーします。####を独自のものに置き換えて、変更したコマンドを実行してください。*my-role* を置き換える場合は前のステップでセクターに指定した値と同じであることを確認してください。

```
cat >sample-application.yaml <<EOF
apiVersion: apps/v1
kind: Deployment
metadata:
 name: my-deployment
 namespace: my-namespace
 labels:
 app: my-app
spec:
 replicas: 4
 selector:
 matchLabels:
 app: my-app
 template:
 metadata:
 labels:
 app: my-app
 role: my-role
 spec:
 terminationGracePeriodSeconds: 120
 containers:
 - name: nginx
 image: public.ecr.aws/nginx/nginx:1.23
 ports:
 - containerPort: 80

apiVersion: v1
kind: Service
metadata:
```

```

name: my-app
namespace: my-namespace
labels:
 app: my-app
spec:
 selector:
 app: my-app
 ports:
 - protocol: TCP
 port: 80
 targetPort: 80
EOF

```

- b. 次のコマンドを使用して、アプリケーションをデプロイします。アプリケーションをデプロイすると、Amazon VPC CNI Plugin for Kubernetes が role ラベルと一致し、前のステップで指定したセキュリティグループが Pod に適用されます。

```
kubectl apply -f sample-application.yaml
```

4. サンプルアプリケーションでデプロイされた Pod を表示します。このトピックの以降の説明ではこのターミナルを TerminalA と呼びます。

```
kubectl get pods -n my-namespace -o wide
```

出力例は次のとおりです。

| NAME                                          | READY | STATUS  | RESTARTS  | AGE   | IP              |
|-----------------------------------------------|-------|---------|-----------|-------|-----------------|
| NODE                                          |       |         | NOMINATED | NODE  | READINESS GATES |
| my-deployment-5df6f7687b-4fbjm                | 1/1   | Running | 0         | 7m51s | 192.168.53.48   |
| ip-192-168-33-28.region-code.compute.internal |       |         | <none>    |       | <none>          |
| my-deployment-5df6f7687b-j9fl4                | 1/1   | Running | 0         | 7m51s | 192.168.70.145  |
| ip-192-168-92-33.region-code.compute.internal |       |         | <none>    |       | <none>          |
| my-deployment-5df6f7687b-rjxcz                | 1/1   | Running | 0         | 7m51s | 192.168.73.207  |
| ip-192-168-92-33.region-code.compute.internal |       |         | <none>    |       | <none>          |
| my-deployment-5df6f7687b-zmb42                | 1/1   | Running | 0         | 7m51s | 192.168.63.27   |
| ip-192-168-33-28.region-code.compute.internal |       |         | <none>    |       | <none>          |

#### Note

Pod がスタックしている場合は次のヒントをお試しください。

- Waiting 状態でスタックした Pod があれば、`kubectl describe pod my-deployment-xxxxxxxxxx-xxxxx -n my-namespace` を実行してください。Insufficient permissions: Unable to create Elastic Network Interface. が表示された場合、前のステップで IAM クラスター役割に IAM ポリシーを追加したことを確認します。
- Pod が Pending 状態でスタックした場合、ノードのインスタンスタイプが [limits.go](https://aws.amazon.com/ec2/instance-types/) のリストに含まれていることを確認します。そのうえで、インスタンスタイプでサポートされるブランチネットワークインターフェイスの最大製品数とノードグループ内のノード数を乗算した数にまだ達していないことを確認します。例えば、m5.large インスタンスでは9つのブランチネットワークインターフェイスがサポートされています。ノードグループに5つのノードがある場合、ノードグループに対して最大45のブランチネットワークインターフェイスを作成できます。46番目のPodをデプロイしようとすると、セキュリティグループに関連付けられた別のPodが削除されるまで Pending 状態のままになります。

`kubectl describe pod my-deployment-xxxxxxxxxx-xxxxx -n my-namespace` を実行したときに次のようなメッセージが表示されている場合は、無視しても問題ありません。このメッセージは、Amazon VPC CNI Plugin for Kubernetes がホストネットワークの設定を試み、ネットワークインターフェイスの作成中に失敗したときに表示される場合があります。プラグインはネットワークインターフェイスが作成されるまで、このイベントをログに記録します。

```
Failed to create Pod sandbox: rpc error: code = Unknown desc = failed to set up
sandbox container "e24268322e55c8185721f52df6493684f6c2c3bf4fd59c9c121fd4cdc894579f"
network for Pod "my-deployment-5df6f7687b-4fbjm": networkPlugin
cni failed to set up Pod "my-deployment-5df6f7687b-4fbjm-c89wx_my-namespace" network:
add cmd: failed to assign an IP address to container
```

インスタンスタイプで実行できる Pod の最大数を超えることはできません。各インスタンスタイプで実行できるポッドの最大数の一覧については、GitHub の「[eni-max-pods.txt](#)」を参照してください。セキュリティグループが関連付けられているポッドを削除するか、ポッドが実行されているノードを削除すると、VPC リソースコントローラーによってブランチネットワークインターフェイスが削除されます。セキュリティグループ用に Pod を含むクラスターを Pod で削除する場合、コントローラーではブランチネットワークインターフェイスを削除しないため、自分で削除する必要があります。ネットワークインターフェイスの削除方法の詳細については「Amazon EC2 ユーザーガイド」の「[ネットワークインターフェイスの削除](#)」を参照してください。

- 別のターミナルで、Pod のいずれかをシェルで操作します。このトピックの以降の説明ではこのターミナルを TerminalB と呼びます。`5df6f7687b-4fbjm` を、前の手順の出力で返された Pod のいずれかの ID に置き換えます。

```
kubectl exec -it -n my-namespace my-deployment-5df6f7687b-4fbjm -- /bin/bash
```

- TerminalB のシェルから、サンプルアプリケーションが動作することを確認します。

```
curl my-app
```

出力例は次のとおりです。

```
<!DOCTYPE html>
<html>
<head>
<title>Welcome to nginx!</title>
[...]
```

アプリケーションを実行しているすべての Pod が作成したセキュリティグループに関連付けられているため、出力を受信できます。そのグループにはセキュリティグループが関連付けられているすべての Pod 間のすべてのトラフィックを許可するルールが含まれています。DNS トラフィックはそのセキュリティグループからノードに関連付けられているクラスターセキュリティグループへのアウトバウンドで許可されます。ノードは、Pod が名前のルックアップを実行した CoreDNS Pod を実行しています。

- TerminalA で、クラスターセキュリティグループへの DNS 通信を許可するセキュリティグループルールを、セキュリティグループから削除します。前のステップで DNS ルールをクラスターセキュリティグループに追加しなかった場合は `$my_cluster_security_group_id` を、ルールを作成したセキュリティグループの ID で置き換えます。

```
aws ec2 revoke-security-group-ingress --group-id $my_cluster_security_group_id --
security-group-rule-ids $my_tcp_rule_id
aws ec2 revoke-security-group-ingress --group-id $my_cluster_security_group_id --
security-group-rule-ids $my_udp_rule_id
```

- TerminalB で、アプリケーションにもう一度アクセスしてみます。

```
curl my-app
```

出力例は次のとおりです。

```
curl: (6) Could not resolve host: my-app
```

クラスターセキュリティグループが関連付けられている CoreDNS Pod に Pod がアクセスできなくなったため、この試行は失敗します。クラスターセキュリティグループから、Pod に関連付けられたセキュリティグループからの DNS 通信を許可するセキュリティグループルールがなくなりました。

前のステップで、いずれかの Pod に返された IP アドレスを使用してアプリケーションにアクセスしようとする、セキュリティグループが関連付けられており、名前のルックアップが不要な Pod 間ですべてのポートが許可されているため、引き続き応答を受信します。

9. 試行し終わったら、作成したサンプルのセキュリティグループポリシー、アプリケーション、およびセキュリティグループを削除できます。TerminalA から、次のコマンドを実行してください。

```
kubectl delete namespace my-namespace
aws ec2 revoke-security-group-ingress --group-id $my_pod_security_group_id --
security-group-rule-ids $my_inbound_self_rule_id
wait
sleep 45s
aws ec2 delete-security-group --group-id $my_pod_security_group_id
```

## Multus を使用して複数のネットワークインターフェイスを Pod にアタッチする

Multus CNI は、Amazon EKS 用のコンテナネットワークインターフェイス (CNI) プラグインです。これにより、Pod に複数のネットワークインターフェイスをアタッチできるようになります。詳細については、GitHub の [Multus-CNI](#) を参照してください。

Amazon EKS では、Amazon VPC CNI プラグインによって割り当てられたネットワークインターフェイスが Pod ごとに 1 つあります。Multus を使用すると、複数のインターフェイスを備えたマルチホーム Pod を作成できます。これは、Multus が「meta-plugin」(他の複数の CNI プラグインを呼び出すことができる CNI プラグイン) として機能することによって達成されます。AWS では、Amazon VPC CNI プラグインをデフォルトのデリゲートプラグインとして設定することで、Multus をサポートしています。

- Amazon EKS では、シングルルートの I/O 仮想化 (SR-IOV)、およびデータプレーン開発キット (DPDK) の CNI プラグインを構築および公開することはありません。ただし、Multus で管理され

たホストデバイスおよび `ipvlan` プラグインを介して、Amazon EC2 Elastic Network Adapters (ENA) と直接接続することで、パケットアクセラレーションを実現することができます。

- Amazon EKS では Multus をサポートすることで、追加の CNI プラグインの簡単な連鎖を可能にする汎用プロセスを提供します。AWS は、Multus とその連鎖プロセスをサポートしています。ただし、連鎖可能な互換性のある CNI プラグインすべてについてのサポートや、連鎖設定とは無関係な CNI プラグインで発生する可能性のある問題のサポートは行っていません。
- Amazon EKS は Multus プラグインのサポートとライフサイクル管理を提供していますが、追加のネットワークインターフェイスに関連付けられた IP アドレスや他の管理については責任を負いません。Amazon VPC CNI プラグインを使用する、デフォルトのネットワークインターフェイスの IP アドレスと管理に変更はありません。
- デフォルトのデリゲートプラグインとして正式にサポートされるのは、Amazon VPC CNI プラグインのみです。Amazon VPC CNI プラグインをプライマリネットワークに使用しないことを選択した場合は、公開されている Multus インストールマニフェストを変更して、デフォルトのデリゲートプラグインを代替の CNI に再設定する必要があります。
- Multus は、Amazon VPC CNI をプライマリ CNI として使用する場合にのみサポートされます。セカンダリなど、高次のインターフェイスに使用する場合、Amazon VPC CNI はサポートされません。
- Pod に追加で割り当てたネットワークインターフェイスが Amazon VPC CNI プラグインの管理対象にならないようにするには、そのネットワークインターフェイスに次のタグを追加します。

key

```
: node.k8s.amazonaws.com/no_manage
```

value

```
: true
```

- Multus にはネットワークポリシーとの互換性がありますが、Pod に追加でアタッチされるネットワークインターフェイスの一部であるポートと IP アドレスを含めるように、ポリシーを強化する必要があります。

実装のチュートリアルについては、GitHub の [Multus セットアップガイド](#) を参照してください。

## Amazon EKS クラスターの代替 CNI プラグイン

[Amazon VPC CNI plugin for Kubernetes](#) は、Amazon EC2 ノードにおいて Amazon EKS によりサポートされている唯一の CNI プラグインです。Amazon EKS は、Amazon EKS Hybrid Nodes 用の Cilium および Calico のコア機能をサポートしています。Amazon EKS では、アップストリーム



Kubernetes が実行されており、クラスター内の Amazon EC2 ノードに互換性のある CNI プラグインをインストールできます。クラスターに Fargate ノードがある場合、Amazon VPC CNI plugin for Kubernetes は既に Fargate ノードに存在します。これは Fargate ノードで使用できる唯一の CNI プラグインです。Fargate ノードに代替 CNI プラグインをインストールしようとすると失敗します。

Amazon EC2 ノードで代替 CNI プラグインを使用する予定であれば、当該プラグインの商用サポートを受けるか、社内のエキスパートによるトラブルシューティングを行い、解決策を CNI プラグインプロジェクトに提供することをお勧めします。

Amazon EKS は、互換性のある代替 CNI プラグインのサポートを提供するパートナーネットワークと連携しています。バージョンと認定、および実行されたテストの詳細については、次のパートナーのドキュメントを参照してください。

| パートナー     | 製品                                                  | ドキュメント                   |
|-----------|-----------------------------------------------------|--------------------------|
| Tigera    | <a href="#">Calico</a>                              | <a href="#">インストール手順</a> |
| Isovalent | <a href="#">Cilium</a>                              | <a href="#">インストール手順</a> |
| ジュニパー     | <a href="#">クラウドネイティブ Contrail Networking (CN2)</a> | <a href="#">インストール手順</a> |
| VMware    | <a href="#">Antrea</a>                              | <a href="#">インストール手順</a> |

Amazon EKS ではすべてのユースケースをカバーする幅広いオプションの提供を目指しています。

## 互換性のある代替ネットワークポリシープラグイン

[Calico](#) は、コンテナネットワークとセキュリティのために広く採用されているソリューションです。Calico on EKS を使用すると、EKS クラスターに完全準拠のネットワークポリシーが適用されます。さらに、基盤となる VPC の IP アドレスを節約する Calico のネットワークを使用することもできます。[Calico Cloud](#) は Calico Open Source の機能を強化すると共に、高度なセキュリティとオペレータビリティ機能を提供します。

関連付けられたセキュリティグループを持つポッドとの間のトラフィックフローに、Calico ネットワークポリシーは適用されず、Amazon VPC セキュリティグループの適用のみに限定されます。

Calico ネットワークポリシーの適用を使用する場合は、Kubernetes の既知の問題を回避するために、環境変数 `ANNOTATE_POD_IP` を `true` に設定することをお勧めします。この機能を使用するに



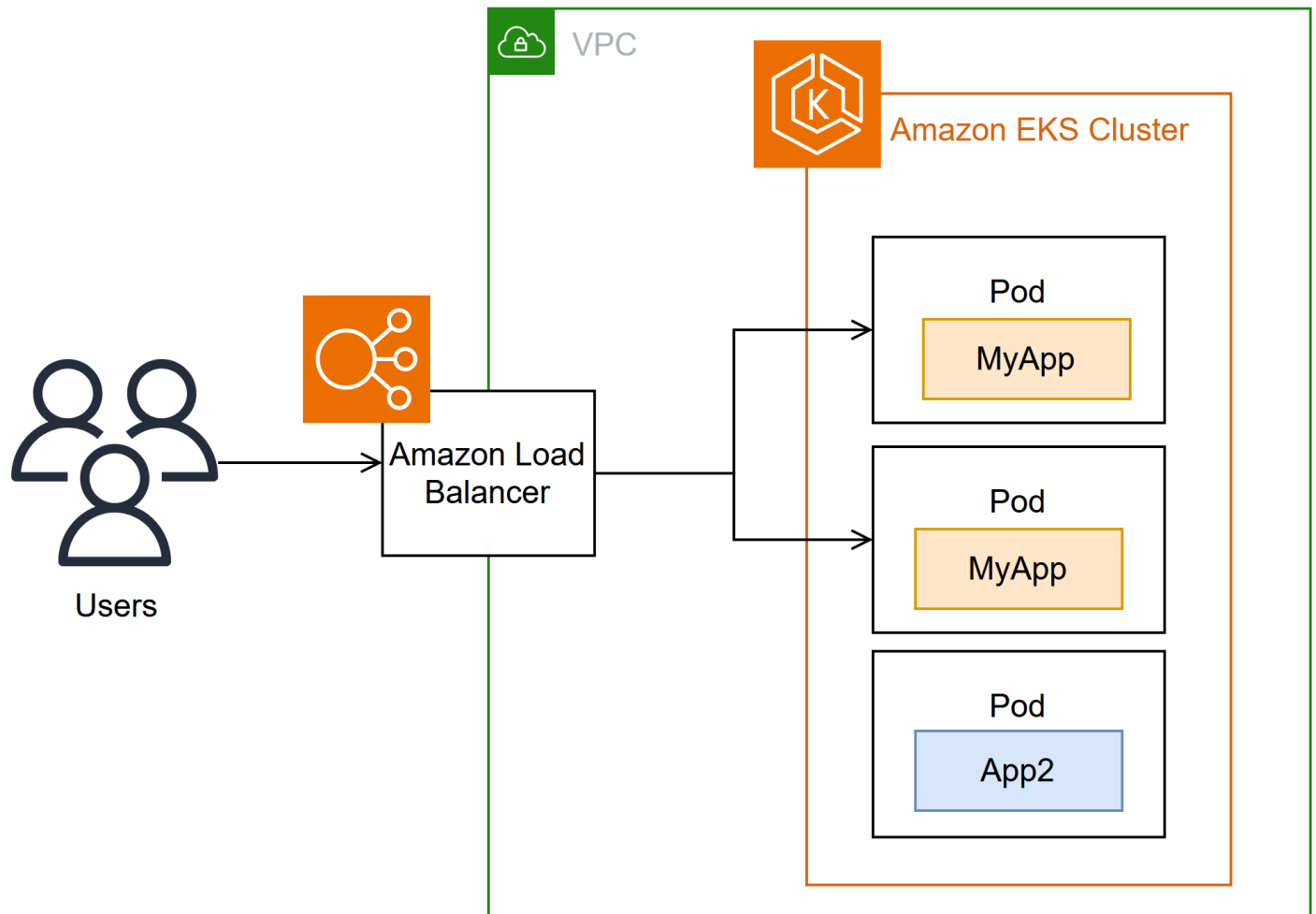
は、ポッドの patch のアクセス許可を aws-node ClusterRole に追加する必要があります。aws-node DaemonSet にパッチのアクセス許可を追加すると、プラグインのセキュリティ範囲が広がることに注意してください。詳細については、GitHub の VPC CNI リポジトリの [ANNOTATE\\_POD\\_IP](#) を参照してください。

## Amazon EKS Auto Mode の考慮事項

Amazon EKS Auto Mode は、代替 CNI プラグインおよびネットワークポリシープラグインをサポートしていません。詳細については、「[EKS Auto Mode](#)」を参照してください。

## AWS Load Balancer Controller を使用してインターネットトラフィックをルーティングする

AWS Load Balancer Controller により、Kubernetes クラスター向けの AWS Elastic Load Balancer を管理できます。コントローラーを使用すると、クラスターアプリケーションをインターネットに公開できます。コントローラーは、クラスターサービスまたは Ingress リソースを指す AWS ロードバランサーをプロビジョニングします。つまり、コントローラーはクラスター内の複数のポッドを指す単一の IP アドレスまたは DNS 名を作成します。



コントローラーは、Kubernetes Ingress または Service のリソースを監視します。これに応じて、適切な AWS Elastic Load Balancing リソースが作成されます。Kubernetes リソースに注釈を適用することで、ロードバランサーの特定の動作を設定できます。例えば、注釈を使用してロードバランサーに AWS セキュリティグループをアタッチできます。

コントローラーは、以下のリソースをプロビジョニングします。

## Kubernetes Ingress

Kubernetes Ingress を作成すると、LBC は [AWS Application Load Balancer \(ALB\)](#) を作成します。[Ingress リソースに適用できる注釈を確認してください。](#)

## LoadBalancer タイプの Kubernetes サービス

LoadBalancer タイプの Kubernetes サービスを作成すると、LBC は [AWS Network Load Balancer \(NLB\)](#) を作成します。[サービスリソースに適用できる注釈を確認してください。](#)

以前は、インスタンスターゲットには Kubernetes Network Load Balancer が使用されていましたが、IP ターゲットには LBC が使用されていました。AWS Load Balancer Controller バージョン 2.3.0 以降では、いずれかのターゲットタイプを使用して NLBs を作成できます。NLB ターゲットタイプの詳細については、Network Load Balancer のユーザーガイドの「[ターゲットタイプ](#)」を参照してください。

コントローラーは GitHub で管理される [オープンソースプロジェクト](#) です。

コントローラーをデプロイする前に、「[Application Load Balancers を使用したアプリケーションおよび HTTP トラフィックのルーティング](#)」および [the section called “ネットワーク負荷分散”](#) で前提条件と考慮事項を確認しておくことをお勧めします。これらのトピックでは、AWS ロードバランサーを含むサンプルアプリケーションをデプロイしています。

## コントローラーのインストール

次のどちらかの手順を使用して AWS Load Balancer Controller をインストールできます。

- Amazon EKS を初めて使用する場合は、AWS Load Balancer Controller のインストールが簡素化されるため、Helm を使用してインストールすることをお勧めします。詳細については、「[the section called “Helm を使用してインストールする”](#)」を参照してください。
- パブリックコンテナレジストリへのネットワークアクセスが制限されているクラスターなどの高度な設定には、Kubernetes マニフェストを使用します。詳細については、「[the section called “マニフェストを使用してインストールする”](#)」を参照してください。

## 非推奨のコントローラーバージョンから移行する

- 非推奨バージョンの AWS Load Balancer Controller がインストールされている場合は、「[the section called “非推奨からの移行”](#)」を参照してください。
- 非推奨バージョンはアップグレードできません。このバージョンを削除し、AWS Load Balancer Controller の最新バージョンをインストールする必要があります。
- 非推奨バージョンには以下が含まれます。
  - AWS Load Balancer Controller の前身である AWS ALB Ingress Controller for Kubernetes (「Ingress Controller」)。
  - AWS Load Balancer Controller の任意の 0.1.x バージョン

## レガシークラウドプロバイダー

Kubernetes には、AWS のレガシークラウドプロバイダーが含まれています。レガシークラウドプロバイダーは AWS Load Balancer Controller と同様に、AWS ロードバランサーをプロビジョニングできます。レガシークラウドプロバイダーは Classic Load Balancer を作成します。AWS Load Balancer Controller をインストールしない場合、Kubernetes はデフォルトでレガシークラウドプロバイダーを使用します。AWS Load Balancer Controller をインストールして、レガシークラウドプロバイダーを使用しないようにしてください。

### Important

バージョン 2.5 以降では、AWS Load Balancer Controller は、`type: LoadBalancer` を持つ Kubernetes サービスリソースのデフォルトコントローラーとなり、サービスごとに AWS Network Load Balancer (NLB) を作成します。これは、サービスの変更ウェブフックを変化することで実行され、これにより `type: LoadBalancer` の新しいサービスの `spec.loadBalancerClass` フィールドが `service.k8s.aws/nlb` に設定されます。Helm チャートの値 `enableServiceMutatorWebhook` を `false` に設定すると、この機能をオフにして、[レガシークラウドプロバイダー](#) をデフォルトのコントローラーとして使用するように戻すことができます。この機能をオフにしない限り、クラスターはサービスに新しい Classic Load Balancer をプロビジョニングしません。既存の Classic Load Balancer は従来どおり機能します。

## Helm による AWS Load Balancer Controller のインストール

### Tip

Amazon EKS Auto Mode では、ネットワーク形成のアドオンをインストールまたはアップグレードする必要はありません。自動モードにはポッドのネットワーク形成とロードバランシング機能が含まれています。

詳細については、「[EKS Auto Mode](#)」を参照してください。

このトピックでは、Kubernetes のパッケージマネージャーである Helm と `eksctl` を使用して AWS Load Balancer Controller をインストールする方法について説明します。コントローラーはデフォルトのオプションでインストールされます。注釈を使用した設定などコントローラーの詳細については、GitHub で「[AWS Load Balancer Controller ドキュメント](#)」を参照してください。

以下のステップでは、####を独自の値に置き換えます。

## 前提条件

このチュートリアルを開始する前に、以下のステップを完了する必要があります。

- Amazon EKS クラスターを作成します。作成する場合は「[はじめに](#)」を参照してください。
- ローカルマシンに [Helm](#) をインストールします。
- Amazon VPC CNI plugin for Kubernetes、kube-proxy、および CoreDNS アドオンが、[サービス アカウントトークン](#)に記載されている最小のバージョンであることを確認してください。
- AWS Elastic Load Balancing の概念を確認します。詳細については、[Elastic Load Balancing ユーザーガイド](#)を参照してください。
- Kubernetes [サービス](#)と [Ingress](#) リソースについて確認します。

## 考慮事項

このページの設定手順に進む前に、以下の点について考慮してください。

- IAM ポリシーとロール (AmazonEKSLoadBalancerControllerRole) は、同じ AWS アカウントの複数の EKS クラスターで再利用できます。
- ロール (AmazonEKSLoadBalancerControllerRole) が最初に作成されたのと同じクラスターにコントローラーをインストールする場合は、ロールが存在することを確認した後、「[ステップ 2: Load Balancer Controller のインストール](#)」に進みます。
- サービスアカウントの IAM ロール (IRSA) を使用している場合、IRSA はクラスターごとに設定する必要があり、ロールの信頼ポリシー内の OpenID Connect (OIDC) プロバイダー ARN は各 EKS クラスターに固有です。さらに、既存の AmazonEKSLoadBalancerControllerRole を使用して新しいクラスターにコントローラーをインストールする場合は、そのロールの信頼ポリシーを更新して新しいクラスターの OIDC プロバイダーを追加し、適切なロール注釈を持つ新しいサービスアカウントを作成します。OIDC プロバイダーが既に存在しているかどうかを確認する、または OIDC プロバイダーを作成するには、「[the section called "IAM OIDC プロバイダー"](#)」を参照してください。

## ステップ 1: eksctl を使用して IAM ロールを作成する

次の手順は、AWS Load Balancer Controller v2.13.0 リリースバージョンに関するものです。すべてのリリースの詳細については、GitHub で「[AWS Load Balancer Controller Release Page](#)」を参照してください。

1. ユーザーに代わって AWS API を呼び出すことを許可する、AWS Load Balancer Controller 用の IAM ポリシーをダウンロードします。

```
curl -O https://raw.githubusercontent.com/kubernetes-sigs/aws-load-balancer-controller/v2.13.0/docs/install/iam_policy.json
```

- AWS パーティションが政府リージョンや中国リージョンなど標準以外のものである場合は、[GitHub でポリシーを確認](#)し、ご使用のリージョンに適切なポリシーをダウンロードします。
2. 前のステップでダウンロードしたポリシー を使用して、IAM ポリシーを作成します。

```
aws iam create-policy \
 --policy-name AWSLoadBalancerControllerIAMPolicy \
 --policy-document file://iam_policy.json
```

#### Note

AWS Management Console でポリシーを確認すると、コンソールには [ELB] サービスに関する警告が表示されますが、[ELB v2] サービスに関する警告は表示されません。これは、ポリシー内のアクションの一部が [ELB v2] には存在するが、[ELB] には存在しないために起こります。[ELB] に関する警告は無視できます。

3. クラスター名、リージョンコード、およびアカウント ID の値を置き換えます。

```
eksctl create iamserviceaccount \
 --cluster=<cluster-name> \
 --namespace=kube-system \
 --name=aws-load-balancer-controller \
 --attach-policy-arn=arn:aws:iam::<AWS_ACCOUNT_ID>:policy/
AWSLoadBalancerControllerIAMPolicy \
 --override-existing-serviceaccounts \
 --region <aws-region-code> \
 --approve
```

## ステップ 2: AWS Load Balancer Controller をインストールする

1. eks-charts Helm チャートリポジトリを追加します。AWS は[このリポジトリ](#)を GitHub で管理しています。

```
helm repo add eks https://aws.github.io/eks-charts
```

2. ローカルリポジトリを更新して、最新のグラフがあることを確認します。

```
helm repo update eks
```

3. AWS Load Balancer コントローラをインストールします。

#### Amazon EC2 インスタンスメタデータサービス (IMDS) に対するアクセスが制限されている

Amazon EC2 ノードにコントローラをデプロイする場合、または Fargate もしくは Amazon EKS Hybrid Nodes にデプロイする場合には、次の helm コマンドに次のフラグを追加します。

- `--set region=region-code`
- `--set vpcId=vpc-xxxxxxxx`

**#####** の部分は自分のクラスター名に置き換えます。次のコマンドの中の `aws-load-balancer-controller` は、前のステップで作成した Kubernetes サービスアカウントです。

Helm チャートの設定の詳細については、GitHub の「[values.yaml](#)」を参照してください。

```
helm install aws-load-balancer-controller eks/aws-load-balancer-controller \
 -n kube-system \
 --set clusterName=my-cluster \
 --set serviceAccount.create=false \
 --set serviceAccount.name=aws-load-balancer-controller \
 --version 1.13.0
```

#### Important

デプロイされたグラフは、セキュリティに関する更新を自動的に受信しません。この更新が利用可能になったら、手動で新しいグラフにアップグレードする必要があります。アップグレードする場合は、前のコマンドで `install` を `upgrade` に変更します。

`helm install` コマンドは、コントローラのカスタムリソース定義 (CRD) を自動的にインストールします。一方、`helm upgrade` コマンドでは自動的にインストールされません。`helm upgrade` コマンドを使用する場合は、CRD を手動でインストールする必要があります。次のコマンドを実行して、CRD をインストールします。

```
wget https://raw.githubusercontent.com/aws/eks-charts/master/stable/aws-load-balancer-controller/crds/crds.yaml
kubectl apply -f crds.yaml
```

### ステップ 3: コントローラーがインストールされていることを確認する

#### 1. コントローラーがインストールされていることを確認します。

```
kubectl get deployment -n kube-system aws-load-balancer-controller
```

出力例は次のとおりです。

| NAME                         | READY | UP-TO-DATE | AVAILABLE | AGE |
|------------------------------|-------|------------|-----------|-----|
| aws-load-balancer-controller | 2/2   | 2          | 2         | 84s |

Helm を使用してデプロイした場合は、前の出力を受け取ります。Kubernetes マニフェストを使用してデプロイした場合、レプリカは 1 つしかありません。

#### 2. コントローラーを使用して AWS リソースをプロビジョニングする場合には、クラスターは特定の要件を満たしている必要があります。詳細については、「[the section called “アプリケーション負荷分散”](#)」および「[the section called “ネットワーク負荷分散”](#)」を参照してください。

### マニフェストによる AWS Load Balancer Controller のインストール

#### Tip

Amazon EKS Auto Mode では、ネットワーク形成のアドオンをインストールまたはアップグレードする必要はありません。自動モードにはポッドのネットワーク形成とロードバランシング機能が含まれています。

詳細については、「[EKS Auto Mode](#)」を参照してください。

このトピックでは、Kubernetes マニフェストをダウンロードし適用することでコントローラーをインストールする方法について説明します。コントローラーについての完全な [ドキュメント](#) は、GitHub でご覧になれます。

以下のステップでは、##### を独自の値に置き換えます。



## 前提条件

このチュートリアルを開始する前に、以下のステップを完了する必要があります。

- Amazon EKS クラスターを作成します。作成する場合は「[はじめに](#)」を参照してください。
- ローカルマシンに [Helm](#) をインストールします。
- Amazon VPC CNI plugin for Kubernetes、kube-proxy、および CoreDNS アドオンが、[サービス アカウントトークン](#)に記載されている最小のバージョンであることを確認してください。
- AWS Elastic Load Balancing の概念を確認します。詳細については、[Elastic Load Balancing ユーザーガイド](#)を参照してください。
- Kubernetes [サービス](#)と [Ingress](#) リソースについて確認します。

## 考慮事項

このページの設定手順に進む前に、以下の点について考慮してください。

- IAM ポリシーとロール (AmazonEKSLoadBalancerControllerRole) は、同じ AWS アカウントの複数の EKS クラスターで再利用できます。
- ロール (AmazonEKSLoadBalancerControllerRole) が最初に作成されたのと同じクラスターにコントローラーをインストールする場合は、ロールが存在することを確認した後、「[ステップ 2: cert-manager のインストール](#)」に進みます。
- サービスアカウントの IAM ロール (IRSA) を使用している場合、IRSA はクラスターごとに設定する必要があります。ロールの信頼ポリシー内の OpenID Connect (OIDC) プロバイダー ARN は各 EKS クラスターに固有です。さらに、既存の AmazonEKSLoadBalancerControllerRole を使用して新しいクラスターにコントローラーをインストールする場合は、そのロールの信頼ポリシーを更新して新しいクラスターの OIDC プロバイダーを追加し、適切なロール注釈を持つ新しいサービスアカウントを作成します。OIDC プロバイダーが既に存在しているかどうかを確認する、または OIDC プロバイダーを作成するには、「[the section called "IAM OIDC プロバイダー"](#)」を参照してください。

## ステップ 1: IAM を設定する

次の手順は、AWS Load Balancer Controller v2.11.0 リリースバージョンに関するものです。すべてのリリースの詳細については、GitHub で「[AWS Load Balancer Controller Release Page](#)」を参照してください。

1. ユーザーに代わって AWS API を呼び出すことを許可する、AWS Load Balancer Controller 用の IAM ポリシーをダウンロードします。

### Example

#### AWS

```
curl -O https://raw.githubusercontent.com/kubernetes-sigs/aws-load-balancer-controller/v2.11.0/docs/install/iam_policy.json
```

#### AWS GovCloud (US)

```
curl -O https://raw.githubusercontent.com/kubernetes-sigs/aws-load-balancer-controller/v2.11.0/docs/install/iam_policy_us-gov.json
```

```
mv iam_policy_us-gov.json iam_policy.json
```

2. 前のステップでダウンロードしたポリシー を使用して、IAM ポリシーを作成します。

```
aws iam create-policy \
 --policy-name AWSLoadBalancerControllerIAMPolicy \
 --policy-document file://iam_policy.json
```

#### Note

AWS Management Console でポリシーを確認すると、コンソールには [ELB] サービスに関する警告が表示されますが、[ELB v2] サービスに関する警告は表示されません。これは、ポリシー内のアクションの一部が [ELB v2] には存在するが、[ELB] には存在しないために起こります。[ELB] に関する警告は無視できます。

### Example

#### eksctl

- a. **my-cluster** はご自分のクラスター名に、**111122223333** はご自分のアカウント ID に置き換えた上で、コマンドを実行します。

```
eksctl create iamserviceaccount \
```

```
--cluster=my-cluster \
--namespace=kube-system \
--name=aws-load-balancer-controller \
--role-name AmazonEKSLoadBalancerControllerRole \
--attach-policy-arn=arn:aws:iam::111122223333:policy/
AWSLoadBalancerControllerIAMPolicy \
--approve
```

## AWS CLI and kubectl

- a. クラスターの OIDC プロバイダー ID を取得して、変数に格納します。

```
oidc_id=$(aws eks describe-cluster --name my-cluster --query
"cluster.identity.oidc.issuer" --output text | cut -d '/' -f 5)
```

- b. クラスターの ID を持つ IAM OIDC プロバイダーが既にアカウントにあるかどうかを確認します。クラスターと IAM の両方に OIDC を設定する必要があります。

```
aws iam list-open-id-connect-providers | grep $oidc_id | cut -d "/" -f4
```

出力が返された場合は、クラスター用の IAM OIDC プロバイダーが既に存在します。出力が返されない場合はクラスター用の IAM OIDC プロバイダーを作成する必要があります。詳細については、「[the section called “IAM OIDC プロバイダー”](#)」を参照してください。

- c. 次のコンテンツをデバイスにコピーします。**111122223333** は、ご自分のアカウント ID に置き換えます。**region-code** を、クラスターのある AWS リージョンに置き換えます。**EXAMPLED539D4633E53DE1B71EXAMPLE** を、前のステップで返された出力に置き換えます。

```
cat >load-balancer-role-trust-policy.json <<EOF
{
 "Version": "2012-10-17",
 "Statement": [
 {
 "Effect": "Allow",
 "Principal": {
 "Federated": "arn:aws:iam::111122223333:oidc-provider/
oidc.eks.region-code.amazonaws.com/id/EXAMPLED539D4633E53DE1B71EXAMPLE"
 },
 "Action": "sts:AssumeRoleWithWebIdentity",
 "Condition": {
```

```

 "StringEquals": {
 "oidc.eks.region-code.amazonaws.com/id/
EXAMPLED539D4633E53DE1B71EXAMPLE:aud": "sts.amazonaws.com",
 "oidc.eks.region-code.amazonaws.com/id/
EXAMPLED539D4633E53DE1B71EXAMPLE:sub": "system:serviceaccount:kube-system:aws-
load-balancer-controller"
 }
 }
}
]
}
EOF

```

- d. IAM ロールを作成します。

```

aws iam create-role \
 --role-name AmazonEKSLoadBalancerControllerRole \
 --assume-role-policy-document file://load-balancer-role-trust-policy.json

```

- e. IAM ロールに、必要な Amazon EKS 管理の IAM ポリシーをアタッチします。111122223333 は、ご自分のアカウント ID に置き換えます。

```

aws iam attach-role-policy \
 --policy-arn arn:aws:iam::111122223333:policy/AWSLoadBalancerControllerIAMPolicy \
 --role-name AmazonEKSLoadBalancerControllerRole

```

- f. 次のコンテンツをデバイスにコピーします。111122223333 は、ご自分のアカウント ID に置き換えます。テキストを置き換えたら、変更したコマンドを実行して aws-load-balancer-controller-service-account.yaml ファイルを作成します。

```

cat >aws-load-balancer-controller-service-account.yaml <<EOF
apiVersion: v1
kind: ServiceAccount
metadata:
 labels:
 app.kubernetes.io/component: controller
 app.kubernetes.io/name: aws-load-balancer-controller
 name: aws-load-balancer-controller
 namespace: kube-system
 annotations:
 eks.amazonaws.com/role-arn: arn:aws:iam::111122223333:role/
 AmazonEKSLoadBalancerControllerRole

```

EOF

- g. クラスター上で Kubernetes サービスアカウントを作成します。aws-load-balancer-controller という名前の Kubernetes サービスアカウントは、作成した IAM ロール (名前: *AmazonEKSLoadBalancerControllerRole*) でアノテーションされています。

```
kubectl apply -f aws-load-balancer-controller-service-account.yaml
```

## ステップ 2: **cert-manager** をインストールする

以下のいずれかの方法で cert-manager をインストールして、Webhook に証明書設定を導入します。詳細については、「cert-manager ドキュメント」の「[開始方法](#)」を参照してください。

cert-manager をインストールするには、quay.io コンテナレジストリを使用することをお勧めします。ノードが quay.io コンテナレジストリにアクセスできない場合は、Amazon ECR を使用して cert-manager をインストールします (以下を参照)。

### Example

#### Quay.io

- a. ノードが quay.io コンテナレジストリにアクセスできる場合は、cert-manager をインストールして、Webhook に証明書設定を挿入します。

```
kubectl apply \
 --validate=false \
 -f https://github.com/jetstack/cert-manager/releases/download/v1.13.5/cert-
manager.yaml
```

#### Amazon ECR

- a. 以下のいずれかの方法で cert-manager をインストールして、Webhook に証明書設定を導入します。詳細については、「cert-manager ドキュメント」の「[開始方法](#)」を参照してください。
- b. マニフェストをダウンロードします。

```
curl -Lo cert-manager.yaml https://github.com/jetstack/cert-manager/releases/
download/v1.13.5/cert-manager.yaml
```

- c. 次のイメージをプルして、ノードがアクセスできるリポジトリにプッシュします。イメージをプルし、タグ付けして独自のリポジトリにプッシュする方法の詳細については、[the section called “リポジトリにイメージをコピーする”](#) を参照してください。

```
quay.io/jetstack/cert-manager-cainjector:v1.13.5
quay.io/jetstack/cert-manager-controller:v1.13.5
quay.io/jetstack/cert-manager-webhook:v1.13.5
```

- d. 三つのイメージのマニフェストの `quay.io` を、独自のレジストリ名に置き換えます。次のコマンドは、プライベートリポジトリの名前がソースリポジトリと同じであることを前提としています。`111122223333.dkr.ecr.region-code.amazonaws.com` をプライベートレジストリに置き換えます。

```
sed -i.bak -e 's|quay.io|111122223333.dkr.ecr.region-code.amazonaws.com|' ./cert-manager.yaml
```

- e. マニフェストを適用します。

```
kubectl apply \
 --validate=false \
 -f ./cert-manager.yaml
```

### ステップ 3: AWS Load Balancer Controller をインストールする

1. Controller の詳細をダウンロードします。Controller の詳細については、GitHub の [ドキュメント](#) を参照してください。

```
curl -Lo v2_11_0_full.yaml https://github.com/kubernetes-sigs/aws-load-balancer-controller/releases/download/v2.11.0/v2_11_0_full.yaml
```

2. ファイルに以下の編集を行います。

- a. `v2_11_0_full.yaml` ファイルをダウンロードした場合は、次のコマンドを実行してマニフェストの `ServiceAccount` セクションを削除します。このセクションを削除しないと、前のステップでサービスアカウントに作成した必須の注釈が上書きされます。コントローラーを削除した場合、このセクションの削除により、前のステップで作成したサービスアカウントも保持されます。

```
sed -i.bak -e '690,698d' ./v2_11_0_full.yaml
```

別のバージョンのファイルをダウンロードした場合は、エディタでファイルを開き、次の行を削除します。

```
apiVersion: v1
kind: ServiceAccount
metadata:
 labels:
 app.kubernetes.io/component: controller
 app.kubernetes.io/name: aws-load-balancer-controller
 name: aws-load-balancer-controller
 namespace: kube-system

```

- b. **My-cluster** をユーザーのクラスター名に置き換えることにより、ファイルの Deployment spec セクションの `your-cluster-name` をクラスターの名前に置き換えます。

```
sed -i.bak -e 's|your-cluster-name|my-cluster|' ./v2_11_0_full.yaml
```

- c. ノードが Amazon EKS Amazon ECR イメージリポジトリにアクセスできない場合は、次のイメージをプルして、ノードがアクセスできるリポジトリにプッシュする必要があります。イメージをプルし、タグ付けして独自のリポジトリにプッシュする方法の詳細については、[the section called “リポジトリにイメージをコピーする”](#) を参照してください。

```
public.ecr.aws/eks/aws-load-balancer-controller:v2.11.0
```

マニフェストにレジストリの名前を追加します。次のコマンドは、プライベートリポジトリの名前がソースリポジトリと同じであると仮定し、プライベートレジストリの名前をファイルに追加します。**111122223333.dkr.ecr.region-code.amazonaws.com** をレジストリに置き換えます。この行は、プライベートリポジトリにソースリポジトリと同じ名前を付けたことを前提としています。そうでない場合は、プライベートレジストリ名の後の `eks/aws-load-balancer-controller` テキストを、リポジトリ名に変更します。

```
sed -i.bak -e 's|public.ecr.aws/eks/aws-load-balancer-controller|
111122223333.dkr.ecr.region-code.amazonaws.com/eks/aws-load-balancer-
controller|' ./v2_11_0_full.yaml
```

- d. (Fargate または制限付き IMDS の場合にのみ必須)

### Amazon EC2 インスタンスメタデータサービス (IMDS) に対するアクセスが制限されている

Amazon EC2 ノードにコントローラをデプロイする場合、または Fargate もしくは Amazon EKS Hybrid Nodes にデプロイする場合には、`- args:` の下に following parameters を追加します。

```
[...]
spec:
 containers:
 - args:
 - --cluster-name=your-cluster-name
 - --ingress-class=alb
 - --aws-vpc-id=vpc-xxxxxxx
 - --aws-region=region-code
[...]
```

#### 3. ファイルを適用します。

```
kubectl apply -f v2_11_0_full.yaml
```

#### 4. IngressClass および IngressClassParams マニフェストをクラスターにダウンロードします。

```
curl -Lo v2_11_0_ingclass.yaml https://github.com/kubernetes-sigs/aws-load-balancer-controller/releases/download/v2.11.0/v2_11_0_ingclass.yaml
```

#### 5. マニフェストをクラスターに適用します。

```
kubectl apply -f v2_11_0_ingclass.yaml
```

### ステップ 4: コントローラーがインストールされていることを確認する

#### 1. コントローラーがインストールされていることを確認します。

```
kubectl get deployment -n kube-system aws-load-balancer-controller
```

出力例は次のとおりです。



| NAME                         | READY | UP-TO-DATE | AVAILABLE | AGE |
|------------------------------|-------|------------|-----------|-----|
| aws-load-balancer-controller | 2/2   | 2          | 2         | 84s |

Helm を使用してデプロイした場合は、前の出力を受け取ります。Kubernetes マニフェストを使用してデプロイした場合、レプリカは 1 つしかありません。

2. コントローラーを使用して AWS リソースをプロビジョニングする場合には、クラスターは特定の要件を満たしている必要があります。詳細については、「[the section called “アプリケーション 負荷分散”](#)」および「[the section called “ネットワーク負荷分散”](#)」を参照してください。

## 非推奨になった ALB Ingress Controller からのアプリの移行

このトピックでは非推奨のコントローラーバージョンから移行する方法を説明します。具体的には、AWS Load Balancer Controller の非推奨バージョンを削除する方法について説明します。

- 非推奨バージョンはアップグレードできません。まずそれらを削除してから、最新バージョンをインストールする必要があります。
- 非推奨バージョンには以下が含まれます。
  - AWS Load Balancer Controller の前身である AWS ALB Ingress Controller for Kubernetes (「Ingress Controller」)。
  - AWS Load Balancer Controller の任意の 0.1.x バージョン

### 非推奨のコントローラーバージョンを削除する

#### Note

非推奨バージョンは、Helm を使用したか、Kubernetes マニフェストに従って手動でインストールされた可能性があります。この手順は元々インストールしてあるツールを使用して実行してください。

1. incubator/aws-alb-ingress-controller Helm チャートをインストールしてある場合はこれをアンインストールします。

```
helm delete aws-alb-ingress-controller -n kube-system
```

- eks-charts/aws-load-balancer-controller のバージョン 0.1.x をインストールしている場合はこれをアンインストールします。0.1.x からバージョン 1.0.0 へのアップグレードはWebhook API のバージョンとの互換性がないため動作しません。

```
helm delete aws-load-balancer-controller -n kube-system
```

- インGRESS・コントローラー がインストール済みであるかどうかを確認します。

```
kubectl get deployment -n kube-system alb-ingress-controller
```

これはコントローラーが取り付けられていない場合の出力です。

```
Error from server (NotFound): deployments.apps "alb-ingress-controller" not found
```

これは、コントローラーが取り付けられている場合の出力です。

| NAME                   | READY | UP-TO-DATE | AVAILABLE | AGE  |
|------------------------|-------|------------|-----------|------|
| alb-ingress-controller | 1/1   | 1          | 1         | 122d |

- 次のコマンドを入力してコントローラーを削除します。

```
kubectl delete -f https://raw.githubusercontent.com/kubernetes-sigs/aws-alb-ingress-controller/v1.1.8/docs/examples/alb-ingress-controller.yaml
kubectl delete -f https://raw.githubusercontent.com/kubernetes-sigs/aws-alb-ingress-controller/v1.1.8/docs/examples/rbac-role.yaml
```

## AWS Load Balancer Controller への移行

ALB Ingress Controller for Kubernetes から AWS Load Balancer Controller に移行するには、以下を実行する必要があります。

- ALB INGRESS・コントローラー を削除します (上記を参照)。
- [AWS ロードバランサー コントローラーをインストールします。](#)
- AWS Load Balancer Controller で使用される IAM ロールにポリシーを追加します。このポリシーにより、ALB Ingress Controller for Kubernetes によって作成されたリソースを LBC が管理できるようになります。

4. IAM ポリシーをダウンロードします。このポリシーにより、ALB Ingress Controller for Kubernetes によって作成されたリソースを AWS Load Balancer Controller が管理できるようになります。[ポリシーを表示](#)することもできます。

```
curl -O https://raw.githubusercontent.com/kubernetes-sigs/aws-load-balancer-controller/v2.11.0/docs/install/iam_policy_v1_to_v2_additional.json
```

5. クラスターが AWS GovCloud (米国東部) または AWS GovCloud (米国西部) の AWS リージョンにある場合は `arn:aws:` を `arn:aws-us-gov:` に置き換えます。

```
sed -i.bak -e 's|arn:aws:|arn:aws-us-gov:|' iam_policy_v1_to_v2_additional.json
```

6. IAM ポリシーを作成し、返された ARN を書き留めます。

```
aws iam create-policy \
 --policy-name AWSLoadBalancerControllerAdditionalIAMPolicy \
 --policy-document file://iam_policy_v1_to_v2_additional.json
```

7. AWS Load Balancer Controller によって使用される IAM ロールに IAM ポリシーをアタッチします。`####` を役割の名前 (例: AmazonEKSLoadBalancerControllerRole) に置き換えます。

`eksctl` を使用して役割を作成している場合、作成された役割名を見つけるには[AWS クラウド フォーメーション コンソール](#)を開き、`eksctl-#####-addon-iamserviceaccount-kube-system-aws-load-balancer-controller` スタックを選択してください。[Resources (リソース)] タブを選択してください。役割名は[Physical ID (物理 ID)] 列で見つかります。

```
aws iam attach-role-policy \
 --role-name your-role-name \
 --policy-arn arn:aws:iam::111122223333:policy/
AWSLoadBalancerControllerAdditionalIAMPolicy
```

## Amazon EKS クラスターで DNS の CoreDNS を管理する

### Tip

Amazon EKS 自動モード ではネットワーク形成のアドオンをインストールまたはアップグレードする必要はありません。自動モード にはポッドのネットワーク形成とロードバランシング機能が含まれています。

詳細については、「[EKS Auto Mode](#)」を参照してください。

CoreDNS は、Kubernetes クラスター DNS として機能する、フレキシブルで拡張可能な DNS サーバーです。少なくとも 1 つのノードで Amazon EKS クラスターを起動すると、クラスターにデプロイされたノード数に関係なく、デフォルトで CoreDNS イメージの 2 つのレプリカがデプロイされます。これらの CoreDNS ポッドは、クラスター内のすべてのポッドの名前解決を行います。クラスターに Fargate プロファイルが含まれ、名前空間が CoreDNS deployment の名前空間と一致する場合、CoreDNS Pod を Fargate ノードにデプロイできます。Fargate プロファイルの詳細については「[the section called “プロファイルを定義する”](#)」を参照してください。CoreDNS の詳細については、Kubernetes ドキュメントの「[Using CoreDNS for Service Discovery](#)」を参照してください。

## CoreDNS のバージョン

次の表は、Kubernetes バージョンごとに Amazon EKS アドオンタイプの最新バージョンを示しています。

| Kubernetes バージョン | CoreDNS のバージョン      |
|------------------|---------------------|
| 1.33             | v1.12.1-eksbuild.2  |
| 1.32             | v1.11.4-eksbuild.14 |
| 1.31             | v1.11.4-eksbuild.14 |
| 1.30             | v1.11.4-eksbuild.14 |
| 1.29             | v1.11.4-eksbuild.14 |
| 1.28             | v1.10.1-eksbuild.18 |
| 1.27             | v1.10.1-eksbuild.18 |
| 1.26             | v1.9.3-eksbuild.22  |

**⚠ Important**

このアドオンを自己管理している場合、表のバージョンは利用可能なセルフマネージドバージョンと同じではない可能性があります。このアドオンのセルフマネージドタイプの更新の詳細については「[the section called “更新 \(セルフマネージド\)”](#)」を参照してください。

## CoreDNS アップグレードに関する重要な考慮事項

- CoreDNS デプロイの安定性と可用性を高めるには、PodDisruptionBudget を使用してバージョン v1.9.3-eksbuild.6 以降および v1.10.1-eksbuild.3 をデプロイします。既存の PodDisruptionBudget をデプロイした場合、これらのバージョンへのアップグレードは失敗する可能性があります。アップグレードが失敗した場合は次のいずれかのタスクを実行することで問題が解決します。
- Amazon EKS アドオンをアップグレードする際は競合解決のオプションとして既存の設定を上書きすることを選択してください。デプロイに他のカスタム設定を行った場合は、アップグレード後に他のカスタム設定を再適用できるように、アップグレード前に必ず設定をバックアップしてください。
- 既存の PodDisruptionBudget を削除して、アップグレードを再試行してください。
- EKS アドオンバージョン v1.9.3-eksbuild.3 以降および v1.10.1-eksbuild.6 以降では、CoreDNS デプロイは /ready エンドポイントを使用するように readinessProbe を設定します。このエンドポイントは、CoreDNS の Corefile 設定ファイルで有効になっています。

カスタム Corefile を使用する場合は、ready プラグインを設定に追加して、プローブが使用できるように /ready エンドポイントを CoreDNS でアクティブにする必要があります。

- EKS アドオンバージョン v1.9.3-eksbuild.7 以降および v1.10.1-eksbuild.4 以降では PodDisruptionBudget を変更できます。次の例ではフィールドを使用して、アドオンを編集したり、[任意の設定項目]でこれらの設定を変更したりできます。この例はデフォルトの PodDisruptionBudget を示しています。

```
{
 "podDisruptionBudget": {
 "enabled": true,
 "maxUnavailable": 1
 }
}
```

maxUnavailable または minAvailable を設定できますが、両方を単一の PodDisruptionBudget で設定することはできません。PodDisruptionBudgets の詳細については Kubernetes ドキュメントの「[PodDisruptionBudgetの指定](#)」を参照してください。

enabled を false に設定しても、PodDisruptionBudget は削除されないことに注意してください。このフィールドを false に設定後、PodDisruptionBudget オブジェクトを削除する必要があります。同様に、PodDisruptionBudget のあるバージョンにアップグレードした後、古いバージョンのアドオンを使用するようにアドオンを編集した場合 (アドオンをダウングレード)、PodDisruptionBudget は削除されません。次のコマンドを実行して、PodDisruptionBudget を削除します。

```
kubectl delete poddisruptionbudget coredns -n kube-system
```

- EKS アドオンバージョン v1.10.1-eksbuild.5 以降ではデフォルトの許容値を KEP 2067 に準拠するように node-role.kubernetes.io/master:NoSchedule から node-role.kubernetes.io/control-plane:NoSchedule に変更してください。KEP 2067 の詳細については、GitHub で「Kubernetes Enhancement Proposals (KEPs)」の「[KEP-2067: Rename the kubeadm "master" label and taint](#)」を参照してください。

EKS アドオンバージョン v1.8.7-eksbuild.8 以降、および v1.9.3-eksbuild.9 以降では、どちらの許容範囲もすべての Kubernetes バージョンと互換性を維持できるように設定されています。

- EKS アドオンバージョン v1.9.3-eksbuild.11 および v1.10.1-eksbuild.7 以降では、CoreDNS デプロイは topologySpreadConstraints のデフォルト値を設定します。このデフォルト値により、複数のアベイラビリティーゾーンに使用可能なノードがある場合には、CoreDNS Pod がアベイラビリティーゾーン全体に分散します。デフォルト値の代わりに使用するカスタム値を設定できます。デフォルト値は次のとおりです。

```
topologySpreadConstraints:
 - maxSkew: 1
 topologyKey: topology.kubernetes.io/zone
 whenUnsatisfiable: ScheduleAnyway
 labelSelector:
 matchLabels:
 k8s-app: kube-dns
```

## CoreDNS v1.11 アップグレードに関する考慮事項

- EKS アドオン バージョン v1.11.1-eksbuild.4 以降ではコンテナイメージは Amazon EKS Distro によって維持される[最小ベースイメージ](#)に基づいています。これには最小限のパッケージが含まれ、シェルスはありません。詳細については「[Amazon EKS Distro](#)」を参照してください。CoreDNS イメージの使用方法和トラブルシューティングは変わりません。

## CoreDNS Amazon EKS アドオンを作成する

CoreDNS Amazon EKS アドオンを作成します。アドオンを作成する前に、クラスターを用意する必要があります。詳細については、「[the section called “クラスターを作成する”](#)」を参照してください。

1. クラスターにインストールされているアドオンのバージョンを確認します。

```
kubectl describe deployment coredns --namespace kube-system | grep coredns: | cut -d : -f 3
```

出力例は次のとおりです。

```
v1.10.1-eksbuild.13
```

2. クラスターにインストールされているアドオンのタイプを確認します。クラスターを作成するために使用したツールによっては、現在クラスターに アマゾン EKS アドオンタイプがインストールされていない場合があります。**#####** の部分は、自分のクラスター名に置き換えます。

```
aws eks describe-addon --cluster-name my-cluster --addon-name coredns --query
addon.addonVersion --output text
```

バージョン番号が返された場合、アマゾン EKS タイプのアドオンがクラスターにインストールされているため、このステップの残りのステップを完了する必要はありません。エラーが返された場合、クラスターに アマゾン EKS タイプのアドオンがインストールされていません。インストールするには、このステップの残りのステップを完了します。

3. 現在インストールされているアドオンの設定を保存します。

```
kubectl get deployment coredns -n kube-system -o yaml > aws-k8s-coredns-old.yaml
```

4. AWS CLI を使用してアドオンを作成します。AWS Management Console または `eksctl` を使用してアドオンを作成する場合は、「[the section called “アドオンを作成する”](#)」を参照して、アドオン名の `coredns` を指定します。デバイスに沿ったコマンドをコピーします。必要に応じてコマンドに次の変更を加え、変更したコマンドを実行してください。

- `#####` の部分は自分のクラスター名に置き換えます。
- `v1.11.3-eksbuild.1` を、使用しているクラスターバージョンに対して[最新バージョンの表](#)に記載されている最新バージョンに置き換えます。

```
aws eks create-addon --cluster-name my-cluster --addon-name coredns --addon-version v1.11.3-eksbuild.1
```

Amazon EKS アドオンのデフォルト設定と競合するカスタム設定を現在のアドオンに適用した場合、作成が失敗する可能性があります。作成に失敗した場合、問題解決に役立つエラーが表示されます。または、前のコマンドに `--resolve-conflicts OVERWRITE` を追加することもできます。これにより、アドオンは既存のカスタム設定を上書きできます。アドオンを作成したら、カスタム設定で更新できます。

5. クラスターの Kubernetes バージョン用のアドオンの最新バージョンがクラスターに追加されたことを確認します。`#####` の部分は、自分のクラスター名に置き換えます。

```
aws eks describe-addon --cluster-name my-cluster --addon-name coredns --query add-on.addonVersion --output text
```

アドオンの作成が完了するまでに数秒かかる場合があります。

出力例は次のとおりです。

```
v1.11.3-eksbuild.1
```

6. 元のアドオンに対してカスタム設定を行った場合は、Amazon EKS アドオンを作成する前に、前のステップで保存した設定を使用して、カスタム設定で Amazon EKS アドオンを更新します。アドオンの更新方法については「[the section called “更新 \(EKS アドオン\)”](#)」を参照してください。

## CoreDNS Amazon EKS アドオンを更新する

Amazon EKS タイプの アドオンを更新します。Amazon EKS アドオンをクラスターに追加していない場合は[アドオンを追加する](#)か、「[the section called “更新 \(セルフマネージド\)”](#)」を参照してください。



開始する前に、アップグレードに関する考慮事項を確認してください。詳細については「[the section called “CoreDNS アップグレードに関する重要な考慮事項”](#)」を参照してください。

1. クラスターにインストールされているアドオンのバージョンを確認します。##### の部分は自分のクラスター名に置き換えます。

```
aws eks describe-addon --cluster-name my-cluster --addon-name coredns --query
"addon.addonVersion" --output text
```

出力例は次のとおりです。

```
v1.10.1-eksbuild.13
```

返されたバージョンが、[最新バージョンの表](#)にあるクラスターの Kubernetes バージョンのバージョンと同じである場合は既に最新バージョンがクラスターにインストールされているため、この手順の残りを完了する必要はありません。出力にバージョン番号ではなくエラーが表示される場合はアマゾン EKS タイプのアドオンがクラスターにインストールされていません。この手順でアドオンを更新する前に、[アドオンを作成](#)する必要があります。

2. 現在インストールされているアドオンの設定を保存します。

```
kubectl get deployment coredns -n kube-system -o yaml > aws-k8s-coredns-old.yaml
```

3. AWS CLI を使用してアドオンを更新します。AWS Management Console または eksctl を使用してアドオンを更新する場合は「[the section called “アドオンを更新する”](#)」を参照してください。デバイスに沿ったコマンドをコピーします。必要に応じてコマンドに次の変更を加え、変更したコマンドを実行してください。

- ##### の部分は自分のクラスター名に置き換えます。
- **v1.11.3-eksbuild.1** を、使用しているクラスターバージョンに対して[最新バージョンの表](#)に記載されている最新バージョンに置き換えます。
- **--resolve-conflicts****PRESERVE** オプションはアドオンの既存の設定値を保存します。アドオン設定にカスタム値を設定していて、このオプションを使用しない場合、アマゾン EKS は値をデフォルト値で上書きします。このオプションを使用する場合、実稼働クラスターのアドオンを更新する前に、非稼働クラスターのフィールドおよび値変更をテストすることをお勧めします。この値を OVERWRITE に変更する場合、すべての設定がアマゾン EKS のデフォルト値に変更されます。いずれかの設定にカスタム値を設定した場合、アマゾン EKS のデフォルト値で上書きされる可能性があります。この値を none に変更した場合、アマゾン EKS は設定の値

を一切変更しませんが、更新が失敗する可能性があります。更新に失敗した場合、競合の解決に役立つエラーメッセージが返されます。

- 構成設定を更新しない場合はコマンドから `--configuration-values` `'{"replicaCount":3}'` を削除します。構成設定を更新する場合は `"replicaCount":3` を設定したい設定に置き換えてください。この例では、CoreDNS のレプリカ数は 3 に設定されています。指定する値は設定スキーマに対して有効である必要があります。設定スキーマが不明である場合は `aws eks describe-addon-configuration --addon-name coredns --addon-version v1.11.3-eksbuild.1` を実行して、`v1.11.3-eksbuild.1` を、設定を表示するアドオンのバージョン番号に置き換えます。出力でスキーマが返されます。既存のカスタム設定があり、それをすべて削除してすべての設定の値を Amazon EKS のデフォルトに戻したい場合はコマンドから `"replicaCount":3` を削除して、`{}` が空になるようにします。設定の詳細については、Kubernetes ドキュメントの「[Customizing DNS Service](#)」を参照してください。

```
aws eks update-addon --cluster-name my-cluster --addon-name coredns --addon-version v1.11.3-eksbuild.1 \
 --resolve-conflicts PRESERVE --configuration-values '{"replicaCount":3}'
```

更新が完了するまでに数秒かかる場合があります。

- アドオンのバージョンが更新されたことを確認します。##### の部分は自分のクラスター名に置き換えます。

```
aws eks describe-addon --cluster-name my-cluster --addon-name coredns
```

更新が完了するまでに数秒かかる場合があります。

出力例は次のとおりです。

```
{
 "addon": {
 "addonName": "coredns",
 "clusterName": "my-cluster",
 "status": "ACTIVE",
 "addonVersion": "v1.11.3-eksbuild.1",
 "health": {
 "issues": []
 },
 "addonArn": "arn:aws:eks:region:111122223333:addon/my-cluster/coredns/d2c34f06-1111-2222-1eb0-24f64ce37fa4",
 }
}
```

```
"createdAt": "2023-03-01T16:41:32.442000+00:00",
"modifiedAt": "2023-03-01T18:16:54.332000+00:00",
"tags": {},
"configurationValues": "{\"replicaCount\":3}"
}
}
```

## CoreDNS Amazon EKS セルフマネージドアドオンを更新する

### Important

セルフマネージド型のアドオンを使用する代わりに、アマゾン EKS タイプのアドオンをクラスターに追加することをお勧めします。タイプの違いがよくわからない場合は「[the section called “Amazon EKS アドオン”](#)」を参照してください。アマゾン EKS アドオンをクラスターに追加する方法については「[the section called “アドオンを作成する”](#)」を参照してください。アマゾン EKS アドオンを使用できない場合はその理由に関する問題を[コンテナロードマップ GitHub リポジトリ](#)に送信することをお勧めします。

開始する前に、アップグレードに関する考慮事項を確認してください。詳細については「[the section called “CoreDNS アップグレードに関する重要な考慮事項”](#)」を参照してください。

1. クラスターにインストールされているアドオンがセルフマネージド型であることを確認します。 **#####** の部分は自分のクラスター名に置き換えます。

```
aws eks describe-addon --cluster-name my-cluster --addon-name coredns --query
addon.addonVersion --output text
```

エラーメッセージが返された場合、クラスターにセルフマネージド型のアドオンがインストールされています。インストールするにはこのステップの残りのステップを完了します。バージョン番号が返された場合、クラスターにアマゾン EKS タイプのアドオンがインストールされています。Amazon EKS タイプのアドオンを更新するにはこの手順を使用するのではなく、「[CoreDNS Amazon EKS アドオンを更新する](#)」の手順を使用してください。アドオンタイプの違いがよくわからない場合は「[the section called “Amazon EKS アドオン”](#)」を参照してください。

2. クラスターに現在インストールされているコンテナイメージのバージョンを確認します。

```
kubectl describe deployment coredns -n kube-system | grep Image | cut -d ":" -f 3
```

出力例は次のとおりです。

```
v1.8.7-eksbuild.2
```

3. 現在の CoreDNS のバージョンが v1.5.0 以降で、かつ [CoreDNS バージョン](#) 表に記載されるバージョンよりも前の場合は、このステップをスキップしてください。現在のバージョンが 1.5.0 より前の場合、プロキシアドオンではなく進んだアドオンを使用するためには、CoreDNS の ConfigMap を修正する必要があります。
  - a. 次のコマンドを使用して ConfigMap を開きます。

```
kubectl edit configmap coredns -n kube-system
```

- b. 次の行の proxy を forward に置き換えます。ファイルを保存し、エディタを終了します。

```
proxy . /etc/resolv.conf
```

4. Kubernetes 1.17 以前にクラスターを最初にデプロイした場合は、CoreDNS マニフェストから廃止された行を削除する必要があります。

#### Important

CoreDNS バージョン 1.7.0 に更新する前に、この手順を完了する必要があります。以前のバージョンに更新する場合でも、この手順を完了することをお勧めします。

- a. CoreDNS マニフェストにその行があるかどうか確認します。

```
kubectl get configmap coredns -n kube-system -o jsonpath='{$.data.Corefile}' |
grep upstream
```

出力が返されない場合、マニフェストにその行がないため、次のステップに進み、CoreDNS を更新することができます。出力が返された場合はその行を削除します。

- b. 以下のコマンドを使用して ConfigMap を編集し、ファイル内の upstream という単語がある行を削除します。このファイル内の他の部分は変更しないでください。行を削除したら、変更を保存します。

```
kubectl edit configmap coredns -n kube-system -o yaml
```

## 5. 現在の CoreDNS イメージバージョンを取得します。

```
kubectl describe deployment coredns -n kube-system | grep Image
```

出力例は次のとおりです。

```
602401143452.dkr.ecr.region-code.amazonaws.com/eks/coredns:v1.8.7-eksbuild.2
```

## 6. CoreDNS 1.8.3 以降に更新する場合は、endpointslices のアクセス許可を system:coredns Kubernetes clusterrole に追加する必要があります。

```
kubectl edit clusterrole system:coredns -n kube-system
```

ファイルの rules セクション内の既存の権限行の下に次の行を追加します。

```
[...]
- apiGroups:
 - discovery.k8s.io
 resources:
 - endpointslices
 verbs:
 - list
 - watch
[...]
```

## 7. **602401143452** および **region-code** を、前のステップで返された出力の値に置き換えて、CoreDNS アドオンを更新します。**v1.11.3-eksbuild.1** を、使用している Kubernetes バージョンについて[最新バージョンの表](#)に記載されている CoreDNS バージョンに置き換えます。

```
kubectl set image deployment.apps/coredns -n kube-system
coredns=602401143452.dkr.ecr.region-code.amazonaws.com/eks/coredns:v1.11.3-eksbuild.1
```

出力例は次のとおりです。

```
deployment.apps/coredns image updated
```

## 8. コンテナイメージのバージョンをもう一度チェックして、前のステップで指定したバージョンに更新されたことを確認します。

```
kubectl describe deployment coredns -n kube-system | grep Image | cut -d ":" -f 3
```

出力例は次のとおりです。

```
v1.11.3-eksbuild.1
```

## CoreDNS ポッドを高い DNS トラフィックにスケールする

1 つ以上のノードで Amazon EKS クラスターを起動すると、クラスターにデプロイされたノード数に関係なく、デフォルトで CoreDNS イメージの 2 つのレプリカのデプロイがデプロイされます。これらの CoreDNS ポッドは、クラスター内のすべてのポッドの名前解決を行います。アプリケーションは名前解決を使用して、クラスター内のポッドとサービスに接続し、同様にクラスター外のサービスに接続します。ポッドからの名前解決 (クエリ) のリクエスト数が増えると、CoreDNS ポッドが処理しきれなくなって速度が低下し、ポッドが処理できないリクエストが拒否される可能性があります。

CoreDNS ポッドの負荷の増加を処理するには、CoreDNS の自動スケーリングシステムを検討してください。Amazon EKS は、CoreDNS の EKS アドオンバージョンで CoreDNS デプロイの自動スケーリングを管理できます。この CoreDNS オートスケーラーは、ノード数や CPU コア数など、クラスターの状態を継続的にモニタリングします。この情報に基づいて、コントローラーは EKS クラスター内の CoreDNS デプロイのレプリカ数を動的に調整します。この機能は CoreDNS v1.9 以降で動作します。CoreDNS 自動スケーリングと互換性のあるバージョンの詳細については、次のセクションを参照してください。

この機能はアプリケーションの全体的な可用性とクラスターのスケーラビリティを向上させるために、他の [EKS クラスターの自動スケーリングのベストプラクティス](#) と組み合わせて使用することをお勧めします。

### 前提条件

Amazon EKS が CoreDNS デプロイをスケーリングするには、次の 3 つの前提条件があります。

- CoreDNS の EKS アドオンバージョンを使用する必要があります。
- クラスターは少なくとも最小のクラスターバージョンとプラットフォームバージョンを実行している必要があります。
- クラスターは少なくとも CoreDNS の EKS アドオンの最小バージョンを実行している必要があります。

## 最小クラスターバージョン

CoreDNS の自動スケーリングは Amazon EKS によって管理されるクラスターコントロールプレーンの新しいコンポーネントによって行われます。このため、新しいコンポーネントを持つ最小プラットフォームバージョンをサポートする EKS リリースにクラスターをアップグレードする必要があります。

新しい Amazon EKS クラスター。デプロイするには「[はじめに](#)」を参照してください。クラスターは次の表に示す Kubernetes バージョンとプラットフォームバージョン、またはそれ以降のいずれかを実行している必要があります。一覧にあるバージョンより後の Kubernetes とプラットフォームのバージョンもサポートされることにご注意ください。現在の Kubernetes バージョンを確認するには、次のコマンドの *my-cluster* をクラスターの名前に置き換えて、変更したコマンドを実行します。

```
aws eks describe-cluster --name my-cluster --query cluster.version --output text
```

| Kubernetes バージョン | プラットフォームバージョン |
|------------------|---------------|
| 1.29.3           | eks.7         |
| 1.28.8           | eks.13        |
| 1.27.12          | eks.17        |
| 1.26.15          | eks.18        |

### Note

以降の Kubernetes バージョンのすべてのプラットフォームバージョンもサポートされています。例えば、Kubernetes バージョン 1.30 の eks.1 以降のバージョンなどです。

## EKS アドオンの最小バージョン

| Kubernetes バージョン | 1.29               | 1.28                | 1.27                | 1.26               |
|------------------|--------------------|---------------------|---------------------|--------------------|
|                  | v1.11.1-eksbuild.9 | v1.10.1-eksbuild.11 | v1.10.1-eksbuild.11 | v1.9.3-eksbuild.15 |

### AWS Management Consoleで CoreDNS 自動スケーリングを設定する

#### 1. クラスターが最小クラスターバージョン以上であることを確認します。

Amazon EKS は同じ Kubernetes バージョンのプラットフォームバージョン間でクラスターを自動的にアップグレードするため、このプロセスを自分で開始することはできません。代わりに、クラスターを次の Kubernetes バージョンにアップグレードすると、クラスターはその K8s バージョンと最新のプラットフォームバージョンにアップグレードされます。

新しい Kubernetes バージョンでは、大幅な変更が加えられている場合があります。このため、本稼働用クラスターで更新を行う前に、新しい Kubernetes バージョンのクラスターを別に用意し、アプリケーションの動作をテストしておくことをお勧めします。

クラスターを新しい Kubernetes バージョンにアップグレードするには、「[Update existing cluster to new Kubernetes version](#)」の手順に従います。

#### 2. セルフマネージド型の CoreDNS デプロイではなく、CoreDNS 用の EKS アドオンがあることを確認します。

クラスターを作成するために使用したツールによっては現在クラスターに Amazon EKS アドオンタイプがインストールされていない場合があります。クラスターにインストールされているアドオンのタイプを確認するには次のコマンドを実行してください。my-cluster を自分のクラスター名に置き換えます。

```
aws eks describe-addon --cluster-name my-cluster --addon-name coredns --query
addon.addonVersion --output text
```

バージョン番号が返された場合、クラスターに Amazon EKS タイプのアドオンがインストールされ、次のステップに進むことができます。エラーが返された場合、クラスターに Amazon EKS タイプのアドオンがインストールされていません。ステップ「[CoreDNS Amazon EKS を作成する](#)」



の残りのステップを完了して、セルフマネージドバージョンを Amazon EKS アドオンに置き換えます。

3. CoreDNS の EKS アドオンが、最小 EKS アドオンバージョンと同じかそれ以上のバージョンであることを確認します。

クラスターにインストールされているアドオンのバージョンを確認します。AWS Management Consoleで確認するか、以下のコマンドを実行できます：

```
kubectl describe deployment coredns --namespace kube-system | grep coredns: | cut -d : -f 3
```

出力例は次のとおりです。

```
v1.10.1-eksbuild.13
```

このバージョンを前のセクションの最小 EKS アドオンバージョンと比較します。必要に応じて、「[CoreDNS Amazon EKS アドオンを更新する](#)」の手順に従って EKS アドオンを上位のバージョンにアップグレードします。

4. 自動スケーリング設定を EKS アドオンの [オプションの設定] に追加します。
  - a. [Amazon EKS コンソール](#)を開きます。
  - b. 左のナビゲーションペインで、[クラスター] を選択し、次にアドオンを設定するクラスター名を選択してください。
  - c. [アドオン] タブを選択してください。
  - d. CoreDNS アドオンボックスの右上にあるボックスを選択し、次に [編集] を選択してください。
  - e. [CoreDNS の設定] ページで、次の操作を行います。
    - i. 使用する [バージョン] を選択してください。前のステップと同じバージョンを保持し、別のアクションでバージョンと設定を更新することをお勧めします。
    - ii. [オプションの構成設定] を展開します。
    - iii. [設定値] で JSON キー "autoscaling": とネストした JSON オブジェクトの値に、キー "enabled": と値 true を指定します。結果のテキストは有効な JSON オブジェクトでなければなりません。このキーと値だけがテキストボックス内のデータである場合はキーと値を中括弧 { } で囲みます。次の例は自動スケーリングが有効になっていることを示しています：

```
{
 "autoScaling": {
 "enabled": true
 }
}
```

- iv. (オプション) 自動スケーリングが CoreDNS ポッド数をスケールできる最小値と最大値を指定できます。

次の例は自動スケーリングが有効で、すべてのオプションキーに値があることを示しています。クラスター内の DNS サービスに回復力を提供できるように、CoreDNS ポッドの最小数は常に 2 より大きくすることをお勧めします。

```
{
 "autoScaling": {
 "enabled": true,
 "minReplicas": 2,
 "maxReplicas": 10
 }
}
```

- f. CoreDNS ポッドを置き換えて新しい設定を適用するには、[変更を保存] を選択します。

Amazon EKS は CoreDNS の Kubernetes デプロイのロールアウトを使用して EKS アドオンに変更を適用します。ロールアウトのステータスは AWS Management Console のアドオン [更新履歴] と `kubectl rollout status deployment/coredns --namespace kube-system` で追跡できます。

`kubectl rollout` は以下のコマンドを実行してください：

```
kubectl rollout

history -- View rollout history
pause -- Mark the provided resource as paused
restart -- Restart a resource
resume -- Resume a paused resource
status -- Show the status of the rollout
undo -- Undo a previous rollout
```

ロールアウトに時間がかかりすぎる場合、Amazon EKS はロールアウトを取り消し、[アドオン更新] のタイプと [失敗] のステータスのメッセージがアドオンの [更新履歴] に追加されます。問題を調査するにはロールアウトの履歴から開始し、CoreDNS のポッドで `kubectl logs` を実行して CoreDNS のログを確認します。

5. [更新履歴] の新しいエントリのステータスが [成功] の場合、ロールアウトが完了し、アドオンはすべての CoreDNS のポッドで新しい設定を使用していることを意味します。クラスター内のノード数とノードの CPU コア数を変更すると、Amazon EKS は CoreDNS デプロイのレプリカ数をスケールします。

## AWS コマンドラインインターフェイスで CoreDNS 自動スケーリングを設定する

1. クラスターが最小クラスターバージョン以上であることを確認します。

Amazon EKS は同じ Kubernetes バージョンのプラットフォームバージョン間でクラスターを自動的にアップグレードするため、このプロセスを自分で開始することはできません。代わりに、クラスターを次の Kubernetes バージョンにアップグレードすると、クラスターはその K8s バージョンと最新のプラットフォームバージョンにアップグレードされます。

新しい Kubernetes バージョンでは、大幅な変更が加えられている場合があります。このため、本稼働用クラスターで更新を行う前に、新しい Kubernetes バージョンのクラスターを別に用意し、アプリケーションの動作をテストしておくことをお勧めします。

クラスターを新しい Kubernetes バージョンにアップグレードするには、「[Update existing cluster to new Kubernetes version](#)」の手順に従います。

2. セルフマネージド型の CoreDNS デプロイではなく、CoreDNS 用の EKS アドオンがあることを確認します。

クラスターを作成するために使用したツールによっては現在クラスターに Amazon EKS アドオンタイプがインストールされていない場合があります。クラスターにインストールされているアドオンのタイプを確認するには次のコマンドを実行してください。my-cluster を自分のクラスター名に置き換えます。

```
aws eks describe-addon --cluster-name my-cluster --addon-name coredns --query
addon.addonVersion --output text
```

バージョン番号が返された場合、クラスターに Amazon EKS タイプのアドオンがインストールされています。エラーが返された場合、クラスターに Amazon EKS タイプのアドオンがインストー

ルされていません。ステップ「[CoreDNS Amazon EKS を作成する](#)」の残りのステップを完了して、セルフマネージドバージョンを Amazon EKS アドオンに置き換えます。

3. CoreDNS の EKS アドオンが、最小 EKS アドオンバージョンと同じかそれ以上のバージョンであることを確認します。

クラスターにインストールされているアドオンのバージョンを確認します。AWS Management Consoleで確認するか、以下のコマンドを実行できます：

```
kubectl describe deployment coredns --namespace kube-system | grep coredns: | cut -d : -f 3
```

出力例は次のとおりです。

```
v1.10.1-eksbuild.13
```

このバージョンを前のセクションの最小 EKS アドオンバージョンと比較します。必要に応じて、「[CoreDNS Amazon EKS アドオンを更新する](#)」の手順に従って EKS アドオンを上位のバージョンにアップグレードします。

4. 自動スケーリング設定を EKS アドオンの [オプションの設定] に追加します。

次の AWS CLI コマンドを実行してください。my-cluster をクラスターの名前に置き換え、IAM ロール ARN を使用するロールに置き換えます。

```
aws eks update-addon --cluster-name my-cluster --addon-name coredns \
 --resolve-conflicts PRESERVE --configuration-values '{"autoScaling":\
{"enabled":true}}'
```

Amazon EKS は CoreDNS の Kubernetes デプロイのロールアウトを使用して EKS アドオンに変更を適用します。ロールアウトのステータスは AWS Management Console のアドオン [更新履歴] と `kubectl rollout status deployment/coredns --namespace kube-system` で追跡できます。

`kubectl rollout` は以下のコマンドを実行してください：

```
kubectl rollout

history -- View rollout history
pause -- Mark the provided resource as paused
```

```
restart -- Restart a resource
resume -- Resume a paused resource
status -- Show the status of the rollout
undo -- Undo a previous rollout
```

ロールアウトに時間がかかりすぎる場合、Amazon EKS はロールアウトを取り消し、[アドオン更新] のタイプと [失敗] のステータスのメッセージがアドオンの [更新履歴] に追加されます。問題を調査するにはロールアウトの履歴から開始し、CoreDNS のポッドで `kubectl logs` を実行して CoreDNS のログを確認します。

5. (オプション) 自動スケーリングが CoreDNS ポッド数をスケールできる最小値と最大値を指定できます。

次の例は自動スケーリングが有効で、すべてのオプションキーに値があることを示しています。クラスター内の DNS サービスに回復力を提供できるように、CoreDNS ポッドの最小数は常に 2 より大きくすることをお勧めします。

```
aws eks update-addon --cluster-name my-cluster --addon-name coredns \
 --resolve-conflicts PRESERVE --configuration-values '{"autoScaling":
{"enabled":true,"minReplicas":2,"maxReplicas":10}}'
```

6. 次のコマンドを実行して、アドオンの更新のステータスを確認します：

```
aws eks describe-addon --cluster-name my-cluster --addon-name coredns
```

"status": "ACTIVE" という行が表示された場合、ロールアウトが完了し、アドオンはすべての CoreDNS ポッドで新しい設定を使用していることを意味します。クラスター内のノード数とノードの CPU コア数を変更すると、Amazon EKS は CoreDNS デプロイのレプリカ数をスケールします。

## CoreDNS メトリクスを使用して Kubernetes DNS 解決をモニタリングする

CoreDNS は、EKS アドオンとして、9153 ポートの CoreDNS からのメトリクスを kube-dns サービス内の Prometheus 形式で公開します。Prometheus、Amazon CloudWatch エージェント、またはその他の互換性のあるシステムを使用して、これらのメトリクスをスクレイピング (収集) できます。

Prometheus と CloudWatch エージェントの両方と互換性のあるスクレイプ設定の例については Amazon CloudWatch ユーザーガイドの「[Prometheus 用の CloudWatch エージェント設定](#)」を参照してください。

## Amazon EKS クラスターで **kube-proxy** を管理する

### Tip

Amazon EKS 自動モード ではネットワーク形成のアドオンをインストールまたはアップグレードする必要はありません。自動モード にはポッドのネットワーク形成とロードバランシング機能が含まれています。

詳細については「[EKS Auto Mode](#)」を参照してください。

セルフマネージド型のアドオンを使用する代わりに、Amazon EKS タイプのアドオンをクラスターに追加することをお勧めします。タイプの違いがよくわからない場合は「[the section called “Amazon EKS アドオン”](#)」を参照してください。Amazon EKS アドオンをクラスターに追加する方法については「[the section called “アドオンを作成する”](#)」を参照してください。Amazon EKS アドオンを使用できない場合はその理由に関する問題を[コンテナロードマップ GitHub リポジトリ](#)に送信することをお勧めします。

kube-proxy アドオンは Amazon EKS クラスター内の各 Amazon EC2 ノードにデプロイされます。これは、ノード上のネットワークルールを維持し、Pod へのネットワーク通信を有効にします。アドオンはクラスター内の Fargate ノードにはデプロイされません。詳細については、Kubernetes ドキュメントの「[kube-proxy](#)」を参照してください。

## Amazon EKS アドオンとしてインストールする

### **kube-proxy** バージョン

次の表は、Kubernetes バージョンごとに Amazon EKS アドオンタイプの最新バージョンを示しています。

| Kubernetes バージョン | <b>kube-proxy</b> バージョン |
|------------------|-------------------------|
| 1.33             | v1.33.0-eksbuild.2      |
| 1.32             | v1.32.3-eksbuild.7      |

| Kubernetes バージョン | kube-proxy バージョン     |
|------------------|----------------------|
| 1.31             | v1.31.7-eksbuild.7   |
| 1.30             | v1.30.11-eksbuild.5  |
| 1.29             | v1.29.15-eksbuild.5  |
| 1.28             | v1.28.15-eksbuild.20 |
| 1.27             | v1.27.16-eksbuild.30 |
| 1.26             | v1.26.15-eksbuild.35 |

**Note**

以前のバージョンのドキュメントには誤りが含まれていました。kube-proxy バージョン v1.28.5、v1.27.9、および v1.26.12 は利用できません。  
このアドオンを自己管理している場合、表のバージョンは利用可能なセルフマネージドバージョンと同じではない可能性があります。

## kube-proxy コンテナイメージ

kube-proxy コンテナイメージは Amazon EKS Distro によって維持される [最小ベースイメージ](#) に基づいています。これには最小限のパッケージが含まれ、シェルはありません。詳細については「[Amazon EKS Distro](#)」を参照してください。

次の表に、各 Amazon EKS クラスターバージョンで利用可能な最新のセルフマネージド kube-proxy コンテナイメージのバージョンを示します。

| バージョン | kube-proxy                 |
|-------|----------------------------|
| 1.33  | v1.33.0-minimal-eksbuild.2 |
| 1.32  | v1.32.3-minimal-eksbuild.7 |
| 1.31  | v1.31.7-minimal-eksbuild.7 |

| バージョン | kube-proxy                   |
|-------|------------------------------|
| 1.30  | v1.30.11-minimal-eksbuild.5  |
| 1.29  | v1.29.15-minimal-eksbuild.5  |
| 1.28  | v1.28.15-minimal-eksbuild.20 |
| 1.27  | v1.27.16-minimal-eksbuild.30 |
| 1.26  | v1.26.15-minimal-eksbuild.35 |

[Amazon EKS アドオンタイプ](#)を更新するときは有効な Amazon EKS アドオンバージョンを指定しますが、この表に記載されているバージョンではない可能性があります。これは[Amazon EKS アドオン](#)のバージョンが、このアドオンのセルフマネージドタイプを更新するときに指定されるコンテナイメージのバージョンと常に一致するとは限らないためです。このアドオンのセルフマネージドタイプを更新するときはこの表に記載されている有効なコンテナイメージのバージョンを指定します。

## Kubernetes **kube-proxy** セルフマネージドアドオンの更新

### Important

セルフマネージド型のアドオンを使用する代わりに、アマゾン EKS タイプのアドオンをクラスターに追加することをお勧めします。タイプの違いがよくわからない場合は「[the section called “Amazon EKS アドオン”](#)」を参照してください。アマゾン EKS アドオンをクラスターに追加する方法については「[the section called “アドオンを作成する”](#)」を参照してください。Amazon EKS アドオンを使用できない場合は、その理由に関する問題を[コンテナロードマップ GitHub リポジトリ](#)に送信することをお勧めします。

### 前提条件

- 既存の Amazon EKS クラスター。デプロイするには「[はじめに](#)」を参照してください。



## 考慮事項

- Amazon EKS クラスターの Kube-proxy は、[Kubernetes と同じ互換性およびスケーラビリティ](#)が適用されます。[Amazon EKS アドオンバージョンのクラスターとの互換性の検証](#)について説明します。

1. クラスターにインストールされているアドオンがセルフマネージド型であることを確認します。##### の部分は自分のクラスター名に置き換えます。

```
aws eks describe-addon --cluster-name my-cluster --addon-name kube-proxy --query
addon.addonVersion --output text
```

エラーメッセージが返された場合、クラスターにセルフマネージド型のアドオンがインストールされています。このトピックの残りの手順は、セルフマネージド型のアドオンを更新することです。バージョン番号が返された場合、クラスターに Amazon EKS タイプのアドオンがインストールされています。Amazon EKS タイプのアドオンを更新するには、このトピックの手順を使用するのではなく、「[Amazon EKS アドオンの更新](#)」の手順を使用してください。アドオンタイプの違いがよくわからない場合は、「[the section called “Amazon EKS アドオン”](#)」を参照してください。

2. クラスターに現在インストールされているコンテナイメージのバージョンを確認します。

```
kubectl describe daemonset kube-proxy -n kube-system | grep Image
```

出力例は次のとおりです。

```
Image: 602401143452.dkr.ecr.region-code.amazonaws.com/eks/kube-proxy:v1.29.1-
eksbuild.2
```

出力例では、**v1.29.1-eksbuild.2** がクラスターにインストールされているバージョンです。

3. **602401143452** および **region-code** を、前のステップの出力の値に置き換えて、kube-proxy アドオンを更新します。**v1.30.6-eksbuild.3** を、「[Amazon EKS クラスターの各バージョンで使用可能な最新のセルフマネージド kube-proxy コンテナイメージバージョン](#)」の表に記載されている kube-proxy バージョンに置き換えます。

**⚠ Important**

各イメージタイプのマニフェストは異なり、デフォルトまたは最小のイメージタイプ間で互換性がありません。エントリポイントと引数が一致するように、前のイメージと同じイメージタイプを使用する必要があります。

```
kubectl set image daemonset.apps/kube-proxy -n kube-system kube-
proxy=602401143452.dkr.ecr.region-code.amazonaws.com/eks/kube-proxy:v1.30.6-
eksbuild.3
```

出力例は次のとおりです。

```
daemonset.apps/kube-proxy image updated
```

- 新しいバージョンがクラスターにインストールされたことを確認します。

```
kubectl describe daemonset kube-proxy -n kube-system | grep Image | cut -d ":" -f 3
```

出力例は次のとおりです。

```
v1.30.0-eksbuild.3
```

- 同じクラスターで x86 ノードと Arm ノードを使用しており、クラスターが 2020 年 8 月 17 日より前にデプロイされている場合。以下のコマンドにより kube-proxy マニフェストを編集して、複数のハードウェアアーキテクチャにノードセクターを含めます。このオペレーションを行うのは 1 回限りです。マニフェストにセクタを追加した後、アドオンを更新するたびに追加する必要はありません。クラスターが 2020 年 8 月 17 日以降にデプロイされている場合、kube-proxy は既にマルチアーキテクチャに対応しています。

```
kubectl edit -n kube-system daemonset/kube-proxy
```

エディタを使用して、次のノードセクターをファイルに追加し、その内容を保存します。エディタ上で、このテキストを挿入する場所の例については、GitHub の [CNI マニフェスト](#) ファイルをご覧ください。これにより、Kubernetes はノードのハードウェアアーキテクチャに基づいて、正しいハードウェアイメージを取得できるようになります。

```
- key: "kubernetes.io/arch"
 operator: In
 values:
 - amd64
 - arm64
```

6. クラスターを最初に作成したのが Kubernetes バージョン 1.14 以降である場合は、既にこの Affinity Rule が kube-proxy に含まれているため、この手順をスキップできます。Amazon EKS クラスターを最初に作成したのが Kubernetes 1.13 以前のバージョンであり、今後そのクラスターで Fargate ノードを使用する場合は、kube-proxy マニフェストを編集して NodeAffinity ルールを含め、kube-proxy Pod が Fargate ノードでスケジュールしないようにしてください。この編集を行うのは 1 回限りです。Affinity Rule をマニフェストに一度追加したら、アドオンを更新するたびに追加する必要はありません。kube-proxy DaemonSet を編集します。

```
kubectl edit -n kube-system daemonset/kube-proxy
```

エディタで、次の Affinity Rule をファイルの DaemonSet spec セクションに追加して、ファイルを保存します。エディタ上で、このテキストを挿入する場所の例については、GitHub の [CNI マニフェスト](#) ファイルをご覧ください。

```
- key: eks.amazonaws.com/compute-type
 operator: NotIn
 values:
 - fargate
```

# Amazon EKS にワークロードとアドオンをデプロイする方法について説明します。

ワークロードはコンテナにデプロイされ、コンテナは Kubernetes の Pod にデプロイされます。Pod には、1 つまたは複数のコンテナが含まれます。通常、Kubernetes サービスには、同じサービスを提供する 1 つまたは複数の Pod がデプロイされます。同じサービスを提供する複数の Pod をデプロイすると、次の操作を実行できます。

- AWS Management Consoleを使用して、各クラスターで[ワークロードに関する情報の表示](#)を実行できます。
- Kubernetes [Vertical Pod Autoscaler](#) を使用して、Pod を垂直方向にスケールアップまたはスケールダウンします。
- Kubernetes [Horizontal Pod Autoscaler](#) で需要を満たすために必要な Pod の数を水平方向にスケールリングします。
- 外部 (インターネットにアクセス可能な Pod の場合) または内部 (プライベート Pod の場合) [ネットワークロードバランサー](#)を作成して、Pod 間でネットワークトラフィックを分散させることができます。ロードバランサーは、OSI モデルのレイヤ 4 でトラフィックをルーティングします。
- [Application Load Balancer](#) を作成して、Pod 間でアプリケーショントラフィックを分散します。アプリケーションロードバランサーは、OSI モデルのレイヤ 7 でトラフィックをルーティングします。
- Kubernetes を初めて利用する場合、このトピックでは、[サンプルアプリケーションをデプロイ](#)します。
- externalIPs を使用して、[サービスに割り当てることができる IP アドレスを制限](#)できます。

## Linux サンプルアプリケーションをデプロイする

このトピックでは、サンプルアプリケーションを Linux ノードのクラスターにデプロイします。

### 前提条件

- 少なくとも 1 つのノードがある既存の Kubernetes クラスター。既存の Amazon EKS クラスターがない場合は、[はじめに](#) のガイドの 1 つを使用して Amazon EKS クラスターをデプロイできます。

- コンピュータに Kubectl がインストールされている。詳細については、「[the section called “kubectl と eksctl のセットアップ”](#)」を参照してください。
- クラスターと通信できるように Kubectl が設定されている。詳細については、「[the section called “kubectl を使用してクラスターにアクセスする”](#)」を参照してください。
- サンプルワークロードを Fargate にデプロイする予定の場合は、名前を変更しない限り、このチュートリアルで作成したのと同じ名前空間 eks-sample-app を含む既存の [Fargate プロファイル](#)が必要です。[はじめに](#) のいずれかを使用してクラスターを作成した場合は、新しいプロファイルを作成するか、既存のプロファイルに名前空間を追加する必要があります。これは、入門ガイドで作成されたプロファイルでは、このチュートリアルで使用する名前空間が指定されていないためです。VPC には、少なくとも 1 つのプライベートサブネットも必要です。

多くの変数は次のステップで変更できますが、変数値は、指定された場合のみ変更することをお勧めします。Kubernetes Pod、デプロイ、およびサービスへの理解を深めたら、他の値の変更を試すことができます。

## 名前空間を作成します。

名前空間を使用すると、Kubernetes 内のリソースをグループ化できます。詳細については、Kubernetes ドキュメントの[名前空間](#)を参照してください。[AWS Fargate でコンピューティング管理を簡素化](#)するためにサンプルアプリケーションをデプロイする場合は、[起動時にどのポッドが AWS Fargate を使用するかを定義する](#)の namespace の値が eks-sample-app であることを確認します。

```
kubectl create namespace eks-sample-app
```

## Kubernetes デプロイメントを作成する

Kubernetes デプロイメントを作成します。このサンプルデプロイでは、パブリックリポジトリからコンテナイメージをプルし、コンテナイメージの 3 つのレプリカ (個別の Pod) をクラスターにデプロイします。詳細については、Kubernetes ドキュメントの[デプロイ](#)を参照してください。

1. 次の内容を eks-sample-deployment.yaml という名前のファイルに保存します。サンプルアプリケーションのコンテナではネットワークストレージを使用しませんが、それを使用する必要があるアプリケーションが存在する場合があります。詳細については、「[アプリデータストレージ](#)」を参照してください。
  - kubernetes.io/arch キーの下に amd64 または arm64 values は、アプリケーションをいずれかのハードウェアアーキテクチャにデプロイできることを意味します (クラスター内に両方

がある場合)。これは、このイメージがマルチアーキテクチャイメージであるため可能ですが、すべてがそうであるわけではありません。イメージを取得するリポジトリ内の[イメージの詳細](#)を表示することで、イメージがサポートされているハードウェアアーキテクチャを判別できます。ハードウェアアーキテクチャのタイプをサポートしないイメージをデプロイする場合、またはイメージをデプロイしない場合は、そのタイプをマニフェストから削除します。詳細については、Kubernetes ドキュメントの[よく知られているラベル、注釈、ティント](#)を参照してください。

- `kubernetes.io/os: linux` nodeSelector は、クラスターに、たとえば、Linux ノードと Windows ノードがある場合、イメージは Linux ノードにのみデプロイされることを意味します。詳細については、Kubernetes ドキュメントの[よく知られているラベル、注釈、ティント](#)を参照してください。

```
apiVersion: apps/v1
kind: Deployment
metadata:
 name: eks-sample-linux-deployment
 namespace: eks-sample-app
 labels:
 app: eks-sample-linux-app
spec:
 replicas: 3
 selector:
 matchLabels:
 app: eks-sample-linux-app
 template:
 metadata:
 labels:
 app: eks-sample-linux-app
 spec:
 affinity:
 nodeAffinity:
 requiredDuringSchedulingIgnoredDuringExecution:
 nodeSelectorTerms:
 - matchExpressions:
 - key: kubernetes.io/arch
 operator: In
 values:
 - amd64
 - arm64
 containers:
 - name: nginx
```

```
image: public.ecr.aws/nginx/nginx:1.23
ports:
 - name: http
 containerPort: 80
imagePullPolicy: IfNotPresent
nodeSelector:
 kubernetes.io/os: linux
```

## 2. デプロイマニフェストをクラスターに適用します。

```
kubectl apply -f eks-sample-deployment.yaml
```

## サービスを作成する

サービスを利用すると、単一の IP アドレスまたは名前を使用して、すべてのレプリカにアクセスできます。詳細については、Kubernetes ドキュメントの[サービス](#)を参照してください。サンプルアプリケーションには実装されていませんが、他の AWS サービスとやり取りする必要があるアプリケーションがある場合、Pod の Kubernetes サービスアカウントを作成し、AWS IAM アカウントに関連付けることをお勧めします。サービスアカウントを指定すると、ポッドには、他のサービスとやり取りするために指定した最小限のアクセス許可だけが与えられます。詳細については、「[the section called “IRSA による認証情報”](#)」を参照してください。

1. 次の内容を eks-sample-service.yaml という名前のファイルに保存します。Kubernetes は、クラスター内からのみアクセスできる独自の IP アドレスをサービスに割り当てます。クラスターの外部からサービスにアクセスするには、[AWS Load Balancer Controller](#) をデプロイして、[アプリケーション](#)または[ネットワーク](#)トラフィックをサービスに負荷分散します。

```
apiVersion: v1
kind: Service
metadata:
 name: eks-sample-linux-service
 namespace: eks-sample-app
 labels:
 app: eks-sample-linux-app
spec:
 selector:
 app: eks-sample-linux-app
 ports:
 - protocol: TCP
 port: 80
```

```
targetPort: 80
```

## 2. サービスマニフェストをクラスターに適用します。

```
kubectl apply -f eks-sample-service.yaml
```

## 作成されたリソースを確認する

### 1. eks-sample-app 名前空間内に存在するすべてのリソースを表示します。

```
kubectl get all -n eks-sample-app
```

出力例は次のとおりです。

| NAME                                              | READY | STATUS  | RESTARTS | AGE |
|---------------------------------------------------|-------|---------|----------|-----|
| pod/eks-sample-linux-deployment-65b7669776-m6qxz  | 1/1   | Running | 0        | 27m |
| pod/eks-sample-linux-deployment-65b7669776-mmxxvd | 1/1   | Running | 0        | 27m |
| pod/eks-sample-linux-deployment-65b7669776-qzn22  | 1/1   | Running | 0        | 27m |

| NAME                             | TYPE      | CLUSTER-IP  | EXTERNAL-IP | PORT(S) |
|----------------------------------|-----------|-------------|-------------|---------|
| service/eks-sample-linux-service | ClusterIP | 10.100.74.8 | <none>      | 80/TCP  |

| NAME                                        | READY | UP-TO-DATE | AVAILABLE | AGE |
|---------------------------------------------|-------|------------|-----------|-----|
| deployment.apps/eks-sample-linux-deployment | 3/3   | 3          | 3         | 27m |

| NAME                                                   | DESIRED | CURRENT | READY |
|--------------------------------------------------------|---------|---------|-------|
| replicaset.apps/eks-sample-linux-deployment-776d8f8fd8 | 3       | 3       | 3     |

出力では、前の手順でデプロイしたサンプルマニフェストで指定されたサービスとデプロイが表示されます。また、3つのポッドも表示されます。これは、サンプルマニフェストで3 replicasが指定されたためです。ポッドの詳細については、Kubernetes ドキュメントの [Pods \(ポッド\)](#) を参照してください。Kubernetes では、サンプルマニフェストで指定されていなくても自動的に replicaset リソースが作成されます。ReplicaSets の詳細については、「Kubernetes ドキュメント」の [ReplicaSet](#) を参照してください。



 Note

Kubernetes では、マニフェストで指定したレプリカ数を維持できます。本稼働デプロイで、Kubernetes を使用してレプリカ数を水平にスケーリングしたり、Pod のコンピューティングリソースを垂直にスケーリングしたりする場合は、「[Horizontal Pod Autoscaler を使用してポッドデプロイをスケールする](#)」と「[Vertical Pod Autoscaler を使用してポッドリソースを調整する](#)」を使用します。

## 2. デプロイされたサービスの詳細を表示します。

```
kubectl -n eks-sample-app describe service eks-sample-linux-service
```

出力例は次のとおりです。

```
Name: eks-sample-linux-service
Namespace: eks-sample-app
Labels: app=eks-sample-linux-app
Annotations: <none>
Selector: app=eks-sample-linux-app
Type: ClusterIP
IP Families: <none>
IP: 10.100.74.8
IPs: 10.100.74.8
Port: <unset> 80/TCP
TargetPort: 80/TCP
Endpoints: 192.168.24.212:80,192.168.50.185:80,192.168.63.93:80
Session Affinity: None
Events: <none>
```

前の出力では、IP: の値は、クラスター内のどのノードまたは Pod からアクセスできる一意の IP アドレスですが、クラスターの外からはアクセスできません。Endpoints の値は、VPC 内からサービスの一部である Pod に割り当てられる IP アドレスです。

## 3. 前の手順で[名前空間を表示](#)したときに、出力にリストされたポッドのうちの 1 つの詳細を表示します。[776d8f8fd8-78w66](#) を、Pod の 1 つに対して返された値に置き換えます。

```
kubectl -n eks-sample-app describe pod eks-sample-linux-deployment-65b7669776-m6qxz
```

簡略化された出力例

```

Name: eks-sample-linux-deployment-65b7669776-m6qxz
Namespace: eks-sample-app
Priority: 0
Node: ip-192-168-45-132.us-west-2.compute.internal/192.168.45.132
[...]
IP: 192.168.63.93
IPs:
 IP: 192.168.63.93
Controlled By: ReplicaSet/eks-sample-linux-deployment-65b7669776
[...]
Conditions:
 Type Status
 Initialized True
 Ready True
 ContainersReady True
 PodScheduled True
[...]
Events:
 Type Reason Age From
 Message

 Normal Scheduled 3m20s default-scheduler
 Successfully assigned eks-sample-app/eks-sample-linux-deployment-65b7669776-m6qxz to
 ip-192-168-45-132.us-west-2.compute.internal
 [...]

```

前の出力では、IP: の値は CIDR ブロックから Pod に割り当てられる一意の IP です。CIDR ブロックは、ノードが存在するサブネットに割り当てられています。異なる CIDR ブロックからのポッド IP アドレスを割り当てたい場合は、デフォルトの動作を変更できます。詳細については、「[the section called “カスタムネットワーキング”](#)」を参照してください。また、Kubernetes スケジューラが IP アドレス **192.168.45.132** を使用して Node 上の Pod をスケジューリングしたことも確認できます。

#### Tip

コマンドラインを使用する代わりに、Pod、サービス、デプロイ、その他の Kubernetes リソースに関する多くの詳細を AWS Management Console に表示できます。詳細については、「[the section called “クラスターリソースにアクセスする”](#)」を参照してください。

## ポッドでシェルを実行する

1. 前のステップで説明したポッドでシェルを実行し、`65b7669776-m6qxz` をいずれかのポッドの ID で置き換えます。

```
kubectl exec -it eks-sample-linux-deployment-65b7669776-m6qxz -n eks-sample-app -- /bin/bash
```

2. Pod シェルから、前の手順でデプロイと共にインストールされたウェブサーバーからの出力を表示します。サービス名のみ指定する必要があります。デフォルトでは、Amazon EKS クラスターと共にデプロイされる CoreDNS によって、サービスの IP アドレスに解決されます。

```
curl eks-sample-linux-service
```

出力例は次のとおりです。

```
<!DOCTYPE html>
<html>
<head>
<title>Welcome to nginx!</title>
[...]
```

3. ポッドシェルから、ポッドの DNS サーバーを表示します。

```
cat /etc/resolv.conf
```

出力例は次のとおりです。

```
nameserver 10.100.0.10
search eks-sample-app.svc.cluster.local svc.cluster.local cluster.local us-
west-2.compute.internal
options ndots:5
```

前の出力では、クラスターにデプロイされたすべてのポッドに対して `10.100.0.10` が `nameserver` として自動的に割り当てられています。

4. `exit` を入力して、ポッドの接続を切断します。
5. サンプルアプリケーションの使用が終了したら、次のコマンドを使用して、サンプルの名前空間、サービス、デプロイを削除できます。

```
kubectl delete namespace eks-sample-app
```

## 次のステップ

サンプルアプリケーションをデプロイしたら、次の演習の一部を試すことができます。

- [Application Load Balancer を使用してアプリケーションと HTTP トラフィックをルーティングする](#)
- [Network Load Balancer を使用して TCP および UDP トラフィックをルーティングする](#)

## Windows サンプルアプリケーションをデプロイする

このトピックでは、サンプルアプリケーションを Windows ノードのクラスターにデプロイします。

### 前提条件

- 少なくとも 1 つのノードがある既存の Kubernetes クラスター。既存の Amazon EKS クラスターがない場合は、[はじめに](#) のガイドの 1 つを使用して Amazon EKS クラスターをデプロイできます。クラスターと少なくとも 1 つの Amazon EC2 Windows ノードで [Windows サポート](#) を有効にする必要があります。
- コンピュータに Kubectl がインストールされている。詳細については、「[the section called “kubectl と eksctl のセットアップ”](#)」を参照してください。
- クラスターと通信できるように Kubectl が設定されている。詳細については、「[the section called “kubectl を使用してクラスターにアクセスする”](#)」を参照してください。
- サンプルワークロードを Fargate にデプロイする予定の場合は、名前を変更しない限り、このチュートリアルで作成したのと同じ名前空間 eks-sample-app を含む既存の [Fargate プロファイル](#) が必要です。[はじめに](#) のいずれかを使用してクラスターを作成した場合は、新しいプロファイルを作成するか、既存のプロファイルに名前空間を追加する必要があります。これは、入門ガイドで作成されたプロファイルでは、このチュートリアルで使用する名前空間が指定されていないためです。VPC には、少なくとも 1 つのプライベートサブネットも必要です。

多くの変数は次のステップで変更できますが、変数値は、指定された場合のみ変更することをお勧めします。Kubernetes Pod、デプロイ、およびサービスへの理解を深めたら、他の値の変更を試すことができます。

## 名前空間を作成します。

名前空間を使用すると、Kubernetes 内のリソースをグループ化できます。詳細については、Kubernetes ドキュメントの[名前空間](#)を参照してください。[AWS Fargate でコンピューティング管理を簡素化](#)するためにサンプルアプリケーションをデプロイする場合は、[起動時にどのポッドが AWS Fargate を使用するかを定義する](#)の namespace の値が eks-sample-app であることを確認します。

```
kubectl create namespace eks-sample-app
```

## Kubernetes デプロイメントを作成する

このサンプルデプロイでは、パブリックリポジトリからコンテナイメージをプルし、コンテナイメージの 3 つのレプリカ (個別の Pod) をクラスターにデプロイします。詳細については、Kubernetes ドキュメントの[デプロイ](#)を参照してください。

1. 次の内容を eks-sample-deployment.yaml という名前のファイルに保存します。サンプルアプリケーションのコンテナではネットワークストレージを使用しませんが、それを使用する必要があるアプリケーションが存在する場合があります。詳細については、「[アプリデータストレージ](#)」を参照してください。
- kubernetes.io/os: windows nodeSelector は、クラスターに、たとえば、Windows ノードと Linux ノードがある場合、イメージは Windows ノードにのみデプロイされることを意味します。詳細については、Kubernetes ドキュメントの[よく知られているラベル、注釈、ティント](#)を参照してください。

```
apiVersion: apps/v1
kind: Deployment
metadata:
 name: eks-sample-windows-deployment
 namespace: eks-sample-app
 labels:
 app: eks-sample-windows-app
spec:
 replicas: 3
 selector:
 matchLabels:
 app: eks-sample-windows-app
 template:
 metadata:
```

```
labels:
 app: eks-sample-windows-app
spec:
 affinity:
 nodeAffinity:
 requiredDuringSchedulingIgnoredDuringExecution:
 nodeSelectorTerms:
 - matchExpressions:
 - key: beta.kubernetes.io/arch
 operator: In
 values:
 - amd64
 containers:
 - name: windows-server-iis
 image: mcr.microsoft.com/windows/servercore:ltsc2019
 ports:
 - name: http
 containerPort: 80
 imagePullPolicy: IfNotPresent
 command:
 - powershell.exe
 - -command
 - "Add-WindowsFeature Web-Server; Invoke-WebRequest -UseBasicParsing
 -Uri 'https://dotnetbinaries.blob.core.windows.net/servicemonitor/2.0.1.6/
 ServiceMonitor.exe' -OutFile 'C:\\ServiceMonitor.exe'; echo '<html><body><br/
 >
<marquee><H1>Hello EKS!!!<H1><marquee></body><html>' > C:\\inetpub\\wwwroot\\
 \\default.html; C:\\ServiceMonitor.exe 'w3svc'; "
 nodeSelector:
 kubernetes.io/os: windows
```

## 2. デプロイマニフェストをクラスターに適用します。

```
kubectl apply -f eks-sample-deployment.yaml
```

## サービスを作成する

サービスを利用すると、単一の IP アドレスまたは名前を使用して、すべてのレプリカにアクセスできます。詳細については、Kubernetes ドキュメントの[サービス](#)を参照してください。サンプルアプリケーションには実装されていませんが、他の AWS サービスとやり取りする必要があるアプリケーションがある場合、Pod の Kubernetes サービスアカウントを作成し、AWS IAM アカウントに関連付けることをお勧めします。サービスアカウントを指定すると、ポッドには、他のサービスとやり

取りするために指定した最小限のアクセス許可だけが与えられます。詳細については、「[the section called “IRSA による認証情報”](#)」を参照してください。

1. 次の内容を `eks-sample-service.yaml` という名前のファイルに保存します。Kubernetes は、クラスター内からのみアクセスできる独自の IP アドレスをサービスに割り当てます。クラスターの外部からサービスにアクセスするには、[AWS Load Balancer Controller](#) をデプロイして、[アプリケーション](#)または[ネットワーク](#)トラフィックをサービスに負荷分散します。

```
apiVersion: v1
kind: Service
metadata:
 name: eks-sample-windows-service
 namespace: eks-sample-app
 labels:
 app: eks-sample-windows-app
spec:
 selector:
 app: eks-sample-windows-app
 ports:
 - protocol: TCP
 port: 80
 targetPort: 80
```

2. サービスマニフェストをクラスターに適用します。

```
kubectl apply -f eks-sample-service.yaml
```

## 作成されたリソースを確認する

1. `eks-sample-app` 名前空間内に存在するすべてのリソースを表示します。

```
kubectl get all -n eks-sample-app
```

出力例は次のとおりです。

| NAME                                               | READY | STATUS  | RESTARTS | AGE |
|----------------------------------------------------|-------|---------|----------|-----|
| pod/eks-sample-windows-deployment-65b7669776-m6qxz | 1/1   | Running | 0        | 27m |
| pod/eks-sample-windows-deployment-65b7669776-mmxvd | 1/1   | Running | 0        | 27m |
| pod/eks-sample-windows-deployment-65b7669776-qzn22 | 1/1   | Running | 0        | 27m |

| NAME                                                                   | TYPE      | CLUSTER-IP  | EXTERNAL-IP | PORT(S) |
|------------------------------------------------------------------------|-----------|-------------|-------------|---------|
| AGE<br>service/eks-sample-windows-service<br>TCP 32m                   | ClusterIP | 10.100.74.8 | <none>      | 80/     |
| NAME                                                                   | READY     | UP-TO-DATE  | AVAILABLE   | AGE     |
| deployment.apps/eks-sample-windows-deployment                          | 3/3       | 3           | 3           | 27m     |
| NAME                                                                   | DESIRED   | CURRENT     | READY       |         |
| AGE<br>replicaset.apps/eks-sample-windows-deployment-776d8f8fd8<br>27m | 3         | 3           | 3           |         |

出力では、前の手順でデプロイしたサンプルマニフェストで指定されたサービスとデプロイが表示されます。また、3つのポッドも表示されます。これは、サンプルマニフェストで3 replicasが指定されたためです。ポッドの詳細については、Kubernetes ドキュメントの [Pods \(ポッド\)](#) を参照してください。Kubernetes では、サンプルマニフェストで指定されていなくても自動的に replicaset リソースが作成されます。ReplicaSets の詳細については、「Kubernetes ドキュメント」の「[ReplicaSet](#)」を参照してください。

#### Note

Kubernetes では、マニフェストで指定したレプリカ数を維持できます。本稼働デプロイで、Kubernetes を使用してレプリカ数を水平にスケーリングしたり、Pod のコンピューティングリソースを垂直にスケーリングしたりする場合は、「[Horizontal Pod Autoscaler を使用してポッドデプロイをスケールする](#)」と「[Vertical Pod Autoscaler を使用してポッドリソースを調整する](#)」を使用します。

## 2. デプロイされたサービスの詳細を表示します。

```
kubectl -n eks-sample-app describe service eks-sample-windows-service
```

出力例は次のとおりです。

```
Name: eks-sample-windows-service
Namespace: eks-sample-app
Labels: app=eks-sample-windows-app
Annotations: <none>
Selector: app=eks-sample-windows-app
Type: ClusterIP
```



```

IP Families: <none>
IP: 10.100.74.8
IPs: 10.100.74.8
Port: <unset> 80/TCP
TargetPort: 80/TCP
Endpoints: 192.168.24.212:80,192.168.50.185:80,192.168.63.93:80
Session Affinity: None
Events: <none>

```

前の出力では、IP: の値は、クラスター内のどのノードまたは Pod からアクセスできる一意の IP アドレスですが、クラスターの外からはアクセスできません。Endpoints の値は、VPC 内からサービスの一部である Pod に割り当てられる IP アドレスです。

3. 前の手順で[名前空間を表示](#)したときに、出力にリストされたポッドのうちの 1 つの詳細を表示します。`776d8f8fd8-78w66` を、Pod の 1 つに対して返された値に置き換えます。

```
kubectl -n eks-sample-app describe pod eks-sample-windows-deployment-65b7669776-m6qxz
```

### 簡略化された出力例

```

Name: eks-sample-windows-deployment-65b7669776-m6qxz
Namespace: eks-sample-app
Priority: 0
Node: ip-192-168-45-132.us-west-2.compute.internal/192.168.45.132
[...]
IP: 192.168.63.93
IPs:
 IP: 192.168.63.93
Controlled By: ReplicaSet/eks-sample-windows-deployment-65b7669776
[...]
Conditions:
 Type Status
 Initialized True
 Ready True
 ContainersReady True
 PodScheduled True
[...]
Events:
 Type Reason Age From
 Message


```

```
Normal Scheduled 3m20s default-scheduler
Successfully assigned eks-sample-app/eks-sample-windows-deployment-65b7669776-m6qxz
to ip-192-168-45-132.us-west-2.compute.internal
[...]
```

前の出力では、IP: の値は CIDR ブロックから Pod に割り当てられる一意の IP です。CIDR ブロックは、ノードが存在するサブネットに割り当てられています。異なる CIDR ブロックからのポッド IP アドレスを割り当てたい場合は、デフォルトの動作を変更できます。詳細については、「[the section called “カスタムネットワーキング”](#)」を参照してください。また、Kubernetes スケジューラが IP アドレス **192.168.45.132** を使用して Node 上の Pod をスケジューリングしたことも確認できます。

#### Tip

コマンドラインを使用する代わりに、Pod、サービス、デプロイ、その他の Kubernetes リソースに関する多くの詳細を AWS Management Console に表示できます。詳細については、「[the section called “クラスターリソースにアクセスする”](#)」を参照してください。

## ポッドでシェルを実行する

1. 前のステップで説明したポッドでシェルを実行し、**65b7669776-m6qxz** をいずれかのポッドの ID で置き換えます。

```
kubectl exec -it eks-sample-windows-deployment-65b7669776-m6qxz -n eks-sample-app -- powershell.exe
```

2. Pod シェルから、前の手順でデプロイと共にインストールされたウェブサーバーからの出力を表示します。サービス名のみ指定する必要があります。デフォルトでは、Amazon EKS クラスターと共にデプロイされる CoreDNS によって、サービスの IP アドレスに解決されます。

```
Invoke-WebRequest -uri eks-sample-windows-service/default.html -UseBasicParsing
```

出力例は次のとおりです。

```
StatusCode : 200
StatusDescription : OK
Content : < h t m l > < b o d y > < b r / > < b r / > < m a r q u e e > < H
 1 > H e l l o
```

```
E K S ! ! ! < H 1 > < m a r q u e e > < / b o d y > < h t m l >
```

### 3. ポッドシェルから、ポッドの DNS サーバーを表示します。

```
Get-NetIPConfiguration
```

#### 省略された出力

```
InterfaceAlias : vEthernet
[...]
IPv4Address : 192.168.63.14
[...]
DNSServer : 10.100.0.10
```

前の出力では、クラスターにデプロイされたすべてのポッドに対して 10.100.0.10 が DNS サーバーとして自動的に割り当てられます。

4. `exit` を入力して、ポッドの接続を切断します。
5. サンプルアプリケーションの使用が終了したら、次のコマンドを使用して、サンプルの名前空間、サービス、デプロイを削除できます。

```
kubectl delete namespace eks-sample-app
```

## 次のステップ

サンプルアプリケーションをデプロイしたら、次の演習の一部を試すことができます。

- [Application Load Balancer を使用してアプリケーションと HTTP トラフィックをルーティングする](#)
- [Network Load Balancer を使用して TCP および UDP トラフィックをルーティングする](#)

## Vertical Pod Autoscaler を使用してポッドリソースを調整する

Kubernetes [Vertical Pod Autoscaler](#) は、Pod の CPU とメモリの予約を自動的に調整し、アプリケーションを「適切なサイズ」にします。この調整により、クラスターリソースの使用率が向上し、他の Pod の CPU とメモリが解放されます。このトピックでは、Vertical Pod Autoscaler をクラスターにデプロイし、動作していることを確認する方法について説明します。

- 既存の Amazon EKS クラスターがあります。そうでない場合は、「[はじめに](#)」を参照してください。
- Kubernetes メトリクスサーバーがインストールされています。詳細については、「」を参照してください[the section called “メトリクスサーバー”](#)
- [Amazon EKS クラスターと通信するように設定された](#) kubectl クライアントを使用しています。
- デバイスに、OpenSSL 1.1.1 以降がインストールされています。

## Vertical Pod Autoscaler をデプロイする

このセクションでは、Vertical Pod Autoscaler をクラスターにデプロイします。

1. ターミナルウィンドウを開き、Vertical Pod Autoscaler ソースコードをダウンロードするディレクトリに移動します。
2. [kubernetes/autoscaler](#) GitHub リポジトリのクローンを作成します。

```
git clone https://github.com/kubernetes/autoscaler.git
```

3. vertical-pod-autoscaler ディレクトリを変更します。

```
cd autoscaler/vertical-pod-autoscaler/
```

4. (オプション) Vertical Pod Autoscaler の別のバージョンをすでにデプロイしている場合は、次のコマンドを使用して削除します。

```
./hack/vpa-down.sh
```

5. ノードが registry.k8s.io コンテナレジストリに対してインターネットからアクセスできない場合は、次のイメージをプルして、独自のプライベートリポジトリにプッシュする必要があります。イメージをプルして独自のプライベートリポジトリにプッシュする方法の詳細については、「[the section called “リポジトリにイメージをコピーする”](#)」を参照してください。

```
registry.k8s.io/autoscaling/vpa-admission-controller:0.10.0
registry.k8s.io/autoscaling/vpa-recommender:0.10.0
registry.k8s.io/autoscaling/vpa-updater:0.10.0
```

プライベートの Amazon ECR リポジトリにイメージをプッシュする場合は、マニフェストの registry.k8s.io をレジストリに置き換えます。**111122223333** は、ご自分のアカウント ID に置き換えます。**region-code** を、クラスターのある AWS リージョンに置き換えます。次のコ

マンドは、お使いのリポジトリにマニフェストのリポジトリ名と同じ名前を付けていることを前提としています。リポジトリに別名を付けた場合も、同様に変更する必要があります。

```
sed -i.bak -e 's/registry.k8s.io/111122223333.dkr.ecr.region-code.amazonaws.com/' ./
deploy/admission-controller-deployment.yaml
sed -i.bak -e 's/registry.k8s.io/111122223333.dkr.ecr.region-code.amazonaws.com/' ./
deploy/recommender-deployment.yaml
sed -i.bak -e 's/registry.k8s.io/111122223333.dkr.ecr.region-code.amazonaws.com/' ./
deploy/updater-deployment.yaml
```

6. 次のコマンドを使用して、Vertical Pod Autoscaler をクラスターにデプロイします。

```
./hack/vpa-up.sh
```

7. Vertical Pod Autoscaler Pod が正常に作成されたことを確認します。

```
kubectl get pods -n kube-system
```

出力例は次のとおりです。

| NAME                                      | READY | STATUS  | RESTARTS | AGE |
|-------------------------------------------|-------|---------|----------|-----|
| [...]                                     |       |         |          |     |
| metrics-server-8459fc497-kfj8w            | 1/1   | Running | 0        | 83m |
| vpa-admission-controller-68c748777d-ppspd | 1/1   | Running | 0        | 7s  |
| vpa-recommender-6fc8c67d85-gljpl          | 1/1   | Running | 0        | 8s  |
| vpa-updater-786b96955c-bgp9d              | 1/1   | Running | 0        | 8s  |

## Vertical Pod Autoscaler のインストールをテストします

このセクションでは、サンプルアプリケーションをデプロイして、Vertical Pod Autoscaler が動作していることを確認します。

1. 次のコマンドを使用して hamster.yaml Vertical Pod Autoscaler の例をデプロイします。

```
kubectl apply -f examples/hamster.yaml
```

2. hamster サンプルアプリケーションから Pod を取得します。

```
kubectl get pods -l app=hamster
```

出力例は次のとおりです。

```
hamster-c7d89d6db-rglf5 1/1 Running 0 48s
hamster-c7d89d6db-znvz5 1/1 Running 0 48s
```

3. cpu と memory の予約を表示する Pod の 1 つを説明します。 *c7d89d6db-rglf5* を前のステップの出力で返された ID の 1 つに置き換えます。

```
kubectl describe pod hamster-c7d89d6db-rglf5
```

出力例は次のとおりです。

```
[...]
Containers:
 hamster:
 Container ID: docker://
e76c2413fc720ac395c33b64588c82094fc8e5d590e373d5f818f3978f577e24
 Image: registry.k8s.io/ubuntu-slim:0.1
 Image ID: docker-pullable://registry.k8s.io/ubuntu-
slim@sha256:b6f8c3885f5880a4f1a7cf717c07242eb4858fdd5a84b5ffe35b1cf680ea17b1
 Port: <none>
 Host Port: <none>
 Command:
 /bin/sh
 Args:
 -c
 while true; do timeout 0.5s yes >/dev/null; sleep 0.5s; done
 State: Running
 Started: Fri, 27 Sep 2019 10:35:16 -0700
 Ready: True
 Restart Count: 0
 Requests:
 cpu: 100m
 memory: 50Mi
[...]
```

元の Pod は 100 millicpu の CPU と 50 メビバイトのメモリを予約していることがわかります。このサンプルアプリケーションでは、100 millicpu は Pod の実行に必要な数よりも少ないため、CPU に制約があります。また、必要とするよりもはるかに少ないメモリを予約します。Vertical Pod Autoscaler `vpa-recommender` のデプロイでは、hamster Pod を分析し

て、CPU とメモリの要件が適切かどうかを確認します。調整が必要な場合、vpa-updater は更新された値で Pod を再起動します。

4. vpa-updater が新しい hamster Pod を起動するまで待ちます。これには 1~2 分かかります。次のコマンドを使用して、Pod をモニタリングできます。

 Note

新しい Pod が起動されたかどうか不明な場合は、Pod 名を前のリストと比較します。新しい Pod が起動すると、新しい Pod 名が表示されます。

```
kubectl get --watch Pods -l app=hamster
```

5. 新しい hamster Pod が開始されたら、それについて説明し、更新された CPU とメモリの予約を表示します。

```
kubectl describe pod hamster-c7d89d6db-jxgfv
```

出力例は次のとおりです。

```
[...]
Containers:
 hamster:
 Container ID:
 docker://2c3e7b6fb7ce0d8c86444334df654af6fb3fc88aad4c5d710eac3b1e7c58f7db
 Image: registry.k8s.io/ubuntu-slim:0.1
 Image ID: docker-pullable://registry.k8s.io/ubuntu-
slim@sha256:b6f8c3885f5880a4f1a7cf717c07242eb485fdd5a84b5ffe35b1cf680ea17b1
 Port: <none>
 Host Port: <none>
 Command:
 /bin/sh
 Args:
 -c
 while true; do timeout 0.5s yes >/dev/null; sleep 0.5s; done
 State: Running
 Started: Fri, 27 Sep 2019 10:37:08 -0700
 Ready: True
 Restart Count: 0
 Requests:
```

```

 cpu: 587m
 memory: 262144k
 [...]

```

以前の出力では、cpu 予約が元の値の 5 倍以上である 587 millicpu に増加したことがわかります。memory は 262,144 キロバイト (約 250 メビバイト、つまり元の値の 5 倍) に増加しました。この Pod はリソース不足だったため、Vertical Pod Autoscaler で見積りをより適切な値で修正しました。

## 6. hamster-vpa リソースの詳細を表示して、新しい推奨事項を表示します。

```
kubectl describe vpa/hamster-vpa
```

出力例は次のとおりです。

```

Name: hamster-vpa
Namespace: default
Labels: <none>
Annotations: kubectl.kubernetes.io/last-applied-configuration:
 {"apiVersion":"autoscaling.k8s.io/v1beta2", "kind": "VerticalPodAutoscaler", "metadata": {"annotations": {}, "name": "hamster-
vpa", "namespace": "d...
API Version: autoscaling.k8s.io/v1beta2
Kind: VerticalPodAutoscaler
Metadata:
 Creation Timestamp: 2019-09-27T18:22:51Z
 Generation: 23
 Resource Version: 14411
 Self Link: /apis/autoscaling.k8s.io/v1beta2/namespaces/default/
verticalpodautoscalers/hamster-vpa
 UID: d0d85fb9-e153-11e9-ae53-0205785d75b0
Spec:
 Target Ref:
 API Version: apps/v1
 Kind: Deployment
 Name: hamster
Status:
 Conditions:
 Last Transition Time: 2019-09-27T18:23:28Z
 Status: True
 Type: RecommendationProvided
 Recommendation:

```



## Container Recommendations:

Container Name: hamster

## Lower Bound:

Cpu: 550m

Memory: 262144k

## Target:

Cpu: 587m

Memory: 262144k

## Uncapped Target:

Cpu: 587m

Memory: 262144k

## Upper Bound:

Cpu: 21147m

Memory: 387863636

Events: &lt;none&gt;

7. アプリケーション例の試用が終了したら、次のコマンドで削除します。

```
kubectl delete -f examples/hamster.yaml
```

## Horizontal Pod Autoscaler を使用してポッドデプロイをスケールする

Kubernetes [Horizontal Pod Autoscaler](#) は、そのリソースの CPU 使用率に基づいて、デプロイ、レプリケーションコントローラー、またはレプリカセット内の Pod の数を自動的にスケールリングします。これにより、アプリケーションは需要の増加に合わせてスケールアウトしたり、リソースがなくなったときにスケールインしたりできるため、ノードを他のアプリケーションに解放できます。ターゲットの CPU 使用率を設定すると、Horizontal Pod Autoscaler はターゲットを満たすようにアプリケーションをスケールインまたはスケールアウトします。

Horizontal Pod Autoscaler は、Kubernetes の標準の API リソースで、Amazon EKS クラスターにメトリクスソース (Kubernetes メトリクスサーバーなど) がインストールされているだけで動作します。アプリケーションのスケールリングを開始するために、クラスターに Horizontal Pod Autoscaler をデプロイまたはインストールする必要はありません。詳細については、Kubernetes ドキュメントの「[Horizontal Pod Autoscaler](#)」を参照してください。

このトピックでは、Amazon EKS クラスターの Horizontal Pod Autoscaler の準備や、サンプルアプリケーションで動作することの確認方法を説明しています。

**Note**

このトピックは、Kubernetes ドキュメントの「[Horizontal Pod autoscaler walkthrough](#)」に基づいています。

- 既存の Amazon EKS クラスターがあります。そうでない場合は、「[はじめに](#)」を参照してください。
- Kubernetes メトリクスサーバーがインストールされています。詳細については、「」を参照してください[the section called “メトリクスサーバー”](#)
- [Amazon EKS クラスターと通信するように設定された](#) kubectl クライアントを使用しています。

## Horizontal Pod Autoscaler テストアプリケーションを実行する

このセクションでは、サンプルアプリケーションをデプロイして、Horizontal Pod Autoscaler が動作していることを確認します。

**Note**

この例は、Kubernetes ドキュメントの「[Horizontal Pod autoscaler walkthrough](#)」に基づいています。

1. 次のコマンドを使用して、シンプルな Apache ウェブサーバーアプリケーションをデプロイします。

```
kubectl apply -f https://k8s.io/examples/application/php-apache.yaml
```

この Apache ウェブサーバー Pod には 500 millicpu の CPU 制限が与えられ、サービスはポート 80 で提供されています。

2. php-apache デプロイ用の Horizontal Pod Autoscaler リソースを作成します。

```
kubectl autoscale deployment php-apache --cpu-percent=50 --min=1 --max=10
```

このコマンドは、デプロイの CPU 使用率の目標が 50%、Pod の数が最低 1 つ、最大 10 個の Autoscaler を作成します。CPU の平均負荷が 50% を下回ると、Autoscaler はデプロイの Pod 数

を最小の 1 に減らそうとします。負荷が 50 % を超えると、Autoscaler はデプロイの Pod 数を最大の 10 まで増やそうとします。詳細については、Kubernetes ドキュメントの「[How does a HorizontalPodAutoscaler work?](#)」を参照してください。

3. 次のコマンドで Autoscaler の詳細を表示します。

```
kubectl get hpa
```

出力例は次のとおりです。

| NAME       | REFERENCE             | TARGETS | MINPODS | MAXPODS | REPLICAS | AGE |
|------------|-----------------------|---------|---------|---------|----------|-----|
| php-apache | Deployment/php-apache | 0%/50%  | 1       | 10      | 1        | 51s |

ご覧のように、現在の CPU 負荷は 0% です。サーバーにまだ負荷がないためです。Pod 数は既に最低の境界 (1) にあるため、スケールインすることはできません。

4. コンテナを実行して、ウェブサーバーのロードを作成します。

```
kubectl run -i \
 --tty load-generator \
 --rm --image=busybox \
 --restart=Never \
 -- /bin/sh -c "while sleep 0.01; do wget -q -O- http://php-apache; done"
```

5. デプロイのスケールアウトを監視するには、前のステップを実行したターミナルとは別のターミナルで次のコマンドを定期的に実行します。

```
kubectl get hpa php-apache
```

出力例は次のとおりです。

| NAME       | REFERENCE             | TARGETS  | MINPODS | MAXPODS | REPLICAS | AGE   |
|------------|-----------------------|----------|---------|---------|----------|-------|
| php-apache | Deployment/php-apache | 250%/50% | 1       | 10      | 5        | 4m44s |

レプリカ数が増えるまで、1 分以上かかる場合があります。実際の CPU のパーセンテージが目標のパーセンテージよりも高い限り、レプリカ数は最大 10 まで増加します。この場合は 250% のため、REPLICAS の数は増え続けます。

 Note

レプリカの数が増えるまで、数分かかる場合があります。例えば、CPU 負荷が 50% 以下に保つために必要なレプリカの数が増える場合、負荷はレプリカの数を超えてスケールすることはありません。

6. 負荷を停止します。負荷を生成しているターミナルウィンドウで、Ctrl+C キーを押して負荷を停止します。スケールインを監視しているターミナルで次のコマンドを再度実行することで、レプリカの数が増えるのを確認できます。

```
kubectl get hpa
```

出力例は次のとおりです。

| NAME       | REFERENCE             | TARGETS | MINPODS | MAXPODS | REPLICAS | AGE |
|------------|-----------------------|---------|---------|---------|----------|-----|
| php-apache | Deployment/php-apache | 0%/50%  | 1       | 10      | 1        | 25m |

 Note

現在の CPU の割合が 0% であっても、スケールダウンのデフォルトの時間枠は 5 分であるため、レプリカの数が増えるまでには時間がかかります。時間枠は変更可能です。詳細については、Kubernetes ドキュメントの「[Horizontal Pod Autoscaler](#)」を参照してください。

7. サンプルアプリケーションのテストが終了したら、php-apache リソースを削除します。

```
kubectl delete deployment.apps/php-apache service/php-apache
horizontalpodautoscaler.autoscaling/php-apache
```

# Network Load Balancer を使用して TCP および UDP トラフィックをルーティングする

## Note

新規: Amazon EKS Auto Mode は、ロードバランシングの定型作業を自動化します。詳細については、以下を参照してください。

- [the section called “ロードバランサーをデプロイする”](#)
- [the section called “サービスの作成”](#)

ネットワークトラフィックは、OSI モデルの L4 で負荷分散されます。L7 でアプリケーショントラフィックの負荷を分散するには、Kubernetes ingress をデプロイし、これによって AWS Application Load Balancer をプロビジョニングします。詳細については、「[the section called “アプリケーション負荷分散”](#)」を参照してください。2 種類の負荷分散の違いについては、AWS ウェブサイトの「[Elastic Load Balancing の特徴](#)」を参照してください。

タイプ LoadBalancer の Kubernetes Service を作成する際、デフォルトでは、AWS クラウドプロバイダーロードバランサーコントローラーにより AWS [Classic Load Balancer](#) が作成されますが、AWS [Network Load Balancer](#) も作成できます。このコントローラーは、将来の重大なバグ修正のみを受けています。AWS クラウドプロバイダーのロードバランサーの使い方については、Kubernetes ドキュメントの「[AWS cloud provider load balancer controller](#)」を参照してください。使用方法については、このトピックでは説明しません。

AWS クラウドプロバイダーロードバランサーコントローラーの代わりに、[AWS ロードバランサーコントローラー](#) のバージョン 2.7.2 以降を使用することをお勧めします。AWS Load Balancer Controller では、AWS Network Load Balancer は作成できますが、AWS Classic Load Balancer は作成できません。このトピックの残りの部分では、AWS Load Balancer Controller の使用について説明します。

AWS Network Load Balancer では、Amazon EC2 IP およびインスタンス[ターゲット](#)、AWS Fargate IP ターゲット、または IP ターゲットとして Amazon EKS Hybrid Nodes にデプロイされた Pod にネットワークトラフィックを負荷分散できます。詳細については、GitHub の [AWS Load Balancer Controller](#) を参照してください。

## 前提条件

AWS Load Balancer Controller を使用してネットワークトラフィックの負荷分散を行うには、次の要件を満たす必要があります。

- 既存のクラスターがある。既存のクラスターがない場合は、「[はじめに](#)」を参照してください。既存のクラスターのバージョンを更新する必要がある場合は、[the section called “Kubernetes バージョンを更新する”](#) を参照して下さい。
- クラスターに AWS Load Balancer Controller がデプロイされている。詳細については、「[the section called “AWS ロードバランサーコントローラー”](#)」を参照してください。バージョン 2.7.2 以降をお勧めします。
- 少なくとも 1 つのサブネット。アベイラビリティゾーンで複数のタグ付きサブネットが見つかった場合、コントローラーは、サブネット ID が辞書式順序で最初に来る最初のサブネットを選択します。サブネットには、利用可能な IP アドレスが最低 8 個必要です。
- AWS Load Balancer Controller バージョン 2.1.1 以前を使用している場合は、次のようにサブネットにタグ付けする必要があります。バージョン 2.1.2 以降を使用している場合、このタグはオプションです。同じ VPC 内で複数のクラスターを実行していたり、VPC 内で複数の AWS のサービスがサブネットを共有していたりして、クラスターごとにロードバランサーがプロビジョニングされる場所をより詳細に制御したい場合は、サブネットにタグを付けることができます。サービスオブジェクトのアノテーションとしてサブネット ID を明示的に指定すると、Kubernetes と AWS Load Balancer Controller は、これらのサブネットを直接使用してロードバランサーを作成します。ロードバランサーのプロビジョニングにこの方法を使用することを選択した場合、サブネットのタグ付けは不要です。次のプライベートサブネットおよびパブリックサブネットのタグ付け要件は省略できます。##### の部分は自分のクラスター名に置き換えます。
  - キー – `kubernetes.io/cluster/<my-cluster>`
  - 値 – `shared` または `owned`
- サービスオブジェクトまたは Ingress オブジェクトのアノテーションとしてサブネット ID を明示的に指定しない場合、パブリックサブネットとプライベートサブネットは次の要件を満たしている必要があります。サービスオブジェクトやイングレスオブジェクトのアノテーションとしてサブネット ID を明示的に指定してロードバランサーをプロビジョニングする場合、Kubernetes と AWS Load Balancer Controller はこれらのサブネットを直接使用してロードバランサーを作成します。次のタグは必要ありません。
  - プライベートサブネット — 次の形式でタグ付けする必要があります。これは、Kubernetes と AWS Load Balancer Controller が、サブネットを内部ロードバランサーに使用できることを認識できるようにするためです。eksctl または Amazon EKS AWS CloudFormation テンプレート

トを使用して、2020 年 3 月 26 日以降に VPC を作成する場合、サブネットは、作成時に適切にタグ付けされます。Amazon EKS AWS CloudFormation VPC テンプレートの詳細については、「[the section called “「VPC を作成する」”](#)」を参照してください。

- キー – `kubernetes.io/role/internal-elb`
- 値 – 1
- パブリックサブネット — 次の形式でタグ付けする必要があります。これは、各アベイラビリティゾーンでパブリックサブネットを選択するのではなく (サブネット ID の辞書順)、Kubernetes がそれらのサブネットのみを外部ロードバランサーに使用することを認識できるようにするためです。eksctl または Amazon EKS AWS CloudFormation テンプレートを使用して、2020 年 3 月 26 日以降に VPC を作成する場合、サブネットは、作成時に適切にタグ付けされます。Amazon EKS AWS CloudFormation VPC テンプレートの詳細については、「[the section called “「VPC を作成する」”](#)」を参照してください。
- キー – `kubernetes.io/role/elb`
- 値 – 1

サブネットロールタグが明示的に追加されていない場合、Kubernetes サービスコントローラーはクラスター VPC サブネットのルートテーブルを調べて、サブネットがプライベートであるかパブリックであるかを判断します。この動作に頼らず、プライベートまたはパブリックロールタグを明示的に追加することをお勧めします。AWS Load Balancer Controller では、ルートテーブルを検査しないため、自動検出を正常に実行するにはプライベートタグとパブリックタグが必要です。

## 考慮事項

- ロードバランサーの設定は、サービス用のマニフェストに追加された注釈により制御されます。AWS Load Balancer Controller を使用する場合は、サービスアノテーションは、AWS クラウドプロバイダーロードバランサーコントローラーを使用する場合のサービスアノテーションとは異なります。サービスをデプロイする前に、必ず AWS Load Balancer Controller の[アノテーション](#)を確認してください。
- [Amazon VPC CNI plugin for Kubernetes](#) を使用する場合は、AWS Load Balancer Controller は Amazon EC2 の IP ターゲットがインスタスタターゲット、および Fargate の IP ターゲットに負荷分散できます。[互換性のある代替 CNI プラグイン](#)を使用する場合、コントローラーはインスタスタターゲットにのみ負荷分散できます (Amazon EKS Hybrid Nodes に負荷分散している場合を除く)。ハイブリッドノードの場合、コントローラーは IP ターゲットを負荷分散できません。Network Load Balancer のターゲットタイプの詳細については、「Network Load Balancer のユーザーガイド」の「[Target type](#)」(ターゲットタイプ)を参照してください。



- ロードバランサーの作成時 (または作成後) にタグを追加する場合は、サービス仕様に次のアノテーションを追加します。詳細については、AWS Load Balancer Controller ドキュメントの[AWS リソースタグ](#)を参照してください。

```
service.beta.kubernetes.io/aws-load-balancer-additional-resource-tags
```

- [Elastic IP アドレス](#)を Network Load Balancer に割り当てるには、次のアノテーションを追加します。####を Elastic IP アドレスの Allocation IDs に置き換えます。Allocation IDs の数は、ロードバランサーに使用されるサブネットの数に一致する必要があります。詳細については、[AWS Load Balancer Controller](#) ドキュメントを参照してください。

```
service.beta.kubernetes.io/aws-load-balancer-eip-allocations: eipalloc-
xxxxxxxxxxxxxxxxxxxxx,eipalloc-yyyyyyyyyyyyyyyyyyy
```

- Amazon EKS は、クライアントトラフィック用のインバウンドルールをノードのセキュリティグループに 1 つ追加し、VPC 内のロードバランサーのサブネットごとに、作成する各 Network Load Balancer のヘルスチェック用のルールを 1 つずつ追加します。タイプ LoadBalancer のサービスの展開は、Amazon EKS が、セキュリティグループに許可されているルールの最大数に対するクォータを超えるルールを作成しようとした場合に失敗することがあります。詳細については、Amazon VPC ユーザーガイドの「[セキュリティグループ](#)」を参照してください。セキュリティグループのルールの最大数を超える可能性を最小限に抑えるために、次のオプションを検討します。
- セキュリティグループクォータあたりのルールの増加をリクエストする。詳細については、Service Quotas ユーザーガイドの「[Requesting a quota increase \(クォータの引き上げのリクエスト\)](#)」を参照してください。
- インスタンスターゲットではなく IP ターゲットを使用する。IP ターゲットを使用すると、同じターゲットポートでルールを共有できます。ロードバランサーのサブネットは、アノテーションを使用して手動で指定できます。詳細については、GitHub の「[アノテーション](#)」を参照してください。
- タイプ LoadBalancer のサービスの代わりにイングレスを使用して、サービスにトラフィックを送信します。AWS Application Load Balancer では、Network Load Balancer よりも必要なルールは少なくなります。複数のイングレス間で ALB を共有できます。詳細については、「[the section called “アプリケーション負荷分散”](#)」を参照してください。複数のサービス間で Network Load Balancer を共有することはできません。
- クラスターを複数のアカウントにデプロイする。



- Pod が Amazon EKS クラスター内の Windows で実行されている場合、ロードバランサーを使用する 1 つのサービスで最大 1,024 個のバックエンド Pod をサポートできます。各ポッドには固有の IP アドレスがあります。
- 新しい Network Load Balancer の作成には、AWS Load Balancer Controller を使用することをおすすめします。AWS クラウドプロバイダーロードバランサーコントローラーで作成された既存の Network Load Balancer を置き換えようとすると、複数の Network Load Balancer でアプリケーションのダウンタイムを引き起こす可能性があります。

## ネットワークロードバランサーを作成する

IP またはインスタンスのターゲットを含むネットワークロードバランサーを作成できます。

### Network Load Balancer の作成 — IP ターゲット

- Amazon EC2 ノード、Fargate、または Amazon EKS Hybrid Nodes にデプロイされた Pod で IP ターゲットを使用できます。Kubernetes サービスはタイプ LoadBalancer として作成する必要があります。詳細については、Kubernetes ドキュメントの「[Garbage Collecting Load Balancers \(ガベージコレクションを行うロードバランサー\)](#)」を参照してください。

IP ターゲットを使用するロードバランサーを作成するには、次のアノテーションをサービスマニフェストに追加し、サービスをデプロイします。aws-load-balancer-type の external 値により、AWS クラウドプロバイダーロードバランサーコントローラーではなく、AWS Load Balancer Controller によって Network Load Balancer が作成されます。アノテーション付きの[サンプリングサービスマニフェスト](#)を表示できます。

```
service.beta.kubernetes.io/aws-load-balancer-type: "external"
service.beta.kubernetes.io/aws-load-balancer-nlb-target-type: "ip"
```

#### Note

IPv6 Pod に負荷分散する場合は、次のアノテーションを追加します。IPv6 を使用して負荷分散を行えるのは、IP ターゲットにのみです。インスタンスターゲットには行えません。このアノテーションを使用しない場合、負荷分散には IPv4 が使用されます。

```
service.beta.kubernetes.io/aws-load-balancer-ip-address-type: dualstack
```

デフォルトでは、Network Load Balancer は `internal aws-load-balancer-scheme` を使用して作成されます。Network Load Balancer は、クラスターの VPC 内の任意のサブネットで起動できます。これには、クラスターの作成時に指定されていないサブネットも含まれます。

Kubernetes は、サブネットがパブリックかプライベートかどうかをルートテーブルで調べます。パブリックサブネットには、インターネットゲートウェイを使ったインターネットへの直接の経路がありますが、プライベートサブネットにはありません。

パブリックサブネットに Network Load Balancer を作成して Amazon EC2 ノードへの負荷分散を行う場合 (Fargate はプライベートのみ可能)、次のアノテーションを付けて `internet-facing` を指定します。

```
service.beta.kubernetes.io/aws-load-balancer-scheme: "internet-facing"
```

#### Note

`service.beta.kubernetes.io/aws-load-balancer-type: "nlb-ip"` アノテーションは、下位互換性のために引き続きサポートされています。ただし、新しいロードバランサーには、`service.beta.kubernetes.io/aws-load-balancer-type: "nlb-ip"` の代わりに以前のアノテーションを使用することをお勧めします。

#### Important

サービスの作成後は、このアノテーションを編集しないでください。変更する必要がある場合は、サービスオブジェクトを削除し、このアノテーションを希望する値にして再度作成します。

## Network Load Balancer の作成 — インスタンスターゲット

- AWS クラウドプロバイダーロードバランサーコントローラーでは、インスタンスターゲットのみを使用して Network Load Balancer を作成します。バージョン 2.2.0 以降の AWS Load Balancer Controller でも同様に、インスタンスターゲットを使用して Network Load Balancer を作成します。新しい Network Load Balancer の作成には、AWS クラウドプロバイダーロードバランサーコントローラーではなく、こちらを使用することをお勧めします。Network Load Balancer のインスタンスターゲットは、Amazon EC2 ノードにデプロイされた Pod では使用できませんが、Fargate

にデプロイされた Pod では使用できません。Fargate にデプロイされたポッド間でネットワークトラフィックを負荷分散するには、IP ターゲットを使用する必要があります。

Network Load Balancer をプライベートサブネットにデプロイするには、サービスの仕様に次のアノテーションが必要です。アノテーション付きの[サンプルサービスマニフェスト](#)を表示できます。aws-load-balancer-type の external 値により、AWS クラウドプロバイダーロードバランサーコントローラーではなく、AWS Load Balancer Controller によって Network Load Balancer が作成されます。

```
service.beta.kubernetes.io/aws-load-balancer-type: "external"
service.beta.kubernetes.io/aws-load-balancer-nlb-target-type: "instance"
```

デフォルトでは、Network Load Balancer は internal aws-load-balancer-scheme を使用して作成されます。内部 Network Load Balancer の場合、Amazon EKS クラスターは、VPC 内で少なくとも 1 つのプライベートサブネットを使用するように設定されている必要があります。Kubernetes は、サブネットがパブリックかプライベートかどうかをルートテーブルで調べます。パブリックサブネットには、インターネットゲートウェイを使ったインターネットへの直接の経路がありますが、プライベートサブネットにはありません。

パブリックサブネットに Network Load Balancer を作成して Amazon EC2 ノードへの負荷分散を行う場合、次のアノテーションを付けて internet-facing を指定します。

```
service.beta.kubernetes.io/aws-load-balancer-scheme: "internet-facing"
```

#### Important

サービスの作成後は、このアノテーションを編集しないでください。変更する必要がある場合は、サービスオブジェクトを削除し、このアノテーションを希望する値にして再度作成します。

## (オプション) サンプルアプリケーションをデプロイする

- クラスター VPC に少なくとも 1 つのパブリックサブネットまたはプライベートサブネットが存在する。

- クラスターに AWS Load Balancer Controller がデプロイされている。詳細については、「[the section called “AWS ロードバランサーコントローラー”](#)」を参照してください。バージョン

2.7.2 以降をお勧めします。

1. Fargate にデプロイする場合は、VPC 内に利用可能なプライベートサブネットがあることを確認し、Fargate プロファイルを作成します。Fargate にデプロイしない場合は、このステップを省略してください。次のコマンドを実行するか、コマンドにある AWS Management Console と name に同じ値を使用して、[namespace](#) でプロファイルを作成できます。*example values* は実際に使用する値に置き換えます。

```
eksctl create fargateprofile \
 --cluster my-cluster \
 --region region-code \
 --name nlb-sample-app \
 --namespace nlb-sample-app
```

2. サンプルアプリケーションをデプロイします。

- a. アプリケーションの名前空間を作成します。

```
kubectl create namespace nlb-sample-app
```

- b. 次の内容をコンピュータ上の `sample-deployment.yaml` という名前のファイルに保存します。

```
apiVersion: apps/v1
kind: Deployment
metadata:
 name: nlb-sample-app
 namespace: nlb-sample-app
spec:
 replicas: 3
 selector:
 matchLabels:
 app: nginx
 template:
 metadata:
 labels:
 app: nginx
 spec:
 containers:
 - name: nginx
 image: public.ecr.aws/nginx/nginx:1.23
```

```
ports:
 - name: tcp
 containerPort: 80
```

- c. マニフェストをクラスターに適用します。

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

3. IP ターゲットに負荷分散を行うインターネット向け Network Load Balancer を使用して、サービスを作成します。

- a. 次の内容をコンピュータ上の `sample-service.yaml` という名前のファイルに保存します。Fargate ノードにデプロイする場合は、`service.beta.kubernetes.io/aws-load-balancer-scheme: internet-facing` 行を削除します。

```
apiVersion: v1
kind: Service
metadata:
 name: nlb-sample-service
 namespace: nlb-sample-app
 annotations:
 service.beta.kubernetes.io/aws-load-balancer-type: external
 service.beta.kubernetes.io/aws-load-balancer-nlb-target-type: ip
 service.beta.kubernetes.io/aws-load-balancer-scheme: internet-facing
spec:
 ports:
 - port: 80
 targetPort: 80
 protocol: TCP
 type: LoadBalancer
 selector:
 app: nginx
```

- b. マニフェストをクラスターに適用します。

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

4. サービスがデプロイされたことを確認します。

```
kubectl get svc nlb-sample-service -n nlb-sample-app
```

出力例は次のとおりです。

| NAME           | TYPE         | CLUSTER-IP     | EXTERNAL-IP                                                                       |                  |
|----------------|--------------|----------------|-----------------------------------------------------------------------------------|------------------|
|                |              |                | PORT(S)                                                                           | AGE              |
| sample-service | LoadBalancer | 10.100.240.137 | k8s-nlbsampl-nlbsampl-xxxxxxxxxx-xxxxxxxxxxxxxxxxxx.elb.region-code.amazonaws.com | 80:32400/TCP 16h |

### Note

**10.100.240.137** および **xxxxxxxxxx-xxxxxxxxxxxxxxxxxx** の値は出力例とは異なります (ロードバランサーに固有のものになります)。**us-west-2** は、クラスターがどの AWS リージョンにあるかによって異なる場合があります。

5. [Amazon EC2 AWS Management Console](#) を開きます。左側のナビゲーションペインで、([Load Balancing] (ロードバランシング) の下の) [Target Groups] (ターゲットグループ) を選択します。[名前] 列で、前のステップの出力である EXTERNAL-IP 列に入っている名前的一部分と [ロードバランサー] 列の値が一致するターゲットグループの名前を選択します。例えば、出力が以前の出力と同じ場合は、k8s-default-samplese-**xxxxxxxxxx** という名前のターゲットグループを選択します。[Target type] (ターゲットタイプ) は IP です。これは、サービスのサンプルマニフェストで指定されているためです。
6. [ターゲットグループ] を選択し、[ターゲット] タブを選択します。[登録済みターゲット] の下に、前のステップでデプロイされた 3 つのレプリカの 3 つの IP アドレスが表示されます。すべてのターゲットのステータスが [正常] になってから次に進みます。すべてのターゲットが healthy になるまで数分かかることがあります。ターゲットは、healthy 状態に変更される前の unhealthy 状態である可能性があります。
7. **xxxxxxxxxx-xxxxxxxxxxxxxxxxxx** および **us-west-2** を EXTERNAL-IP の [前のステップ](#) の出力で返された値に置き換え、サービスにトラフィックを送信します。プライベートサブネットにデプロイした場合は、踏み台ホストなどの VPC 内のデバイスからページを表示する必要があります。詳細については、[AWS での Linux 踏み台ホスト](#) を参照してください。

```
curl k8s-default-samplese-xxxxxxxxxx-xxxxxxxxxxxxxxxxxx.elb.region-code.amazonaws.com
```

出力例は次のとおりです。

```
<!DOCTYPE html>
<html>
<head>
<title>Welcome to nginx!</title>
```

```
[...]
```

8. サンプルのデプロイ、サービス、および名前空間の使用が終了したら、それらを削除します。

```
kubectl delete namespace nlb-sample-app
```

## Application Load Balancer を使用してアプリケーションと HTTP トラフィックをルーティングする

### Note

新規: Amazon EKS Auto Mode は、ロードバランシングの定型作業を自動化します。詳細については、以下を参照してください。

- [the section called “ロードバランサーをデプロイする”](#)
- [the section called “ingress class の作成”](#)

Kubernetes ingress を作成するとき、アプリケーショントラフィックの負荷分散を行う AWS Application Load Balancer (ALB) がプロビジョニングされます。詳細については、「Application Load Balancer ユーザーガイド」の「[Application Load Balancer とは？](#)」および Kubernetes ドキュメントの「[Ingress](#)」を参照してください。ALB は、ノードまたは AWS Fargate にデプロイされたポッドで使用できます。ALB をパブリックサブネットまたはプライベートサブネットにデプロイできます。

アプリケーショントラフィックは、OSI モデルの L7 で負荷分散されます。L4 でネットワークトラフィックの負荷を分散するには、LoadBalancer タイプの Kubernetes service をデプロイします。このタイプは、AWS Network Load Balancer をプロビジョニングします。詳細については、「[the section called “ネットワーク負荷分散”](#)」を参照してください。2 種類の負荷分散の違いについては、AWS ウェブサイトの「[Elastic Load Balancing の特徴](#)」を参照してください。

## 前提条件

あるアプリケーションに対してアプリケーショントラフィックの負荷分散を行うには、次の要件を満たす必要があります。

- 既存のクラスターがある。既存のクラスターがない場合は、「[はじめに](#)」を参照してください。既存のクラスターのバージョンを更新する必要がある場合は、[the section called “Kubernetes バージョンを更新する”](#) を参照して下さい。
- クラスターに AWS Load Balancer Controller がデプロイされている。詳細については、「[the section called “AWS ロードバランサーコントローラー”](#)」を参照してください。バージョン 2.7.2 以降をお勧めします。
- 少なくとも 2 つのサブネットが異なるアベイラビリティーゾーンに存在している。AWS Load Balancer Controller は、各アベイラビリティーゾーンから 1 つずつサブネットを選択します。タグ付けされたサブネットがアベイラビリティーゾーンで複数見つかった場合、コントローラーは、サブネット ID が辞書式順序で最初に来るサブネットを選択します。各サブネットには最低 8 個の利用可能な IP アドレスが必要です。

ワーカーノードにアタッチされた複数のセキュリティグループを使用している場合は、次のように 1 つのセキュリティグループにタグを付ける必要があります。##### の部分は自分のクラスター名に置き換えます。

- キー – `kubernetes.io/cluster/<my-cluster>`
- 値 – `shared` または `owned`
- AWS Load Balancer Controller バージョン 2.1.1 以前を使用している場合、サブネットに次のようにタグ付けする必要があります。バージョン 2.1.2 以降を使用している場合、このタグ付けはオプションです。ただし、次のいずれかの場合は、サブネットにタグ付けることをお勧めします。同じ VPC で実行されている複数のクラスターがあるか、VPC でサブネットを共有する複数の AWS サービスがあります。または、各クラスターに対してロードバランサーをプロビジョニングする場所を詳細に制御する必要があります。##### の部分は自分のクラスター名に置き換えます。
- キー – `kubernetes.io/cluster/<my-cluster>`
- 値 – `shared` または `owned`
- パブリックサブネットとプライベートサブネットは次の要件を満たしている必要があります。サービスオブジェクトまたは Ingress オブジェクトのアノテーションとしてサブネット ID を明示的に指定しない限り、これは例外です。サービスオブジェクトまたは Ingress オブジェクトのアノテーションとしてサブネット ID を明示的に指定して、ロードバランサーをプロビジョニングすることを想定しています。このような状況では、Kubernetes と AWS Load Balancer Controller は、これらのサブネットを直接使用してロードバランサーを作成します。次のタグは必要ありません。
- プライベートサブネット — 次の形式でタグ付けする必要があります。これは、Kubernetes と AWS ロードバランサーコントローラーが、サブネットを内部ロードバランサーに使用できることを認識できるようにするためです。eksctl または Amazon EKS AWS CloudFormation テンプレートを使用して、2020 年 3 月 26 日以降に VPC を作成する場合、サブネットは、作成時に



適切にタグ付けされます。Amazon EKS AWS CloudFormation VPC テンプレートの詳細については、「[the section called “「VPC を作成する」”](#)」を参照してください。

- キー – `kubernetes.io/role/internal-elb`
- 値 – 1
- パブリックサブネット — 次の形式でタグ付けする必要があります。これは、Kubernetes は、外部ロードバランサー用に指定されたサブネットのみを使用できることを認識できるようにするためです。この方法では、Kubernetes は各アベイラビリティゾーンでパブリックサブネットを (サブネット ID に基づいて辞書順に) 選択しません `eksctl` または Amazon EKS AWS CloudFormation テンプレートを使用して、2020 年 3 月 26 日以降に VPC を作成する場合、サブネットは、作成時に適切にタグ付けされます。Amazon EKS AWS CloudFormation VPC テンプレートの詳細については、「[the section called “「VPC を作成する」”](#)」を参照してください。
- キー – `kubernetes.io/role/elb`
- 値 – 1

サブネットロールタグが明示的に追加されていない場合、Kubernetes サービスコントローラーはクラスター VPC サブネットのルートテーブルを調べます。これは、サブネットがプライベートかパブリックかを判断するためです。この動作に頼らないことをお勧めします。代わりに、プライベートまたはパブリックロールタグを明示的に追加します。AWS Load Balancer Controller は、ルートテーブルを検査しません。また、自動検出を正常に実行するには、プライベートタグとパブリックタグが必要です。

- Kubernetes のインGRESSリソースが `kubernetes.io/ingress.class: alb` アノテーションを使用してクラスターで作成されるたびに、[AWS Load Balancer Controller](#) により ALB およびサポートされる必要な AWS リソースが作成されます。インGRESSリソースは、HTTP または HTTPS トラフィックをクラスター内の別の Pod にルーティングするように ALB を設定します。インGRESSオブジェクトで AWS Load Balancer Controller を使用できるようにするには、Kubernetes インGRESS仕様に次のアノテーションを追加します。詳細については、GitHub の「[Ingress specification \(Ingress 仕様\)](#)」を参照してください。

```
annotations:
 kubernetes.io/ingress.class: alb
```

#### Note

IPv6 Pod への負荷分散を行う場合は、インGRESS仕様に次のアノテーションを追加します。IPv6 を使用して負荷分散を行えるのは、IP ターゲットにのみです。インスタンス

ターゲットには行えません。このアノテーションを使用しない場合、負荷分散には IPv4 が使用されます。

```
alb.ingress.kubernetes.io/ip-address-type: dualstack
```

- AWS Load Balancer Controller は、次のトラフィックモードをサポートしています。
- インスタンスクラスター内のノードを ALB のターゲットとして登録します。ALB に到達するトラフィックは、サービスの NodePort にルーティングされてから、Pod にプロキシされます。これがデフォルトのトラフィックモードです。alb.ingress.kubernetes.io/target-type: instance の注釈を使用して明示的に指定することもできます。

#### Note

Kubernetes サービスでは、このトラフィックモードを使用する NodePort または LoadBalancer タイプを指定する必要があります。

- IP - ポッドを ALB のターゲットとして登録します。ALB に到達するトラフィックは、サービスの Pod に直接ルーティングされます。このトラフィックモードを使用するには、alb.ingress.kubernetes.io/target-type: ip の注釈を指定する必要があります。IP ターゲットタイプは、ターゲット Pod が Fargate または Amazon EKS Hybrid Nodes で実行されている場合には必須です。
- コントローラーによって作成された ALB にタグを付けるには、次のアノテーションをコントローラーに追加します: alb.ingress.kubernetes.io/tags。AWS Load Balancer Controller でサポートされるすべての使用可能なアノテーションのリストについては、GitHub の「[Ingress アノテーション](#)」を参照してください。
- ALB コントローラーのバージョンをアップグレードまたはダウングレードすると、ALB コントローラーのバージョンに依存する機能が大きく変更される可能性があります。各リリースで導入された新しい変更の詳細については、GitHub で [ALB Controller リリースノート](#) を参照してください。

## イングレスグループで ALB を再利用する

IngressGroups を使用して複数のサービスのリソース間で Application Load Balancer を共有できます。

インGRESSをグループに入れるには、Kubernetes のインGRESSリソース仕様に次のアノテーションを追加します。

```
alb.ingress.kubernetes.io/group.name: my-group
```

グループ名は、次のようにする必要があります。

- 長さが 63 文字以下。
- 名前は、小文字の英文字、-、および . で構成されます。
- 数字または文字で始めます。

コントローラーは、同じインGRESSグループ内のすべてのインGRESSのインGRESSルールを自動的にマージします。これは、単一のALBでそれらをサポートしています。インGRESSで定義されたほとんどのアノテーションは、そのインGRESSで定義されたパスにのみ適用されます。デフォルトでは、インGRESSリソースはどのインGRESSグループにも属していません。

#### Warning

##### 潜在的なセキュリティリスク

インGRESSリソースを作成または変更する RBAC 権限を持つすべての Kubernetes ユーザーが同じ信頼境界内にいる場合にのみ、インGRESSにインGRESSグループを指定します。アノテーションをグループ名で追加すると、他の Kubernetes ユーザーが、同じインGRESSグループに属するインGRESSを作成または変更する可能性があります。これにより、優先度の高いルールで既存のルールを上書きするなど、望ましくない動作が発生する可能性があります。

インGRESSリソースの順序番号を追加できます。

```
alb.ingress.kubernetes.io/group.order: '10'
```

番号は 1 ~ 1000 の範囲で指定できます。同じインGRESSグループ内のすべてのインGRESSについて、最小の番号を持つものが最初に評価されます。このアノテーションがないインGRESSはすべて、値 0 で評価されます。番号が小さいルールと番号が大きいルールが重複すると、番号が小さいルールが上書きされる可能性があります。デフォルトでは、同じインGRESSグループ内のインGRESS間のルールの順序は、名前空間と名前に基づき辞書式順序で決定されます。

**⚠ Important**

同じインGRESグループの各インGRESに、一意のプライオリティ番号があることを確認します。インGRES間で重複する順序番号を持つことはできません。

## (オプション) サンプルアプリケーションをデプロイする

- クラスター VPC に少なくとも 1 つのパブリックサブネットまたはプライベートサブネットが存在する。
- クラスターに AWS Load Balancer Controller がデプロイされている。詳細については、「[the section called “AWS ロードバランサーコントローラー”](#)」を参照してください。バージョン 2.7.2 以降をお勧めします。

Amazon EC2 ノード、Fargate ポッド、またはその両方を持つクラスターで、サンプルアプリケーションを実行できます。

1. Fargate にデプロイしない場合は、このステップを省略してください Fargate にデプロイする場合は、Fargate プロファイルを作成します。次のコマンドを実行するか、コマンドにある AWS Management Console と name に同じ値を使用して、[namespace](#) でプロファイルを作成できます。*example values* は実際に使用する値に置き換えます。

```
eksctl create fargateprofile \
 --cluster my-cluster \
 --region region-code \
 --name alb-sample-app \
 --namespace game-2048
```

2. サンプルアプリケーションとしてゲーム [2048](#) をデプロイし、AWS Load Balancer Controller がインGRESオブジェクトの結果として AWS ALB を作成することを確認します。デプロイ先のサブネットのタイプに関する手順を実行します。
  - a. IPv6 ファミリーを使用して作成したクラスター内の Pod にデプロイしている場合には、次のステップに進みます。
    - Public::

```
kubectl apply -f https://raw.githubusercontent.com/kubernetes-sigs/aws-load-balancer-controller/v2.11.0/docs/examples/2048/2048_full.yaml
```

- プライベート::

- A. マニフェストをダウンロードします。

```
curl -O https://raw.githubusercontent.com/kubernetes-sigs/aws-load-balancer-controller/v2.11.0/docs/examples/2048/2048_full.yaml
```

- B. ファイルを編集して、`alb.ingress.kubernetes.io/scheme: internet-facing` と記述された行を探します。

- C. ##### を `internal` に変更し、ファイルを保存します。

- D. マニフェストをクラスターに適用します。

```
kubectl apply -f 2048_full.yaml
```

- b. [IPv6 ファミリー](#) を使用して作成したクラスター内の Pod にデプロイする場合、次の手順を完了します。

- i. マニフェストをダウンロードします。

```
curl -O https://raw.githubusercontent.com/kubernetes-sigs/aws-load-balancer-controller/v2.11.0/docs/examples/2048/2048_full.yaml
```

- ii. エディタでファイルを開き、イングレス仕様のアノテーションに次の行を追加します。

```
alb.ingress.kubernetes.io/ip-address-type: dualstack
```

- iii. インターネット向けの Pod ではなく、内部の Pod への負荷分散を行う場合は、`alb.ingress.kubernetes.io/scheme: internet-facing` という行を `alb.ingress.kubernetes.io/scheme: internal` に変更します。

- iv. ファイルを保存します。

- v. マニフェストをクラスターに適用します。

```
kubectl apply -f 2048_full.yaml
```

- 3. 数分後、次のコマンドを使用して、イングレスリソースが作成されたことを確認します。

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

出力例は次のとおりです。

| NAME                   | CLASS  | HOSTS | ADDRESS                                              |
|------------------------|--------|-------|------------------------------------------------------|
|                        |        | PORTS | AGE                                                  |
| ingress-2048           | <none> | *     | k8s-game2048-ingress2-xxxxxxxxxx-yyyyyyyyyyy.region- |
| code.elb.amazonaws.com |        | 80    | 2m32s                                                |

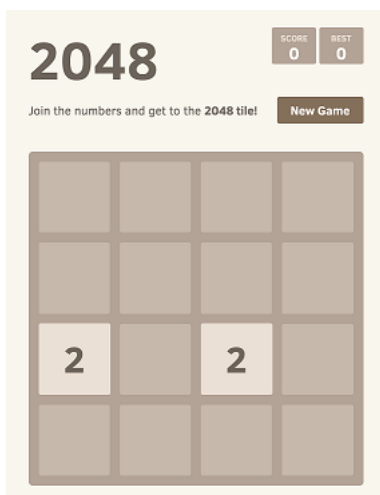
### Note

プライベートサブネットでロードバランサーを作成した場合、前の出力の ADDRESS の下の値の前に `internal-` が付けられます。

イングレスが数分後に作成されていない場合は、次のコマンドを実行して AWS Load Balancer Controller のログを表示します。これらのログには、デプロイに関する問題の診断に使用できるエラーメッセージが含まれている場合があります。

```
kubectl logs -f -n kube-system -l app.kubernetes.io/instance=aws-load-balancer-controller
```

- パブリックサブネットにデプロイした場合は、ブラウザを開き、前のコマンドの出力から ADDRESS URL に移動してサンプルアプリケーションを表示します。何も表示されない場合は、ブラウザを更新してからもう一度試してください。プライベートサブネットにデプロイした場合は、踏み台ホストなどの VPC 内のデバイスからページを表示する必要があります。詳細については、[AWS での Linux 踏み台ホスト](#)を参照してください。



- サンプルアプリケーションの試用が終了したら、次のコマンドのいずれかを実行して削除します。

- ダウンロードしたコピーを適用するのではなくマニフェストを適用した場合は、次のコマンドを使用します。

```
kubectl delete -f https://raw.githubusercontent.com/kubernetes-sigs/aws-load-balancer-controller/v2.11.0/docs/examples/2048/2048_full.yaml
```

- マニフェストをダウンロードして編集した場合は、次のコマンドを使用します。

```
kubectl delete -f 2048_full.yaml
```

## サービスに割り当てることができる外部IPアドレスを制限する

Kubernetes サービスには、クラスターの内部から次を経由することで到達できます。

- Kubernetes によって自動的に割り当てられるクラスター IP アドレス
- サービス仕様で `externalIPs` プロパティに指定した任意の IP アドレス 外部 IP アドレスは Kubernetes によって管理されるのではなく、クラスターの管理者が管理します。externalIPs で指定される外部 IP アドレスは、クラウドプロバイダーによって提供される LoadBalancer タイプのサービスに割り当てられる外部 IP アドレスとは異なります。

Kubernetes サービスの詳細については、Kubernetes ドキュメントの「[サービス](#)」を参照してください。サービス仕様で externalIPs に指定できる IP アドレスを制限できます。

- webhook の証明書を管理するには cert-manager をデプロイします。詳細については「[cert-manager](#)」(証明書マネージャー) ドキュメントを参照してください。

```
kubectl apply -f https://github.com/jetstack/cert-manager/releases/download/v1.5.4/cert-manager.yaml
```

- cert-manager Pod が実行中であることを確認します。

```
kubectl get pods -n cert-manager
```

出力例は次のとおりです。

| NAME                                     | READY | STATUS  | RESTARTS | AGE |
|------------------------------------------|-------|---------|----------|-----|
| cert-manager-58c8844bb8-nlx7q            | 1/1   | Running | 0        | 15s |
| cert-manager-cainjector-745768f6ff-696h5 | 1/1   | Running | 0        | 15s |

```
cert-manager-webhook-67cc76975b-4v4nk 1/1 Running 0 14s
```

3. 既存のサービスを確認して、アドレスを制限する CIDR ブロックに含まれない外部 IP アドレスが割り当てられていないことを確認します。

```
kubectl get services -A
```

出力例は次のとおりです。

| NAMESPACE                    | EXTERNAL-IP | NAME                                  | PORT(S)        | AGE  | TYPE      |
|------------------------------|-------------|---------------------------------------|----------------|------|-----------|
| cert-manager                 |             | cert-manager                          | 9402/TCP       | 20m  | ClusterIP |
| cert-manager                 |             | cert-manager-webhook                  | 443/TCP        | 20m  | ClusterIP |
| default                      |             | kubernetes                            | 443/TCP        | 2d1h | ClusterIP |
| externalip-validation-system |             | externalip-validation-webhook-service | 443/TCP        | 16s  | ClusterIP |
| kube-system                  |             | kube-dns                              | 53/UDP, 53/TCP | 2d1h | ClusterIP |
| my-namespace                 |             | my-service                            | 80/TCP         | 149m | ClusterIP |

いずれかの値が、アクセスを制限するブロック内にない IP アドレスである場合はブロック内に存在するアドレスを変更し、サービスを再びデプロイする必要があります。例えば、前の出力の my-service サービスにはステップ 5 の CIDR ブロックの例に含まれていない外部 IP アドレスが割り当てられています。

4. 外部 IP のウェブフックマニフェストをダウンロードします。また、GitHub で[ウェブフックのソースコード](#)を確認できます。

```
curl -O https://s3.us-west-2.amazonaws.com/amazon-eks/docs/externalip-webhook.yaml
```

5. CIDR ブロックを指定します。ダウンロードしたファイルをエディタで開き、次の行の最初の \# を削除します。

```
#args:
#- --allowed-external-ip-cidrs=10.0.0.0/8
```



10.0.0.0/8 を独自の CIDR ブロックに置き換えます。ブロックはいくつでも指定できます。複数のブロックを指定する場合はブロック間にコンマを追加します。

6. クラスターが us-west-2 AWS に存在しない場合、ファイル内の us-west-2、602401143452、および amazonaws.com を次のコマンドに置き換えます。コマンドを実行する前に、#### および 111122223333 を「[Amazon EKS アドオンの Amazon コンテナイメージレジストリを表示する](#)」のリストから AWS リージョンの値に置き換えます。

```
sed -i.bak -e 's|602401143452|111122223333|' externalip-webhook.yaml
sed -i.bak -e 's|us-west-2|region-code|' externalip-webhook.yaml
sed -i.bak -e 's|amazonaws.com||' externalip-webhook.yaml
```

7. マニフェストをクラスターに適用します。

```
kubectl apply -f externalip-webhook.yaml
```

「CIDR ブロックを指定します」のステップで指定したブロックに含まれていない、externalIPs に指定された IP アドレスを使用して、サービスをクラスターにデプロイしようとすると、失敗します。

## あるリポジトリから別のリポジトリにコンテナイメージをコピーする

このトピックでは、ノードがアクセスできないリポジトリからコンテナイメージをプルし、ノードがアクセスできるリポジトリにイメージをプッシュする方法について説明します。イメージを、Amazon ECR またはノードがアクセスできる代替リポジトリにプッシュできます。

- コンピュータにインストールされ設定されている Docker エンジン。手順については、Docker ドキュメントの「[Docker Engine のインストール](#)」を参照してください。
- ご使用のデバイスまたは AWS クラウドシェル で、バージョン 2.12.3 以降、または AWS コマンドラインインターフェイス (AWS CLI のバージョン 1.27.160 以降がインストールおよび設定されていること。現在のバージョンを確認するには「aws --version | cut -d / -f2 | cut -d ' ' -f1」を参照してください。yum、apt-get、macOS 用の Homebrew などのパッケージマネージャーは、多くの場合 AWS CLI の最新バージョンより数バージョン古くなっています。最新バージョンをインストールするには「AWS コマンドラインインターフェイスユーザーガイド」の「[インストール](#)」および「[aws configure を使用したクイック設定](#)」を参照してください。AWS クラウドシェル にインストールされている AWS CLI バージョンも最新バージョンよ

り数バージョン遅れることがあります。更新するには、「AWS CloudShell ユーザーガイド」の「[ホームディレクトリへの AWS CLI のインストール](#)」を参照してください。

- ノードで Amazon のネットワークを介してプライベート Amazon ECR リポジトリからコンテナイメージのプルを行ったり、コンテナイメージをプッシュする場合は、Amazon ECR のインターフェイス VPC エンドポイント。詳細については、Amazon Elastic コンテナレジストリ ユーザーガイドの「[Amazon ECR 用の VPC エンドポイントを作成する](#)」を参照してください。

リポジトリからコンテナイメージをプルし、独自のリポジトリにプッシュする場合は、次の手順を完了してください。このトピックで紹介する次の例では、[Amazon VPC CNI Plugin for Kubernetes メトリクスヘルパー](#)のイメージがプルされます。これらのステップを実行する場合は、必ず *example values* を置き換えてください。

1. Amazon ECR リポジトリまたは別のリポジトリがまだ作成されていない場合は、ノードがアクセスできるリポジトリを作成します。次のコマンドは、Amazon ECR プライベートリポジトリを作成します。Amazon ECR プライベートリポジトリ名は文字で始まる必要があります。小文字、数字、ハイフン (-)、アンダースコア (\_)、フォワードスラッシュ (/) のみ含むことができます。詳細については、Amazon Elastic Container Registry ユーザーガイドの「[プライベートリポジトリの作成](#)」を参照してください。

*cni-metrics-helper* を選択したものに置き換えることができます。ベストプラクティスとして、各イメージに個別のリポジトリを作成します。イメージタグはリポジトリ内で一意である必要があるため、この方法をお勧めします。*region-code* を [Amazon ECR でサポートされている AWS リージョン](#) に置き換えます。

```
aws ecr create-repository --region region-code --repository-name cni-metrics-helper
```

2. ノードがプルする必要があるイメージのレジストリ、リポジトリ、およびタグ (オプション) を決めます。この情報は `registry/repository[:tag]` 形式です。

Amazon EKS のイメージのインストールに関するトピックの多くは、マニフェストファイルを適用するか、Helm チャートを使用してイメージをインストールする必要があります。ただし、マニフェストファイルの適用や、Helm チャートをインストールする前に、まずマニフェストやチャートの `values.yaml` ファイルの内容を確認してください。そうすることで、プルするレジストリ、リポジトリ、およびタグを決定することができます。

例えば、[Amazon VPC CNI plugin for Kubernetes メトリクスヘルパー](#)の [マニフェストファイル](#)に次の行があります。602401143452.dkr.ecr.us-west-2.amazonaws.com がレジストリで、

これは Amazon ECR のプライベートレジストリです。cni-metrics-helper が、レポジトリです。

```
image: "602401143452.dkr.ecr.us-west-2.amazonaws.com/cni-metrics-helper:v1.12.6"
```

イメージの場所として、次のバリエーションがあります:

- repository-name:tag のみ。この場合、通常 docker.io はレジストリですが、レジストリが指定されていない場合は、Kubernetes がデフォルトでリポジトリ名の前にレジストリをプリペンドするため、指定されていません。
- repository-name/repository-namespace/repository:tag. リポジトリの名前空間は任意ですが、イメージを分類するためにリポジトリ所有者が指定される場合があります。例えば、[Amazon ECR Public Gallery にあるすべての Amazon EC2 イメージ](#)は、aws-ec2 名前空間を使用します。

Helm を使用してイメージをインストールする前に、Helm values.yaml ファイルを表示してイメージの場所を決定します。例えば、[Amazon VPC CNI plugin for Kubernetes メトリクスヘルパー](#)の [values.yaml](#) ファイルに次の行があります。

```
image:
 region: us-west-2
 tag: v1.12.6
 account: "602401143452"
 domain: "amazonaws.com"
```

### 3. マニフェストファイルで指定されたコンテナイメージをプルします。

- [Amazon ECR Public Gallery](#) など、パブリックレジストリからプルする場合は、認証が不要なため、次のサブステップにスキップできます。この例では、CNI メトリクスヘルパーイメージのリポジトリを含む Amazon ECR プライベートレジストリに対して認証します。Amazon EKS は、「[Amazon EKS アドオンの Amazon コンテナイメージレジストリを表示する](#)」に記載されている各レジストリにイメージを保持します。**602401143452** と **region-code** を別のレジストリの情報に置き換えることで、任意のレジストリに対して認証が可能です。[Amazon EKS がサポートされている AWS リージョン](#) には、それぞれに個別のレジストリが存在します。

```
aws ecr get-login-password --region region-code | docker login --username AWS --password-stdin 602401143452.dkr.ecr.region-code.amazonaws.com
```

- イメージをプルする。この例では、前のサブステップで認証したレジストリからプルします。**602401143452** と **region-code** を前のサブステップで指定した情報に置き換えます。

```
docker pull 602401143452.dkr.ecr.region-code.amazonaws.com/cni-metrics-helper:v1.12.6
```

4. プルしたイメージに、レジストリ、リポジトリ、タグでタグ付けします。次の例では、マニフェストファイルからイメージをプルし、最初のステップで作成した Amazon ECR プライベートリポジトリにイメージをプッシュすることを前提としています。**111122223333** は、ご自分のアカウント ID に置き換えます。**region-code** を、Amazon ECR プライベートリポジトリを作成した AWS リージョンに置き換えます。

```
docker tag cni-metrics-helper:v1.12.6 111122223333.dkr.ecr.region-code.amazonaws.com/cni-metrics-helper:v1.12.6
```

5. レジストリに対して認証します。この例では、最初のステップで作成した Amazon ECR プライベートレジストリに対して認証します。詳細については、Amazon Elastic Container Registry ユーザーガイドの「[レジストリの認証](#)」を参照してください。

```
aws ecr get-login-password --region region-code | docker login --username AWS --password-stdin 111122223333.dkr.ecr.region-code.amazonaws.com
```

6. リポジトリにプッシュするイメージを付けます。この例では、最初のステップで作成した Amazon ECR プライベートリポジトリにイメージをプッシュします。詳細については、Amazon Elastic Container Registry ユーザーガイドの「[Docker イメージをプッシュする](#)」を参照してください。

```
docker push 111122223333.dkr.ecr.region-code.amazonaws.com/cni-metrics-helper:v1.12.6
```

7. プッシュしたイメージ用の `registry/repository:tag` を使用して前のステップでイメージの決定に使用したマニフェストファイルを更新します。Helm チャートを使用してインストールする場合、`registry/repository:tag` を指定するオプションが多くあります。チャートをインストールする場合、リポジトリにプッシュしたイメージの `registry/repository:tag` を指定します。

## Amazon EKS アドオンの Amazon コンテナイメージレジストリを表示する

[AWS Amazon EKS アドオン](#) をクラスターにデプロイすると、ノードはアドオンのインストールメカニズムで指定されたレジストリ (インストールマニフェストや Helm values.yaml ファイルな

ど から必要なコンテナイメージを取得します。画像はAmazon EKS Amazon ECR プライベートリポジトリからプルします。Amazon EKS は Amazon EKSがサポートする各 AWS リージョンのリポジトリにイメージをレプリケートします。ノードは次のレジストリのいずれかからインターネット経由でコンテナイメージをプルできます。またはVPC 内に [Amazon ECR \(AWS プライベートリンク用のインターフェイス VPC エンドポイント\)](#) を作成した場合、ノードは Amazon のネットワーク経由でイメージをプルできます。レジストリにはAWS IAM アカウントでの認証が必要です。ノードは [Amazon EKS ノードの IAM ロール](#) を使用して認証します。このロールには関連付けられた [AmazonEC2ContainerRegistryReadOnly](#) で管理される IAM ポリシーのアクセス許可があります。

| AWS リージョン      | レジストリ                                             |
|----------------|---------------------------------------------------|
| af-south-1     | 877085696533.dkr.ecr.af-south-1.amazonaws.com     |
| ap-east-1      | 800184023465.dkr.ecr.ap-east-1.amazonaws.com      |
| ap-east-2      | 533267051163.dkr.ecr.ap-east-1.amazonaws.com      |
| ap-southeast-3 | 296578399912.dkr.ecr.ap-southeast-3.amazonaws.com |
| ap-south-2     | 900889452093.dkr.ecr.ap-south-2.amazonaws.com     |
| ap-southeast-4 | 491585149902.dkr.ecr.ap-southeast-4.amazonaws.com |
| ap-south-1     | 602401143452.dkr.ecr.ap-south-1.amazonaws.com     |
| ap-northeast-3 | 602401143452.dkr.ecr.ap-northeast-3.amazonaws.com |
| ap-northeast-2 | 602401143452.dkr.ecr.ap-northeast-2.amazonaws.com |

| AWS リージョン      | レジストリ                                                |
|----------------|------------------------------------------------------|
| ap-southeast-1 | 602401143452.dkr.ecr.ap-southeast-1.amazonaws.com    |
| ap-southeast-2 | 602401143452.dkr.ecr.ap-southeast-2.amazonaws.com    |
| ap-southeast-7 | 121268973566.dkr.ecr.ap-southeast-7.amazonaws.com    |
| ap-northeast-1 | 602401143452.dkr.ecr.ap-northeast-1.amazonaws.com    |
| cn-north-1     | 918309763551.dkr.ecr.cn-north-1.amazonaws.com.cn     |
| cn-northwest-1 | 961992271922.dkr.ecr.cn-northwest-1.amazonaws.com.cn |
| eu-central-1   | 602401143452.dkr.ecr.eu-central-1.amazonaws.com      |
| eu-west-1      | 602401143452.dkr.ecr.eu-west-1.amazonaws.com         |
| eu-west-2      | 602401143452.dkr.ecr.eu-west-2.amazonaws.com         |
| eu-south-1     | 590381155156.dkr.ecr.eu-south-1.amazonaws.com        |
| eu-west-3      | 602401143452.dkr.ecr.eu-west-3.amazonaws.com         |
| eu-south-2     | 455263428931.dkr.ecr.eu-south-2.amazonaws.com        |
| eu-north-1     | 602401143452.dkr.ecr.eu-north-1.amazonaws.com        |

| AWS リージョン     | レジストリ                                            |
|---------------|--------------------------------------------------|
| eu-central-2  | 900612956339.dkr.ecr.eu-central-2.amazonaws.com  |
| il-central-1  | 066635153087.dkr.ecr.il-central-1.amazonaws.com  |
| mx-central-1  | 730335286997.dkr.ecr.mx-central-1.amazonaws.com  |
| me-south-1    | 558608220178.dkr.ecr.me-south-1.amazonaws.com    |
| me-central-1  | 759879836304.dkr.ecr.me-central-1.amazonaws.com  |
| us-east-1     | 602401143452.dkr.ecr.us-east-1.amazonaws.com     |
| us-east-2     | 602401143452.dkr.ecr.us-east-2.amazonaws.com     |
| us-west-1     | 602401143452.dkr.ecr.us-west-1.amazonaws.com     |
| us-west-2     | 602401143452.dkr.ecr.us-west-2.amazonaws.com     |
| ca-central-1  | 602401143452.dkr.ecr.ca-central-1.amazonaws.com  |
| ca-west-1     | 761377655185.dkr.ecr.ca-west-1.amazonaws.com     |
| sa-east-1     | 602401143452.dkr.ecr.sa-east-1.amazonaws.com     |
| us-gov-east-1 | 151742754352.dkr.ecr.us-gov-east-1.amazonaws.com |



| AWS リージョン     | レジストリ                                            |
|---------------|--------------------------------------------------|
| us-gov-west-1 | 013241004608.dkr.ecr.us-gov-west-1.amazonaws.com |

## Amazon EKS アドオン

アドオンは、Kubernetes アプリケーションをサポートする操作機能を提供するソフトウェアですが、アプリケーションに固有のものではありません。これには、監視エージェントや Kubernetes ドライバなどのソフトウェアが含まれます。これらのソフトウェアにより、クラスターは、ネットワーク、コンピューティング、およびストレージ用の基盤となる AWS リソースと相互作用できるようになります。アドオンソフトウェアは、通常、Kubernetes コミュニティや AWS のようなクラウドプロバイダ、またはサードパーティーベンダーによって構築、保守されます。Amazon EKS は、Amazon VPC CNI plugin for Kubernetes、kube-proxy、および、CoreDNS などのセルフマネージド型アドオンを、すべてのクラスターで自動的にインストールします。VPC CNI アドオンは Amazon EKS Hybrid Nodes と互換性がなく、ハイブリッドノードにデプロイされないことに注意してください。必要に応じて、アドオンのデフォルト設定は変更と更新が可能です。

Amazon EKS アドオンは Amazon EKS クラスター用に厳選されたアドオンセットのインストールと管理を実行してください。Amazon EKS アドオンには最新のセキュリティパッチ、バグ修正が含まれており、また Amazon EKS と連携するために AWS により検証されています。Amazon EKS アドオンを使用すると、Amazon EKS クラスターの安全性と安定性を一貫して保証でき、アドオンのインストール、設定、および更新に必要な作業量を削減できます。セルフマネージド型のアドオン (kube-proxy など) がすでにクラスターで実行されており、Amazon EKS アドオンとして利用可能な場合、kube-proxy Amazon EKS アドオンをインストールすれば、その機能からのメリットを活用できるようになります。

Amazon EKS API を使用することで、Amazon EKS アドオンのための、Amazon EKS 管理の特定の設定フィールドを更新できます。Amazon EKS で管理されていない設定フィールドは、アドオンの起動後に Kubernetes クラスター内で直接変更することもできます。これには、アドオン向けの特定の設定フィールドの定義が含まれます (該当する場合)。これらの変更は一度実行されると Amazon EKS によって上書きされることはありません。これは、Kubernetes によるサーバー側の適用機能を使用して実現しています。詳細については、「[the section called “カスタマイズ可能なフィールド”](#)」を参照してください。

Amazon EKS アドオンはどの Amazon EKS ノードタイプでも使用できます。詳細については「[コンピューティングを管理する](#)」を参照してください。



Amazon EKS アドオンはAmazon EKS API、AWS Management Console、AWS CLI、および `eksctl` を使用して、追加、更新、または削除ができます。Amazon EKS アドオンは[AWS クラウド フォーメーション](#) を使用しても作成できます。

## 考慮事項

Amazon EKS アドオンを使用する場合は次の点を考慮してください:

- クラスターでアドオンの設定を行うにはアドオンを操作するための IAM 許可を、[IAM プリンシパル](#)が持っている必要があります。詳細については「[Amazon Elastic Kubernetes Service で定義されるアクション](#)」で、Addon の名前ごとにアクションを参照してください。
- Amazon EKS アドオンはクラスター用にプロビジョニングまたは設定するノードで実行されます。ノードタイプには Amazon EC2 インスタンス、Fargate、ハイブリッドノードがあります。
- Amazon EKS で管理されていないフィールドを変更して、Amazon EKS アドオンのインストールをカスタマイズできます。詳細については、「[the section called “カスタマイズ可能なフィールド”](#)」を参照してください。
- AWS Management Console を使用してクラスターを作成すると、Amazon EKS kube-proxy、Amazon VPC CNI plugin for Kubernetes、および CoreDNS Amazon EKS アドオンが自動的にクラスターに追加されます。また、`eksctl` を使用して config ファイルでクラスターを作成する場合は`eksctl` は Amazon EKS アドオンを持つクラスターを作成できます。config ファイルなしで (あるいは他のツールを使用して) `eksctl` によりクラスターを作成する場合は、Amazon EKS アドオンではなく、セルフマネージドの kube-proxy や、Amazon VPC CNI plugin for Kubernetes、および CoreDNS アドオンがインストールされます。これらは自分で管理することが可能で、また、クラスターの作成後に Amazon EKS アドオンを手動で追加することもできます。クラスターの作成にどの方法を使用するかにかかわらず、VPC CNI アドオンはハイブリッドノードにインストールされません。
- `eks:addon-cluster-admin ClusterRoleBinding` は `cluster-admin ClusterRole` を `eks:addon-manager Kubernetes` の ID にバインドします。このロールには `eks:addon-manager ID` が Kubernetes 名前空間を作成し、アドオンを名前空間にインストールするために必要なアクセス許可があります。`eks:addon-cluster-admin ClusterRoleBinding` が削除されても、Amazon EKS クラスターは引き続き機能しますが、Amazon EKS はアドオンを管理できなくなります。次のプラットフォームバージョンから始まるすべてのクラスターは新しい `ClusterRoleBinding` バージョンを使用します。
- AWS からの EKS アドオンのサブセットは Amazon EKS Hybrid Nodes との互換性が検証されています。詳細については[the section called “AWSアドオン”](#) の互換性の表を参照してください。

## Amazon EKS 自動モードI の考慮事項

Amazon EKS 自動 モードには以下のような重要なクラスター機能を提供する機能が含まれています：

- ポッドネットワーキング
- サービスネットワーキング
- クラスター DNS
- 自動スケーリング
- ブロックストレージ
- ロードバランサーコントローラー
- Pod Identity エージェント
- ノードモニタリングエージェント

自動 mode コンピューティングでは以下に示すような、一般的に使用される多くの EKS アドオンが冗長になります：

- Amazon VPC CNI
- kube-proxy
- CoreDNS
- Amazon EBS CSI ドライバー
- EKS Pod Identity エージェント

ただし、クラスターが 自動 mode をセルフマネージド EC2 インスタンス、マネージド型ノードグループ、AWS Fargate などの他のコンピューティングオプションと組み合わせる場合、これらのアドオンは引き続き必要です。AWS はアンチアフィニティルールを用いて EKS アドオンを強化し、アドオンポッドがサポートされているコンピュートタイプのみ自動的にスケジュールされるようにしています。さらに、ユーザーは EKS アドオン DescribeAddonVersions API を活用して、各アドオンとその特定のバージョンでサポートされている computeTypes を検証できるようになりました。加えて EKS 自動 モードでは上記のコントローラーは AWS の所有するインフラストラクチャで実行されます。そのため、EKS 自動 モードを他のタイプのコンピューティングと組み合わせて使用している場合をのぞいて、これらのコントローラーをアカウントで見ることすらないかもしれません。組み合わせて使用している場合はクラスターにインストールしたコントローラーが表示されます。

既存のクラスターで EKS 自動モードI を有効にする予定がある場合は特定のアドオンのバージョンアップグレードが必要になる場合があります。詳細については「EKS 自動モードI の [the section called “必要なアドオンバージョン”](#)」を参照してください。

## サポート

AWS はサポートレベルが異なる複数のタイプのアドオンを発行します。

- AWS アドオン：これらのアドオンは によって構築され、完全にサポートされていますAWS。
  - AWS アドオンを使用して、Amazon EFS などのAWS 他のサービスと連携します。
  - 詳細については「[the section called “AWSアドオン”](#)」を参照してください。
- AWS マーケットプレイス アドオン：これらのアドオンは独立したAWSパートナーによってスキャンAWSされ、サポートされます。
  - マーケットプレイス アドオンを使用して、Splunk によるモニタリングなど、クラスターに価値のある高度な機能を追加します。
  - 詳細については「[the section called “マーケットプレイス アドオン”](#)」を参照してください。
- コミュニティアドオン: これらのアドオンは によってスキャンされますAWSが、オープンソースコミュニティによってサポートされます。
  - コミュニティアドオンを使用すると、Kubernetes メトリクスサーバーなどの一般的なオープンソースソフトウェアのインストールの複雑さを軽減できます。
  - コミュニティアドオンは AWS によって構築されます。AWS は、バージョンの互換性についてのみコミュニティアドオンを検証します。
  - 詳細については、「[the section called “コミュニティアドオン”](#)」を参照してください。

次の表は各アドオンタイプのサポート範囲の詳細を示しています：

| カテゴリ | 機能              | AWSアドオン | AWS マーケットプレイス アドオン | コミュニティアドオン |
|------|-----------------|---------|--------------------|------------|
| 開発   | AWS によって構築されました | はい      | なし                 | はい         |
| 開発   | AWS によって検証済み    | はい      | いいえ                | あり*        |

| カテゴリ            | 機能                                   | AWSアドオン | AWS マーケット<br>プレイス アドオ<br>ン | コミュニティア<br>ドオン |
|-----------------|--------------------------------------|---------|----------------------------|----------------|
| 開発              | AWS パートナー<br>によって検証済<br>み            | いいえ     | あり                         | いいえ            |
| メンテナンス          | AWS によってス<br>キャンされまし<br>た            | はい      | あり                         | はい             |
| メンテナンス          | AWS によって修<br>正されました                  | はい      | なし                         | はい             |
| メンテナンス          | AWS パートナー<br>によってパッチ<br>が適用されまし<br>た | いいえ     | あり                         | いいえ            |
| ディストリ<br>ビューション | AWS によって公<br>開                       | はい      | なし                         | はい             |
| ディストリ<br>ビューション | AWS パートナー<br>によって公開                  | いいえ     | あり                         | いいえ            |
| サポート            | AWSによる基本<br>インストールサ<br>ポート           | はい      | あり                         | はい             |
| サポート            | フルAWSサポー<br>ト                        | はい      | いいえ                        | いいえ            |
| サポート            | 完全なAWSパー<br>トナーサポート                  | いいえ     | あり                         | いいえ            |

\*: コミュニティアドオンの検証には、Kubernetes バージョンの互換性のみが含まれます。例えば、クラスターにコミュニティアドオンをインストールすると、AWS はクラスターの Kubernetes バージョンと互換性があるかどうかを確認します。

AWS マーケットプレイス アドオンは の外部ソースから追加のソフトウェア依存関係をダウンロードできますAWS。これらの外部依存関係は によってスキャンまたは検証されませんAWS。外部の依存関係を取得する AWS マーケットプレイス アドオンをデプロイするときはセキュリティ要件を考慮してください。

## AWSアドオン

以下の Amazon EKS アドオンをクラスターで作成できます。eksctl、AWS Management Console、または AWS CLI を使用して、使用可能なアドオンの最新リストを表示できます。使用可能なすべてのアドオンを確認したり、アドオンをインストールしたりするには[the section called “アドオンを作成する”](#)を参照してください。アドオンが IAM アクセス権を要求する場合、クラスター用に IAM OpenID Connect (OIDC) プロバイダーが必要です。既に存在しているかどうかを確認する、または作成するには「[the section called “IAM OIDC プロバイダー”](#)」を参照してください。アドオンはインストール後に、作成または削除できます。詳細については「[the section called “アドオンを更新する”](#)」または「[the section called “アドオンを削除する”](#)」を参照してください。Amazon EKS Hybrid Nodes で EKS アドオンを実行する際の考慮事項の詳細については「[the section called “CNI アドオンの設定”](#)」を参照してください。

次の Amazon EKS アドオンのいずれかを使用できます。

| 説明                                                 | 詳細はこちら                                                               | 互換性のあるコンピューティングタイプ                     |
|----------------------------------------------------|----------------------------------------------------------------------|----------------------------------------|
| クラスターのためにネイティブ VPC ネットワークを提供する                     | <a href="#">the section called “Amazon VPC CNI Kubernetes用プラグイン”</a> | EC2                                    |
| Kubernetes クラスター DNS として機能する、フレキシブルで拡張可能な DNS サーバー | <a href="#">the section called “CoreDNS”</a>                         | EC2、Fargate、EKS 自動モード、EKS Hybrid Nodes |
| 各 Amazon EC2 ノードのネットワークルールを管理する                    | <a href="#">the section called “Kube-proxy ”</a>                     | EC2、EKS Hybrid Nodes                   |

| 説明                                                                                                             | 詳細はこちら                                                                  | 互換性のあるコンピューティングタイプ                     |
|----------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------|----------------------------------------|
| クラスターのために Amazon EBS ストレージを提供する                                                                                | <a href="#">the section called “Amazon EBS CSI ドライバー”</a>               | EC2                                    |
| クラスターのために Amazon EFS ストレージを提供する                                                                                | <a href="#">the section called “Amazon EFS CSI ドライバー”</a>               | EC2、EKS 自動モード                          |
| クラスターに Amazon FSx for Lustre ストレージを提供する                                                                        | <a href="#">the section called “Amazon FSx CSI ドライバー”</a>               | EC2、EKS 自動モード                          |
| クラスターのために Amazon S3 ストレージを提供する                                                                                 | <a href="#">the section called “Mountpoint for Amazon S3 CSI ドライバー”</a> | EC2、EKS 自動モード                          |
| 追加ノードのヘルス問題を検出する                                                                                               | <a href="#">the section called “ノードモニタリングエージェント”</a>                    | EC2、EKS Hybrid Nodes                   |
| Amazon EBS CSI ドライバーなどの互換性のある CSI ドライバーでスナップショット機能の使用を有効にする                                                    | <a href="#">the section called “CSI スナップショットコントローラー”</a>                | EC2、Fargate、EKS 自動モード、EKS Hybrid Nodes |
| SageMaker HyperPod タスクガバナンスは Amazon EKS クラスター内のチーム間のコンピューティングリソースの割り当てと使用状況を最適化し、タスクの優先順位付けとリソース共有の非効率性に対処します。 | <a href="#">the section called “Amazon SageMaker HyperPod タスクガバナンス”</a> | EC2、EKS 自動モード                          |

| 説明                                                                                                                                                        | 詳細はこちら                                                            | 互換性のあるコンピューティングタイプ                     |
|-----------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------|----------------------------------------|
| ネットワークフローデータを収集して Amazon CloudWatch に報告する Kubernetes エージェント。クラスターノード間の TCP 接続を包括的にモニタリングできます。                                                             | <a href="#">the section called “ネットワーク・フロー・モニターの AWS”</a>         | EC2、EKS 自動モード                          |
| オープンテレメトリー プロジェクトの、安全かつ本番に対応した、AWS によってサポートされているディストリビューション                                                                                               | <a href="#">the section called “AWS オープンテレメトリー用ディストロ”</a>         | EC2、Fargate、EKS 自動モード、EKS Hybrid Nodes |
| AWS CloudTrail 管理イベントや Amazon VPC フローログなどの基本的なデータソースを分析および処理するセキュリティモニタリングサービス。Amazon GuardDuty は、Kubernetes 監査ログやランタイムモニタリングなどの機能も処理します                  | <a href="#">the section called “Amazon GuardDuty エージェント”</a>      | EC2、EKS 自動モード                          |
| AWS によって提供されるモニタリングおよびオブザーバビリティサービス。このアドオンは CloudWatch エージェントをインストールし、Amazon EKS のオブザーバビリティが強化された CloudWatch アプリケーションシグナルと CloudWatch コンテナインサイトの両方を有効にします | <a href="#">the section called “Amazon CloudWatch 観測可能エージェント”</a> | EC2、EKS 自動モード、EKS Hybrid Nodes         |

| 説明                                                                       | 詳細はこちら                                                                        | 互換性のあるコンピューティングタイプ                     |
|--------------------------------------------------------------------------|-------------------------------------------------------------------------------|----------------------------------------|
| EC2 インスタンスプロファイルから EC2 インスタンスに認証情報を提供する場合と同じような方法で、アプリケーションの認証情報を管理する機能。 | <a href="#">the section called “EKS Pod Identity エージェント”</a>                  | EC2、EKS Hybrid Nodes                   |
| cert-manager を有効にして、AWS プライベート CA から X.509 証明書を発行します。cert-manager が必要です。 | <a href="#">the section called “AWS プライベート CA の Connector for Kubernetes”</a> | EC2、Fargate、EKS 自動モード、EKS Hybrid Nodes |

## Amazon VPC CNI Kubernetes用プラグイン

Amazon VPC CNI Plugin for Kubernetes の Amazon EKS アドオンは、クラスターにネイティブ VPC ネットワークを提供する Kubernetes コンテナネットワークインターフェイス (CNI) プラグインです。このアドオンのセルフマネージドタイプまたはマネージドタイプが、デフォルトで各 Amazon EC2 ノードにインストールされます。詳細については「[Kubernetesコンテナネットワークインターフェイス \( CNI \) プラグイン](#)」を参照してください。

### Note

このアドオンを Amazon EKS 自動モード クラスターにインストールする必要はありません。詳細については「[the section called “Amazon EKS 自動モードの考慮事項”](#)」を参照してください。

Amazon EKS アドオン名は vpc-cni です。

### 必要な IAM 許可

このアドオンはAmazon EKS のサービスアカウント用の IAM 役割機能を使用します。詳細については「[the section called “IRSA による認証情報”](#)」を参照してください。



クラスターが IPv4 ファミリーを使用する場合は[AmazonEKS\\_CNI\\_Policy](#) のアクセス権限が必要です。クラスターが IPv6 ファミリーを使用する場合は[IPv6 モード](#)のアクセス許可を持つ [IAM ポリシー](#)を作成する必要があります。次のコマンドで、IAM ロールを作成し、そのロールにポリシーの 1 つをアタッチして、アドオンが使用する Kubernetes サービスアカウントに注釈を付けることができます。

**#####** をクラスターの名前に置き換え、**AmazonEKSVPCCNIRole** を役割の名前に置き換えます。クラスターが IPv6 ファミリーを使用する場合は**AmazonEKS\_CNI\_Policy** を作成したポリシーの名前に置き換えてください。このコマンドを使用するにはデバイスに [eksctl](#) がインストールされている必要があります。別のツールを使用してロールを作成し、ポリシーをアタッチして、Kubernetes サービスアカウントに注釈を付ける必要がある場合は、「[the section called “IAM ロールを割り当てる”](#)」を参照してください。

```
eksctl create iamserviceaccount --name aws-node --namespace kube-system --cluster my-cluster --role-name AmazonEKSVPCCNIRole \
 --role-only --attach-policy-arn arn:aws:iam::aws:policy/AmazonEKS_CNI_Policy --approve
```

## 情報を更新する

一度に更新できるマイナーバージョンは 1 つのみです。例えば、現在のバージョンが **1.28.x-eksbuild.y** で、これを **1.30.x-eksbuild.y** に更新する場合は現在のバージョンを **1.29.x-eksbuild.y** に更新してから、再度 **1.30.x-eksbuild.y** に更新する必要があります。アドオンの更新の詳細については「[the section called “更新 \(EKS アドオン\)”](#)」を参照してください。

## CoreDNS

CoreDNS の Amazon EKS アドオンは、Kubernetes クラスター DNS として機能できる柔軟で拡張可能な DNS サーバーです。このアドオンのセルフマネージドタイプまたはマネージドタイプが、クラスターの作成時にデフォルトでインストールされました。少なくとも 1 つのノードで Amazon EKS クラスターを起動すると、クラスターにデプロイされたノード数に関係なく、デフォルトで CoreDNS イメージの 2 つのレプリカがデプロイされます。これらの CoreDNS ポッドは、クラスター内のすべてのポッドの名前解決を行います。クラスターに Fargate プロファイルが含まれ、名前空間が CoreDNS デプロイの名前空間と一致している場合、CoreDNS Pod を Fargate ノードにデプロイできます。詳細については、[the section called “プロファイルを定義する”](#)を参照してください。

**Note**

このアドオンを Amazon EKS 自動モード クラスターにインストールする必要はありません。詳細については「[the section called “Amazon EKS 自動モードI の考慮事項”](#)」を参照してください。

Amazon EKS アドオン名は `coredns` です。

必要な IAM 許可

このアドオンには許可は必要ありません。

追加情報

CoreDNS の詳細については、Kubernetes ドキュメントの「[サービスディスカバリーに CoreDNS を使用する](#)」および「[Customizing DNS Service](#)」を参照してください。

## Kube-proxy

Kube-proxy Amazon EKS アドオンは各 Amazon EC2 ノードのネットワークルールを維持します。これにより、Pod とのネットワーク通信が可能になります。このアドオンのセルフマネージドタイプまたはマネージドタイプが、デフォルトでクラスター内の各 Amazon EC2 ノードにインストールされます。

**Note**

このアドオンを Amazon EKS 自動モード クラスターにインストールする必要はありません。詳細については「[the section called “Amazon EKS 自動モードI の考慮事項”](#)」を参照してください。

Amazon EKS アドオン名は `kube-proxy` です。

必要な IAM 許可

このアドオンには許可は必要ありません。

情報を更新する

現在のバージョンを更新する前に、次の要件を考慮してください:

- Amazon EKS クラスターの Kube-proxy は [Kubernetes と同じ互換性およびスキューポリシー](#) が適用されます。

## 追加情報

kube-proxy の詳細については、Kubernetes ドキュメントの「[kube-proxy](#)」を参照してください。

## Amazon EBS CSI ドライバー

Amazon EBS CSI ドライバーの Amazon EKS アドオンは、クラスター用に Amazon EBS ストレージを提供する Kubernetes Container Storage Interface (CSI) プラグインです。

### Note

このアドオンを Amazon EKS 自動モード クラスターにインストールする必要はありません。自動モード にはブロックストレージ機能があります。詳細については「[the section called “ステートフルワークロードをデプロイする”](#)」を参照してください。

Amazon EKS アドオン名は `aws-ebs-csi-driver` です。

## 必要な IAM 許可

このアドオンは Amazon EKS のサービスアカウント用の IAM 役割機能を使用します。詳細については「[the section called “IRSA による認証情報”](#)」を参照してください。[AmazonEBSCSIDriverPolicy](#) AWS マネージドポリシーの許可が必要です。次のコマンドで IAM 役割を作成して、それにマネージドポリシーをアタッチできます。`#####` をクラスターの名前に置き換え、`AmazonEKS_EBS_CSI_DriverRole` を役割の名前に置き換えます。このコマンドを使用するにはデバイスに `eksctl` がインストールされている必要があります。別のツールを使用する必要がある場合や、暗号化にカスタム [KMS キー](#) を使用する必要がある場合は [the section called “ステップ 1: IAM ロールを作成する”](#) を参照してください。

```
eksctl create iamserviceaccount \
 --name ebs-csi-controller-sa \
 --namespace kube-system \
 --cluster my-cluster \
 --role-name AmazonEKS_EBS_CSI_DriverRole \
 --role-only \
 --attach-policy-arn arn:aws:iam::aws:policy/service-role/AmazonEBSCSIDriverPolicy \
 --approve
```

## 追加情報

アドオンの詳細については「[the section called “Amazon EBS”](#)」を参照してください。

## Amazon EFS CSI ドライバー

Amazon EFS CSI ドライバー Amazon EKS アドオンは、クラスター用に Amazon EFS ストレージを提供する Kubernetes Container Storage Interface (CSI) プラグインです。

Amazon EKS アドオン名は `aws-efs-csi-driver` です。

### 必要な IAM 許可

必要な IAM アクセス権限 — このアドオンは Amazon EKS の サービスアカウントの IAM 役割機能を利用します。詳細については「[the section called “IRSA による認証情報”](#)」を参照してください。[AmazonEFSCSIDriverPolicy](#) AWS マネージドポリシーの許可が必要です。次のコマンドで IAM 役割を作成して、その役割にマネージドポリシーをアタッチできます。`#####` をクラスターの名前に置き換え、`AmazonEKS_EFS_CSI_DriverRole` を役割の名前に置き換えます。これらのコマンドを使用するにはデバイスに `eksctl` がインストールされている必要があります。別のツールを使用する必要がある場合は[the section called “ステップ 1: IAM ロールを作成する”](#)を参照してください。

```
export cluster_name=my-cluster
export role_name=AmazonEKS_EFS_CSI_DriverRole
eksctl create iamserviceaccount \
 --name efs-csi-controller-sa \
 --namespace kube-system \
 --cluster $cluster_name \
 --role-name $role_name \
 --role-only \
 --attach-policy-arn arn:aws:iam::aws:policy/service-role/AmazonEFSCSIDriverPolicy \
 --approve
TRUST_POLICY=$(aws iam get-role --output json --role-name $role_name --query
 'Role.AssumeRolePolicyDocument' | \
 sed -e 's/efs-csi-controller-sa/efs-csi-*/' -e 's/StringEquals/StringLike/')
aws iam update-assume-role-policy --role-name $role_name --policy-document
"$TRUST_POLICY"
```

## 追加情報

アドオンの詳細については「[the section called “Amazon EFS”](#)」を参照してください。

## Amazon FSx CSI ドライバー

Amazon FSx CSI ドライバー Amazon EKS アドオンは、クラスター用に Amazon FSx for Lustre ストレージを提供する Kubernetes Container Storage Interface (CSI) プラグインです。

Amazon EKS アドオン名は `aws-fsx-csi-driver` です。

### Note

- クラスターに既存の Amazon FSx CSI ドライバーをインストールすると、アドオンのインストールに失敗する可能性があります。EKS 以外の FSx CSI ドライバーが存在するところに Amazon EKS のアドオンバージョンをインストールしようとする、リソースの競合によりインストールが失敗します。この問題を解決するには、インストール時に `OVERWRITE` フラグを使用します。

```
aws eks create-addon --addon-name aws-fsx-csi-driver --cluster-name my-cluster
--resolve-conflicts OVERWRITE
```

- Amazon FSx CSI ドライバー EKS アドオンの認証には、EKS Pod Identity エージェントが必要です。このコンポーネントがないと、アドオンは「Amazon EKS Pod Identity agent is not installed in the cluster」というエラーで失敗し、ボリュームオペレーションが回避されます。FSx CSI ドライバーアドオンをデプロイする前後に Pod Identity エージェントをインストールします。詳細については、「[the section called “エージェントのセットアップ”](#)」を参照してください。

### 必要な IAM アクセス許可

このアドオンは Amazon EKS のサービスアカウント用の IAM 役割機能を使用します。詳細については、「[the section called “IRSA による認証情報”](#)」を参照してください。[AmazonFSxFullAccess](#) AWS マネージドポリシーの許可が必要です。次のコマンドで IAM 役割を作成して、それに マネージドポリシーをアタッチできます。`#####` をクラスターの名前に置き換え、`AmazonEKS_FSx_CSI` をロールの名前に置き換えます。このコマンドを使用するにはデバイスに `eksctl` がインストールされている必要があります。

```
eksctl create iamserviceaccount \
 --name fsx-csi-controller-sa \
 --namespace kube-system \
 --cluster my-cluster \
```

```
--role-name AmazonEKS_FSx_CSI_DriverRole \
--role-only \
--attach-policy-arn arn:aws:iam::aws:policy/AmazonFSxFullAccess \
--approve
```

## 追加情報

アドオンの詳細については「[the section called “Amazon FSx for Lustre”](#)」を参照してください。

## Mountpoint for Amazon S3 CSI ドライバー

Mountpoint for Amazon S3 CSI ドライバーの Amazon EKS アドオンは、クラスター用に Amazon S3 ストレージを提供する Kubernetes Container Storage Interface (CSI) プラグインです。

Amazon EKS アドオン名は `aws-mountpoint-s3-csi-driver` です。

### 必要な IAM 許可

このアドオンは Amazon EKS のサービスアカウント用の IAM 役割機能を使用します。詳細については「[the section called “IRSA による認証情報”](#)」を参照してください。

作成される IAM 役割には S3 へのアクセスを許可するポリシーが必要です。ポリシーを作成するときは[Mountpoint IAM アクセス許可の推奨事項](#)に従ってください。別の方法として、AWS マネージドポリシー [AmazonS3FullAccess](#) を使用することもできますが、このマネージドポリシーでは Mountpoint に必要以上のアクセス権限が付与されます。

次のコマンドで IAM 役割を作成して、その役割にマネージドポリシーをアタッチできます。##### をクラスターの名前に、##### を正しい AWS リージョンコードに、*AmazonEKS\_S3\_CSI\_DriverRole* を役割の名前に、*AmazonEKS\_S3\_CSI\_DriverRole\_ARN* を役割 ARN に置き換えます。これらのコマンドを使用するにはデバイスに [eksctl](#) がインストールされている必要があります。IAM コンソールまたは AWS CLI を使用する手順については「[the section called “ステップ 2: IAM ロールを作成する”](#)」を参照してください。

```
CLUSTER_NAME=my-cluster
REGION=region-code
ROLE_NAME=AmazonEKS_S3_CSI_DriverRole
POLICY_ARN=AmazonEKS_S3_CSI_DriverRole_ARN
eksctl create iamserviceaccount \
 --name s3-csi-driver-sa \
 --role-name $ROLE_NAME \
 --attach-policy-arn $POLICY_ARN
```

```
--namespace kube-system \
--cluster $CLUSTER_NAME \
--attach-policy-arn $POLICY_ARN \
--approve \
--role-name $ROLE_NAME \
--region $REGION \
--role-only
```

## 追加情報

アドオンの詳細については「[the section called “Mountpoint for Amazon S3”](#)」を参照してください。

## CSI スナップショットコントローラー

コンテナ・ストレージ・インターフェース (CSI) スナップショットコントローラーを使用すると、Amazon EBS CSI ドライバーなどの互換性のある CSI ドライバーのスナップショット機能を使用できます。

Amazon EKS アドオン名は `snapshot-controller` です。

### 必要な IAM 許可

このアドオンには許可は必要ありません。

## 追加情報

アドオンの詳細については「[the section called “CSI スナップショットコントローラー”](#)」を参照してください。

## Amazon SageMaker HyperPod タスクガバナンス

SageMaker HyperPod タスクガバナンスはリソースの割り当てを合理化し、Amazon EKS クラスターのチームやプロジェクト全体でコンピューティングリソースを効率的に活用できるように設計されています。これにより、管理者は次の設定を行うことができます。

- さまざまなタスクの優先度
- 各チームのコンピューティング割り当て
- 各チームがアイドル状態のコンピューティングリソースを貸し借りする方法
- チームが自らのタスクを先取りした場合

HyperPod タスクガバナンスは Amazon EKS クラスターオブザーバビリティも提供し、クラスター容量をリアルタイムで可視化します。これにはコンピューティングの可用性と使用状況、チームの割り当てと使用状況、タスクの実行と待機時間に関する情報が含まれ、情報に基づいた意思決定とプロアクティブなリソース管理のための設定を行います。

Amazon EKS アドオン名は `amazon-sagemaker-hyperpod-taskgovernance` です。

必要な IAM 許可

このアドオンには許可は必要ありません。

追加情報

アドオンの詳細については [SageMaker HyperPod タスクガバナンス](#)」を参照してください。

## ネットワーク・フロー・モニター の AWS

Amazon CloudWatch ネットワーク・フロー・モニター エージェントはクラスター内のすべてのノードから TCP 接続統計を収集し、ネットワークフローレポートを Amazon CloudWatch ネットワーク・フロー・モニター 摄取 APIs。

Amazon EKS アドオン名は `aws-network-flow-monitoring-agent` です。

必要な IAM 許可

このアドオンには IAM 権限が必要です。

アドオンに `CloudWatchNetworkFlowMonitorAgentPublishPolicy` 管理ポリシーを添付する必要があります。

必要な IAM セットアップの詳細については Amazon CloudWatch ネットワーク・フロー・モニター エージェント GitHub リポジトリの [「IAM ポリシー」](#) を参照してください。

管理ポリシーの詳細については、Amazon CloudWatch ユーザーガイドの [CloudWatchNetworkFlowMonitorAgentPublishPolicy](#) を参照してください。

追加情報

アドオンの詳細については [Amazon CloudWatch ネットワーク・フロー・モニター エージェント GitHub リポジトリ](#) を参照してください。



## ノードモニタリングエージェント

ノードモニタリングエージェントの Amazon EKS アドオンは追加ノードヘルスの問題を検出できます。これらの追加のヘルスシグナルはオプションのノード自動修復機能でも活用でき、必要に応じてノードを自動的に置き換えることができます。

### Note

このアドオンを Amazon EKS 自動モード クラスターにインストールする必要はありません。詳細については「[the section called “Amazon EKS 自動モードI の考慮事項”](#)」を参照してください。

Amazon EKS アドオン名は eks-node-monitoring-agent です。

必要な IAM 許可

このアドオンに追加の許可は必要ありません。

追加情報

詳細については「[the section called “ノードのヘルス”](#)」を参照してください。

## AWS オープンテレメトリー用ディストロ

AWS オープンテレメトリー用ディストロ Amazon EKS アドオンは安全な本番環境対応の、AWS によってサポートされている オープンテレメトリー プロジェクトのディストリビューションです。詳細についてはGitHubで「[AWS オープンテレメトリー用ディストロ](#)」を参照してください。

Amazon EKS アドオン名は adot です。

必要な IAM 許可

このアドオンには詳細設定でオプトインできる事前設定されたカスタムリソースの 1 つを使用している場合のみ IAM 許可が必要です。

追加情報

詳細については、AWS Distro for OpenTelemetry ドキュメントの「[Getting Started with AWS Distro for OpenTelemetry using EKS Add-Ons](#)」を参照してください。

ADOT では前提条件として cert-manager アドオンがクラスターにデプロイされている必要があります。そうでなければ、<https://registry.terraform.io/modules/terraform-aws-modules/eks/aws/latest> cluster\_addons プロパティを使用して直接デプロイした場合、このアドオンは機能しません。その他の要件については、AWS Distro for OpenTelemetry ドキュメントの「[Requirements for Getting Started with AWS Distro for OpenTelemetry using EKS Add-Ons](#)」を参照してください。

## Amazon GuardDuty エージェント

Amazon GuardDuty エージェント Amazon EKS アドオンでは、GuardDuty Runtime Monitoring による分析のために、EKS クラスターノードから[ランタイムイベント](#) (ファイルアクセス、プロセス実行、ネットワーク接続) を収集します。GuardDuty 自体 (エージェントではない) は、AWS CloudTrail 管理イベントや Amazon VPC フローログなどの[基本的なデータソース](#)と、Kubernetes 監査ログやランタイムモニタリングなどの[機能](#)を分析して処理するセキュリティモニタリングサービスです。

Amazon EKS アドオン名は aws-guardduty-agent です。

必要な IAM 許可

このアドオンには許可は必要ありません。

追加情報

詳細については「[Amazon GuardDuty における Amazon EKS クラスターのランタイム・モニタリング](#)」を参照してください。

- Amazon EKS クラスターで潜在的なセキュリティの脅威を検出するには Amazon GuardDuty ランタイムモニタリングを有効にし、ガードデューティ セキュリティエージェントを Amazon EKS クラスターにデプロイします。

## Amazon CloudWatch 観測可能エージェント

Amazon CloudWatch 観測可能エージェント Amazon EKS アドオンは AWS が提供するモニタリングおよびオブザーバビリティサービスです。このアドオンは CloudWatch エージェントをインストールし、Amazon EKS のオブザーバビリティが強化された CloudWatch アプリケーションシグナルと CloudWatch コンテナインサイトの両方を有効にします。詳細については「[Amazon CloudWatch エージェント](#)」を参照してください。

Amazon EKS アドオン名は amazon-cloudwatch-observability です。

## 必要な IAM 許可

このアドオンはAmazon EKS のサービスアカウント用の IAM 役割機能を使用します。詳細については「[the section called “IRSA による認証情報”](#)」を参照してください。[AWSXrayWriteOnlyAccess](#) と [CloudWatchAgentServerPolicy](#) AWS の管理ポリシーのパーミッションが必要です。次のコマンドで IAM ロールを作成し、それに管理ポリシーをアタッチして、アドオンが使用する Kubernetes サービスアカウントに注釈を付けることができます。**#####** をクラスターの名前に置き換え、**AmazonEKS\_Observability\_role** を役割の名前に置き換えます。このコマンドを使用するにはデバイスに [eksctl](#) がインストールされている必要があります。別のツールを使用してロールを作成し、ポリシーをアタッチして、Kubernetes サービスアカウントに注釈を付ける必要がある場合は、「[the section called “IAM ロールを割り当てる”](#)」を参照してください。

```
eksctl create iamserviceaccount \
 --name cloudwatch-agent \
 --namespace amazon-cloudwatch \
 --cluster my-cluster \
 --role-name AmazonEKS_Observability_Role \
 --role-only \
 --attach-policy-arn arn:aws:iam::aws:policy/AWSXrayWriteOnlyAccess \
 --attach-policy-arn arn:aws:iam::aws:policy/CloudWatchAgentServerPolicy \
 --approve
```

## 追加情報

詳細については「[CloudWatch エージェントのインストール](#)」を参照してください。

## AWS プライベート CA の Connector for Kubernetes

Kubernetes 用 AWS プライベート CA Connector は、ユーザーが AWS プライベート Certificate Authority (AWS プライベート CA) から証明書を取得できるようにする cert-manager のアドオンです。

- Amazon EKS アドオン名は aws-privateca-connector-for-kubernetes です。
- アドオン名前空間は aws-privateca-issuer です。

このアドオンには cert-manager が必要です。cert-manager はコミュニティアドオンとして Amazon EKS で使用できます。このアドオンの詳細については、「[the section called “Cert Manager”](#)」を参照してください。アドオンのインストールに関する詳細については「[the section called “アドオンを作成する”](#)」を参照してください。

## 必要な IAM 許可

このアドオンには IAM 権限が必要です。

EKS Pod Identities を使用して `AWSPrivateCAConnectorForKubernetesPolicy` IAM ポリシーを `aws-privateca-issuer` Kubernetes サービスアカウントにアタッチします。詳細については、「[the section called “Pod Identity を使用する”](#)」を参照してください。

必要な権限の詳細については、「AWS Managed Policy Reference」の「[AWSPrivateCAConnectorForKubernetesPolicy](#)」を参照してください。

## 追加情報

詳細については、[AWS Private CA Issuer for Kubernetes GitHub リポジトリ](#)を参照してください。

アドオン設定の詳細については、`aws-privateca-issuer` GitHub リポジトリの「[values.yaml](#)」を参照してください。`values.yaml` のバージョンが、クラスターにインストールされているアドオンのバージョンと一致していることを確認します。

このアドオンは、EKS Auto Mode の `system NodePool` で使用される `CriticalAddonsOnly` ティントを許容します。詳細については、「[the section called “重要なアドオンを実行する”](#)」を参照してください。

## EKS Pod Identity エージェント

Amazon EKS Pod Identity Agent Amazon EKS アドオンは EC2 インスタンスプロファイルから EC2 インスタンスに認証情報を提供する場合と同じような方法で、アプリケーションの認証情報を管理する機能を提供します。

### Note

このアドオンを Amazon EKS 自動モード クラスターにインストールする必要はありません。Amazon EKS 自動モード は EKS Pod Identity と統合されています。詳細については「[the section called “Amazon EKS 自動モードの考慮事項”](#)」を参照してください。

Amazon EKS アドオン名は `eks-pod-identity-agent` です。

## 必要な IAM 許可

Pod Identity Agent アドオン自体には IAM ロールは必要ありません。[Amazon EKS ノードの IAM ロール](#)からの許可を使用して機能しますが、アドオン専用の IAM ロールは必要ありません。

## 情報を更新する

一度に更新できるマイナーバージョンは 1 つのみです。例えば、現在のバージョンが 1.28.x-eksbuild.y で、これを 1.30.x-eksbuild.y に更新する場合は現在のバージョンを 1.29.x-eksbuild.y に更新してから、再度 1.30.x-eksbuild.y に更新する必要があります。アドオンの更新の詳細については「[the section called “アドオンを更新する”](#)」を参照してください。

## コミュニティアドオン

AWS API を使用して、Kubernetes Metrics Server などのコミュニティ アドオンをインストールできます。複数のクラスターでソフトウェアを保守する際の複雑さを軽減するために、コミュニティアドオンを Amazon EKS アドオンとしてインストールすることもできます。

例えば、AWS API、CLI、または管理コンソールを使用してコミュニティ アドオンをインストールできます。クラスターの作成時にコミュニティアドオンをインストールできます。

コミュニティアドオンは既存の Amazon EKS アドオンと同様に管理します。コミュニティアドオンは独自のサポート範囲があるという点で、既存のアドオンとは異なります。

### Note

コミュニティアドオンの使用はユーザーが自由に決定できます。お客様と AWS 間の[責任共有モデル](#)の一環として、これらのサードパーティープラグインに関して、何をクラスターにインストールするかを理解しておく必要があります。また、お客様はクラスターのセキュリティニーズを満たすコミュニティアドオンについても責任を負います。詳細については、「[the section called “EKS にデプロイされたソフトウェアのサポート”](#)」を参照してください。

コミュニティアドオンは AWS によってパッケージ化され、AWS はバージョンの互換性についてのみコミュニティアドオンを検証します。例えば、クラスターにコミュニティアドオンをインストールすると、AWS はクラスターの Kubernetes バージョンと互換性があるかどうかを確認します。

重要なのは AWS はコミュニティアドオンを完全にサポートしていないことです。AWS はアドオンのインストールやアドオンの削除など、AWS API を使用して行われたライフサイクルオペレーションのみをサポートします。

Sコミュニティアドオンのサポートが必要な場合は、既存のプロジェクトリソースを使用します。例えば、プロジェクトのリポジトリに GitHub の問題を作成できます。

## アドオンタイプを決定する

AWS CLI を使用して、Amazon EKS アドオンのタイプを判断できます。

アドオンに関する情報を取得するには、次の CLI コマンドを使用します。metrics-server を任意のアドオンの名前に置き換えることができます。

```
aws eks describe-addon-versions --addon-name metrics-server
```

owner フィールドの CLI 出力を確認します。

```
{
 "addons": [
 {
 "addonName": "metrics-server",
 "type": "observability",
 "owner": "community",
 "addonVersions": [
```

owner の値が community の場合、アドオンはコミュニティ アドオンです。AWS は、アドオンのインストール、更新、および削除のサポートのみを提供します。アドオン自体の機能とオペレーションについて質問がある場合は、GitHub の問題などのコミュニティリソースを使用してください。

## コミュニティアドオンをインストールまたは更新する

コミュニティアドオンは他の Amazon EKS アドオンと同じ方法でインストールまたは更新します。

- [the section called “アドオンを作成する”](#)
- [the section called “アドオンを更新する”](#)
- [the section called “アドオンを削除する”](#)

## 利用可能なコミュニティアドオン

Amazon EKS で利用できるコミュニティアドオンは次のとおりです。

### Kubernetes メトリクスサーバー

Kubernetes Metrics Server は、Kubernetes の組み込み自動スケーリングパイプラインのコンテナリソースメトリクスのスケーラブルで効率的なソースです。Kubelets からリソースメトリクスを

収集し、Horizontal Pod Autoscaler と Vertical Pod Autoscaler で使用するメトリクス API を通じて Kubernetes apiserver で公開します。

| プロパティ          | 値                             |
|----------------|-------------------------------|
| アドオン名          | metrics-server                |
| 名前空間           | kube-system                   |
| ドキュメント         | <a href="#">GitHub Readme</a> |
| サービスアカウント名     | なし                            |
| マネージド IAM ポリシー | なし                            |
| カスタム IAM 許可    | なし                            |

### kube-state-metrics

クラスターレベルのメトリクスを生成して公開するアドオンエージェント。

Kubernetes API の Kubernetes オブジェクトの状態は、メトリクスとして公開できます。kube-state-metrics というアドオンエージェントは、Kubernetes API サーバーに接続し、クラスター内の個々のオブジェクトの状態から生成されたメトリクスを使用して HTTP エンドポイントを公開することができます。ラベルと注釈、起動時間と終了時間、オブジェクトの現在のステータスまたはフェーズなど、オブジェクトの状態に関するさまざまな情報が公開されます。

| プロパティ          | 値                                                                          |
|----------------|----------------------------------------------------------------------------|
| アドオン名          | kube-state-metrics                                                         |
| 名前空間           | kube-state-metrics                                                         |
| ドキュメント         | Kubernetes ドキュメントの「 <a href="#">Metrics for Kubernetes Object States</a> 」 |
| サービスアカウント名     | なし                                                                         |
| マネージド IAM ポリシー | なし                                                                         |

| プロパティ       | 値  |
|-------------|----|
| カスタム IAM 許可 | なし |

## Prometheus Node Exporter

プラグブルメトリクスコレクターを使用して Go で記述された \*NIX カーネルによって公開される、ハードウェアおよび OS メトリクス用の Prometheus エクスポーター。Prometheus Node Exporter は、さまざまなハードウェアおよびカーネル関連のメトリクスを公開します。

| プロパティ          | 値                                                                                          |
|----------------|--------------------------------------------------------------------------------------------|
| アドオン名          | prometheus-node-exporter                                                                   |
| 名前空間           | prometheus-node-exporter                                                                   |
| ドキュメント         | Prometheus ドキュメントの「 <a href="#">Monitoring Linux host metrics with the Node Exporter</a> 」 |
| サービスアカウント名     | なし                                                                                         |
| マネージド IAM ポリシー | なし                                                                                         |
| カスタム IAM 許可    | なし                                                                                         |

## Cert Manager

Cert Manager を使用すると、証明書の作成と更新を管理できます。

| プロパティ      | 値                                    |
|------------|--------------------------------------|
| アドオン名      | cert-manager                         |
| 名前空間       | cert-manager                         |
| ドキュメント     | <a href="#">Cert Manager のドキュメント</a> |
| サービスアカウント名 | なし                                   |



| プロパティ          | 値  |
|----------------|----|
| マネージド IAM ポリシー | なし |
| カスタム IAM 許可    | なし |

## 外部 DNS

外部 DNS EKS アドオンを使用すると、Kubernetes リソースを介して Route53 DNS レコードを管理できます。

外部 DNS アクセス許可を、管理対象とするホストゾーンの `route53:ChangeResourceRecordSets`、`route53:ListHostedZones`、`route53:ListResourceRecords` に縮小できます。

| プロパティ          | 値                                               |
|----------------|-------------------------------------------------|
| アドオン名          | external-dns                                    |
| 名前空間           | external-dns                                    |
| ドキュメント         | <a href="#">GitHub Readme</a>                   |
| サービスアカウント名     | external-dns                                    |
| マネージド IAM ポリシー | arn:aws:iam::aws:policy/AmazonRoute53FullAccess |
| カスタム IAM 許可    | なし                                              |

## 属性を表示する

コミュニティアドオンのオープンソース属性とライセンス情報をダウンロードできます。

1. 属性をダウンロードするアドオンの名前とバージョンを決定します。
2. その名前とバージョンで次のコマンドを更新します。

```
curl -O https://amazon-eks-docs.s3.amazonaws.com/attributions/<add-on-name>/<add-on-version>/attributions.zip
```

例:

```
curl -O https://amazon-eks-docs.s3.amazonaws.com/attributions/kube-state-metrics/v2.14.0-eksbuild.1/attributions.zip
```

3. コマンドを使用して、ファイルをダウンロードします。

この zip ファイルを使用して、ライセンス属性に関する情報を表示します。

## AWS マーケットプレイス アドオン

前述の Amazon EKS アドオンのリストに加えて、独立系ソフトウェアベンダーのさまざまな運用ソフトウェア Amazon EKS アドオンを追加することもできます。アドオンを選択すると、そのアドオンとそのインストール要件について詳しく知ることができます。

### Accuknox

アドオン名は `accuknox_kubearmor` で、名前空間は `kubearmor` です。アドオンの公開は、Accuknoxで行います。

アドオンの詳細についてはKubeArmor ドキュメントの「[KubeArmorを始める](#)」を参照してください。

#### サービスアカウント名

サービスアカウントはこのアドオンでは使用されません。

#### AWS マネージド IAM ポリシー

このアドオンではマネージドポリシーは使用されません。

#### カスタム IAM 許可

このアドオンではカスタム許可は使用されません。

## Akuity

アドオン名は `akuity_agent` で、名前空間は `akuity` です。アドオンの公開は、Akuity で行います。

アドオンの使用方法の詳細についてはAkuity Platform ドキュメントの「[Akuity EKSアドオンによるAmazon EKSへのAkuityエージェントのインストール](#)」を参照してください。

### サービスアカウント名

サービスアカウントはこのアドオンでは使用されません。

### AWS マネージド IAM ポリシー

このアドオンではマネージドポリシーは使用されません。

### カスタム IAM 許可

このアドオンではカスタム許可は使用されません。

## Calyptia

アドオン名は `calyptia_fluent-bit` で、名前空間は `calyptia-fluentbit` です。アドオンの公開は、Calyptia で行います。

アドオンの詳細についてはCalyptia ドキュメントウェブサイトの「[Calyptia Core Agentを使い始める](#)」を参照してください。

### サービスアカウント名

サービスアカウント名は `calyptia-fluentbit` です。

### AWS マネージド IAM ポリシー

このアドオンは `AWSMarketplaceMeteringRegisterUsage` マネージドポリシーを使用します。詳細については「AWS マネージドポリシーリファレンスガイド」の「[AWSMarketplaceMeteringRegisterUsage](#)」を参照してください。

### 必要な IAM 役割を作成するコマンド

次のコマンドではクラスターの IAM OpenID Connect (OIDC) プロバイダーが必要です。既に存在しているかどうかを確認する、または作成するには「[the section called “IAM OIDC プロバイダー”](#)」を参照してください。`#####` をクラスターの名前に置き換え、`my-calyptia-role` を役割の名前に置き換えます。このコマンドを使用するにはデバイスに `eksctl` がインストールされている必要が

あります。別のツールを使用してロールを作成し、Kubernetes サービスアカウントに注釈を付ける必要がある場合は「[the section called “IAM ロールを割り当てる”](#)」を参照してください。

```
eksctl create iamserviceaccount --name service-account-name --namespace calyptia-
fluentbit --cluster my-cluster --role-name my-calyptia-role \
 --role-only --attach-policy-arn arn:aws:iam::aws:policy/
AWSMarketplaceMeteringRegisterUsage --approve
```

## Cisco Observability Collector

アドオン名は `cisco_cisco-cloud-observability-collectors` で、名前空間は `appdynamics` です。アドオンの公開は、Cisco で行います。

アドオンの詳細についてはCisco AppDynamics ドキュメントの「[Cisco Cloud Observabilityを使用する AWS マーケットプレイス Add-Ons](#)」を参照してください。

サービスアカウント名

サービスアカウントはこのアドオンでは使用されません。

AWS マネージド IAM ポリシー

このアドオンではマネージドポリシーは使用されません。

カスタム IAM 許可

このアドオンではカスタム許可は使用されません。

## Cisco Observability Operator

アドオン名は `cisco_cisco-cloud-observability-operators` で、名前空間は `appdynamics` です。アドオンの公開は、Cisco で行います。

アドオンの詳細についてはCisco AppDynamics ドキュメントの「[Cisco Cloud Observabilityを使用する AWS マーケットプレイス Add-Ons](#)」を参照してください。

サービスアカウント名

サービスアカウントはこのアドオンでは使用されません。

AWS マネージド IAM ポリシー

このアドオンではマネージドポリシーは使用されません。

## カスタム IAM 許可

このアドオンではカスタム許可は使用されません。

## CLOUDSOFT

アドオン名は `cloudsoft_cloudsoft-amp` で、名前空間は `cloudsoft-amp` です。アドオンの公開は、CLOUDSOFT で行います。

アドオンの詳細についてはCLOUDSOFT ドキュメントの「[Amazon EKS ADDON](#)」を参照してください。

## サービスアカウント名

サービスアカウントはこのアドオンでは使用されません。

## AWS マネージド IAM ポリシー

このアドオンではマネージドポリシーは使用されません。

## カスタム IAM 許可

このアドオンではカスタム許可は使用されません。

## Cribl

アドオン名は `cribl_cribledge` で、名前空間は `cribledge` です。アドオンの公開は、Cribl で行います。

アドオンの詳細についてはCribl ドキュメントの「[Installing the Cribl Amazon EKS Add-on for Edge](#)」を参照してください。

## サービスアカウント名

サービスアカウントはこのアドオンでは使用されません。

## AWS マネージド IAM ポリシー

このアドオンではマネージドポリシーは使用されません。

## カスタム IAM 許可

このアドオンではカスタム許可は使用されません。

## Dynatrace

アドオン名は `dynatrace_dynatrace-operator` で、名前空間は `dynatrace` です。アドオンの公開は、Dynatrace で行います。

アドオンの詳細については、Dynatrace ドキュメントの「[Kubernetes のオブザーバビリティと監視](#)」を参照してください。

### サービスアカウント名

サービスアカウントはこのアドオンでは使用されません。

### AWS マネージド IAM ポリシー

このアドオンではマネージドポリシーは使用されません。

### カスタム IAM 許可

このアドオンではカスタム許可は使用されません。

## Datree

アドオン名は `datree_engine-pro` で、名前空間は `datree` です。アドオンの公開は、Datree で行います。

アドオンの詳細についてはDatree ドキュメントの「[Amazon EKS-intergration](#)」を参照してください。

### サービスアカウント名

サービスアカウント名は`datree-webhook-server-awssmp` です。

### AWS マネージド IAM ポリシー

マネージドポリシーは`AWSLicenseManagerConsumptionPolicy` です。詳細については「AWS マネージドポリシーリファレンスガイド」の「[AWSLicenseManagerConsumptionPolicy](#)」を参照してください。

### 必要な IAM 役割を作成するコマンド

次のコマンドではクラスターの IAM OpenID Connect (OIDC) プロバイダーが必要です。既に存在しているかどうかを確認する、または作成するには「[the section called “IAM OIDC プロバイダー”](#)」を

参照してください。##### をクラスターの名前に置き換え、*my-datree-role* を役割の名前に置き換えます。このコマンドを使用するにはデバイスに [eksctl](#) がインストールされている必要があります。別のツールを使用してロールを作成し、Kubernetes サービスアカウントに注釈を付ける必要がある場合は「[the section called “IAM ロールを割り当てる”](#)」を参照してください。

```
eksctl create iamserviceaccount --name datree-webhook-server-awsmvp --namespace datree
--cluster my-cluster --role-name my-datree-role \
 --role-only --attach-policy-arn arn:aws:iam::aws:policy/service-role/
AWSLicenseManagerConsumptionPolicy --approve
```

## カスタム権限

このアドオンではカスタム許可は使用されません。

## Datadog

アドオン名は `datadog_operator` で、名前空間は `datadog-agent` です。アドオンの公開は、Datadogで行います。

アドオンの詳細についてはDatadog ドキュメントの「[Datadog オペレーターアドオンを使用した Amazon EKS への Datadog エージェントのインストール](#)」を参照してください。

## サービスアカウント名

サービスアカウントはこのアドオンでは使用されません。

## AWS マネージド IAM ポリシー

このアドオンではマネージドポリシーは使用されません。

## カスタム IAM 許可

このアドオンではカスタム許可は使用されません。

## Groundcover

アドオン名は `groundcover_agent` で、名前空間は `groundcover` です。アドオンの公開は、Groundcoverで行います。

アドオンの詳細についてはgroundcover ドキュメントの「[グランドカバー Amazon EKS アドオンのインストール](#)」を参照してください。

## サービスアカウント名

サービスアカウントはこのアドオンでは使用されません。

## AWS マネージド IAM ポリシー

このアドオンではマネージドポリシーは使用されません。

## カスタム IAM 許可

このアドオンではカスタム許可は使用されません。

## Grafana Labs

アドオン名は grafana-labs\_kubernetes-monitoring で、名前空間は monitoring です。アドオンの公開は、Grafana Labs で行います。

アドオンの詳細についてはGrafana Labs ドキュメントの「[Amazon EKS でKubernetes 監視をアドオンとして設定する](#)」を参照してください。

## サービスアカウント名

サービスアカウントはこのアドオンでは使用されません。

## AWS マネージド IAM ポリシー

このアドオンではマネージドポリシーは使用されません。

## カスタム IAM 許可

このアドオンではカスタム許可は使用されません。

## Guance

- パブリッシャー – Guance
- 名前 – guance\_datakit
- 名前空間 – datakit
- サービスアカウント名 – サービスアカウントはこのアドオンでは使用されません。
- AWS マネージド IAM ポリシー – このアドオンではマネージドポリシーは使用されません。
- カスタム IAM アクセス許可 – このアドオンではカスタムアクセス許可は使用されません。
- セットアップと使用に関する指示 - Guance ドキュメントの「[Using Amazon EKS add-on](#)」を参照してください。



## HA Proxy

名前は haproxy-technologies\_kubernetes-ingress-ee で、名前空間は haproxy-controller です。アドオンの公開は、HA Proxyで行います。

アドオンの詳細については Datree ドキュメントの「[Amazon EKS-intergration](#)」を参照してください。

### サービスアカウント名

サービスアカウント名は customer defined です。

### AWS マネージド IAM ポリシー

マネージドポリシーは AWSLicenseManagerConsumptionPolicy です。詳細については「AWS マネージドポリシーリファレンスガイド」の「[AWSLicenseManagerConsumptionPolicy](#)」を参照してください。

### 必要な IAM 役割を作成するコマンド

次のコマンドではクラスターの IAM OpenID Connect (OIDC) プロバイダーが必要です。既に存在しているかどうかを確認する、または作成するには「[the section called “IAM OIDC プロバイダー”](#)」を参照してください。##### をクラスターの名前に置き換え、*my-haproxy-role* を役割の名前に置き換えます。このコマンドを使用するにはデバイスに [eksctl](#) がインストールされている必要があります。別のツールを使用してロールを作成し、Kubernetes サービスアカウントに注釈を付ける必要がある場合は「[the section called “IAM ロールを割り当てる”](#)」を参照してください。

```
eksctl create iamserviceaccount --name service-account-name --namespace haproxy-controller --cluster my-cluster --role-name my-haproxy-role \
 --role-only --attach-policy-arn arn:aws:iam::aws:policy/service-role/AWSLicenseManagerConsumptionPolicy --approve
```

### カスタム権限

このアドオンではカスタム許可は使用されません。

## Kpow

アドオン名は factorhouse\_kpow で、名前空間は factorhouse です。アドオンの公開は、Factorhouseで行います。

アドオンの詳細については、Kpow ドキュメントの「[AWS Marketplace LM](#)」を参照してください。

## サービスアカウント名

サービスアカウント名は kpow です。

## AWS マネージド IAM ポリシー

マネージドポリシーはAWSLicenseManagerConsumptionPolicy です。詳細については「AWS マネージドポリシーリファレンスガイド」の「[AWSLicenseManagerConsumptionPolicy](#)」を参照してください。

## 必要な IAM 役割を作成するコマンド

次のコマンドではクラスターの IAM OpenID Connect (OIDC) プロバイダーが必要です。既に存在しているかどうかを確認する、または作成するには「[the section called “IAM OIDC プロバイダー”](#)」を参照してください。##### をクラスターの名前に置き換え、*my-role-name* を IAM 役割の名前に置き換えます。このコマンドを使用するにはデバイスに [eksctl](#) がインストールされている必要があります。別のツールを使用してロールを作成し、Kubernetes サービスアカウントに注釈を付ける必要がある場合は「[the section called “IAM ロールを割り当てる”](#)」を参照してください。

```
eksctl create iamserviceaccount --name kpow --namespace factorhouse --cluster my-cluster --role-name my-kpow-role \
 --role-only --attach-policy-arn arn:aws:iam::aws:policy/service-role/AWSLicenseManagerConsumptionPolicy --approve
```

## カスタム権限

このアドオンではカスタム許可は使用されません。

## Kubecost

アドオン名は kubecost\_kubecost で、名前空間は kubecost です。アドオンの公開は、Kubecost で行います。

アドオンの詳細については、Kubecost ドキュメントの「[AWS Cloud Billing Integration](#)」を参照してください。

クラスターに「[Amazon EBS で Kubernetes ボリュームを保存する](#)」がインストールされている必要があります。インストールされていないと、エラーが発生します。

## サービスアカウント名

サービスアカウントはこのアドオンでは使用されません。

## AWS マネージド IAM ポリシー

このアドオンではマネージドポリシーは使用されません。

## カスタム IAM 許可

このアドオンではカスタム許可は使用されません。

## Kasten

アドオン名は `kasten_k10` で、名前空間は `kasten-io` です。アドオンの公開は、Kasten by Veeam で行います。

アドオンの詳細については Kasten ドキュメントの「Amazon EKS アドオンを使用したAWSへの K10 のインストール」を参照してください。

デフォルトが StorageClass のクラスターに Amazon EBS CSI ドライバーがインストールされている必要があります。

## サービスアカウント名

サービスアカウント名は `k10-k10` です。

## AWS マネージド IAM ポリシー

マネージドポリシーは `AWSLicenseManagerConsumptionPolicy` です。詳細については「AWS マネージドポリシーリファレンスガイド」の「[AWSLicenseManagerConsumptionPolicy](#)」を参照してください。

## 必要な IAM 役割を作成するコマンド

次のコマンドではクラスターの IAM OpenID Connect (OIDC) プロバイダーが必要です。既に存在しているかどうかを確認する、または作成するには「[the section called “IAM OIDC プロバイダー”](#)」を参照してください。##### をクラスターの名前に置き換え、`my-kasten-role` を役割の名前に置き換えます。このコマンドを使用するにはデバイスに `eksctl` がインストールされている必要があります。別のツールを使用してロールを作成し、Kubernetes サービスアカウントに注釈を付ける必要がある場合は「[the section called “IAM ロールを割り当てる”](#)」を参照してください。

```
eksctl create iamserviceaccount --name k10-k10 --namespace kasten-io --cluster my-cluster --role-name my-kasten-role \
 --role-only --attach-policy-arn arn:aws:iam::aws:policy/service-role/AWSLicenseManagerConsumptionPolicy --approve
```

## カスタム権限

このアドオンではカスタム許可は使用されません。

## Kong

アドオン名は `kong_konnect-ri` で、名前空間は `kong` です。アドオンの公開は、Kong で行います。

アドオンの詳細についてはKong ドキュメントの「[Installing the Kong Gateway EKS Add-on](#)」を参照してください。

クラスターに「[Amazon EBS で Kubernetes ボリュームを保存する](#)」がインストールされている必要があります。インストールされていないと、エラーが発生します。

### サービスアカウント名

サービスアカウントはこのアドオンでは使用されません。

### AWS マネージド IAM ポリシー

このアドオンではマネージドポリシーは使用されません。

### カスタム IAM 許可

このアドオンではカスタム許可は使用されません。

## LeakSignal

アドオン名は `leaksignal_leakagent` で、名前空間は `leakagent` です。アドオンの公開は、LeakSignal で行います。

アドオンの詳細についてはLeakSignal ドキュメントの <https://www.leaksignal.com/docs/LeakAgent/Deployment/> AWS%20EKS%20Addon/[Install the LeakAgent add-on] を参照してください。

クラスターに「[Amazon EBS で Kubernetes ボリュームを保存する](#)」がインストールされている必要があります。インストールされていないと、エラーが発生します。

### サービスアカウント名

サービスアカウントはこのアドオンでは使用されません。

## AWS マネージド IAM ポリシー

このアドオンではマネージドポリシーは使用されません。

## カスタム IAM 許可

このアドオンではカスタム許可は使用されません。

## NetApp

アドオン名は `netapp_trident-operator` で、名前空間は `trident` です。アドオンの公開は、NetApp で行います。

アドオンの詳細についてはNetApp ドキュメントの「[Configure the Trident EKS add-on](#)」を参照してください。

## サービスアカウント名

サービスアカウントはこのアドオンでは使用されません。

## AWS マネージド IAM ポリシー

このアドオンではマネージドポリシーは使用されません。

## カスタム IAM 許可

このアドオンではカスタム許可は使用されません。

## New Relic

アドオン名は `new-relic_kubernetes-operator` で、名前空間は `newrelic` です。アドオンの公開は、New Relic で行います。

アドオンの詳細についてはNew Relic ドキュメントの「[Installing the New Relic Add-on for EKS](#)」を参照してください。

## サービスアカウント名

サービスアカウントはこのアドオンでは使用されません。

## AWS マネージド IAM ポリシー

このアドオンではマネージドポリシーは使用されません。

## カスタム IAM 許可

このアドオンではカスタム許可は使用されません。

## Rafay

アドオン名は `rafay-systems_rafay-operator` で、名前空間は `rafay-system` です。アドオンの公開は、Rafay で行います。

アドオンの詳細についてはRafay ドキュメントの「[Installing the Rafay Amazon EKS Add-on](#)」を参照してください。

## サービスアカウント名

サービスアカウントはこのアドオンでは使用されません。

## AWS マネージド IAM ポリシー

このアドオンではマネージドポリシーは使用されません。

## カスタム IAM 許可

このアドオンではカスタム許可は使用されません。

## Rad Security

- パブリッシャー – RAD SECURITY
- 名前 – `rad-security_rad-security`
- 名前空間 – `ksoc`
- サービスアカウント名 – サービスアカウントはこのアドオンでは使用されません。
- AWS マネージド IAM ポリシー – このアドオンではマネージドポリシーは使用されません。
- カスタム IAM アクセス許可 – このアドオンではカスタムアクセス許可は使用されません。
- セットアップと使用に関する指示 – Rad Security ドキュメントの「[AWS マーケットプレイス](#)経由でのRADのインストール」を参照してください。

## SolarWinds

- パブリッシャー – SOLARWINDS

- 名前 – solarwinds\_swo-k8s-collector-addon
- 名前空間 – solarwinds
- サービスアカウント名 – サービスアカウントはこのアドオンでは使用されません。
- AWS マネージド IAM ポリシー – このアドオンではマネージドポリシーは使用されません。
- カスタム IAM アクセス許可 – このアドオンではカスタムアクセス許可は使用されません。
- セットアップと使用に関する指示 – SolarWinds ドキュメントの「[Amazon EKS クラスターを監視する](#)」を参照してください。

## Solo

アドオン名は solo-io\_istio-distro で、名前空間は istio-system です。アドオンの公開は、Solo で行います。

アドオンの詳細については Solo.io ドキュメントの「[Istio のインストール](#)」を参照してください。

### サービスアカウント名

サービスアカウントはこのアドオンでは使用されません。

### AWS マネージド IAM ポリシー

このアドオンではマネージドポリシーは使用されません。

### カスタム IAM 許可

このアドオンではカスタム許可は使用されません。

## Snyk

- パブリッシャー – SNYK
- 名前 – snyk\_runtime-sensor
- 名前空間 – snyk\_runtime-sensor
- サービスアカウント名 – サービスアカウントはこのアドオンでは使用されません。
- AWS マネージド IAM ポリシー – このアドオンではマネージドポリシーは使用されません。
- カスタム IAM アクセス許可 – このアドオンではカスタムアクセス許可は使用されません。
- セットアップと使用に関する指示 – Snyk ユーザードキュメントの「[Snyk runtime sensor](#)」を参照してください。

## Stormforge

アドオン名は `stormforge_optimize-Live` で、名前空間は `stormforge-system` です。アドオンの公開は、Stormforge で行います。

アドオンの詳細についてはStormForge ドキュメントの「[StormForgeエージェントのインストール](#)」を参照してください。

### サービスアカウント名

サービスアカウントはこのアドオンでは使用されません。

### AWS マネージド IAM ポリシー

このアドオンではマネージドポリシーは使用されません。

### カスタム IAM 許可

このアドオンではカスタム許可は使用されません。

## Splunk

アドオン名は `splunk_splunk-otel-collector-chart` で、名前空間は `splunk-monitoring` です。アドオンの公開は、Splunk で行います。

アドオンの詳細についてはSplunk ドキュメントの「[Amazon EKS 用 Splunk アドオンをインストールする](#)」を参照してください。

### サービスアカウント名

サービスアカウントはこのアドオンでは使用されません。

### AWS マネージド IAM ポリシー

このアドオンではマネージドポリシーは使用されません。

### カスタム IAM 許可

このアドオンではカスタム許可は使用されません。

## Teleport

アドオン名は `teleport_teleport` で、名前空間は `teleport` です。アドオンの公開は、Teleport で行います。



アドオンの詳細については、Teleport ドキュメントの「[How Teleport Works](#)」を参照してください。

### サービスアカウント名

サービスアカウントはこのアドオンでは使用されません。

### AWS マネージド IAM ポリシー

このアドオンではマネージドポリシーは使用されません。

### カスタム IAM 許可

このアドオンではカスタム許可は使用されません。

## Tetrate

アドオン名は `tetrate-io_istio-distro` で、名前空間は `istio-system` です。アドオンの公開は、Tetrate で行います。

アドオンの詳細についてはウェブサイトの「[テトレート・イスティ・ディオストロ](#)」を参照してください。

### サービスアカウント名

サービスアカウントはこのアドオンでは使用されません。

### AWS マネージド IAM ポリシー

このアドオンではマネージドポリシーは使用されません。

### カスタム IAM 許可

このアドオンではカスタム許可は使用されません。

## Upbound Universal Crossplane

アドオン名は `upbound_universal-crossplane` で、名前空間は `upbound-system` です。アドオンの公開は、Upbound で行います。

アドオンの詳細についてはUpbound ドキュメントの「[Upbound Universal Crossplane \(UXP\)](#)」を参照してください。

## サービスアカウント名

サービスアカウントはこのアドオンでは使用されません。

## AWS マネージド IAM ポリシー

このアドオンではマネージドポリシーは使用されません。

## カスタム IAM 許可

このアドオンではカスタム許可は使用されません。

## Upwind

アドオン名は upwind で、名前空間は upwind です。アドオンの公開は、Upwind で行います。

アドオンの詳細については[Upwind のドキュメント](#)を参照してください。

## サービスアカウント名

サービスアカウントはこのアドオンでは使用されません。

## AWS マネージド IAM ポリシー

このアドオンではマネージドポリシーは使用されません。

## カスタム IAM 許可

このアドオンではカスタム許可は使用されません。

## Amazon EKS アドオンを作成する

Amazon EKS アドオンはAmazon EKS クラスター用のアドオンソフトウェアです。すべての Amazon EKS アドオン:

- 最新のセキュリティパッチやバグ修正が含まれています。
- AWS によって Amazon EKS で動作することが検証されています。
- アドオンソフトウェアの管理に必要な作業量を削減します。

アマゾン EKS アドオンは eksctl、AWS Management Console、AWS CLI を使用して作成できます。アドオンに IAM ロールが必要な場合、ロールの作成の詳細については [Amazon EKS アドオンの特定のアドオンの詳細](#)を参照してください。

## 前提条件

アドオンを作成する前に、次を完了します:

- クラスターのアドオンを作成する前に、そのクラスターが存在している必要があります。詳細については「[the section called “クラスターを作成する”](#)」を参照してください。
- アドオンに IAM ロールが必要かどうかを確認します。詳細については「[the section called “互換性を確認する”](#)」を参照してください。
- Amazon EKS アドオンバージョンがクラスターと互換性があることを検証します。詳細については「[the section called “互換性を確認する”](#)」を参照してください。
- 使用しているコンピュータまたは AWS クラウドシェル に、eksctl コマンドラインツールのバージョン 0.190.0 以降がインストールされていることを確認します。詳細についてはeksctl ウェブサイトの「[インストール](#)」を参照してください。

## 手順

アマゾン EKS アドオンは eksctl、AWS Management Console、AWS CLI を使用して作成できます。アドオンに IAM ロールが必要な場合、ロールの作成の詳細については「[使用可能な AWS の アマゾン EKS アドオン](#)」の特定のアドオンの詳細を参照してください。

### アドオンの作成 (eksctl)

1. クラスターバージョンで利用可能なアドオンの名前を表示します。[1.32](#) をお持ちのクラスターのバージョンに置き換えます。

```
eksctl utils describe-addon-versions --kubernetes-version 1.32 | grep AddonName
```

出力例は次のとおりです。

```
"AddonName": "aws-ebs-csi-driver",
 "AddonName": "coredns",
 "AddonName": "kube-proxy",
 "AddonName": "vpc-cni",
 "AddonName": "adot",
 "AddonName": "dynatrace_dynatrace-operator",
 "AddonName": "upbound_universal-crossplane",
 "AddonName": "teleport_teleport",
 "AddonName": "factorhouse_kpow",
```

[...]

2. 作成するアドオンで利用できるバージョンを表示します。**1.32** をお持ちのクラスターのバージョンに置き換えます。***name-of-addon*** を表示するバージョンのアドオン名に置き換えます。名前は前のステップで返された名前のいずれかである必要があります。

```
eksctl utils describe-addon-versions --kubernetes-version 1.32 --name name-of-addon |
grep AddonVersion
```

次の出力は `vpc-cni` という名前のアドオンに対して返される内容の例です。このアドオンには複数のバージョンがあることがわかります。

```
"AddonVersions": [
 "AddonVersion": "v1.12.0-eksbuild.1",
 "AddonVersion": "v1.11.4-eksbuild.1",
 "AddonVersion": "v1.10.4-eksbuild.1",
 "AddonVersion": "v1.9.3-eksbuild.1",
```

- a. 作成するアドオンが Amazon EKS か AWS マーケットプレイス アドオンか判断します。AWS マーケットプレイス にはサードパーティ製のアドオンがあり、アドオンを作成するために追加のステップを実行する必要があります。

```
eksctl utils describe-addon-versions --kubernetes-version 1.32 --name name-of-
addon | grep ProductUrl
```

出力が返されない場合、アドオンは Amazon EKS です。出力が返された場合、アドオンは AWS マーケットプレイス アドオンです。次の出力は `teleport_teleport` という名前のアドオンのものです。

```
"ProductUrl": "https://aws.amazon.com/marketplace/pp?sku=3bda70bb-566f-4976-806c-
f96faef18b26"
```

返される URL を含んだ AWS マーケットプレイス のアドオンの詳細について説明します。アドオンにサブスクリプションが必要な場合、AWS マーケットプレイス を通じてアドオンにサブスクライブできます。AWS マーケットプレイス からアドオンを作成する場合、アドオンの作成に使用する [IAM プリンシパル](#) には [AWSServiceRoleForAWSライセンスマネージャーRole](#) のサービスにリンクされたロールを作成する許可が必要です。IAM エンティティへの許可の割

り当てに関する詳細については「IAM ユーザーガイド」の「[IAM アイデンティティ許可の追加と削除](#)」を参照してください。

3. アマゾン EKS アドオンを作成します。コマンドをコピーし、次のように *user-data* を置き換えます:

- ##### の部分は自分のクラスター名に置き換えます。
- ##### を作成するアドオンの名前に置き換えます。
- 最新バージョンよりも前のバージョンのアドオンが必要な場合、## を使用する前のステップの出力で返されたバージョン番号に置き換えます。
- アドオンがサービスアカウントロールを使用している場合、111122223333 をアカウント ID に置き換え、role-name をロールの名前に置き換えます。サービスアカウントのロールを作成する手順については作成するアドオンのドキュメントを参照してください。アドオンのリストについては「[the section called “AWSアドオン”](#)」を参照してください。サービスアカウント役割を指定するにはクラスター用に IAM OpenID Connect (OIDC) プロバイダーが必要です。クラスター用に持っているかどうかを確認、あるいは作成するには「[the section called “IAM OIDC プロバイダー”](#)」を参照してください。

アドオンがサービスアカウントロールを使用していない場合、--service-account-role-arn:aws:iam::111122223333:role/role-name を削除してください。

- このコマンド例はアドオンの既存のセルフマネージドバージョンがある場合、その設定を上書きします。既存のセルフマネージド型アドオンの設定を上書きしたくない場合、--force オプションを削除してください。オプションを削除し、アマゾン EKS アドオンが既存のセルフマネージド型アドオンの設定を上書きする必要がある場合、アマゾン EKS アドオンの作成が失敗します。その場合、競合を解決するためのエラーメッセージが表示されます。このオプションを指定する前に、自分が管理する必要がある設定をアマゾン EKS アドオンが管理していないことを確認してください。これらの設定はこのオプションの指定により上書きされます。

```
eksctl create addon --cluster my-cluster --name name-of-addon --version latest \
 --service-account-role-arn arn:aws:iam::111122223333:role/role-name --force
```

コマンドにすべての利用可能なオプションのリストを確認できます。

```
eksctl create addon --help
```

利用可能なオプションの詳細については「eksctl ドキュメント」の「[アドオン](#)」を参照してください。

## アドオンの作成 (AWS コンソール)

1. [アマゾン EKS コンソール](#)を開きます。
2. 左のナビゲーションペインで [クラスター] を選択してください。
3. アドオンを作成するクラスターの名前を選択してください。
4. [アドオン] タブを選択してください。
5. [その他のアドオンを入手] を選択してください。
6. [アドオンの選択] ページで、クラスターに追加するアドオンを選択してください。[Amazon EKS アドオン] と [AWS マーケットプレイス アドオン] は必要な数だけ追加できます。

AWS マーケットプレイス アドオンの場合、アドオンを作成するために使用する [IAM プリンシパル](#)にはAWS ライセンスマネージャー からアドオンの資格を読み取る許可が必要です。AWS ライセンスマネージャー はユーザーに代わってライセンスを管理する AWS リソースを許可する「[AWSServiceRoleForAWSライセンスマネージャーRole](#)」のサービスリンクロール (SLR が必要になります。SLR はアカウントごとに 1 回限りの要件であり、アドオンやクラスターごとに別々の SLR を作成する必要はありません。[IAM プリンシパル](#)への許可の割り当ての詳細については「IAM ユーザーガイド」の「[IAM ID のアクセス許可の追加および削除](#)」を参照してください。

インストールする AWS マーケットプレイス アドオンがリストされていない場合はページ番号をクリックして追加のページ結果を表示するか、または検索ボックスで検索できます。[フィルタリング] オプションで、[カテゴリ]、[ベンダー]、または [料金モデル] によってフィルタリングして、検索結果からアドオンを選択することもできます。インストールするアドオンを選択したら、[次へ] を選択してください。

7. [選択したアドオンセッティングの設定] ページで、次の操作を行います：
  - a. [サブスクリプションオプションを表示] を選択し、[サブスクリプションのオプション] フォームを開きます。[料金の詳細] と [法務] のセクションを確認し、[登録] ボタンを選択して続行します。
  - b. [バージョン] で、インストールするバージョンを選択してください。作成する個々のアドオンが別のバージョンを推奨している場合を除き、[最新] と表示されたバージョンをお勧めします。アドオンに推奨バージョンがあるかどうかを確認するには作成しているアドオンのドキュメントを参照してください。アドオンのリストについては「[the section called “AWS アドオン”](#)」を参照してください。
  - c. アドオンの役割を設定するにはEKS ポッド・アイデンティティー IAM 役割とサービスアカウント用の IAM 役割 (IRSA) の 2 つのオプションがあります。希望するオプションについては以下の適切なステップに従います。選択したすべてのアドオンの [ステータス] で [サブスクリプションが必要] と表示されている場合、[次へ] を選択してください。クラスターが作成された後

にサブスクライブするまで、それ以上これらのアドオンを設定することはできません。[ステータス] で [サブスクリプションが必要] と表示されていないアドオンの場合、次を実行してください：

- i. [サービスアカウント用の ポッド・アイデンティティ IAM 役割] で、既存の EKS Pod Identity IAM 役割を使用するか、または [推奨役割を作成] ボタンを使用して EKS Pod Identity IAM 役割を作成できます。このフィールドには適切な信頼ポリシーを持つオプションのみが表示されます。選択する役割がない場合は一致する信頼ポリシーを持つ既存の役割が存在しないことになります。選択したアドオンのサービスアカウント用の EKS Pod Identity IAM 役割を設定するには[推奨役割を作成] を選択してください。役割作成ウィザードが別のウィンドウで開きます。ウィザードは次のように役割の情報を自動的に入力してください。EKS Pod Identity IAM 役割を作成するアドオンごとに、IAM ウィザードのステップを次のように実行してください。
  - [信頼されたエンティティを選択] ステップでは[EKS] の AWS サービスオプションと [EKS - Pod Identity] のユースケースが事前に選択されており、アドオンのために適切な信頼ポリシーが自動的に入力されます。例えば、役割は「[the section called “EKS Pod Identity の利点”](#)」で詳述されているように、pods.eks.amazonaws.com の IAM プリンシパルを含む適切な信頼ポリシーを使用して作成されます。[次へ] を選択してください。
  - [許可を追加] ステップでは役割ポリシー用の適切なマネージドポリシーがアドオンのために事前に選択されています。例えば、Amazon VPC CNI アドオンの場合、役割は「[the section called “Amazon VPC CNI Kubernetes用プラグイン”](#)」で詳述されているように、マネージドポリシー AmazonEKS\_CNI\_Policy を使用して作成されます。[Next] を選択します。
  - [名前、確認、作成] のステップにおいて、[役割名] で、アドオンのためにデフォルトの役割名が自動的に入力されます。例えば、アマゾン VPC CNI アドオンの場合、役割は アマゾンEKSPodIdentityアマゾンVPCCNIRole という名前で作成されます。[説明] ではアドオンのためにデフォルトの適切な説明が自動的に入力されます。例えば、アマゾン VPC CNI アドオンの場合、役割は AWS リソースにアクセスすることを「アマゾン EKS クラスターで実行されているポッドに許可します」という説明で作成されます。[信頼ポリシー] で、アドオンのために事前入力された信頼ポリシーを表示します。[ロールの作成] を選択してください。

注: デフォルトのロール名を保持すると、EKS は新しいクラスターでアドオン用のロールを事前に選択したり、既存のクラスターにアドオンを追加したりできます。この名前は引き続き上書きでき、ロールはクラスター全体でアドオンのために使用できますが、ロールはドロップダウンから手動で選択する必要があります。



- ii. [ステータス] が [サブスクリプションが必要] でなく、IRSA を使用して役割を設定するアドオンについてはIAM ポリシーを作成して役割にアタッチするために作成するアドオンに関するドキュメントを参照してください。アドオンのリストについては「[the section called “AWSアドオン”](#)」を参照してください。IAM ロールを選択するにはクラスター用に OpenID Connect (OIDC) プロバイダーが必要です。クラスター用に持っているかどうかを確認、あるいは作成するには「[the section called “IAM OIDC プロバイダー”](#)」を参照してください。
  - iii. [オプション設定のセッティング] を選択してください。
  - iv. アドオンに設定が必要な場合、[設定値] ボックスに入力します。アドオンに設定情報が必要かどうかを判断するには作成するアドオンのドキュメントを参照してください。アドオンのリストについては「[the section called “AWSアドオン”](#)」を参照してください。
  - v. [競合解決方法] で利用可能なオプションをいずれか選択してください。[コンフリクト解決方法] で [上書き] を選択すると、既存のアドオンでの 1 つ以上の設定が、Amazon EKS アドオンの設定で上書きされます。このオプションが有効でない状態で既存の設定との競合が発生する場合はオペレーションが失敗します。表示されたエラーメッセージを使用して、競合をトラブルシューティングできます。このオプションを選択する前に、自分で管理する必要のある設定が、Amazon EKS アドオンにより管理されていないことを確認します。
  - vi. [次へ] を選択してください。
8. [確認と追加] ページで、[作成] を選択してください。アドオンのインストールが完了した後、インストールしたアドオンが表示されます。
9. インストールしたアドオンにサブスクリプションが必要な場合、次のステップを実行してください：
- a. アドオンの右下隅にある [サブスクライブ] ボタンを選択してください。AWS マーケットプレイスのアドオンのページが表示されます。[製品概要] や [価格情報] など、アドオンに関する情報をお読みください。
  - b. アドオンページの右上にある [サブスクライブを続ける] ボタンを選択してください。
  - c. [利用規約] をよくお読みください。利用規約に同意する場合、[利用規約に同意] を選択してください。サブスクリプションの処理に数分かかることがあります。サブスクリプションの処理中、[Amazon EKS コンソールに戻る] ボタンがグレー表示されます。
  - d. サブスクリプションの処理が完了すると、[Amazon EKS コンソールに戻る] ボタンがグレー表示されなくなります。ボタンを選択し、クラスターの Amazon EKS コンソールの [アドオン] タブに戻ります。
  - e. サブスクライブしているアドオンについては[削除して再インストール] を選択し、次に [アドオンの再インストール] を選択してください。アドオンのインストールに数分かかることがあります。インストールが完了すると、アドオンを設定できます。



## アドオンの作成 (AWS CLI)

1. ご使用のデバイスまたは AWS クラウドシェル で、バージョン 2.12.3 以降、または AWS コマンドラインインターフェイス (AWS CLI のバージョン 1.27.160 以降が必要です。現在のバージョンを確認するには「`aws --version | cut -d / -f2 | cut -d ' ' -f1`」を参照してください。yum、apt-get、macOS 用の Homebrew などのパッケージマネージャーは、多くの場合 AWS CLI の最新バージョンより数バージョン古くなっています。最新バージョンをインストールするには「AWS コマンドラインインターフェイスユーザーガイド」の「[インストール](#)」および「[aws configure を使用したクイック設定](#)」を参照してください。AWS クラウドシェル にインストールされている AWS CLI バージョンも最新バージョンより数バージョン遅れることがあります。更新するには「AWS クラウドシェル ユーザーガイド」の「[ホームディレクトリへの AWS CLI のインストール](#)」を参照してください。
2. 利用可能なアドオンを確認します。利用可能なすべてのアドオン、種類、発行元を確認できます。AWS マーケットプレイスを介して利用可能なアドオンの URL も確認できます。[1.32](#) をお持ちのクラスターのバージョンに置き換えます。

```
aws eks describe-addon-versions --kubernetes-version 1.32 \
 --query 'addons[].[MarketplaceProductId: marketplaceInformation.productId,
 Name: addonName, Owner: owner Publisher: publisher, Type: type]' --output table
```

出力例は次のとおりです。

```

|
| DescribeAddonVersions
|
+-----+
+-----+-----+-----+
+-----+
| MarketplaceProductId | Name
| Owner | Publisher | Type |
+-----+
+-----+-----+-----+
+-----+
| None | aws | eks | storage | aws-ebs-csi-driver
| None | aws | eks | networking | coredns
| None | aws | eks | networking | kube-proxy
```

```

| None | aws | eks | networking | vpc-cni
| None | aws | eks | networking | adot
| https://aws.amazon.com/marketplace/pp/prodview-brb73nceicv7u |
dynatrace_dynatrace-operator | aws-marketplace | dynatrace | monitoring
| https://aws.amazon.com/marketplace/pp/prodview-uhc2iwi5xysoc | upbound_universal-
crossplane | aws-marketplace | upbound | infra-management |
| https://aws.amazon.com/marketplace/pp/prodview-hd2ydsrgqy4li | teleport_teleport
| aws-marketplace | teleport | policy-management |
| https://aws.amazon.com/marketplace/pp/prodview-vgghgqdsplhvc | factorhouse_kpow
| aws-marketplace | factorhouse | monitoring |
| [...] | [...] | [...] | [...] | [...]
+-----+
+-----+-----+-----+
+-----+

```

出力が異なる場合があります。この出力例ではタイプ `networking` で利用可能なアドオンが 3 種類あり、タイプ `eks` の発行元でアドオンが 5 種類あります。Owner 列に `aws-marketplace` があるアドオンはインストールする前にサブスクリプションが必要な場合があります。URL にアクセスしてアドオンの詳細を確認したり、アドオンをサブスクライブしたりできます。

3. アドオンごとに利用可能なバージョンを確認できます。`1.32` をクラスターのバージョンに置き換え、`vpc-cni` を前のステップで返されたアドオンの名前に置き換えます。

```

aws eks describe-addon-versions --kubernetes-version 1.32 --addon-name vpc-cni \
 --query 'addons[].addonVersions[].{Version: addonVersion, Defaultversion:
compatibilities[0].defaultVersion}' --output table

```

出力例は次のとおりです。

```

| DescribeAddonVersions |
+-----+-----+
| Defaultversion | Version |
+-----+-----+
False	v1.12.0-eksbuild.1
True	v1.11.4-eksbuild.1
False	v1.10.4-eksbuild.1
False	v1.9.3-eksbuild.1

```

```
+-----+-----+
```

Defaultversion 列に True と表示されているバージョンはデフォルトでアドオンが作成されたバージョンです。

#### 4. (オプション) 次のコマンドを実行して、選択したアドオンの設定オプションを調べます：

```
aws eks describe-addon-configuration --addon-name vpc-cni --addon-version v1.12.0-eksbuild.1
```

```
{
 "addonName": "vpc-cni",
 "addonVersion": "v1.12.0-eksbuild.1",
 "configurationSchema": "{ \"$ref\": \"#/definitions/VpcCni\", \"$schema\": \"http://json-schema.org/draft-06/schema#\", \"definitions\": { \"Cri\": { \"additionalProperties\": false, \"properties\": { \"hostPath\": { \"$ref\": \"#/definitions/HostPath\" } }, \"title\": \"Cri\", \"type\": \"object\" }, \"Env\": { \"additionalProperties\": false, \"properties\": { \"ADDITIONAL_ENI_TAGS\": { \"type\": \"string\" }, \"AWS_VPC_CNI_NODE_PORT_SUPPORT\": { \"format\": \"boolean\", \"type\": \"string\" }, \"AWS_VPC_ENI_MTU\": { \"format\": \"integer\", \"type\": \"string\" }, \"AWS_VPC_K8S_CNI_CONFIGURE_RPFILTER\": { \"format\": \"boolean\", \"type\": \"string\" }, \"AWS_VPC_K8S_CNI_CUSTOM_NETWORK_CFG\": { \"format\": \"boolean\", \"type\": \"string\" }, \"AWS_VPC_K8S_CNI_EXTERNALSNAT\": { \"format\": \"boolean\", \"type\": \"string\" }, \"AWS_VPC_K8S_CNI_LOGLEVEL\": { \"type\": \"string\" }, \"AWS_VPC_K8S_CNI_LOG_FILE\": { \"type\": \"string\" }, \"AWS_VPC_K8S_CNI_RANDOMIZESNAT\": { \"type\": \"string\" }, \"AWS_VPC_K8S_CNI_VETHPREFIX\": { \"type\": \"string\" }, \"AWS_VPC_K8S_PLUGIN_LOG_FILE\": { \"type\": \"string\" }, \"AWS_VPC_K8S_PLUGIN_LOG_LEVEL\": { \"type\": \"string\" }, \"DISABLE_INTROSPECTION\": { \"format\": \"boolean\", \"type\": \"string\" }, \"DISABLE_METRICS\": { \"format\": \"boolean\", \"type\": \"string\" }, \"DISABLE_NETWORK_RESOURCE_PROVISIONING\": { \"format\": \"boolean\", \"type\": \"string\" }, \"ENABLE_POD_ENI\": { \"format\": \"boolean\", \"type\": \"string\" }, \"ENABLE_PREFIX_DELEGATION\": { \"format\": \"boolean\", \"type\": \"string\" }, \"WARM_ENI_TARGET\": { \"format\": \"integer\", \"type\": \"string\" }, \"WARM_PREFIX_TARGET\": { \"format\": \"integer\", \"type\": \"string\" } }, \"title\": \"Env\", \"type\": \"object\" }, \"HostPath\": { \"additionalProperties\": false, \"properties\": { \"path\": { \"type\": \"string\" } }, \"title\": \"HostPath\", \"type\": \"object\" }, \"Limits\": { \"additionalProperties\": false, \"properties\": { \"cpu\": { \"type\": \"string\" }, \"memory\": { \"type\": \"string\" } }, \"title\": \"Limits\", \"type\": \"object\" }, \"Resources\": { \"additionalProperties\": false, \"properties\": { \"limits\": { \"$ref\": \"#/definitions/Limits\" }, \"requests\": { \"$ref\": \"#/definitions/Limits\" } }, \"title\": \"Resources\", \"type\": \"object\" }, \"VpcCni\": { \"additionalProperties\": false, \"properties\": { \"cri\": { \"$ref\": \"#/definitions/Cri\" }, \"env\": { \"$ref\": \"#/definitions/Env\" }, \"resources\": { \"$ref\": \"#/definitions/Resources\" } }, \"title\": \"VpcCni\", \"type\": \"object\" } } }
```

```
}
```

出力は標準の JSON スキーマです。

上記のスキーマで利用できる有効な設定値の例 (JSON フォーマット) を次に示します。

```
{
 "resources": {
 "limits": {
 "cpu": "100m"
 }
 }
}
```

上記のスキーマで利用できる有効な設定値の例 (YAML フォーマット) を次に示します。

```
resources:
 limits:
 cpu: 100m
```

5. アドオンに IAM アクセス許可が必要かどうかを確認します。その場合は(1 EKS Pod Identity またはサービスアカウントの IAM ロール (IRSA 使用するかどうかを決定し、(2 アドオンで使用する IAM ロールの ARN を決定し、(3) アドオンで使用する Kubernetes サービスアカウントの名前を決定する必要があります。詳細については「[the section called “IAM情報を取得する”](#)」を参照してください。
- Amazon EKS ではアドオンが EKS Pod Identity をサポートしている場合はEKS Pod Identity の使用を推奨しています。そのためには[Pod Identity エージェントがクラスターにインストールされている](#)必要があります。アドオンで Pod Identity を使用する場合の詳細については「[the section called “IAM ロール”](#)」を参照してください。
  - アドオンまたはクラスターが EKS Pod Identity 用にセットアップされていない場合はIRSA を使用します。[クラスターで IRSA が設定されていることを確認します。](#)
  - [Amazon EKS アドオンのドキュメントを参照して、アドオンに IAM アクセス許可と、関連する Kubernetes サービスアカウントの名前が必要かどうかを確認します。](#)
    - a. アマゾン EKS アドオンを作成します。デバイスに沿ったコマンドをコピーします。必要に応じてコマンドに次の変更を加え、変更したコマンドを実行してください：
      - **#####** の部分は自分のクラスター名に置き換えます。
      - **vpc-cni** を前のステップの出力で返された作成するアドオン名に置き換えます。

- `vpc-cni` を使用する前のステップの出力で返されたバージョンに置き換えます。
- アドオンに IAM アクセス許可が必要ない場合は `<service-account-configuration>` を削除してください。
- 次のいずれかを行います：
  - アドオンが (1) IAM 許可が必要で、(2) クラスターが EKS Pod Identity を使用している場合は `<service-account-configuration>` を次の Pod Identity の関連付けに置き換えます。 `<service-account-name>` をアドオンで使用されるサービスアカウント名に置き換えます。 `<role-arn>` を IAM ロールの ARN に置き換えます。ロールには EKS Pod Identity に必要な信頼ポリシーが必要です。

```
--pod-identity-associations 'serviceAccount=<service-account-name>,roleArn=<role-arn>'
```

- アドオンが (1) IAM アクセス許可が必要で、(2) クラスターが IRSA を使用している場合は `<service-account-configuration>` を次の IRSA 設定に置き換えます。 `111122223333` を、アカウント ID に置き換えます。また、 `role-name` を、作成した既存の IAM ロールの名前に置き換えます。ロールの作成手順については作成するアドオンのドキュメントを参照してください。アドオンのリストについては「[the section called “AWS アドオン”](#)」を参照してください。サービスアカウント役割を指定するにはクラスター用に IAM OpenID Connect (OIDC) プロバイダーが必要です。クラスター用に持っているかどうかを確認、あるいは作成するには「[the section called “IAM OIDC プロバイダー”](#)」を参照してください。

```
--service-account-role-arn arn:aws::iam::111122223333:role/role-name
```

- このコマンド例はアドオンの既存のセルフマネージドバージョンの `--configuration-values` オプションがある場合、それを上書きします。これを、文字列やファイル入力などの目的の設定値に置き換えます。設定値を指定したくない場合、`--configuration-values` オプションを削除します。AWS CLI に既存のセルフマネージド型アドオンの設定を上書きされたくない場合、`--resolve-conflicts OVERWRITE` オプションを削除してください。オプションを削除し、アマゾン EKS アドオンが既存のセルフマネージド型アドオンの設定を上書きする必要がある場合、アマゾン EKS アドオンの作成が失敗します。その場合、競合を解決するためのエラーメッセージが表示されます。このオプションを指定する前に、自分が管理する必要がある設定をアマゾン EKS アドオンが管理していないことを確認してください。これらの設定はこのオプションの指定により上書きされます。

```
aws eks create-addon --cluster-name my-cluster --addon-name vpc-cni --addon-version
version-number \
 <service-account-configuration> --configuration-values '{"resources":
{"limits":{"cpu":"100m"}}}' --resolve-conflicts OVERWRITE
```

```
aws eks create-addon --cluster-name my-cluster --addon-name vpc-cni --addon-version
version-number \
 <service-account-configuration> --configuration-values 'file://example.yaml' --
resolve-conflicts OVERWRITE
```

使用可能なすべてのオプションのリストについては「Amazon EKS コマンドラインリファレンス」の「[create-addon](#)」を参照してください。作成したアドオンが前のステップの Owner 列に aws-marketplace が一覧表示されている場合、作成が失敗して次のエラーと同様なエラーメッセージが表示されることがあります。

```
{
 "addon": {
 "addonName": "addon-name",
 "clusterName": "my-cluster",
 "status": "CREATE_FAILED",
 "addonVersion": "version",
 "health": {
 "issues": [
 {
 "code": "AddonSubscriptionNeeded",
 "message": "You are currently not subscribed to this add-on. To
subscribe, visit the AWS Marketplace console, agree to the seller EULA, select the
pricing type if required, then re-install the add-on"
 }
]
 }
 }
}
```

前の出力のエラーと同様なものが表示された場合、前のステップに出力された URL にアクセスしてアドオンをサブスクライブしてください。サブスクライブしたら create-addon コマンドを再実行してください。

## Amazon EKS アドオンを更新する

Amazon EKS は新しいバージョンがリリースされた場合、あるいはクラスターを新しい Kubernetes マイナーバージョンに更新しても、アドオンを自動的に更新しません。既存のクラスターのアドオンを更新するには更新を開始する必要があります。更新を開始した後、Amazon EKS がアドオンを更新します。アドオンを更新する前に、アドオンの現行のドキュメントを確認してください。利用可能なアドオンのリストについては「[the section called “AWS アドオン”](#)」を参照してください。アドオンに IAM 役割が必要な場合、役割の作成の詳細については「[使用可能な AWS の アマゾン EKS アドオン](#)」の特定のアドオンの詳細を参照してください。

### 前提条件

アドオンを作成する前に、次を完了します:

- アドオンに IAM 役割が必要かどうかを確認します。詳細については「[the section called “Amazon EKS アドオン”](#)」を参照してください。
- Amazon EKS アドオンバージョンがクラスターと互換性があることを検証します。詳細については「[the section called “互換性を確認する”](#)」を参照してください。

### 手順

eksctl、AWS Management Console、AWS CLI のいずれかを使用して Amazon EKS アドオンを更新できます。

#### アドオンの更新 (eksctl)

- クラスターにインストールされている現在のアドオンおよびアドオンのバージョンを確認します。**#####** の部分は自分のクラスター名に置き換えます。

```
eksctl get addon --cluster my-cluster
```

出力例は次のとおりです。

| NAME       | VERSION            | STATUS | ISSUES | IAMROLE | UPDATE AVAILABLE                                                                               |
|------------|--------------------|--------|--------|---------|------------------------------------------------------------------------------------------------|
| coredns    | v1.8.7-eksbuild.2  | ACTIVE | 0      |         |                                                                                                |
| kube-proxy | v1.23.7-eksbuild.1 | ACTIVE | 0      |         | v1.23.8-eksbuild.2                                                                             |
| vpc-cni    | v1.10.4-eksbuild.1 | ACTIVE | 0      |         | v1.12.0-eksbuild.1,v1.11.4-eksbuild.1,v1.11.3-eksbuild.1,v1.11.2-eksbuild.1,v1.11.0-eksbuild.1 |

クラスターにあるアドオンおよびバージョンによって、出力が異なる場合があります。前述の出力例ではクラスターの既存のアドオン 2 つは UPDATE AVAILABLE 列に新しいバージョンがあることがわかります。

## 2. アドオンを更新します。

a. デバイスに沿ったコマンドをコピーします。必要に応じてコマンドに次の変更を加えます。

- ##### の部分は自分のクラスター名に置き換えます。
- ##### を、クラスターのある AWS リージョンに置き換えます。
- `vpc-cni` を更新する前のステップの出力で返されたアドオンの名前に置き換えます。
- 利用可能な最新バージョンよりも前のバージョンに更新する場合、`latest` を使用する前のステップの出力で返されたバージョン番号に置き換えます。一部のアドオンには推奨バージョンがあります。詳細については更新するアドオンのドキュメントを参照してください。アドオンのリストについては「[the section called “AWS アドオン”](#)」を参照してください。\* アドオンが Kubernetes サービスアカウントおよび IAM ロールを使用する場合、`111122223333` をアカウント ID に置き換え、`role-name` を作成した既存の IAM ロールの名前に置き換えます。役割の作成手順については作成するアドオンのドキュメントを参照してください。アドオンのリストについては「[the section called “AWS アドオン”](#)」を参照してください。サービスアカウント役割を指定するにはクラスター用に IAM OpenID Connect (OIDC) プロバイダーが必要です。クラスター用に持っているかどうかを確認、あるいは作成するには「[the section called “IAM OIDC プロバイダー”](#)」を参照してください。

アドオンが Kubernetes サービスアカウントおよび IAM ロールを使用しない場合、`serviceAccountRoleARN: arn:aws:iam::111122223333:role/role-name` 行を削除してください。

- `##` オプションはアドオンの既存値を保存します。アドオン設定にカスタム値を設定していて、このオプションを使用しない場合、アマゾン EKS は値をデフォルト値で上書きします。このオプションを使用する場合、実稼働クラスターのアドオンを更新する前に、非稼働クラスターのフィールドおよび値変更をテストすることをお勧めします。この値を `overwrite` に変更する場合、すべての設定がアマゾン EKS のデフォルト値に変更されます。いずれかの設定にカスタム値を設定した場合、アマゾン EKS のデフォルト値で上書きされる可能性があります。この値を `none` に変更した場合、アマゾン EKS は設定の値を一切変更しませんが、更新が失敗する可能性があります。更新に失敗した場合、競合の解決に役立つエラーメッセージが返されます。

```
cat >update-addon.yaml <<EOF
apiVersion: eksctl.io/v1alpha5
```



```
kind: ClusterConfig
metadata:
 name: my-cluster
 region: region-code

addons:
- name: vpc-cni
 version: latest
 serviceAccountRoleARN: arn:aws:iam::111122223333:role/role-name
 resolveConflicts: preserve
EOF
```

- b. 変更コマンドを実行して update-addon.yaml ファイルを作成します。
- c. クラスターに設定ファイルを適用します。

```
eksctl update addon -f update-addon.yaml
```

アドオンの更新の詳細についてはeksctl ドキュメントの「[アドオンの更新](#)」を参照してください。

## アドオンの更新 (AWS コンソール)

1. [アマゾン EKS コンソール](#)を開きます。
2. 左のナビゲーションペインで [クラスター] を選択してください。
3. アドオンを更新するクラスターの名前を選択してください。
4. [アドオン] タブを選択してください。
5. 更新するアドオンを選択してください。
6. [編集] を選択してください。
7. [#####を設定] ページで、次を実行します:
  - a. 使用する [バージョン] を選択してください。アドオンには推奨バージョンがある場合があります。詳細については更新するアドオンのドキュメントを参照してください。アドオンのリストについては「[the section called “AWSアドオン”](#)」を参照してください。
  - b. アドオンの役割を設定するにはEKS ポッド・アイデンティティー IAM 役割とサービスアカウント用の IAM 役割 (IRSA) の 2 つのオプションがあります。希望するオプションについては以下の適切なステップに従います。選択したすべてのアドオンの [ステータス] で [サブスクリプションが必要] と表示されている場合、[次へ] を選択してください。[ステータス] で [サブスクリプションが必要] と表示されていないアドオンの場合、次を実行してください。

- i. [サービスアカウント用の ポッド・アイデンティティ IAM 役割] で、既存の EKS Pod Identity IAM 役割を使用するか、または [推奨役割を作成] ボタンを使用して EKS Pod Identity IAM 役割を作成できます。このフィールドには適切な信頼ポリシーを持つオプションのみが表示されます。選択する役割がない場合は一致する信頼ポリシーを持つ既存の役割が存在しないことになります。選択したアドオンのサービスアカウント用の EKS Pod Identity IAM 役割を設定するには[推奨役割を作成] を選択してください。役割作成ウィザードが別のウィンドウで開きます。ウィザードは次のように役割の情報を自動的に入力してください。EKS Pod Identity IAM 役割を作成するアドオンごとに、IAM ウィザードのステップを次のように実行してください。
- [信頼されたエンティティを選択] ステップでは[EKS] の AWS サービスオプションと [EKS - Pod Identity] のユースケースが事前に選択されており、アドオンのために適切な信頼ポリシーが自動的に入力されます。例えば、役割は「[the section called “EKS Pod Identity の利点”](#)」で詳述されているように、pods.eks.amazonaws.com の IAM プリンシパルを含む適切な信頼ポリシーを使用して作成されます。[次へ] を選択してください。
  - [許可を追加] ステップでは役割ポリシー用の適切なマネージドポリシーがアドオンのために事前に選択されています。例えば、Amazon VPC CNI アドオンの場合、役割は「[the section called “Amazon VPC CNI Kubernetes用プラグイン”](#)」で詳述されているように、マネージドポリシー AmazonEKS\_CNI\_Policy を使用して作成されます。[Next] を選択します。
  - [名前、確認、作成] のステップにおいて、[役割名] で、アドオンのためにデフォルトの役割名が自動的に入力されます。例えば、アマゾン VPC CNI アドオンの場合、役割は アマゾンEKSPodIdentityアマゾンVPCCNIRole という名前で作成されます。[説明] ではアドオンのためにデフォルトの適切な説明が自動的に入力されます。例えば、アマゾン VPC CNI アドオンの場合、役割は AWS リソースにアクセスすることを「アマゾン EKS クラスターで実行されているポッドに許可します」という説明で作成されます。[信頼ポリシー] で、アドオンのために事前入力された信頼ポリシーを表示します。[役割の作成] を選択してください。

 Note

デフォルトの役割名を保持すると、EKS は新しいクラスターでアドオン用の役割を事前に選択したり、既存のクラスターにアドオンを追加したりできます。この名前は引き続き上書きでき、役割はクラスター全体でアドオンのために使用できますが、役割はドロップダウンから手動で選択する必要があります。

- ii. [ステータス] が [サブスクリプションが必要] でなく、IRSA を使用して役割を設定するアドオンについては IAM ポリシーを作成して役割にアタッチするために作成するアドオンに関するドキュメントを参照してください。アドオンのリストについては「[the section called “AWS アドオン”](#)」を参照してください。IAM ロールを選択するにはクラスター用に OpenID Connect (OIDC) プロバイダーが必要です。クラスター用に持っているかどうかを確認、あるいは作成するには「[the section called “IAM OIDC プロバイダー”](#)」を参照してください。
  - c. [オプションの構成設定] を展開します。
  - d. [設定値] で、アドオン固有の設定情報を入力してください。詳細については更新するアドオンのドキュメントを参照してください。アドオンのリストについては「[the section called “AWS アドオン”](#)」を参照してください。[コンフリクト解決方法] で、いずれかのオプションを選択してください。アドオン設定にカスタム値を設定している場合、[保存] オプションをお勧めします。このオプションを選択しない場合、Amazon EKS は値をデフォルト値で上書きします。このオプションを使用する場合、実稼働クラスターのアドオンを更新する前に、非稼働クラスターのフィールドおよび値変更をテストすることをお勧めします。この値を [上書き] に変更する場合、すべての設定が Amazon EKS のデフォルト値に変更されます。いずれかの設定にカスタム値を設定した場合、アマゾン EKS のデフォルト値で上書きされる可能性があります。この値を [なし] に変更した場合、Amazon EKS は設定の値を一切変更しませんが、更新が失敗する可能性があります。更新に失敗した場合、競合の解決に役立つエラーメッセージが返されます。
8. [Save changes] (変更の保存) をクリックします。

## アドオンの更新 (AWS CLI)

1. ご使用のデバイスまたは AWS CloudShell で、バージョン 2.12.3 以降、または AWS コマンドラインインターフェイス (AWS CLI) のバージョン 1.27.160 以降が必要です。現在のバージョンを確認するには「`aws --version | cut -d / -f2 | cut -d ' ' -f1`」を参照してください。yum、apt-get、macOS 用の Homebrew などのパッケージマネージャーは、多くの場合 AWS CLI の最新バージョンより数バージョン古くなっています。最新バージョンをインストールするには「AWS コマンドラインインターフェイスユーザーガイド」の「[インストール](#)」および「[aws configure を使用したクイック設定](#)」を参照してください。AWS クラウドシェルにインストールされている AWS CLI バージョンも最新バージョンより数バージョン遅れることがあります。更新するには「AWS CloudShell ユーザーガイド」の「[ホームディレクトリへの AWS CLI のインストール](#)」を参照してください。
2. インストールされているアドオンのリストを参照してください。##### の部分は自分のクラスター名に置き換えます。

```
aws eks list-addons --cluster-name my-cluster
```

出力例は次のとおりです。

```
{
 "addons": [
 "coredns",
 "kube-proxy",
 "vpc-cni"
]
}
```

3. 更新するアドオンの現在のバージョンを表示します。**#####** をクラスターの名前に置き換えて、**vpc-cni** を更新するアドオンの名前に置き換えます。

```
aws eks describe-addon --cluster-name my-cluster --addon-name vpc-cni --query
"addon.addonVersion" --output text
```

出力例は次のとおりです。

```
v1.10.4-eksbuild.1
```

4. クラスターのバージョンで利用できるアドオンのバージョンを確認します。**1.32** をクラスターのバージョンに置き換えて、**vpc-cni** を更新するアドオンの名前に置き換えます。

```
aws eks describe-addon-versions --kubernetes-version 1.32 --addon-name vpc-cni \
 --query 'addons[].addonVersions[].{Version: addonVersion, Defaultversion:
compatibilities[0].defaultVersion}' --output table
```

出力例は次のとおりです。

```

| DescribeAddonVersions |
+-----+-----+
| Defaultversion | Version |
+-----+-----+
False	v1.12.0-eksbuild.1
True	v1.11.4-eksbuild.1
False	v1.10.4-eksbuild.1
+-----+-----+
```

```
| False | v1.9.3-eksbuild.1 |
+-----+-----+
```

Defaultversion 列に True と表示されているバージョンはデフォルトでアドオンが作成されたバージョンです。

5. アドオンを更新してください。デバイスに沿ったコマンドをコピーします。必要に応じてコマンドに次の変更を加え、変更したコマンドを実行してください。このコマンドの詳細については「Amazon EKS コマンドラインリファレンス」の「[update-addon](#)」を参照してください。
  - ##### の部分は自分のクラスター名に置き換えます。
  - *vpc-cni* を前のステップの出力で返された更新するアドオンの名前に置き換えます。
  - *version-number* を更新する前のステップの出力で返されたバージョンに置き換えます。一部のアドオンには推奨バージョンがあります。詳細については更新するアドオンのドキュメントを参照してください。アドオンのリストについては「[the section called “AWSアドオン”](#)」を参照してください。\* アドオンが Kubernetes サービスアカウントおよび IAM ロールを使用する場合、*111122223333* をアカウント ID に置き換え、*role-name* を作成した既存の IAM ロールの名前に置き換えます。役割の作成手順については作成するアドオンのドキュメントを参照してください。アドオンのリストについては「[the section called “AWSアドオン”](#)」を参照してください。サービスアカウント役割を指定するにはクラスター用に IAM OpenID Connect (OIDC) プロバイダーが必要です。クラスター用に持っているかどうかを確認、あるいは作成するには「[the section called “IAM OIDC プロバイダー”](#)」を参照してください。

アドオンが Kubernetes サービスアカウントおよび IAM ロールを使用しない場

合、serviceAccountRoleARN: arn:aws:iam::*111122223333*:role/*role-name* 行を削除してください。

- `--resolve-conflicts PRESERVE` オプションはアドオンの既存値を保存します。アドオン設定にカスタム値を設定していて、このオプションを使用しない場合、アマゾン EKS は値をデフォルト値で上書きします。このオプションを使用する場合、実稼働クラスターのアドオンを更新する前に、非稼働クラスターのフィールドおよび値変更をテストすることをお勧めします。この値を `OVERWRITE` に変更する場合、すべての設定がアマゾン EKS のデフォルト値に変更されます。いずれかの設定にカスタム値を設定した場合、アマゾン EKS のデフォルト値で上書きされる可能性があります。この値を `NONE` に変更した場合、アマゾン EKS は設定の値を一切変更しませんが、更新が失敗する可能性があります。更新に失敗した場合、競合の解決に役立つエラーメッセージが返されます。
- すべてのカスタム設定を削除する場合は `--configuration-values '{}'` オプションを使用して更新を実行してください。これにより、すべてのカスタム設定がデフォルト値に設定され

ます。カスタム設定を変更しない場合、`--configuration-values` フラグを指定しないでください。カスタム設定を調整する場合、`{}` を新しいパラメータに置き換えます。

```
aws eks update-addon --cluster-name my-cluster --addon-name vpc-cni --addon-version
version-number \
 --service-account-role-arn arn:aws:iam::111122223333:role/role-name --
configuration-values '{}' --resolve-conflicts PRESERVE
```

6. 更新のステータスを確認します。`#####` をクラスターの名前に置き換えて、`vpc-cni` を更新するアドオンの名前に置き換えます。

```
aws eks describe-addon --cluster-name my-cluster --addon-name vpc-cni
```

出力例は次のとおりです。

```
{
 "addon": {
 "addonName": "vpc-cni",
 "clusterName": "my-cluster",
 "status": "UPDATING",
 }
}
```

ステータスが ACTIVE に変わると更新が完了します。

## Amazon EKS アドオンバージョンのクラスターとの互換性を検証する

Amazon EKS アドオンを作成する前に、Amazon EKS アドオンのバージョンがクラスターと互換性があることを検証する必要があります。

[アドオンバージョンを記述する API](#) を使用して、EKS アドオンの利用可能なバージョンと、各アドオンバージョンがサポートしている Kubernetes バージョンを一覧表示します。

1. AWS CLI がインストールされ、`aws sts get-caller-identity` と連携していることを確認します。このコマンドが機能しない場合は「[AWS CLI の使用を開始する](#)」を参照してください。
2. バージョンの互換性情報を取得するアドオンの名前 (amazon-cloudwatch-observability など) を特定します。
3. クラスターの Kubernetes バージョンを特定します (1.32 など)。

4. AWS CLI を使用して、クラスターの Kubernetes バージョンと互換性のあるアドオンのバージョンを取得します。

```
aws eks describe-addon-versions --addon-name amazon-cloudwatch-observability --kubernetes-version 1.32
```

出力例は次のとおりです。

```
{
 "addons": [
 {
 "addonName": "amazon-cloudwatch-observability",
 "type": "observability",
 "addonVersions": [
 {
 "addonVersion": "vX.X.X-eksbuild.X",
 "architecture": [
 "amd64",
 "arm64"
],
 "computeTypes": [
 "ec2",
 "auto",
 "hybrid"
],
 "compatibilities": [
 {
 "clusterVersion": "1.32",
 "platformVersions": [
 "*"
],
 "defaultVersion": true
 }
],
 }
],
 }
]
}
```

この出力はアドオンのバージョン `vX.X.X-eksbuild.X` が、Kubernetes クラスターのバージョン 1.32 と互換性があることを示しています。



## コンピューティングタイプとのアドオン互換性

describe-addon-versions 出力の computeTypes フィールドはアドオンと EKS 自動モード I マネージドノードまたはハイブリッドノードとの互換性を示します。auto とマークされたアドオンはEKS 自動モード I のクラウドベース AWS マネージドインフラストラクチャで動作しますが、hybrid とマークされたアドオンはEKS クラウドコントロールプレーンに接続されたオンプレミスノードで実行できます。

詳細については「[the section called “Amazon EKS 自動モード I の考慮事項”](#)」を参照してください。

## クラスターから Amazon EKS アドオンを削除する

eksctl、AWS Management Console、AWS CLI を使用して Amazon EKS アドオンをクラスターから削除できます。

クラスターから Amazon EKS アドオンを削除する場合:

- アドオンが提供する機能にはダウンタイムはありません。
- サービスアカウントの IAM 役割 (IRSA) を使用していて、アドオンに IAM 役割が関連付けられている場合、IAM 役割は削除されません。
- Pod Identities を使用している場合、アドオンが所有する Pod Identity の関連付けはすべて削除されます。AWS CLI に --preserve オプションを指定すると、関連付けは保持されます。
- Amazon EKS はアドオンの設定の管理を停止します。
- 新しいバージョンが利用可能になると、コンソールが通知を停止します。
- AWS ツールや API を使用してアドオンを更新することはできません。
- 自己管理できるようにアドオンソフトウェアをクラスターに保持するか、クラスターからアドオンソフトウェアを削除するかを選択できます。そのアドオンが提供している機能に依存するクラスターにリソースがない場合のみ、クラスターからアドオンソフトウェアを削除してください。

### 前提条件

アドオンを作成する前に、次を完了します:

- 既存の Amazon EKS クラスター。デプロイするには「[はじめに](#)」を参照してください。
- アドオンに IAM 役割が必要かどうかを確認します。詳細については「」を参照してください。



- デバイスまたは AWS CloudShell にインストールされている eksctl コマンドラインツールのバージョン 0.210.0 以降。eksctl をインストールまたはアップグレードするには eksctl ドキュメントの「[インストール](#)」を参照してください。

## 手順

Amazon EKS アドオンを削除する場合、次の 2 つのオプションがあります。

- クラスター上のアドオンソフトウェアを保持する — このオプションではすべての設定の Amazon EKS 管理が削除されます。また、Amazon EKS による更新の通知機能や、更新の開始後の Amazon EKS アドオンの自動更新機能も削除されます。ただし、クラスター上のアドオンソフトウェアは保持されます。このオプションを選択すると、アドオンは Amazon EKS アドオンではなく、セルフマネージド型インストールになります。このオプションを使用すると、アドオンのダウンタイムは発生しません。
- クラスターからアドオンソフトウェアを完全に削除する — クラスターから Amazon EKS アドオンを削除するのはアドオンに依存するリソースがクラスター上にない場合のみとすることをお勧めします。

Amazon EKS アドオンは eksctl、AWS Management Console、AWS CLI を使用して削除できます。

### アドオンの削除 (eksctl)

1. クラスターに現在インストールされているアドオンを確認します。##### の部分は自分のクラスター名に置き換えます。

```
eksctl get addon --cluster my-cluster
```

出力例は次のとおりです。

| NAME       | VERSION            | STATUS | ISSUES | IAMROLE | UPDATE AVAILABLE |
|------------|--------------------|--------|--------|---------|------------------|
| coredns    | v1.8.7-eksbuild.2  | ACTIVE | 0      |         |                  |
| kube-proxy | v1.23.7-eksbuild.1 | ACTIVE | 0      |         |                  |
| vpc-cni    | v1.10.4-eksbuild.1 | ACTIVE | 0      |         |                  |
| [...]      |                    |        |        |         |                  |

クラスターにあるアドオンおよびバージョンによって、出力が異なる場合があります。

2. アドオンを削除します。**#####** をクラスターの名前に置き換えて、**name-of-add-on** を削除する前のステップの出力で返されたアドオン名に置き換えます。**--preserve** オプションを削除する場合、Amazon EKS がアドオンを管理しなくなるだけでなく、アドオンソフトウェアがクラスターから削除されます。

```
eksctl delete addon --cluster my-cluster --name name-of-addon --preserve
```

アドオンの削除の詳細についてはeksctl ドキュメントの「[Deleting addons](#)」を参照してください。

## アドオンの削除 (AWS コンソール)

1. [Amazon EKS コンソール](#)を開きます。
2. 左のナビゲーションペインで [クラスター] を選択してください。
3. Amazon EKS アドオンを削除するクラスターの名前を選択してください。
4. [アドオン] タブを選択してください。
5. 削除するアドオンを選択してください。
6. [削除] を選択してください。
7. [削除: #####] 確認ダイアログボックスで、次を実行します:
  - a. Amazon EKS でのアドオンの設定の管理を停止する場合は[クラスターで保持する]を選択してください。クラスターでアドオンソフトウェアを保持する場合はこの操作を実行してください。これはアドオンのすべての設定を自分で管理できるようにするためです。
  - b. アドオン名を入力してください。
  - c. [削除] を選択してください。

## アドオンの削除 (AWS CLI)

1. デバイスまたは AWS クラウドシェル にインストールされている eksctl コマンドラインツールのバージョン 0.210.0 以降が必要です。eksctl をインストールまたはアップグレードするにはeksctl ドキュメントの「[インストール](#)」を参照してください。
2. インストールされているアドオンのリストを参照してください。**#####** の部分は自分のクラスター名に置き換えます。

```
aws eks list-addons --cluster-name my-cluster
```

出力例は次のとおりです。

```
{
 "addons": [
 "coredns",
 "kube-proxy",
 "vpc-cni",
 "name-of-addon"
]
}
```

3. インストールされているアドオンを削除します。##### をクラスターの名前に置き換えて、*name-of-addon* を削除するアドオン名に置き換えます。--*preserve* を削除することにより、アドオンソフトウェアがクラスターから削除されます。

```
aws eks delete-addon --cluster-name my-cluster --addon-name name-of-addon --preserve
```

簡略化した出力例を次に示します。

```
{
 "addon": {
 "addonName": "name-of-add-on",
 "clusterName": "my-cluster",
 "status": "DELETING",
 }
}
```

4. 削除のステータスを確認します。##### をクラスターの名前に置き換えて、*name-of-addon* を削除するアドオン名に置き換えます。

```
aws eks describe-addon --cluster-name my-cluster --addon-name name-of-addon
```

アドオンを削除した後の出力例は次のとおりです。

```
An error occurred (ResourceNotFoundException) when calling the DescribeAddon
operation: No addon: name-of-addon found in cluster: my-cluster
```

## Amazon EKS アドオンの IAM ロール

特定の Amazon EKS アドオンでは、AWS API を呼び出す IAM ロールおよび許可が必要です。例えば、Amazon VPC CNI アドオンは特定の AWS API を呼び出して、アカウント内のネットワークリソースを設定します。これらのアドオンには、IAM を使用してアクセス許可を付与する必要があります。具体的には、アドオンを実行しているポッドのサービスアカウントを、特定の IAM ポリシーを持つ IAM ロールに関連付ける必要があります。

クラスターワークロードに AWS アクセス許可を付与する際の推奨方法は、Amazon EKS 機能 Pod Identity を使用することです。Pod Identity の関連付けを使用して、アドオンのサービスアカウントを IAM ロールにマッピングできます。ポッドが関連付けられたサービスアカウントを使用する場合、Amazon EKS はポッドのコンテナに環境変数を設定します。環境変数は、AWS CLI を含む AWS SDK が EKS Pod Identity の認証情報を使用するように設定します。詳細については、「[the section called “Pod Identity”](#)」を参照してください。

Amazon EKS アドオンは、アドオンに対応する Pod Identity の関連付けのライフサイクルを管理するのに役立ちます。例えば、Amazon EKS アドオンと必要な Pod Identity の関連付けを 1 回の API コールで作成または更新できます。Amazon EKS には、推奨される IAM ポリシーを取得するための API も用意されています。

1. [Amazon EKS Pod Identity エージェント](#)がクラスターで設定されていることを確認します。
2. インストールするアドオンに、`describe-addon-versions` AWS CLI オペレーションを使用して IAM アクセス許可が必要かどうかを確認します。`requiresIamPermissions` フラグが `true` の場合、`describe-addon-configurations` オペレーションを使用してアドオンに必要なアクセス許可を決定する必要があります。レスポンスには、推奨されるマネージド IAM ポリシーのリストが含まれます。
3. `describe-addon-configuration` CLI オペレーションを使用して、Kubernetes サービスアカウントの名前と IAM ポリシーを取得します。提案されたポリシーの範囲をセキュリティ要件に照らして評価します。
4. 提案されたアクセス許可ポリシーと Pod Identity に必要な信頼ポリシーを使用して IAM ロールを作成します。詳細については、「[the section called “Pod Identity の関連付けを作成する \(AWS コンソール\)”](#)」を参照してください。
5. CLI を使用して Amazon EKS アドオンを作成または更新します。少なくとも 1 つの Pod Identity の関連付けを指定します。Pod Identity の関連付けは、Kubernetes サービスアカウントの名前、および IAM ロールの ARN です。
  - アドオン API を使用して作成された Pod Identity の関連付けは、それぞれのアドオンによって所有されます。アドオンを削除すると、Pod Identity の関連付けも削除されます。AWS CLI ま

たは API を使用してアドオンを削除するときに、`preserve` オプションを使用してこのカスケード削除を防ぐことができます。必要に応じて、Pod Identity の関連付けを直接更新または削除することもできます。アドオンは、既存の Pod Identity の関連付けの所有権を引き受けることはできません。既存の関連付けを削除し、アドオンの作成または更新オペレーションを使用して再作成する必要があります。

- Amazon EKS では、Pod Identity の関連付けを使用してアドオンの IAM アクセス許可を管理することをお勧めします。前の方法であるサービスアカウント (IRSA) の IAM ロールは引き続きサポートされています。アドオンの `IRSA serviceAccountRoleArn` と Pod Identity の関連付けの両方を指定できます。EKS Pod Identity エージェントがクラスターにインストールされている場合、`serviceAccountRoleArn` は無視され、EKS は指定された Pod Identity の関連付けを使用します。Pod Identity が有効になっていない場合、`serviceAccountRoleArn` が使用されます。
- 既存のアドオンの Pod Identity の関連付けを更新すると、Amazon EKS はアドオンポッドのローリング再起動を開始します。

## Amazon EKS アドオンに関する IAM 情報を取得する

アドオンを作成する前に、AWS CLI を使用して次を判断します:

- アドオンに IAM 許可が必要かどうか
- 使用する推奨 IAM ポリシー

### 手順

1. インストールするアドオンの名前とクラスターの Kubernetes バージョンを決定します。アドオンの詳細については「[the section called “Amazon EKS アドオン”](#)」を参照してください。
2. AWS CLI を使用してアドオンに IAM アクセス許可が必要かどうかを確認します。

```
aws eks describe-addon-versions \
--addon-name <addon-name> \
--kubernetes-version <kubernetes-version>
```

例:

```
aws eks describe-addon-versions \
--addon-name aws-ebs-csi-driver \
--kubernetes-version <kubernetes-version>
```

```
--kubernetes-version 1.30
```

以下の出力例を確認します。requiresIamPermissions は true であり、デフォルトのアドオンバージョンであることに注意してください。推奨される IAM ポリシーを取得するときはアドオンバージョンを指定する必要があります。

```
{
 "addons": [
 {
 "addonName": "aws-ebs-csi-driver",
 "type": "storage",
 "addonVersions": [
 {
 "addonVersion": "v1.31.0-eksbuild.1",
 "architecture": [
 "amd64",
 "arm64"
],
 "compatibilities": [
 {
 "clusterVersion": "1.30",
 "platformVersions": [
 "*"
],
 "defaultVersion": true
 }
],
 "requiresConfiguration": false,
 "requiresIamPermissions": true
 }
],
 "requiresConfiguration": false,
 "requiresIamPermissions": true
 },
 [...]
]
}
```

3. アドオンに IAM アクセス許可が必要な場合はAWS CLI を使用して推奨される IAM ポリシーを取得します。

```
aws eks describe-addon-configuration \
--query podIdentityConfiguration \
--addon-name <addon-name> \
--addon-version <addon-version>
```

例:

```
aws eks describe-addon-configuration \
--query podIdentityConfiguration \
--addon-name aws-ebs-csi-driver \
--addon-version v1.31.0-eksbuild.1
```

以下の出力を確認します。recommendedManagedPolicies を書き留めます。

```
[
 {
 "serviceAccount": "ebs-csi-controller-sa",
 "recommendedManagedPolicies": [
 "arn:aws:iam::aws:policy/service-role/AmazonEBSCSIDriverPolicy"
]
 }
]
```

4. IAM 役割を作成し、管理ポリシーをアタッチします。または管理ポリシーを確認し、必要に応じてアクセス許可の範囲を絞り込みます。詳細については「[the section called “Pod Identity の関連付けを作成する \(AWS コンソール\)”](#)」を参照してください。

## Pod Identity サポートリファレンス

次の表は特定の Amazon EKS アドオンが EKS Pod Identity をサポートしているかどうかを示しています。

| アドオン名                                              | Pod Identity のサポート | 必要な最小バージョン         |
|----------------------------------------------------|--------------------|--------------------|
| <a href="#">Amazon EBS CSI ドライバー</a>               | はい                 | v1.26.0-eksbuild.1 |
| <a href="#">Amazon VPC CNI</a>                     | はい                 | v1.15.5-eksbuild.1 |
| <a href="#">Amazon EFS CSI ドライバー</a>               | はい                 | v2.0.5-eksbuild.1  |
| <a href="#">AWS オープンテレメトリー用ディストロ</a>               | はい                 | v0.94.1-eksbuild.1 |
| <a href="#">Mountpoint for Amazon S3 CSI ドライバー</a> | いいえ                | 該当なし               |

| アドオン名                                        | Pod Identity のサポート | 必要な最小バージョン        |
|----------------------------------------------|--------------------|-------------------|
| <a href="#">Amazon CloudWatch 観測可能エージェント</a> | はい                 | v3.1.0-eksbuild.1 |

この表は 2024 年 10 月 28 日に最終更新されました。

## Pod Identity を使用して Amazon EKS アドオンに IAM 役割を割り当てる

特定の Amazon EKS アドオンでは IAM 役割および許可が必要です。Pod Identity の関連付けを使用するように Amazon EKS アドオンを更新する前に、使用する役割とポリシーを検証します。詳細については「[the section called “IAM情報を取得する”](#)」を参照してください。

### 1. 決定事項:

- cluster-name – アドオンをインストールするクラスターの名前。
- addon-name – インストールするアドオンの名前。
- service-account-name – アドオンで使用する Kubernetes サービスアカウントの名前。
- iam-role-arn – アドオンに十分なアクセス許可を持つ IAM 役割の ARN。役割には EKS Pod Identity に必要な信頼ポリシーが必要です。詳細については「[the section called “Pod Identity の関連付けを作成する \(AWS コンソール\)”](#)」を参照してください。

2. AWS CLI を使用してアドオンを更新します。同じ `--pod-identity-associations` 構文を使用して、アドオンの作成時に Pod Identity の関連付けを指定することもできます。アドオンの更新中に Pod Identity の関連付けを指定すると、以前のすべての Pod Identity の関連付けが上書きされることに注意してください。

```
aws eks update-addon --cluster-name <cluster-name> \
--addon-name <addon-name> \
--pod-identity-associations 'serviceAccount=<service-account-name>,roleArn=<role-arn>'
```

例 :

```
aws eks update-addon --cluster-name mycluster \
--addon-name aws-ebs-csi-driver \
--pod-identity-associations 'serviceAccount=ebs-csi-controller-sa,roleArn=arn:aws:iam::123456789012:role/StorageDriver'
```



### 3. Pod Identity の関連付けが作成されたことを確認します:

```
aws eks list-pod-identity-associations --cluster-name <cluster-name>
```

成功すると、次のような出力が表示されます。EKS アドオンの OwnerARN を書き留めます。

```
{
 "associations": [
 {
 "clusterName": "mycluster",
 "namespace": "kube-system",
 "serviceAccount": "ebs-csi-controller-sa",
 "associationArn": "arn:aws:eks:us-west-2:123456789012:podidentityassociation/mycluster/a-4wvljrezsukshq1bv",
 "associationId": "a-4wvljrezsukshq1bv",
 "ownerArn": "arn:aws:eks:us-west-2:123456789012:addon/mycluster/aws-ebs-csi-driver/9cc7ce8c-2e15-b0a7-f311-426691cd8546"
 }
]
}
```

## Amazon EKS アドオンから Pod Identity の関連付けを削除する

Amazon EKS アドオンから Pod Identity の関連付けを削除します。

### 1. 決定事項:

- cluster-name – アドオンをインストールする EKS クラスターの名前。
- addon-name – インストールする Amazon EKS アドオンの名前。

### 2. アドオンを更新して、Pod Identity の関連付けの空の配列を指定します。

```
aws eks update-addon --cluster-name <cluster-name> \
--addon-name <addon-name> \
--pod-identity-associations "[]"
```

## EKS アドオンの Pod Identity のトラブルシューティング

AWS API、SDK、または CLI オペレーションの試行中にアドオンでエラーが発生した場合は以下を確認します：

- Pod Identity エージェントがクラスター にインストールされている。
  - Pod Identity Agent のインストール方法については「[the section called “エージェントのセットアップ”](#)」を参照してください。
- アドオンに有効な Pod Identity の関連付けがある。
  - AWS CLI を使用して、アドオンで使用するサービスアカウント名の関連付けを取得します。

```
aws eks list-pod-identity-associations --cluster-name <cluster-name>
```

- IAM ロールに、Pod Identity に必要な信頼ポリシーがある。
  - AWS CLI を使用して、アドオンの信頼ポリシーを取得します。

```
aws iam get-role --role-name <role-name> --query Role.AssumeRolePolicyDocument
```

- IAM ロールに、アドオンに必要な許可がある。
  - AWS クラウドTrail を使用して AccessDenied または UnauthorizedOperation イベントを確認します。
- Pod Identity の関連付けのサービスアカウント名はアドオンで使用するサービスアカウント名と一致している。
  - 利用可能なアドオンについては「[the section called “AWSアドオン”](#)」を参照してください。
- pod-identity-webhook という名前の MutatingWebhookConfiguration の設定を確認する。
  - ウェブフックの admissionReviewVersions は v1beta1 である必要があり、v1 では機能しません。

## Amazon EKS アドオンのためにカスタマイズできるフィールドを決定する

Amazon EKS アドオンは標準的なベストプラクティスによる設定を使用して、クラスターにインストールされます。アマゾン EKS アドオンをクラスターに追加する方法については「[the section called “Amazon EKS アドオン”](#)」を参照してください。

Amazon EKS アドオンの設定をカスタマイズすると、高度な機能を有効にすることが可能です。Amazon EKS は、Kubernetes サーバー側の適用機能を使用して、Amazon EKS で管理されていない (ユーザーによる) 設定の内容を上書きすることなく、Amazon EKS によるアドオンの管理を可能にします。詳細については、Kubernetes ドキュメントの「[Server Side Apply](#)」を参照してください。これを実現するために、Amazon EKS は、インストールするアドオンごとに最低限のフィールドセットを管理します。Amazon EKS や別の Kubernetes コントロールプレーンプロセス (kube-

controller-manager など) によって管理されていないすべてのフィールドは、ユーザーが問題なく管理できます。

#### Important

Amazon EKS で管理しているフィールドを変更すると、Amazon EKS がアドオンを管理できなくなるため、アドオンの更新時にユーザーによる変更が上書きされる可能性があります。

## フィールド管理構文

Kubernetes オブジェクトの詳細を表示すると、マネージド型フィールドとアンマネージド型フィールドの両方が出力に返されます。管理対象フィールドは、次のいずれかになります。

- 完全マネージド型 - フィールドのすべてのキーは Amazon EKS によって管理されます。このフィールドの値を変更すると、競合の原因となります。
- 部分的マネージド型 - フィールドの一部のキーは Amazon EKS によって管理されます。Amazon EKS によって明示的に管理されているキーに対する変更のみが競合の原因となります。

どちらのタイプのフィールドも `manager: eks` でタグづけされています。

各キーはフィールド自体を表す . (これは常に空のセットがマッピングされます) であるか、またはサブフィールドまたは項目を表す文字列です。フィールドの管理状況の出力は以下のタイプの宣言で構成されます。

- `f:name` (`name` はリスト内のフィールドの名前)。
- `k:keys` (`keys` はリスト項目のフィールドのマップ)。
- `v:value` (`value` はリスト項目を正確な JSON 形式で記述した値)。
- `i:index` (`index` はリスト内の項目の位置)。

CoreDNS アドオン出力内で次に続く部分には、以前の宣言を表示します。

- フルマネージド型フィールド — マネージド型フィールドに `f: (フィールド)` が指定されていますが、`k:(キー)` はありません。この場合は、フィールド全体が管理されています。このフィールドの値を変更すると、競合の原因となります。

次の出力ではcoredns という名前のコンテナが、eks によって管理されていることがわかります。args、image、および imagePullPolicy サブフィールドも eks によって管理されています。このフィールドの値を変更すると、競合の原因となります。

```
[...]
f:containers:
 k:{"name":"coredns"}:
 .: {}
 f:args: {}
 f:image: {}
 f:imagePullPolicy: {}
[...]
manager: eks
[...]
```

- 部分的マネージド型キー - マネージド型キーに値が指定されている場合、宣言されたキーはそのフィールドで管理されます。指定されたキーを変更すると、競合の原因となります。

次の出力ではname キーを使用する eks が、config-volume および tmp ボリュームセットを管理していることがわかります。

```
[...]
f:volumes:
 k:{"name":"config-volume"}:
 .: {}
 f:configMap:
 f:items: {}
 f:name: {}
 f:name: {}
 k:{"name":"tmp"}:
 .: {}
 f:name: {}
[...]
manager: eks
[...]
```

- 部分的マネージド型フィールドへのキーの追加 - 特定のキーバリューのみが管理されている場合であれば、競合を引き起こすことなく、引数などの追加のキーをフィールドに追加できます。キーを追加する場合はまず、そのフィールドがマネージド型でないことを確認してください。マネージド型の値を追加または変更すると、競合の原因となります。

次の出力ではname キーおよび name フィールドの両方が、マネージド型であることがわかります。コンテナ名を追加または変更すると、このマネージド型のキーとの競合が発生します。

```
[...]
f:containers:
 k:{"name":"coredns"}:
[...]
```

```
[...]
 f:name: {}
[...]
```

```
manager: eks
[...]
```

## 手順

kubectl を使用すると、任意の Amazon EKS アドオンについて、どのフィールドが Amazon EKS によって管理されているかを知ることができます。

Amazon EKS や別の Kubernetes コントロールプレーンプロセス (kube-controller-manager など) によって管理されていないすべてのフィールドは、ユーザーが問題なく管理できます。

1. 確認するアドオンを決定します。クラスターにデプロイされた deployments と DaemonSet をすべて表示するには、「[the section called “クラスターリソースにアクセスする”](#)」を参照してください。
2. アドオンのマネージド型フィールドを表示するには、次のコマンドを実行します。

```
kubectl get type/add-on-name -n add-on-namespace -o yaml
```

たとえば、次のコマンドを使用すると、CoreDNS アドオンのマネージド型フィールドを表示できます。

```
kubectl get deployment/coredns -n kube-system -o yaml
```

フィールドの管理状態は返される出力の中の、次のセクションに記載されています。

```
[...]
managedFields:
 - apiVersion: apps/v1
 fieldsType: FieldsV1
```

```
fieldsV1:
[...]
```

#### Note

出力に、managedFields が表示されない場合、--show-managed-fields をコマンドに追加し、もう一度実行してください。使用している kubectl のバージョンによって、管理フィールドがデフォルトで返されるかどうかが決まります。

## 次のステップ

アドオンのために、AWS が所有していないフィールドをカスタマイズします。

## デプロイ中にコンテナイメージの署名を検証する

[AWS Signer](#) を使用し、デプロイ時に署名済みのコンテナイメージを検証したい場合は、次のいずれかのソリューションが使用可能です。

- [Gatekeeper と Ratify](#) — Gatekeeper をアドミッションコントローラーとして使用し、AWS Signer プラグインで構成された Ratify を署名を検証するためのウェブフックとして使用します。
- [Kyverno](#) — 署名を検証するための AWS Signer プラグインで構成された Kubernetes ポリシーエンジン。

#### Note

コンテナイメージの署名を検証する前に、選択したアドミッションコントローラーの要求に応じて、[Notation](#) トラストストアとトラストポリシーを構成します。

# クラスターリソースを整理およびモニタリングする

この章では、クラスターの管理に役立つ以下のトピックについて説明します。また、AWS Management Console で [Kubernetes リソース](#)に関する情報を表示することもできます。

- Kubernetes ダッシュボードは、Kubernetes クラスター用の汎用でウェブベースの UI です。これにより、ユーザーはクラスター自体を管理するだけでなく、クラスターで実行されているアプリケーションの管理およびトラブルシューティングを行うことができます。詳細については、「[Kubernetes ダッシュボード](#)」GitHub リポジトリを参照してください。
- [the section called “メトリクスサーバー”](#) – Kubernetes メトリクスサーバーは、クラスター内のリソース使用状況データを集約します。デフォルトでは、クラスターにはデプロイされませんが、Kubernetes ダッシュボードや [the section called “Horizontal Pod Autoscaler”](#) などの Kubernetes のアドオンで使用されます。このトピックでは、メトリクスサーバーのインストール方法を説明します。
- [the section called “Helm を使用してアプリケーションをデプロイする”](#) – Kubernetes 用の Helm パッケージマネージャーを使用すると、Kubernetes クラスターにアプリケーションをインストールし、管理できます。このトピックは、Helm バイナリをインストールして実行する際に役立ちます。これにより、ローカルコンピュータで Helm CLI を使用してチャートをインストールおよび管理することができます。
- [the section called “リソースのタグ付け”](#) – Amazon EKS リソースを管理しやすくするために、タグ形式で各リソースに独自のメタデータを割り当てることができます。ここでは、タグとその作成方法について説明します。
- [the section called “Service Quotas”](#) – AWS アカウントには、AWS のサービスごとにデフォルトのクォータ (以前は制限と呼ばれていました) があります。Amazon EKS のクォータと拡大方法について説明します。

## Amazon EKS クラスターのコストをモニタリングして最適化する

コストのモニタリングは、Amazon EKS で Kubernetes クラスターを管理する上で不可欠な要素です。クラスターのコストを可視化することで、リソースの使用率を最適化し、予算を設定し、デプロイに関するデータ主導の意思決定を行うことができます。Amazon EKS には 2 つのコストモニタリングソリューションがあり、それぞれに独自の利点があり、コストを効果的に追跡して割り当てるのに役立ちます。

AWS 請求を活用した Amazon EKS のコスト配分データの分割 — このネイティブ機能は、AWS 請求コンソールにシームレスに統合されるため、他の AWS のサービスと同様の使い慣れたインターフェイスとワークフローを使って、コストの分析および割り当てを行うことができます。コスト配分の分割を使用すると、Kubernetes コストに関するインサイトを他の AWS 支出と並行して直接得ることができるため、AWS 環境全体でコストを簡単に、総合的に最適化できるようになります。コストカテゴリーやコスト異常検出など、既存の AWS 請求の機能を活用して、コスト管理機能をさらに強化することもできます。詳細については、「AWS ユーザーガイド」の「[Understanding split cost allocation data](#)」を参照してください。

Kubecost — Amazon EKS は、Kubernetes のコストモニタリングツールである Kubecost をサポートしています。Kubecost には、リソースごとの詳細なコストの内訳、コスト最適化のために推奨される方法、すぐに使用できるダッシュボードやレポートなど、Kubernetes ネイティブのコストモニタリングに特化した機能が多数用意されています。また、Kubecost を使用して AWS コストと使用状況レポートと統合すると、正確な価格データを取得でき、Amazon EKS のコストを正確に把握することができます。[Kubecost をインストール](#)する方法について説明します。無料の Kubecost サブスクリプションの取得については、[Kubecost](#) AWS Marketplace ページを参照してください。

## 分割コスト割り当てを使用した AWS 請求で Pod ごとにコストを表示する

### Amazon EKS の AWS コスト配分データの分割を使用したコストモニタリング

Amazon EKS の AWS コスト配分データの分割を使用すると、Amazon EKS クラスターのコストをきめ細かく把握することができます。これにより、Kubernetes アプリケーションのコストと使用状況の分析、最適化、チャージバックが可能になります。アプリケーションのコストは、Kubernetes アプリケーションが使用した Amazon EC2 CPU とメモリリソースに基づいて個々の部門およびチームに割り当てられます。Amazon EKS のコスト配分データの分割を使用することで、Pod あたりのコストを可視化し、名前空間、クラスター、その他の Kubernetes プリミティブを使用している Pod ごとのコストデータを集計することができます。以下は、Amazon EKS コスト配分データの分析に使用できる Kubernetes プリミティブの一例です。

- クラスター名
- デプロイ
- 名前空間
- ノード
- ワークロード名
- ワークロードタイプ



[ユーザー定義のコスト配分タグ](#)もサポートされています。コスト配分データの分割の詳細については、「AWS Billing ユーザーガイド」の「[Understanding split cost allocation data](#)」を参照してください。

## コストと使用状況レポートの設定

コスト管理のコンソール、AWS コマンドラインインターフェイス、AWS SDK のいずれかで、[EKS のコスト配分データの分割] をオンにします。

コスト配分データの分割には以下を使用します。

1. コスト配分データの分割にオプトインします。詳細については、「AWS コストと使用状況レポート ユーザーガイド」の「[分割コスト配分データの有効化](#)」を参照してください。
2. 新しいレポートまたは既存のレポートにデータを含めます。
3. レポートを表示します。請求とコスト管理コンソールを使用するか、Amazon Simple Storage Service でレポートファイルを表示できます。

## Kubecost のインストール

Amazon EKS がサポートする Kubecost を使用して、Pod、ノード、名前空間、ラベルなどの Kubernetes リソースごとにコストを分類して監視することができます。このトピックでは、Kubecost のインストールと Kubecost ダッシュボードへのアクセスについて説明します。

Amazon EKS は、クラスターコストの可視化のため、AWS に最適化された Kubecost のバンドルを提供します。既存の AWS サポート契約を使用してサポートを受けることができます。使用できる Kubecost のバージョンの詳細については、「[the section called “Kubecost の詳細”](#)」を参照してください。

### Note

Kubecost v2 では、いくつかの主要な新機能が導入されています。[Kubecost v2 の詳細はこちらです。](#)

Kubecost の詳細については、[Kubecost](#) ドキュメントと「[よくある質問](#)」を参照してください。

## Amazon EKS 最適化 Kubecost バンドルをインストールする

Amazon EKS 最適化 Kubecost バンドルをインストールするには、次のいずれかの手順を使用します。

- 開始する前に、「[Kubecost - Architecture Overview](#)」を確認して、Kubecost が Amazon EKS でどのように機能するかを理解することをお勧めします。
- Amazon EKS を初めて使用する場合は、Amazon EKS 最適化 Kubecost バンドルのインストールが簡素化されるため、Helm を使用してインストールすることをお勧めします。詳細については、「[Deploying Kubecost on an Amazon EKS cluster using Amazon EKS add-on](#)」を参照してください。
- インストールをカスタマイズするには、Amazon EKS 最適化 Kubecost バンドルを Helm で設定できます。詳細については、Kubecost ドキュメントの「[Deploying Kubecost on an Amazon EKS cluster using Helm](#)」を参照してください。

## Kubecost ダッシュボードにアクセスする

Amazon EKS 最適化 Kubecost バンドルのセットアップが完了したら、Kubecost ダッシュボードにアクセスできます。詳細については、「[the section called “Kubecost ダッシュボードにアクセスする”](#)」を参照してください。

## Kubecost ダッシュボードにアクセスする

### 前提条件

1. kubecost 関連のポッドの状態が「実行中」であることを確認します。

```
kubectl get pods --namespace kubecost
```

## Kubecost ダッシュボードにアクセスする

1. デバイス上でポート転送を有効にして、Kubecost ダッシュボードを公開します。

```
kubectl port-forward deployment/kubecost-cost-analyzer 9090 --namespace kubecost
```

あるいは、[AWS Load Balancer Controller](#) を使用して、認証、認可、およびユーザー管理のために Kubecost を公開して Amazon Cognito を使用することもできます。詳細については、「[How](#)

[to use Application Load Balancer and Amazon Cognito to authenticate users for your Kubernetes web apps](#)」(Application Load Balancer および Amazon Cognito を使用して Kubernetes Web アプリのユーザーを認証する方法)を参照してください。

2. 前の手順を実行したのと同じデバイスで、Web ブラウザを開き、次のアドレスを入力します。

```
http://localhost:9090
```

ブラウザに Kubecost の概要ページが表示されます。クラスターのサイズによっては、Kubecost がメトリクスを収集するまでに 5～10 分 (またはそれ以上) かかる場合があります。クラスターにかかる累積コスト、関連する Kubernetes 資産コスト、および毎月の総支出を含む Amazon EKS の支出額を確認できます。

3. クラスターレベルでコストを追跡するには、請求対象の Amazon EKS リソースにタグを付けます。詳細については、「[the section called “請求用のリソースにタグを付ける”](#)」を参照してください。
  - コスト配分 — Amazon EKS の月間費用と、各名前空間およびその他のディメンションの過去 7 日間の累積費用を表示します。これは、アプリケーションのどの部分が Amazon EKS の支出に寄与しているかを理解するのに役立ちます。
  - アセット — Amazon EKS リソースに関連する AWS インフラストラクチャアセットの費用を表示します。

## Kubecost の詳細

Amazon EKS は、クラスターコストの可視化のため、AWS に最適化された Kubecost のバンドルを提供します。Amazon EKS がサポートする Kubecost を使用して、Pod、ノード、名前空間、ラベルなどの Kubernetes リソースごとにコストを分類して監視することができます。

このトピックでは、Kubecost の使用可能なバージョンと、使用可能な利用枠の違いについて説明します。EKS は Kubecost のバージョン 1 とバージョン 2 をサポートしています。各バージョンは異なる利用枠で利用できます。Amazon EKS 最適化 Kubecost バンドルは、Amazon EKS クラスターにおいて追加コストなしで使用できます。Amazon Managed Service for Prometheus などの関連する AWS サービスの使用に対して課金される場合があります。また、既存の AWS サポート契約を使用してサポートを受けることもできます。

Kubernetes のプラットフォーム管理者および財務リーダーとして、Kubecost を使用して Amazon EKS の請求の内訳を可視化できます。また、コストを配分し、アプリケーションチームなどの組織単位にチャージバックすることも可能です。社内チームや事業部門に、実際の AWS の請求に基づく

透明で正確なコストデータを提供できます。さらに、インフラストラクチャ環境とクラスター内の使用パターンに基づいて、コストを最適化するためのカスタマイズされた推奨事項を取得することもできます。Kubecost の詳細については、[Kubecost](#) ドキュメントを参照してください。

Kubecost のカスタムバンドルと Kubecost の無料バージョン (OpenCost としても知られている) の違いは何ですか？

AWS と Kubecost が協力して Kubecost のカスタマイズバージョンを提供しました。このバージョンには、追加料金なしで商用機能のサブセットが含まれています。Kubecost のカスタムバンドルに含まれる機能については、以下の表を参照してください。

## Kubecost v2

Kubecost の v1 と v2 の違いは何ですか？

Kubecost 2.0 は以前のバージョンからのメジャーアップグレードであり、まったく新しい API バックエンドなどの主要な新機能が含まれています。なお、[Allocation](#) API と [Assets](#) API は完全に下位互換性があります。[Kubecost ドキュメントを確認して、スムーズに移行してください](#)。機能強化の完全なリストについては、[Kubecost v2.0 の発表](#)と[すべてのリリースノート](#)を参照してください。

### Important

[アップグレードの前に Kubecost ドキュメントを確認してください](#)。アップグレードすると、レポートの可用性に影響する場合があります。

主要機能の比較:

| 機能             | Kubecost 無料利用枠 2.0       | Amazon EKS 最適化 Kubecost バンドル 2.0                                | Kubecost Enterprise 2.0             |
|----------------|--------------------------|-----------------------------------------------------------------|-------------------------------------|
| [クラスターコストの可視化] | クラスター数に制限なし (コア数は最大 250) | Amazon Managed Service for Prometheus と統合した場合、コア制限のない統合マルチクラスター | 無制限の数の環境 (マルチクラウドなど) で統合クラスターの数が無制限 |
| デプロイ           | ユーザーホスト                  | ユーザーホスト                                                         | ユーザーホスティング、Kubecost ホス              |

| 機能                            | Kubecost 無料利用枠 2.0          | Amazon EKS 最適化 Kubecost バンドル 2.0                                                               | Kubecost Enterprise 2.0                                                                        |
|-------------------------------|-----------------------------|------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------|
|                               |                             |                                                                                                | ティング (専用テナント)、SaaS                                                                             |
| [サポートされるデータベース]               | ローカル Prometheus             | Amazon Managed Service for Prometheus またはローカル Prometheus                                       | 任意の Prometheus 風およびカスタムのデータベース                                                                 |
| データベース保持のサポート (未加工メトリクス)      | 15 日間                       | 無制限の履歴データ                                                                                      | 無制限の履歴データ                                                                                      |
| Kubecost API および UI の保持 (ETL) | 15 日間                       | 15 日間                                                                                          | 無制限                                                                                            |
| [ハイブリッドクラウドの可視性]              | -                           | Amazon EKS と Amazon EKS Anywhere クラスター                                                         | マルチクラウドおよびハイブリッドクラウド                                                                           |
| [アラートと定期レポート]                 | プライマリクラスターでのみサポート、250 コアに制限 | すべてのクラスターで <a href="#">効率性アラート</a> 、 <a href="#">予算アラート</a> 、 <a href="#">支出変更アラート</a> などをサポート | すべてのクラスターで <a href="#">効率性アラート</a> 、 <a href="#">予算アラート</a> 、 <a href="#">支出変更アラート</a> などをサポート |
| [保存されたレポート]                   | -                           | 15 日間のメトリクスを使用したレポート                                                                           | 無制限の履歴データおよびメトリクスを使用したレポート                                                                     |
| [クラウド請求の統合]                   | プライマリクラスターでのみサポート、250 コアに制限 | AWS のカスタム価格サポート (複数のクラスターと複数のアカウントを含む)                                                         | 任意のクラウドに対するカスタム料金のサポート                                                                         |

| 機能                                    | Kubecost 無料利用枠 2.0          | Amazon EKS 最適化 Kubecost バンドル 2.0 | Kubecost Enterprise 2.0               |
|---------------------------------------|-----------------------------|----------------------------------|---------------------------------------|
| [割引に関する推奨事項]                          | プライマリクラスターでのみサポート、250 コアに制限 | プライマリクラスターのインサイト、ただし 250 コアの制限なし | マルチクラスターの洞察                           |
| [ガバナンス: 監査]                           | -                           | -                                | 過去のコストイベントの監査                         |
| [シングルサインオン (SSO) のサポート]               | -                           | Amazon Cognito のサポート             | Okta、Auth0、PingID、KeyCloak、その他のカスタムのも |
| SAML 2.0 使用したロールベースのアクセスコントロール (RBAC) | -                           | -                                | Okta、Auth0、PingID、KeyCloak、その他のカスタムのも |
| [エンタープライズトレーニングとオンボーディング]             | -                           | -                                | フルサービストレーニングおよび FinOps オンボーディング       |
| チーム                                   | -                           | -                                | はい                                    |

#### 新機能:

以下の機能にはメトリクス制限があります。

- Kubecost アグリゲータ
- ネットワークモニタリング
- Kubecost アクション
- コレクション
- 異常検出

- コンテナリクエストの適切なサイズ設定
- Kubecost 予測
- フィルタリングと集計のオートコンプリート

メトリクスの制限:

| メトリクス         | Kubecost 無料利用枠 2.0       | Amazon EKS 最適化 Kubecost バンドル 2.0 | Kubecost Enterprise 2.0 |
|---------------|--------------------------|----------------------------------|-------------------------|
| クラスターサイズ      | クラスター数に制限なし (コア数は最大 250) | 無制限                              | 無制限                     |
| メトリクスの保持      | 15 日間                    | 15 日間                            | 無制限                     |
| マルチクラスターのサポート | 利用不可                     | 利用可能                             | 利用可能                    |
| コアの制限         | クラスターあたり 250 コア          | コア制限なし                           | コア制限なし                  |

## Kubecost v1

| 機能              | Kubecost 無料利用枠  | Amazon EKS 最適化 Kubecost バンドル | Kubecost Enterprise                              |
|-----------------|-----------------|------------------------------|--------------------------------------------------|
| デプロイメント         | ユーザーホスト         | ユーザーホスト                      | ユーザーホストまたは Kubecost ホスト (SaaS)                   |
| [サポートされるクラスター数] | 無制限             | 無制限                          | 無制限                                              |
| [サポートされるデータベース] | ローカル Prometheus | ローカル Prometheus または Amazon   | Prometheus、Amazon Managed Service for Prometheus |

| 機能                    | Kubecost 無料利用枠 | Amazon EKS 最適化<br>Kubecost バンドル        | Kubecost Enterprise                    |
|-----------------------|----------------|----------------------------------------|----------------------------------------|
|                       |                | Managed Service for Prometheus         | s、Cortex、または Thanos                    |
| [データベース保持サポート]        | 15 日間          | 無制限の履歴データ                              | 無制限の履歴データ                              |
| Kubecost API 保持 (ETL) | 15 日間          | 15 日間                                  | 無制限の履歴データ                              |
| [クラスターコストの可視化]        | 単一クラスター        | 統合マルチクラスター                             | 統合マルチクラスター                             |
| [ハイブリッドクラウドの可視性]      | -              | Amazon EKS と Amazon EKS Anywhere クラスター | マルチクラウドとハイブリッドクラウドのサポート                |
| [アラートと定期レポート]         | -              | 効率性アラート、予算アラート、支出変更アラートなどをサポート         | 効率性アラート、予算アラート、支出変更アラートなどをサポート         |
| [保存されたレポート]           | -              | 15 日間のデータを使用したレポート                     | 無制限の履歴データを使用したレポート                     |
| [クラウド請求の統合]           | 個々のクラスターに必須    | AWS のカスタム価格サポート (複数のクラスターと複数のアカウントを含む) | AWS のカスタム価格サポート (複数のクラスターと複数のアカウントを含む) |
| [割引に関する推奨事項]          | 単一クラスターの洞察     | 単一クラスターの洞察                             | マルチクラスターの洞察                            |
| [ガバナンス: 監査]           | -              | -                                      | 過去のコストイベントの監査                          |



| 機能                                       | Kubecost 無料利用枠 | Amazon EKS 最適化<br>Kubecost バンドル | Kubecost Enterprise             |
|------------------------------------------|----------------|---------------------------------|---------------------------------|
| [シングルサインオン (SSO) のサポート]                  | -              | Amazon Cognito のサポート            | Okta、Auth0、PingID、KeyCloak      |
| [SAML 2.0 を使用したロールベースのアクセスコントロール (RBAC)] | -              | -                               | Okta、Auth0、PingID、Keycloak      |
| [エンタープライズトレーニングとオンボーディング]                | -              | -                               | フルサービストレーニングおよび FinOps オンボーディング |

## よくある質問

Amazon EKS での Kubecost の使用に関する次の一般的な質問と回答を参照してください。

Kubecost API 保持 (ETL) 機能とは何ですか？

Kubecost ETL 機能では、メトリクスを集約して整理し、さまざまな詳細度レベル (namespace-level、pod-level、deployment-level など) で表面的なコストを可視化します。Amazon EKS 最適化 Kubecost バンドルでは、お客様は過去 15 日間のメトリクスからデータとインサイトを取得します。

アラート機能と定期レポート機能とは何ですか？ どのようなアラートとレポートが含まれていますか？

Kubecost アラートにより、チームはリアルタイムの Kubecost 支出とクラウド支出に関する最新情報を受け取ることができます。定期レポートでは、チームは Kubernetes 支出とクラウド支出の履歴のカスタマイズされたビューを受け取ることができます。どちらも、Kubecost UI または Helm 値で設定できます。E メール、Slack、Microsoft Teams がサポートされています。

保存されたレポートには何が含まれていますか？

Kubecost の保存されたレポートは、コストと効率のメトリクスに関する事前定義されたビューです。クラスター、名前空間、ラベル、その他を基準とするコストが含まれます。

クラウド請求インテグレーションとは何ですか？

AWS 請求 API との統合により、Kubecost でクラスター以外のコスト (Amazon S3 など) を表示できます。さらに、Kubecost で、スポット使用量、Savings Plans、法人割引を考慮して、Kubecost のクラスター内予測を実際の請求データと一致させられるようになります。

割引に関する推奨事項にはどのようなものがありますか？

Kubecost では、ユーザーが Kubernetes インフラストラクチャおよび支出を最適化するのに役立つインサイトと自動化が提供されます。

この機能には料金がかかりますか？

いいえ。Amazon EKS 最適化 Kubecost バンドルは、追加料金なしで使用できます。これに含まれていない Kubecost 機能が追加で必要な場合は、AWS Marketplace を通じて、または直接 Kubecost から、Kubecost のエンタープライズライセンスを購入できます。

Amazon EKS 最適化 Kubecost バンドルでは、サポートを利用できますか？

はい。Amazon EKS 最適化 Kubecost バンドルを使用している場合にのみ利用可能です。

どうすれば、Amazon EKS 最適化 Kubecost バンドルのサポートを受けることができますか？

[AWS へのお問い合わせ](#)から AWS サポートチームに対してサポートケースを開くことができます。

Amazon EKS 統合によって提供される Kubecost 機能を使用するのにライセンスが必要ですか？

いいえ。

より正確なレポートを作成するために、Kubecost と AWS コストと使用状況レポートを統合できますか？

はい。AWS コストと使用状況レポートからデータを取り込むように Kubecost を構成することで、割引、スポット料金、予約インスタンス料金などを含む正確なコストを可視化することができます。詳細については、Kubecost ドキュメントの「[AWS Cloud Billing Integration](#)」を参照してください。

このバージョンは、Amazon EC2 でのセルフマネージド Kubernetes クラスターのコスト管理をサポートしていますか？

いいえ。Amazon EKS 最適化 Kubecost バンドルは、Amazon EKS クラスターとのみ互換性があります。

Kubecost は AWS Fargate での Amazon EKS のコストを追跡できますか？

Kubecost では、Fargate での Amazon EKS のクラスターコストが可能な限り可視化されますが、Amazon EC2 での Amazon EKS に比べて精度が低くなります。これは主に、使用量に対する課金方法の違いによるものです。Fargate での Amazon EKS では、消費したリソースに対して課金されます。Amazon EC2 ノードでの Amazon EKS では、プロビジョニングしたリソースに対して課金されます。Kubecost は、CPU、RAM、エフェメラルストレージなどのノード仕様に基づいて Amazon EC2 ノードのコストを計算します。Fargate では、コストは Fargate ポッドのリクエストされたリソースに基づいて計算されます。

Kubecost の更新と新バージョンはどうすれば入手できますか？

Kubecost バージョンは、標準の Helm アップグレード手順でアップグレードできます。最新バージョンは、[Amazon ECR Public Gallery](#) にあります。

**kubect1-cost** CLI はサポートされていますか？どのようにインストールすればよいですか？

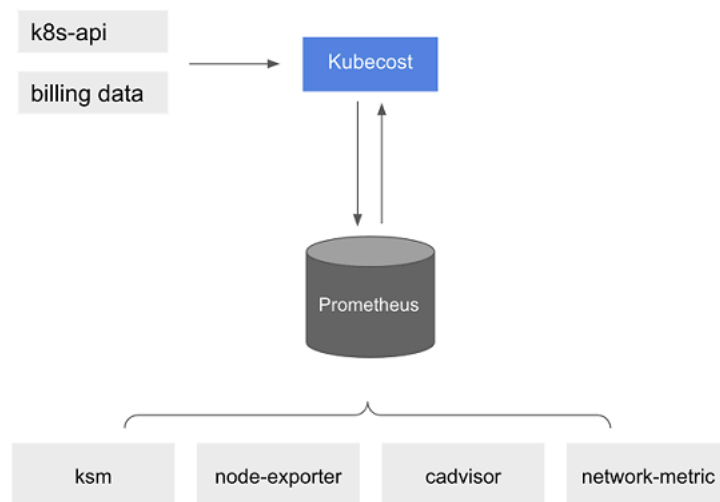
はい。Kubect1-cost は、Kubecost (Apache 2.0 ライセンス) によるオープンソースツールで、Kubernetes コスト配分のメトリクスに CLI でアクセスできます。kubect1-cost をインストールするには、「GitHub」の「[インストール](#)」を参照してください。

Kubecost ユーザーインターフェイスはサポートされていますか？どうすれば使用できますか？

Kubecost では、kubect1 ポート転送、イングレス、またはロードバランサーを介してアクセスできる Web ダッシュボードが提供されます。または、AWS Load Balancer Controller を使用して Kubecost を公開し、Amazon Cognito を使用して認証、認可、およびユーザー管理を行うこともできます。詳細については、AWS ブログの「[How to use Application Load Balancer and Amazon Cognito to authenticate users for your Kubernetes web apps](#)」を参照してください。

## Kubecost のその他の機能

- Kubecost の v1 と v2 の両方で次の機能を使用できます。
  - コストメトリクスのエクスポート — Amazon EKS に最適化されたコストモニタリングは、オープンソースのモニタリングシステムおよび時系列データベースである Kubecost および Prometheus と一緒にデプロイされます。Kubecost は Prometheus からメトリクスを読み取り、コスト配分計算を実行して、メトリクスを Prometheus に書き込みます。Kubecost フロントエンドが Prometheus からメトリクスを読み込み、それらを Kubecost ユーザーインターフェイスに表示します。このアーキテクチャを以下の図に示します。



[Prometheus](#) がプリインストールされていると、クエリを記述して Kubecost データをインジェストし、現在のビジネスインテリジェンスシステムに取り込んでさらに分析できます。また、現在の [Grafana](#) ダッシュボードのデータソースとして使用して、社内チームが精通している Amazon EKS クラスターの費用を表示することもできます。Prometheus クエリを作成する方法の詳細については、GitHub の <https://opencost.io/docs/installation/prometheus/> の readme ファイルを参照するか、[Kubecost Github リポジトリ](#) の Grafana JSON モデル例を参照として使用してください。

- AWS コストと使用状況レポートの統合 — Amazon EKS クラスターのコスト配分計算を実行するために、Kubecost は AWS Price List API から AWS サービスおよび AWS リソースの公開価格情報を取得します。Kubecost と AWS コストと使用状況レポートを統合して、AWS アカウントに固有の価格情報の正確性を高めることもできます。この情報には、エンタープライズディスカウントプログラム、リザーブドインスタンスの使用量、Savings Plans、スポット使用量が含まれます。AWS コストと使用状況レポートの統合の動作方法の詳細については、Kubecost ドキュメントの「[AWS Cloud Billing Integration](#)」を参照してください。

## Kubernetes Metrics Server でのリソース使用状況の表示

Kubernetes Metrics Server は、クラスター内のリソース使用状況データを集約する働きをし、デフォルトでは Amazon EKS クラスターにデプロイされません。詳細については、GitHub の「[Kubernetes Metrics Server \(Kubernetes メトリクスサーバー\)](#)」を参照してください。Kubernetes Metrics Server は、[Scale pod deployments with Horizontal Pod Autoscaler](#) や [Kubernetes Dashboard](#) など他の Kubernetes アドオンによってよく使用されます。詳細については、Kubernetes ドキュメ

ントの「[Resource metrics pipeline](#)」を参照してください。このトピックでは、Kubernetes メトリクスサーバーを Amazon EKS クラスターにデプロイする方法について説明します。

#### Important

メトリクスはポイントインタイム分析を目的としており、履歴分析の正確なソースではありません。監視ソリューションとして、またはその他の自動スケーリング以外の目的には使用できません。監視ツールの詳細については「[クラスターをモニタリングする](#)」を参照してください。

## 考慮事項

- マニフェストを使用して Kubernetes メトリクスサーバーを Fargate ノードに手動でデプロイする場合は、デフォルトの 10250 以外のポートを使用するように metrics-server デプロイを設定します。このポートは Fargate で予約されています。メトリクスサーバーの Amazon EKS アドオンバージョンは、10251 ポートを使用するように事前設定されています。
- セキュリティグループとネットワーク ACL で、metrics-server ポッドと他のすべてのノードおよびポッド間の 10250 ポートが許可されていることを確認します。Kubernetes メトリクスサーバーでは、引き続き 10250 ポートを使用してクラスター内の他のエンドポイントからメトリクスを収集します。Fargate ノードにデプロイする場合は、設定された代替メトリクスサーバーポートと 10250 ポートの両方を許可します。

## Amazon EKS アドオンを使用してコミュニティアドオンとしてデプロイする

新規: AWSコンソールまたは Amazon EKS APIs Server をコミュニティアドオンとしてデプロイできるようになりました。

### AWS コンソールでデプロイ

1. AWS コンソールで EKS クラスターを開きます
2. 「アドオン」タブから Get More Add-ons. を選択します。
3. 「コミュニティアドオン」セクションで、メトリクスサーバーを選択し、次へを選択してください。

4. EKS はクラスターに適したバージョンのアドオンを決定します。バージョンドロップダウンメニューを使用してバージョンを変更できます。
5. 次へ を選択し、作成 を選択してアドオンをインストールします。

## 追加リソース

[the section called “コミュニティアドオン”](#) の詳細を確認してください。

コミュニティアドオンは他の Amazon EKS アドオンと同じ方法でインストールまたは更新します。

- [the section called “アドオンを作成する”](#)
- [the section called “アドオンを更新する”](#)
- [the section called “アドオンを削除する”](#)

## マニフェストを使用してデプロイする

新規: AWSコンソールまたは Amazon EKS APIs Server をコミュニティアドオンとしてデプロイできるようになりました。これらのマニフェストのインストール手順はアーカイブされます。

1. 次のコマンドを使用してメトリクスサーバーをデプロイします。

```
kubectl apply -f https://github.com/kubernetes-sigs/metrics-server/releases/latest/download/components.yaml
```

Fargate を使用している場合はこのファイルを変更する必要があります。デフォルト設定ではメトリクスサーバーはポート 10250 を使用します。このポートは Fargate で予約されています。components.yaml のポート 10250 への参照を 10251 などの別のポートに置き換えます。

2. 次のコマンドを使用して、metrics-server デプロイに必要な数の Pod が実行されていることを確認します。

```
kubectl get deployment metrics-server -n kube-system
```

出力例は次のとおりです。

| NAME           | READY | UP-TO-DATE | AVAILABLE | AGE |
|----------------|-------|------------|-----------|-----|
| metrics-server | 1/1   | 1          | 1         | 6m  |

3. ノードのリソース (CPU/メモリ) 使用状況を表示して、メトリクスサーバーが動作しているかテストします。

```
kubectl top nodes
```

4. エラーメッセージ Error from server (Forbidden) が表示された場合は Kubernetes RBAC 設定を更新する必要があります。Kubernetes RBAC ID にはクラスターメトリクスを読み取るための十分なアクセス許可が必要です。[メトリクスを読み取るために必要な最低限の Kubernetes API アクセス許可](#)を GitHub で確認します。[役割などの AWS IAM ID に Kubernetes API へのアクセス権を付与する](#)方法について説明します。

## Helm を使用して Amazon EKS にアプリケーションをデプロイする

Kubernetes 用の Helm パッケージマネージャーを使用すると、Kubernetes クラスターにアプリケーションをインストールし、管理できます。詳細については[Helm のドキュメント](#)を参照してください。このトピックは Helm バイナリをインストールして実行する際に役立ちます。これにより、ローカルシステムで Helm CLI を使用してチャートをインストールおよび管理することができます。

### Important

Amazon EKS クラスターに Helm チャートをインストールするにはあらかじめ Amazon EKS で動作するように kubectl を設定しておく必要があります。この設定をまだ行っていない場合は続行する前に「[the section called “kubectl を使用してクラスターにアクセスする”](#)」を参照してください。次のコマンドがクラスターに対して成功した場合は正しく設定されています。

```
kubectl get svc
```

1. クライアントオペレーティングシステムに適したコマンドを実行してください。
  - macOS で [Homebrew](#) を使用している場合は次のコマンドを使用してバイナリをインストールします。

```
brew install helm
```



- Windows で [Chocolatey](#) を使用している場合は、次のコマンドを使用してバイナリをインストールします。

```
choco install kubernetes-helm
```

- Linux を使用している場合は、次のコマンドを使用してバイナリをインストールします。

```
curl -fsSL -o get_helm.sh https://raw.githubusercontent.com/helm/helm/main/scripts/get-helm-3
chmod 700 get_helm.sh
./get_helm.sh
```

 Note

最初に openssl をインストールする必要があるというメッセージを受け取った場合、次のコマンドを使用してインストールできます。

```
sudo yum install openssl
```

1. PATH で新しいバイナリを取得するため、現在のターミナルウィンドウを閉じて新しいターミナルウィンドウを開きます。
2. インストールした Helm のバージョンを確認してください。

```
helm version --template='{{ .Version }}{{ "\n" }}'
```

出力例は次のとおりです。

```
v3.17.2
```

3. インストールされているバージョンがクラスターのバージョンと互換性があることを確認してください。詳細については、「[Supported Version Skew](#)」を確認してください。例えば、3.17.x で稼働している場合、サポート対象の Kubernetes バージョンは 1.29.x ~ 1.32.x の範囲外であってはなりません。
4. この時点で、任意の Helm コマンド (`helm install chart-name` など) 実行して、クラスター内の Helm チャートをインストール、変更、削除、またはクエリすることができます。Helm を初めて使用する場合は、インストールするチャートがないときは次の操作を実行できます：



- サンプルチャートをインストールして試します。Helm の「[クイックスタートガイド](#)」の「[サンプルチャートをインストールする](#)」を参照してください。
- サンプルチャートを作成し、Amazon ECR にプッシュします。詳細については「Amazon Elastic Container Registry ユーザーガイド」の「[Pushing a Helm chart \(Helm チャートをプッシュする\)](#)」を参照してください。
- [eks-charts](#) GitHub リポジトリか [ArtifactHub](#) から Amazon EKS チャートをインストールします。

## タグを使用して Amazon EKS リソースを整理する

タグを使用すると、Amazon EKS リソースの管理に役立ちます。このトピックでは、タグ機能の概要とタグの作成方法について説明します。

### トピック

- [タグの基本](#)
- [リソースのタグ付け](#)
- [タグの制限](#)
- [請求用のリソースにタグを付ける](#)
- [コンソールでのタグの処理](#)
- [CLI、API または eksctl でのタグの操作](#)

#### Note

タグは、Kubernetes のラベルやアノテーションとは別のメタデータのタイプです。これらのその他のメタデータタイプの詳細については、Kubernetes ドキュメントの次のセクションを参照してください。

- [ラベル \(Labels\) とセレクター \(Selectors\)](#)
- [注釈](#)

## タグの基本

タグとはAWS リソースに割り当てるラベルです。各タグはキーとオプションの値で構成されます。

タグを使用して AWS リソースを分類できます。例えば、目的、所有者、環境などに基づいてリソースを分類できます。同じ型のリソースが多い場合に、特定のリソースに割り当てたタグを使用して、そのリソースをすばやく識別できます。例えば Amazon EKS クラスターに一連のタグを定義して、各クラスターの所有者とスタックレベルを追跡できます。リソースタイプごとに一貫した一連のタグキーを考案することをお勧めします。追加したタグに基づいてリソースを検索およびフィルタリングできます。

タグを追加したら、いつでもタグキーと値は編集でき、タグはリソースからいつでも削除できます。リソースを削除すると、リソースのタグも削除されます。

タグには Amazon EKS に関する意味論的な意味はなく、完全に文字列として解釈されます。タグの値は空の文字列に設定できます。ただし、タグの値を null に設定することはできません。そのリソースの既存のタグと同じキーを持つタグを追加した場合、古い値は新しい値によって上書きされます。

AWS Identity and Access Management (IAM) を使用すると、AWS アカウント内のどのユーザーがタグを管理するアクセス許可を持つかをコントロールできます。

## リソースのタグ付け

Amazon EKS の以下のリソースがタグをサポートしています。

- クラスター
- マネージド型ノードグループ
- Fargate プロファイル

以下を使用して、これらのリソースにタグ付けできます。

- Amazon EKS コンソールを使用している場合、新規または既存のリソースにいつでもタグを適用できます。これを行うには、関連リソースページの [Tags] (タグ) タブを使用します。詳細については、「[the section called “コンソールでのタグの処理”](#)」を参照してください。
- eksctl を使用している場合は、--tags オプションを使ってリソースを作成する際に、リソースにタグを適用できます。
- AWS CLI、Amazon EKS API、または AWS SDK を使用している場合は、関連する API アクションの tags パラメータを使用して、新しいリソースにタグを適用できます。TagResource API アクションを使用して既存のリソースにタグを適用することもできます。詳細については、「[TagResource](#)」を参照してください。

リソース作成アクションによっては、リソースの作成時にリソースのタグを指定することもできます。リソースの作成中にタグを適用できない場合、リソースの作成は失敗します。これにより、タグ付けするリソースが、指定したタグで作成されているか、まったく作成されていないかが確認できます。リソースの作成時にタグ付けを行う場合、リソースの作成後にカスタムタグ付けスクリプトを実行する必要はありません。

タグは、作成したリソースに関連付けられている他のリソースには伝達されません。例えば、Fargate プロファイルのタグは、それによってスケジュールされた Pod などの、Fargate プロファイルに関連付けられている他のリソースには適用されません。

## タグの制限

タグには以下の制限があります。

- 1 つのリソースに対して最大 50 個のタグを関連付けることができます。
- 1 つのリソースに対してタグキーを繰り返すことはできません。各タグキーは一意である必要があり、それぞれに使用できる値は 1 つのみです。
- キーの値には最大 128 UTF-8 文字を使用できます。
- 値の最大長は 256 UTF-8 文字です。
- 複数の AWS サービスおよびリソースがタグ付けスキーマを使用する場合、使用する文字の種類を制限します。一部のサービスでは、使用できる文字に制限がある場合があります。通常、使用できる文字は、英字、数字、スペース、および特殊文字 +、-、=、.、\_、:、/、@ です。
- タグのキーと値では、大文字と小文字が区別されます。
- キーまたは値のプレフィックスとして、aws:、AWS:、またはその大文字と小文字の組み合わせを変えたものは使用しないでください。これらは AWS でのみ使用するよう予約されています。このプレフィックスを持つタグのキーや値を編集または削除することはできません。このプレフィックスを持つタグは、リソースあたりのタグ数の制限にはカウントされません。

## 請求用のリソースにタグを付ける

Amazon EKS クラスターにタグを適用すると、そのタグを [Cost & Usage Reports] (コストと使用状況レポート) 内のコスト配分に使用できます。[Cost & Usage Reports] (コストと使用状況レポート) の計測データには、すべての Amazon EKS タスクの使用状況が示されます。詳細については、「AWS 請求ユーザーガイド」の「[AWS コストと使用状況レポート](#)」を参照してください。

AWS 生成されたコスト配分タグ、特に aws:eks:cluster-name では、[Cost Explorer] (コストエクスプローラー) で Amazon EC2 インスタンスのコストを個々の Amazon EKS クラスターごとに分

類できます。ただし、このタグにはコントロールプレーンの費用を捕捉するものではありません。このタグは、Amazon EKS クラスターに参加している Amazon EC2 インスタンスに自動的に追加されます。この動作は、インスタンスが Amazon EKS マネージド型ノードグループまたは Karpenter を使用して、または直接 Amazon EC2 を使用してプロビジョニングされたかどうかに関係なく、発生します。この特定のタグは 50 個のタグの上限にはカウントされません。タグを使用するには、アカウント所有者が AWS 請求コンソールまたは API を使用してタグをアクティブ化する必要があります。AWS Organizations 管理アカウントの所有者がこのタグをアクティブ化すると、すべての組織のメンバーアカウントにも有効になります。

同じタグキー値を持つリソースに基づいて、請求情報を整理することもできます。例えば、複数のリソースに特定のアプリケーション名のタグを付け、請求情報を整理することができます。これにより、複数のサービスを利用しているアプリケーションの合計コストを確認することができます。タグによるコスト配分レポートの設定の詳細については、AWS 請求ユーザーガイドの「[コスト配分月次レポート](#)」を参照してください。

#### Note

レポートを有効にすると、約 24 時間後に、今月のデータを表示できるようになります。

Cost Explorer は、AWS 無料利用枠の一部として利用できるレポートツールです。Cost Explorer を使用すると、過去 13 か月間の Amazon EKS リソースのグラフを表示できます。また、次の 3 か月間のリソース使用量も予測できます。時間の経過と共に AWS リソースに費やす金額のパターンを確認できます。例えば、Cost Explorer を使用して、さらに調べる必要がある分野を特定し、コストを把握するために使用できる傾向を確認できます。データの時間範囲を指定したり、時間データを日または月ごとに表示することもできます。

## コンソールでのタグの処理

Amazon EKS コンソールを使用して、新規または既存のクラスターおよびマネージド型ノードグループに関連付けられたタグを管理できます。

Amazon EKS コンソールでリソース固有のページを選択すると、そのページにリソースのリストが表示されます。例えば、左のナビゲーションペインで [Clusters] (クラスター) を選択すると、Amazon EKS クラスターのリストがコンソールに表示されます。これらのリストの1つ(例えば、特定のクラスター)からタグをサポートするリソースを選択すると、Tags タブでタグを表示および管理できます。

また、AWS Management Console の [Tag Editor] (タグエディタ) を使用することもできます。これにより、タグを一貫性のある方法で管理できます。詳細については、「AWS タグエディタユーザーガイド」の「[タグエディタを使用して AWS リソースにタグを付ける](#)」を参照してください。

## リソース作成時のタグの追加

Amazon EKS クラスターおよびマネージド型ノードグループ、および Fargate プロファイルの作成時に、タグを追加できます。詳細については、「[the section called “クラスターを作成する”](#)」を参照してください。

## リソースでのタグの追加と削除

クラスターに関連付けられたタグをリソースのページから直接追加または削除できます。

1. [アマゾン EKS コンソール](#)を開きます。
2. ナビゲーションバーから、使用する AWS リージョンを選択します。
3. 左のナビゲーションペインで [Clusters] (クラスター) を選択してください。
4. 指定するクラスターを選択します。
5. [タグ] タブを選択し、[タグ管理] を選択します。
6. [Manage tags] (タグの管理) ページで、必要に応じてタグを追加または削除します。
  - タグを追加するには、[タグの追加] を選択します。その後、タグごとにキーと値を指定します。
  - タグを削除するには、[Remove tag] (タグの削除) を選択します。
7. 追加または削除するタグごとにこのプロセスを繰り返します。
8. [更新] を選択して終了します。

## CLI、API または `eksctl` でのタグの操作

リソースのタグの追加、更新、リスト表示、および削除には、次の AWS CLI コマンドまたは Amazon EKS API オペレーションを使用します。1 つのコマンドで新しいリソースを同時に作成しながらタグを追加するのに使用できるのは `eksctl` だけです。

| タスク                    | AWS CLI                  | AWS Tools for Windows PowerShell      | API アクション                     |
|------------------------|--------------------------|---------------------------------------|-------------------------------|
| 1 つ以上のタグを追加、または上書きします。 | <a href="#">タグリソース</a>   | <a href="#">Add-EKSResourceTag</a>    | <a href="#">TagResource</a>   |
| 1 つ以上のタグを削除します。        | <a href="#">タグなしリソース</a> | <a href="#">Remove-EKSResourceTag</a> | <a href="#">UntagResource</a> |

以下の例では、AWS CLI を使用して、リソースに対してタグ付けまたはタグ削除する方法を示しています。

#### 例 1: 既存のクラスターへのタグ付け

次のコマンドは既存のクラスターにタグ付けします。

```
aws eks tag-resource --resource-arn resource_ARN --tags team=devs
```

#### 例 2: 既存のクラスターでのタグ削除

次のコマンドは既存のクラスターからタグを削除します。

```
aws eks untag-resource --resource-arn resource_ARN --tag-keys tag_key
```

#### 例 3: リソースのタグのリスト取得

次のコマンドは、既存のリソースに関連付けられているタグのリストを取得します。

```
aws eks list-tags-for-resource --resource-arn resource_ARN
```

一部のリソース作成アクションでは、リソースの作成と同時にタグを指定できます。次のアクションでは、リソース作成時のタグの指定がサポートされます。

| タスク                       | AWS CLI                                | AWS Tools<br>for Windows<br>PowerShell | API アクション                                 | eksctl                   |
|---------------------------|----------------------------------------|----------------------------------------|-------------------------------------------|--------------------------|
| クラスターを作成する                | <a href="#">クラスター作成</a>                | <a href="#">New-EKSCluster</a>         | <a href="#">CreateCluster</a>             | create<br>cluster        |
| マネージド型<br>ノードグループ<br>の作成* | <a href="#">create-nodegroup</a>       | <a href="#">New-EKSNodegroup</a>       | <a href="#">CreateNodegroup</a>           | create<br>nodegroup      |
| Fargate プロファイルの作成         | <a href="#">create-fargate-profile</a> | <a href="#">New-EKSFargateProfile</a>  | <a href="#">CreateFargateProfile.html</a> | create<br>fargateprofile |

- マネージド型ノードグループの作成時に Amazon EC2 インスタンスにもタグ付けを行う場合は、起動テンプレートを使用してマネージド型ノードグループを作成します。詳細については、「[the section called “Amazon EC2 インスタンスへのタグ付け”](#)」を参照してください。インスタンスがすでに存在する場合は、インスタンスに手動でタグ付けを行うことができます。詳細については、「Amazon EC2 ユーザーガイド」の「[リソースのタグ付け](#)」を参照してください。

## Amazon EKS と Fargate Service Quotas を表示して管理する

Amazon EKS は、Service Quotas と統合されています。Service Quotas は、クォータを一元的な場所から表示および管理するために使用できる AWS サービスです。詳細については、「Service Quotas ユーザーガイド」の「[Service Quotas とは](#)」を参照してください。Service Quotas との統合により、AWS Management Console および AWS CLI を使用して、Amazon EKS および AWS Fargate Service Quotas の値を簡単に検索できます。

### AWS Management Console で EKS Service Quotas を表示する

1. [Service Quotas コンソール](#)を開きます。
2. 左のナビゲーションペインで [AWS サービス] を選択します。
3. [AWS サービス] リストから、[Amazon Elastic Kubernetes Service (Amazon EKS)] または [AWS Fargate] を選択します。



[Service Quotas] の一覧には、Service Quotas 名、適用された値 (使用可能な場合)、AWS のデフォルトのクォータ、クォータ値が調整可能かどうかが表示されます。

4. 説明など、Service Quotas に関する追加情報を表示するには、クォータ名を選択します。
5. (オプション) クォータの引き上げをリクエストするには、[Request quota increase (クォータ引き上げリクエスト)] を選択、または必要な情報を入力または選択して、[Request (リクエスト)] を選択します。

AWS Management Console を使用してさらに Service Quotas の操作を行うには、「[Service Quotas ユーザーガイド](#)」を参照してください。クォータの引き上げをリクエストするには、Service Quotas ユーザーガイドの「[クォータ引き上げリクエスト](#)」を参照してください。

## AWS CLI で EKS サービスクォータを表示する

次のコマンドを実行して、Amazon EKS クォータを表示します。

```
aws service-quotas list-aws-default-service-quotas \
 --query 'Quotas[*].
{Adjustable:Adjustable,Name:QuotaName,Value:Value,Code:QuotaCode}' \
 --service-code eks \
 --output table
```

次のコマンドを実行して、Fargate クォータを表示します。

```
aws service-quotas list-aws-default-service-quotas \
 --query 'Quotas[*].
{Adjustable:Adjustable,Name:QuotaName,Value:Value,Code:QuotaCode}' \
 --service-code fargate \
 --output table
```

### Note

返されるクォータは、現在の AWS リージョンのこのアカウントの Fargate で同時に実行可能な Amazon ECS タスクまたは Amazon EKS Pod の数です。

AWS CLI を使用して、さらに Service Quotas の操作を実行するには、AWS CLI コマンドリファレンスの「[Service Quotas](#)」を参照してください。クォータの引き上げをリクエストするには、AWS CLI コマンドリファレンスの [request-service-quota-increase](#) コマンドを参照してください。



## Amazon EKS Service Quotas

AWS では、AWS Management Console を使用して現在のクォータを表示することをお勧めします。詳細については、「[the section called “AWS Management Console で EKS Service Quotas を表示する”](#)」を参照してください。

デフォルトの EKS サービスクォータを表示するには、「AWS 全般のリファレンス」の「[Amazon Elastic Kubernetes Service endpoints and quotas](#)」を参照してください。

これらの Service Quotas は、Service Quotas コンソールの [Amazon Elastic Kubernetes Service (Amazon EKS)] に表示されます。調整可能として表示される値のクォータの引き上げをリクエストすることもできます。「Service Quotas ユーザーガイド」の「[クォータ引き上げのリクエスト](#)」を参照してください。

### Note

以下のクォータは Service Quotas では使用できません。

- クラスターあたりの Pod Identity の関連付けは、サポートされているリージョンごとに 1,000 個あり、このクォータは調整できません。
- リモートノードネットワークには最大 15 CIDR、ハイブリッドノードのクラスターごとにリモートポッドネットワークには最大 15 CIDR を使用できます。これは調整可能なクォータではありません。

## AWS Fargate サービスクォータ

Service Quotas コンソールの AWS Fargate サービスは、いくつかのサービスクォータを一覧表示します。使用量がサービスクォータに近づいたときに警告するアラームを設定することもできます。詳細については、「[the section called “Fargate のリソース使用状況メトリクスをモニタリングするための CloudWatch アラームの作成”](#)」を参照してください。

新規 AWS アカウントの初期の低いクォータは、経時的に引き上げられる可能性があります。Fargate は、各 AWS リージョン内のアカウント使用率を常に監視し、使用率に基づいてクォータを自動的に引き上げます。調整可能として表示される値のクォータの引き上げをリクエストすることもできます。詳細については「Service Quotas ユーザーガイド」の「[クォータの引き上げのリクエスト](#)」を参照してください。

AWS では、AWS Management Console を使用して現在のクォータを表示することをお勧めします。詳細については、「[the section called “AWS Management Console で EKS Service Quotas を表示する”](#)」を参照してください。

AWS Fargate on EKS のデフォルトのサービスクォータを表示するには、「AWS 全般のリファレンス」の「[Fargate サービスクォータ](#)」を参照してください。

 Note

Fargate は、Amazon ECS タスクと Amazon EKS Pod の起動レートクォータを追加で適用します。詳細については、「Amazon ECS ガイド」の「[AWS スロットリングのクォータ](#)」を参照してください。

# Amazon EKS のセキュリティ

「AWS」ではクラウドセキュリティが最優先事項です。セキュリティを最も重視する組織の要件を満たすために構築された「AWS」のデータセンターとネットワークアーキテクチャは、お客様に大きく貢献します。

セキュリティは、「AWS」と顧客の間の責任共有です。[責任共有モデル](#)では、この責任がクラウドのセキュリティおよびクラウド内のセキュリティとして説明されています。

- クラウドのセキュリティ – AWS は、AWS クラウドで AWS のサービスを実行するインフラストラクチャを保護する責任を担います。Amazon EKS において、AWS は Kubernetes コントロールプレーンの責任を分担しており、この範囲にはコントロールプレーンノードと etcd データベースが含まれています。[AWS コンプライアンスプログラム](#)の一環として、サードパーティーの監査が定期的にセキュリティの有効性をテストおよび検証しています。Amazon EKS に適用されるコンプライアンスプログラムについては、「[コンプライアンスプログラムによる対象範囲内の AWS サービス](#)」を参照してください。
- クラウド内のセキュリティ – お客様の責任事項には、次の領域が含まれます。
  - データプレーンのセキュリティ設定。これには、Amazon EKS コントロールプレーンからお客様の VPC 内へのトラフィックの通過を許可する、セキュリティグループの設定を含みます。
  - ノードおよびコンテナに対する設定。
  - ノードのオペレーティングシステム (更新やセキュリティパッチを含みます)
  - その他の関連アプリケーションソフトウェア：
    - ファイアウォールのルールなど、ネットワークコントロールの設定および管理。
    - IAM を使用する、またはそれに追加して行う、Identity and Access Managementのプラットフォームレベルの管理。
  - お客様のデータの機密性、企業の要件、該当する法律および規制

Amazon EKS は、規制対象および機密性の高いアプリケーションの複数のコンプライアンスプログラムで認定されています。Amazon EKS は、[SOC](#)、[PCI](#)、[ISO](#)、[FedRAMP-Moderate](#)、[IRAP](#)、[C5](#)、[K-ISMS](#)、[ENS High](#)、[OSPAR](#)、[HITRUST CSF](#) に準拠しており、[HIPAA](#) 対象サービスです。詳細については、「[アクセスを管理する](#)」を参照してください。

このドキュメントは、Amazon EKS を使用する際の責任共有モデルの適用について理解するのに役立ちます。以下のトピックで、セキュリティおよびコンプライアンスの目的を満たすよう

に、Amazon EKS を設定する方法について説明します。Amazon EKS リソースのモニタリングやセキュリティ確保に役立つ他の AWS サービスの使用方法についても説明します。

#### Note

Linux コンテナはコントロールグループ (cgroup) と名前空間で構成されています。これらにより、コンテナのアクセスが可能な対象を制限しながら、すべてのコンテナに、ホストの Amazon EC2 インスタンスと同じ Linux カーネルを共有させることができます。コンテナをルートユーザー (UID 0) として実行したり、ホストリソースや (ホストネットワークやホスト PID の) 名前空間へのアクセスをコンテナに許可したりすることは一切お勧めしません。これは、コンテナが提供する分離の有効性を低下させます。

## トピック

- [ベストプラクティスによる Amazon EKS クラスターの保護](#)
- [Amazon EKS の脆弱性を分析する](#)
- [Amazon EKS クラスターのコンプライアンス検証](#)
- [アマゾン エラスティックKubernetesサービス のセキュリティに関する考慮事項](#)
- [Kubernetes のセキュリティに関する考慮事項](#)
- [Amazon EKS Auto Mode におけるセキュリティに関する考慮事項](#)
- [Amazon EKS の Identity and Access Management](#)

## ベストプラクティスによる Amazon EKS クラスターの保護

Amazon EKS でのセキュリティのベストプラクティスは、「Amazon EKS のベストプラクティスガイド」の「[セキュリティのベストプラクティス](#)」に記載されています。

## Amazon EKS の脆弱性を分析する

セキュリティは、Kubernetes クラスターとアプリケーションの設定や管理を行う際の重要な考慮事項です。以下に、EKS クラスターのセキュリティ設定を分析するためのリソース、脆弱性をチェックするためのリソース、およびその分析を実行できる AWS サービスとの統合を示します。

## Amazon EKS 用 Center for Internet Security (CIS) ベンチマーク

[Center for Internet Security \(CIS\) Kubernetes Benchmark](#) では、Amazon EKS のセキュリティ設定に関するガイダンスを提供しています。ベンチマーク:

- ユーザーが Kubernetes コンポーネントのセキュリティ設定を担当する (マネージド型とセルフマネージド型の両方の) Amazon EC2 ノードに適用されます。
- Amazon EKS の使用時に Kubernetes クラスターとノードを安全に設定できるように、コミュニティが承認した標準的な方法を提供します。
- これは、コントロールプレーンでのログ記録の設定、ノードのセキュリティ設定、ポリシー、マネージド型サービスの 4 つのセクションで構成されます。
- Amazon EKS で現在入手可能なすべての Kubernetes バージョンをサポートし、[kube-bench](#) (Kubernetes クラスターで CIS ベンチマークを使用して構成をチェックするための標準的なオープンソースのツール) により実行されます。

詳細については、「[Introducing The CIS Amazon EKS Benchmark \(CIS Amazon EKS ベンチマークの紹介\)](#)」を参照してください。

CIS ベンチマークされた AMI でノードグループを更新する自動 `aws-sample` パイプラインについては、「[EKS-Optimized AMI Hardening Pipeline](#)」を参照してください。

## Amazon EKS のプラットフォームバージョン

Amazon EKS プラットフォームのバージョンは、クラスターコントロールプレーンの機能を表しています。これには、Kubernetes API サーバーでどのフラグが有効であるかや、現在の Kubernetes パッチバージョンなどの情報が含まれます。新しいクラスターは、最新のプラットフォームバージョンでデプロイされます。詳細については、「[the section called “プラットフォームのバージョン”](#)」を参照してください。

[Amazon EKS クラスター](#) を更新し、新しい Kubernetes バージョンにすることができます。新しい Kubernetes バージョンが Amazon EKS で利用可能になったら、利用可能な最新のバージョンが使用できるように、クラスターをタイムリーに更新することをお勧めします。EKS での Kubernetes のバージョンの詳細については、「[the section called “Kubernetes バージョン”](#)」を参照してください。

# オペレーティングシステムの脆弱性リスト

## AL2023 脆弱性リスト

[Amazon Linux セキュリティセンター](#)で Amazon Linux 2023 のセキュリティイベントまたはプライバシーイベントを追跡するか、関連する [RSS フィード](#)をサブスクライブします。セキュリティおよびプライバシーイベントには、影響を受ける問題の概要、パッケージ、および問題を修正するためにインスタンスを更新する手順などがあります。

## Amazon Linux 2 脆弱性リスト

[Amazon Linux セキュリティセンター](#)で Amazon Linux 2 のセキュリティイベントまたはプライバシーイベントを追跡するか、関連する [RSS フィード](#)をサブスクライブします。セキュリティおよびプライバシーイベントには、影響を受ける問題の概要、パッケージ、および問題を修正するためにインスタンスを更新する手順などがあります。

## Amazon Inspector によるノード検出

[Amazon Inspector](#) を使用すると、ノードからネットワークに意図しない接続が可能か、また、それらの Amazon EC2 インスタンスに脆弱性があるかを確認できます。

## Amazon GuardDuty によるクラスターとノードの検出

Amazon GuardDuty は、AWS 環境内のアカウント、コンテナ、ワークロード、データを保護する脅威検知サービスです。GuardDuty には、EKS クラスターに対する潜在的な脅威を検出する EKS Protection とランタイムモニタリングの 2 つの機能があります。

詳細については「[the section called “Amazon GuardDuty”](#)」を参照してください。

## Amazon EKS クラスターのコンプライアンス検証

AWS のサービスが特定のコンプライアンスプログラムの対象であるかどうかを確認するには、「[コンプライアンスプログラムによる対象範囲内の AWS サービス](#)」をご覧ください。関心のあるコンプライアンスプログラムを選択してください。一般的な情報については、「[AWS コンプライアンスプログラム](#)」を参照してください。

サードパーティーの監査報告書は、AWS アーティファクトを使用してダウンロードすることができます。詳細については、「[Downloading reports in AWS Artifact](#)」を参照してください。

AWS のサービスの使用時におけるユーザーのコンプライアンス責任は、データの機密性、企業のコンプライアンス目的、適用法と規制に応じて異なります。AWS は、コンプライアンスに役立つ以下のリソースを提供しています。

- [セキュリティのコンプライアンスとガバナンス](#) – これらのソリューション実装ガイドでは、アーキテクチャ上の考慮事項について説明し、セキュリティとコンプライアンスの機能をデプロイする手順を示します。
- [HIPAA 対応サービスのリファレンス](#) – HIPAA 対応サービスの一覧が提供されています。すべての AWS のサービスが HIPAA 対象であるわけではありません。
- [AWS コンプライアンスのリソース](#) – このワークブックとガイドのコレクションは、お客様の業界や所在地に適用される場合があります。
- [AWS Customer Compliance Guide](#) – コンプライアンスの観点から見た責任共有モデルを理解できます。このガイドは、AWS のサービスを保護するためのベストプラクティスを要約したものであり、複数のフレームワーク (米国標準技術研究所 (NIST)、ペイメントカード業界セキュリティ標準評議会 (PCI)、国際標準化機構 (ISO) など) にわたるセキュリティ統制へのガイダンスがまとめられています。
- AWS Config デベロッパーガイドの「[ルールでのリソースの評価](#)」 – AWS Config サービスは、リソースの設定が内部規定、業界のガイドライン、規制にどの程度適合しているかを評価します。
- [AWS セキュリティハブ](#) – この AWS サービスは、AWS 内のセキュリティ状態に関する包括的なビューを提供します。Security Hub では、セキュリティコントロールを使用して AWS リソースを評価し、セキュリティ業界標準とベストプラクティスに対するコンプライアンスをチェックします。サポートされているサービスとコントロールの一覧については、[Security Hub のコントロールリファレンス](#)を参照してください。
- [Amazon GuardDuty](#) – この AWS サービスは、環境をモニタリングして、疑わしいアクティビティや悪意のあるアクティビティがないか調べることで、AWS アカウント、ワークロード、コンテナ、データに対する潜在的な脅威を検出します。GuardDuty を使用すると、特定のコンプライアンスフレームワークで義務付けられている侵入検知要件を満たすことで、PCI DSS などのさまざまなコンプライアンス要件に対応できます。
- [AWS Audit Manager](#) – この AWS サービスでは、AWS の使用状況を継続的に監査し、リスクの管理方法と、規制や業界標準へのコンプライアンスの管理方法を簡素化できます。

## アマゾン エラスティック Kubernetes サービス のセキュリティに関する考慮事項

以下はアマゾン EKS に影響するため、クラウドのセキュリティに関する考慮事項です。



## トピック

- [Amazon EKS でのインフラストラクチャセキュリティ](#)
- [Amazon EKS クラスターの耐障害性の説明](#)

## Amazon EKS でのインフラストラクチャセキュリティ

マネージドサービスの Amazon Elastic Kubernetes Service は、AWS グローバルネットワークセキュリティによって保護されています。AWS セキュリティサービスと AWS がインフラストラクチャを保護する方法については「[AWS クラウドセキュリティ](#)」を参照してください。インフラストラクチャセキュリティのベストプラクティスを使用して AWS 環境を設計するには「セキュリティの柱 - AWS 適切なアーキテクチャを備えたフレームワーク」の「[インフラストラクチャの保護](#)」を参照してください。

AWS が公開している API コールを使用し、ネットワーク経由で Amazon EKS にアクセスします。クライアントは以下をサポートする必要があります。

- Transport Layer Security (TLS)。TLS 1.2 が必須で、TLS 1.3 をお勧めします。
- DHE (楕円ディフィー・ヘルマン鍵共有) や ECDHE (楕円曲線ディフィー・ヘルマン鍵共有) などの完全前方秘匿性 (PFS) による暗号スイート。これらのモードは Java 7 以降など、ほとんどの最新システムでサポートされています。

また、リクエストにはアクセスキー ID と、IAM プリンシパルに関連付けられているシークレットアクセスキーを使用して署名する必要があります。または、[AWS Security Token Service](#) (AWS STS) で一時的なセキュリティ認証情報を生成して、リクエストに署名することもできます。

Amazon EKS のクラスターの作成時には、使用するクラスターの VPC サブネットを指定します。Amazon EKS には、2 つ以上のアベイラビリティゾーンにサブネットが必要です。パブリックサブネットとプライベートサブネットを持つ VPC をお勧めします。これにより、Kubernetes がパブリックサブネットにパブリックロードバランサーを作成し、プライベートサブネットにあるノードで実行されている Pod へのトラフィックを負荷分散します。

VPC に関する考慮事項の詳細については、「」を参照してください [the section called “VPC およびサブネットの要件”](#)

チュートリアル「[Amazon EKS の使用を開始する](#)」で提供されている AWS CloudFormation テンプレートを使用して VPC およびノードグループを作成した場合、コントロールプレーンとノードのセキュリティグループは推奨設定で構成されます。



セキュリティグループの考慮事項の詳細については、「」を参照してください[the section called “セキュリティグループの要件”](#)

新しいクラスターを作成すると、Amazon EKS によってマネージド型の Kubernetes API サーバーのエンドポイントが作成されます。ユーザーはこのエンドポイントを、(kubectl などの Kubernetes 管理ツールにより) クラスターとの通信に使用します。デフォルトでは、この API サーバーエンドポイントはインターネットに公開されます。API サーバーへのアクセスの保護には、AWS Identity and Access Management (IAM) と、ネイティブの Kubernetes [ロールベースアクセスコントロール \(RBAC\)](#) が組み合わせて使用されます。

Kubernetes API サーバーへのプライベートアクセスを有効にすると、ノードと API サーバー間のすべての通信が VPC 内で行われるようになります。インターネットから API サーバーにアクセスできる IP アドレスを制限したり、API サーバーへのインターネットアクセスを完全に無効にしたりできます。

クラスターエンドポイントアクセスの変更の詳細については、「」を参照してください[the section called “クラスターエンドポイントのアクセスの変更”](#)

Amazon VPC CNI を使用するか、[Project Calico](#) などのサードパーティーツールを使用して、Kubernetes ネットワークポリシーを実装できます。Amazon VPC CNI をネットワークポリシーに使用方法の詳細については、「[the section called “Kubernetes ポリシー”](#)」を参照してください。Project Calico はサードパーティのオープンソースプロジェクトです。詳細については、[Project Calico のドキュメント](#)を参照してください。

## AWS プライベートリンクを使用して Amazon EKS にアクセスする

AWS プライベートリンク を使用して、VPC と Amazon Elastic Kubernetes Service 間にプライベート接続を作成できます。インターネットゲートウェイ、NAT デバイス、VPN 接続、AWS Direct Connect 接続のいずれかを使用せずに、VPC 内にあるかのように Amazon EKS にアクセスできます。VPC のインスタンスはパブリック IP アドレスがなくても Amazon EKS にアクセスできます。

AWS プライベートリンク が電源を供給するインターフェイスエンドポイントを作成することにより、このプライベート接続を確立します。インターフェイスエンドポイントに対して有効にするサブネットにエンドポイントネットワークインターフェイスを作成します。これらは Amazon EKS 宛てのトラフィックのエントリポイントとして機能するリクエスト管理型ネットワークインターフェイスです。

詳細については「AWS プライベートリンク ガイド」の「[AWS プライベートリンク を介した AWS サービスへのアクセス](#)」を参照してください。

## Amazon EKS の考慮事項

- Amazon EKS のインターフェイスエンドポイントを設定する前に、「AWS プライベートリンク ガイド」の「[考慮事項](#)」を確認してください。
- Amazon EKS はインターフェイスエンドポイントを介してすべての API アクションの呼び出しをサポートしていますが、Kubernetes API の呼び出しはサポートしていません。Kubernetes API サーバーは、既に[プライベートエンドポイント](#)をサポートしています。Kubernetes API サーバーのプライベートエンドポイントは、クラスターとの通信に使用する Kubernetes API サーバー用のプライベートエンドポイントを作成します (kubectl などの Kubernetes 管理ツールを使用)。Kubernetes API サーバーへの[プライベートアクセス](#)を有効にすると、ノードと API サーバー間のすべての通信が VPC 内で行われるようになります。AWS プライベートリンク for the Amazon EKS API はパブリックインターネットにトラフィックを公開することなく、VPC から Amazon EKS API を呼び出すのに役立ちます。
- Amazon EKS をインターフェイスエンドポイントを介してのみアクセスできるように設定することはできません。
- Amazon EKS のインターフェイスエンドポイントには AWS プライベートリンク の標準料金が適用されます。各アベイラビリティゾーンでインターフェイスエンドポイントがプロビジョニングされる 1 時間ごと、ならびにインターフェイスエンドポイントを介して処理されたデータに対して請求されます。詳細については「[AWS プライベートリンク の料金](#)」を参照してください。
- VPC エンドポイントポリシーは Amazon EKS をサポートしていません。これらのポリシーを使用すると、インターフェイスエンドポイントを介して Amazon EKS へのアクセスを制御できます。さらに、セキュリティグループをエンドポイントのネットワークインターフェイスに関連付けて、インターフェイスエンドポイントを介して Amazon EKS へのトラフィックを制御することもできます。詳細については、Amazon VPC ドキュメントの「[Control access to VPC endpoints using endpoint policies](#)」を参照してください。
- VPC フローログを使用して、インターフェイスエンドポイントを含めたネットワークインターフェイス間で送受信される IP トラフィックに関する情報を取得できます。フローログデータは Amazon CloudWatch または Amazon S3 に発行できます。詳細については「Amazon VPC ユーザーガイド」の「[VPC フローログを使用した IP トラフィックのログ記録](#)」を参照してください。
- Amazon EKS API はインターフェイスエンドポイントがある VPC に接続することで、オンプレミスのデータセンターからアクセスできます。AWS Direct Connect または AWS Site-to-Site VPN を使用して、オンプレミスサイトを VPC に接続できます。
- 他の VPC は AWS トランジットゲートウェイまたは VPC ピア接続を使用してインターフェイスエンドポイントを備えた VPC に接続できます。VPC ピアリングは 2 つの VPC 間のネットワーク接続です。VPC 間または他のアカウントで VPC を使用して VPC ピアリング接続を確立でき

ます。VPC は異なる AWS リージョンの間でも使用できます。ピア接続された VPC 間のトラフィックは AWS ネットワーク上に留まります。トラフィックは公共インターネットを経由しません。トランジット・ゲートウェイは VPC 間で相互接続するために使用できるネットワークの中継ハブです。VPC とトランジット・ゲートウェイ間のトラフィックは AWS グローバルプライベートネットワークに残ります。トラフィックは公共インターネットに公開されません

- 2024 年 8 月より前には Amazon EKS の VPC インターフェイスエンドポイントには `eks.region.amazonaws.com` を使用して IPv4 経由でのみアクセスできました。2024 年 8 月以降に作成された新しい VPC インターフェイスエンドポイントは IPv4 と IPv6 IP アドレスのデュアルスタックと、`eks.region.amazonaws.com` と `eks.region.api.aws` の両方の DNS 名を使用します。
- EKS API の AWS プライベートリンクサポートは、アジアパシフィック (マレーシア) (ap-southeast-5)、アジアパシフィック (タイ) (ap-southeast-7)、メキシコ (中部) (mx-central-1)、アジアパシフィック (台北) (ap-east-2) の AWS リージョンでは利用できません。AWSEKS Pod Identity の eks-auth に対する PrivateLink サポートは、アジア太平洋 (マレーシア) (ap-southeast-5) リージョンで利用できます。

## Amazon EKS 用のインターフェイスエンドポイントを作成します

Amazon VPC コンソールまたは AWS コマンドラインインターフェイス (AWS CLI) を使用して、Amazon EKS のインターフェイスエンドポイントを作成できます。詳細については『AWS プライベートリンク ガイド』の「[VPC エンドポイントを作成](#)」を参照してください。

次のサービス名を使用して Amazon EKS のインターフェイスエンドポイントを作成します:

- EKS API

```
com.amazonaws.region-code.eks
```

- EKS Auth API (EKS Pod Identity)

```
com.amazonaws.region-code.eks-auth
```

プライベート DNS 機能は Amazon EKS や他の AWS サービスのインターフェイスエンドポイントを作成するときにデフォルトで有効になります。プライベート DNS 機能を使用するには次の VPC 属性が `true: enableDnsHostnames` および `enableDnsSupport` に設定されているようにする必要があります。詳細については「Amazon VPC ユーザーガイド」の「[VPC の DNS 属性の表示と更](#)

[新](#)」を参照してください。インターフェイスエンドポイントでプライベート DNS 機能を有効にすると、次のことができます。

- デフォルトのリージョン DNS 名を使用して Amazon EKS にあらゆる API リクエストを行うことができます。2024 年 8 月以降、Amazon EKS API の新しい VPC インターフェイスエンドポイントには 2 つのデフォルトのリージョンレベルの DNS 名があり、IP アドレスタイプに dualstack を選択できます。最初の DNS 名は eks.**region**.api.aws であり、これはデュアルスタックです。IPv4 アドレスと IPv6 アドレスの両方に解決されます。2024 年 8 月より前には Amazon EKS は IPv4 アドレスのみに解決される eks.**region**.amazonaws.com のみを使用していました。既存の VPC インターフェイスエンドポイントで IPv6 とデュアルスタック IP アドレスを使用する場合は IP アドレスの dualstack タイプを使用するようにエンドポイントを更新できますが、eks.**region**.amazonaws.com DNS 名のみが含まれます。この設定では既存のエンドポイントはその名前が IPv4 と IPv6 の両方の IP アドレスをポイントするように更新されます。API のリストについては「Amazon EKS API リファレンス」の「[アクション](#)」を参照してください。
- EKS API を呼び出すアプリケーションに変更を加える必要はありません。

ただし、AWS CLI でデュアルスタックエンドポイントを使用するには「AWS SDK およびツール リファレンスガイド」の「[Dual-stack and FIPS endpoints](#)」の設定を参照してください。

- Amazon EKS のデフォルトサービスエンドポイントへの呼び出しはプライベート AWS ネットワーク経由でインターフェイスエンドポイントを介して自動的にルーティングされます。

## Amazon EKS クラスターの耐障害性の説明

AWS のグローバルインフラストラクチャは、AWS リージョンとアベイラビリティーゾーンを中心として構築されています。AWS リージョンには、低レイテンシー、高いスループット、そして高度の冗長ネットワークで接続されている複数の物理的に独立し隔離されたアベイラビリティーゾーンがあります。アベイラビリティーゾーンでは、アベイラビリティーゾーン間で中断せずに、自動的にフェイルオーバーするアプリケーションとデータベースを設計および運用することができます。アベイラビリティーゾーンは、従来の単一または複数のデータセンターインフラストラクチャよりも可用性、耐障害性、および拡張性が優れています。

Amazon EKS は、複数の AWS アベイラビリティーゾーンで Kubernetes コントロールプレーンインスタンスを実行およびスケールリングして、高可用性を実現します。Amazon EKS は、負荷に基づいてコントロールプレーンインスタンスを自動的にスケールリングします。また、異常なコントロールプレーンインスタンスを自動的に検出して置換し、そのコントロールプレーンに自動的にパッチを適用します。バージョン更新を開始すると、Amazon EKS によってコントロールプレーンが自動的に更新され、更新中のコントロールプレーンの高可用性が維持されます。

このコントロールプレーンは、少なくとも 2 つの API サーバーインスタンスと、(1 つの AWS リージョンで 3 つのアベイラビリティゾーンにまたがっている) 3 つの etcd インスタンスを含みます。Amazon EKS:

- コントロールプレーンインスタンスの負荷をアクティブに監視し、高パフォーマンスを確保するために、それらのインスタンスを自動的にスケーリングします。
- 異常なコントロールプレーンインスタンスを自動的に検出して置換します。それらのコントロールプレーンインスタンスは、必要に応じて AWS リージョン内のアベイラビリティゾーンで再起動されます。
- 高可用性を維持するために、AWS リージョンのアーキテクチャを活用します。そのため、Amazon EKS では [API サーバーエンドポイント向けの可用性の SLA](#) をご利用いただけます。

AWS リージョンとアベイラビリティゾーンの詳細については、「[AWSグローバルインフラストラクチャ](#)」を参照してください。

## Kubernetes のセキュリティに関する考慮事項

以下は、Amazon EKS クラスターの Kubernetes に影響する、クラウドのセキュリティに関する考慮事項です。Kubernetes のセキュリティコントロールとプラクティスの詳細については、Kubernetes ドキュメントの「[Cloud Native Security and Kubernetes](#)」を参照してください。

### トピック

- [Kubernetes 証明書でワークロードを保護する](#)
- [Amazon EKS 作成の RBAC ロールおよびユーザーの説明](#)
- [既存のクラスターで KMS を使用して Kubernetes シークレットを暗号化する](#)
- [Amazon EKS ポッドで AWS 秘密マネジャー シークレットを使用する](#)
- [すべての Kubernetes API データを対象とするデフォルトのエンベロープ暗号化](#)

## Kubernetes 証明書でワークロードを保護する

Kubernetes Certificates API により、[X.509](#) 認証情報のプロビジョニングを自動化します。この API には、Kubernetes API クライアントが [X.509 証明書](#) を認証機関 (CA) に要求し、取得するためのコマンドラインインターフェイスが備わっています。CertificateSigningRequest (CSR) リソースを使用して、示された署名者に証明書への署名を要求できます。要求は、署名前に承認または拒



否されます。Kubernetes では、組み込み型の署名者とカスタムの署名者の両方がサポートされており、動作が詳細に定義されています。この構成により、クライアントは自分の CSR に何が起こるかを予測できます。証明書の署名の詳細については、「[signing requests](#)」(リクエストへの署名)を参照してください。

組み込みの署名者の中には、`kubernetes.io/legacy-unknown` も含まれます。CSR リソースの `v1beta1` API では、この未知のレガシー署名者を拒否していません。ただし、CSR の安定版 `v1` APIでは、`signerName` を `kubernetes.io/legacy-unknown` に設定することはできません。

クラスターでの証明書の生成に Amazon EKS CA を使用する場合は、カスタム署名者を使用する必要があります。CSR `v1` API バージョンを使用して新しい証明書を生成するには、既存のマニフェストと API クライアントを移行する必要があります。古い `v1beta1` API で作成された既存の証明書の有効性は保たれ、その有効期限が切れるまで機能します。これには以下が含まれます。

- 信頼のディストリビューション: なし Kubernetes クラスターには、この署名者のための標準的な信頼またはディストリビューションはありません。
- 許可された対象: 任意
- 許可された `x509` 拡張機能: `subjectAltName` とキーの使用の拡張機能を受け入れ、他の拡張機能を破棄します
- 許可された鍵の使用法: `["key encipherment", "digital signature", "server auth"]` (鍵暗号化、デジタル署名、サーバー認証) 以外の使用法を含めないでください。

#### Note

クライアント証明書署名はサポートされていません。

- 有効期限/証明書のライフタイム: 1 年 (デフォルトかつ最大の値)
- CA ビットの許可/不許可: 不許可

## signerName を使用した CSR 生成の例

次の手順では、`signerName: beta.eks.amazonaws.com/app-serving` を使用した DNS 名 `myserver.default.svc` の配信用証明書の生成方法を示します。これは、実際の環境を構築するためのガイドとして使用できます。

1. `openssl genrsa -out myserver.key 2048` コマンドを実行して、RSA プライベートキーを生成します。

```
openssl genrsa -out myserver.key 2048
```

2. 次のコマンドを実行して、証明書リクエストを生成します。

```
openssl req -new -key myserver.key -out myserver.csr -subj "/CN=myserver.default.svc"
```

3. CSR リクエストの base64 値を生成し、後の手順で使用するために変数に格納します。

```
base_64=$(cat myserver.csr | base64 -w 0 | tr -d "
")
```

4. 次のコマンドを実行して、mycsr.yaml という名前のファイルを作成します。次の例では、beta.eks.amazonaws.com/app-serving が signerName に使用されています。

```
cat >mycsr.yaml <<EOF
apiVersion: certificates.k8s.io/v1
kind: CertificateSigningRequest
metadata:
 name: myserver
spec:
 request: $base_64
 signerName: beta.eks.amazonaws.com/app-serving
 usages:
 - digital signature
 - key encipherment
 - server auth
EOF
```

5. CSR を送信します。

```
kubectl apply -f mycsr.yaml
```

6. 配信用証明書を承認します。

```
kubectl certificate approve myserver
```

7. 証明書が発行されたことを確認します。

```
kubectl get csr myserver
```

出力例は次のとおりです。

| NAME     | AGE   | SIGNERNAME                         | REQUESTOR        | CONDITION        |
|----------|-------|------------------------------------|------------------|------------------|
| myserver | 3m20s | beta.eks.amazonaws.com/app-serving | kubernetes-admin | Approved, Issued |

## 8. 発行された証明書をエクスポートします。

```
kubectl get csr myserver -o jsonpath='{.status.certificate}' | base64 -d > myserver.crt
```

## Amazon EKS 作成の RBAC ロールおよびユーザーの説明

Kubernetes クラスターを作成すると、Kubernetes が正常に機能するようにそのクラスターに複数のデフォルト Kubernetes ID が作成されます。Amazon EKS はデフォルトコンポーネントごとに Kubernetes ID を作成します。ID はクラスターコンポーネントの Kubernetes ロールベースの承認制御 (RBAC) を提供します。詳細については Kubernetes ドキュメントの「[RBAC 認可を使用する](#)」を参照してください。

オプションの[アドオン](#)をクラスターにインストールすると、クラスターに追加の Kubernetes ID が追加されることがあります。このトピックで扱われていない ID の詳細については「アドオンのドキュメント」を参照してください。

AWS Management Console または `kubectl` コマンドラインツールを使用して、クラスターで Amazon EKS が作成した Kubernetes ID のリストを表示できます。すべてのユーザー ID はお客様に対して、Amazon CloudWatch を通じて利用可能な kube 監査ログに表示されます。

## AWS Management Console

### 前提条件

使用する [IAM プリンシパル](#) には「[必要なアクセス許可](#)」に記載されている許可が必要です。

AWS Management Console を使用して Amazon EKS で作成された ID を表示するには

1. [アマゾン EKS コンソール](#)を開きます。
2. [Clusters] (クラスター) のリストで、表示したい ID を含んでいるクラスターを選択してください。
3. [リソース] タブを選択してください。



4. [Resource types] (リソースタイプ) で [Authorization] (承認) を 選択してください。
5. [クラスター役割]、[クラスター役割バインディング]、[役割]、または [バインディング役割] を選択してください。[eks] というプレフィックスが付いたリソースはすべて、Amazon EKS によって作成されます。Amazon EKS で作成される追加の ID リソースは次のとおりです：
  - [aws-node] という名前の [クラスター役割] と [クラスター役割バインディング]。[aws-node] リソースは Amazon EKS がすべてのクラスターにインストールする [Amazon VPC CNI plugin for Kubernetes](#) をサポートします。
  - [vpc-resource-controller-role] という名前の [クラスター役割] および [vpc-resource-controller-rolebinding] という名前の [クラスター役割バインディング]。これらのリソースは Amazon EKS がすべてのクラスターにインストールする「[Amazon VPC resource controller](#)」(Amazon VPC リソースコントローラー) サポートします。

コンソールに表示されるリソースに加えて、次の特別なユーザー ID がクラスターに存在しますが、クラスターの設定には表示されません：

- **eks:cluster-bootstrap** - クラスターブートストラップ中の kubectl 操作に使用されます。
  - **eks:support-engineer** - クラスター管理操作に使用されます。
6. 特定のリソースを選択すると、そのリソースの詳細が表示されます。デフォルトでは情報は [Structured view] (構造化ビュー) で表示されます。詳細ページの右上隅で、[Raw View] (生のビュー) 選択すると、リソースの情報をすべて表示できます。

## Kubectl

### 前提条件

クラスター上の Kubernetes リソースを一覧表示するために使用するエンティティ (AWS Identity and Access Management (IAM) または OpenID Connect (OIDC)) は IAM または OIDC ID プロバイダーによって認証される必要があります。エンティティにはそのエンティティに使用させたいクラスター上の、Role、ClusterRole、RoleBinding、および ClusterRoleBinding リソースに対して Kubernetes get および list 動詞を使用するアクセス許可が付与されている必要があります。IAM エンティティにクラスターへのアクセスを付与する方法の詳細については「[the section called “Kubernetes API アクセス”](#)」を参照してください。独自の OIDC プロバイダーによって認証されたエンティティにクラスターへのアクセスを付与する方法の詳細については「[the section called “OIDC プロバイダーをリンクする”](#)」を参照してください。

**kubectl** を使用して Amazon EKS で作成された ID を表示するには

表示するリソースのタイプのコマンドを実行します。[eks] というプレフィックスが付いたすべての返されるリソースは Amazon EKS によって作成されます。コマンドからの出力で返されるリソースに加えて、次の特別なユーザー ID がクラスターに存在しますが、クラスターの設定には表示されません：

- **eks:cluster-bootstrap** - クラスターブートストラップ中の **kubectl** 操作に使用されます。
- **eks:support-engineer** - クラスター管理操作に使用されます。

ClusterRoles - ClusterRoles はクラスターにスコープが設定されているため、ロールに付与されたすべてのアクセス許可はクラスター上の任意の Kubernetes 名前空間内のリソースに適用されます。

次のコマンドは、クラスター上で Amazon EKS によって作成されたすべての Kubernetes ClusterRoles を返します。

```
kubectl get clusterroles | grep eks
```

出力で返されるのはプレフィックスが付けられた ClusterRoles の他に、次の ClusterRoles が存在します。

- **aws-node** - この ClusterRole はアマゾン EKS がすべてのクラスターにインストールする [Amazon VPC CNI plugin for Kubernetes](#), をサポートします。
- **vpc-resource-controller-role** - この ClusterRole はアマゾン EKS がすべてのクラスターにインストールする「[アマゾン VPC resource controller](#)」(アマゾン VPC リソースコントローラー) サポートします。

ClusterRole の仕様を確認するには次のコマンドの **eks:k8s-metrics** を前のコマンドの出力で返された ClusterRole に置き換えます。次の例では **eks:k8s-metrics** ClusterRole の仕様を返します。

```
kubectl describe clusterrole eks:k8s-metrics
```

出力例は次のとおりです。

```
Name: eks:k8s-metrics
```

```
Labels: <none>
Annotations: <none>
PolicyRule:
 Resources Non-Resource URLs Resource Names Verbs

 ["/metrics"] [] [get]
 endpoints [] [] [list]
 nodes [] [] [list]
 pods [] [] [list]
 deployments.apps [] [] [list]
```

クラスター役割Bindings - ClusterRoleBindings はクラスターを対象としています。

次のコマンドは、クラスター上で Amazon EKSによって作成されたすべての Kubernetes ClusterRoleBindings を返します。

```
kubectl get clusterrolebindings | grep eks
```

出力で返される ClusterRoleBindings 以外に、次の ClusterRoleBindings が存在します。

- **aws-node** – この ClusterRoleBinding はAmazon EKS がすべてのクラスターにインストールする [Amazon VPC CNI plugin for Kubernetes](#), をサポートします。
- **vpc-resource-controller-rolebinding** - この ClusterRoleBinding はアマゾン EKS がすべてのクラスターにインストールする「[アマゾン VPC resource controller](#)」(アマゾン VPC リソースコントローラー) サポートします。

ClusterRoleBinding の仕様を確認するには次のコマンドの *eks:k8s-metrics* を前のコマンドの出力で返された ClusterRoleBinding に置き換えます。次の例では*eks:k8s-metrics* ClusterRoleBinding の仕様を返します。

```
kubectl describe clusterrolebinding eks:k8s-metrics
```

出力例は次のとおりです。

```
Name: eks:k8s-metrics
Labels: <none>
Annotations: <none>
Role:
 Kind: ClusterRole
 Name: eks:k8s-metrics
```

## Subjects:

| Kind | Name            | Namespace |
|------|-----------------|-----------|
| ---- | ----            | -----     |
| User | eks:k8s-metrics |           |

Roles - Roles は Kubernetes 名前空間にスコープされます。Roles で作成されたすべての アマゾン EKS は kube-system 名前空間にスコープされます。

次のコマンドは、クラスター上で Amazon EKSによって作成されたすべての Kubernetes Roles を返します。

```
kubectl get roles -n kube-system | grep eks
```

Role の仕様を確認するには、次のコマンドの *eks:k8s-metrics* を、前のコマンドの出力で返された Role の名前に置き換えます。次の例では *eks:k8s-metrics* Role の仕様を返します。

```
kubectl describe role eks:k8s-metrics -n kube-system
```

出力例は次のとおりです。

```
Name: eks:k8s-metrics
Labels: <none>
Annotations: <none>
PolicyRule:
 Resources Non-Resource URLs Resource Names Verbs

 daemonsets.apps [] [aws-node] [get]
 deployments.apps [] [vpc-resource-controller] [get]
```

RoleBindings - RoleBindings は Kubernetes 名前空間にスコープされます。RoleBindings で作成されたすべての アマゾン EKS は kube-system 名前空間にスコープされます。

次のコマンドは、クラスター上で Amazon EKSによって作成されたすべての Kubernetes RoleBindings を返します。

```
kubectl get rolebindings -n kube-system | grep eks
```

RoleBinding の仕様を確認するには次のコマンドの *eks:k8s-metrics* を前のコマンドの出力で返された RoleBinding に置き換えます。次の例では *eks:k8s-metrics* RoleBinding の仕様を返します。

```
kubectl describe rolebinding eks:k8s-metrics -n kube-system
```

出力例は次のとおりです。

```
Name: eks:k8s-metrics
Labels: <none>
Annotations: <none>
Role:
 Kind: Role
 Name: eks:k8s-metrics
Subjects:
 Kind Name Namespace
 ---- --- -
 User eks:k8s-metrics
```

## 既存のクラスターで KMS を使用して Kubernetes シークレットを暗号化する

### Important

この手順の対象となるのは、Kubernetes バージョン 1.27 以前を実行している EKS クラスターのみです。Kubernetes バージョン 1.28 以降を実行している場合、デフォルトでは Kubernetes シークレットはエンベロープ暗号化で保護されます。詳細については、「[the section called “すべての Kubernetes API データを対象とするデフォルトのエンベロープ暗号化”](#)」を参照してください。

[シークレット暗号化](#)を有効にすると、選択した AWS KMS キーを使用して Kubernetes シークレットが暗号化されます。KMS キーは次の条件を満たす必要があります：

- 対称
- データの暗号化と復号が可能
- クラスターと同じ AWS リージョンに作成
- KMS キーが別のアカウントで作成された場合、[IAM プリンシパル](#)にはその KMS キーへのアクセス権が必要となります。

詳細については「[AWS キーマネジメントサービス開発者ガイド](#)」の「[他のアカウントの IAM プリンシパルが KMS キーを使用することを許可する](#)」を参照してください。

 Warning

一度有効化したシークレット暗号化は無効化できません。このアクションを元に戻すことはできません。

## eksctl

この手順の対象となるのは、Kubernetes バージョン 1.27 以前を実行している EKS クラスターのみです。詳細については、「[the section called “すべての Kubernetes API データを対象とするデフォルトのエンベロープ暗号化”](#)」を参照してください。

暗号化は次の 2 つの方法のいずれかで有効にできます：

- 1 つのコマンドでクラスターに暗号化を追加します。

シークレットを自動的に再暗号化するには次のコマンドを実行してください。

```
eksctl utils enable-secrets-encryption \
 --cluster my-cluster \
 --key-arn arn:aws:kms:region-code:account:key/key
```

シークレットの自動的な再暗号化をオプトアウトするには次のコマンドを実行してください。

```
eksctl utils enable-secrets-encryption
 --cluster my-cluster \
 --key-arn arn:aws:kms:region-code:account:key/key \
 --encrypt-existing-secrets=false
```

- kms-cluster.yaml ファイルを使用してクラスターに暗号化を追加します。

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

metadata:
 name: my-cluster
 region: region-code
```

```
secretsEncryption:
 keyARN: arn:aws:kms:region-code:account:key/key
```

シークレットに自動的な再暗号化をさせるには次のコマンドを実行してください。

```
eksctl utils enable-secrets-encryption -f kms-cluster.yaml
```

シークレットの自動的な再暗号化をオプトアウトするには次のコマンドを実行してください。

```
eksctl utils enable-secrets-encryption -f kms-cluster.yaml --encrypt-existing-secrets=false
```

## AWS Management Console

- この手順の対象となるのは、Kubernetes バージョン 1.27 以前を実行している EKS クラスターのみです。詳細については、「[the section called “すべての Kubernetes API データを対象とするデフォルトのエンベロープ暗号化”](#)」を参照してください。
- [Amazon EKS コンソール](#)を開きます。
- KMS 暗号化を追加するクラスターを選択してください。
- [Overview] (概要 タブを選択してください (これはデフォルトで選択されています))。
- [秘密の暗号化] (シークレットの暗号化 セクションまでスクロールダウンし、[Enable] (有効化 ボタンを選択してください。
- ドロップダウンリストからキーを選択し、[Enable] (有効化 ボタンを選択してください。キーが一覧表示されていない場合は最初にキーを作成する必要があります。詳細については「[キーの作成](#)」を参照してください。
- [Confirm] (確認) ボタンを選択して、指定したキーを使用します。

## AWS CLI

- この手順の対象となるのは、Kubernetes バージョン 1.27 以前を実行している EKS クラスターのみです。詳細については、「[the section called “すべての Kubernetes API データを対象とするデフォルトのエンベロープ暗号化”](#)」を参照してください。
- 次の AWS CLI コマンドを使用して、[シークレット暗号化](#)設定をクラスターに関連付けます。### は実際に使用する値に置き換えます。

```
aws eks associate-encryption-config \
 --cluster-name my-cluster \
```

```
--encryption-config '[{"resources":["secrets"],"provider":
{"keyArn":"arn:aws:kms:region-code:account:key/key"}}]'
```

出力例は次のとおりです。

```
{
 "update": {
 "id": "3141b835-8103-423a-8e68-12c2521ffa4d",
 "status": "InProgress",
 "type": "AssociateEncryptionConfig",
 "params": [
 {
 "type": "EncryptionConfig",
 "value": "[{\"resources\":[\"secrets\"],\"provider\":{\"keyArn\":
\\\"arn:aws:kms:region-code:account:key/key\\\"}}]"
 }
],
 "createdAt": 1613754188.734,
 "errors": []
 }
}
```

- c. 次のコマンドを使用すると、暗号化更新のステータスをモニタリングできます。前の出力で返された特定の cluster name と update ID を使用します。Successful ステータスが表示されると、更新は完了です。

```
aws eks describe-update \
 --region region-code \
 --name my-cluster \
 --update-id 3141b835-8103-423a-8e68-12c2521ffa4d
```

出力例は次のとおりです。

```
{
 "update": {
 "id": "3141b835-8103-423a-8e68-12c2521ffa4d",
 "status": "Successful",
 "type": "AssociateEncryptionConfig",
 "params": [
 {
 "type": "EncryptionConfig",
```



```
 "value": "[{\"resources\":[\"secrets\"],\"provider\":{\"keyArn\":
 \"arn:aws:kms:region-code:account:key/key\"}}]"
 }
],
 "createdAt": 1613754188.734>,
 "errors": []
}
}
```

- d. クラスターで暗号化が有効になっていることを確認するには `describe-cluster` コマンドを実行してください。レスポンスの内容は文字列 `EncryptionConfig` です。

```
aws eks describe-cluster --region region-code --name my-cluster
```

クラスターで暗号化を有効にしたら、既存のすべてのシークレットを新しいキーで暗号化する必要があります：

#### Note

`eksctl` を使用する場合、シークレットを自動的に再暗号化することをオプトアウトした場合にのみ、次のコマンドを実行する必要があります。

```
kubectl get secrets --all-namespaces -o json | kubectl annotate --overwrite -f - kms-
encryption-timestamp="time value"
```

#### Warning

既存のクラスターで [シークレット暗号化](#) を有効にし、使用する KMS キーが削除されている場合はこのクラスターを復旧する手段はありません。KMS キーを削除すると、クラスターは永続的にパフォーマンスが低下した状態になります。詳細については「[AWSKMS キーを削除する](#)」を参照してください。

#### Note

`create-key` コマンドではデフォルトで [対称暗号化 KMS キー](#) が作成されます。この際にはアカウントのルート管理者に AWS KMS アクションとリソースへのアクセスを許可する、

キーポリシーが使用されます。アクセス許可の範囲を絞り込む場合は `create-cluster` API を呼び出すプリンシパルのポリシーで、`kms:DescribeKey` および `kms:CreateGrant` アクションが許可されていることを確認します。

KMS エンベロープ暗号化を使用するクラスターの場合、`kms:CreateGrant` アクセス許可が必要です。条件 `kms:GrantIsForAWSResource` はクラスター作成 アクションでサポートされていないため、クラスター作成 を実行するユーザーの `kms:CreateGrant` アクセス許可を制御する KMS ポリシーで使わないでください。

## Amazon EKS ポッドで AWS 秘密マネジャー シークレットを使用する

Secrets Manager のシークレットとパラメータストアのパラメータを、Amazon EKS Pod にマウントされたファイルとして表示するには、[Kubernetes の Secrets Store CSI ドライバー](#) 向けの AWS Secrets and Configuration Provider (ASCP) を使用します。

ASCP を使用すると、Secrets Manager でシークレットを保存および管理し、Amazon EKS で実行されているワークロードを介して、そのシークレットを取得することが可能です。IAM ロールとポリシーを使用して、シークレットへのアクセスをクラスターにある特定の Kubernetes Pod に制限することができます。ASCP は、Pod Identity を取得して IAM ロールと交換します。ASCP が Pod の IAM ロールを引き受けると、Secrets Manager からそのロールに対して認可されているシークレットを取得できます。

シークレットに 秘密マネジャー の自動ローテーションを使用する場合は シークレットストア CSI ドライバーのローテーション調整機能を使用して、秘密マネジャー から最新のシークレットを取得することもできます。

### Note

AWS Fargate (Fargate) ノードグループはサポートされていません。

詳細についてはAWS 秘密マネジャー ユーザーガイドの「[Using 秘密マネジャー secrets in Amazon EKS](#)」を参照してください。

# すべての Kubernetes API データを対象とするデフォルトのエンベロープ暗号化

Amazon Elastic Kubernetes Service (Amazon EKS) はデフォルトでエンベロープ暗号化を備えており、Kubernetes バージョン 1.28 以降を実行している EKS クラスター内のすべての Kubernetes API データが暗号化の対象となります。

エンベロープ暗号化は、Kubernetes API サーバーで保存するデータを保護します。例えば、エンベロープ暗号化は ConfigMaps といった Kubernetes クラスターの設定に適用されます。一方、ノードや EBS ボリューム上のデータにはエンベロープ暗号化は適用されません。EKS は以前には Kubernetes シークレットの暗号化をサポートしていましたが、現在ではこのエンベロープ暗号化の対象がすべての Kubernetes API データへと広がっています。

Kubernetes アプリケーションの多層防御を実装するマネージドサービスがデフォルトで提供されるため、お客様側で対策を講じる必要はありません。

Amazon EKS は、[Kubernetes KMS プロバイダー v2](#) で AWS [Key Management Service \(KMS\)](#) を使用して、[Amazon Web Services 所有のキー](#)と、AWS KMS から独自の[カスタマーマネージドキー \(CMK\)](#)を取り込めるオプションにより、このセキュリティ層を追加しています。

## エンベロープ暗号化について

エンベロープ暗号化とは、データストア (etcd) への送信に先だってプレーンテキストデータをデータ暗号化キー (DEK) で暗号化し、次にリモートで一元管理される KMS システム (AWS KMS) に保存されているルート KMS キーで DEK を暗号化するプロセスです。これは多層防御といわれる戦略で、暗号化キー (DEK) でデータを保護し、さらにその DEK をキー暗号化キー (KEK) という別途安全に保存された暗号化キーで保護してセキュリティ層をもう 1 つ追加することからそう呼ばれています。

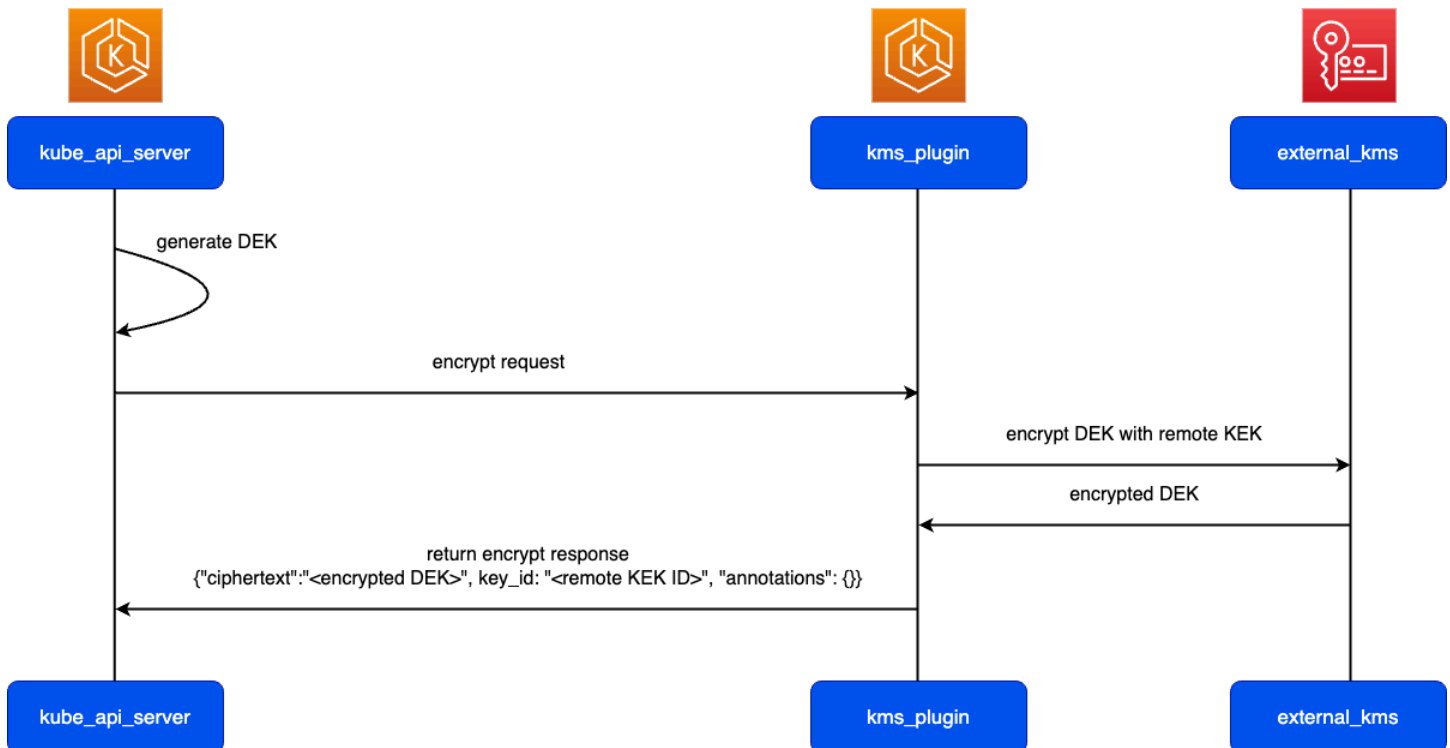
## Amazon EKS が AWS KMS v2 と KMS でデフォルトのエンベロープ暗号化を実現する仕組み

Amazon EKS は、[KMS v2](#) を使用して、マネージド Kubernetes コントロールプレーン内のすべての API データをデフォルトでエンベロープ暗号化してから [etcd](#) データベースに保存します。起動時に、クラスター API サーバーはシークレットシードからデータ暗号化キー (DEK) を生成し、それをランダムに生成されたデータと組み合わせて使用します。同じく起動時に、API サーバーは KMS プラグインへの呼び出しを行い、AWS KMS からリモートのキー暗号化キー (KEK) を使用して DEK シードを暗号化します。これは、API サーバーの起動時と KEK のローテーション時に実行される 1

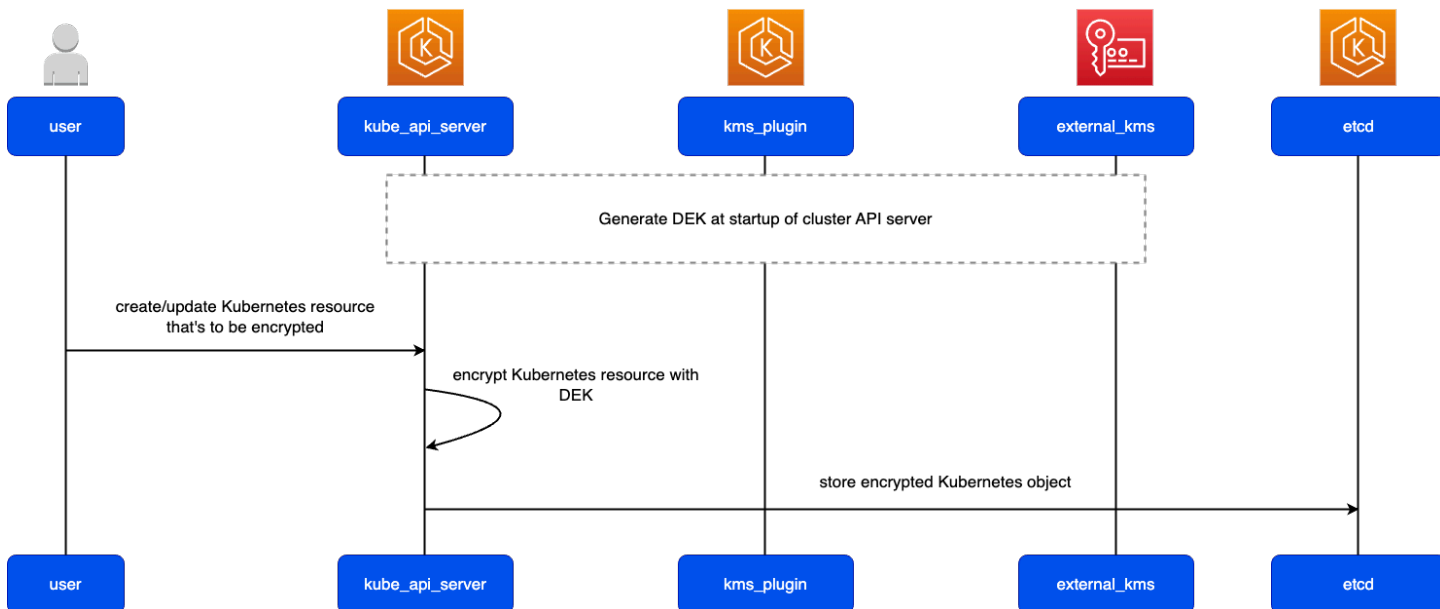
一回限りの呼び出しです。暗号化された DEK シードは API サーバーにキャッシュされます。API サーバーは、そのキャッシュした DEK シードを使用して、キー導出関数 (KDF) に基づいて 1 回限り使用される DEK を別途生成します。こうして生成した各 DEK を Kubernetes リソースごとに 1 回のみ使用して暗号化し、その後 etcd に保存します。KMS v2 で暗号化してキャッシュされた DEK シードを使用すると、API サーバーで Kubernetes リソースを暗号化するプロセスのパフォーマンスとコスト効率が向上します。

この KEK はデフォルトでは AWS によって所有されているものですが、必要に応じて AWS KMS から独自のものを取り込むこともできます。

以下の図は、API サーバーの起動時に行われる DEK の生成と暗号化を示しています。



以下の概要図は、etcd に保存される前に行われる Kubernetes リソースの暗号化を示しています。



## よくある質問

デフォルトのエンベロープ暗号化によって、EKS クラスターのセキュリティ体制はどのように改善されますか？

メタデータとカスタマーコンテンツが暗号化されない対象領域と期間を減らします。デフォルトのエンベロープ暗号化により、メタデータとカスタマーコンテンツは etcd に保存される前に kube-apiserver のメモリ内で一時的に暗号化されていない状態になるだけです。kube-apiserver のメモリは、[Nitro システム](#)によって安全に保護されます。Amazon EKS で [Nitro ベースの EC2 インスタンス](#)が使用されるのは、マネージド Kubernetes コントロールプレーンに限られます。これらのインスタンスでは、システムや個人がメモリにアクセスできないよう、しっかりとセキュリティ管理が設計されています。

この機能を利用するためには、Kubernetes のどのバージョンを実行する必要がありますか？

デフォルトのエンベロープ暗号化を有効にするには、Kubernetes バージョン 1.28 以降で Amazon EKS クラスターを実行している必要があります。

この機能をサポートしていない Kubernetes クラスターバージョンを実行していても、データは引き続き安全に保護されますか？

はい。AWS では、[セキュリティを最優先事項としています](#)。Amazon のデジタルトランスフォーメーションとイノベーションはすべてセキュリティが最も高い運用プラクティスに基づいており、Amazon はそのレベルアップに真摯に取り組んでいます。

実行中の Kubernetes バージョンに関係なく、etcd に保存されているすべてのデータが EKS クラスターごとにディスクレベルで暗号化されます。EKS では、ルートキーを使用してボリューム暗号化キーを生成し、それを EKS サービスで管理しています。また、あらゆる Amazon EKS クラスターが、クラスター固有の仮想マシンを使用して、分離された VPC で実行されます。このアーキテクチャに加え、運用セキュリティにまつわる Amazon のプラクティスにより、Amazon EKS は SOC 1、2、3、PCI-DSS、ISO、HIPAA 適格性といった[コンプライアンス評価と基準をいくつも実現しています](#)。こうしたコンプライアンス評価と基準は、デフォルトのエンベロープ暗号化の有無にかかわらず、すべての EKS クラスターに対して維持されています。

Amazon EKS では、エンベロープ暗号化はどのように機能しますか？

起動時に、クラスター API サーバーはシークレットシードからデータ暗号化キー (DEK) を生成し、それをランダムに生成されたデータと組み合わせて使用します。同じく起動時に、API サーバーは KMS プラグインへの呼び出しを行い、AWS KMS からリモートのキー暗号化キー (KEK) を使用して DEK を暗号化します。これは、API サーバーの起動時と KEK のローテーション時に実行される 1 回限りの呼び出しです。暗号化された DEK シードは API サーバーにキャッシュされます。API サーバーは、そのキャッシュした DEK シードを使用して、キー導出関数 (KDF) に基づいて 1 回限り使用される DEK を別途生成します。こうして生成した各 DEK を Kubernetes リソースごとに 1 回のみ使用して暗号化し、その後 etcd に保存します。

AWS KMS 統合のヘルスと通常の機能性を検証するために、API サーバーから追加で呼び出しが行われることに留意してください。こうした追加のヘルスチェックは、AWS CloudTrail に表示されます。

この機能を EKS クラスターで動作させるために何かを実行したり、アクセス許可を変更したりする必要はありますか？

いいえ。特に何もする必要はありません。Amazon EKS のエンベロープ暗号化は今ではデフォルトの設定となり、Kubernetes バージョン 1.28 以降を実行しているすべてのクラスターで有効になっています。AWS KMS 統合は、AWS 側で管理する Kubernetes API サーバーによって確立されます。つまり、クラスターで KMS 暗号化を使い始めるためにアクセス許可を設定する必要はありません。

クラスターでデフォルトのエンベロープ暗号化が有効になっているかどうかを確認するには、どうすればよいですか？

独自の CMK の使用に切り替えると、クラスターに関連付けられている KMS キーの ARN が表示されます。また、クラスターの CMK の使用に関連する AWS CloudTrail イベントログを確認することもできます。

クラスターが AWS 所有のキーを使用している場合には、その情報が EKS コンソールに詳しく表示されます (キーの ARN を除く)。

Amazon EKS でデフォルトのエンベロープ暗号化に使用される AWS 所有のキーに AWS からアクセスできますか？

いいえ。AWS では Amazon EKS のセキュリティ管理は厳しく、etcd データベース内のデータの保護に使用されるプレーンテキストの暗号化キーには誰もアクセスできません。こうしたセキュリティ対策は、AWS 所有の KMS キーにも適用されます。

デフォルトのエンベロープ暗号化は既存の EKS クラスターで有効になっていますか？

Kubernetes バージョン 1.28 以降で Amazon EKS クラスターを実行している場合は、すべての Kubernetes API データに対してエンベロープ暗号化が有効になっています。既存のクラスターの場合、Amazon EKS は `eks:kms-storage-migrator` RBAC ClusterRole を使用して、以前に etcd でエンベロープ暗号化されていなかったデータをこの新しい暗号化状態に移行します。

これは、EKS クラスターに対してシークレットのエンベロープ暗号化を既に有効にしていた場合にはどうなりますか？

Kubernetes シークレットをエンベロープ暗号化するために使用されたカスタマーマネージドキー (CMK) が KMS に既にある場合は、その同じキーがクラスター内のすべての Kubernetes API データタイプをエンベロープ暗号化するための KEK として使用されます。

デフォルトのエンベロープ暗号化を有効にして EKS クラスターを実行すると、追加でコストが発生しますか？

デフォルトのエンベロープ暗号化に [Amazon Web Services 所有のキー](#)を使用している場合、マネージド Kubernetes コントロールプレーンに関連して追加でコストが発生することはありません。デフォルトでは、Kubernetes バージョン 1.28 以降を実行しているすべての EKS クラスターが [Amazon Web Services 所有のキー](#)を使用します。ただし、独自の AWS KMS キーを使用した場合は、通常の [KMS 料金](#)が適用されます。

独自の AWS KMS キーを使用してクラスター内の Kubernetes API データを暗号化するには、どのくらいのコストがかかりますか？

作成または KMS にインポートするカスタムキーを保しておくには、1 か月あたり 1 ドルの支払いが発生します。KMS では、暗号化と復号をリクエストすると課金されます。アカウントごとに 1 か月あたり 20,000 件のリクエストという無料利用枠があり、この無料利用枠を超えると 1 か月あたり 10,000 件のリクエストごとに 0.03 ドルの支払いが発生します。これはアカウントで使用したすべて



の KMS に適用されるため、クラスターで独自の AWS KMS キーを使用した場合のコストは、このキーをアカウント内の他のクラスターや AWS リソースで使った場合の影響を受けます。

カスターマネージドキー (CMK) を使用してシークレットだけでなくすべての Kubernetes API データをエンベロープ暗号化すると、KMS の料金は高くなりますか？

いいえ。KMS v2 で実装したことにより、AWS KMS に対する呼び出しの数が大幅に減ります。そのため、EKS クラスターで Kubernetes データが追加で暗号化または復号されたとしても、CMK に関連するコストは削減されます。

既に詳述したように、Kubernetes リソースの暗号化のために生成された DEK シードは、リモート KEK で暗号化されたうえで Kubernetes API サーバーのキャッシュにローカルに保存されます。暗号化された DEK シードが API サーバーのキャッシュ内にはない場合、API サーバーは AWS KMS を呼び出して DEK シードを暗号化します。その後、API サーバーは暗号化された DEK シードをキャッシュして今後 KMS を呼び出さなくてもクラスターで使えるようにします。同じく復号リクエストでも、API サーバーは最初の復号リクエストに対して AWS KMS を呼び出し、その後復号された DEK シードをキャッシュして今後の復号操作に使用します。

詳細については、GitHub で Kubernetes 拡張機能の「[KEP-3299: KMS v2 Improvements](#)」を参照してください。

複数の Amazon EKS クラスターで同じ CMK キーを使用できますか？

はい。キーを再度使用するには、キーの作成時に ARN をクラスターに関連付けることで、キーを同じリージョン内のクラスターにリンクします。ただし、複数の EKS クラスターで同じ CMK を使用している場合は、CMK が任意に無効にされないように必要な対策を講じる必要があります。そうしないと、無効になった CMK が複数の EKS クラスターに関連付けられて、キーによってはクラスターに幅広く影響が及びます。

デフォルトのエンベロープ暗号化が有効になった後で CMK が使用できなくなった場合、EKS クラスターはどうなりますか？

KMS キーを無効にすると、そのキーは[暗号化オペレーション](#)で使えなくなります。既存の CMK にアクセスできないと、API サーバーは新規に作成された Kubernetes オブジェクトを暗号化して保持できないほか、以前に暗号化されて etcd に保存された Kubernetes オブジェクトを復号できなくなります。CMK が無効になると、クラスターはすぐに異常/パフォーマンス低下状態になり、関連付けられた CMK を再び有効にするまで[サービスコミットメント](#)を満たすことができなくなります。

CMK が無効になると、EKS クラスターがパフォーマンス低下状態になり、Kubernetes コントロールプレーンリソースを正常に復元するためには CMK が無効になってから 30 日以内に再び有効にする必要があるとの通知が届きます。



CMK が無効化/削除された場合の影響から EKS クラスターを保護するには、どうすればよいですか？

このような事態の発生から EKS クラスターを保護するには、キー管理者が最小特権の原則に基づく IAM ポリシーを使用して KMS キーオペレーションへのアクセスを管理し、EKS クラスターに関連付けられているキーが任意に無効化または削除されるリスクを軽減する必要があります。また、CMK の状態を通知する [CloudWatch アラーム](#)を設定できます。

CMK を再び有効にすると、EKS クラスターが復元されますか？

EKS クラスターが正常に復元されるように、CMK が無効になってから 30 日以内に再び有効にすることを強くお勧めします。ただし、EKS クラスターが正常に復元されるかどうかは、クラスターが異常/パフォーマンス低下状態のときに自動 Kubernetes アップグレードが発生したために API が大きく変更されたかどうかによっても異なります。

CMK を無効にした後で、EKS クラスターが異常/パフォーマンス低下状態になるのはなぜですか？

EKS コントロールプレーンの API サーバーは、暗号化されて API サーバーのメモリにキャッシュされた DEK キーを使用して、作成/更新操作時にすべてのオブジェクトを暗号化してから etcd に保存します。etcd から既存のオブジェクトを取得するときに、API サーバーはキャッシュされたその同じ DEK キーを使用して、Kubernetes リソースオブジェクトを復号します。CMK を無効にしても、DEK キーが API サーバーのメモリにキャッシュされているため、API サーバーはすぐには影響を受けません。ただし、API サーバーインスタンスを再起動すると、キャッシュされた DEK が保持されないため、暗号化と復号の操作を行うには AWS KMS を呼び出す必要があります。CMK がない場合、このプロセスは KMS\_KEY\_DISABLED エラーコードで失敗し、API サーバーが正常に起動しなくなります。

CMK を削除すると、EKS クラスターはどうなりますか？

EKS クラスターに関連付けられている CMK キーを削除すると、ヘルスが低下して回復の見込みがなくなります。クラスターの CMK がないと、API サーバーは新規の Kubernetes オブジェクトを暗号化して保持できないほか、以前に暗号化されて etcd データベースに保存された Kubernetes オブジェクトを復号できなくなります。EKS クラスターの CMK キーを削除するのは、今後 EKS クラスターを使用する必要はないと確信できるときだけにしてください。

CMK が見つからない (KMS\_KEY\_NOT\_FOUND) 場合やクラスターに関連付けられた CMK の許可が取り消されている (KMS\_GRANT\_REVOKED) 場合は、クラスターを回復できなくなるので注意してください。クラスターのヘルスとエラーコードの詳細については、「[Cluster health FAQs and error codes with resolution paths](#)」を参照してください。

CMK を無効または削除したために EKS クラスターでパフォーマンス低下/異常が発生した場合でも課金されますか？

はい。CMK が無効になると EKS コントロールプレーンは使用できなくなりますが、EKS クラスターに割り当てられた専用のインフラストラクチャリソースはお客様が削除するまで AWS で引き続き稼働します。また、このような状況では Amazon の[サービスコミットメント](#)は適用されません。お客様が自発的にアクションを起こしたこと、あるいは何もアクションを起こさなかったことで、EKS クラスターの通常のヘルスとオペレーションが阻害されたためです。

CMK が無効であるために異常/パフォーマンス低下状態になったときでも、EKS クラスターを自動的にアップグレードできますか？

はい。ただし、クラスターで CMK が無効である場合、30 日以内であれば再び有効にできます。この 30 日の間、Kubernetes クラスターは自動的にアップグレードされません。ただし、この期間が過ぎても CMK を再び有効にしなかった場合、クラスターは次のバージョン (n+1) に自動的にアップグレードされます。これは標準のサポートであり、EKS の Kubernetes バージョンライフサイクルに従った動作です。

クラスターが影響を受けていることに気づいたら、無効になった CMK をすぐに再び有効にすることを強くお勧めします。なお、EKS はこうした影響を受けたクラスターを自動的にアップグレードしますが、正常に回復する保証はありません。特に、クラスターが複数回の自動アップグレードを経験している場合にはそうです。この場合、Kubernetes API に変更が加えられ、API サーバーのブートストラッププロセスが予期しない動作をする可能性があるからです。

KMS キーのエイリアスを使用できますか？

はい。Amazon EKS は、[KMS キーのエイリアスの使用をサポートしています](#)。エイリアスは、[Amazon Web Services KMS キー](#)をわかりやすくした名前です。例えば、エイリアスを使用すると、KMS キーを **1234abcd-12ab-34cd-56ef-1234567890ab** ではなく my-key として参照できます。

独自の Kubernetes バックアップソリューションを使用して、クラスターリソースをバックアップおよび復元できますか？

はい。Kubernetes クラスターのディザスタリカバリ、データ移行、データ保護に Kubernetes バックアップソリューション ([Velero](#) など) を使用できます。API サーバーを介してクラスターリソースにアクセスする Kubernetes バックアップソリューションを実行している場合、アプリケーションが取得するデータは復号されてからクライアントに送られます。そのため、別の Kubernetes クラスターにクラスターリソースを復元できます。

# Amazon EKS Auto Mode におけるセキュリティに関する考慮事項

このトピックでは、Amazon EKS Auto Mode のセキュリティアーキテクチャ、コントロール、ベストプラクティスについて説明します。組織がコンテナ化されたアプリケーションを大規模にデプロイするにつれて、強力なセキュリティ体制を維持することがますます複雑になります。EKS Auto Mode は、自動化されたセキュリティコントロールを実装し、AWS セキュリティサービスと統合して、クラスターのインフラストラクチャ、ワークロード、データを保護します。EKS Auto Mode は、強制ノードライフサイクル管理や自動パッチデプロイなどの組み込みセキュリティ機能を通じて、運用オーバーヘッドを削減しながらセキュリティのベストプラクティスを維持するのに役立ちます。

このトピックを進める前に、基本的な EKS Auto Mode の概念を理解していること、およびクラスターで EKS Auto Mode を有効にするための前提条件を確認しているようにしてください。Amazon EKS セキュリティに関する一般的な情報については、「[セキュリティ](#)」を参照してください。

Amazon EKS Auto Mode は、Amazon EKS の既存のセキュリティ基盤に基づいて構築されると同時に、EC2 マネージドインスタンスに対する追加の自動セキュリティコントロールを導入します。

## API セキュリティと認証

Amazon EKS Auto Mode は、AWS プラットフォームのセキュリティメカニズムを使用して、Amazon EKS API への呼び出しを保護および認証します。

- Kubernetes API へのアクセスは、AWS IAM ID と統合された EKS アクセスエントリを介して保護されます。
  - 詳細については、「[the section called “許可を付与する”](#)」を参照してください。
- お客様は、EKS アクセスエントリの設定を通じて、Kubernetes API エンドポイントへのきめ細かなアクセスコントロールを実装できます。

## ネットワークセキュリティ

Amazon EKS Auto Mode は、複数のネットワークセキュリティレイヤーをサポートしています。

- VPC 統合
  - Amazon Virtual Private Cloud (VPC) 内で動作
  - カスタムの VPC 設定とサブネットレイアウトをサポート
  - クラスターコンポーネント間のプライベートネットワーキングを有効化

- 詳細については、「[Amazon Virtual Private Cloud のセキュリティの責任の管理](#)」を参照してください
- ネットワークポリシー
  - Kubernetes ネットワークポリシーのネイティブサポート
  - 詳細なネットワークトラフィックルールを定義する機能
  - 詳細については、「[the section called “Kubernetes ポリシー”](#)」を参照してください。

## EC2 マネージドインスタンスのセキュリティ

Amazon EKS Auto Mode は、次のセキュリティコントロールを使用して EC2 マネージドインスタンスを操作します。

### EC2 セキュリティ

- EC2 マネージドインスタンスは、Amazon EC2 のセキュリティ機能を維持します。
- EC2 マネージドインスタンスの詳細については、「[Security in Amazon EC2](#)」を参照してください。

### インスタンスのライフサイクル管理

EKS Auto Mode によって運用される EC2 マネージドインスタンスの最大存続期間は 21 日です。Amazon EKS Auto Mode は、この存続期間を超えたインスタンスを自動的に終了します。この存続期間制限は、設定のドリフトを防ぎ、セキュリティ体制を維持するのに役立ちます。

### データ保護

- Amazon EC2 インスタンスストレージは暗号化されており、インスタンスに直接アタッチされたストレージになっています。詳細については、「[Amazon EC2 でのデータ保護](#)」を参照してください。
- EKS Auto Mode は、ルートボリュームやデータボリュームなど、作成時に EC2 インスタンスにアタッチされたボリュームを管理します。EKS Auto Mode は、Kubernetes 永続ストレージ機能を使用して作成された EBS ボリュームを完全には管理しません。

### パッチ管理

- Amazon EKS Auto Mode は、マネージドインスタンスにパッチを自動的に適用します。

- パッチには以下が含まれます。
  - オペレーティングシステムの更新
  - セキュリティパッチ
  - Amazon EKS Auto Mode コンポーネント

#### Note

これらのインスタンスで実行されているワークロードを保護および更新するのはお客様です。

## アクセス制御

- インスタンスへの直接アクセスは制限されています。
  - SSH アクセスは使用できません。
  - AWS Systems Manager Session Manager (SSM) アクセスは使用できません。
- 管理オペレーションは、Amazon EKS API と Kubernetes API を介して実行されます。

## 自動リソース管理

Amazon EKS Auto Mode は、Kubernetes 永続ストレージ機能を使用して作成された Amazon Elastic Block Store (Amazon EBS) ボリュームを完全には管理しません。EKS Auto Mode は Elastic Load Balancer (ELB) も管理しません。Amazon EKS Auto Mode は、これらのリソースのルーチンタスクを自動化します。

## ストレージセキュリティ

- AWS では、Kubernetes 永続ストレージ機能によってプロビジョニングされた EBS ボリュームの暗号化を有効にすることをお勧めします。詳細については、「[the section called “StorageClass を作成する”](#)」を参照してください。
- AWS KMS を使用した保管時の暗号化
- 作成した新しい EBS ボリュームとスナップショットコピーの暗号化を強制するように AWS アカウントを設定できます。詳細については、「Amazon EBS ユーザーガイド」の「[Enable Amazon EBS encryption by default](#)」を参照してください。
- 詳細については、「[Security in Amazon EBS](#)」を参照してください。

## ロードバランサーのセキュリティ

- Elastic Load Balancer の自動設定
- AWS Certificate Manager 統合による SSL/TLS 証明書管理
- ロードバランサーアクセスコントロールのセキュリティグループ自動化
- 詳細については、「[Security in Elastic Load Balancing](#)」を参照してください。

## セキュリティに関するベストプラクティス

次のセクションでは、Amazon EKS Auto Mode におけるセキュリティのベストプラクティスについて説明します。

- AWS IAM ポリシーと EKS アクセスエントリを定期的に確認します。
- ワークロードに最小特権のアクセスパターンを実装します。
- AWS CloudTrail と Amazon CloudWatch を使用してクラスターのアクティビティをモニタリングします。詳細については、[the section called “AWS CloudTrail”](#) および [the section called “Amazon CloudWatch”](#) を参照してください。
- セキュリティ体制の評価には AWS Security Hub を使用します。
- ワークロードに適したポッドセキュリティ標準を実装します。

## Amazon EKS の Identity and Access Management

AWS Identity and Access Management (IAM) は、管理者が AWS リソースへのアクセスを安全に制御するのに役立つ AWS のサービスです。IAM 管理者は、誰の (サインイン) を認証し、誰による Amazon EKS リソースの使用を承認する (アクセス許可を付与する) かを制御します。IAM は、AWS のサービスで追加料金は発生しません。

### 対象者

AWS Identity and Access Management (IAM) の使用方法は、Amazon EKS で行う作業によって異なります。

サービスユーザー – ジョブを実行するために Amazon EKS サービスを使用する場合は、管理者から必要なアクセス許可と認証情報が与えられます。さらに多くの Amazon EKS 機能を使用して作業を行う場合は、追加のアクセス許可が必要になることがあります。アクセスの管理方法を理解すると、



管理者から適切なアクセス許可をリクエストするのに役に立ちます。Amazon EKS の機能にアクセスできない場合は、「[the section called “トラブルシューティング”](#)」を参照してください。

サービス管理者 – 社内の Amazon EKS リソースを管理しているユーザーには、通常、Amazon EKS への完全なアクセス権限があります。サービスのユーザーがどの Amazon EKS 機能やリソースにアクセスするかを決めるのは管理者の仕事です。その後、IAM 管理者にリクエストを送信して、サービスユーザーの権限を変更する必要があります。このページの情報を確認して、IAM の基本概念を理解してください。企業が Amazon EKS で IAM を利用する方法については、「[the section called “IAM および Amazon EKS”](#)」を参照してください。

IAM 管理者 – IAM 管理者には、Amazon EKS へのアクセスを管理するポリシーの作成方法を、詳細に把握する必要が生じる場合があります。IAM で使用可能な、Amazon EKS アイデンティティベースのポリシーの例を確認するには、「[the section called “アイデンティティベースのポリシー”](#)」を参照してください。

## アイデンティティを使用した認証

認証とは、アイデンティティ認証情報を使用して AWS にサインインする方法です。ユーザーは、AWS アカウントのルートユーザーもしくは IAM ユーザーとして、または IAM ロールを引き受けることによって、認証を受ける (AWS にサインインする) 必要があります。

ID ソースから提供された認証情報を使用して、フェデレーテッドアイデンティティとして AWS にサインインできます。AWS フェデレーテッドアイデンティティの例としては、IAM アイデンティティセンター (IAM アイデンティティセンター) ユーザー、貴社のシングルサインオン認証、Google または Facebook の認証情報などがあります。フェデレーテッド ID としてサインインする場合、IAM ロールを使用して、前もって管理者により ID フェデレーションが設定されています。フェデレーションを使用して AWS にアクセスする場合、間接的にロールを引き受けることになります。

ユーザーのタイプに応じて、AWS Management Console または AWS アクセスポータルにサインインできます。AWS へのサインインの詳細については、AWS サインインユーザーガイドの「[AWS アカウントにサインインする方法](#)」を参照してください。

プログラムを使用して AWS にアクセスする場合、AWS は Software Development Kit (SDK) とコマンドラインインターフェイス (CLI) を提供し、認証情報を使用してリクエストに暗号で署名します。AWS ツールを使用しない場合は、リクエストに自分で署名する必要があります。リクエストに署名する推奨方法の使用については、「IAM ユーザーガイド」の「[AWS API リクエストの署名](#)」を参照してください。

使用する認証方法を問わず、追加セキュリティ情報の提供をリクエストされる場合もあります。例えば、AWS は、アカウントのセキュリティを強化するために多要素認証 (MFA) を使用することをお勧めします。詳細については、「AWS IAM Identity Center ユーザーガイド」の「[多要素認証](#)」および「IAM ユーザーガイド」の「[AWS での多要素認証 \(MFA\) の使用](#)」を参照してください。

## AWS アカウントのルートユーザー

AWS アカウントを作成する場合は、このアカウントのすべての AWS サービスとリソースに対して完全なアクセス権を持つ 1 つのサインインアイデンティティから始めます。このアイデンティティは AWS アカウントルートユーザーと呼ばれ、アカウントの作成に使用したメールアドレスとパスワードでのサインインによりアクセスされます。日常的なタスクには、ルートユーザーを使用しないことを強くお勧めします。ルートユーザーの認証情報は保護し、ルートユーザーでしか実行できないタスクを実行するときに使用します。ルートユーザーとしてサインインする必要があるタスクの完全なリストについては、IAM ユーザーガイドの「[ルートユーザー認証情報が必要なタスク](#)」を参照してください。

## IAM ユーザーとグループ

[IAM ユーザー](#)は、単一のユーザーまたはアプリケーションに対する特定の許可を持つ AWS アカウント内のアイデンティティです。可能であれば、パスワードやアクセスキーなどの長期的な認証情報を保有する IAM ユーザーを作成する代わりに、一時的な認証情報を使用することをお勧めします。ただし、IAM ユーザーでの長期的な認証情報が必要な特定のユースケースがある場合は、アクセスキーをローテーションすることをお勧めします。詳細については、「IAM ユーザーガイド」の「[長期的な認証情報を必要とするユースケースのためにアクセスキーを定期的にローテーションする](#)」を参照してください。

[IAM グループ](#)は、IAM ユーザーの集団を指定するアイデンティティです。グループとしてサインインすることはできません。グループを使用して、複数のユーザーに対して一度に権限を指定できます。多数のユーザーグループがある場合、グループを使用することで権限の管理が容易になります。例えば、IAMAdmins という名前のグループを設定して、そのグループに IAM リソースを管理する許可を与えることができます。

ユーザーは、ロールとは異なります。ユーザーは 1 人の人または 1 つのアプリケーションに一意に関連付けられますが、ロールはそれを必要とする任意の人が引き受けるようになっています。ユーザーには永続的な長期の認証情報がありますが、ロールでは一時的な認証情報が提供されます。詳細については、「IAM ユーザーガイド」の「[IAM ユーザー \(ロールではなく\) の作成が適している場合](#)」を参照してください。



## IAM ロール

[IAM ロール](#)は、特定のアクセス許可を持つ、AWS アカウント内のアイデンティティです。これは IAM ユーザーに似ていますが、特定のユーザーに関連付けられていません。[ロールを切り替える](#)ことによって、AWS Management Consoleで IAM ロールを一時的に引き受けることができます。ロールを引き受けるには、AWS CLI または AWS API オペレーションを呼び出すか、カスタム URL を使用します。ロールを使用する方法の詳細については、「IAM ユーザーガイド」の「[IAM ロールの使用](#)」を参照してください。

IAM ロールと一時的な認証情報は、次の状況で役立ちます:

- フェデレーションユーザーアクセス - フェデレーティッド ID に許可を割り当てるには、ロールを作成してそのロールの許可を定義します。フェデレーティッド ID が認証されると、その ID はロールに関連付けられ、ロールで定義されている許可が付与されます。フェデレーションの詳細については、「IAM ユーザーガイド」の「[サードパーティーアイデンティティプロバイダー向けロールの作成](#)」を参照してください。IAM Identity Center を使用する場合は、許可セットを設定します。アイデンティティが認証後にアクセスできるものを制御するため、IAM Identity Center は、権限セットを IAM のロールに関連付けます。許可セットの詳細については、「AWS IAM Identity Center ユーザーガイド」の「[許可セット](#)」を参照してください。
- 一時的な IAM ユーザー権限 - IAM ユーザーまたはロールは、特定のタスクに対して複数の異なる権限を一時的に IAM ロールで引き受けることができます。
- クロスアカウントアクセス - IAM ロールを使用して、自分のアカウントのリソースにアクセスすることを、別のアカウントの人物 (信頼済みプリンシパル) に許可できます。ロールは、クロスアカウントアクセスを許可する主な方法です。ただし、一部の AWS のサービスでは、(ロールをプロキシとして使用する代わりに) リソースにポリシーを直接アタッチできます。クロスアカウントアクセスにおけるロールとリソースベースのポリシーの違いについては、「IAM ユーザーガイド」の「[IAM でのクロスアカウントのリソースへのアクセス](#)」を参照してください。
- クロスサービスアクセス - 一部の AWS のサービスは、AWS の他のサービスの機能を使用します。例えば、サービスで呼び出しを行うと、通常そのサービスによって Amazon EC2 でアプリケーションが実行されたり、Amazon S3 にオブジェクトが保存されたりします。サービスは、呼び出し元のプリンシパルの許可、サービスロール、サービスリンクロールを使用してこれを行う場合があります。
- 転送アクセスセッション (FAS) - IAM ユーザーまたはロールを使用して AWS でアクションを実行するユーザーは、プリンシパルと見なされます。一部のサービスを使用する際に、アクションを実行することで、別のサービスの別のアクションがトリガーされることがあります。FAS は、AWS サービスを呼び出すプリンシパルの権限を、AWS サービスのリクエストと合わせて使用し、ダウンストリームのサービスに対してリクエストを行います。FAS リクエストは、

サービスが、完了するために他の AWS サービスまたはリソースとのやりとりを必要とするリクエストを受け取ったときにのみ行われます。この場合、両方のアクションを実行するためのアクセス許可が必要です。FAS リクエストを行う際のポリシーの詳細については、「[転送アクセスセッション](#)」を参照してください。

- サービスロール - サービスがユーザーに代わってアクションを実行するために引き受ける [IAM ロール](#)です。IAM 管理者は、IAM 内からサービスロールを作成、変更、削除することができます。詳細については、『IAM ユーザーガイド』の「[AWS のサービスにアクセス許可を委任するロールの作成](#)」を参照してください。
- サービスリンクロール - サービスリンクロールは、AWS のサービスにリンクされているサービスロールの一種です。サービスがロールを引き受け、ユーザーに代わってアクションを実行できるようになります。サービスにリンクされたロールは、AWS アカウント内に表示され、サービスによって所有されます。IAM 管理者は、サービスリンクロールのアクセス許可を表示できますが、編集することはできません。
- Amazon EC2 で実行されているアプリケーション - EC2 インスタンスで実行され、AWS CLI または AWS API リクエストを作成しているアプリケーションの一時的な認証情報を管理するには、IAM ロールを使用します。これは、EC2 インスタンス内でのアクセスキーの保存に推奨されます。AWS ロールを EC2 インスタンスに割り当て、そのすべてのアプリケーションで使用できるようにするには、インスタンスに添付されたインスタンスプロファイルを作成します。インスタンスプロファイルにはロールが含まれ、EC2 インスタンスで実行されるプログラムは一時的な認証情報を取得できます。詳細については、IAM ユーザーガイドの[Amazon EC2 インスタンスで実行されるアプリケーションに IAM ロールを使用して許可を付与する](#)を参照してください。

IAM ロールと IAM ユーザーのどちらを使用するかについては、IAM ユーザーガイドの[\(IAM ユーザーではなく\) IAM ロールをいつ作成したら良いのか?](#)を参照してください。

## ポリシーを使用したアクセス権の管理

AWS でアクセスを制御するには、ポリシーを作成して AWS ID またはリソースにアタッチします。ポリシーは AWS のオブジェクトであり、アイデンティティやリソースに関連付けて、これらのアクセス許可を定義します。AWS は、プリンシパル (ユーザー、ルートユーザー、またはロールセッション) がリクエストを行うと、これらのポリシーを評価します。ポリシーでの権限により、リクエストが許可されるか拒否されるかが決まります。大半のポリシーは JSON ドキュメントとして AWS に保存されます。JSON ポリシードキュメントの構造と内容の詳細については、IAM ユーザーガイドの [JSON ポリシー概要](#)を参照してください。

管理者は AWS JSON ポリシーを使用して、だれが何にアクセスできるかを指定できます。つまり、どのプリンシパルがどのリソースに対してどのような条件下でアクションを実行できるかということです。

デフォルトでは、ユーザーやロールに権限はありません。IAM 管理者は、リソースに必要なアクションを実行するための権限をユーザーに付与する IAM ポリシーを作成できます。その後、管理者はロールに IAM ポリシーを追加し、ユーザーはロールを引き受けることができます。

IAM ポリシーは、オペレーションの実行方法を問わず、アクションの許可を定義します。例えば、iam:GetRole アクションを許可するポリシーがあるとします。このポリシーがあるユーザーは、AWS Management Console、AWS CLI、または AWS API からロール情報を取得できます。

## アイデンティティベースのポリシー

アイデンティティベースポリシーは、IAM ユーザーグループ、ユーザーのグループ、ロールなど、アイデンティティにアタッチできる JSON 許可ポリシードキュメントです。これらのポリシーは、ユーザーとロールが実行できるアクション、リソース、および条件をコントロールします。アイデンティティベースのポリシーを作成する方法については、「IAM ユーザーガイド」の「[IAM ポリシーの作成](#)」を参照してください。

アイデンティティベースのポリシーは、さらにインラインポリシーまたはマネージドポリシーに分類できます。インラインポリシーは、単一のユーザー、グループ、またはロールに直接埋め込まれています。管理ポリシーは、AWS アカウント内の複数のユーザー、グループ、およびロールにアタッチできるスタンドアロンポリシーです。管理ポリシーには、AWS 管理ポリシーとカスタマー管理ポリシーが含まれます。マネージドポリシーまたはインラインポリシーのいずれかを選択する方法については、IAM ユーザーガイドの[マネージドポリシーとインラインポリシーの比較](#)を参照してください。

## リソースベースのポリシー

リソースベースのポリシーは、リソースに添付する JSON ポリシードキュメントです。リソースベースのポリシーには例として、IAM ロールの信頼ポリシーや Amazon S3 バケットポリシーがあげられます。リソースベースのポリシーをサポートするサービスでは、サービス管理者はポリシーを使用して特定のリソースへのアクセスを制御できます。ポリシーがアタッチされているリソースの場合、指定されたプリンシパルがそのリソースに対して実行できるアクションと条件は、ポリシーによって定義されます。リソースベースのポリシーで、[\[specify a principal\]](#) (プリンシパルを指定する) 必要があります。プリンシパルには、アカウント、ユーザー、ロール、フェデレーテッドユーザー、または AWS のサービスを含めることができます。

リソースベースのポリシーは、そのサービス内にあるインラインポリシーです。リソースベースのポリシーでは、IAM からの AWS マネージド型ポリシーを使用することはできません。

## アクセスコントロールリスト (ACL)

アクセスコントロールリスト (ACL) は、どのプリンシパル (アカウントメンバー、ユーザー、またはロール) がリソースにアクセスするための許可を持つかを制御します。ACL はリソースベースのポリシーに似ていますが、JSON ポリシードキュメント形式は使用しません。

Amazon S3、AWS WAF、および Amazon VPC は、ACL をサポートするサービスの例です。ACL の詳細については、「Amazon Simple Storage Service デベロッパーガイド」の「[アクセスコントロールリスト \(ACL\) の概要](#)」を参照してください。

## その他のポリシータイプ

AWS では、他の一般的ではないポリシータイプをサポートしています。これらのポリシータイプでは、より一般的なポリシータイプで付与された最大の権限を設定できます。

- **アクセス許可の境界** - アクセス許可の境界は、アイデンティティベースポリシーによって IAM エンティティ (IAM ユーザーまたはロール) に付与できる権限の上限を設定する高度な機能です。エンティティにアクセス許可の境界を設定できます。結果として得られるアクセス許可は、エンティティのアイデンティティベースポリシーとそのアクセス許可の境界の共通部分になります。Principal フィールドでユーザーまたはロールを指定するリソースベースのポリシーでは、アクセス許可の境界は制限されません。これらのポリシーのいずれかを明示的に拒否した場合、権限は無効になります。アクセス許可の境界の詳細については、「IAM ユーザーガイド」の「[IAM エンティティのアクセス許可の境界](#)」を参照してください。
- **サービスコントロールポリシー (SCP)** - SCP とは、AWS Organizations 内の組織または組織単位 (OU) に対し、アクセス許可の上限を指定するための JSON ポリシーです。AWS Organizations は、ビジネスが所有する複数の AWS アカウントを、グループ化および一元管理するためのサービスです。組織内のすべての機能を有効にすると、サービスコントロールポリシー (SCP) を一部またはすべてのアカウントに適用できます。SCP はメンバーアカウントのエンティティに対するアクセス許可を制限します (各 AWS アカウントルートユーザーなど)。Organizations と SCP の詳細については、「AWS Organizations ユーザーガイド」の「[サービスコントロールポリシー](#)」を参照してください。
- **セッションポリシー** - セッションポリシーは、ロールまたはフェデレーションユーザーの一時的なセッションをプログラムで作成する際にパラメータとして渡す高度なポリシーです。結果として得られるセッションの許可は、ユーザーまたはロールのアイデンティティベースポリシーとセッションポリシーの共通部分です。また、リソースベースのポリシーから権限が派生する場合もありま

す。これらのポリシーのいずれかを明示的に拒否した場合、権限は無効になります。詳細については、「IAM ユーザーガイド」の「[セッションポリシー](#)」を参照してください。

## 複数のポリシータイプ

1 つのリクエストに複数のタイプのポリシーが適用されると、結果として作成される権限を理解するのがさらに難しくなります。複数のポリシータイプが関連するとき、リクエストを許可するかどうかを AWS が決定する方法の詳細については、IAM ユーザーガイドの[ポリシーの評価ロジック](#)を参照してください。

## Amazon EKS で IAM を使用する方法

IAM を使用して Amazon EKS へのアクセスを管理する前に、Amazon EKS で使用できる IAM 機能について理解しておく必要があります。Amazon EKS およびその他の AWS のサービスが IAM と連携する方法の概要を把握するには IAM ユーザーガイドの「[IAM と連携する AWS のサービス](#)」を参照してください。

### トピック

- [Amazon EKS のアイデンティティベースのポリシー](#)
- [Amazon EKS でのリソースベースのポリシー](#)
- [Amazon EKS タグに基づいた認可](#)
- [Amazon EKS での IAM 役割](#)

## Amazon EKS のアイデンティティベースのポリシー

IAM アイデンティティベースのポリシーでは許可または拒否するアクションとリソース、またアクションを許可または拒否する条件を指定できます。Amazon EKS は特定のアクション、リソース、および条件キーをサポートしています。JSON ポリシーで使用するすべての要素については「IAM ユーザーガイド」の「[IAM JSON ポリシーエレメントのリファレンス](#)」を参照してください。

### アクション

管理者は AWS JSON ポリシーを使用して、誰が何にアクセスできるかを指定できます。つまり、どのプリンシパルがどのリソースに対してどのような条件下でアクションを実行できるかということです。

JSON ポリシーの Action 要素にはポリシー内のアクセスを許可または拒否するために使用できるアクションが記述されます。ポリシーアクションの名前は通常、関連する AWS API オペレーション

と同じです。一致する API オペレーションのないアクセス許可のみのアクションなど、いくつかの例外があります。また、ポリシーに複数のアクションが必要なオペレーションもあります。これらの追加アクションは依存アクションと呼ばれます。

このアクションは関連付けられたオペレーションを実行するためのアクセス許可を付与するポリシーで使用されます。

Amazon EKS のポリシーアクションはアクションの前にプレフィックス `eks:` を付加します。例えば、Amazon EKS クラスターについて説明する情報を入手するアクセス許可を他のユーザーに付与するにはポリシーに `DescribeCluster` アクションを含めます。ポリシーステートメントには `Action` または `NotAction` 要素を含める必要があります。

単一のステートメントに複数のアクションを指定するには次のようにコンマで区切ります。

```
"Action": ["eks:action1", "eks:action2"]
```

ワイルドカード (\*) を使用して複数アクションを指定できます。例えば、`Describe` という単語で始まるすべてのアクションを指定するには次のアクションを含めます。

```
"Action": "eks:Describe*"
```

Amazon EKS アクションの一覧については「サービス認可リファレンス」の「[Amazon Elastic Kubernetes Serviceで定義されるアクション](#)」を参照してください。

## リソース

管理者は AWS JSON ポリシーを使用して、誰が何にアクセスできるかを指定できます。つまり、どのプリンシパルが、どのリソースに対してどのような条件下でアクションを実行できるかということです。

Resource JSON ポリシー要素はアクションが適用されるオブジェクトを指定します。ステートメントには `Resource` または `NotResource` 要素を含める必要があります。ベストプラクティスとして、[Amazon リソースネーム \(ARN\)](#) を使用してリソースを指定します。これはリソースレベルの許可と呼ばれる特定のリソースタイプをサポートするアクションに対して実行できます。

オペレーションのリスト化など、リソースレベルの許可をサポートしないアクションの場合はステートメントがすべてのリソースに適用されることを表示するワイルドカード (\*) を使用します。

```
"Resource": "*"
```



Amazon EKS クラスターリソースの ARN は次のようになります。

```
arn:aws:eks:region-code:account-id:cluster/cluster-name
```

ARN の形式の詳細については「[Amazon リソースネーム \(ARN\) と AWS のサービスの名前空間](#)」を参照してください。

例えば、ステートメントで ##### という名前のクラスターを指定するには次の ARN を使用します。

```
"Resource": "arn:aws:eks:region-code:111122223333:cluster/my-cluster"
```

特定のアカウントと AWS リージョンに属するすべてのクラスターを指定するにはワイルドカード (\*) を使用します。

```
"Resource": "arn:aws:eks:region-code:111122223333:cluster/*"
```

リソースの作成を含む、一部の Amazon EKS アクションは特定のリソースで実行できません。このような場合はワイルドカード \* を使用する必要があります。

```
"Resource": ""
```

Amazon EKS リソースタイプとその ARN の一覧については「サービス認可リファレンス」の「[Amazon Elastic Kubernetes Service で定義されるリソース](#)」を参照してください。各リソースの ARN を、どのアクションで指定できるかについては「[Amazon Elastic Kubernetes Service で定義されるアクション](#)」を参照してください。

## 条件キー

Amazon EKS では独自の条件キーが定義されており、また一部のグローバル条件キーの使用がサポートされています。すべての AWS グローバル条件キーを確認するには、IAM ユーザーガイドの「[AWS グローバル条件コンテキストキー](#)」を参照してください。

OpenID Connect プロバイダーをクラスターに関連付ける場合に条件キーを設定できます。詳細については、「[the section called "IAM ポリシーの例"](#)」を参照してください。

すべての Amazon EC2 アクションは aws:RequestedRegion および ec2:Region 条件キーをサポートします。詳細については「[例: 特定の AWS リージョンへのアクセスの制限](#)」を参照してください。

アマゾン EKS 条件キーのリストについては「サービス認可リファレンス」の「[アマゾン エラスティックKubernetesサービスによって定義された条件](#)」を参照してください。条件キーを使用できるアクションとリソースについては「[アマゾン エラスティックKubernetesサービスで定義されるアクション](#)」を参照してください。

## 例

Amazon EKS のアイデンティティベースのポリシーの例は「[the section called “アイデンティティベースのポリシー”](#)」でご確認ください。

アマゾン EKS クラスターを作成すると、このクラスターを作成する [IAM プリンシパル](#)にはアマゾン EKS コントロールプレーンのクラスター役割ベースアクセスコントロール (RBAC) 設定で、system:masters 許可が自動的に付与されます。このプリンシパルは表示可能な設定の中には表示されません。したがって、どのプリンシパルが最初にクラスターを作成したのかを追跡する必要があります。追加の IAM プリンシパルがクラスターとインタラクションできるようにするには Kubernetes 内の aws-auth ConfigMap を編集し、aws-auth ConfigMap で指定した group の名前を使用して、Kubernetes rolebinding または clusterrolebinding を作成します。

ConfigMap の使用に関する詳細については「[the section called “Kubernetes API アクセス”](#)」を参照してください。

## Amazon EKS でのリソースベースのポリシー

Amazon EKS では リソースベースのポリシーはサポートされていません。

## Amazon EKS タグに基づいた認可

タグは Amazon EKS リソースにアタッチすることも、Amazon EKS へのリクエストの際に渡すこともできます。タグに基づいてアクセスを制御するにはaws:ResourceTag/*key-name*、aws:RequestTag/*key-name*、または aws:TagKeys の条件キーを使用して、ポリシーの[条件要素](#)でタグ情報を提供します。Amazon EKS リソースのタグ付けの詳細については「[the section called “リソースのタグ付け”](#)」を参照してください。条件キーでタグを使用できるアクションの詳細については「[Service Authorization Reference](#)」(サービス認可のリファレンス)の「[Amazon EKS で定義されたアクション](#)」を参照してください。

## Amazon EKS でのIAM 役割

[IAM 役割](#) は特定のアクセス許可を持つ、AWS アカウント内のエンティティです。



## Amazon EKS での一時的な認証情報の使用

一時的な認証情報を使用して、フェデレーションでサインインする、IAM 役割を引き受ける、またはクロスアカウント役割を引き受けることができます。一時的なセキュリティ認証情報を取得するには [AssumeRole](#) または [GetFederationToken](#) などの AWS STS API オペレーションを呼び出します。

Amazon EKS では一時認証情報が使用できます。

### サービスにリンクされた役割

```
link:IAM/latest/UserGuide/id_roles.html#iam-term-service-linked-role[Service-linked roles,type="documentation"] allow {aws} services to access resources in other services to complete an action on your behalf. Service-linked roles appear in your IAM account and are owned by the service. An administrator can view but can't edit the permissions for service-linked roles.
```

Amazon EKS はサービスにリンクされた役割をサポートしています。Amazon EKS でのサービスにリンクされた役割の作成または管理の詳細については「[the section called “サービスにリンクされた役割”](#)」を参照してください。

### サービス役割

この機能により、ユーザーに代わってサービスが[サービス役割](#)を引き受けることが許可されます。この役割により、サービスがお客様に代わって他のサービスのリソースにアクセスし、アクションを完了することが許可されます。サービス役割はIAM アカウントに表示され、アカウントによって所有されます。つまり、IAM 管理者はこの役割の権限を変更できます。ただし、これを行うことにより、サービスの機能が損なわれる場合があります。

Amazon EKS ではサービス役割がサポートされています。詳細については[the section called “クラスター IAM ロール”](#)および[the section called “ノード の IAM ロール”](#)を参照してください。

### Amazon EKS での IAM 役割の選択

Amazon EKS でクラスターリソースを作成する場合は他のいくつかの AWS リソースに Amazon EKS が自動的にアクセスすることを許可するための役割を、ユーザーが選択する必要があります。以前に作成したサービス役割がある場合、Amazon EKS により、選択できる役割のリストが提示されます。Amazon EKS 管理のポリシーがアタッチされた役割を選択することが重要です。詳細については[the section called “既存のクラスターロールの確認”](#)および[the section called “既存のノードロールの確認”](#)を参照してください。

## Amazon EKS でのアイデンティティベースのポリシーの例

デフォルトでは、IAM ユーザーおよびロールには Amazon EKS リソースを作成または変更するアクセス許可はありません。また、AWS Management Console、AWS CLI、または AWS API を使用してタスクを実行することもできません。IAM 管理者は、ユーザーとロールに必要な、指定されたリソースで特定の API オペレーションを実行する権限をユーザーとロールに付与する IAM ポリシーを作成する必要があります。続いて、管理者はそれらの権限が必要な IAM ユーザーまたはグループにそのポリシーをアタッチする必要があります。

これらの JSON ポリシードキュメント例を使用して IAM のアイデンティティベースのポリシーを作成する方法については、『IAM ユーザーガイド』の「[JSON タブでのポリシーの作成](#)」を参照してください。

Amazon EKS クラスターを作成すると、このクラスターを作成する [IAM プリンシパル](#)には、Amazon EKS コントロールプレーンのクラスターロールベースアクセスコントロール (RBAC) 設定で、system:masters 許可が自動的に付与されます。このプリンシパルは表示可能な設定の中には表示されません。したがって、どのプリンシパルが最初にクラスターを作成したのかを追跡する必要があります。追加の IAM プリンシパルがクラスターとインタラクションできるようにするには、Kubernetes 内の aws-auth ConfigMap を編集し、aws-auth ConfigMap で指定した group の名前を使用して、Kubernetes rolebinding または clusterrolebinding を作成します。

ConfigMap の使用に関する詳細については「[the section called “Kubernetes API アクセス”](#)」を参照してください。

### トピック

- [ポリシーに関するベストプラクティス](#)
- [Amazon EKS コンソールの使用](#)
- [自分のアクセス許可の表示を IAM ユーザーに許可する](#)
- [AWS Cloud に Kubernetes クラスターを作成する](#)
- [Outpost にローカル Kubernetes クラスターを作成する](#)
- [Kubernetes クラスターの更新](#)
- [すべてのクラスターの一覧表示または説明](#)

### ポリシーに関するベストプラクティス

ID ベースのポリシーは、ユーザーのアカウント内で誰かが Amazon EKS リソースを作成、アクセス、または削除できるかどうかを決定します。これらのアクションを実行すると、AWS アカウントに

追加料金が発生する可能性があります。アイデンティティベースポリシーを作成したり編集したりする際には、以下のガイドラインと推奨事項に従ってください:

- AWS マネージドポリシーを使用して開始し、最小特権の許可に移行する – ユーザーとワークロードへの許可の付与を開始するには、多くの一般的なユースケースのために許可を付与する AWS マネージドポリシーを使用します。これらは AWS アカウントで使用できます。ユースケース別に AWS カスタマーマネージドポリシーを定義して、マネージドポリシーを絞り込むことをお勧めします。詳細については、「IAM ユーザーガイド」の「[AWS マネージドポリシー](#)」または「[ジョブ機能の AWS マネージドポリシー](#)」を参照してください。
- 最小特権を適用する – IAM ポリシーで許可を設定する場合は、タスクの実行に必要な許可のみを付与します。これを行うには、特定の条件下で特定のリソースに対して実行できるアクションを定義します。これは、最小特権アクセス許可とも呼ばれています。IAM を使用して許可を適用する方法の詳細については、「IAM ユーザーガイド」の「[IAM でのポリシーとアクセス許可](#)」を参照してください。
- IAM ポリシーで条件を使用してアクセスをさらに制限する – ポリシーに条件を追加して、アクションやリソースへのアクセスを制限できます。例えば、ポリシー条件を記述して、すべてのリクエストを SSL を使用して送信するように指定できます。また、AWS CloudFormation などの特定の AWS サービスを介して使用する場合、条件を使用してサービスアクションへのアクセスを許可することもできます。詳細については、「IAM ユーザーガイド」の「[IAM JSON ポリシー要素: 条件](#)」を参照してください。
- IAM Access Analyzer を使用して IAM ポリシーを検証し、安全で機能的な権限を確保する – IAM Access Analyzer は、新規および既存のポリシーを検証して、ポリシーが IAM ポリシー言語 (JSON) および IAM のベストプラクティスに準拠するようにします。IAM アクセスアナライザーは 100 を超えるポリシーチェックと実用的な推奨事項を提供し、安全で機能的なポリシーの作成をサポートします。詳細については、IAM ユーザーガイドの[IAM Access Analyzer ポリシーの検証](#)を参照してください。
- 多要素認証 (MFA) を要求する – AWS アカウントで IAM ユーザーまたはルートユーザーを要求するシナリオがある場合は、セキュリティを強化するために MFA をオンにします。API オペレーションが呼び出されるときに MFA を必須にするには、ポリシーに MFA 条件を追加します。詳細については、IAM ユーザーガイドの[MFA 保護 API アクセスの設定](#)を参照してください。

IAM でのベストプラクティスの詳細については、IAM ユーザーガイドの[IAM でのセキュリティのベストプラクティス](#)を参照してください。

## Amazon EKS コンソールの使用

Amazon EKS コンソールにアクセスするには、[IAM プリンシパル](#)に最小限の許可のセットが必要です。これらの許可により、プリンシパルは AWS アカウントの Amazon EKS リソースの詳細をリストおよび表示できます。最小限必要な許可よりも制限されたアイデンティティベースのポリシーを作成すると、そのポリシーをアタッチしたプリンシパルに対してはコンソールが意図したとおりに機能しません。

これらの IAM プリンシパルがまだ Amazon EKS コンソールを使用できるようにするには、AmazonEKSAAdminPolicy など、独自の名前を付けてポリシーを作成します。ポリシーをプリンシパルにアタッチします。詳細については、「IAM ユーザーガイド」の「[IAM ID アクセス許可の追加および削除](#)」を参照してください。

### Important

次のポリシー例では、プリンシパルはコンソールの [設定] タブで情報を表示できます。AWS Management Console で [概要] タブと [リソース] タブの情報を表示するには、プリンシパルに Kubernetes の許可も必要です。詳細については、「[the section called “必要なアクセス許可”](#)」を参照してください。

```
{
 "Version": "2012-10-17",
 "Statement": [
 {
 "Effect": "Allow",
 "Action": [
 "eks:*"
],
 "Resource": "*"
 },
 {
 "Effect": "Allow",
 "Action": "iam:PassRole",
 "Resource": "*",
 "Condition": {
 "StringEquals": {
 "iam:PassedToService": "eks.amazonaws.com"
 }
 }
 }
]
}
```

```
 }
]
}
```

AWS CLI または AWS API のみ呼び出すプリンシパルには、最小限のコンソール許可を付与する必要はありません。代わりに、実行する API オペレーションと一致するアクションのみへのアクセスを許可します。

## 自分のアクセス許可の表示を IAM ユーザーに許可する

この例では、ユーザーアイデンティティにアタッチされたインラインおよびマネージドポリシーの表示を IAM ユーザーに許可するポリシーの作成方法を示します。このポリシーには、コンソールで、または AWSCLI もしくは AWSAPI を使用してプログラマ的に、このアクションを完了するアクセス許可が含まれています。

```
{
 "Version": "2012-10-17",
 "Statement": [
 {
 "Sid": "ViewOwnUserInfo",
 "Effect": "Allow",
 "Action": [
 "iam:GetUserPolicy",
 "iam:ListGroupsForUser",
 "iam:ListAttachedUserPolicies",
 "iam:ListUserPolicies",
 "iam:GetUser"
],
 "Resource": ["arn:aws:iam::*:user/${aws:username}"]
 },
 {
 "Sid": "NavigateInConsole",
 "Effect": "Allow",
 "Action": [
 "iam:GetGroupPolicy",
 "iam:GetPolicyVersion",
 "iam:GetPolicy",
 "iam:ListAttachedGroupPolicies",
 "iam:ListGroupPolicies",
 "iam:ListPolicyVersions",
 "iam:ListPolicies",
 "iam:ListUsers"
]
 }
]
}
```

```

],
 "Resource": "*"
 }
]
}

```

## AWS Cloud に Kubernetes クラスターを作成する

このポリシー例には、**us-west-2** AWS リージョンに **my-cluster** という名前の Amazon EKS クラスターを作成するために必要な最小限の許可が含まれています。AWS リージョンを、クラスターを作成する AWS リージョンに置き換えることができます。AWS Management Console で [ポリシーのアクションはリソースレベルの許可をサポートしていないため、**All resources** を選択する必要があります] という警告が表示された場合、無視しても問題ありません。アカウントに **AWSServiceRoleForAmazonEKS** ロールが既にある場合は、ポリシーから **iam:CreateServiceLinkedRole** アクションを削除できます。アカウントで Amazon EKS クラスターを作成したことがある場合、削除していない限り、このロールは既に存在します。

```

{
 "Version": "2012-10-17",
 "Statement": [
 {
 "Effect": "Allow",
 "Action": "eks:CreateCluster",
 "Resource": "arn:aws:eks:us-west-2:111122223333:cluster/my-cluster"
 },
 {
 "Effect": "Allow",
 "Action": "iam:CreateServiceLinkedRole",
 "Resource": "arn:aws:iam::111122223333:role/aws-service-role/eks.amazonaws.com/AWSServiceRoleForAmazonEKS",
 "Condition": {
 "ForAnyValue:StringEquals": {
 "iam:AWSServiceName": "eks"
 }
 }
 },
 {
 "Effect": "Allow",
 "Action": "iam:PassRole",
 "Resource": "arn:aws:iam::111122223333:role/cluster-role-name"
 }
]
}

```

```
}
```

## Outpost にローカル Kubernetes クラスターを作成する

このポリシー例には、**us-west-2** AWS リージョンの Outpost 上に **my-cluster** という名前の Amazon EKS ローカルクラスターを作成するために必要な最小限の許可が含まれています。AWS リージョンを、クラスターを作成する AWS リージョンに置き換えることができます。AWS Management Console で [ポリシーのアクションはリソースレベルの許可をサポートしていないため、**All resources** を選択する必要があります] という警告が表示された場合、無視しても問題ありません。アカウントに `AWSServiceRoleForAmazonEKSLocalOutpost` ロールがすでにある場合、ポリシーから `iam:CreateServiceLinkedRole` アクションを削除できます。アカウントで Outpost 上に Amazon EKS ローカルクラスターを作成したことがある場合、削除していない限り、このロールはすでに存在します。

```
{
 "Version": "2012-10-17",
 "Statement": [
 {
 "Effect": "Allow",
 "Action": "eks:CreateCluster",
 "Resource": "arn:aws:eks:us-west-2:111122223333:cluster/my-cluster"
 },
 {
 "Action": [
 "ec2:DescribeSubnets",
 "ec2:DescribeVpcs",
 "iam:GetRole"
],
 "Resource": "*",
 "Effect": "Allow"
 },
 {
 "Effect": "Allow",
 "Action": "iam:CreateServiceLinkedRole",
 "Resource": "arn:aws:iam::111122223333:role/aws-service-role/outposts.eks-local.amazonaws.com/AWSServiceRoleForAmazonEKSLocalOutpost"
 },
 {
 "Effect": "Allow",
 "Action": [
 "iam:PassRole",
 "iam:ListAttachedRolePolicies"
]
 }
]
}
```

```

]
 "Resource": "arn:aws:iam::111122223333:role/cluster-role-name"
 },
 {
 "Action": [
 "iam:CreateInstanceProfile",
 "iam:TagInstanceProfile",
 "iam:AddRoleToInstanceProfile",
 "iam:GetInstanceProfile",
 "iam>DeleteInstanceProfile",
 "iam:RemoveRoleFromInstanceProfile"
],
 "Resource": "arn:aws:iam::*:instance-profile/eks-local-*",
 "Effect": "Allow"
 },
]
}

```

## Kubernetes クラスターの更新

このポリシー例には、us-west-2 AWS リージョンに *my-cluster* という名前のクラスターを更新するために必要な最小限の許可が含まれています。

```

{
 "Version": "2012-10-17",
 "Statement": [
 {
 "Effect": "Allow",
 "Action": "eks:UpdateClusterVersion",
 "Resource": "arn:aws:eks:us-west-2:111122223333:cluster/my-cluster"
 }
]
}

```

## すべてのクラスターの一覧表示または説明

このポリシーの例には、アカウント内のすべてのクラスターを一覧表示して記述するために必要な最小限の許可が含まれています。update-kubeconfig AWS CLI コマンドを使用するには、[IAM プリンシパル](#)がクラスターを一覧表示および記述する必要があります。

```

{
 "Version": "2012-10-17",

```



```
"Statement": [
 {
 "Effect": "Allow",
 "Action": [
 "eks:DescribeCluster",
 "eks:ListClusters"
],
 "Resource": "*"
 }
]
```

## Amazon EKS でのサービスにリンクされたロールの使用

Amazon エラスティック Kus サービスはAWS アイデンティティとアクセス管理 (IAM) の[サービスにリンクされた役割](#)を使用します。サービスにリンクされた役割はAmazon EKS に直接リンクされた一意のタイプの IAM 役割です。サービスにリンクされた役割はAmazon EKS で事前定義されています。この役割にはサービスがユーザーに代わって他の AWS のサービスを呼び出すために必要な、すべてのアクセス許可が付与されています。

### トピック

- [Amazon EKS クラスターのロールの使用](#)
- [アマゾン EKS ノードグループでのロールの使用](#)
- [Amazon EKS Fargate プロファイルの役割の使用](#)
- [ロールを使用して Kubernetes クラスターを Amazon EKS に接続する](#)
- [アウトポスト で アマゾン EKS ローカルクラスターの役割の使用](#)
- [Amazon EKS ダッシュボードのロールの使用](#)

## Amazon EKS クラスターのロールの使用

アマゾン エラスティック Kus サービスはAWS アイデンティティとアクセス管理 (IAM) の[サービスにリンクされた役割](#)を使用します。サービスにリンクされた役割はアマゾン EKS に直接リンクされた一意のタイプの IAM 役割です。サービスにリンクされた役割はアマゾン EKS で事前定義されています。この役割にはサービスがユーザーに代わって他の AWS のサービスを呼び出すために必要な、すべてのアクセス許可が付与されています。

サービスにリンクされた役割を使用することで、必要なアクセス許可を手動で追加する必要がなくなるため、アマゾン EKS の設定が簡単になります。サービスにリンクされた役割のアクセス許可はア

マゾン EKS により定義されます。特に指定されている場合を除き、アマゾン EKS のみがその役割を引き受けることができます。定義される許可は信頼ポリシーと許可ポリシーに含まれており、その許可ポリシーを他の IAM エンティティにアタッチすることはできません。

サービスリンク役割はまずその関連リソースを削除しなければ削除できません。これにより、リソースへのアクセス許可を不用意に削除することが防止され、アマゾン EKS リソースが保護されます。

サービスにリンクされた役割をサポートする他のサービスについては「[IAM と連動する AWS サービス](#)」を参照し、[サービス-linked roles] (サービスにリンクされた役割) の列内で [Yes] (はい) と表記されたサービスを確認してください。サービスにリンクされた役割に関するドキュメントをサービスで表示するには[はい] リンクを選択してください。

### アマゾン EKS でのサービスにリンクされた役割のアクセス許可

Amazon EKS は、AWSServiceRoleForAmazonEKS という名前のサービスにリンクされたロールを使用します。このロールにより、Amazon EKS はアカウント内のクラスターを管理できます。アタッチされたポリシーにより、ロールはネットワークインターフェイス、セキュリティグループ、ログ、VPC のリソースを管理できるようになります。

#### Note

AWSServiceRoleForAmazonEKS サービスにリンクされたロールは、クラスターの作成に必要なロールとは異なります。詳細については「[the section called “クラスター IAM ロール”](#)」を参照してください。

AWSServiceRoleForAmazonEKS サービスリンク役割は役割の引き受けについて以下のサービスを信頼します。

- [eks.amazonaws.com](https://eks.amazonaws.com)

役割のアクセス許可ポリシーは指定したリソースに対して以下のアクションを実行することをアマゾン EKS に許可します。

- [アマゾンEKSサービスロールポリシー](#)

サービスにリンクされた役割の作成、編集、削除を IAM エンティティ (ユーザー、グループ、役割など) に許可するにはアクセス許可を設定する必要があります。詳細についてはIAM ユーザーガイドの「[サービスにリンクされた役割のアクセス許可](#)」を参照してください。

## アマゾン EKS でのサービスにリンクされた役割の作成

サービスリンク役割を手動で作成する必要はありません。AWS Management Console、AWS CLI、または AWS API でクラスターを作成すると、アマゾン EKS によってサービスリンク役割が作成されます。

このサービスリンク役割を削除した後で再度作成する必要がある場合は同じ方法でアカウントに役割を再作成できます。サービスにリンクされた役割はクラスターの作成時に アマゾン EKS で自動的に再作成されます。

## アマゾン EKS でのサービスにリンクされた役割の編集

アマゾン EKS では `AWSServiceRoleForAmazonEKS` のサービスにリンクされた役割を編集することはできません。サービスにリンクされた役割を作成すると、多くのエンティティによって役割が参照される可能性があるため、役割名を変更することはできません。ただし、IAM を使用してロールの説明を編集することはできます。詳細については『IAM ユーザーガイド』の「[サービスにリンクされた役割の編集](#)」を参照してください。

## アマゾン EKS でのサービスにリンクされた役割の削除

サービスリンク役割が必要な機能またはサービスが不要になった場合にはその役割を削除することをお勧めします。そうすることで、モニタリングや保守が積極的に行われていない未使用のエンティティを排除できます。ただし、手動で削除する前に、サービスリンク役割をクリーンアップする必要があります。

### サービスリンク役割のクリーンアップ

IAM を使用してサービスにリンクされた役割を削除するには最初に、その役割で使用されているリソースをすべて削除する必要があります。

#### Note

リソースの削除を試みた際に、対応する役割が アマゾン EKS サービスで使用されている場合、削除が失敗することがあります。失敗した場合は数分待ってから操作を再試行してください。

1. [アマゾン EKS コンソール](#)を開きます。
2. 左のナビゲーションペインで [Clusters] (クラスター) を選択してください。

3. クラスターにノードグループまたは Fargate プロファイルがある場合は、クラスターを削除する前にそれらを削除する必要があります。詳細については「[the section called “削除”](#)」および「[the section called “プロファイルを削除する”](#)」を参照してください。
4. [Clusters (クラスター)] ページで、削除するクラスターを選択し、[Delete (削除)] を選択してください。
5. 削除確認ウィンドウにクラスター名を入力し、[Delete (削除)] を選択してください。
6. この手順をアカウント内の他のすべてのクラスターに対して繰り返します。すべての削除操作が完了するまで待ちます。

## サービスリンク役割の手動による削除

サービスにリンクされた役割 `AWSServiceRoleForAmazonEKS` を削除するには IAM コンソール、AWS CLI、または AWS API を使用します。詳細については IAM ユーザーガイド の「[サービスにリンクされた役割の削除](#)」を参照してください。

## Amazon EKS のサービスにリンクされた役割がサポートされるリージョン

Amazon EKS ではこのサービスを利用できるすべてのリージョンで、サービスにリンクされた役割の使用がサポートされます。詳細については「[Amazon EKS endpoints and quotas](#)」(Amazon EKS エンドポイントとクォータ) を参照してください。

## アマゾン EKS ノードグループでのロールの使用

アマゾン EKS は AWS アイデンティティとアクセス管理 (IAM) の [サービスにリンクされた役割](#) を使用します。サービスにリンクされた役割はアマゾン EKS に直接リンクされた一意のタイプの IAM 役割です。サービスにリンクされた役割はアマゾン EKS で事前定義されています。この役割にはサービスがユーザーに代わって他の AWS のサービスを呼び出すために必要な、すべてのアクセス許可が付与されています。

サービスにリンクされた役割を使用することで、必要なアクセス許可を手動で追加する必要がなくなるため、アマゾン EKS の設定が簡単になります。サービスにリンクされた役割のアクセス許可はアマゾン EKS により定義されます。特に指定されている場合を除き、アマゾン EKS のみがその役割を引き受けることができます。定義される許可は信頼ポリシーと許可ポリシーに含まれており、その許可ポリシーを他の IAM エンティティにアタッチすることはできません。

サービスリンク役割はまずその関連リソースを削除しなければ削除できません。これにより、リソースへのアクセス許可を不用意に削除することが防止され、アマゾン EKS リソースが保護されます。

サービスにリンクされた役割をサポートする他のサービスについては「[IAM と連動する AWS サービス](#)」を参照し、[サービス-linked roles] (サービスにリンクされた役割) の列内で [Yes] (はい) と表記されたサービスを確認してください。サービスにリンクされた役割に関するドキュメントをサービスで表示するには[はい] リンクを選択してください。

## アマゾン EKS でのサービスにリンクされた役割のアクセス許可

アマゾン EKS は `AWSServiceRoleForAmazonEKSNodegroup` という名前のサービスにリンクされたロールを使用します。このロールにより、アマゾン EKS はアカウント内のノードグループを管理できます。アタッチされた `AWSServiceRoleForAmazonEKSNodegroup` ポリシーにより、ロールは 自動スケーリング グループ、セキュリティグループ、起動テンプレート、および IAM インスタンスプロファイルのリソースを管理できるようになります。詳細については「[the section called “AWS 管理ポリシー: AWSServiceRoleForAmazonEKSNodegroup”](#)」を参照してください。

`AWSServiceRoleForAmazonEKSNodegroup` サービスリンク役割は役割の引き受けについて以下のサービスを信頼します。

- `eks-nodegroup.amazonaws.com`

ロールのアクセス許可ポリシーは指定したリソースに対して以下のアクションを実行することを アマゾン EKS に許可します。

- [AWSServiceRoleForアマゾンEKSNodegroup](#)

サービスにリンクされたロールの作成、編集、削除を IAM エンティティ (ユーザー、グループ、ロールなど) に許可するにはアクセス許可を設定する必要があります。詳細についてはIAM ユーザーガイドの「[サービスにリンクされた役割のアクセス許可](#)」を参照してください。

## アマゾン EKS でのサービスにリンクされた役割の作成

サービスリンクロールを手動で作成する必要はありません。AWS Management Console、AWS CLI、または AWS API でノードグループを作成すると、アマゾン EKS によってサービスリンクロールが作成されます。

### Important

このサービスリンクロールはこのロールでサポートされている機能を使用する別のサービスでアクションが完了した場合にアカウントに表示されます。2017 年 1 月 1 日より前に アマゾン EKS サービスを使用している場合、サービスにリンクされたロールのサポートが開始

された時点で、AWSServiceRoleForAmazonEKSCluster ロールは アマゾン EKS によりアカウントに作成されています。詳細については[IAM アカウントに新しいロールが表示される](#)を参照してください。

## Amazon EKS でのサービスにリンクされた役割の作成 (AWS API)

サービスリンクロールを手動で作成する必要はありません。AWS Management Console、AWS CLI、または AWS API でマネージド型ノードグループを作成すると、アマゾン EKS によってサービスリンクロールが作成されます。

このサービスリンクロールを削除した後で再度作成する必要がある場合は同じ方法でアカウントにロールを再作成できます。別のマネージド型ノードグループを作成すると、Amazon EKS によってサービスリンク役割が再度作成されます。

## アマゾン EKS でのサービスにリンクされた役割の編集

アマゾン EKS ではAWSServiceRoleForAmazonEKSCluster のサービスにリンクされた役割を編集することはできません。サービスにリンクされた役割を作成すると、多くのエンティティによって役割が参照される可能性があるため、役割名を変更することはできません。ただし、IAM を使用してロールの説明を編集することはできます。詳細については『IAM ユーザーガイド』の「[サービスにリンクされた役割の編集](#)」を参照してください。

## アマゾン EKS でのサービスにリンクされた役割の削除

サービスリンク役割が必要な機能またはサービスが不要になった場合にはその役割を削除することをお勧めします。そうすることで、モニタリングや保守が積極的に行われていない未使用のエンティティを排除できます。ただし、手動で削除する前に、サービスリンク役割をクリーンアップする必要があります。

## サービスリンク役割のクリーンアップ

IAM を使用してサービスにリンクされた役割を削除するには最初に、その役割で使用されているリソースをすべて削除する必要があります。

### Note

リソースの削除を試みた際に、対応する役割が アマゾン EKS サービスで使用されている場合、削除が失敗することがあります。失敗した場合は数分待ってから操作を再試行してください。



1. [アマゾン EKS コンソール](#)を開きます。
2. 左のナビゲーションペインで [クラスター] を選択してください。
3. [Compute] (コンピューティング) タブを選択してください。
4. [Node groups] (ノードグループ) セクションで、削除するノードグループを選択してください。
5. 削除確認ウィンドウにノードグループの名前を入力し、[Delete (削除)] を選択してください。
6. この手順をクラスター内の他のすべてのノードグループに対して繰り返します。すべての削除操作が完了するまで待ちます。

## サービスリンク役割の手動による削除

サービスにリンクされた役割 `AWSServiceRoleForAmazonEKSTernodegroup` を削除するには IAM コンソール、AWS CLI、または AWS API を使用します。詳細については IAM ユーザーガイド の「[サービスにリンクされた役割の削除](#)」を参照してください。

## Amazon EKS のサービスにリンクされた役割がサポートされるリージョン

Amazon EKS ではこのサービスを利用できるすべてのリージョンで、サービスにリンクされた役割の使用がサポートされます。詳細については「[Amazon EKS endpoints and quotas](#)」(Amazon EKS エンドポイントとクォータ) を参照してください。

## Amazon EKS Fargate プロファイルの役割の使用

アマゾン EKS は AWS アイデンティティとアクセス管理 (IAM) の [サービスにリンクされた役割](#) を使用します。サービスにリンクされた役割はアマゾン EKS に直接リンクされた一意のタイプの IAM 役割です。サービスにリンクされた役割はアマゾン EKS で事前定義されています。この役割にはサービスがユーザーに代わって他の AWS のサービスを呼び出すために必要な、すべてのアクセス許可が付与されています。

サービスにリンクされた役割を使用することで、必要なアクセス許可を手動で追加する必要がなくなるため、アマゾン EKS の設定が簡単になります。サービスにリンクされた役割のアクセス許可はアマゾン EKS により定義されます。特に指定されている場合を除き、アマゾン EKS のみがその役割を引き受けることができます。定義される許可は信頼ポリシーと許可ポリシーに含まれており、その許可ポリシーを他の IAM エンティティにアタッチすることはできません。

サービスリンク役割はまずその関連リソースを削除しなければ削除できません。これにより、リソースへのアクセス許可を不用意に削除することが防止され、アマゾン EKS リソースが保護されます。

サービスにリンクされた役割をサポートする他のサービスについては「[IAM と連動する AWS サービス](#)」を参照し、[サービス-linked roles] (サービスにリンクされた役割) の列内で [Yes] (はい) と表記さ

れたサービスを確認してください。サービスにリンクされた役割に関するドキュメントをサービスで表示するには[はい] リンクを選択してください。

## アマゾン EKS でのサービスにリンクされた役割のアクセス許可

アマゾン EKS は `AWSServiceRoleForAmazonEKSFargate` という名前のサービスにリンクされた役割を使用します。このロールにより、Amazon EKS Fargate は Fargate Pod に必要な VPC ネットワーキングを設定できます。アタッチされたポリシーにより、役割は エラスティック・ネットワーク・インターフェイス の作成と削除、エラスティック・ネットワーク・インターフェイスとリソースの記述ができるようになります。

`AWSServiceRoleForAmazonEKSFargate` サービスリンク役割は役割の引き受けについて以下のサービスを信頼します。

- `eks-fargate.amazonaws.com`

役割のアクセス許可ポリシーは指定したリソースに対して以下のアクションを実行することを アマゾン EKS に許可します。

- [AmazonEKSFargateServiceRolePolicy](#)

サービスリンク役割の作成、編集、削除を IAM エンティティ (ユーザー、グループ、役割など) に許可するにはアクセス許可を設定する必要があります。詳細についてはIAM ユーザーガイドの「[サービスにリンクされた役割のアクセス許可](#)」を参照してください。

## アマゾン EKS でのサービスにリンクされた役割の作成

サービスリンク役割を手動で作成する必要はありません。AWS Management Console、AWS CLI、または AWS API で Fargate プロファイルを作成すると、アマゾン EKS がサービスリンク役割を作成します。

### Important

このサービスリンク役割はこの役割でサポートされている機能を使用する別のサービスでアクションが完了した場合にアカウントに表示されます。2019 年 12 月 13 日より前に Amazon EKS サービスを使用している場合、サービスにリンクされた役割のサポートが開始された時点で、`AWSServiceRoleForAmazonEKSFargate` は Amazon EKS によりアカウントに作成されています。詳細については「[IAM アカウントに表示される新しい役割](#)」を参照してください。



## Amazon EKS でのサービスにリンクされた役割の作成 (AWS API)

サービスリンク役割を手動で作成する必要はありません。AWS Management Console、AWS CLI、または AWS API で Fargate プロファイルを作成すると、アマゾン EKS がサービスリンク役割を作成します。

このサービスリンク役割を削除した後で再度作成する必要がある場合は同じ方法でアカウントに役割を再作成できます。別のマネージド型ノードグループを作成すると、Amazon EKS によってサービスリンク役割が再度作成されます。

## アマゾン EKS でのサービスにリンクされた役割の編集

アマゾン EKS では `AWSServiceRoleForAmazonEKSForFargate` のサービスにリンクされた役割を編集することはできません。サービスにリンクされた役割を作成すると、多くのエンティティによって役割が参照される可能性があるため、役割名を変更することはできません。ただし、IAM を使用してロールの説明を編集することはできます。詳細については『IAM ユーザーガイド』の「[サービスにリンクされた役割の編集](#)」を参照してください。

## アマゾン EKS でのサービスにリンクされた役割の削除

サービスリンク役割が必要な機能またはサービスが不要になった場合にはその役割を削除することをお勧めします。そうすることで、モニタリングや保守が積極的に行われていない未使用のエンティティを排除できます。ただし、手動で削除する前に、サービスリンク役割をクリーンアップする必要があります。

## サービスリンク役割のクリーンアップ

IAM を使用してサービスにリンクされた役割を削除するには最初に、その役割で使用されているリソースをすべて削除する必要があります。

### Note

リソースの削除を試みた際に、対応する役割が アマゾン EKS サービスで使用されている場合、削除が失敗することがあります。失敗した場合は数分待ってから操作を再試行してください。

1. [アマゾン EKS コンソール](#)を開きます。
2. 左のナビゲーションペインで [Clusters] (クラスター) を選択してください。
3. [Clusters] (クラスター) ページで、クラスターを選択してください。

4. [Compute] (コンピューティング) タブを選択してください。
5. [Fargate profiles] (Fargateプロファイル) セクションに Fargate プロファイルがある場合、それぞれを個別に選択し、[Delete] (削除)を選択してください。
6. 削除確認ウィンドウにプロファイルの名前を入力し、[Delete (削除)] を選択してください。
7. クラスター内のその他の Fargate プロファイルと、アカウント内のその他のクラスターについて、この手順を繰り返します。

## サービスリンク役割の手動による削除

IAM コンソール、AWS CLI、または AWS API を使用し

て、AWSServiceRoleForAmazonEKSFargate のサービスにリンクされた役割を削除します。詳細についてはIAM ユーザーガイドの「[サービスにリンクされた役割の削除](#)」を参照してください。

## Amazon EKS のサービスにリンクされた役割がサポートされるリージョン

Amazon EKS ではこのサービスを利用できるすべてのリージョンで、サービスにリンクされた役割の使用がサポートされます。詳細については「[Amazon EKS endpoints and quotas](#)」(Amazon EKS エンドポイントとクォータ)を参照してください。

## ロールを使用して Kubernetes クラスターを Amazon EKS に接続する

アマゾン EKS はAWS アイデンティティとアクセス管理 (IAM) の[サービスにリンクされた役割](#)を使用します。サービスにリンクされた役割はアマゾン EKS に直接リンクされた一意のタイプの IAM 役割です。サービスにリンクされた役割はアマゾン EKS で事前定義されています。この役割にはサービスがユーザーに代わって他の AWS のサービスを呼び出すために必要な、すべてのアクセス許可が付与されています。

サービスにリンクされた役割を使用することで、必要なアクセス許可を手動で追加する必要がなくなるため、アマゾン EKS の設定が簡単になります。サービスにリンクされた役割のアクセス許可はアマゾン EKS により定義されます。特に指定されている場合を除き、アマゾン EKS のみがその役割を引き受けることができます。定義される許可は信頼ポリシーと許可ポリシーに含まれており、その許可ポリシーを他の IAM エンティティにアタッチすることはできません。

サービスリンク役割はまずその関連リソースを削除しなければ削除できません。これにより、リソースへのアクセス許可を不用意に削除することが防止され、アマゾン EKS リソースが保護されます。

サービスにリンクされた役割をサポートする他のサービスについては「[IAM と連動する AWS サービス](#)」を参照し、[サービス-linked roles] (サービスにリンクされた役割) の列内で [Yes] (はい) と表記さ

れたサービスを確認してください。サービスにリンクされた役割に関するドキュメントをサービスで表示するには[はい] リンクを選択してください。

## アマゾン EKS でのサービスにリンクされた役割のアクセス許可

アマゾン EKS は `AWSServiceRoleForAmazonEKSCloudConnector` という名前のサービスにリンクされた役割を使用します。このロールにより、Amazon EKS は Kubernetes クラスターに接続できます。アタッチされたポリシーにより、ロールは、登録された Kubernetes クラスターに接続するために必要なリソースを管理できます。

`AWSServiceRoleForAmazonEKSCloudConnector` サービスリンク役割は役割の引き受けについて以下のサービスを信頼します。

- `eks-cloud-connector.amazonaws.com`

役割のアクセス許可ポリシーは指定したリソースに対して以下のアクションを実行することを アマゾン EKS に許可します。

- [AmazonEKSCloudConnectorServiceRolePolicy](#)

サービスリンク役割の作成、編集、削除を IAM エンティティ (ユーザー、グループ、役割など) に許可するにはアクセス許可を設定する必要があります。詳細についてはIAM ユーザーガイドの「[サービスにリンクされた役割のアクセス許可](#)」を参照してください。

## アマゾン EKS でのサービスにリンクされた役割の作成

サービスにリンクされた役割を手動で作成してクラスターを接続する必要はありません。AWS Management Console、AWS CLI、`eksctl` または AWS API でクラスターを接続すると、Amazon EKS によってサービスにリンクされた役割が作成されます。

このサービスリンク役割を削除した後で再度作成する必要がある場合は同じ方法でアカウントに役割を再作成できます。サービスにリンクされた役割はクラスターの接続時に Amazon EKS で自動的に再作成されます。

## アマゾン EKS でのサービスにリンクされた役割の編集

アマゾン EKS では `AWSServiceRoleForAmazonEKSCloudConnector` のサービスにリンクされた役割を編集することはできません。サービスにリンクされた役割を作成すると、多くのエンティティによって役割が参照される可能性があるため、役割名を変更することはできません。ただし、IAM を使用

してロールの説明を編集することはできます。詳細については『IAM ユーザーガイド』の「[サービスにリンクされた役割の編集](#)」を参照してください。

## アマゾン EKS でのサービスにリンクされた役割の削除

サービスリンク役割が必要な機能またはサービスが不要になった場合にはその役割を削除することをお勧めします。そうすることで、モニタリングや保守が積極的に行われていない未使用のエンティティを排除できます。ただし、手動で削除する前に、サービスリンク役割をクリーンアップする必要があります。

### サービスリンク役割のクリーンアップ

IAM を使用してサービスにリンクされた役割を削除するには最初に、その役割で使用されているリソースをすべて削除する必要があります。

#### Note

リソースの削除を試みた際に、対応する役割が アマゾン EKS サービスで使用されている場合、削除が失敗することがあります。失敗した場合は数分待ってから操作を再試行してください。

1. [アマゾン EKS コンソール](#)を開きます。
2. 左のナビゲーションペインで [Clusters] (クラスター) を選択してください。
3. [Clusters] (クラスター) ページで、クラスターを選択してください。
4. 登録解除タブをクリックし、Ok タブを選択してください。

### サービスリンク役割の手動による削除

IAM コンソール、AWS CLI、または AWS API を使用し

て、AWSServiceRoleForAmazonEKSCluster のサービスにリンクされた役割を削除します。詳細については IAM ユーザーガイド の「[サービスにリンクされた役割の削除](#)」を参照してください。

## アウトポスト で アマゾン EKS ローカルクラスターの役割の使用

アマゾン エラスティック K8s サービス はAWS アイデンティティとアクセス管理 (IAM) の[サービスにリンクされた役割](#)を使用します。サービスにリンクされた役割はアマゾン EKS に直接リンクされた一意のタイプの IAM 役割です。サービスにリンクされた役割はアマゾン EKS で事前定義

されています。この役割にはサービスがユーザーに代わって他の AWS のサービス呼び出すために必要な、すべてのアクセス許可が付与されています。

サービスにリンクされた役割を使用することで、必要なアクセス許可を手動で追加する必要がなくなるため、アマゾン EKS の設定が簡単になります。サービスにリンクされた役割のアクセス許可はアマゾン EKS により定義されます。特に指定されている場合を除き、アマゾン EKS のみがその役割を引き受けることができます。定義される許可は信頼ポリシーと許可ポリシーに含まれており、その許可ポリシーを他の IAM エンティティにアタッチすることはできません。

サービスリンク役割はまずその関連リソースを削除しなければ削除できません。これにより、リソースへのアクセス許可を不用意に削除することが防止され、アマゾン EKS リソースが保護されます。

サービスにリンクされた役割をサポートする他のサービスについては「[IAM と連動する AWS サービス](#)」を参照し、[サービス-linked roles] (サービスにリンクされた役割) の列内で [Yes] (はい) と表記されたサービスを確認してください。サービスにリンクされた役割に関するドキュメントをサービスで表示するには[はい] リンクを選択してください。

アマゾン EKS でのサービスにリンクされた役割のアクセス許可

アマゾン EKS は `AWSServiceRoleForAmazonEKSLocalOutpost` という名前のサービスにリンクされた役割を使用します。この役割により、アマゾン EKS はアカウント内のローカルクラスターを管理できます。アタッチされたポリシーにより、役割はネットワークインターフェイス、セキュリティグループ、ログ、アマゾン EC2 インスタンスのリソースを管理できるようになります。

#### Note

`AWSServiceRoleForAmazonEKSLocalOutpost` サービスにリンクされた役割はクラスターの作成に必要な役割とは異なります。詳細については「[the section called “クラスター IAM ロール”](#)」を参照してください。

`AWSServiceRoleForAmazonEKSLocalOutpost` サービスリンク役割は役割の引き受けについて以下のサービスを信頼します。

- `outposts.eks-local.amazonaws.com`

役割のアクセス許可ポリシーは指定したリソースに対して以下のアクションを実行することをアマゾン EKS に許可します。

- [アマゾンEKSサービスロールポリシー](#)

サービスにリンクされた役割の作成、編集、削除を IAM エンティティ (ユーザー、グループ、役割など) に許可するにはアクセス許可を設定する必要があります。詳細については IAM ユーザーガイドの「[サービスにリンクされた役割のアクセス許可](#)」を参照してください。

## アマゾン EKS でのサービスにリンクされた役割の作成

サービスリンク役割を手動で作成する必要はありません。AWS Management Console、AWS CLI、または AWS API でクラスターを作成すると、アマゾン EKS によってサービスリンク役割が作成されます。

このサービスリンク役割を削除した後で再度作成する必要がある場合は同じ方法でアカウントに役割を再作成できます。サービスにリンクされた役割はクラスターの作成時に アマゾン EKS で自動的に再作成されます。

## アマゾン EKS でのサービスにリンクされた役割の編集

アマゾン EKS では `AWSServiceRoleForAmazonEKSLocalOutpost` のサービスにリンクされた役割を編集することはできません。サービスリンク役割を作成すると、多くのエンティティによって役割が参照される可能性があるため、役割名を変更することはできません。ただし、IAM を使用した役割記述の編集はできます。詳細については『IAM ユーザーガイド』の「[サービスにリンクされた役割の編集](#)」を参照してください。

## アマゾン EKS でのサービスにリンクされた役割の削除

サービスリンク役割が必要な機能またはサービスが不要になった場合にはその役割を削除することをお勧めします。そうすることで、モニタリングや保守が積極的に行われていない未使用のエンティティを排除できます。ただし、手動で削除する前に、サービスリンク役割をクリーンアップする必要があります。

## サービスリンク役割のクリーンアップ

IAM を使用してサービスにリンクされた役割を削除するには最初に、その役割で使用されているリソースをすべて削除する必要があります。

### Note

リソースの削除を試みた際に、対応する役割が アマゾン EKS サービスで使用されている場合、削除が失敗することがあります。失敗した場合は数分待ってから操作を再試行してください。



1. [アマゾン EKS コンソール](#)を開きます。
2. 左のナビゲーションペインで アマゾン EKS [Clusters] (クラスター) を選択してください。
3. クラスターにノードグループまたは Fargate プロファイルがある場合はクラスターを削除する前にそれらを削除する必要があります。詳細については「[the section called “削除”](#)」および「[the section called “プロファイルを削除する”](#)」を参照してください。
4. [Clusters (クラスター)] ページで、削除するクラスターを選択し、[Delete (削除)] を選択してください。
5. 削除確認ウィンドウにクラスター名を入力し、[Delete (削除)] を選択してください。
6. この手順をアカウント内の他のすべてのクラスターに対して繰り返します。すべての削除操作が完了するまで待ちます。

## サービスリンク役割の手動による削除

サービスにリンクされた役割 `AWSServiceRoleForAmazonEKSLocalOutpost` を削除するには IAM コンソール、AWS CLI、または AWS API を使用します。詳細については IAM ユーザーガイドの「[サービスにリンクされた役割の削除](#)」を参照してください。

## Amazon EKS のサービスにリンクされた役割がサポートされるリージョン

Amazon EKS ではこのサービスを利用できるすべてのリージョンで、サービスにリンクされた役割の使用がサポートされます。詳細については「[Amazon EKS endpoints and quotas](#)」(Amazon EKS エンドポイントとクォータ) を参照してください。

## Amazon EKS ダッシュボードのロールの使用

Amazon EKS ダッシュボードでは、このサービスにリンクされたロールを使用して、複数のリージョンとアカウントからクラスターリソースに関する情報を集約します。ダッシュボードは AWS Organizations と統合され、組織内の複数のアカウントに関する情報を安全に取り込みます。

Amazon EKS ダッシュボードの詳細については、「[the section called “Amazon EKS ダッシュボード”](#)」を参照してください。

## 背景

Amazon EKS は AWS アイデンティティとアクセス管理 (IAM) の [サービスにリンクされた役割](#) を使用します。サービスにリンクされた役割は Amazon EKS に直接リンクされた一意のタイプの IAM 役割です。サービスにリンクされた役割は Amazon EKS で事前定義されています。この役割にはサービスがユーザーに代わって他の AWS のサービスを呼び出すために必要な、すべてのアクセス許可が付与されています。

サービスにリンクされた役割を使用することで、必要なアクセス許可を手動で追加する必要がなくなるため、Amazon EKS の設定が簡単になります。サービスにリンクされた役割のアクセス許可は Amazon EKS により定義されます。特に指定されている場合を除き、Amazon EKS のみがその役割を引き受けることができます。定義される許可は信頼ポリシーと許可ポリシーに含まれており、その許可ポリシーを他の IAM エンティティにアタッチすることはできません。

サービスリンク役割はまずその関連リソースを削除しなければ削除できません。これにより、リソースへのアクセス許可を不用意に削除することが防止され、Amazon EKS リソースが保護されます。

サービスにリンクされた役割をサポートする他のサービスについては「[IAM と連動する AWS サービス](#)」を参照し、[サービス-linked roles] (サービスにリンクされた役割) の列内で [Yes] (はい) と表記されたサービスを確認してください。サービスにリンクされた役割に関するドキュメントをサービスで表示するには[はい] リンクを選択してください。

### Amazon EKS でのサービスにリンクされた役割のアクセス許可

Amazon EKS は `AWSServiceRoleForAmazonEKSDashboard` という名前のサービスにリンクされた役割を使用します。このロールにより、Amazon EKS はクラスターリソースに関して集約された情報を含む EKS ダッシュボードを生成して表示できます。アタッチされた `AmazonEKSDashboardServiceRolePolicy` ポリシーにより、ロールは 自動スケーリング グループ、セキュリティグループ、起動テンプレート、および IAM インスタンスプロファイルのリソースを管理できるようになります。詳細については、「[the section called “AWS マネージドポリシー: AmazonEKSDashboardServiceRolePolicy”](#)」を参照してください。

`AWSServiceRoleForAmazonEKSDashboard` サービスリンク役割は役割の引き受けについて以下のサービスを信頼します。

- `dashboard.eks.amazonaws.com`

JSON ポリシードキュメントの最新バージョンを確認するには『AWS管理ポリシーリファレンスガイド』の「[AmazonEKSDashboardServiceRolePolicy](#)」を参照してください。

サービスリンク役割の作成、編集、削除を IAM エンティティ (ユーザー、グループ、役割など) に許可するにはアクセス許可を設定する必要があります。詳細についてはIAM ユーザーガイドの「[サービスにリンクされた役割のアクセス許可](#)」を参照してください。

### Amazon EKS でのサービスにリンクされた役割の作成

サービスリンク役割を手動で作成する必要はありません。AWS コンソールでダッシュボードを有効にすると、Amazon EKS によってサービスにリンクされたロールが作成されます。



EKS ダッシュボードの有効化についての詳細は、「[the section called “組織を設定する”](#)」を参照してください。

**⚠ Important**

このサービスリンク役割はこの役割でサポートされている機能を使用する別のサービスでアクションが完了した場合にアカウントに表示されます。

## Amazon EKS でのサービスにリンクされた役割の編集

Amazon EKS ではAWSServiceRoleForAmazonEKSDashboard のサービスにリンクされた役割を編集することはできません。サービスにリンクされた役割を作成すると、多くのエンティティによって役割が参照される可能性があるため、役割名を変更することはできません。ただし、IAM を使用してロールの説明を編集することはできます。詳細については『IAM ユーザーガイド』の「[サービスにリンクされた役割の編集](#)」を参照してください。

## Amazon EKS でのサービスにリンクされた役割の削除

サービスリンク役割が必要な機能またはサービスが不要になった場合にはその役割を削除することをお勧めします。そうすることで、モニタリングや保守が積極的に行われていない未使用のエンティティを排除できます。ただし、手動で削除する前に、サービスリンク役割をクリーンアップする必要があります。

### サービスリンク役割のクリーンアップ

IAM を使用してサービスにリンクされた役割を削除するには最初に、その役割で使用されているリソースをすべて削除する必要があります。

**i Note**

リソースの削除を試みた際に、対応する役割が Amazon EKS サービスで使用されている場合、削除が失敗することがあります。失敗した場合は数分待ってから操作を再試行してください。

EKS ダッシュボードの無効化についての詳細は、「[the section called “組織を設定する”](#)」を参照してください。

## サービスリンク役割の手動による削除

サービスにリンクされた役割 `AWSServiceRoleForAmazonEKSDashboard` を削除するには IAM コンソール、AWS CLI、または AWS API を使用します。詳細については IAM ユーザーガイド の「[サービスにリンクされた役割の削除](#)」を参照してください。

## Amazon EKS のサービスにリンクされた役割がサポートされるリージョン

Amazon EKS ではこのサービスを利用できるすべてのリージョンで、サービスにリンクされた役割の使用がサポートされます。詳細については「[Amazon EKS endpoints and quotas](#)」(Amazon EKS エンドポイントとクォータ) を参照してください。

## Amazon EKS Pod 実行 IAM ロール

Pod を AWS Fargate インフラストラクチャで実行するには、Amazon EKS Pod 実行ロールが必要になります。

クラスターが AWS Fargate インフラストラクチャ上で Pod を作成する場合は、Fargate インフラストラクチャ上で実行されているコンポーネントがユーザーに代わって AWS API を呼び出す必要があります。これは、Amazon ECR からコンテナイメージをプルしたり、ログを他の AWS サービスにルーティングしたりするなどのアクションを実行できるようにするためです。Amazon EKS の Pod 実行ロールにより、これらを行うための IAM アクセス許可が付与されます。

Fargate プロファイルを作成するときは、プロファイルを使用して Fargate インフラストラクチャで Amazon EKS コンポーネントを実行できるように、Pod 実行ロールを指定する必要があります。このロールは、承認のためにクラスターの Kubernetes [ロールベースのアクセス制御](#) (RBAC) に追加されます。これにより、Fargate インフラストラクチャで実行されている kubelet が Amazon EKS クラスターに登録され、クラスター内でノードとして表示されるようになります。

### Note

Fargate プロファイルには、Amazon EC2 ノードグループとは異なる IAM ロールが必要です。

### Important

Fargate Pod で実行されているコンテナは、Pod 実行ロールに関連付けられた IAM アクセス許可を引き受けることはできません。Fargate Pod 内のコンテナに他の AWS サービスへの

アクセス許可を付与するには、[IAM ロールをサービスアカウントとして使用する](#)必要があります。

Fargate プロファイルを作成する前に、[AmazonEKSFargatePodExecutionRolePolicy](#) で IAM ロールを作成する必要があります。

## 正しく設定された既存の Pod 実行ロールをチェックする

次の手順を使用して、正しく設定された Amazon EKS の Pod 実行ロールがアカウントに既に存在しているかどうかをチェックします。「混乱した代理」のセキュリティ上の問題を回避するには、ロールが SourceArn に基づいてアクセスを制限することが重要です。必要に応じて実行ロールを変更し、他のクラスター上で Fargate プロファイルのサポートを含めることができます。

1. IAM コンソール (<https://console.aws.amazon.com/iam/>) を開きます。
2. 左のナビゲーションペインで、[ロール] を選択してください。
3. [ロール] ページで、ロールの一覧から AmazonEKSFargatePodExecutionRole を検索します。ロールが存在しない場合は、「[the section called “Amazon EKS での Pod 実行ロールの作成”](#)」を参照してロールを作成します。ロールが存在する場合は、そのロールを選択します。
4. [AmazonEKSFargatePodExecutionRole] ページで、次の操作を実行します。
  - a. [許可] を選択します。
  - b. AmazonEKSFargatePodExecutionRolePolicy Amazon マネージドポリシーがロールにアタッチされていることを確認します。
  - c. [信頼関係] を選択します。
  - d. [信頼ポリシーを編集] を選択します。
5. [信頼ポリシーを編集] ページで、信頼関係に次のポリシーが含まれており、クラスター上で Fargate プロファイルの行が存在していることを確認します。そうである場合は、[キャンセル] を選択します。

```
{
 "Version": "2012-10-17",
 "Statement": [
 {
 "Effect": "Allow",
 "Condition": {
 "ArnLike": {
 "aws:SourceArn": "arn:aws:eks:region-code:111122223333:fargateprofile/my-cluster/*"
 }
 }
 }
]
}
```

```
 }
 },
 "Principal": {
 "Service": "eks-fargate-pods.amazonaws.com"
 },
 "Action": "sts:AssumeRole"
}
]
}
```

ポリシーは一致するが、クラスター上で Fargate プロファイルを指定する行が存在しない場合は、ArnLike オブジェクトの上部に次の行を追加します。*region-code* はクラスターがある AWS リージョンに、*111122223333* はアカウント ID に、*my-cluster* はクラスターの名前に置き換えます。

```
"aws:SourceArn": "arn:aws:eks:region-code:111122223333:fargateprofile/my-cluster/*",
```

ポリシーが一致しない場合は、前のポリシー全体をフォームにコピーして、[ポリシーの更新] を選択します。*region-code* を、クラスターのある AWS リージョンに置き換えます。アカウントで、すべての AWS リージョンで同じロールを使用する場合は、*region-code* を \* に置き換えます。*111122223333* をアカウント ID に、*my-cluster* をクラスター名に置き換えます。アカウント内のすべてのクラスターに同じロールを使用する場合は、*my-cluster* を \* に置き換えます。

## Amazon EKS での Pod 実行ロールの作成

クラスター用の Amazon EKS Pod 実行ロールをまだ作成していない場合は、AWS Management Console または AWS CLI を使用して作成できます。

### AWS Management Console

- a. IAM コンソール (<https://console.aws.amazon.com/iam/>) を開きます。
- b. 左のナビゲーションペインで、[ロール] を選択してください。
- c. [ロール] ページで、[ロールの作成] を選択してください。
- d. [信頼されたエンティティを選択] ページで、以下の操作を実行してください：
  - i. [信頼するエンティティのタイプ] で [AWS サービス] を選択します。
  - ii. [他の AWS サービスのユースケース] ドロップダウンリストから、[EKS] を選択します。
  - iii. [EKS - Fargate Pod] を選択します。

- iv. [Next] を選択します。
- e. [アクセス許可を追加] ページで [次へ] を選択します。
- f. [名前を付けて、レビューし、作成する] ページで、以下の操作を実行します。
  - i. [ロール名] に、AmazonEKSFargatePodExecutionRole などのロールの一意の名前を入力します。
  - ii. [タグの追加 (オプション)] で、タグをキーバリューのペアとして添付して、メタデータをロールに追加します。IAM でのタグの使用に関する詳細については『IAM ユーザーガイド』の「[IAM リソースにタグを付ける](#)」を参照してください。
  - iii. [ロールの作成] を選択します。
- g. [ロール] ページで、ロールの一覧から AmazonEKSFargatePodExecutionRole を検索します。ロールを選択します。
- h. [AmazonEKSFargatePodExecutionRole] ページで、次の操作を実行します。
  - i. [信頼関係] を選択します。
  - ii. [信頼ポリシーを編集] を選択します。
- i. [信頼ポリシーを編集] ページで、次の操作を実行します。
  - i. 次の内容をコピーして、[信頼ポリシーを編集] フォームに貼り付けます。 *region-code* を、クラスターのある AWS リージョンに置き換えます。アカウントで、すべての AWS リージョンで同じロールを使用する場合は、*region-code* を \* に置き換えます。 *111122223333* をアカウント ID に、*my-cluster* をクラスター名に置き換えます。アカウント内のすべてのクラスターに同じロールを使用する場合は、*my-cluster* を \* に置き換えます。

```
{
 "Version": "2012-10-17",
 "Statement": [
 {
 "Effect": "Allow",
 "Condition": {
 "ArnLike": {
 "aws:SourceArn": "arn:aws:eks:region-code:111122223333:fargateprofile/my-cluster/*"
 }
 },
 "Principal": {
 "Service": "eks-fargate-pods.amazonaws.com"
 },
 "Action": "sts:AssumeRole"
 }
]
}
```

```
 }
]
}
```

ii. [ポリシーの更新] を選択してください。

## AWS CLI

- a. 次の内容をコピーして、pod-execution-role-trust-policy.json という名前のファイルに貼り付けます。*region-code* を、クラスターのある AWS リージョンに置き換えます。アカウントで、すべての AWS リージョンで同じロールを使用する場合は、*region-code* を \* に置き換えます。*111122223333* をアカウント ID に、*my-cluster* をクラスター名に置き換えます。アカウント内のすべてのクラスターに同じロールを使用する場合は、*my-cluster* を \* に置き換えます。

```
{
 "Version": "2012-10-17",
 "Statement": [
 {
 "Effect": "Allow",
 "Condition": {
 "ArnLike": {
 "aws:SourceArn": "arn:aws:eks:region-code:111122223333:fargateprofile/
my-cluster/*"
 }
 },
 "Principal": {
 "Service": "eks-fargate-pods.amazonaws.com"
 },
 "Action": "sts:AssumeRole"
 }
]
}
```

- b. Pod 実行 IAM ロールを作成します。

```
aws iam create-role \
 --role-name AmazonEKSFargatePodExecutionRole \
 --assume-role-policy-document file://"/pod-execution-role-trust-policy.json"
```

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

```
aws iam attach-role-policy \
 --role-name AmazonEKSFargatePodExecutionRole \
 --policy-name AmazonEKS_FargatePodExecutionRolePolicy
```

```
--policy-arn arn:aws:iam::aws:policy/AmazonEKSFargatePodExecutionRolePolicy \
--role-name AmazonEKSFargatePodExecutionRole
```

## Amazon EKS コネクタの IAM ロール

Kubernetes クラスターを接続して AWS Management Console に表示することができま  
す。Kubernetes クラスターに接続するには、IAM ロールを作成します。

### 既存の EKS Connector ロールの確認

以下の手順を使用して、アカウントに既に Amazon EKS コネクタロールがあるかどうかを確認でき  
ます。

1. IAM コンソール (<https://console.aws.amazon.com/iam/>) を開きます。
2. 左のナビゲーションペインで、[ロール] を選択してください。
3. ロールのリストで AmazonEKSConnectorAgentRole を検索しま  
す。AmazonEKSConnectorAgentRole が含まれているロールが存在しない場合は、[the  
section called “Amazon EKS コネクタエージェントロールの作成”](#) を参照してロールを作成しま  
す。AmazonEKSConnectorAgentRole が含まれているロールが存在する場合はこのロールを選  
択してアタッチされているポリシーを表示します。
4. [許可] を選択します。
5. AmazonEKSConnectorAgentPolicy 管理ポリシーがロールにアタッチされていることを確認しま  
す。ポリシーがアタッチされている場合、Amazon EKS Connector ロールは適切に設定されてい  
ます。
6. [信頼関係] を選択し、[信頼ポリシーの編集] を選択してください。
7. 信頼関係に以下のポリシーが含まれていることを確認します。信頼関係が以下のポリシーと一致  
する場合、[キャンセル] を選択してください。信頼関係が一致しない場合、ポリシーを [信頼ポリ  
シーの編集] ウィンドウにコピーし、[ポリシーの更新] を選択します。

```
{
 "Version": "2012-10-17",
 "Statement": [
 {
 "Effect": "Allow",
 "Principal": {
 "Service": [
 "ssm.amazonaws.com"
]
 }
 }
]
}
```

```
 },
 "Action": "sts:AssumeRole"
 }
]
}
```

## Amazon EKS コネクタエージェントロールの作成

コネクタエージェントロールを作成するには、AWS Management Console または AWS CloudFormation を使用できます。

### AWS CLI

- a. IAM ロールに使用する次の JSON が含まれる `eks-connector-agent-trust-policy.json` という名前のファイルを作成します。

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

- b. IAM ロールに使用する次の JSON が含まれる `eks-connector-agent-policy.json` という名前のファイルを作成します。

```
{
 "Version": "2012-10-17",
 "Statement": [
 {
 "Sid": "SsmControlChannel",
 "Effect": "Allow",
 "Action": [
 "ssmmessages:CreateControlChannel"
]
 }
]
}
```



```

],
 "Resource": "arn:aws:eks:*:*:cluster/*"
 },
 {
 "Sid": "ssmDataplaneOperations",
 "Effect": "Allow",
 "Action": [
 "ssmmessages:CreateDataChannel",
 "ssmmessages:OpenDataChannel",
 "ssmmessages:OpenControlChannel"
],
 "Resource": "*"
 }
]
}

```

- c. 前のリストアイテムで作成した信頼ポリシーとポリシーを使用して、Amazon EKS Connector エージェントロールを作成します。

```

aws iam create-role \
 --role-name AmazonEKSCoordinatorAgentRole \
 --assume-role-policy-document file://eks-coordinator-agent-trust-policy.json

```

- d. Amazon EKS Connector エージェントのロールにポリシーを添付します。

```

aws iam put-role-policy \
 --role-name AmazonEKSCoordinatorAgentRole \
 --policy-name AmazonEKSCoordinatorAgentPolicy \
 --policy-document file://eks-coordinator-agent-policy.json

```

## AWS CloudFormation

- a. 以下の AWS CloudFormation テンプレートを、テキストファイルとしてローカルシステムに保存します。

### Note

このテンプレートでは、ここで作成されなければ `registerCluster` API が呼び出されたときに作成される、サービスにリンクされたロールも作成されます。詳細については、「[the section called “クラスターコネクタの役割”](#)」を参照してください。

```

AWSTemplateFormatVersion: '2010-09-09'
Description: 'Provisions necessary resources needed to register clusters in EKS'
Parameters: {}
Resources:
 EKSConectorSLR:
 Type: AWS::IAM::ServiceLinkedRole
 Properties:
 AWSServiceName: eks-connector.amazonaws.com

 EKSConectorAgentRole:
 Type: AWS::IAM::Role
 Properties:
 AssumeRolePolicyDocument:
 Version: '2012-10-17'
 Statement:
 - Effect: Allow
 Action: ['sts:AssumeRole']
 Principal:
 Service: 'ssm.amazonaws.com'

 EKSConectorAgentPolicy:
 Type: AWS::IAM::Policy
 Properties:
 PolicyName: EKSConectorAgentPolicy
 Roles:
 - {Ref: 'EKSConectorAgentRole'}
 PolicyDocument:
 Version: '2012-10-17'
 Statement:
 - Effect: 'Allow'
 Action: ['ssmmessages:CreateControlChannel']
 Resource:
 - Fn::Sub: 'arn:${AWS::Partition}:eks:*:*:cluster/*'
 - Effect: 'Allow'
 Action: ['ssmmessages:CreateDataChannel',
 'ssmmessages:OpenDataChannel', 'ssmmessages:OpenControlChannel']
 Resource: "*"

Outputs:
 EKSConectorAgentRoleArn:
 Description: The agent role that EKS connector uses to communicate with AWS
 services.
```

```
Value: !GetAtt EKSCollectorAgentRole.Arn
```

- b. [AWS CloudFormation コンソール](#)を開きます。
- c. 新しいリソースを使用して [スタックを作成] を選択します (標準)。
- d. [テンプレートの指定] で、[テンプレートファイルのアップロード] を選択し、[ファイルの選択] を選択します。
- e. 作成したファイルを選択し、[Next (次へ)] を選択します。
- f. [スタックの名前] に eksCollectorAgentRole などのロール名を入力し、[次へ] を選択します。
- g. [スタックオプションの設定] ページで、[Next (次へ)] を選択します。
- h. [Review (レビュー)] ページの情報から、スタックにより IAM リソースが作成されることを確認し、[Create stack (スタックの作成)] を選択します。

## Amazon Elastic Kubernetes Service に関する AWS 管理ポリシー

AWS マネージドポリシーはAWS が作成および管理するスタンドアロンポリシーです。AWS マネージドポリシーは多くの一般的なユースケースに対してアクセス許可を提供するように設計されているため、ユーザー、グループ、役割へのアクセス権の割り当てを開始できます。

AWS マネージドポリシーは特定のユースケースに対して最小特権の許可を付与しない場合があることに注意してください。これはすべての AWS 顧客が使用できる状態になっているためです。ユースケースに固有の[カスタマー管理ポリシー](#)を定義して、アクセス許可を絞り込むことをお勧めします。

AWS マネージドポリシーで定義したアクセス権限は変更できません。AWS が AWS マネージドポリシーに定義されているアクセス許可を更新すると、更新はポリシーがアタッチされているすべてのプリンシパルアイデンティティ (ユーザー、グループ、役割) に影響します。新しい AWS サービスを起動するか、既存のサービスで新しい API オペレーションが使用可能になると、AWS が AWS マネージドポリシーを更新する可能性が最も高くなります。

詳細については「IAM ユーザーガイド」の「[AWS マネージドポリシー](#)」を参照してください。

### AWS 管理ポリシー: AmazonEKS\_CNI\_Policy

AmazonEKS\_CNI\_Policy ポリシーを IAM エンティティにアタッチできます。Amazon EC2 ノードグループを作成する前に、このポリシーを[ノードの IAM ロール](#)、または Amazon VPC CNI Plugin for Kubernetes 専用で使用する IAM ロールにアタッチする必要があります。これはユーザーに代わってアクションを実行できるようにするためです。プラグインでのみ使用される役割にポリシーを

アタッチすることをお勧めします。詳細については[the section called “Amazon VPC CNI”](#)および[the section called “IRSA 用の設定”](#)を参照してください。

## 許可の詳細

このポリシーには Amazon EKS に以下のタスクを完了させるための以下の権限が含まれています。

- **ec2:\*NetworkInterface** および **ec2:\*PrivateIpAddresses** - Amazon VPC CNI プラグインが Pod の Elastic Network Interface や IP アドレスのプロビジョニングなどのアクションを実行し、Amazon EKS で実行されるアプリケーションにネットワークを提供できるようにします。
- **ec2** 読み取りアクション - Amazon VPC CNI プラグインがインスタンスやサブネットを記述して、Amazon VPC サブネット内の空き IP アドレスの量を確認するなどのアクションを実行できるようにします。VPC CNI は各サブネットの空き IP アドレスを使用して、エラスティック・ネットワーク・インターフェイスの作成時に使用する空き IP アドレスが最も多いサブネットを選択できます。

JSON ポリシードキュメントの最新バージョンを確認するには『AWS管理ポリシーリファレンスガイド』の「[AmazonEKS\\_CNI\\_Policy](#)」を参照してください。

## AWS 管理ポリシー: AmazonEKSClusterPolicy

IAM エンティティに AmazonEKSClusterPolicy をアタッチできます。クラスターを作成する前に、このポリシーをアタッチした[クラスター IAM ロール](#)が必要となります。Amazon EKS によって管理される Kubernetes クラスターは、ユーザーに代わって他の AWS のサービス呼び出します。これにより、サービスで使用するリソースを管理します。

このポリシーには、Amazon EKS に以下のタスクを完了させるための以下の権限が含まれています。

- **autoscaling** – 自動スケーリング グループの設定を読み取り、更新します。これらの権限は Amazon EKS では使用されませんが、下位互換性のためにポリシーに残ります。
- **ec2** – Amazon EC2 ノードに関連付けられているボリュームとネットワークリソースを操作します。これは、Kubernetes コントロールプレーンがインスタンスをクラスターに結合し、Kubernetes 永続ボリュームによってリクエストされる Amazon EBS ボリュームを動的にプロビジョニングおよび管理できるようにするために必要です。
- **ec2** - VPC CNI によって作成された Elastic Network Interface を削除します。こうした権限付与が必要なのは、VPC CNI が予期せず終了した場合に、残った Elastic Network Interface を EKS によってクリーンアップするためです。

- **elasticloadbalancing** – Elastic ロードバランサー を操作し、ノードをターゲットとして追加します。これは、Kubernetes コントロールプレーンが Kubernetes サービスによってリクエストされる Elastic Load Balancer を動的にプロビジョニングできるようにするために必要です。
- **iam** – サービスにリンクされた役割を作成します。これは、Kubernetes コントロールプレーンが Kubernetes サービスによってリクエストされる Elastic Load Balancer を動的にプロビジョニングできるようにするために必要です。
- **kms** – AWS KMS からキーを読み取ります。これは、Kubernetes コントロールプレーンが etcd に保存される Kubernetes シークレットの[シークレット暗号化](#)を管理するために必要です。

JSON ポリシードキュメントの最新バージョンを確認するには『AWS管理ポリシーリファレンスガイド』の「[AmazonEKSClusterPolicy](#)」を参照してください。

## AWS 管理ポリシー: AmazonEKSFargatePodExecutionRolePolicy

IAM エンティティに AmazonEKSFargatePodExecutionRolePolicy をアタッチできます。Fargate プロファイルを作成する前に、Fargate Pod 実行ロールを作成し、このポリシーをアタッチする必要があります。詳細については、「[the section called “ステップ 2: Fargate Pod 実行ロールを作成する”](#)」および「[the section called “プロファイルを定義する”](#)」を参照してください。

このポリシーはロールに対して、Fargate で Amazon EKS Pod を実行するために必要な他の AWS サービスリソースにアクセスするための権限を付与します。

### 許可の詳細

このポリシーには、Amazon EKS に以下のタスクを完了させるための以下の権限が含まれています。

- **ecr** – Fargate で実行中のポッドが、Amazon ECR に格納されているコンテナイメージを取得できるようにします。

JSON ポリシードキュメントの最新バージョンを確認するには『AWS管理ポリシーリファレンスガイド』の「[AmazonEKSFargatePodExecutionRolePolicy](#)」を参照してください。

## AWS 管理ポリシー: AmazonEKSForFargateServiceRolePolicy

IAM エンティティに AmazonEKSForFargateServiceRolePolicy をアタッチすることはできません。このポリシーはユーザーに代わって Amazon EKS がアクションを実行

することを許可する、サービスにリンクされた役割にアタッチされます。詳細については「[AWSServiceRoleforAmazonEKSToFargate](#)」を参照してください。

このポリシーは Fargate タスクを実行するために必要なアクセス権限を Amazon EKS に付与します。このポリシーは Fargate ノードがある場合にのみ使用されます。

#### アクセス許可の詳細

このポリシーには、Amazon EKS が以下のタスクを完了できるようにする以下の権限が含まれています。

- **ec2** – エラスティック・ネットワーク・インターフェイス を作成および削除し、エラスティック・ネットワーク・インターフェイス とリソースについて説明します。これは Amazon EKS Fargate サービスが Fargate ポッドに必要な VPC ネットワーキングを設定できるようにするために必要です。

JSON ポリシードキュメントの最新バージョンを確認するには『AWS管理ポリシーリファレンスガイド』の「[AmazonEKSToFargateServiceRolePolicy](#)」を参照してください。

#### AWS マネージドポリシー: AmazonEKSToComputePolicy

IAM エンティティに AmazonEKSToComputePolicy をアタッチできます。このポリシーを [クラスター IAM 役割](#) にアタッチして、EKS がアカウントで管理できるリソースを拡張できます。

このポリシーは Amazon EKS が EKS クラスター用の EC2 インスタンスを作成および管理するために必要な許可と、EC2 を設定するために必要な IAM 許可を付与します。

#### 許可の詳細

このポリシーには、Amazon EKS に以下のタスクを完了させるための以下の権限が含まれています。

- **ec2 許可:**
  - `ec2:CreateFleet` および `ec2:RunInstances` - EC2 インスタンスの作成と、EKS クラスターノードのための特定の EC2 リソース (イメージ、セキュリティグループ、サブネット) の使用を許可します。
  - `ec2:CreateLaunchTemplate` - EKS クラスターノードのための EC2 起動テンプレートの作成を許可します。
  - また、このポリシーにはこれらの EC2 許可の使用を、EKS クラスター名および他の関連タグでタグ付けされたリソースに制限する条件も含まれています。

- `ec2:CreateTags - CreateFleet`、`RunInstances`、および `CreateLaunchTemplate` アクションによって作成された EC2 リソースへのタグの追加を許可します。
- **iam 許可:**
  - `iam:AddRoleToInstanceProfile` - EKS コンピューティングインスタンスプロファイルへの IAM 役割の追加を許可します。
  - `iam:PassRole` - 必要な IAM 役割を EC2 サービスに渡すことを許可します。

JSON ポリシードキュメントの最新バージョンを確認するには「AWS マネージドポリシーリファレンスガイド」の「[AmazonEKSCoordinatePolicy](#)」を参照してください。

## AWS 管理ポリシー: AmazonEKSElasticLoadBalancingPolicy

IAM エンティティに `AmazonEKSElasticLoadBalancingPolicy` をアタッチできます。このポリシーを [クラスター IAM 役割](#) にアタッチして、EKS がアカウントで管理できるリソースを拡張できます。

このポリシーは Amazon EKS が EKS クラスターのネットワークインターフェイスを作成および管理するために必要なアクセス許可を付与し、コントロールプレーンとワーカーノードが適切に通信および機能するように設計されています。

### アクセス許可の詳細

このポリシーは Amazon EKS がクラスターのネットワークインターフェイスを管理できるようにする以下のアクセス許可を付与します。

- **ec2 ネットワークインターフェイスアクセス許可:**
  - `ec2:CreateNetworkInterface` - EC2 ネットワークインターフェイスの作成を許可します。
  - このポリシーにはこのアクセス許可の使用を EKS クラスター名と Kubernetes CNI ノード名でタグ付けされたネットワークインターフェイスに制限するための条件が含まれています。
  - `ec2:CreateTags - CreateNetworkInterface` アクションによって作成されたネットワークインターフェイスへのタグの追加を許可します。
- **ec2 ネットワークインターフェイス管理のアクセス許可:**
  - `ec2:AttachNetworkInterface`、`ec2:DetachNetworkInterface` - EC2 インスタンスへのネットワークインターフェイスのアタッチとデタッチを許可します。
  - `ec2:UnassignPrivateIpAddresses`、`ec2:UnassignIpv6Addresses`、`ec2:AssignPrivateIpAddresses` - ネットワークインターフェイスの IP アドレス割り当ての管理を許可します。



- これらのアクセス許可はEKS クラスター名でタグ付けされたネットワークインターフェイスに制限されます。

JSON ポリシードキュメントの最新バージョンを確認するにはAWS マネージドポリシーリファレンスガイドの「[AmazonEKSNetworkingPolicy](#)」を参照してください。

## AWS マネージドポリシー: AmazonEKSBlockStoragePolicy

IAM エンティティに AmazonEKSBlockStoragePolicy をアタッチできます。このポリシーを[クラスター IAM 役割](#)にアタッチして、EKS がアカウントで管理できるリソースを拡張できます。

このポリシーは Amazon EKS が EKS クラスターのために EC2 ボリュームとスナップショットを作成、管理、メンテナンスするために必要な許可を付与し、Kubernetes ワークロードが必要とする永続的ストレージをコントロールプレーンとワーカーノードがプロビジョニングおよび使用できるようにします。

### アクセス許可の詳細

この IAM ポリシーは Amazon EKS が EC2 ボリュームとスナップショットを管理できるように、次の許可を付与します:

- **ec2 ボリューム管理の許可:**
  - ec2:AttachVolume、ec2:DetachVolume、ec2:ModifyVolume、ec2:EnableFastSnapshotRecovery - EC2 ボリュームの高速スナップショット復元のアタッチ、デタッチ、変更、有効化を許可します。
  - これらの許可はEKS クラスター名でタグ付けされたボリュームに制限されます。
  - ec2:CreateTags - CreateVolume および CreateSnapshot アクションによって作成された EC2 ボリュームとスナップショットへのタグの追加を許可します。
- **ec2 ボリューム作成の許可:**
  - ec2:CreateVolume - 新しい EC2 ボリュームの作成を許可します。
  - このポリシーにはこの許可の使用を、EKS クラスター名および他の関連タグでタグ付けされたボリュームに制限する条件が含まれています。
  - ec2:CreateSnapshot - 新しい EC2 ボリュームスナップショットの作成を許可します。
  - このポリシーにはこの許可の使用を、EKS クラスター名および他の関連タグでタグ付けされたスナップショットに制限する条件が含まれています。



JSON ポリシードキュメントの最新バージョンを確認するには「AWS マネージドポリシーリファレンスガイド」の「[AmazonEKSBlockStoragePolicy](#)」を参照してください。

## AWS マネージドポリシー: AmazonEKSLoadBalancingPolicy

IAM エンティティに AmazonEKSLoadBalancingPolicy をアタッチできます。このポリシーを[クラスター IAM 役割](#)にアタッチして、EKS がアカウントで管理できるリソースを拡張できます。

この IAM ポリシーは Amazon EKS が Elastic ロードバランサー (ELB) および関連リソースを管理するためにさまざまな AWS サービスと連携するために必要な許可を付与します。

### アクセス許可の詳細

このポリシーによって付与される主な許可は次のとおりです:

- **elasticloadbalancing**: Elastic ロードバランサー とターゲットグループの作成、変更、管理を許可します。これにはロードバランサー、ターゲットグループ、リスナー、ルールを作成、更新、削除するための許可が含まれます。
- **ec2**: Kubernetes コントロールプレーンがインスタンスをクラスターに参加させ、Amazon EBS ボリュームを管理するために必要なセキュリティグループの作成と管理を許可します。また、インスタンス、VPC、サブネット、セキュリティグループ、その他のネットワーク リソースなどの EC2 リソースを記述および一覧表示することもできます。
- **iam**: Kubernetes コントロールプレーンが ELB を動的にプロビジョニングするために必要な、Elastic Load Balancing 用のサービスにリンクされた役割の作成を許可します。
- **kms**: etcd に保存されている Kubernetes シークレットの暗号化をサポートするために Kubernetes コントロールプレーンが必要とする AWS KMS からのキーの読み取りを許可します。
- **wafv2** および **shield**: ウェブ ACL 関連付けと関連付け解除、および Elastic ロードバランサーのための AWS Shield 保護の作成/削除を許可します。
- **cognito-idp**、**acm**、および **elasticloadbalancing**: ユーザープールクライアントの記述、証明書の一覧表示および記述、ならびにターゲットグループの記述を実行するための許可を付与します。これは Kubernetes コントロールプレーンが Elastic ロードバランサーを管理するために必要です。

また、このポリシーには `eks:eks-cluster-name` タグを使用して、許可の範囲が管理対象の特定の EKS クラスターに設定されているようにするための条件チェックもいくつか含まれています。

JSON ポリシードキュメントの最新バージョンを確認するには「AWS マネージドポリシーリファレンスガイド」の「[AmazonEKSLoadBalancingPolicy](#)」を参照してください。

## AWS 管理ポリシー: AmazonEKSServicePolicy

IAM エンティティに AmazonEKSServicePolicy をアタッチできます。2020 年 4 月 16 日より前に作成されたクラスターでは IAM 役割を作成し、このポリシーをアタッチする必要があります。2020 年 4 月 16 日以降に作成されたクラスターでは役割を作成する必要はなく、このポリシーの割り当ても必要ありません。IAM プリンシパルを使用してクラスターを作成する場合、iam:CreateServiceLinkedRole 権限がある場合、[AWSServiceRoleforAmazonEKS](#) のサービスにリンクされた役割が自動的に作成されます。このサービスにリンクされた役割には [管理ポリシー: AmazonEKSServiceRolePolicy](#) がアタッチされています。

このポリシーにより、Amazon EKS は Amazon EKS クラスターを操作するために必要なリソースを作成および管理できるようになります。

### アクセス許可の詳細

このポリシーには、Amazon EKS が以下のタスクを完了できるようにする以下の権限が含まれています。

- **eks** – 更新を開始した後、クラスターの Kubernetes バージョンを更新します。この権限は Amazon EKS では使用されませんが、下位互換性のためにポリシーに残ります。
- **ec2** – エラスティック・ネットワーク・インターフェイス およびその他のネットワークリソースとタグを操作します。これは、ノードと Kubernetes コントロールプレーン間の通信を容易にする ネットワーキングを設定するために、Amazon EKS で必要です。セキュリティグループに関する情報を確認します。セキュリティグループのタグを更新します。
- **route53** – VPC をホストゾーンに関連付けます。これは、Kubernetes クラスター API サーバーのプライベートエンドポイントネットワーキングを有効にするために、Amazon EKS で必要です。
- **logs** – ログイベント これは、Amazon EKS が Kubernetes コントロールプレーンログを CloudWatch に送信できるようにするために必要です。
- **iam** - サービスにリンクされた役割を作成します。これは Amazon EKS がユーザーに代わって [the section called “アマゾン EKS でのサービスにリンクされた役割のアクセス許可”](#) のサービスにリンクされた役割を作成するために必要です。

JSON ポリシードキュメントの最新バージョンを確認するには『AWS管理ポリシーリファレンスガイド』の「[AmazonEKSServicePolicy](#)」を参照してください。

## AWS 管理ポリシー: AmazonEKSServiceRolePolicy

IAM エンティティに AmazonEKSServiceRolePolicy をアタッチすることはできません。このポリシーはユーザーに代わって Amazon EKS がアクションを実行することを許可する、サービスにリンクされた役割にアタッチされます。詳細については「[the section called “アマゾン EKS でのサービスにリンクされた役割のアクセス許可”](#)」を参照してください。iam:CreateServiceLinkedRole 権限のある IAM プリンシパルを使用してクラスターを作成する場合、[AWSServiceRoleforAmazonEKS](#) のサービスにリンクされた役割が自動的に作成され、このポリシーがアタッチされます。

このポリシーにより、サービスにリンクされた役割はユーザーに代わって AWS サービスを呼び出します。

### アクセス許可の詳細

このポリシーには、Amazon EKS が以下のタスクを完了できるようにする以下の権限が含まれています。

- **ec2** – エラスティック・ネットワーク・インターフェイス と Amazon EC2 インスタンス、クラスターセキュリティグループ、およびクラスターの作成に必要な VPC を作成、記述します。詳細については「[the section called “セキュリティグループの要件”](#)」を参照してください。セキュリティグループに関する情報を確認します。セキュリティグループのタグを更新します。
- **ec2 Auto Mode** - EKS Auto Mode によって作成された EC2 インスタンスを終了します。詳細については、「[EKS Auto Mode](#)」を参照してください。
- **iam** – IAM 役割にアタッチされているすべての管理ポリシーを一覧表示します。これは、Amazon EKS がクラスターの作成に必要なすべての管理ポリシーと権限を一覧表示および検証できるようにするために必要です。
- **VPC をホストゾーンに関連付ける** – これは、Kubernetes クラスターの API サーバーのプライベートエンドポイントネットワーキングを有効にするために、Amazon EKS で必要です。
- **ログイベント** – これは、Amazon EKS が Kubernetes コントロールプレーンログを CloudWatch に送信できるようにするために必要です。
- **Put メトリクス** – これは Amazon EKS が Kubernetes コントロールプレーンログを CloudWatch に送信できるようにするために必要です。
- **eks** - クラスターアクセスエントリとポリシーを管理し、EKS リソースにアクセスできるユーザーと、それらのユーザーが実行できるアクションをきめ細かく制御できるようにします。これにはコンピューティング、ネットワーキング、ロードバランシング、ストレージオペレーションのために標準アクセスポリシーを関連付けることが含まれます。

- **elasticloadbalancing** - EKS クラスターに関連付けられているロードバランサーとそのコンポーネント (リスナー、ターゲットグループ、証明書) を作成、管理、削除します。ロードバランサーの属性とヘルスステータスを表示します。
- **events** - EKS クラスターに関連する EC2 イベントと AWS ヘルスイベントをモニタリングするための EventBridge ルールを作成および管理し、インフラストラクチャの変更とヘルスアラートへの自動応答を可能にします。
- **iam** - EKS ノードの管理に必要な作成、削除、役割の関連付けなど、「eks」プレフィックスを使用して EC2 インスタンスプロファイルを管理します。
- **pricing** および **shield** - AWS の料金情報と Shield 保護ステータスにアクセスし、EKS リソースのコスト管理と高度なセキュリティ機能を可能にします。
- リソースのクリーンアップ - クラスターのクリーンアップオペレーション中に、ボリューム、スナップショット、起動テンプレート、ネットワークインターフェイスなどの EKS タグ付きリソースを安全に削除します。

JSON ポリシードキュメントの最新バージョンを確認するには『AWS管理ポリシーリファレンスガイド』の「[AmazonEKSServiceRolePolicy](#)」を参照してください。

## AWS 管理ポリシー: AmazonEKSVPCResourceController

AmazonEKSVPCResourceController ポリシーを IAM アイデンティティにアタッチできます。[ポッドのセキュリティグループ](#)を使用している場合はユーザーに代わってアクションを実行させるために、このポリシーを [Amazon EKS クラスターの IAM 役割](#)にアタッチする必要があります。

このポリシーはノードの エラスティック・ネットワーク・インターフェイス と IP アドレスを管理するアクセス権限をクラスター役割に付与します。

### 許可の詳細

このポリシーには、Amazon EKS に以下のタスクを完了させるための以下の権限が含まれています。

- **ec2** – Elastic Network Interface と IP アドレスを管理し、Pod のセキュリティグループと Windows ノードをサポートします。

JSON ポリシードキュメントの最新バージョンを確認するには『AWS管理ポリシーリファレンスガイド』の「[AmazonEKSVPCResourceController](#)」を参照してください。

## AWS 管理ポリシー: AmazonEKSWorkerNodePolicy

AmazonEKSWorkerNodePolicy ポリシーを IAM エンティティにアタッチできます。このポリシーを、Amazon EKS がユーザーに代わってアクションを実行することを許可する Amazon EC2 ノードを作成するときに指定する [ノード IAM 役割](#) にアタッチしなければなりません。eksctl を使用してノードグループを作成すると、ノード IAM 役割が作成され、このポリシーを役割に自動的にアタッチします。

このポリシーは Amazon EKS Amazon EC2 ノードに Amazon EKS クラスターに接続するためのアクセス権限を付与します。

### 許可の詳細

このポリシーには、Amazon EKS に以下のタスクを完了させるための以下の権限が含まれています。

- **ec2** – インスタンスボリュームとネットワーク情報を読み取ります。これは、Kubernetes ノードが Amazon EKS クラスターに参加するのに必要な Amazon EC2 リソースに関する情報を記述できるようにするために必要です。
- **eks** – オプションで、ノードのブートストラップの一部としてクラスターを記述します。
- **eks-auth:AssumeRoleForPodIdentity** – ノード上の EKS ワークロードの認証情報を取得できるようにします。これは EKS Pod Identity が正しく機能するために必要です。

JSON ポリシードキュメントの最新バージョンを確認するには『AWS管理ポリシーリファレンスガイド』の「[AmazonEKSWorkerNodePolicy](#)」を参照してください。

## AWS マネージドポリシー: AmazonEKSWorkerNodeMinimalPolicy

IAM エンティティに AmazonEKSWorkerNodeMinimalPolicy をアタッチできます。このポリシーを、Amazon EKS がユーザーに代わってアクションを実行することを許可する Amazon EC2 ノードを作成するときに指定するノード IAM 役割にアタッチすることができます。

このポリシーは Amazon EKS Amazon EC2 ノードに Amazon EKS クラスターに接続するためのアクセス権限を付与します。このポリシーのアクセス許可は AmazonEKSWorkerNodePolicy に比べて少なくなります。

### 許可の詳細

このポリシーには、Amazon EKS に以下のタスクを完了させるための以下の権限が含まれています。

- `eks-auth:AssumeRoleForPodIdentity` – ノード上の EKS ワークロードの認証情報を取得できるようにします。これは EKS Pod Identity が正しく機能するために必要です。

JSON ポリシードキュメントの最新バージョンを確認するには『AWS管理ポリシーリファレンスガイド』の「[AmazonEKSWorkerNodePolicy](#)」を参照してください。

## AWS 管理ポリシー: AWSServiceRoleForAmazonEKSNodegroup

IAM エンティティに `AWSServiceRoleForAmazonEKSNodegroup` をアタッチすることはできません。このポリシーはユーザーに代わって Amazon EKS がアクションを実行することを許可する、サービスにリンクされた役割にアタッチされます。詳細については「[the section called “アマゾン EKS でのサービスにリンクされた役割のアクセス許可”](#)」を参照してください。

このポリシーは `AWSServiceRoleForAmazonEKSNodegroup` 役割権限を付与し、アカウント内の Amazon EC2 ノードグループを作成および管理できるようにします。

### 許可の詳細

このポリシーには、Amazon EKS に以下のタスクを完了させるための以下の権限が含まれています。

- **ec2** – セキュリティグループ、タグ、キャパシティ予約、および起動テンプレートを操作します。これは Amazon EKS マネージド型ノードグループがリモートアクセス設定を有効にし、マネージド型ノードグループで利用できるキャパシティ予約を記述するために必要です。さらに、Amazon EKS マネージド型ノードグループはユーザーに代わって起動テンプレートを作成します。これは各マネージド型ノードグループをバックアップする Amazon EC2 Auto Scaling グループを設定するためです。
- **iam** – サービスにリンクされた役割を作成し、役割を渡します。これは Amazon EKS マネージド型ノードグループが、マネージド型ノードグループの作成時に渡される役割のインスタンスプロファイルを管理するために必要です。このインスタンスプロファイルはマネージド型ノードグループの一部として起動される Amazon EC2 インスタンスによって使用されます。Amazon EKS は Amazon EC2 Auto Scaling グループなどの他のサービスに対してサービスリンクされた役割を作成する必要があります。これらの権限はマネージド型ノードグループの作成に使用されます。
- **autoscaling** – セキュリティの自動スケーリンググループを使用します。これは Amazon EKS マネージド型ノードグループが、各マネージド型ノードグループをバックアップする Amazon EC2 Auto Scaling グループを管理するために必要です。また、ノードグループの更新中にノードが終了またはリサイクルされたときに Pod を強制撤去させるなどの機能をサポートするためにも使用されます。



JSON ポリシードキュメントの最新バージョンを確認するには「AWS マネージドポリシーリファレンスガイド」の「[AWSServiceRoleForAmazonEKSNodegroup](#)」を参照してください。

## AWS マネージドポリシー: AmazonEKSDashboardServiceRolePolicy

IAM エンティティに AmazonEKSDashboardServiceRolePolicy をアタッチすることはできません。このポリシーはユーザーに代わって Amazon EKS がアクションを実行することを許可する、サービスにリンクされた役割にアタッチされます。詳細については、「[the section called “Amazon EKS でのサービスにリンクされた役割のアクセス許可”](#)」を参照してください。

このポリシーは AWSServiceRoleForAmazonEKSDashboard 役割権限を付与し、アカウント内の Amazon EC2 ノードグループを作成および管理できるようにします。

### アクセス許可の詳細

このポリシーには、以下のタスクを完了させるためにアクセスできる次の権限が含まれています。

- **organizations** – AWS Organizations の構造とアカウントに関する情報を表示します。これには、組織内のアカウントを一覧表示する、組織単位とルートを表示する、委任された管理者を一覧表示する、組織にアクセスできるサービスを表示する、および組織とアカウントに関する詳細情報を取得する権限が含まれます。

JSON ポリシードキュメントの最新バージョンを確認するには『AWS管理ポリシーリファレンスガイド』の「[AmazonEKSDashboardServiceRolePolicy](#)」を参照してください。

## AWS 管理ポリシー: AmazonEBSCSIDriverPolicy

AmazonEBSCSIDriverPolicy ポリシーは Amazon EBS コンテナ・ストレージ・インターフェース (CSI) ドライバーが、ユーザーに代わってボリュームを作成、変更、アタッチ、デタッチ、削除できるようにします。これには既存のボリュームのタグの変更や、EBS ボリュームでの高速スナップショット復元 (FSR) の有効化が含まれます。また、EBS CSI ドライバーに、スナップショットを作成、復元、削除し、インスタンス、ボリューム、スナップショットを一覧表示する権限も付与します。

JSON ポリシードキュメントの最新バージョンを確認するには『AWS管理ポリシーリファレンスガイド』の「[AmazonEBSCSIDriverServiceRolePolicy](#)」を参照してください。

## AWS マネージドポリシー: AmazonEFSCSIDriverPolicy

AmazonEFSCSIDriverPolicy ポリシーでは Amazon EFS コンテナストレージインターフェイス (CSI) がユーザーに代わってアクセスポイントを作成および削除できるようにすることができます。

また、Amazon EFS CSI ドライバーに、アクセスポイント、ファイルシステム、マウントターゲット、Amazon EC2 アベイラビリティゾーンを一覧表示するアクセス許可を付与することもできます。

JSON ポリシードキュメントの最新バージョンを確認するにはAWS マネージドポリシーリファレンスガイドの「[AmazonEFSCSIDriverServiceRolePolicy](#)」を参照してください。

## AWS マネージドポリシー: AmazonEKSLocalOutpostClusterPolicy

このポリシーを IAM エンティティにアタッチできます。ローカルクラスターを作成する前に、このポリシーを[クラスターロール](#)にアタッチする必要があります。Amazon EKS によって管理される Kubernetes クラスターは、ユーザーに代わって他の AWS のサービス呼び出します。これにより、サービスで使用するリソースを管理します。

AmazonEKSLocalOutpostClusterPolicy には以下の権限が含まれています。

- **ec2** 読み取りアクション – コントロールプレーンインスタンスがアベイラビリティーゾーン、ルートテーブル、インスタンス、ネットワークインターフェイスのプロパティを記述できるようにします。Amazon EC2 インスタンスがコントロールプレーンインスタンスとして正常にクラスターに参加するために必要なアクセス許可です。
- **ssm** - Amazon EC2 システム・マネージャー がコントロールプレーンインスタンスに接続できるようにします。これはアカウントのローカルクラスターと通信して管理するために、Amazon EKS によって使用されます。
- **logs** - インスタンスが Amazon CloudWatch にログをプッシュできるようにします。
- **secretsmanager** - インスタンスがコントロールプレーンインスタンスのブートストラップデータを AWS 秘密マネージャー から安全に取得および削除できるようにします。
- **ecr** - コントロールプレーンインスタンス上で動作している Pod とコンテナが、Amazon Elastic Container Registry に格納されているコンテナイメージをプルできるようにします。

JSON ポリシードキュメントの最新バージョンを確認するには『AWS管理ポリシーリファレンスガイド』の「[AmazonEKSLocalOutpostClusterPolicy](#)」を参照してください。

## AWS マネージドポリシー: AmazonEKSLocalOutpostServiceRolePolicy

このポリシーを IAM エンティティにアタッチすることはできません。iam:CreateServiceLinkedRole 権限のある IAM プリンシパルを使用してクラスターを作成する場合、Amazon EKS は [AWSServiceRoleforAmazonEKSLocalOutpost](#) のサービスにリンクされ



た役割を自動的に作成し、このポリシーをアタッチします。このポリシーはサービスにリンクされた役割はローカルクラスターの代わりに AWS サービスを呼び出すことを許可します。

AmazonEKSLocalOutpostServiceRolePolicy には以下の権限が含まれています。

- **ec2** - Amazon EKS がセキュリティ、ネットワーク、その他のリソースと連携して、アカウント内のコントロールプレーンインスタンスを正常に起動および管理できるようにします。
- **ssm** - Amazon EC2 システム・マネージャー がコントロールプレーンインスタンスに接続できるようにします。これはアカウントのローカルクラスターと通信して管理するために、Amazon EKS によって使用されます。
- **iam** - Amazon EKS がコントロールプレーンインスタンスに関連付けられたインスタンスプロフィールを管理できるようにします。
- **secretsmanager** - Amazon EKS がコントロールプレーンインスタンスのブートストラップデータを AWS 秘密マネージャー に格納できるようにします。これにより、インスタンスのブートストラップ中に安全に参照できるようになります。
- **outposts** - Amazon EKS がお客様のアカウントから Outpost 情報を取得して、Outpost でローカルクラスターを正常に起動できるようにします。

JSON ポリシードキュメントの最新バージョンを確認するには『AWS管理ポリシーリファレンスガイド』の「[AmazonEKSLocalOutpostServiceRolePolicy](#)」を参照してください。

### Amazon EKS による AWS 管理ポリシーの更新

Amazon EKS の AWS 管理ポリシーの更新に関する詳細を、このサービスがこれらの変更の追跡を開始した以降の分について表示します。

この特定のドキュメントページへのすべてのソースファイルの変更の通知を受け取るには、RSS リーダーを使用して次の URL をサブスクライブできます。

```
https://github.com/awsdocs/amazon-eks-user-guide/commits/mainline/latest/ug/security/iam-reference/security-iam-awsmanpol.adoc.atom
```

| 変更                                                                    | 説明                               | 日付              |
|-----------------------------------------------------------------------|----------------------------------|-----------------|
| <a href="#">the section called “AWS マネージドポリシー: AmazonEKSDashboard</a> | 新しい AmazonEKS DashboardServiceRo | 2025 年 5 月 21 日 |

| 変更                                                         | 説明                                                                                                                                                                                                  | 日付              |
|------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----------------|
| <a href="#">ServiceRolePolicy</a> ” について説明しました。            | lePolicy ポリシーを導入しました。                                                                                                                                                                               |                 |
| <a href="#">AmazonEKSClusterPolicy</a> に権限を追加しました。         | VPC CNI が予期せず終了した場合に残された Elastic Network Interface を Amazon EKS が削除できるように、ec2:DeleteNetworkInterfaces アクセス許可が追加されました。                                                                               | 2025 年 4 月 16 日 |
| <a href="#">AmazonEKSServiceRolePolicy</a> にアクセス許可を追加しました。 | EKS 1.33 バージョンリリースの一環として、EKS AI/ML の顧客が EFA と互換性のあるセキュリティグループ Egress (アウトバウンド) ルールをデフォルトの EKS クラスター SG に追加できるように、ec2:RevokeSecurityGroupEgress および ec2:AuthorizeSecurityGroupEgress アクセス許可が追加されました。 | 2025 年 4 月 14 日 |
| <a href="#">AmazonEKSServiceRolePolicy</a> に許可を追加しました。     | EKS Auto Mode によって作成された EC2 インスタンスを終了できるようにアクセス許可を追加しました。                                                                                                                                           | 2025 年 2 月 28 日 |

| 変更                                                                                             | 説明                                                                                                                                                                                                                                                                                                                                                                                                                          | 日付               |
|------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------------|
| <a href="#">AmazonEBSCSIDriverPolicy</a> にアクセス許可を追加しました。                                       | <p>EBS CSI ドライバーがすべてのスナップショットを復元することを許可する新しいステートメントを追加しました。これは既存のポリシーでは以前は許可されていましたが、CreateVolume の IAM の処理が変更されたため、新しい明示的なステートメントが必要になりました。</p> <p>EBS CSI ドライバーが既存のボリュームのタグを変更する機能を追加しました。EBS CSI ドライバーは、Kubernetes VolumeAttributesClasses のパラメータを介して既存のボリュームのタグを変更できます。</p> <p>EBS ボリューム上で高速スナップショット復元 (FSR) を有効にするために EBS CSI ドライバーの機能を追加しました。EBS CSI ドライバーは、Kubernetes ストレージクラスのパラメータを介して、新しいボリュームで FSR を有効にできます。</p> | 2025年1月13日       |
| 許可を <a href="#">the section called “AWS マネージドポリシー : AmazonEKSLoadBalancingPolicy”</a> に追加しました。 | AmazonEKSLoadBalancingPolicy を更新して、ネットワークおよび IP アドレス リソースの一覧表示と説明ができるようになりました。                                                                                                                                                                                                                                                                                                                                              | 2024 年 12 月 26 日 |

| 変更                                                                                                      | 説明                                                                                                                                                                     | 日付               |
|---------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------------|
| 許可を <a href="#">the section called “AWS 管理ポリシー: AWSServiceRoleForAmazonEKSNodegroup”</a> に追加しました。       | AWSServiceRoleForAmazonEKSNodegroup が中国リージョンとの互換性のために更新されました。                                                                                                          | 2024 年 11 月 22 日 |
| アクセス許可を <a href="#">the section called “AWS マネージドポリシー: AmazonEKSLocalOutpostClusterPolicy”</a> に追加しました。 | 各ノードが存在するアベイラビリティゾーンを、クラスターコントロールプレーンの AWS クラウド コントローラー マネージャー が識別できるように、AmazonEKSLocalOutpostClusterPolicy に ec2:DescribeAvailabilityZones 許可を追加しました。                 | 2024 年 11 月 21 日 |
| 許可を <a href="#">the section called “AWS 管理ポリシー: AWSServiceRoleForAmazonEKSNodegroup”</a> に追加しました。       | Amazon EKS マネージドノードグループによって作成されたインスタンスのために ec2:RebootInstances を許可するように AWSServiceRoleForAmazonEKSNodegroup ポリシーを更新しました。Amazon EC2 リソースについての ec2:CreateTags 許可を制限しました。 | 2024 年 11 月 20 日 |

| 変更                                                                                          | 説明                                                                                                             | 日付               |
|---------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------|------------------|
| 許可を <a href="#">the section called “AWS 管理ポリシー: AmazonEKSServiceRolePolicy”</a> に追加しました。    | EKS が AWS マネージドポリシー AmazonEKSServiceRolePolicy を更新しました。EKS アクセスポリシー、ロードバランサーの管理、クラスターリソースの自動クリーンアップの許可を追加しました。 | 2024 年 11 月 16 日 |
| <a href="#">the section called “AWS マネージドポリシー: AmazonEKSComputePolicy”</a> について説明しました。      | EKS が AWS マネージドポリシー AmazonEKSComputePolicy を更新しました。iam:AddRoleToInstanceProfile アクションのリソース許可を更新しました。           | 2024 年 11 月 7 日  |
| <a href="#">the section called “AWS マネージドポリシー: AmazonEKSComputePolicy”</a> について説明しました。      | AWS は AmazonEKSComputePolicy を導入しました。                                                                          | 2024 年 11 月 1 日  |
| アクセス許可を AmazonEKSClusterPolicy に追加しました。                                                     | Amazon EKS がトポロジ情報をラベルとしてノードにアタッチできるようにする ec2:DescribeInstanceTopology 許可を追加しました。                              | 2024 年 11 月 1 日  |
| <a href="#">the section called “AWS マネージドポリシー: AmazonEKSBlockStoragePolicy”</a> について説明しました。 | AWS は AmazonEKSBlockStoragePolicy を導入しました。                                                                     | 2024 年 10 月 30 日 |

| 変更                                                                                           | 説明                                                                                      | 日付               |
|----------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------|------------------|
| <a href="#">the section called “AWS マネージドポリシー: AmazonEKSLoadBalancingPolicy”</a> について説明しました。 | AWS は AmazonEKS LoadBalancingPolicy を導入しました。                                            | 2024 年 10 月 30 日 |
| <a href="#">AmazonEKSServiceRolePolicy</a> に許可を追加しました。                                       | Amazon EKS が Amazon CloudWatch にメトリクスを発行できるようにする cloudwatch:PutMetricData 許可を追加しました。    | 2024 年 10 月 29 日 |
| <a href="#">the section called “AWS 管理ポリシー: AmazonEKS NetworkingPolicy”</a> について説明しました。      | AWS は AmazonEKS NetworkingPolicy を導入しました。                                               | 2024 年 10 月 28 日 |
| AmazonEKSServicePolicy および AmazonEKS ServiceRolePolicy にアクセス許可を追加しました。                       | EKS がセキュリティグループ情報を読み取り、関連するタグを更新できるように、ec2:GetSecurityGroupsForVpc および関連するタグの許可を追加しました。 | 2024 年 10 月 10 日 |
| <a href="#">AmazonEKSWorkerNodeMinimalPolicy</a> を導入しました。                                    | AWS は AmazonEKS WorkerNodeMinimalPolicy を導入しました。                                        | 2024 年 10 月 3 日  |

| 変更                                                              | 説明                                                                                                                                                                                                 | 日付              |
|-----------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----------------|
| <a href="#">AWSServiceRoleForAmazonEKSNodegroup</a> に権限を追加しました。 | Amazon EKS が管理する 自動スケーリング グループで Amazon EKS が AZRebalance を一時停止および再開できるようにする autoscaling:ResumeProcesses および autoscaling:SuspendProcesses アクセス許可が追加されました。                                           | 2024 年 8 月 21 日 |
| <a href="#">AWSServiceRoleForAmazonEKSNodegroup</a> に権限を追加しました。 | Amazon EKS がユーザーのアカウントでキャパシティ予約を記述できるようにする ec2:DescribeCapacityReservations アクセス許可を追加しました。CAPACITY_BLOCK ノードグループでスケジュールされたスケーリングの設定を有効にする autoscaling:PutScheduledUpdateGroupAction アクセス許可を追加しました。 | 2024 年 6 月 27 日 |

| 変更                                                      | 説明                                                                                                                                                                                                                                       | 日付               |
|---------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------------|
| <a href="#">AmazonEKS_CNI_Policy</a> – 既存のポリシーへの更新      | Amazon EKS は、Amazon VPC CNI Plugin for Kubernetes で Amazon VPC サブネット内の空き IP アドレスの量を確認できるようにする新しい <code>ec2:DescribeSubnets</code> 許可を追加しました。VPC CNI は各サブネットの空き IP アドレスを使用して、エラスティック・ネットワーク・インターフェイスの作成時に使用する空き IP アドレスが最も多いサブネットを選択できます。 | 2024 年 3 月 4 日   |
| <a href="#">AmazonEKSWorkerNodePolicy</a> – 既存のポリシーへの更新 | Amazon EKS は EKS Pod Identity を許可する新しいアクセス権限を追加しました。Amazon EKS Pod Identity エージェントはノード役割を使用します。                                                                                                                                          | 2023 年 11 月 26 日 |
| <a href="#">AmazonEFSCSIDriverPolicy</a> を導入しました。       | AWS は AmazonEFS CSIDriverPolicy を導入しました。                                                                                                                                                                                                 | 2023 年 7 月 26 日  |
| <a href="#">AmazonEKSClusterPolicy</a> に権限を追加しました。      | ロードバランサーを作成しながら、サブネットの自動検出中に Amazon EKS が AZ の詳細を取得できるようにする <code>ec2:DescribeAvailabilityZones</code> 許可が追加されました。                                                                                                                       | 2023 年 2 月 7 日   |



| 変更                                                                     | 説明                                                                                                                                                                                                                                                      | 日付               |
|------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------------|
| <a href="#">AmazonEBSCSIDriverPolicy</a> のポリシー条件が更新されました。              | StringLike キーフィールドにワイルドカード文字を含む無効なポリシー条件を削除しました。また ec2:DeleteVolume に新しい条件 ec2:ResourceTag/kubernetes.io/create-d-for/pvc/name: "*" も追加され、ツリー内プラグインで作成されたボリュームを EBS CSI ドライバーが削除できるようになりました。                                                            | 2022 年 11 月 17 日 |
| <a href="#">AmazonEKSLocalOutpostServiceRolePolicy</a> にアクセス許可を追加しました。 | 前提条件の検証とマネージドライフサイクルの管理が向上するように、ec2:DescribeVPCAttribute、ec2:GetConsoleOutput、ec2:DescribeSecret を追加しました。また、アウトポストのコントロールプレーン Amazon EC2 インスタンスの配置制御をサポートするため、ec2:DescribePlacementGroups、"arn:aws:ec2:*:*:placement-group/*"、ec2:RunInstances が追加しました。 | 2022 年 10 月 24 日 |

| 変更                                                                                                    | 説明                                                                                                                                                                                                                       | 日付               |
|-------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------------|
| <a href="#">AmazonEKSLocalOutpostClusterPolicy</a> の Amazon Elastic Container Registry のアクセス権限を更新します。 | ecr:GetDownloadUrlForLayer アクションを、全リソースセクションからスコープされたセクションに移動しました。 arn:aws:ecr:*:*:repository/eks/ リソースを追加しました。 arn:aws:ecr:*:*:repository/eks/* リソースを削除しました。 このリソースは追加された arn:aws:ecr:*:*:repository/eks/* リソースでカバーします。 | 2022 年 10 月 20 日 |
| <a href="#">AmazonEKSLocalOutpostClusterPolicy</a> にアクセス許可を追加しました。                                    | クラスターコントロールプレーンインスタンスがいくつかの kubelet 引数を更新できるように、 arn:aws:ecr:*:*:repository/kubelet-config-updater Amazon Elastic Container Registry リポジトリを追加しました。                                                                       | 2022 年 8 月 31 日  |
| <a href="#">AmazonEKSLocalOutpostClusterPolicy</a> を導入しました。                                           | AWS は AmazonEKSLocalOutpostClusterPolicy を導入しました。                                                                                                                                                                        | 2022 年 8 月 24 日  |
| <a href="#">AmazonEKSLocalOutpostServiceRolePolicy</a> を導入しました。                                       | AWS は AmazonEKSLocalOutpostServiceRolePolicy を導入しました。                                                                                                                                                                    | 2022 年 8 月 23 日  |

| 変更                                                              | 説明                                                                                                                                                                                                   | 日付               |
|-----------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------------|
| <a href="#">AmazonEBSCSIDriverPolicy</a> を導入しました。               | AWS は AmazonEBS CSIDriverPolicy を導入しました。                                                                                                                                                             | 2022 年 4 月 4 日   |
| <a href="#">AmazonEKSWorkerNodePolicy</a> に権限を追加しました。           | インスタンスレベルのプロパティを自動検出できる Amazon EKS に最適化された AMI を有効化する <code>ec2:DescribeInstanceTypes</code> を追加しました。                                                                                                | 2022 年 3 月 21 日  |
| <a href="#">AWSServiceRoleForAmazonEKSNodegroup</a> に権限を追加しました。 | Amazon EKS がメトリクスの収集を有効にすることを許可する <code>autoscaling:EnableMetricsCollection</code> アクセス許可を追加しました。                                                                                                    | 2021 年 12 月 13 日 |
| <a href="#">AmazonEKSClusterPolicy</a> に権限を追加しました。              | <code>ec2:DescribeAccountAttributes</code> 、 <code>ec2:DescribeAddresses</code> 、 <code>ec2:DescribeInternetGateways</code> の権限を追加し、Amazon EKS が Network Load Balancer のサービスにリンクされた役割を作成できるようになりました。 | 2021 年 6 月 17 日  |
| Amazon EKS が変更の追跡を開始しました。                                       | Amazon EKS が AWS 管理ポリシーの変更の追跡を開始しました。                                                                                                                                                                | 2021 年 6 月 17 日  |

## IAM のトラブルシューティング

このトピックでは、IAM での Amazon EKS の使用中に表示される一般的なエラーとその回避方法について説明します。

### AccessDeniedException

AWS API オペレーションの呼び出し時に `AccessDeniedException` を受け取った場合、使用している [IAM プリンシパル](#) の認証情報には、その呼び出しに必要な許可がありません。

```
An error occurred (AccessDeniedException) when calling the DescribeCluster operation:
User: arn:aws:iam::111122223333:user/user_name is not authorized to perform:
eks:DescribeCluster on resource: arn:aws:eks:region:111122223333:cluster/my-cluster
```

前記のメッセージの例では、ユーザーには Amazon EKS `DescribeCluster` API オペレーションを呼び出す許可がありません。IAM プリンシパルに Amazon EKS 管理者の許可を付与するには、「[the section called “アイデンティティベースのポリシー”](#)」を参照してください。

IAM の一般的な情報については、IAM ユーザーガイドの「[ポリシーを使用したアクセスの制御](#)」を参照してください。

[Compute] (コンピューティング) タブに [Nodes] (ノード) が表示されず、また [Resources] (リソース) タブにも何も表示されず、AWS Management Console でエラーが表示される

Your current user or role does not have access to Kubernetes objects on this EKS clusterというコンソールのエラーメッセージが表示されることがあります。AWS Management Console を使用している [IAM プリンシパル](#) ユーザーに必要な許可があることを確認してください。詳細については、「[the section called “必要なアクセス許可”](#)」を参照してください。

aws-auth **ConfigMap** がクラスターへのアクセスを許可しない

[AWS IAM Authenticator](#) は、ConfigMap で使用されるロール ARN でパスを許可しません。したがって、rolearn を指定する前に、パスを削除してください。例えば、`arn:aws:iam::111122223333:role/team/developers/eks-admin` を `arn:aws:iam::111122223333:role/eks-admin` に変更します。

## iam:PassRole を実行する権限がありません

iam:PassRole アクションを実行する権限がないというエラーが表示された場合は、ポリシーを更新して Amazon EKS にロールを渡せるようにする必要があります。

一部の AWS サービスでは、新しいサービスロールまたはサービスリンクロールを作成せずに、既存のロールをサービスに渡すことができます。そのためには、サービスにロールを渡す権限が必要です。

次の例のエラーは、marymajor という IAM ユーザーがコンソールを使用して Amazon EKS でアクションを実行しようとする場合に発生します。ただし、このアクションをサービスが実行するには、サービスロールから付与された権限が必要です。メアリーには、ロールをサービスに渡す許可がありません。

```
User: {arn-aws}iam::123456789012:user/marymajor is not authorized to perform:
iam:PassRole
```

この場合、メアリーのポリシーを更新してメアリーに iam:PassRole アクションの実行を許可する必要があります。

サポートが必要な場合は、AWS 管理者に問い合わせてください。サインイン認証情報を提供した担当者が管理者です。

## AWS アカウント以外のユーザーに Amazon EKS リソースへのアクセスを許可したい

他のアカウントのユーザーや組織外のユーザーが、リソースへのアクセスに使用できるロールを作成できます。ロールの引き受けを委託するユーザーを指定できます。リソースベースのポリシーまたはアクセスコントロールリスト (ACL) をサポートするサービスの場合、それらのポリシーを使用して、リソースへのアクセスを付与できます。

詳細については、以下を参照してください。

- Amazon EKS がこれらの機能をサポートしているかどうかについては、「[the section called “IAM および Amazon EKS”](#)」を参照してください。
- 所有している AWS アカウント間でリソースへのアクセスを付与する方法については、IAM ユーザーガイドの「[所有している別の AWS アカウントへのアクセスを IAM ユーザーに許可](#)」を参照してください。
- リソースへのアクセスをサードパーティーの AWS アカウントに提供する方法については、IAM ユーザーガイドの「[サードパーティーが所有する AWS アカウントへのアクセスの提供](#)」を参照してください。

- ID フェデレーションを介してアクセスを提供する方法については、「IAM ユーザーガイド」の「[外部で認証されたユーザー \(ID フェデレーション\) へのアクセスの許可](#)」を参照してください。
- クロスアカウントアクセスにおけるロールとリソースベースのポリシーの使用方法の違いについては、「IAM ユーザーガイド」の「[IAM でのクロスアカウントのリソースへのアクセス](#)」を参照してください。

## ポッドコンテナは次のエラーを受け取ります: **An error occurred (SignatureDoesNotMatch) when calling the GetCallerIdentity operation: Credential should be scoped to a valid region**

アプリケーションが AWS STS グローバルエンドポイント ( <https://sts.amazonaws> ) に明示的にリクエストを行っていて、Kubernetes サービスアカウントがリージョンエンドポイントを使用するように設定されている場合、コンテナはこのエラーを受け取ります。次のいずれかのオプションを使用して問題を解決できます。

- アプリケーションコードを更新して、AWS STS グローバルエンドポイントへの明示的な呼び出しを削除します。
- アプリケーションコードを更新して、 <https://sts.us-west-2.amazonaws.com> などのリージョンエンドポイントへの明示的な呼び出しを行います。AWS リージョン内のサービスに障害が発生した場合に別の AWS リージョンを選択するよう、アプリケーションに冗長性が組み込まれている必要があります。詳細については、IAM ユーザーガイドの「[AWS リージョンでの AWS STS の管理](#)」を参照してください。
- グローバルエンドポイントを使用するようにサービスアカウントを設定します。クラスターでは、デフォルトでリージョンエンドポイントが使用されます。詳細については、「[the section called “STS エンドポイント”](#)」を参照してください。

## Amazon EKS クラスター の IAM ロール

Amazon EKS クラスター IAM ロールはクラスターごとに必要です。Amazon EKS によって管理される Kubernetes クラスターはこのロールを使用してノードを管理し、[レガシークラウドプロバイダー](#)はこのロールを使用して Elastic Load Balancing によるロードバランサーをサービス用に作成します。

Amazon EKS クラスターを使用するには、事前に次の IAM ポリシーのいずれかを持つ IAM ロールを作成する必要があります。

- [AmazonEKSClusterPolicy](#)
- カスタム IAM ポリシー。以下の最小限の許可では、Kubernetes クラスターがノードを管理することはできますが、[レガシークラウドプロバイダー](#)が Elastic Load Balancing によるロードバランサーを作成することはできません。カスタム IAM ポリシーには、少なくとも以下の許可が必要です。

```
{
 "Version": "2012-10-17",
 "Statement": [
 {
 "Effect": "Allow",
 "Action": [
 "ec2:CreateTags"
],
 "Resource": "arn:aws:ec2:*:*:instance/*",
 "Condition": {
 "ForAnyValue:StringLike": {
 "aws:TagKeys": "kubernetes.io/cluster/*"
 }
 }
 },
 {
 "Effect": "Allow",
 "Action": [
 "ec2:DescribeInstances",
 "ec2:DescribeNetworkInterfaces",
 "ec2:DescribeVpcs",
 "ec2:DescribeDhcpOptions",
 "ec2:DescribeAvailabilityZones",
 "ec2:DescribeInstanceTopology",
 "kms:DescribeKey"
],
 "Resource": "*"
 }
]
}
```

#### Note

2023 年 10 月 3 日より前は、各クラスターの IAM ロールには [AmazonEKSClusterPolicy](#) が必要でした。

2020 年 4 月 16 日までは、[AmazonEKSServicePolicy](#) および [AmazonEKSClusterPolicy](#) が必須であり、ロールに推奨される名前は eksServiceRole でした。AWSServiceRoleForAmazonEKS のサービスにリンクされたロールにより、2020 年 4 月 16 日以降に作成されたクラスターに対しては、[AmazonEKSServicePolicy](#) ポリシーは必要なくなりました。

## 既存のクラスターロールの確認

次の手順を使用して、アカウントに アマゾン EKS クラスターロールが既に存在しているかどうかを確認できます。

1. IAM コンソール の <https://console.aws.amazon.com/iam/> を開きます
2. 左のナビゲーションペインで、[ロール] を選択してください。
3. ロールのリストで eksClusterRole を検索します。eksClusterRole が含まれているロールが存在しない場合は、[the section called “アマゾン EKS でのクラスターロールの作成”](#) を参照してロールを作成します。eksClusterRole が含まれているロールが存在する場合はこのロールを選択してアタッチされているポリシーを表示します。
4. [許可] を選択してください。
5. アマゾンEKSクラスターポリシー 管理ポリシーがロールにアタッチされていることを確認します。ポリシーがアタッチされている場合、アマゾン EKS クラスターロールは適切に設定されています。
6. [Trust relationships] (信頼関係) を 選択し、[Edit trust policy] (信頼ポリシーの編集) を選択してください。
7. 信頼関係に以下のポリシーが含まれていることを確認します。信頼関係が以下のポリシーと一致する場合、[キャンセル] を選択してください。信頼関係が一致しない場合、ポリシーを [信頼ポリシーの編集] ウィンドウにコピーし、[ポリシーの更新] を選択します。

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



```
]
}
```

## アマゾン EKS でのクラスターロールの作成

クラスターロールを作成するにはAWS Management Console または AWS CLI を使用できます。

### AWS Management Console

- IAM コンソール の<https://console.aws.amazon.com/iam/>を開きます
- [ロール] を選択してから [ロールの作成] を選びます。
- [信頼できるエンティティタイプ] の下で、[AWS サービス] を選択してください。
- [他の AWS サービスのユースケース] ドロップダウンリストから、[EKS] を選択してください。
- ユースケースに [EKS - Cluster] (EKS - クラスター ) を 選択し、[ 次へ] (次へ ) を 選択してください。
- [Add permissions] (アクセス許可を追加する) タブで、[Next] (次へ) を選択します。
- [ロール名] に、eksClusterRole などのロールの一意の名前を入力します。
- [Description] (説明 ) にはAmazon EKS - Cluster role のような説明文を入力してください。
- [ロールの作成] を選択してください。

### AWS CLI

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

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

- ロールを作成します。*eksClusterRole* は、任意の名前で置き換えることができます。

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

c. 必要な IAM ポリシーをロールにアタッチします。

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

## Amazon EKS ノードの IAM ロール

Amazon EKS ノード kubelet デーモンが、ユーザーに代わって AWS API への呼び出しを実行します。ノードは、IAM インスタンスプロファイルおよび関連ポリシーを通じて、これらの API コールのアクセス許可を受け取ります。ノードを起動してクラスターに登録する前に、起動するとき使用するノード用の IAM ロールを作成する必要があります。この要件は、Amazon が提供する Amazon EKS 最適化 AMI で起動されたノード、または使用する予定の他のノード AMI で起動されたノードに適用されます。さらに、この要件はマネージド型ノードグループとセルフマネージド型ノードの両方に適用されます。

### Note

クラスターの作成に使用したロールは使用できません。

ノードを使用するには、次の アクセス許可を持つ IAM ロールを作成する必要があります。

- kubelet が VPC 内の Amazon EC2 リソースを記述するためのアクセス許可 ([AmazonEKSWorkerNodePolicy](#) ポリシーで規定されているなど)。このポリシーは、Amazon EKS Pod Identity エージェントのアクセス許可も提供します。
- kubelet が Amazon Elastic Container Registry (Amazon ECR) のコンテナイメージを使用するためのアクセス許可 ([AmazonEC2ContainerRegistryPullOnly](#) ポリシーで規定されているなど)。ネットワーク用組み込みアドオンは Amazon ECR のコンテナイメージを使用するポッドを実行するため、Amazon Elastic Container Registry (Amazon ECR) のコンテナイメージを使用する許可が必要です。
- (オプション) Amazon EKS Pod Identity エージェントが eks-auth:AssumeRoleForPodIdentity アクションを使用してポッドの認証情報を取得するための

アクセス許可。[AmazonEKSWorkerNodePolicy](#) を使用しない場合、EKS Pod Identity を使用するには EC2 アクセス許可に加えてこのアクセス許可も提供する必要があります。

- (任意) IRSA または EKS Pod Identity を使用して VPC CNI ポッドにアクセス許可を付与しない場合は、インスタンスロールで VPC CNI のアクセス許可を付与する必要があります。[AmazonEKS\\_CNI\\_Policy](#) マネージドポリシー (IPv4 ファミリーを使用してクラスターを作成した場合) または [作成した IPv6 ポリシー](#) (IPv6 ファミリーを使用してクラスターを作成した場合) のいずれかを使用できます。ただし、このロールにポリシーをアタッチするのではなく、Amazon VPC CNI アドオン専用の別のロールにポリシーをアタッチすることをお勧めします。Amazon VPC CNI アドオン用に別のロールを作成する方法の詳細については、「[the section called “IRSA 用の設定”](#)」を参照してください。

#### Note

Amazon EC2 ノードグループには、Fargate プロファイルとは異なる IAM ロールが必要です。詳細については、「[the section called “ポッド実行IAMロール”](#)」を参照してください。

## 既存のノードロールの確認

以下の手順を使用して、アカウントに既に Amazon EKS ノードロールがあるかどうかを確認できます。

1. IAM コンソール の <https://console.aws.amazon.com/iam/> 開きます
2. 左のナビゲーションペインで、[ロール] を選択してください。
3. ロールのリストで eksNodeRole、AmazonEKSNodeRole、または NodeInstanceRole を検索します。これらの名前のいずれかを持つロールが存在しない場合、「[the section called “Amazon EKS ノードでの IAM ロールの作成”](#)」を参照してロールを作成してください。eksNodeRole、AmazonEKSNodeRole、あるいは NodeInstanceRole を含むロールが存在する場合、そのロールを選択して、アタッチされているポリシーを表示します。
4. [許可] を選択します。
5. AmazonEKSWorkerNodePolicy および AmazonEC2ContainerRegistryPullOnly のマネージドポリシーがロールにアタッチされていること、またはカスタムポリシーが最小限の許可でアタッチされていることを確認します。

 Note

そのロールに [AmazonEKS\_CNI\_Policy] ポリシーがアタッチされている場合は、そのポリシーを一度削除して、aws-node Kubernetes サービスアカウントにマッピングされた IAM ロールにアタッチし直すことをお勧めします。詳細については、「[the section called “IRSA 用の設定”](#)」を参照してください。

6. [信頼関係] を選択し、[信頼ポリシーの編集] を選択します。
7. 信頼関係に以下のポリシーが含まれていることを確認します。信頼関係が以下のポリシーと一致する場合、[キャンセル] を選択してください。信頼関係が一致しない場合、ポリシーを [信頼ポリシーの編集] ウィンドウにコピーし、[ポリシーの更新] を選択してください。

```
{
 "Version": "2012-10-17",
 "Statement": [
 {
 "Effect": "Allow",
 "Action": [
 "sts:AssumeRole"
],
 "Principal": {
 "Service": [
 "ec2.amazonaws.com"
]
 }
 }
]
}
```

## Amazon EKS ノードでの IAM ロールの作成

AWS Management Console または AWS CLI を使用して、ノードの IAM ロールを作成できます。

### AWS Management Console

- a. IAM コンソール の<https://console.aws.amazon.com/iam/> 開きます
- b. 左のナビゲーションペインで、[ロール] を選択してください。
- c. [ロール] ページで、[ロールの作成] を選択してください。

- d. [信頼されたエンティティを選択] ページで、以下の操作を実行してください：
  - i. [信頼するエンティティのタイプ] で [AWS サービス] を選択してください。
  - ii. [ユースケース] で、[EC2] を選択してください。
  - iii. [Next] を選択します。
- e. [アクセス許可を追加] ページで、カスタムポリシーをアタッチするか、以下の操作を行います。
  - i. [フィルタポリシー] ボックスに AmazonEKSWorkerNodePolicy と入力します。
  - ii. 検索結果の [AmazonEKSWorkerNodePolicy] の左にあるチェックボックスを選択します。
  - iii. [フィルターのクリア] を選択します。
  - iv. [フィルタポリシー] ボックスに AmazonEC2ContainerRegistryPullOnly と入力します。
  - v. 検索結果の [AmazonEC2ContainerRegistryPullOnly] の左にあるチェックボックスを選択します。

このロールまたは aws-node Kubernetes サービスアカウントにマッピングされた別のロールに、AmazonEKS\_CNI\_Policy マネージドポリシーまたは作成した [IPv6 ポリシー](#) をアタッチする必要もあります。このロールに対してではなく、Kubernetes サービスアカウントに関連付けられたロールに、前出のポリシーを割り当てることをお勧めします。詳細については、「[the section called “IRSA 用の設定”](#)」を参照してください。

  - vi. [Next] を選択します。
- f. [名前を付けて、レビューし、作成する] ページで、以下の操作を実行します。
  - i. [ロール名] に、AmazonEKSNodeRole などのロールの一意の名前を入力します。
  - ii. [説明] では現在のテキストを「Amazon EKS - Node role」などの説明文に置き換えます。
  - iii. [タグの追加 (オプション)] で、タグをキーバリューのペアとして添付して、メタデータをロールに追加します。IAM でのタグの使用に関する詳細については『IAM ユーザーガイド』の「[IAM リソースにタグを付ける](#)」を参照してください。
  - iv. [ロールの作成] を選択してください。

## AWS CLI

- a. 次のコマンドを実行して、node-role-trust-relationship.json ファイルを作成します。

```
cat >node-role-trust-relationship.json <<EOF
{
```

```

 "Version": "2012-10-17",
 "Statement": [
 {
 "Effect": "Allow",
 "Action": [
 "sts:AssumeRole"
],
 "Principal": {
 "Service": [
 "ec2.amazonaws.com"
]
 }
 }
]
 }
}
EOF

```

- b. IAM ロールを作成します。

```

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

```

- c. IAM ロールに、2 つの必須なマネージド IAM ポリシーをアタッチします。

```

aws iam attach-role-policy \
 --policy-arn arn:aws:iam::aws:policy/AmazonEKSWorkerNodePolicy \
 --role-name AmazonEKSNodeRole
aws iam attach-role-policy \
 --policy-arn arn:aws:iam::aws:policy/AmazonEC2ContainerRegistryPullOnly \
 --role-name AmazonEKSNodeRole

```

- d. クラスターを作成した IP ファミリーに応じて、次の IAM ポリシーの 1 つを IAM ロールにアタッチします。このポリシーは、このロール、または、Kubernetes の aws-node サービスアカウントに関連付けられたロールにアタッチされる必要があります。これらのロールは、Amazon VPC CNI plugin for Kubernetes で使用されます。このポリシーは、Kubernetes サービスアカウントに関連付けられたロールに割り当てることをお勧めします。Kubernetes サービスアカウントに関連付けられたロールへのポリシーの割り当てについては、[「the section called “IRSA 用の設定”」](#)を参照してください。

- IPv4

```

aws iam attach-role-policy \

```

```
--policy-arn arn:aws:iam::aws:policy/AmazonEKS_CNI_Policy \
--role-name AmazonEKSNodeRole
```

- IPv6

A. 次のテキストをコピーし、`vpc-cni-ipv6-policy.json` という名前のファイルに保存します。

```
{
 "Version": "2012-10-17",
 "Statement": [
 {
 "Effect": "Allow",
 "Action": [
 "ec2:AssignIpv6Addresses",
 "ec2:DescribeInstances",
 "ec2:DescribeTags",
 "ec2:DescribeNetworkInterfaces",
 "ec2:DescribeInstanceTypes"
],
 "Resource": "*"
 },
 {
 "Effect": "Allow",
 "Action": [
 "ec2:CreateTags"
],
 "Resource": [
 "arn:aws:ec2:*:*:network-interface/*"
]
 }
]
}
```

B. IAM ポリシーを作成する

```
aws iam create-policy --policy-name AmazonEKS_CNI_IPv6_Policy --policy-
document file://vpc-cni-ipv6-policy.json
```

C. IAM ロールに IAM ポリシーをアタッチします。`111122223333` は、ご自分のアカウント ID に置き換えます。

```
aws iam attach-role-policy \

```

```
--policy-arn arn:aws:iam::111122223333:policy/AmazonEKS_CNI_IPv6_Policy \
--role-name AmazonEKSNodeRole
```

## Amazon EKS 自動モードI クラスター IAM ロール

Amazon EKS クラスター IAM ロールはクラスターごとに必要です。Amazon EKS によって管理される Kubernetes クラスターはこのロールを使用して、ストレージ、ネットワーキング、コンピューティングの自動スケーリングのルーチンタスクを自動化します。

Amazon EKS クラスターを使用するにはEKS 自動モードIに必要なポリシーを持つ IAM ロールを作成する必要があります。推奨の AWS IAM マネージドポリシーをアタッチするか、同等のアクセス許可を持つカスタムポリシーを作成することができます。

- [アマゾンEKSコンピュートポリシー](#)
- [アマゾンEKSブロックストレージポリシー](#)
- [アマゾンEKSロードバランシングポリシー](#)
- [アマゾンEKSネットワーキングポリシー](#)
- [アマゾンEKSクラスターポリシー](#)

### 既存のクラスターロールの確認

次の手順を使用して、アカウントに アマゾン EKS クラスターロールが既に存在しているかどうかを確認できます。

1. IAM コンソール の<https://console.aws.amazon.com/iam/> を開きます
2. 左のナビゲーションペインで、[ロール] を選択してください。
3. ロールのリストで AmazonEKSAutoClusterRole を検索します。AmazonEKSAutoClusterRole を含むロールが存在しない場合は次のセッションの手順を参照してロールを作成します。AmazonEKSAutoClusterRole が含まれているロールが存在する場合はこのロールを選択してアタッチされているポリシーを表示します。
4. [許可] を選択してください。
5. アマゾンEKSクラスターポリシー 管理ポリシーがロールにアタッチされていることを確認します。ポリシーがアタッチされている場合、アマゾン EKS クラスターロールは適切に設定されています。



6. [Trust relationships] (信頼関係) を選択し、[Edit trust policy] (信頼ポリシーの編集) を選択してください。
7. 信頼関係に以下のポリシーが含まれていることを確認します。信頼関係が以下のポリシーと一致する場合、[キャンセル] を選択してください。信頼関係が一致しない場合、ポリシーを [信頼ポリシーの編集] ウィンドウにコピーし、[ポリシーの更新] を選択してください。

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

#### Note

AWS ではこのロールの名前 AmazonEKSAutoClusterRole は必要ありません。

## アマゾン EKS でのクラスターロールの作成

クラスターロールを作成するにはAWS Management Console または AWS CLI を使用できます。

### AWS Management Console

1. IAM コンソール の<https://console.aws.amazon.com/iam/>を開きます
2. [ロール] を選択してから [ロールの作成] を選びます。
3. [信頼できるエンティティタイプ] の下で、[AWS サービス] を選択してください。
4. [他の AWS サービスのユースケース] ドロップダウンリストから、[EKS] を選択してください。

5. ユースケースに [EKS - Cluster] (EKS - クラスター ) を 選択し、[ 次へ] (次へ ) を 選択してください。
6. [アクセス許可を追加] タブで、ポリシーを選択し、[次へ] を選択してください。
  - [アマゾンEKSコンピュートポリシー](#)
  - [アマゾンEKSブロックストレージポリシー](#)
  - [アマゾンEKSロードバランシングポリシー](#)
  - [アマゾンEKSネットワーキングポリシー](#)
  - [アマゾンEKSクラスターポリシー](#)
7. [ロール名] に、AmazonEKSAutoClusterRole などのロールの一意の名前を入力します。
8. [Description] (説明 ) にはAmazon EKS - Cluster role のような説明文を入力してください。
9. [ロールの作成] を選択してください。

## AWS CLI

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

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

2. ロールを作成します。*AmazonEKSAutoClusterRole* は任意の名前に置き換えることができます。

```
aws iam create-role \
 --role-name AmazonEKSAutoClusterRole \
```

```
--assume-role-policy-document file://\"cluster-trust-policy.json\"
```

### 3. 必要な IAM ポリシーをロールにアタッチします :

#### アマゾンEKSクラスターポリシー:

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

#### アマゾンEKSコンピュートポリシー:

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

#### アマゾンEKSブロックストレージポリシー:

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

#### アマゾンEKSロードバランシングポリシー:

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

#### アマゾンEKSネットワーキングポリシー:

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

## アマゾン EKS 自動モードI ノードの IAM ロール

### Note

クラスターの作成に使用したロールは使用できません。

ノードを作成する前に、次のパイプラインまたは同等のアクセス許可を持つ IAM ロールを作成する必要があります：

- [アマゾンEKSワーカーノードミニマルポリシー](#)
- [アマゾンEC2コンテナレジストリプルオンリー](#)

## 既存のノードロールの確認

以下の手順を使用して、アカウントに既に Amazon EKS ノードロールがあるかどうかを確認できます。

1. IAM コンソール の <https://console.aws.amazon.com/iam/> 開きます
2. 左のナビゲーションペインで、[ロール] を選択してください。
3. ロールのリストで AmazonEKSAutoNodeRole を検索します。これらの名前のいずれかを持つロールが存在しない場合は次のセクションの手順を参照してロールを作成します。AmazonEKSAutoNodeRole を含むロールがある場合はロールを選択して、アタッチされたポリシーを表示します。
4. [許可] を選択してください。
5. 上記の必要なポリシーまたは同等のカスタムポリシーがアタッチされていることを確認します。
6. [信頼関係] を選択し、[信頼ポリシーの編集] を選択してください。
7. 信頼関係に以下のポリシーが含まれていることを確認します。信頼関係が以下のポリシーと一致する場合、[キャンセル] を選択してください。信頼関係が一致しない場合、ポリシーを [信頼ポリシーの編集] ウィンドウにコピーし、[ポリシーの更新] を選択してください。

```
{
 "Version": "2012-10-17",
 "Statement": [
 {
 "Effect": "Allow",
 "Principal": {
 "Service": "ec2.amazonaws.com"
 },
 "Action": "sts:AssumeRole"
 }
]
}
```

## Amazon EKS ノードでの IAM ロールの作成

AWS Management Console または AWS CLI を使用して、ノードの IAM ロールを作成できます。

### AWS Management Console

1. IAM コンソール の <https://console.aws.amazon.com/iam/> 開きます
2. 左のナビゲーションペインで、[ロール] を選択してください。
3. [ロール] ページで、[ロールの作成] を選択してください。
4. [信頼されたエンティティを選択] ページで、以下の操作を実行してください：
  - a. [信頼するエンティティのタイプ] で [AWS サービス] を選択してください。
  - b. [ユースケース] で、[EC2] を選択してください。
  - c. [次へ] を選択してください。
5. [アクセス許可を追加] ページで、次のポリシーをアタッチします：
  - [アマゾンEKSワーカーノードミニマルポリシー](#)
  - [アマゾンEC2コンテナレジストリプルオンリー](#)
6. [名前を付けて、レビューし、作成する] ページで、以下の操作を実行してください：
  - a. [ロール名] に、AmazonEKSAutoNodeRole などのロールの一意の名前を入力します。
  - b. [説明] では現在のテキストを「Amazon EKS - Node role」などの説明文に置き換えます。
  - c. [タグの追加 (オプション)] で、タグをキーバリューのペアとして添付して、メタデータをロールに追加します。IAM でのタグの使用に関する詳細については『IAM ユーザーガイド』の [「IAM リソースにタグを付ける」](#) を参照してください。
  - d. [ロールの作成] を選択してください。

### AWS CLI

#### ノード IAM ロールを作成する

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

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

#### ロールの ARN をメモする

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

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

必要なポリシーをアタッチする

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

アマゾンEKSワーカーノードミニマルポリシー をアタッチするには：

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

アマゾンEC2コンテナレジストリプルオンリー をアタッチするには：

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

# クラスターのパフォーマンスをモニタリングし、ログを表示する

Amazon EKS では、利用可能な多くのモニタリングツールまたはログ記録ツールを使用して、データを観測できます。Amazon EKS ログデータを AWS サービスまたはパートナーツールにストリーミングしてデータ分析することができます。Amazon EKS の問題をトラブルシューティングするためのデータを提供する多くのサービスが、AWS Management Console で利用可能です。また、AWS でサポートされているオープンソースソリューションを使用して、[Amazon EKS インフラストラクチャをモニタリング](#)することもできます。

Amazon EKS コンソールの左側のナビゲーションペインで [クラスター] を選択した後、クラスターの名前を選択して [オブザーバビリティ] タブを選択すると、クラスターのヘルスと詳細を表示できます。クラスターにデプロイされている既存の Kubernetes リソースの詳細を表示するには、「[the section called “クラスターリソースにアクセスする”](#)」を参照してください。

モニタリングは、Amazon EKS と AWS ソリューションの信頼性、可用性、パフォーマンスを維持する上で重要なパートです。モニタリングのデータを AWS ソリューションのすべての部分から収集することをお勧めします。これにより、マルチポイント障害が発生した場合は、その障害をより簡単にデバッグできます。Amazon EKS のモニタリングを開始する前に、モニタリング計画が以下の質問に対処していることを確認してください。

- 目標は何ですか。クラスターが劇的に拡大する場合、リアルタイムの通知が必要ですか。
- どのリソースを観測する必要がありますか。
- これらのリソースをどのくらいの頻度で観測する必要がありますか。あなたの会社はリスクに迅速に対応したいですか。
- どのツールを使用する予定ですか。起動の一環として既に AWS Fargate を実行している場合は、組み込み[ログルーター](#)を使用できます。
- 誰がモニタリングタスクを実行する予定ですか。
- 何か問題が発生したときに通知を誰に送信したいですか。

## Amazon EKS でのモニタリングとログ記録

Amazon EKS は、モニタリングとログ記録用の組み込みツールを提供します。サポートされているバージョンでは、オブザーバビリティダッシュボードは、クラスターのパフォーマンスに関する可視性を提供します。これは、問題を迅速に検出、トラブルシューティング、是正するのに役立ちま

す。モニタリング機能に加えて、コントロールプレーンの監査ログに基づくリストも含まれています。Kubernetes コントロールプレーンは、コンソールの外部でのスクレイピングも可能な多数のメトリクスを公開します。

コントロールプレーンのログ記録は、クラスターへのすべての API 呼び出し、クラスターに対してどのユーザーがどのアクションを実行したかをキャプチャする監査情報、およびロールベースの情報を記録します。詳細については、AWS 規範的ガイダンスの「[Amazon EKS でのロギングとモニタリング](#)」を参照してください。

Amazon EKS コントロールプレーンのログ記録により、アカウント内で Amazon EKS コントロールプレーンから CloudWatch Logs に対し、監査および診断ログを直接送れるようになります。これらのログを使用すると、クラスターの保護と実行が容易になります。必要なログタイプを正確に選択することで、CloudWatch 内で各 Amazon EKS クラスターのためのグループに対し、ログストリームの形態でログを送信できます。詳細については、「[the section called “コントロールプレーンのログ”](#)」を参照してください。

#### Note

Amazon CloudWatch で Amazon EKS の認証ログをチェックすると、次のサンプルテキストのようなテキストを含むエントリが表示されます。

```
level=info msg="mapping IAM role" groups="[]"
role="arn:aws:iam::111122223333:role/XXXXXXXXXXXXXXXXXXXX-NodeManagerRole-
XXXXXXXXXX" username="eks:node-manager"
```

このテキストを含むエントリが必要です。username は、マネージド型ノードグループと Fargate の特定のオペレーションを実行する、Amazon EKS の内部サービスロールです。低レベルでカスタマイズ可能なログ記録には、[Kubernetes ログ記録](#)が利用可能です。

Amazon EKS は、AWS CloudTrail と統合されています。このサービスにより、Amazon EKS のユーザー、ロールまたは AWS のサービスによって実行されたアクションを記録します。CloudTrail は、Amazon EKS へのすべての API コールをイベントとしてキャプチャします。キャプチャされた呼び出しには、Amazon EKS コンソールからの呼び出しと、Amazon EKS API オペレーションへのコード呼び出しが含まれます。詳細については、「[the section called “AWS CloudTrail”](#)」を参照してください。

Kubernetes API サーバーは、モニタリングと分析に役立つ多数のメトリクスを公開します。詳細については、「[the section called “Prometheus メトリクス”](#)」を参照してください。



Fluent Bit をカスタムの Amazon CloudWatch Logs に設定するには、「Amazon CloudWatch ユーザーガイド」の「[Fluent Bit の設定](#)」を参照してください。

## Amazon EKS のモニタリングとログ記録のツール

Amazon Web Services では、Amazon EKS のモニタリングに使用できるさまざまなツールを提供しています。一部のツールは自動モニタリングを設定できますが、手動呼び出しが必要なツールもあります。環境および既存のツールセットで許容される範囲で、できるだけモニタリングタスクを自動化することをお勧めします。

次の表に、さまざまなモニタリングツールのオプションを示します。

| エリア                 | ツール                                           | 説明                                                                                                  | セットアップ               |
|---------------------|-----------------------------------------------|-----------------------------------------------------------------------------------------------------|----------------------|
| コントロールプレーン          | <a href="#">オブザーバビリティダッシュボード</a>              | サポートされているバージョンでは、オブザーバビリティダッシュボードは、クラスターのパフォーマンスに関する可視性を提供します。これは、問題を迅速に検出、トラブルシューティング、是正するのに役立ちます。 | <a href="#">設定手順</a> |
| アプリケーション/コントロールプレーン | <a href="#">Prometheus</a>                    | Prometheus は、アプリケーションとコントロールプレーンのメトリクスとアラートをモニタリングするために使用できます。                                      | <a href="#">設定手順</a> |
| アプリケーション            | <a href="#">CloudWatch Container Insights</a> | CloudWatch Container Insights は、コンテナ化されたアプリケーション                                                    | <a href="#">設定手順</a> |

| エリア      | ツール                                                 | 説明                                                                                                           | セットアップ               |
|----------|-----------------------------------------------------|--------------------------------------------------------------------------------------------------------------|----------------------|
|          |                                                     | とマイクロサービスのメトリクスとログを収集、集約、要約します。                                                                              |                      |
| アプリケーション | <a href="#">AWS Distro for OpenTelemetry (ADOT)</a> | ADOT は、関連メトリクス、トレースデータ、メタデータを収集し、AWS モニタリングサービスまたはパートナーに送信できます。これは、CloudWatch Container Insights を介して設定できます。 | <a href="#">設定手順</a> |
| アプリケーション | <a href="#">Amazon DevOps Guru</a>                  | Amazon DevOps Guru は、ノードレベルの運用パフォーマンスと可用性を検出します。                                                             | <a href="#">設定手順</a> |

| エリア      | ツール                               | 説明                                                                                                                                          | セットアップ               |
|----------|-----------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------|----------------------|
| アプリケーション | <a href="#">AWS X-Ray</a>         | AWS X-Ray はアプリケーションに関するトレースデータを受信します。このトレースデータには、着信と送信のリクエスト、およびリクエストに関するメタデータが含まれます。Amazon EKS の場合、実装には OpenTelemetry アドオンが必要です。           | <a href="#">設定手順</a> |
| アプリケーション | <a href="#">Amazon CloudWatch</a> | CloudWatch は、サポートされているバージョンでいくつかの基本的な Amazon EKS メトリクスを無料で提供します。CloudWatch Observability Operator を使用してこの機能を拡張し、メトリクス、ログ、トレースデータの収集を処理できます。 | <a href="#">設定手順</a> |

次の表に、さまざまなログ記録ツールのオプションを示します。

| エリア        | ツール                              | 説明                 | セットアップ               |
|------------|----------------------------------|--------------------|----------------------|
| コントロールプレーン | <a href="#">オブザーバビリティダッシュボード</a> | サポートされているバージョンでは、オ | <a href="#">設定手順</a> |

| エリア        | ツール                                                  | 説明                                                                                             | セットアップ               |
|------------|------------------------------------------------------|------------------------------------------------------------------------------------------------|----------------------|
|            |                                                      | ブザーバビリティダッシュボードは、コントロールプレーンの監査ログに基づくリストを表示します。また、Amazon CloudWatch のコントロールプレーンログへのリンクも含まれています。 |                      |
| アプリケーション   | <a href="#">Amazon CloudWatch Container Insights</a> | Amazon CloudWatch Container Insights は、コンテナ化されたアプリケーションおよびマイクロサービスのメトリクスとログを収集、集約、要約します。       | <a href="#">設定手順</a> |
| コントロールプレーン | <a href="#">Amazon CloudWatch Logs</a>               | 監査ログと診断ログを Amazon EKS コントロールプレーンからアカウントの CloudWatch Logs に直接送信できます。                            | <a href="#">設定手順</a> |
| コントロールプレーン | <a href="#">AWS CloudTrail</a>                       | ユーザー、ロール、またはサービスによる API 呼び出しをログに記録します。                                                         | <a href="#">設定手順</a> |

| エリア                       | ツール                                | 説明                                                                                                                                                | セットアップ               |
|---------------------------|------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------|----------------------|
| AWS Fargate インスタンスの複数のエリア | <a href="#">AWS Fargate ログルーター</a> | AWS Fargate インスタンスの場合、ログルーターはログを AWS サービスまたはパートナーツールにストリーミングします。 <a href="#">AWS for Fluent Bit</a> を使用します。ログは他の AWS サービスまたはパートナーツールにストリーミングできます。 | <a href="#">設定手順</a> |

## オブザーバビリティダッシュボードを使用してクラスターをモニタリングする

Amazon EKS コンソールにはクラスターのパフォーマンスに関する可視性を提供するオブザーバビリティダッシュボードが含まれています。このダッシュボードの情報は、問題を迅速に検出、トラブルシューティング、是正するのに役立ちます。オブザーバビリティダッシュボードの該当するセクションを開くには、[ヘルスとパフォーマンスの概要] で項目を選択してください。この概要は[オブザーバビリティ] タブなど、いくつかの場所に含まれています。

オブザーバビリティダッシュボードは、複数のタブに分かれています。

### 概要

[ヘルスとパフォーマンスの概要] にはさまざまなカテゴリの項目数が一覧表示されます。各数値はそのカテゴリのリストを含むオブザーバビリティダッシュボード内の場所へのハイパーリンクとして機能します。

### クラスターヘルス

クラスターヘルスは注意すべき重要な通知であり、その中には可能な限り早急に対処しなければならない可能性のあるものも含まれています。このリストでは説明と、影響を受けるリソースを確認でき

ます。クラスターヘルスには、[ヘルスの問題] と [設定情報] という 2 つのテーブルが含まれています。[ヘルスの問題] のステータスを更新するには、更新ボタン (  ) を選択してください。[設定情報] は、24 時間に 1 回自動的に更新されます。手動で更新することはできません。

[ヘルスの問題] の詳細については、「[the section called “クラスターの正常性に関するよくある質問 およびエラーコードと解決パス”](#)」を参照してください。[設定情報] の詳細については、「[the section called “クラスターのインサイト”](#)」を参照してください。

## コントロールプレーンのモニタリング

[コントロールプレーンのモニタリング] タブは 3 つのセクションに分かれており、それぞれがクラスターのコントロールプレーンのモニタリングとトラブルシューティングに役立ちます。

### メトリクス

Kubernetes バージョン 1.28 以降のクラスターの場合、[メトリクス] セクションにはさまざまなコントロールプレーンコンポーネントについて収集された複数のメトリクスのグラフが表示されます。

セクションの上部で選択することで、すべてのグラフの X 軸で使用する期間を設定できます。更新ボタン (  ) を使用してデータを更新できます。個別のグラフごとに、縦三点リーダーのボタン (  ) をクリックすると、CloudWatch のオプションを含むメニューが開きます。

これらのメトリクスなどは CloudWatch における AWS/EKS 名前空間の基本的なモニタリングメトリクスとして自動的に使用できます。詳細については「Amazon CloudWatch ユーザーガイド」の「[基本モニタリングと詳細モニタリング](#)」を参照してください。より詳細なメトリクス、ビジュアライゼーション、インサイトを取得するには「Amazon CloudWatch ユーザーガイド」の「[Container Insights](#)」を参照してください。あるいは、Prometheus ベースのモニタリングが必要な場合は「[the section called “Prometheus メトリクス”](#)」を参照してください。

次の表には使用可能なメトリクスが記載されています。

| メトリクス                 | 説明                                                              |
|-----------------------|-----------------------------------------------------------------|
| APIServer リクエスト       | API サーバーに対して実行された 1 分あたりのリクエスト。                                 |
| APIServer 合計リクエスト 4XX | HTTP 4XX レスponseコード (クライアント側のエラー) が発生した、1 分あたりの API サーバリクエストの数。 |

| メトリクス                  | 説明                                                             |
|------------------------|----------------------------------------------------------------|
| APIServer 合計リクエスト 5XX  | HTTP 5XX レスポンスコード (サーバー側のエラー) が発生した、1 分あたりの API サーバーリクエストの数。   |
| APIServer 合計リクエスト 429  | HTTP 429 レスポンスコード (リクエストが多すぎます) が発生した、1 分あたりの API サーバーリクエストの数。 |
| ストレージサイズ               | ストレージデータベース (etcd) のサイズ。                                       |
| スケジューラーの試行             | 「unschedulable」、「error」、「scheduled」の結果による、ポッドのスケジュールの試行回数。     |
| 保留中のポッド                | キュータイプが「active」、「backoff」、「unschedulable」、「gated」の保留中のポッドの数。   |
| API サーバーリクエストのレイテンシー   | API サーバーリクエストのレイテンシー。                                          |
| API サーバーの現在処理中のリクエスト   | API サーバーの現在処理中のリクエスト。                                          |
| ウェブフックリクエスト            | 1 分あたりのウェブフックリクエスト。                                            |
| ウェブフックリクエストの拒否         | 拒否されたウェブフックリクエストの数。                                            |
| ウェブフックリクエストのレイテンシー P99 | 外部サードパーティーのウェブフックリクエストの 99 パーセンタイルレイテンシー。                      |

## CloudWatch Log Insights

[CloudWatch Log Insights] セクションにはコントロールプレーンの監査ログに基づくさまざまなリストが表示されます。この機能を使用するには Amazon EKS コントロールプレーンログをオンにする必要があります。これは [CloudWatch のコントロールプレーンログを表示] セクションから実行できます。

データを収集するのに十分な時間が経過したら、[すべてのクエリを実行] したり、一回に 1 つのリストについて [クエリを実行] を選択したりできます。クエリを実行するたびに、CloudWatch から追加料金が課金します。セクションの上部に表示する結果の期間を選択してください。クエリをより高度に制御したい場合は [CloudWatch で表示] を選択できます。これにより、ニーズに合わせて CloudWatch でクエリを更新できます。

詳細については「Amazon CloudWatch Logs ユーザーガイド」の「[CloudWatch Logs Insights を使用したログデータの分析](#)」を参照してください。

## CloudWatch でコントロールプレーンのログを表示する

[ログ記録を管理] を選択して、使用可能なログタイプを更新します。ログ記録を有効にした後、CloudWatch Logs にログが表示されるまでに数分かかります。十分な時間が経過したら、このセクションのリンクを [表示] を選択して、該当するログに移動します。

詳細については「[the section called “コントロールプレーンのログ”](#)」を参照してください。

## クラスターのインサイト

[インサイトをアップグレード] の表では問題が表示されるほか、是正措置が推奨されます。これにより、新しい Kubernetes バージョンにアップグレードするための検証プロセスが加速されます。Amazon EKS は Kubernetes バージョンのアップグレードに影響を及ぼす可能性のある問題のリストに照らしてクラスターを自動的にスキャンします。[インサイトをアップグレード] の表にはこのクラスターに対して Amazon EKS によって実行されるインサイトチェックと、関連付けられたステータスが一覧表示されます。

Amazon EKS は Kubernetes プロジェクトの変更と、新しいバージョンに関連する Amazon EKS サービスの変更の評価に基づいて実行されるインサイトチェックのリストを維持し、定期的に更新します。Amazon EKS コンソールは各インサイトのステータスを自動的に更新します。これは最終更新時刻の列で確認できます。

詳細については「[the section called “クラスターのインサイト”](#)」を参照してください。

## ノードのヘルスの問題

Amazon EKS ノードモニタリングエージェントはノードログを自動的に読み取り、ヘルスの問題を検出します。自動修復設定に関係なく、必要に応じて調査できるように、すべてのノードのヘルスの問題が報告されます。問題タイプが説明なしで一覧にされている場合はそのポップオーバー要素で説明を読み取ることができます。



ページを更新すると、解決された問題はリストから消えます。自動修復が有効になっている場合、アクションを起こさなくても解決されるヘルスの問題が一時的に表示されます。自動修復でサポートされていない問題の場合はタイプによっては手動アクションが必要になる場合があります。

ノードのヘルスの問題を報告するにはクラスターで Amazon EKS 自動モード を使用するか、ノードモニタリングエージェントのアドオンが必要です。詳細については「[the section called “ノードのヘルス”](#)」を参照してください。

## Prometheus を使用してクラスターのメトリクスをモニタリングする

[Prometheus](#) は、エンドポイントをスクレイピングするモニタリングおよび時系列データベースです。収集したデータをクエリ、集計、保存することができます。また、アラートやアラートの集計にも使用できます。このトピックでは、Prometheus をマネージドオプションまたはオープンソースオプションとして設定する方法について説明します。Amazon EKS コントロールプレーンメトリクスのモニタリングは一般的なユースケースです。

Amazon Managed Service for Prometheus は Prometheus 互換のモニタリングおよびアラートサービスで、コンテナ化されたアプリケーションやインフラストラクチャを大規模に監視することが容易になります。これは、メトリクスの取り込み、ストレージ、クエリ、アラートを自動的にスケールリングするフルマネージド型サービスです。また、AWS セキュリティサービスと統合して、データへの高速かつ安全なアクセスを可能にします。オープンソースの PromQL クエリ言語を使用して、メトリクスをクエリし、それらに関するアラートを作成できます。また、Amazon Managed Service for Prometheus のアラートマネージャーを使用して、重要なアラートのアラートルールを設定することもできます。その後、これらの重要なアラートを Amazon SNS トピックに通知として送信できます。

Amazon EKS で Prometheus を使用するにあたっては、数種類のオプションがあります。

- Amazon EKS クラスターを最初に作成するときに Prometheus メトリクスをオンにできます。また、既存のクラスター用に独自の Prometheus スクレイパーを作成することもできます。このトピックでは、これらのオプションの両方を説明します。
- Helm を使用して Prometheus をデプロイできます。詳細については、「[the section called “Helm を使用してデプロイする”](#)」を参照してください。
- コントロールプレーンの未加工メトリクスを Prometheus 形式で表示できます。詳細については、「[the section called “コントロールプレーン”](#)」を参照してください。

## ステップ 1: Prometheus メトリクスを有効にする

### Important

Amazon Managed Service for Prometheus リソースはクラスターのライフサイクル外にあるため、クラスターとは別途維持する必要があります。クラスターを削除するときは、対象コストを抑えるために、該当するスクレイパーも削除してください。詳細については、Amazon Managed Service for Prometheus ユーザーガイドの「[スクレイパーの検出と作成](#)」を参照してください。

Prometheus は、スクレイピングと呼ばれるプルベースのモデルを通じて、クラスターからメトリクスを検出して収集します。スクレイパーは、クラスターインフラストラクチャーとコンテナ化されたアプリケーションからデータを収集するように設定されています。Prometheus メトリクスを送信するオプションをオンにすると、Amazon Managed Service for Prometheus は完全マネージド型のエージェントレススクレイパーを提供します。

クラスターをまだ作成していない場合は、クラスターを初めて作成するときにこのオプションをオンにしてメトリクスを Prometheus に送信できます。Amazon EKS コンソールでは、このオプションはクラスターの新規作成の [オブザーバビリティの設定] ステップにあります。詳細については、「[the section called “クラスターを作成する”](#)」を参照してください。

既存のクラスターが既にある場合は、独自の Prometheus スクレイパーを作成できます。Amazon EKS コンソールでこれを行うには、クラスターの [オブザーバビリティ] タブに移動し、[スクレイパーの追加] ボタンを選択します。AWS API または AWS CLI でこれを行う場合は、「Amazon Managed Service for Prometheus ユーザーガイド」の「[Create a scraper](#)」を参照してください。

Amazon EKS コンソールでスクレイパーを作成する場合、次のオプションを使用できます。

### スクレイパーエイリアスの名前

(オプション) スクレイパーの一意的エイリアスの入力。

### 送信先

Amazon Managed Service for Prometheus ワークスペースを選択します。ワークスペースは、Prometheus メトリクスの保存とクエリに使用される専用の論理スペースです。このワークスペースでは、そのワークスペースにアクセスできるアカウントの Prometheus メトリクスを表示できます。[新しいワークスペースを作成する] オプションでは、指定した [ワークスペースエイリアス] を使用してユーザーに代わってワークスペースを作成するよう Amazon EKS に指示し

ます。[既存のワークスペースを選択] オプションでは、ドロップダウンリストから既存のワークスペースを選択できます。ワークスペースの詳細については、「Amazon Managed Service for Prometheus ユーザーガイド」の「[ワークスペースの管理](#)」を参照してください。

## サービスアクセス

このセクションでは、Prometheus メトリクスを送信する際に付与するアクセス許可の概要を説明します。

- Amazon Managed Service for Prometheus に、スクレイピングされた Amazon EKS クラスターを記述できるようにします
- Amazon マネージド Prometheus ワークスペースへのリモート書き込みを許可する

AmazonManagedScrapperRole がすでに存在する場合、スクレイパーはそれを使用します。AmazonManagedScrapperRole リンクを選択すると、[アクセス許可の詳細] が表示されます。AmazonManagedScrapperRole がまだ存在しない場合は、[アクセス許可の詳細を表示] リンクを選択すると、Prometheus メトリクスを送信して付与している特定のアクセス許可を確認できます。

## サブネット

必要に応じてスクレイパーが継承するサブネットを変更します。グレー表示するサブネットオプションを追加する必要がある場合は、クラスターの作成でネットワークを指定する手順に戻ってください。

## スクレイパー設定

必要に応じて YAML 形式でスクレイパー設定を変更します。これを行うには、フォームを使用するか、代替の YAML ファイルをアップロードします。詳細については、「Amazon Managed Service for Prometheus ユーザーガイド」の「[スクレイパー設定](#)」を参照してください。

Amazon Managed Service for Prometheus は、AWS マネージドコレクターとしてクラスターと一緒に作成されるエージェントレススクレイパーを指します。AWS マネージドコレクターの詳細については、「Amazon Managed Service for Prometheus ユーザーガイド」の「[AWS マネージドコレクターによるメトリクスの取り込み](#)」を参照してください。

### Important

- AWS CLI または AWS API を使用して Prometheus スクレイパーを作成する場合は、その設定を調整して、スクレイパーにクラスター内アクセス許可を付与する必要があります

す。詳細については、「Amazon Managed Service for Prometheus ユーザーガイド」の「[Amazon EKS クラスターの設定](#)」を参照してください。

- アクセスエントリの代わりに aws-auth ConfigMap を使用する Prometheus スクレイパーが 2024 年 11 月 11 日より前に作成されている場合は、Amazon EKS クラスターコントロールプレーンから追加のメトリクスにアクセスするように更新する必要があります。更新された設定については、「Amazon Managed Service for Prometheus ユーザーガイド」の「[Manually configuring Amazon EKS for scraper access](#)」を参照してください。

## ステップ 2: Prometheus メトリクスを使用する

クラスターで Prometheus メトリクスをオンにした後の使用方法の詳細については、「[Amazon Managed Service for Prometheus ユーザーガイド](#)」を参照してください。

## ステップ 3: Prometheus スクレイパーを管理する

スクレイパーを管理するには、Amazon EKS コンソールで [オブザーバビリティ] タブを選択します。表には、スクレイパー ID、エイリアス、ステータス、作成日などの情報を含む、クラスターのスクレイパーのリストが表示されます。スクレイパーの追加、編集、削除、または現在のスクレイパーに関する詳細情報の表示を行うことができます。

スクレイパーの詳細を表示するには、スクレイパー ID リンクを選択します。例えば、ARN、環境、ワークスペース ID、IAM ロール、設定、ネットワーク情報を表示できます。スクレイパー ID は、[DescribeScraper](#)、[UpdateScraper](#)、[DeleteScraper](#) などの Amazon Managed Service for Prometheus API オペレーションへの入力として使用できます。Prometheus API の使用に関する詳細については、「[Amazon Managed Service for Prometheus API リファレンス](#)」を参照してください。

## Helm を使用して Prometheus をデプロイする

Amazon Managed Service for Prometheus の代わりに、Helm を使用してクラスターに Prometheus をデプロイできます。既に Helm がインストールされている場合は、`helm version` コマンドでバージョンを確認できます。Helm は Kubernetes クラスターのパッケージマネージャーです。Helm の詳細およびインストール方法については、「[the section called “Helm を使用してアプリケーションをデプロイする”](#)」を参照してください。

Amazon EKS クラスター用に Helm を設定した後、次の手順で、それを使用して Prometheus をデプロイできます。

## 1. Prometheus 名前空間を作成します。

```
kubectl create namespace prometheus
```

## 2. prometheus-community グラフリポジトリの追加

```
helm repo add prometheus-community https://prometheus-community.github.io/helm-charts
```

## 3. Prometheus をデプロイします。

```
helm upgrade -i prometheus prometheus-community/prometheus \
 --namespace prometheus \
 --set alertmanager.persistence.storageClass="gp2" \
 --set server.persistentVolume.storageClass="gp2"
```

### Note

このコマンドの実行時に `Error: failed to download "stable/prometheus"` (hint: running `helm repo update` may help) エラーが発生した場合、`helm repo update prometheus-community` を実行し、ステップ 2 のコマンドをもう一度実行してみてください。

`Error: rendered manifests contain a resource that already exists` エラーが発生した場合、`helm uninstall your-release-name -n namespace` を実行し、ステップ 3 のコマンドをもう一度実行してみてください。

## 4. prometheus 名前空間のすべての Pod が READY 状態になっていることを確認します。

```
kubectl get pods -n prometheus
```

出力例は次のとおりです。

| NAME                                           | READY | STATUS  | RESTARTS | AGE |
|------------------------------------------------|-------|---------|----------|-----|
| prometheus-alertmanager-59b4c8c744-r7bgp       | 1/2   | Running | 0        | 48s |
| prometheus-kube-state-metrics-7cfd87cf99-jkz2f | 1/1   | Running | 0        | 48s |
| prometheus-node-exporter-jcjzq                 | 1/1   | Running | 0        | 48s |
| prometheus-node-exporter-jxv2h                 | 1/1   | Running | 0        | 48s |
| prometheus-node-exporter-vbdk5                 | 1/1   | Running | 0        | 48s |
| prometheus-pushgateway-76c444b68c-82tnw        | 1/1   | Running | 0        | 48s |

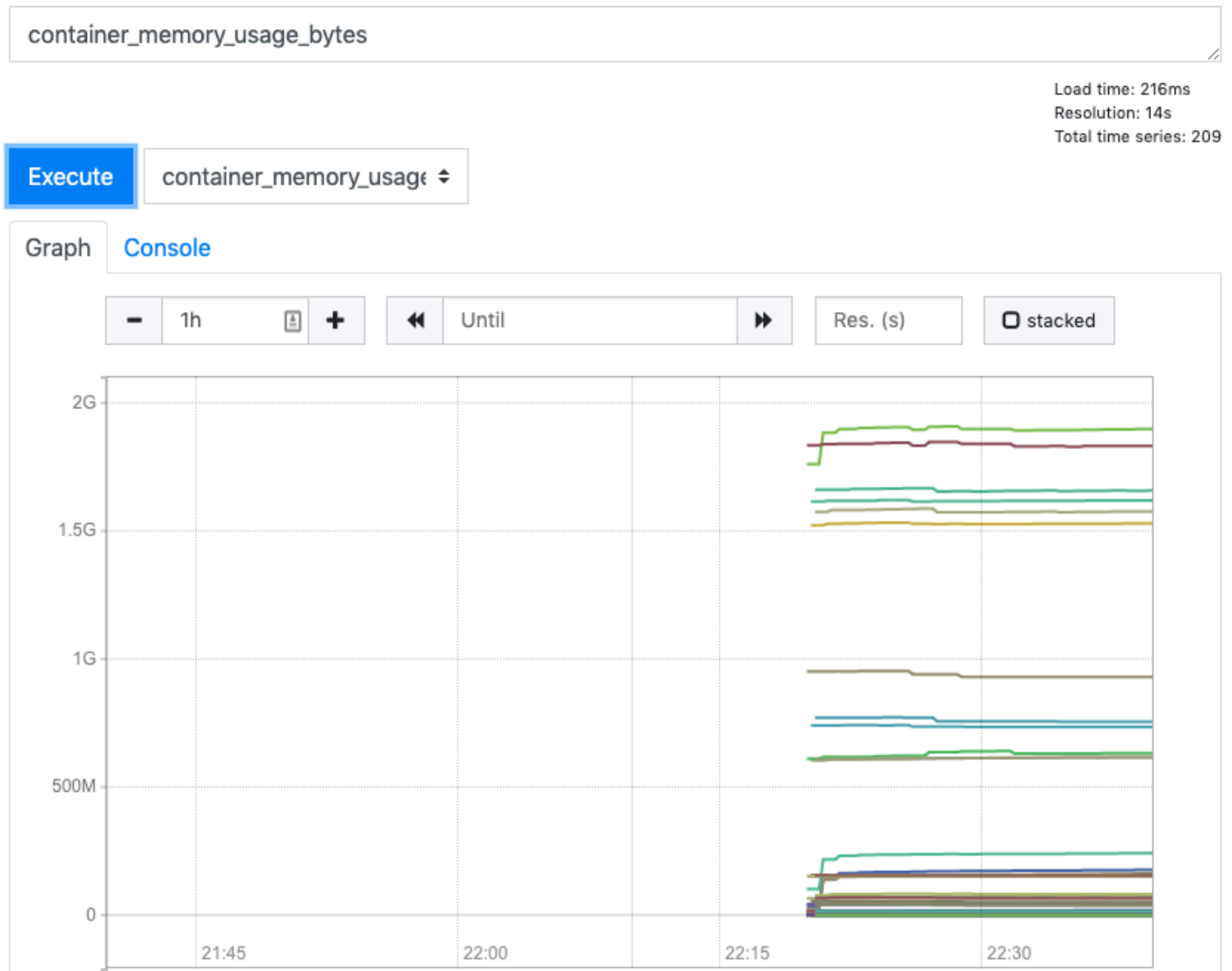
```
prometheus-server-775957f748-mmht9 1/2 Running 0 48s
```

5. `kubect1` を使用して、Prometheus コンソールをローカルマシンにポート転送します。

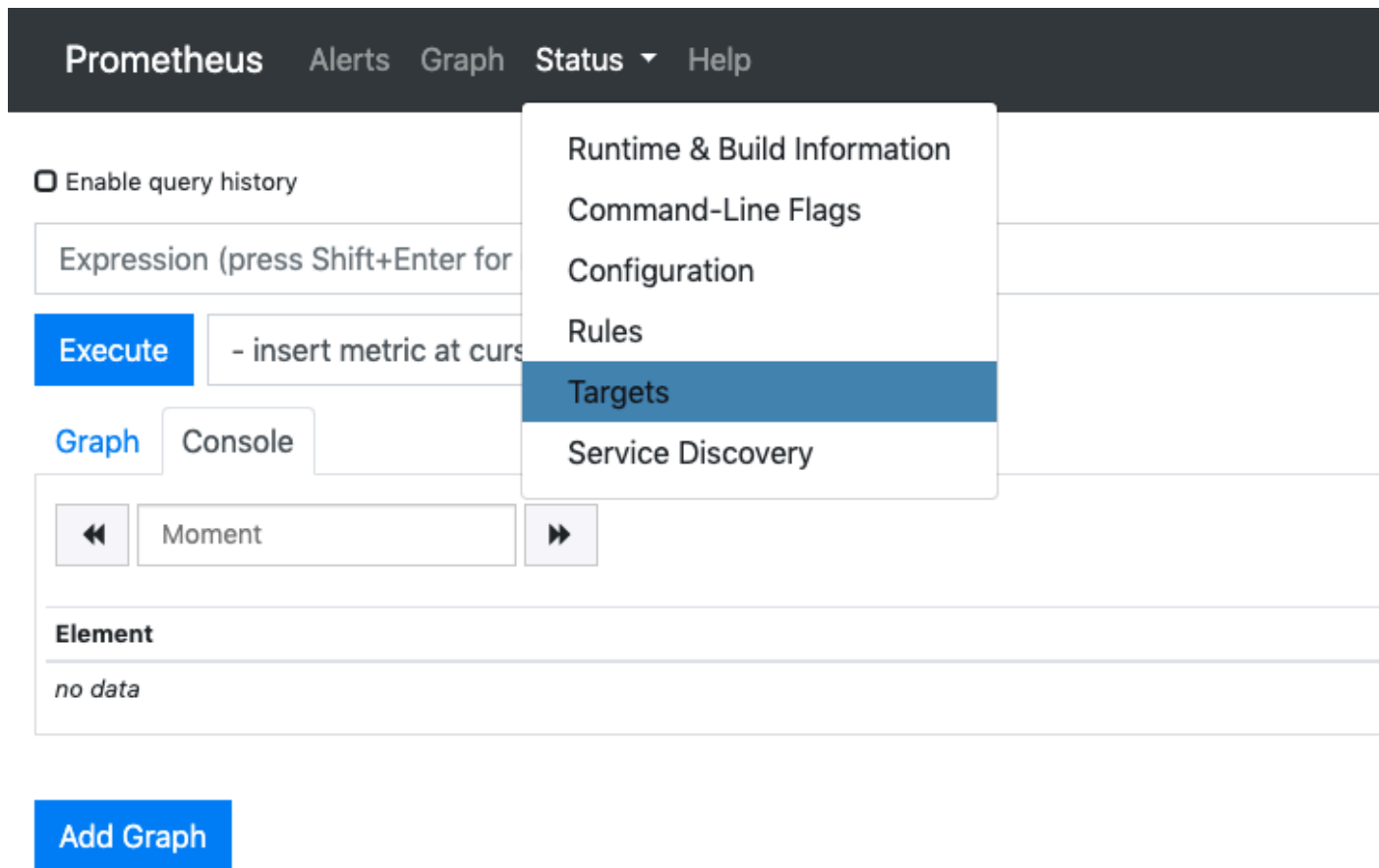
```
kubectl --namespace=prometheus port-forward deploy/prometheus-server 9090
```

6. ウェブブラウザで <http://localhost:9090> にアクセスして、Prometheus コンソールを表示します。

7. [- カーソルの位置のメトリクスを挿入] メニューからメトリクスを選択し、[実行] を選択します。[グラフ] タブを選択して、一定期間のメトリクスを表示します。次の図では、一定期間の `container_memory_usage_bytes` を示します。



8. 上部のナビゲーションバーから、[ステータス]、[ターゲット] の順に選択します。



サービス検出を使用して Prometheus に接続されているすべての Kubernetes エンドポイントが表示されます。

## コントロールプレーンの未加工メトリクスを Prometheus 形式で取得する

Kubernetes コントロールプレーンは、[Prometheus 形式](#)で表される多数のメトリクスを公開します。これらのメトリクスはモニタリングおよび分析に役立ちます。これらはメトリクスエンドポイントを通じて内部的に公開され、Prometheus を完全にデプロイしなくてもアクセスできます。ただし、Prometheus をデプロイすることで、時間の経過に伴うメトリクスの分析がより簡単になります。

未加工メトリクス出力を表示するには、`endpoint`を置き換えて次のコマンドを実行してください。

```
kubectl get --raw endpoint
```

このコマンドを使用すると、任意のエンドポイントパスを渡して未加工のレスポンスを返すことができます。出力にはさまざまなメトリクスが行ごとにリストされ、各行にはメトリクス名、タグ、および値が含まれます。

```
metric_name{tag="value"[,...]} value
```

## API サーバーからメトリクスを取得する

一般的な API サーバーエンドポイントは Amazon EKS コントロールプレーンで公開されます。このエンドポイントは主に特定のメトリクスを調べる際に役立ちます。

```
kubectl get --raw /metrics
```

出力例は次のとおりです。

```
[...]
HELP rest_client_requests_total Number of HTTP requests, partitioned by status code,
method, and host.
TYPE rest_client_requests_total counter
rest_client_requests_total{code="200",host="127.0.0.1:21362",method="POST"} 4994
rest_client_requests_total{code="200",host="127.0.0.1:443",method="DELETE"} 1
rest_client_requests_total{code="200",host="127.0.0.1:443",method="GET"} 1.326086e+06
rest_client_requests_total{code="200",host="127.0.0.1:443",method="PUT"} 862173
rest_client_requests_total{code="404",host="127.0.0.1:443",method="GET"} 2
rest_client_requests_total{code="409",host="127.0.0.1:443",method="POST"} 3
rest_client_requests_total{code="409",host="127.0.0.1:443",method="PUT"} 8
HELP ssh_tunnel_open_count Counter of ssh tunnel total open attempts
TYPE ssh_tunnel_open_count counter
ssh_tunnel_open_count 0
HELP ssh_tunnel_open_fail_count Counter of ssh tunnel failed open attempts
TYPE ssh_tunnel_open_fail_count counter
ssh_tunnel_open_fail_count 0
```

この未加工出力は API サーバーが公開する内容をそのまま返します。

## **metrics.eks.amazonaws.com** を使用してコントロールプレーンのメトリクスを取得する

Kubernetes バージョン 1.28 以降のクラスターの場合、Amazon EKS は API グループ **metrics.eks.amazonaws.com** のメトリクスも公開します。これらのメトリクスには kube-



scheduler や kube-controller-manager などのコントロールプレーンコンポーネントが含まれます。

#### Note

クラスターに新しい APIService リソース `v1.metrics.eks.amazonaws.com` の作成をブロックするウェブフック設定がある場合、メトリクスエンドポイント機能が使用できない可能性があります。`v1.metrics.eks.amazonaws.com` キーワードを検索することで、`kube-apiserver` 監査ログで確認することができます。

### **kube-scheduler** メトリクスを取得する

kube-scheduler メトリクスを取得するには次のコマンドを使用します。

```
kubectl get --raw "/apis/metrics.eks.amazonaws.com/v1/ksh/container/metrics"
```

出力例は次のとおりです。

```
TYPE scheduler_pending_pods gauge
scheduler_pending_pods{queue="active"} 0
scheduler_pending_pods{queue="backoff"} 0
scheduler_pending_pods{queue="gated"} 0
scheduler_pending_pods{queue="unschedulable"} 18
HELP scheduler_pod_scheduling_attempts [STABLE] Number of attempts to successfully
 schedule a pod.
TYPE scheduler_pod_scheduling_attempts histogram
scheduler_pod_scheduling_attempts_bucket{le="1"} 79
scheduler_pod_scheduling_attempts_bucket{le="2"} 79
scheduler_pod_scheduling_attempts_bucket{le="4"} 79
scheduler_pod_scheduling_attempts_bucket{le="8"} 79
scheduler_pod_scheduling_attempts_bucket{le="16"} 79
scheduler_pod_scheduling_attempts_bucket{le="+Inf"} 81
[...]
```

### **kube-controller-manager** メトリクスを取得する

kube-controller-manager メトリクスを取得するには次のコマンドを使用します。

```
kubectl get --raw "/apis/metrics.eks.amazonaws.com/v1/kcm/container/metrics"
```

出力例は次のとおりです。

```
[...]
workqueue_work_duration_seconds_sum{name="pvprotection"} 0
workqueue_work_duration_seconds_count{name="pvprotection"} 0
workqueue_work_duration_seconds_bucket{name="replicaset",le="1e-08"} 0
workqueue_work_duration_seconds_bucket{name="replicaset",le="1e-07"} 0
workqueue_work_duration_seconds_bucket{name="replicaset",le="1e-06"} 0
workqueue_work_duration_seconds_bucket{name="replicaset",le="9.999999999999999e-06"} 0
workqueue_work_duration_seconds_bucket{name="replicaset",le="9.999999999999999e-05"} 19
workqueue_work_duration_seconds_bucket{name="replicaset",le="0.001"} 109
workqueue_work_duration_seconds_bucket{name="replicaset",le="0.01"} 139
workqueue_work_duration_seconds_bucket{name="replicaset",le="0.1"} 181
workqueue_work_duration_seconds_bucket{name="replicaset",le="1"} 191
workqueue_work_duration_seconds_bucket{name="replicaset",le="10"} 191
workqueue_work_duration_seconds_bucket{name="replicaset",le="+Inf"} 191
workqueue_work_duration_seconds_sum{name="replicaset"} 4.265655885000002
[...]
```

## スケジューラーとコントローラーマネージャーのメトリクスを理解します

Prometheus スタイルスクレイピングに使用できるスケジューラとコントローラーマネージャーのメトリクスを次の表に示します。これらのメトリクスの詳細については、Kubernetes ドキュメントの「[Kubernetes Metrics Reference](#)」を参照してください。

| メトリクス                                            | コントロールプレーンのコンポーネント | 説明                                                               |
|--------------------------------------------------|--------------------|------------------------------------------------------------------|
| <code>scheduler_pending_pods</code>              | スケジューラ             | 実行のためにノードにスケジュールされるのを待っているポッドの数です。                               |
| <code>scheduler_schedule_attempts_total</code>   | スケジューラ             | ポッドのスケジュールの試行回数です。                                               |
| <code>scheduler_preemption_attempts_total</code> | スケジューラ             | 優先度の低いポッドをエビクションさせて、優先度の高いポッドをスケジュールするためにスケジューラーによって実行された試行回数です。 |

| メトリクス                                                              | コントロールプレーンのコンポーネント | 説明                                                                                      |
|--------------------------------------------------------------------|--------------------|-----------------------------------------------------------------------------------------|
| <code>scheduler_preemption_victims</code>                          | スケジューラ             | 優先順位の高いポッドのスペースを確保するためにエビクション対象として選択されたポッドの数です。                                         |
| <code>scheduler_pod_scheduling_attempts</code>                     | スケジューラ             | ポッドの正常なスケジュールの試行回数です。                                                                   |
| <code>scheduler_scheduling_attempt_duration_seconds</code>         | スケジューラ             | リソースの可用性やスケジューリングルールなどのさまざまな要因に基づいて、スケジューラがポッドを実行するのに適した場所をどの程度迅速に見つけることができるかを示します。     |
| <code>scheduler_pod_scheduling_slack_duration_seconds</code>       | スケジューラ             | ポッドがスケジューリングキューに入る時点からの、スケジュールされているポッドのエンドツーエンドのレイテンシーです。これには複数のスケジューリング試行が含まれる場合があります。 |
| <code>cronjob_controller_job_creation_skew_duration_seconds</code> | コントローラーマネージャー      | <code>cronjob</code> の実行がスケジュールされてから、対応するジョブが作成されるまでの時間です。                              |
| <code>workqueue_depth</code>                                       | コントローラーマネージャー      | 現在のキューの深さです。                                                                            |
| <code>workqueue_adds_total</code>                                  | コントローラーマネージャー      | ワークキューが処理する合計追加数。                                                                       |

| メトリクス                            | コントロールプレーンのコンポーネント | 説明                                |
|----------------------------------|--------------------|-----------------------------------|
| workqueue_queue_duration_seconds | コントローラーマネージャー      | 項目がリクエストされる前にワークキューに留まる時間 (秒) です。 |
| workqueue_work_duration_seconds  | コントローラーマネージャー      | ワークキューからの項目の処理にかかる時間 (秒) です。      |

## プロメテウス スクレイパーをデプロイしてメトリクスを一貫してスクレイピングする

プロメテウス スクレイパーをデプロイしてメトリクスを一貫してスクレイピングするには次の設定を使用します:

```

apiVersion: v1
kind: ConfigMap
metadata:
 name: prometheus-conf
data:
 prometheus.yml: |-
 global:
 scrape_interval: 30s
 scrape_configs:
 # apiserver metrics
 - job_name: apiserver-metrics
 kubernetes_sd_configs:
 - role: endpoints
 scheme: https
 tls_config:
 ca_file: /var/run/secrets/kubernetes.io/serviceaccount/ca.crt
 insecure_skip_verify: true
 bearer_token_file: /var/run/secrets/kubernetes.io/serviceaccount/token
 relabel_configs:
 - source_labels:
 [
 __meta_kubernetes_namespace,
 __meta_kubernetes_service_name,
 __meta_kubernetes_endpoint_port_name,
]
```

```

 action: keep
 regex: default;kubernetes;https
Scheduler metrics
- job_name: 'ksh-metrics'
 kubernetes_sd_configs:
 - role: endpoints
 metrics_path: /apis/metrics.eks.amazonaws.com/v1/ksh/container/metrics
 scheme: https
 tls_config:
 ca_file: /var/run/secrets/kubernetes.io/serviceaccount/ca.crt
 insecure_skip_verify: true
 bearer_token_file: /var/run/secrets/kubernetes.io/serviceaccount/token
 relabel_configs:
 - source_labels:
 [
 __meta_kubernetes_namespace,
 __meta_kubernetes_service_name,
 __meta_kubernetes_endpoint_port_name,
]
 action: keep
 regex: default;kubernetes;https
Controller Manager metrics
- job_name: 'kcm-metrics'
 kubernetes_sd_configs:
 - role: endpoints
 metrics_path: /apis/metrics.eks.amazonaws.com/v1/kcm/container/metrics
 scheme: https
 tls_config:
 ca_file: /var/run/secrets/kubernetes.io/serviceaccount/ca.crt
 insecure_skip_verify: true
 bearer_token_file: /var/run/secrets/kubernetes.io/serviceaccount/token
 relabel_configs:
 - source_labels:
 [
 __meta_kubernetes_namespace,
 __meta_kubernetes_service_name,
 __meta_kubernetes_endpoint_port_name,
]
 action: keep
 regex: default;kubernetes;https

apiVersion: v1
kind: Pod
metadata:
```

```
name: prom-pod
spec:
 containers:
 - name: prom-container
 image: prom/prometheus
 ports:
 - containerPort: 9090
 volumeMounts:
 - name: config-volume
 mountPath: /etc/prometheus/
 volumes:
 - name: config-volume
 configMap:
 name: prometheus-conf
```

ポッドが新しいメトリクスエンドポイントにアクセスするには次の許可が必要です。

```
{
 "effect": "allow",
 "apiGroups": [
 "metrics.eks.amazonaws.com"
],
 "resources": [
 "kcm/metrics",
 "ksh/metrics"
],
 "verbs": [
 "get"
] },
```

使用中のロールにパッチを適用するには次のコマンドを使用できます。

```
kubectl patch clusterrole <role-name> --type=json -p='[
 {
 "op": "add",
 "path": "/rules/-",
 "value": {
 "verbs": ["get"],
 "apiGroups": ["metrics.eks.amazonaws.com"],
 "resources": ["kcm/metrics", "ksh/metrics"]
 }
 }
]
```

```
]'
```

その後、Prometheus スクレイパーのポートをローカルポートにプロキシすることで、Prometheus ダッシュボードを表示できます。

```
kubect1 port-forward pods/prom-pod 9090:9090
```

Amazon EKS クラスターの場合、コア Kubernetes コントロールプレーンメトリクスも AWS/EKS 名前空間の Amazon CloudWatch メトリクスに取り込まれます。これらを表示するには [CloudWatch コンソール](#)を開き、左側のナビゲーションペインから [すべてのメトリクス] を選択してください。[メトリクス] の選択ページで、クラスターの AWS/EKS 名前空間とメトリクスディメンションを選択してください。

## Amazon CloudWatch でクラスターデータをモニタリングする

Amazon CloudWatch はクラウドリソースからメトリクスとログを収集するモニタリングサービスです。バージョン 1.28 以降の新しいクラスターを使用する場合、CloudWatch は基本的な Amazon EKS メトリクスを無料で提供します。ただし、CloudWatch オペサビリティオペレーターを Amazon EKS アドオンとして使用すると、オブザーバビリティ機能を強化できます。

### Amazon CloudWatch の基本メトリクス

Kubernetes バージョン 1.28 以降のクラスターの場合、CloudWatch の Vended メトリクスを AWS/EKS 名前空間で無料で取得できます。次の表はサポートされているバージョンで利用できる基本的なメトリクスのリストです。リストされているあらゆるメトリクスの頻度は 1 分です。

| メトリクス名                                         | 説明                                                                                                                                                                     |
|------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>scheduler_schedule_attempts_total</code> | <p>一定期間にクラスター内の Pod をスケジュールするための、スケジューラによる合計試行回数。このメトリクスは、スケジューラのワークロードのモニタリングに役立ち、スケジューリングのプレッシャーや Pod 配置の潜在的な問題を示している可能性があります。</p> <p>単位: カウント</p> <p>有効な統計: Sum</p> |

| メトリクス名                                                 | 説明                                                                                                             |
|--------------------------------------------------------|----------------------------------------------------------------------------------------------------------------|
| <code>scheduler_schedule_attempts_SCHEDULED</code>     | <p>一定期間にクラスター内のノードに Pod をスケジュールするための、スケジューラによる成功した試行回数。</p> <p>単位: カウント</p> <p>有効な統計: Sum</p>                  |
| <code>scheduler_schedule_attempts_UNSCHEDULABLE</code> | <p>ノードの CPU やメモリの不足などの有効な制約により、一定期間スケジュールできなかった Pod をスケジュールするための試行回数。</p> <p>単位: カウント</p> <p>有効な統計: Sum</p>    |
| <code>scheduler_schedule_attempts_ERROR</code>         | <p>API サーバーの接続の問題などの、スケジューラ自体の内部的な問題により、一定期間失敗していた Pod をスケジュールするための試行回数。</p> <p>単位: カウント</p> <p>有効な統計: Sum</p> |
| <code>scheduler_pending_pods</code>                    | <p>一定期間にクラスター内のスケジューラによってスケジュールされる、保留中の Pod の合計数。</p> <p>単位: カウント</p> <p>有効な統計: Sum</p>                        |



| メトリクス名                                   | 説明                                                                                                     |
|------------------------------------------|--------------------------------------------------------------------------------------------------------|
| scheduler_pending_pods_ACTIVEQ           | <p>一定期間にクラスターでスケジュールされることを待機している、activeQ で保留中の Pod の数。</p> <p>単位: カウント</p> <p>有効な統計: Sum</p>           |
| scheduler_pending_pods_UNSC<br>HEDULABLE | <p>スケジューラがスケジュールを試みて失敗し、再試行のためにスケジュールできない状態で保持されている、保留中の Pod の数。</p> <p>単位: カウント</p> <p>有効な統計: Sum</p> |
| scheduler_pending_pods_BACKOFF           | <p>バックオフ期間の終了を待機している、バックオフ状態で backoffQ で保留中の Pod の数。</p> <p>単位: カウント</p> <p>有効な統計: Sum</p>             |
| scheduler_pending_pods_GATED             | <p>必要な条件を満たすまでスケジュールできないため、制限あり状態で現在待機している保留中の Pod の数。</p> <p>単位: カウント</p> <p>有効な統計: Sum</p>           |
| apiserver_request_total                  | <p>クラスター内のすべての API サーバーで行われた HTTP リクエストの数。</p> <p>単位: カウント</p> <p>有効な統計: Sum</p>                       |

| メトリクス名                            | 説明                                                                                                                                 |
|-----------------------------------|------------------------------------------------------------------------------------------------------------------------------------|
| apiserver_request_total_4XX       | <p>4XX (クライアントエラー) ステータスコードが発生した、クラスター内のすべての API サーバーに対して行われた HTTP リクエストの数。</p> <p>単位: カウント</p> <p>有効な統計: Sum</p>                  |
| apiserver_request_total_429       | <p>クライアントがレート制限のしきい値を超えたときに発生する 429 ステータスコードが発生した、クラスター内のすべての API サーバーに対して行われた HTTP リクエストの数。</p> <p>単位: カウント</p> <p>有効な統計: Sum</p> |
| apiserver_request_total_5XX       | <p>5XX (サーバーエラー) ステータスコードが発生した、クラスター内のすべての API サーバーに対して行われた HTTP リクエストの数。</p> <p>単位: カウント</p> <p>有効な統計: Sum</p>                    |
| apiserver_request_total_LIST_PODS | <p>クラスター内のすべての API サーバーに対して行われた Pod の LIST リクエストの数。</p> <p>単位: カウント</p> <p>有効な統計: Sum</p>                                          |

| メトリクス名                                       | 説明                                                                                                                                                 |
|----------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------|
| apiserver_request_duration_seconds_PUT_P99   | <p>クラスター内のすべての API サーバーでのすべてのリクエストから計算された、PUT リクエストのレイテンシーの 99 パーセンタイル。すべての PUT リクエストの 99% が完了する応答時間を表しています。</p> <p>単位: 秒</p> <p>有効な統計: 平均</p>     |
| apiserver_request_duration_seconds_PATCH_P99 | <p>クラスター内のすべての API サーバーでのすべてのリクエストから計算された、PATCH リクエストのレイテンシーの 99 パーセンタイル。すべての PATCH リクエストの 99% が完了する応答時間を表しています。</p> <p>単位: 秒</p> <p>有効な統計: 平均</p> |
| apiserver_request_duration_seconds_POST_P99  | <p>クラスター内のすべての API サーバーでのすべてのリクエストから計算された、POST リクエストのレイテンシーの 99 パーセンタイル。すべての POST リクエストの 99% が完了する応答時間を表しています。</p> <p>単位: 秒</p> <p>有効な統計: 平均</p>   |

| メトリクス名                                        | 説明                                                                                                                                                   |
|-----------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------|
| apiserver_request_duration_seconds_GET_P99    | <p>クラスター内のすべての API サーバーでのすべてのリクエストから計算された、GET リクエストのレイテンシーの 99 パーセンタイル。すべての GET リクエストの 99% が完了する応答時間を表しています。</p> <p>単位: 秒</p> <p>有効な統計: 平均</p>       |
| apiserver_request_duration_seconds_LIST_P99   | <p>クラスター内のすべての API サーバーでのすべてのリクエストから計算された、LIST リクエストのレイテンシーの 99 パーセンタイル。すべての LIST リクエストの 99% が完了する応答時間を表しています。</p> <p>単位: 秒</p> <p>有効な統計: 平均</p>     |
| apiserver_request_duration_seconds_DELETE_P99 | <p>クラスター内のすべての API サーバーでのすべてのリクエストから計算された、DELETE リクエストのレイテンシーの 99 パーセンタイル。すべての DELETE リクエストの 99% が完了する応答時間を表しています。</p> <p>単位: 秒</p> <p>有効な統計: 平均</p> |

| メトリクス名                                          | 説明                                                                                                                                                   |
|-------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------|
| apiserver_current_inflight_requests_MUTATING    | <p>クラスター内のすべての API サーバーで現在処理されている、変更リクエスト (POST、PUT、DELETE、PATCH) の数。このメトリクスは、処理中であり、まだ処理を完了していないリクエストを表しています。</p> <p>単位: カウント</p> <p>有効な統計: Sum</p> |
| apiserver_current_inflight_requests_READONLY    | <p>クラスター内のすべての API サーバーで現在処理されている、読み取り専用リクエスト (GET、LIST) の数。このメトリクスは、処理中であり、まだ処理を完了していないリクエストを表しています。</p> <p>単位: カウント</p> <p>有効な統計: Sum</p>          |
| apiserver_admission_webhook_request_total       | <p>クラスター内のすべての API サーバーで行われたアドミSSIONウェブフックリクエストの数。</p> <p>単位: カウント</p> <p>有効な統計: Sum</p>                                                             |
| apiserver_admission_webhook_request_total ADMIT | <p>クラスター内のすべての API サーバーで行われた変更アドミSSIONウェブフックリクエストの数。</p> <p>単位: カウント</p> <p>有効な統計: Sum</p>                                                           |

| メトリクス名                                                 | 説明                                                                                       |
|--------------------------------------------------------|------------------------------------------------------------------------------------------|
| apiserver_admission_webhook_request_total_VALIDATING   | クラスター内のすべての API サーバーで行われた検証アドミSSIONウェブフックリクエストの数。<br><br>単位: カウント<br><br>有効な統計: Sum      |
| apiserver_admission_webhook_rejection_count            | 拒否されたクラスター内のすべての API サーバーで行われたアドミSSIONウェブフックリクエストの数。<br><br>単位: カウント<br><br>有効な統計: Sum   |
| apiserver_admission_webhook_rejection_count_ADMIT      | 拒否されたクラスター内のすべての API サーバーで行われた変更アドミSSIONウェブフックリクエストの数。<br><br>単位: カウント<br><br>有効な統計: Sum |
| apiserver_admission_webhook_rejection_count_VALIDATING | 拒否されたクラスター内のすべての API サーバーで行われた検証アドミSSIONウェブフックリクエストの数。<br><br>単位: カウント<br><br>有効な統計: Sum |

| メトリクス名                                                                             | 説明                                                                                                                                                                                  |
|------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>apiserver_admission_webhook_admission_duration_seconds</code>                | <p>クラスター内のすべての API サーバーでのすべてのリクエストから計算された、サードパーティのアドミッションウェブフックリクエストのレイテンシーの 99 パーセンタイル。すべてのサードパーティのアドミッションウェブフックリクエストの 99% が完了する応答時間を表しています。</p> <p>単位: 秒</p> <p>有効な統計: 平均</p>     |
| <code>apiserver_admission_webhook_admission_duration_seconds_ADMIT_P99</code>      | <p>クラスター内のすべての API サーバーでのすべてのリクエストから計算された、サードパーティの変更アドミッションウェブフックリクエストのレイテンシーの 99 パーセンタイル。すべてのサードパーティの変更アドミッションウェブフックリクエストの 99% が完了する応答時間を表しています。</p> <p>単位: 秒</p> <p>有効な統計: 平均</p> |
| <code>apiserver_admission_webhook_admission_duration_seconds_VALIDATING_P99</code> | <p>クラスター内のすべての API サーバーでのすべてのリクエストから計算された、サードパーティの検証アドミッションウェブフックリクエストのレイテンシーの 99 パーセンタイル。すべてのサードパーティの検証アドミッションウェブフックリクエストの 99% が完了する応答時間を表しています。</p> <p>単位: 秒</p> <p>有効な統計: 平均</p> |

| メトリクス名                       | 説明                                                                                                                                       |
|------------------------------|------------------------------------------------------------------------------------------------------------------------------------------|
| apiserver_storage_size_bytes | <p>クラスター内の API サーバーで使用される、etcd ストレージデータベースファイルの物理サイズ (バイト単位)。このメトリクスは、ストレージに割り当てられた実際のディスク容量を表しています。</p> <p>単位: バイト</p> <p>有効な統計: 最大</p> |

## Amazon CloudWatch オブザーバビリティオペレータ

Amazon CloudWatch Observability は、リアルタイムログ、メトリクス、トレースデータを収集します。それらを [Amazon CloudWatch](#) と [AWS X-Ray](#) に送信します。このアドオンをインストールすると、CloudWatch アプリケーションシグナルと Amazon EKS のオブザーバビリティが強化された CloudWatch コンテナインサイトの両方を有効にできます。これにより、インフラストラクチャとコンテナ化されたアプリケーションのヘルスとパフォーマンスをモニタリングできます。Amazon CloudWatch オペサビリティオペレーター は必要なコンポーネントをインストールして設定するように設計されています。

Amazon EKS は [Amazon EKS アドオン](#) として CloudWatch オペサビリティオペレーター をサポートしています。このアドオンは、クラスターの Linux と Windows の両方のワーカーノードで Container Insights を有効にします。Container Insights を Windows で有効にするには、Amazon EKS アドオンのバージョンが 1.5.0 以上である必要があります。現在、CloudWatch Application Signals は Amazon EKS Windows ではサポートされていません。

以下のトピックでは Amazon EKS クラスターのために CloudWatch オペサビリティオペレーター の使用を開始する方法について説明します。

- このアドオンのインストール手順については「Amazon CloudWatch ユーザーガイド」の「[Amazon CloudWatch オペサビリティ EKS アドオンまたは Helm チャートを使用して CloudWatch エージェントをインストールする](#)」を参照してください。
- CloudWatch アプリケーションシグナルの詳細については「Amazon CloudWatch ユーザーガイド」の「[Application Signals](#)」を参照してください。



- 詳細については、Amazon CloudWatch ユーザーガイドの「[Container Insights の使用](#)」を参照してください。

## コントロールプレーンログを CloudWatch Logs に送信する

Amazon EKS コントロールプレーンのログ記録により、アカウント内で Amazon EKS コントロールプレーンから CloudWatch Logs に対し、監査および診断ログを直接送れるようになります。これらのログを使用すると、クラスターの保護と実行が容易になります。必要なログタイプを正確に選択することで、CloudWatch 内で各 Amazon EKS クラスターのためのグループに対し、ログストリームの形態でログを送信できます。CloudWatch サブスクリプションフィルタを使用して、ログをリアルタイムで分析したり、他のサービスに転送したりできます (ログは Base64 でエンコードされ、gzip 形式で圧縮されます)。詳細については、「[Amazon CloudWatch logging](#)」(Amazon CloudWatch ログ記録) を参照してください。

Amazon EKS コントロールプレーンのログ記録の使用を開始するには、新規または既存の Amazon EKS クラスターごとに有効にするログタイプを選択します。クラスターごとに各ログタイプを有効または無効にするには、AWS Management Console、AWS CLI (バージョン 1.16.139 以降)、または Amazon EKS API を使用します。有効化したログタイプのログが、Amazon EKS クラスターから同じアカウントの CloudWatch Logs に自動的に送信されるようになります。

Amazon EKS コントロールプレーンのログ記録を使用すると、実行しているクラスターごとに Amazon EKS の標準料金が発生します。クラスターから CloudWatch Logs に送信されるすべてのログに対して、CloudWatch Logs のデータ取り込みおよび保存に関する標準料金が発生します。また、Amazon EC2 インスタンスや Amazon EBS ボリュームなど、クラスターの一部としてプロビジョニングしている AWS リソースに対しても料金が発生します。

以下のクラスターコントロールプレーンのログタイプが使用可能です。各ログタイプは、Kubernetes コントロールプレーンのコンポーネントに対応しています。これらのコンポーネントの詳細については、Kubernetes ドキュメントの [Kubernetes Components \(Kubernetes コンポーネント\)](#) を参照してください。

### API サーバー `api` ()

クラスターの API サーバーは、Kubernetes API を公開するコントロールプレーンコンポーネントです。クラスターを起動する際、またはその直後に API サーバーのログを有効にすると、ログには、API サーバーの起動に使用された API サーバーフラグが含まれます。詳細については、Kubernetes ドキュメントの「[kube-apiserver](#)」と [監査ポリシー](#) を参照してください。

## 監査[audit](#) ()

Kubernetes 監査ログは、クラスターに影響を与えた個々のユーザー、管理者、またはシステムコンポーネントの記録を提供します。詳細については、Kubernetes ドキュメントの「[監査](#)」を参照してください。

## Authenticator[authenticator](#) ()

Authenticator ログは、Amazon EKS に固有です。これらのログは、Amazon EKS が使用するコントロールプレーンコンポーネントに対応します。各コンポーネントは、Amazon EKS が、Kubernetes の (IAM 認証情報を使用する) [ロールベースのアクセスコントロール](#) (RBAC) 認証のために使用しています。詳細については、「[クラスターの管理](#)」を参照してください。

## コントローラーマネージャー[controllerManager](#) ()

コントローラーマネージャーは、Kubernetes に付属するコアコントロールループを管理します。詳細については、Kubernetes ドキュメントの [kube-controller-manager](#) を参照してください。

## スケジューラ[scheduler](#) ()

スケジューラコンポーネントは、クラスター内で Pod を実行するタイミングと場所を管理します。詳細については、Kubernetes ドキュメントの [kube-scheduler](#) を参照してください。

# コントロールプレーンログを有効または無効にする

デフォルトでは、クラスターコントロールプレーンのログは CloudWatch Logs に送信されません。クラスターのログを送信するには、各ログタイプを個別に有効にする必要があります。CloudWatch Logs で設定された取り込み、アーカイブストレージ、およびデータスキャンレートが、有効化されたコントロールプレーンログに適用されます。詳細については、「[CloudWatch 料金表](#)」を参照してください。

コントロールプレーンのログ記録設定を更新するために、Amazon EKS は各サブネット内で最大 5 つの使用可能な IP アドレスを必要とします。ログタイプを有効にすると、ログの詳細レベル 2 でログが送信されます。

[AWS Management Console](#) または [AWS CLI](#) のいずれかを使用して、コントロールプレーンログを有効または無効にできます。

## AWS Management Console

1. [Amazon EKS コンソール](#)を開きます。
2. クラスター名を選択すると、そのクラスターの情報を表示されます。

3. [オプザバビリティ] タブを選択します。
4. [コントロールプレーンのロギング] セクションで、[ロギングの管理] を選択します。
5. ログタイプごとに、そのログタイプを [有効] にするか [無効] にするかを選択します。デフォルトでは、それぞれのログタイプは無効化されています。
6. [変更を保存] を選択して終了します。

## AWS CLI

1. 以下のコマンドを使用して、AWS CLI のバージョンを確認します。

```
aws --version
```

AWS CLI のバージョンが 1.16.139 より前である場合は、まず最新バージョンに更新する必要があります。AWS CLI をインストールまたはアップグレードするには、「AWS コマンドラインインターフェイスユーザーガイド」の「[AWS コマンドラインインターフェイスのインストール](#)」を参照してください。

2. 次の AWS CLI コマンドを使用して、クラスターのコントロールプレーンログのエクスポート設定を更新します。*my-cluster* をクラスター名に置き換え、目的のエンドポイントアクセス値を指定します。

### Note

次のコマンドは、使用可能なすべてのログタイプを CloudWatch Logs に送信します。

```
aws eks update-cluster-config \
 --region region-code \
 --name my-cluster \
 --logging '{"clusterLogging":[{"types":
["api","audit","authenticator","controllerManager","scheduler"],"enabled":true}]}'
```

出力例は次のとおりです。

```
{
 "update": {
 "id": "883405c8-65c6-4758-8cee-2a7c1340a6d9",
 "status": "InProgress",
```

```

 "type": "LoggingUpdate",
 "params": [
 {
 "type": "ClusterLogging",
 "value": "{\"clusterLogging\": [{\"types\": [\"api\", \"audit\",
\\\"authenticator\\\", \"controllerManager\\\", \"scheduler\\\"], \"enabled\": true}]}\"
 }
],
 "createdAt": 1553271814.684,
 "errors": []
 }
}

```

3. 次のコマンドでログ設定更新のステータスをモニタリングします。その際、以前のコマンドで返されたクラスター名と更新 ID を使用します。ステータスが `Successful` と表示されたら、更新は完了です。

```

aws eks describe-update \
 --region region-code \
 --name my-cluster \
 --update-id 883405c8-65c6-4758-8cee-2a7c1340a6d9

```

出力例は次のとおりです。

```

{
 "update": {
 "id": "883405c8-65c6-4758-8cee-2a7c1340a6d9",
 "status": "Successful",
 "type": "LoggingUpdate",
 "params": [
 {
 "type": "ClusterLogging",
 "value": "{\"clusterLogging\": [{\"types\": [\"api\", \"audit\",
\\\"authenticator\\\", \"controllerManager\\\", \"scheduler\\\"], \"enabled\": true}]}\"
 }
],
 "createdAt": 1553271814.684,
 "errors": []
 }
}

```

## クラスターのコントロールプレーンログを表示する

Amazon EKS クラスターのいずれかのコントロールプレーンログタイプを有効にすると、それらを CloudWatch コンソールで表示できるようになります。

CloudWatch でログを表示、分析、および管理する方法については、[Amazon CloudWatch Logs ユーザーガイド](#)を参照してください。

1. [CloudWatch コンソール](#)を開きます。このリンクでは、現在使用可能なロググループが表示され、プレフィックス `/aws/eks` でフィルタリングされます。
2. ログを表示するクラスターを選択します。ロググループの名前の形式は `/aws/eks/my-cluster/cluster` です。
3. 表示するログストリームを選択します。次のリストで、各ログタイプのログストリーム名の形式について説明します。

### Note

ログストリームデータが大きくなるにつれて、ログストリーム名のローテーションが行われます。特定のログタイプのログストリームが複数存在する場合、最新のログストリームを表示するには、[最終のイベント時刻] が最新のログストリーム名を見つけ出します。

- Kubernetes API サーバーコンポーネントログ (**api**) – kube-apiserver-*1234567890abcdef01234567890abcde*
  - 監査 (**audit**) – kube-apiserver-audit-*1234567890abcdef01234567890abcde*
  - 認証システム (**authenticator**) – authenticator-*1234567890abcdef01234567890abcde*
  - コントローラマネージャー (**controllerManager**) – kube-controller-manager-*1234567890abcdef01234567890abcde*
  - スケジューラ (**scheduler**) – kube-scheduler-*1234567890abcdef01234567890abcde*
4. ログストリームのイベントを確認してください。

たとえば、kube-apiserver-*1234567890abcdef01234567890abcde* のトップを表示すると、クラスターの初期 API サーバーが表示されます。

 Note

ログストリームの先頭に API サーバーログが表示されない場合は、サーバーで API サーバーログ記録を有効にする前に API サーバーのログファイルがローテーションされた可能性があります。API サーバーのログ記録が有効になる前にローテーションされたログファイルは、CloudWatch にエクスポートできません。

ただし、同じ Kubernetes バージョンで新しいクラスターを作成し、クラスターの作成時に API サーバーのログ記録を有効にすることはできます。同じプラットフォームバージョンを持つクラスターでは、同じフラグが有効になるため、フラグは新しいクラスターのフラグと一致する必要があります。CloudWatch で新しいクラスターのフラグを確認し終えたら、新しいクラスターを削除します。

## API コールを AWS CloudTrail イベントとしてログします。

Amazon EKS は AWS CloudTrail と統合します。CloudTrail は、Amazon EKS のユーザー、ロール、または AWS サービスによるアクションを記録するサービスです。CloudTrail は、Amazon EKS へのすべての API コールをイベントとしてキャプチャします。これには、Amazon EKS コンソールからの呼び出しと、Amazon EKS API オペレーションへのコード呼び出しが含まれます。

証跡を作成する場合は、 のイベントなど、Amazon S3 バケットへの CloudTrail イベントの継続的な配信を有効にすることができます。これには、Amazon EKS のイベントが含まれます。追跡を設定しない場合でも、CloudTrail コンソールのイベント履歴で最新のイベントを表示できます。CloudTrail で収集された情報に基づいて、リクエストに関する詳細を確認できます。例えば、Amazon EKS に対するリクエストの日時や、リクエスト元の IP アドレス、リクエストの実行者を確認できます。

CloudTrail に関する詳細は、[AWS CloudTrail ユーザーガイド](#)を参照してください。

### トピック

- [AWS CloudTrail に役立つリファレンスを表示する](#)
- [AWS CloudTrail ログファイルエントリを分析する](#)
- [Amazon EC2 Auto Scaling グループのメトリクスを表示する](#)

## AWS CloudTrail に役立つリファレンスを表示する

AWS アカウントを作成すると、AWS アカウントで CloudTrail も有効になります。Amazon EKS でイベントアクティビティが発生すると、そのアクティビティは [Event history] (イベント履歴) で他の AWS サービスのイベントとともに CloudTrail イベントに記録されます。最近のイベントは、AWS アカウントで表示、検索、ダウンロードできます。詳細については、[CloudTrail イベント履歴でのイベントの表示](#)を参照してください。

Amazon EKS のイベントなど、AWS アカウントのイベントの継続的な記録については、証跡を作成します。証跡により、ログファイルを CloudTrail で Amazon S3 バケットに配信できます。デフォルトでは、コンソールで証跡を作成すると、すべての AWS リージョンに証跡が適用されます。追跡では、AWS パーティション内のすべての AWS リージョンからのイベントをログに記録し、指定した Simple Storage Service (Amazon S3) バケットにログファイルを配信します。さらに、CloudTrail ログで収集したイベントデータをより詳細に分析し、それに基づいて対応するようにその他の AWS サービスを設定できます。詳細については、以下のリソースを参照してください。

- [証跡作成の概要](#)
- 「[CloudTrail がサポートされているサービスと統合](#)」
- 「[CloudTrail の Amazon SNS 通知の設定](#)」
- 「[複数のリージョンから CloudTrail ログファイルを受け取る](#)」および「[複数のアカウントから CloudTrail ログファイルを受け取る](#)」

すべての Amazon EKS アクションは、CloudTrail により記録され、[Amazon EKS API リファレンス](#)で文書化されます。例えば、[CreateCluster](#)、[ListClusters](#) および [DeleteCluster](#) の各セクションを呼び出すと、CloudTrail ログファイルにエントリが生成されます。

すべてのイベントまたはログエントリには、リクエストを行った IAM アイデンティティのタイプとどの認証情報が使用されたかに関する詳細が含まれます。一時的認証情報が使用された場合、エントリは、認証情報がどのように取得されたかを示します。

詳細については、「[CloudTrail userIdentity エlement](#)」を参照してください。

## AWS CloudTrail ログファイルエントリを分析する

「トレイル」は、指定した Amazon S3 バケットにイベントをログファイルとして配信するように設定できます。CloudTrail のログファイルには、ログエントリが 1 つ以上あります。イベントでは、あらゆるソースからの単一のリクエストが示され、リクエストされたアクションに関する情報が含まれています。これには、アクションの日時、使用されたリクエストパラメータなどの情報が含まれます。

す。CloudTrail ログファイルは、公開 API コールの順序付けられたスタックトレースではないため、特定の順序では表示されません。

以下の例は、[CreateCluster](#) アクションを示す CloudTrail ログエントリです。

```
{
 "eventVersion": "1.05",
 "userIdentity": {
 "type": "IAMUser",
 "principalId": "AKIAIOSFODNN7EXAMPLE",
 "arn": "arn:aws:iam::111122223333:user/username",
 "accountId": "111122223333",
 "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
 "userName": "username"
 },
 "eventTime": "2018-05-28T19:16:43Z",
 "eventSource": "eks.amazonaws.com",
 "eventName": "CreateCluster",
 "awsRegion": "region-code",
 "sourceIPAddress": "205.251.233.178",
 "userAgent": "PostmanRuntime/6.4.0",
 "requestParameters": {
 "resourcesVpcConfig": {
 "subnetIds": [
 "subnet-a670c2df",
 "subnet-4f8c5004"
]
 }
 },
 "roleArn": "arn:aws:iam::111122223333:role/AWSServiceRoleForAmazonEKS-CAC1G1VH3ZKZ",
 "clusterName": "test"
},
"responseElements": {
 "cluster": {
 "clusterName": "test",
 "status": "CREATING",
 "createdAt": 1527535003.208,
 "certificateAuthority": {},
 "arn": "arn:aws:eks:region-code:111122223333:cluster/test",
 "roleArn": "arn:aws:iam::111122223333:role/AWSServiceRoleForAmazonEKS-CAC1G1VH3ZKZ",
 "version": "1.10",
 "resourcesVpcConfig": {
 "securityGroupIds": [],
```



```

 "vpcId": "vpc-21277358",
 "subnetIds": [
 "subnet-a670c2df",
 "subnet-4f8c5004"
]
 },
 "requestID": "a7a0735d-62ab-11e8-9f79-81ce5b2b7d37",
 "eventID": "eab22523-174a-499c-9dd6-91e7be3ff8e3",
 "readOnly": false,
 "eventType": "AwsApiCall",
 "recipientAccountId": "111122223333"
}

```

## Amazon EKS サービスにリンクされたロールのログエントリ

Amazon EKS サービスにリンクされたロールは、AWS リソースへの API コールを行います。Amazon EKS サービスにリンクされたロールによって行われた呼び出しには、CloudTrail ログエントリが `username: AWSServiceRoleForAmazonEKS` と `username: AWSServiceRoleForAmazonEKSNodegroup` で表示されます。Amazon EKS およびサービスにリンクされたロールの詳細については、「[the section called “サービスにリンクされた役割”](#)」を参照してください。

次の例では、`sessionContext` に記録され、`AWSServiceRoleForAmazonEKSNodegroup` サービスにリンクされたロールによって行われた [DeleteInstanceProfile](#) アクションを示す CloudTrail ログエントリが示されています。

```

{
 "eventVersion": "1.05",
 "userIdentity": {
 "type": "AssumedRole",
 "principalId": "AR0A3WHGPEZ7SJ2CW55C5:EKS",
 "arn": "arn:aws:sts::111122223333:assumed-role/AWSServiceRoleForAmazonEKSNodegroup/EKS",
 "accountId": "111122223333",
 "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
 "sessionContext": {
 "sessionIssuer": {
 "type": "Role",
 "principalId": "AR0A3WHGPEZ7SJ2CW55C5",

```

```

 "arn": "arn:aws:iam::111122223333:role/aws-service-role/eks-
nodegroup.amazonaws.com/AWSServiceRoleForAmazonEKSNodegroup",
 "accountId": "111122223333",
 "userName": "AWSServiceRoleForAmazonEKSNodegroup"
 },
 "webIdFederationData": {},
 "attributes": {
 "mfaAuthenticated": "false",
 "creationDate": "2020-02-26T00:56:33Z"
 }
},
"invokedBy": "eks-nodegroup.amazonaws.com"
},
"eventTime": "2020-02-26T00:56:34Z",
"eventSource": "iam.amazonaws.com",
"eventName": "DeleteInstanceProfile",
"awsRegion": "region-code",
"sourceIPAddress": "eks-nodegroup.amazonaws.com",
"userAgent": "eks-nodegroup.amazonaws.com",
"requestParameters": {
 "instanceProfileName": "eks-11111111-2222-3333-4444-abcdef123456"
},
"responseElements": null,
"requestID": "11111111-2222-3333-4444-abcdef123456",
"eventID": "11111111-2222-3333-4444-abcdef123456",
"eventType": "AwsApiCall",
"recipientAccountId": "111122223333"
}

```

## Amazon EC2 Auto Scaling グループのメトリクスを表示する

Amazon EKS マネージドノードグループでは、Amazon EC2 Auto Scaling グループのメトリクスがデフォルトで有効になっており、追加料金はかかりません。Auto Scaling グループは、サンプリングされたデータを毎分 Amazon CloudWatch に送信します。これらのメトリクスは、Auto Scaling グループの名前で絞り込むことができます。これらにより、時間の経過に伴うグループのサイズの変化など、マネージドノードグループで使用する Auto Scaling グループの履歴を継続的に把握できます。Auto Scaling グループのメトリクスは、[Amazon CloudWatch](#) または Auto Scaling コンソールで使用できます。詳細については、「[Monitor CloudWatch metrics for your Auto Scaling groups and instances](#)」を参照してください。

Auto Scaling グループのメトリクス収集を使用することで、マネージドノードグループのスケールリングをモニタリングできます。Auto Scaling グループのメトリクスは、Auto Scaling グループの最小サ

イズ、最大サイズ、および必要なサイズを報告します。ノードグループ内のノード数が最小サイズを下回った場合にアラームを作成できます。これは、ノードグループに異常があることを示しています。ノードグループサイズを追跡することは、データプレーンの容量が不足しないように最大数を調整する場合にも役立ちます。

これらのメトリクスを収集しない場合は、全部または一部のメトリクスのみを無効にすることができます。例えば、これを実行して、CloudWatch ダッシュボードのノイズを避けることができます。詳細については、「[Amazon CloudWatch metrics for Amazon EC2 Auto Scaling](#)」を参照してください。

## ADOT Operator を使用してメトリクスとトレースデータを送信する

Amazon EKS は、AWS Management Console、AWS CLI、および Amazon EKS API を使用して、[AWS Distro for OpenTelemetry \(ADOT\)](#) 演算子をインストールおよび管理することをサポートしています。これにより、Amazon EKS 上で実行されているアプリケーションが、メトリクスやトレースデータを [Amazon CloudWatch](#)、[Prometheus](#)、[X-Ray](#) などの複数のモニタリングサービスオプションに送信することが容易になります。

詳細については、AWS Distro for OpenTelemetry ドキュメントの「[Getting Started with AWS Distro for OpenTelemetry using EKS Add-Ons](#)」を参照してください。

## EKS ダッシュボードを使用してクラスターリソースに関する集計データを表示する

image::images/eks-dashboard.png[screenshot of account level cluster metrics]

### Amazon EKS ダッシュボードとは

Amazon EKS ダッシュボードは、複数の AWS リージョンと AWS アカウントにまたがる Kubernetes クラスターの統合可視性を提供します。このダッシュボードでは、次のことができます。

- 今後 90 日以内にサポート終了の自動アップグレードが予定されているクラスターを追跡する。
- 延長サポートのクラスターにおける EKS コントロールプレーンのコストを予測する。
- アップグレード前に注意が必要なインサイトを含むクラスターを確認する。
- 特定の AMI バージョンを実行しているマネージド型ノードグループを特定する。

- クラスターサポートタイプの分布 (標準サポートと拡張サポート) をモニタリングする。

EKS ダッシュボードは EKS クラスターインサイトと統合され、非推奨の Kubernetes API の使用など、クラスターの問題を表示します。詳細については、「[the section called “クラスターのインサイト”](#)」を参照してください。

#### Note

EKS ダッシュボードはリアルタイムではなく、12 時間ごとに更新されます。クラスターのリアルタイムモニタリングについては、「[クラスターをモニタリングする](#)」を参照してください

## ダッシュボードは AWS Organizations をどのように使用しますか？

Amazon EKS ダッシュボードの機能には AWS Organizations の統合が必要です。AWS Organizations を活用して、アカウント間でクラスター情報を安全に収集します。この統合により、AWS インフラストラクチャのスケールに応じた一元管理とガバナンスが可能になります。

AWS Organizations がインフラストラクチャに対して有効になっていない場合は、セットアップ手順について「AWS [Organizations ユーザーガイド](#)」を参照してください。

### クロスリージョンおよびクロスアカウントアクセス

EKS ダッシュボードは、AWS Organization のメンバーである任意のアカウントのクラスターリソースを表示できます。組織内の AWS アカウントのリストを生成するには、「[Export details for all accounts in an organization](#)」を参照してください。

us-east-1 AWS リージョンはダッシュボードを生成します。ダッシュボードを表示するには、このリージョンにログインする必要があります。ダッシュボードは AWS リージョン間でデータを集約しますが、これには GovCloud または中国リージョンは含まれません。

### 重要な用語

- AWS Organization: 複数の AWS アカウントの統一された管理構造。
- 管理アカウント: AWS Organization を制御するプライマリアカウント。
- メンバーアカウント: 管理アカウントを除く組織内の任意のアカウント。
- 委任管理者: 特定のクロスアカウント管理権限を付与されたメンバーアカウント。管理アカウント内で、AWS サービスごとに 1 つの委任管理者アカウントを選択できます。

- 信頼されたアクセス: 組織アカウント間でクラスター情報にアクセスするための EKS ダッシュボードの承認。
- サービスリンクロール (SLR): AWS サービスに直接リンクされた一意のタイプの IAM ロール。EKS ダッシュボードは SLR を使用して、アカウントと組織に関する情報を読み込みます。
- 詳細については、「AWS Organizations ユーザーガイド」の「[Terminology and concepts](#)」を参照してください。

## 全般的な概要

1. AWS Organization の管理アカウントにアクセスします。
  - 管理アカウントにアクセスする手順は、AWS Organization の設定方法によって異なります。例えば、AWS [Identity Center](#) または [Okta](#) を介して管理アカウントにアクセスできます。
2. EKS コンソールを介して信頼されたアクセスを有効にします。
3. AWS アカウント ID を使用して委任管理者を割り当てます。
4. 委任管理者アカウントに切り替えます。
5. 組織全体の可視性を備えた拡張 EKS コンソールにアクセスします。

## AWS コンソールを使用して EKS ダッシュボードを有効にする

### Important

EKS ダッシュボードを有効にするには、AWS Organization の管理アカウントにログインする必要があります。

## EKS ダッシュボード設定にアクセスする

1. 次の点を確認します。
  - a. AWS Organizations が有効で設定されている。
  - b. 組織の管理アカウントにログインしている。
  - c. us-east-1 リージョンで AWS マネジメントコンソールを表示している。
2. EKS コンソールに移動します。
3. 左側のサイドバーで、ダッシュボード設定を開きます。

## Amazon EKS ダッシュボードへのアクセスを設定する

1. EKS ダッシュボードの表示を許可する AWS アカウントの AWS アカウント ID を見つけます。
  - a. このステップはオプションですが、推奨されます。このステップを実行しない場合は、管理アカウントからのみダッシュボードにアクセスできます。ベストプラクティスとして、管理アカウントへのアクセスを制限する必要があります。
2. [トラステッドアクセスを有効化] をクリックします。
  - a. これで、管理アカウントからダッシュボードを表示できるようになりました。
3. [委任管理者を登録] をクリックし、ダッシュボードの表示に使用する AWS アカウントのアカウント ID を入力します。
  - a. これで、委任管理者アカウントまたは管理アカウントからダッシュボードを表示できるようになりました。

ダッシュボードを有効にするために必要なアクセス許可の詳細については、「[最低限必要な IAM ポリシー](#)」を参照してください。

## EKS ダッシュボードを表示する

1. 委任管理者アカウント (推奨) または管理アカウントにログインします。
2. us-east-1 リージョンにログインします。
3. EKS サービスに移動し、左側のサイドバーからダッシュボードを選択します。

### Note

[EKS ダッシュボードの表示に必要な IAM アクセス許可を確認します。](#)

## ダッシュボードを設定する

ダッシュボードのビューを設定し、リソースをフィルタリングできます。

### 利用可能なリソース

- クラスター: EKS クラスターのステータスと場所に関する集計情報を確認します。
  - ヘルスの問題があるクラスター。
  - EKS のクラスター延長サポート。

- Kubernetes バージョン別のクラスターの内訳。
- マネージド型ノードグループ: マネージド型ノードグループと EC2 インスタンスを確認します。
  - Amazon Linux や Bottlerocket などの AMI タイプ別のノードグループ。
  - ノードグループヘルスの問題。
  - インスタンスタイプの分散。
- アドオン: インストールした Amazon EKS アドオンとそのステータスについて確認します。
  - アドオンあたりのインストール数。
  - ヘルスの問題があるアドオン。
  - アドオンあたりのバージョン分布。

## 利用可能なビュー

- グラフビュー
  - 選択したリソースのグラフと視覚化を表示するカスタマイズ可能なウィジェットビュー。
  - ウィジェットの削除など、グラフビューへの変更は、EKS ダッシュボードのすべてのユーザーに表示されます。
- リソースビュー
  - 選択したリソースのリストビュー。フィルターをサポートします。
- マップビュー
  - 選択したリソースの地理的分布を表示します。

## EKS ダッシュボードをフィルタリングする

EKS ダッシュボードは、次の方法でフィルタリングできます。

- AWS アカウント
- AWS Organizations で定義される組織単位
- AWS リージョン

## AWS コンソールを使用して EKS ダッシュボードを無効にする

1. 次の点を確認します。
  - a. AWS Organizations が有効で設定されている。

- b. 組織の管理アカウントにログインしている。
  - c. us-east-1 リージョンで AWS マネジメントコンソールを表示している。
2. EKS コンソールに移動します。
3. 左側のサイドバーで、ダッシュボード設定を開きます。
4. [トラステッドアクセスを無効化] をクリックします。

## EKS ダッシュボードのトラブルシューティング

### EKS ダッシュボードの有効化に関する問題

- AWS Organization の管理アカウントにログインする必要があります。
  - AWS Organization がない場合は、作成します。「[Create and configure an organization](#)」ではその方法を説明しています。
  - AWS アカウントが既に AWS Organization のメンバーである場合は、組織の管理者を特定します。
- AWS Organizations リソースを作成および更新するには、十分な IAM アクセス許可で AWS アカウントにログインする必要があります。

### EKS ダッシュボードの表示に関する問題

- 次のいずれかの AWS アカウントにログインする必要があります。
  - AWS Organization の管理アカウント
  - 管理アカウントの EKS ダッシュボード設定で識別される委任管理者アカウント。
- EKS ダッシュボードを最近有効にした場合、最初のデータセットには最大 12 時間かかる場合があることに注意してください。

### ダッシュボードウィジェットが予期せず移動する

- EKS ダッシュボードは、設定可能なウィジェットビューを AWS アカウントレベルで保存します。ウィジェットビューを変更すると、同じ AWS アカウントを使用している他のユーザーに変更が表示されます。



## EKS ダッシュボードと AWS Organizations の統合を設定する

このセクションでは、EKS ダッシュボードの AWS Organizations との統合を設定するための手順を説明します。サービス間の信頼されたアクセスを有効または無効にする方法と、委任管理者アカウントを登録および登録解除する方法について説明します。各設定タスクは、AWS コンソールまたは AWS CLI のいずれかを使用して実行できます。

### 信頼されたアクセスを有効にする

信頼されたアクセスは、EKS ダッシュボードが組織内のすべてのアカウントでクラスター情報に安全にアクセスすることを許可します。

#### AWS コンソールを使用する場合

1. AWS Organization の管理アカウントにログインします。
2. us-east-1 リージョンの EKS コンソールに移動します。
3. 左側のサイドバーで、ダッシュボード設定を選択します。
4. [トラステッドアクセスを有効化] をクリックします。

#### Note

EKS コンソールを使用して信頼されたアクセスを有効にすると、AWSServiceRoleForAmazonEKSDashboard サービスにリンクされたロールが自動的に作成されます。AWS CLI または AWS Organizations コンソールを使用して信頼されたアクセスを有効にしても、この自動作成は行われません。

#### AWS CLI の使用

1. AWS Organization の管理アカウントにログインします。
2. 以下の コマンドを実行します。

```
aws iam create-service-linked-role --aws-service-name dashboard.eks.amazonaws.com
aws organizations enable-aws-service-access --service-principal eks.amazonaws.com
```

## 信頼されたアクセスを無効にする

信頼されたアクセスを無効にすると、組織のアカウント全体でクラスター情報にアクセスするための EKS ダッシュボードのアクセス許可が取り消されます。

### AWS コンソールを使用する場合

1. AWS Organization の管理アカウントにログインします。
2. us-east-1 リージョンの EKS コンソールに移動します。
3. 左側のサイドバーで、ダッシュボード設定を選択します。
4. [トラステッドアクセスを無効化] をクリックします。

### AWS CLI の使用

1. AWS Organization の管理アカウントにログインします。
2. 次のコマンドを実行してください。

```
aws organizations disable-aws-service-access --service-principal eks.amazonaws.com
```

## 委任管理者アカウントを有効にする

委任管理者は、EKS ダッシュボードへのアクセス許可が付与されたメンバーアカウントです。

### AWS コンソールを使用する場合

1. AWS Organization の管理アカウントにログインします。
2. us-east-1 リージョンの EKS コンソールに移動します。
3. 左側のサイドバーで、ダッシュボード設定を選択します。
4. [委任管理者を登録] をクリックします。
5. 委任管理者として選択する AWS アカウントのアカウント ID を入力します。
6. 登録を確認します。

### AWS CLI の使用

1. AWS Organization の管理アカウントにログインします。
2. 123456789012 を自分のアカウント ID に置き換えて、以下のコマンドを実行します。

```
aws organizations register-delegated-administrator --account-id 123456789012 --
service-principal eks.amazonaws.com
```

## 委任管理者アカウントを無効にする

委任管理者を無効にすると、EKS ダッシュボードにアクセスするためのアカウントのアクセス許可が削除されます。

### AWS コンソールを使用する場合

1. AWS Organization の管理アカウントにログインします。
2. us-east-1 リージョンの EKS コンソールに移動します。
3. 左側のサイドバーで、ダッシュボード設定を選択します。
4. リストで委任管理者を見つけます。
5. 委任管理者として削除するアカウントの横にある [登録解除] をクリックします。

### AWS CLI の使用

1. AWS Organization の管理アカウントにログインします。
2. 委任管理者のアカウント ID を 123456789012 に置き換えて、次のコマンドを実行します。

```
aws organizations deregister-delegated-administrator --account-id 123456789012 --
service-principal eks.amazonaws.com
```

## 最低限必要な IAM ポリシー

このセクションでは、信頼されたアクセスを有効にし、EKS ダッシュボードと AWS Organizations の統合のために管理者を委任するために必要な最小限の IAM ポリシーの概要を説明します。

### 信頼されたアクセスを有効にするためのポリシー

EKS ダッシュボードと AWS Organizations 間の信頼されたアクセスを有効にするには、以下のアクセス許可が必要です。

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

```

 "Statement": [
 {
 "Effect": "Allow",
 "Action": [
 "organizations:EnableAWSServiceAccess",
 "organizations:DescribeOrganization",
 "organizations:ListAWSServiceAccessForOrganization"
],
 "Resource": "*"
 },
 {
 "Effect": "Allow",
 "Action": [
 "iam:CreateServiceLinkedRole"
],
 "Resource": "arn:aws:iam::*:role/aws-service-role/
dashboard.eks.amazonaws.com/AWSServiceRoleForAmazonEKSDashboard"
 }
]
 }
}

```

## 管理者を委任するためのポリシー

EKS ダッシュボードの委任管理者を登録または登録解除するには、以下のアクセス許可が必要です。

```

{
 "Version": "2012-10-17",
 "Statement": [
 {
 "Effect": "Allow",
 "Action": [
 "organizations:RegisterDelegatedAdministrator",
 "organizations:DeregisterDelegatedAdministrator",
 "organizations:ListDelegatedAdministrators"
],
 "Resource": "*"
 }
]
}

```

## EKS ダッシュボードを表示するポリシー

```
{
 "Version": "2012-10-17",
 "Statement": [
 {
 "Sid": "AmazonEKSDashboardReadOnly",
 "Effect": "Allow",
 "Action": [
 "eks:ListDashboardData",
 "eks:ListDashboardResources",
 "eks:DescribeClusterVersions"
],
 "Resource": "*"
 },
 {
 "Sid": "AmazonOrganizationsReadOnly",
 "Effect": "Allow",
 "Action": [
 "organizations:DescribeOrganization",
 "organizations:ListAWSServiceAccessForOrganization",
 "organizations:ListRoots",
 "organizations:ListAccountsForParent",
 "organizations:ListOrganizationalUnitsForParent"
],
 "Resource": "*"
 },
 {
 "Sid": "AmazonOrganizationsDelegatedAdmin",
 "Effect": "Allow",
 "Action": [
 "organizations:ListDelegatedAdministrators"
],
 "Resource": [
 "*"
],
 "Condition": {
 "StringEquals": {
 "organizations:ServicePrincipal": "eks.amazonaws.com"
 }
 }
 }
]
}
```

```
}
```

**Note**

これらのポリシーは、AWS Organization の管理アカウントの IAM プリンシパル (ユーザーまたはロール) にアタッチする必要があります。メンバーアカウントは、信頼されたアクセスを有効にしたり、管理者を委任したりすることはできません。

# 統合 AWS サービスによる EKS の強化

他のセクションで説明されているサービスに加えて、Amazon EKS はより多くの AWS サービスと連携して追加のソリューションを提供しています。このトピックで説明するサービスは、Amazon EKS を使用して機能を追加するか、タスクを実行するために Amazon EKS に使用されるかのいずれかです。

## トピック

- [AWS CloudFormation を使用して Amazon EKS リソースを作成する](#)
- [Amazon Detective を使用して EKS のセキュリティイベントを分析する](#)
- [Amazon GuardDuty で脅威を検出する](#)
- [AWS Resilience Hub で EKS クラスターの耐障害性を評価する](#)
- [Security Lake を使用して EKS セキュリティデータを一元化して分析する](#)
- [Amazon VPC Lattice との安全なクラスター間接続を有効にする](#)
- [AWS Local Zones を使用して低レイテンシーの EKS クラスターを起動する](#)

## AWS CloudFormation を使用して Amazon EKS リソースを作成する

Amazon EKS は AWS CloudFormation と統合されています。これは、リソースとインフラストラクチャの作成と管理の所要時間を短縮できるように AWS リソースをモデル化して設定するためのサービスです。すべての必要な AWS リソース (Amazon EKS クラスターなど) を記述するテンプレートを作成すれば、これらのリソースのプロビジョニングと設定は AWS CloudFormation で代行されます。

AWS CloudFormation を使用すると、テンプレートを再利用して Amazon EKS リソースを同じように繰り返してセットアップできます。リソースを一度記述するだけで、複数の AWS アカウントとリージョンに同じリソースを何度もプロビジョニングすることができます。

## Amazon EKS と AWS CloudFormation のテンプレート

Amazon EKS および関連サービスのリソースをプロビジョニングして設定するには、[AWS CloudFormation テンプレート](#)について理解しておく必要があります。テンプレートは、JSON や YAML でフォーマットされたテキストファイルです。これらのテンプレートは、AWS CloudFormation スタックでプロビジョニングするリソースについて記述します。JSON または

YAML に詳しくない場合は、AWS CloudFormation Designer を使用して AWS CloudFormation テンプレートの使用を開始できます。詳細については、AWS CloudFormation ユーザーガイドの「[AWS CloudFormation デザイナーとは](#)」を参照してください。

Amazon EKS では、AWS CloudFormation でのクラスターとノードグループの作成がサポートされています。Amazon EKS リソースの JSON テンプレートと YAML テンプレートの例を含む詳細情報については、AWS CloudFormation ユーザーガイドの「[Amazon EKS リソースタイプのリファレンス](#)」を参照してください。

## AWS CloudFormation の詳細はこちら

AWS CloudFormation の詳細については、以下のリソースを参照してください。

- [AWS CloudFormation](#)
- [AWS CloudFormation ユーザーガイド](#)
- [AWS CloudFormation コマンドラインインターフェイスユーザーガイド](#)

## Amazon Detective を使用して EKS のセキュリティイベントを分析する

[Amazon Detective](#) を使用すると、セキュリティに関する検出結果や疑わしいアクティビティの根本原因を分析、調査、および迅速に特定できます。Detective は、AWS リソースからログデータを自動的に収集します。その後、機械学習、統計分析、グラフ理論を使用して、セキュリティ調査をより迅速かつ効率的に行うのに役立つビジュアライゼーションを生成します。Detective の事前に作成されたデータの集計、要約、およびコンテキストは、考えられるセキュリティ問題の性質と範囲を迅速に分析および特定するのに役立ちます。詳細については、「[Amazon Detective ユーザーガイド](#)」を参照してください。

Detective は、Kubernetes と AWS のデータを次のような結果に整理します。

- Amazon EKS クラスターの詳細 (クラスターやクラスターのサービスロールを作成した IAM ID など)。Detective を使用して、これらの IAM ID の AWS および Kubernetes API アクティビティを調査できます。
- コンテナの詳細 (イメージやセキュリティコンテキストなど)。終了済みの Pod の詳細を確認することもできます。
- Kubernetes API アクティビティ (API アクティビティの全体的な傾向と特定の API 呼び出しの詳細の両方を含む)。例えば、選択した時間範囲中に発行されて、成功および失敗した Kubernetes API



呼び出しの数を表示できます。また、新たに確認された API 呼び出しに関するセクションは、疑わしいアクティビティを特定するのに役立つ場合があります。

Amazon EKS 監査ログは、Detective の動作グラフに追加できるオプションのデータソースパッケージです。利用可能なオプションのソースパッケージとそのステータスは、アカウント内で確認できます。詳細については、「Amazon Detective ユーザーガイド」の「[Amazon EKS audit logs for Detective](#)」を参照してください。

## Amazon EKS との連携で Amazon Detective を使用する

結果を確認できるようにするには、クラスターが配置されているのと同じ AWS リージョンで、Detective を少なくとも 48 時間有効にする必要があります。詳細については、「Amazon Detective ユーザーガイド」の「[Amazon Detective の設定](#)」を参照してください。

1. <https://console.aws.amazon.com/detective/> で Detective コンソールを開きます。
2. 左のナビゲーションペインで [検索] を選択します。
3. [タイプを選択] を選択し、[EKS クラスター] を選択します。
4. クラスター名または ARN を入力し、[検索] を選択します。
5. 検索結果で、アクティビティを表示するクラスターの名前を選択します。表示できる内容の詳細については、「Amazon Detective ユーザーガイド」の「[Overall Kubernetes API activity involving an Amazon EKS cluster](#)」を参照してください。

## Amazon GuardDuty で脅威を検出する

Amazon GuardDuty は、AWS 環境内のアカウント、コンテナ、ワークロード、データを保護する脅威検知サービスです。GuardDuty は、機械学習 (ML) モデル、異常および脅威検出機能を使用して、さまざまなログソースとランタイムアクティビティを継続的に監視し、環境内の潜在的なセキュリティリスクと悪意のあるアクティビティを特定して優先順位を付けます。

GuardDuty には、EKS クラスターに対する潜在的な脅威を検出する EKS Protection とランタイムモニタリングの 2 つの機能があります。

### Note

新規: Amazon EKS Auto Mode は GuardDuty と統合されています。

## EKS Protection

脅威検出を備えた機能であり、関連付けられた Kubernetes 監査ログをモニタリングすることで、Amazon EKS クラスターを保護できるようになります。Kubernetes 監査ログは、ユーザー、Kubernetes API を使用するアプリケーション、コントロールプレーンからのアクティビティなど、クラスター内のシーケンシャルアクションをキャプチャします。例えば、Kubernetes クラスター内のリソースを改ざんするために呼び出された可能性のある API が、認証されていないユーザーによって呼び出された場合、GuardDuty はそのことを識別できます。

EKS Protection を有効にすると、GuardDuty は継続的な脅威検出のためにのみ Amazon EKS 監査ログにアクセスできるようになります。GuardDuty は、クラスターに対する潜在的な脅威を識別すると、特定のタイプの関連する Kubernetes 監査ログの検出結果を生成します。Kubernetes 監査ログから得られる検出結果のタイプの詳細については、「Amazon GuardDuty ユーザーガイド」の「[Amazon GuardDuty User Guide](#)」を参照してください。

詳細については、「Amazon GuardDuty ユーザーガイド」の「[EKS Protection](#)」を参照してください。

## Runtime Monitoring

この機能は、オペレーティングシステムレベル、ネットワーク、ファイルのイベントを監視して分析し、環境内の特定の AWS ワークロードにおける潜在的な脅威を検出するのに役立ちます。

ランタイムモニタリングを有効にして Amazon EKS クラスターに GuardDuty エージェントをインストールすると、GuardDuty はこのクラスターに関連付けられたランタイムイベントのモニタリングを開始します。GuardDuty エージェントとランタイムモニタリングは Amazon EKS Hybrid Nodes では使用できないため、ランタイムモニタリングはハイブリッドノードで発生するランタイムイベントでは使用できないことにご注意ください。GuardDuty がクラスターに対する潜在的な脅威を特定すると、関連するランタイムモニタリングの結果を生成します。例えば、脅威は、脆弱なウェブアプリケーションを実行している 1 つのコンテナを危険にさらすことから始まる可能性があります。このウェブアプリケーションには、基盤となるコンテナとワークロードへのアクセス権限が存在する可能性があります。この場合、認証情報が正しく設定されていないと、アカウントとその中に保存されているデータへのアクセスを制御できない可能性があります。

ランタイムモニタリングを設定するには Amazon EKS アドオンとしてクラスターに GuardDuty エージェントをインストールします。アドオンの詳細については、「[the section called “AWS アドオン”](#)」を参照してください。

詳細については、「Amazon GuardDuty ユーザーガイド」の「[ランタイムモニタリング](#)」を参照してください。

## AWS Resilience Hub で EKS クラスターの耐障害性を評価する

AWS は、Amazon EKS クラスターのインフラストラクチャを分析して、その耐障害性を評価します。AWSResilience Hub は、Kubernetes のロールベースのアクセスコントロール (RBAC) 設定を使用して、クラスターにデプロイされた Kubernetes のワークロードを評価します。詳細については、「AWS Resilience Hub ユーザーガイド」の「[Amazon EKS クラスターへの AWS Resilience Hub のアクセスを有効にする](#)」を参照してください。

## Security Lake を使用して EKS セキュリティデータを一元化して分析する

Amazon Security Lake は、Amazon EKS を含むさまざまなソースからのセキュリティデータを一元化するフルマネージドのセキュリティデータレイクサービスです。Amazon EKS を Security Lake と統合することで、Kubernetes リソースで実行されるアクティビティに関するより深いインサイトを手に入れ、Amazon EKS クラスターのセキュリティ体制を強化することができます。

### Note

Amazon EKS で Security Lake を使用方法、およびデータソースを設定する方法の詳細については、[Amazon Security Lake のドキュメント](#)を参照してください。

## Security Lake を Amazon EKS で使用する利点

セキュリティデータを一元化 - Security Lake は、Amazon EKS クラスターのセキュリティデータを、他の AWS のサービス、SaaS プロバイダー、オンプレミスソース、サードパーティーソースのデータとともに、自動的に収集して一元化します。これにより、組織全体のセキュリティ体制を包括的に把握することができます。

標準化されたデータ形式 — Security Lake は、収集したデータを標準的なオープンソースのスキーマである [Open Cybersecurity Schema Framework \(OCSF\) 形式](#)に変換します。この標準化により、分析が容易になり、他のセキュリティツールやサービスとの統合が可能になります。

脅威検出の向上 — Amazon EKS コントロールプレーンのログを含む一元化されたセキュリティデータを分析することで、Amazon EKS クラスター内の疑わしいアクティビティをより効果的に検出することができます。これにより、セキュリティインシデントをすみやかに特定し、対応することができます。

データ管理の簡素化 — Security Lake は、カスタマイズ可能な保持とレプリケーションの設定によって、セキュリティデータのライフサイクルを管理します。これにより、データ管理のタスクが簡素化され、コンプライアンスや監査に必要なデータを確実に保持することができます。

## Amazon EKS の Security Lake の有効化

1. EKS クラスターの Amazon EKS コントロールプレーンログ記録を有効にします。詳細な手順については、「[コントロールプレーンログの有効化と無効化](#)」を参照してください。
2. [Amazon EKS 監査ログを Security Lake のソースとして追加します](#)。次に、Security Lake が、EKS クラスターで実行中の Kubernetes リソースで行われたアクティビティの詳細情報の収集を開始します。
3. 要件に基づいて、Security Lake のセキュリティデータの[保持とレプリケーションを設定します](#)。
4. Security Lake に保存されている標準化された OCSF データは、インシデント対応、セキュリティ分析、その他の AWS のサービスやサードパーティーツールとの統合に使用できます。例えば、[Amazon OpenSearch Ingestion を使用して Amazon Security Lake のデータからセキュリティインサイトを生成](#)することができます。

## Security Lake の EKS ログの分析

Security Lake は EKS ログイベントを OCSF 形式に標準化し、データの分析や、他のセキュリティイベントとの関連付けを簡単に行えるようにします。標準化されたデータは、Amazon Athena、Amazon QuickSight、サードパーティーのセキュリティ分析ツールなどさまざまなツールやサービスを使ってクエリしたり、可視化したりできます。

EKS ログイベントの OCSF マッピングの詳細については、OCSF GitHub リポジトリの「[https://github.com/ocsf/examples/tree/main/mappings/markdown/AWS/v1.1.0/EKS Audit Logs\[mapping reference\]](https://github.com/ocsf/examples/tree/main/mappings/markdown/AWS/v1.1.0/EKS%20Audit%20Logs[mapping%20reference])」を参照してください。

## Amazon VPC Lattice との安全なクラスター間接続を有効にする

Amazon VPC Lattice は、AWS ネットワークインフラストラクチャに直接組み込まれた完全マネージド型のアプリケーションネットワーキングサービスで、複数のアカウントや仮想プライベートクラウド (VPC) を通してサービスの接続、保護、モニタリングに使用できます。Amazon EKS では、Kubernetes 「[ゲートウェイ API](#)」の実装である AWS ゲートウェイ API コントローラーを使用して Amazon VPC Lattice を活用できます。Amazon VPC Lattice を使用すると、標準的な Kubernetes セマンティクスでクラスター間の接続を簡単かつ一貫した方法でセットアップできます。Amazon

EKS で Amazon VPC Lattice の使用を開始するには、「[AWS ゲートウェイ API コントローラのユーザーガイド](#)」を参照してください。

## AWS Local Zones を使用して低レイテンシーの EKS クラスターを起動する

[AWS Local Zones](#) は AWS リージョンの拡張であり、ユーザーに近い場所に配置されます。Local Zones は、インターネットへの独自の接続を持ち、[AWS Direct Connect](#) をサポートします。Local Zone で作成したリソースは、低いレイテンシーの通信をローカルユーザーに提供できます。詳細については、「[AWS Local Zones ユーザーガイド](#)」と、「Amazon EC2 ユーザーガイド」の「[Local Zones](#)」を参照してください。

Amazon EKS では、Local Zones の特定のリソースをサポートしています。これには、[マネージドノードグループ](#)、[セルフマネージド Amazon EC2 ノード](#)、Amazon EBS ボリューム、および Application Load Balancer (ALB) が含まれます。Amazon EKS クラスターの一部として Local Zones を使用する場合は、次を考慮することをお勧めします。

- Amazon EKS を使用して Local Zones に Fargate ノードを作成することはできません。
- Amazon EKS が管理する Kubernetes コントロールプレーンは、常に AWS リージョンで実行されます。Amazon EKS が管理する Kubernetes コントロールプレーンを Local Zone で実行することはできません。Local Zones は VPC 内のサブネットとして表示されるため、Kubernetes は Local Zone のリソースをそのサブネットの一部として認識します。
- Amazon EKS Kubernetes クラスターは、Amazon EKS が管理する [Elastic Network Interface](#) を使用して、AWS リージョンまたは Local Zone で実行する Amazon EC2 インスタンスと通信します。Amazon EKS ネットワークアーキテクチャの詳細については、「[ネットワークを設定する](#)」を参照してください。
- リージョンサブネットとは異なり、Amazon EKS はローカルゾーンサブネットにネットワークインターフェイスを配置できません。つまり、クラスターの作成時に Local Zone サブネットを指定することはできません。

# Amazon EKS クラスターとノードに関する問題をトラブルシューティングする

この章では Amazon EKS の使用中に表示される一般的なエラーとその回避方法について説明します。特定の Amazon EKS 領域のトラブルシューティングが必要な場合、別の [the section called “トラブルシューティング”](#)、[the section called “EKS Connector をトラブルシューティングする”](#)、および [「EKS アドオンを使用した ADOT のトラブルシューティング」](#) を参照してください。

その他のトラブルシューティング情報については AWS re:Post の [「Amazon Elastic Kubernetes Service に関するナレッジセンターのコンテンツ」](#) を参照してください。

## 容量不足

Amazon EKS クラスターを作成しようとするときに次のエラーが表示された場合、指定したいいずれかのアベイラビリティゾーンに、クラスターをサポートするための十分な容量がありません。

```
Cannot create cluster 'example-cluster' because region-1d, the targeted Availability Zone, does not currently have sufficient capacity to support the cluster. Retry and choose from these Availability Zones: region-1a, region-1b, region-1c
```

このエラーメッセージによって返されるアベイラビリティゾーンでホストされているクラスター VPC 内のサブネットを使用して、クラスターを再作成してください。

クラスターを配置できないアベイラビリティゾーンがあります。サブネットが属するアベイラビリティゾーンを、[サブネットの要件と考慮事項](#)内のアベイラビリティゾーンのリストと比較してください。

## ノードをクラスターに結合できません

ノードをクラスターに結合できない一般的な理由はいくつかあります。

- ノードがマネージド型の場合、Amazon EKS はノードグループの作成時に、aws-auth ConfigMap にエントリを追加します。エントリが削除または変更された場合は再度追加する必要があります。詳細についてはターミナルに `eksctl create iamidentitymapping --help` を入力してください。現在の aws-auth ConfigMap エントリを確認するには次のコマンドの `####` をクラスターの名前に置き換えて、変更したコマンド `eksctl get iamidentitymapping`



--cluster *my-cluster* を実行してください。指定する役割の ARN には / 以外の [パス](#) を含めることはできません。例えば、役割の名前が development/apps/my-role の場合、役割の ARN を指定するときに my-role に変更する必要があります。ノード IAM 役割 ARN (インスタンスプロファイルの ARN ではない) を指定していることを確認してください。

ノードがセルフマネージド型で、ノードの IAM [役割](#) の ARN のアクセスエントリを作成していない場合はマネージド型ノード用にリストされているのと同じコマンドを実行してください。ノード IAM 役割の ARN のアクセスエントリを作成している場合、そのエントリがアクセスエントリで正しく設定されていない可能性があります。aws-auth ConfigMap エントリまたはアクセスエントリのプリンシパル ARN として、ノード IAM 役割 ARN (インスタンスプロファイルの ARN ではない) が指定されていることを確認してください。アクセスエントリの詳細については「[the section called “許可を付与する”](#)」を参照してください。

- ノード AWS クラウドフォーメーション テンプレートの クラスター名 が、ノードを結合するクラスターの名前と正確に一致しません。このフィールドに正しくない値を渡すと、ノードの /var/lib/kubelet/kubeconfig ファイルの設定が正しくなくなるため、ノードはクラスターに結合されません。
- ノードはクラスターによって所有されているためタグ付けされていません。ノードには次のタグが適用されている必要があります。##### はクラスターの名前に置き換えられます。

| キー                                                  | 値     |
|-----------------------------------------------------|-------|
| kubernetes.io/cluster/ <i>my-cluste</i><br><i>r</i> | owned |

- ノードはパブリック IP アドレスを使用してクラスターにアクセスできない場合があります。パブリックサブネットにデプロイされたノードにパブリック IP アドレスが割り当てられていることを確認してください。割り当てられていない場合は起動後にノードに Elastic IP アドレスを関連付けることができます。詳細については「[Elastic IP アドレスをインスタンスまたはネットワークインターフェイスに関連付ける](#)」を参照してください。デプロイされたインスタンスにパブリック IP アドレスを自動的に割り当てるようにパブリックサブネットが設定されていない場合はその設定を有効にすることをお勧めします。詳細については「[サブネットの IPv4 アドレス指定属性の変更](#)」を参照してください。ノードがプライベートサブネットにデプロイされている場合、サブネットにはパブリック IP アドレスが割り当てられた NAT ゲートウェイへのルートが必要です。
- ノードのデプロイ先 AWS リージョンの AWS STS エンドポイントはアカウントに対して有効になっていません。リージョンを有効にするには「[AWS リージョンでの AWS STS のアクティブ化と非アクティブ化](#)」を参照してください。

- ノードにはプライベート DNS エントリがないため、node "" not found エラーを含む kubelet ログが記録されます。ノードが作成される VPC に、DHCP options set の Options として domain-name と domain-name-servers の値のセットがあることを確認してください。デフォルト値は domain-name:<region>.compute.internal および domain-name-servers:AmazonProvidedDNS です。詳細については「Amazon VPC ユーザーガイド」の「[DHCP オプションセット](#)」を参照してください。
- マネージドノードグループ内のノードが 15 分以内にクラスターに接続しない場合、「NodeCreationFailure」という正常性の問題が出力され、コンソールのステータスが Create failed に設定されます。起動時間が遅い Windows AMI の場合、この問題は[高速起動](#)を使用して解決できます。

ワーカーノードがクラスターに参加できない一般的な原因を特定してトラブルシューティングするにはAWSsupport-TroubleshootEKSWorkerNode ランブックを使用できます。詳細についてはAWS システム・マネージャー Automation ランブックリファレンスの「[AWSsupport-TroubleshootEKSWorkerNode](#)」を参照してください。

## 許可されていないか、アクセスが拒否されました (kubectl)

kubectl コマンドの実行中に次のいずれかのエラーが発生した場合は、kubectl が Amazon EKS に対して適切に設定されていないか、使用している IAM プリンシパルの認証情報 (役割またはユーザー) が、Amazon EKS クラスターで Kubernetes オブジェクトに対して十分なアクセス許可を持つ Kubernetes ユーザー名にマッピングされていません。

- could not get token: AccessDenied: Access denied
- error: You must be logged in to the server (Unauthorized)
- error: the server doesn't have a resource type "svc"

この原因としては次のいずれかが考えられます。

- クラスターはある IAM プリンシパルの認証情報を使用して作成されており、kubectl は別の IAM プリンシパルの認証情報を使用するように設定されています。この問題を解決するにはクラスターを作成した認証情報を使用するように kube config ファイルを更新してください。詳細については「[the section called “kubectl を使用してクラスターにアクセスする”](#)」を参照してください。
- クラスターが [EKS アクセスエントリを使用して Kubernetes へのアクセスを IAM ユーザーに許可する](#) の前提条件セクションの最小プラットフォーム要件を満たしている場合、IAM プリンシパル



にはアクセスエントリは存在しません。存在する場合、必要な Kubernetes グループ名が定義されていないか、適切なアクセスポリシーが関連付けられていません。詳細については、「[the section called “許可を付与する”](#)」を参照してください。

- クラスターが [EKS アクセス エントリを使用して IAM ユーザーに Kubernetes へのアクセスを許可する](#) の最小プラットフォーム要件を満たしていない場合、aws-auth ConfigMap に IAM プリンシパルのエントリが存在しません。存在しても、Kubernetes Role または ClusterRole にバインドされている、必要なアクセス許可を持つ Kubernetes グループ名にはマップされません。Kubernetes 役割ベースの承認 (RBAC) オブジェクトの詳細については、Kubernetes ドキュメントの「[RBAC 認可を使用する](#)」を参照してください。現在の aws-auth ConfigMap エントリを確認するには次のコマンドの ##### をクラスターの名前に置き換えて、変更したコマンド `eksctl get iamidentitymapping --cluster my-cluster` を実行してください。IAM プリンシパルの ARN を含むエントリが ConfigMap にはない場合はターミナルに `eksctl create iamidentitymapping --help` を入力して作成方法を確認してください。

AWS CLI をインストールして設定する場合は使用する IAM 認証情報を設定できます。詳細についてはAWS コマンドラインインターフェイスユーザーガイドの「[AWS CLI を設定する](#)」を参照してください。クラスター上の Kubernetes オブジェクトにアクセスする IAM ロールを引き受ける場合は、IAM ロールを使用するように kubectl を設定することもできます。詳細については、「[the section called “kubectl を使用してクラスターにアクセスする”](#)」を参照してください。

## hostname doesn't match

システムの Python のバージョンは 2.7.9 以降であることが必要です。それ以外の場合はAWS CLI で Amazon EKS を呼び出すと hostname doesn't match エラーが表示されます。詳細については「Python Requests のよくある質問」の「[What are "hostname doesn't match" errors?](#)」(ホスト名の不一致) エラーとは) を参照してください。

## getsockopt: no route to host

Docker は Amazon EKS クラスターの 172.17.0.0/16 CIDR 範囲で実行されます。クラスターの VPC サブネットがこの範囲と重ならないようにすることをお勧めします。重なっている場合は以下のエラーが発生します。

```
Error: : error upgrading connection: error dialing backend: dial tcp
172.17.<nn>.<nn>:10250: getsockopt: no route to host
```

# Instances failed to join the Kubernetes cluster

AWS Management Console で Instances failed to join the Kubernetes cluster エラーが表示された場合、クラスターのプライベートエンドポイントアクセスが有効になっているか、パブリックエンドポイントアクセス用に CIDR ブロックが正しく設定されていることを確認します。詳細については「[the section called “クラスターエンドポイントのアクセス”](#)」を参照してください。

## マネージド型ノードグループのエラー

マネージド型ノードグループでハードウェアのヘルスに問題が発生した場合は Amazon EKS によって問題の診断に役立つエラーメッセージが返されます。これらのヘルスチェックは [Amazon EC2 ヘルスチェック](#) に基づいているため、ソフトウェアの問題は検出されません。次のリストはエラーコードについての説明です。

### AccessDenied

Amazon EKS または 1 つ以上のマネージド型ノードが、Kubernetes クラスター API サーバーでの認証または認可に失敗しています。共通のレスポンスヘッダーの詳細については「[the section called “AccessDenied マネージド型ノードグループエラーを修正する”](#)」を参照してください。プライベート Windows AMI では、Not authorized for images エラーメッセージと共にこのエラーコードが表示されることもあります。詳細については、「[the section called “Not authorized for images”](#)」を参照してください。

### AmiIdNotFound

起動テンプレートに関連付けられている AMI ID が見つかりませんでした。AMI が存在し、アカウントと共有されていることを確認してください。

### AutoScalingGroupNotFound

マネージド型ノードグループに関連付けられている 自動スケーリング グループが見つかりませんでした。同じ設定で 自動スケーリング グループを再作成して復旧できる場合があります。

### ClusterUnreachable

Amazon EKS または 1 つ以上のマネージド型ノードが Kubernetes クラスター API サーバーと通信できません。このエラーはネットワークが中断したり、API サーバーがリクエストの処理をタイムアウトしている場合に発生する可能性があります。

## Ec2セキュリティグループNotFound

クラスターのクラスターセキュリティグループが見つかりませんでした。クラスターを再作成する必要があります。

## Ec2セキュリティグループDeletionFailure

マネージド型ノードグループのリモートアクセスセキュリティグループを削除できませんでした。セキュリティグループから依存関係を削除します。

## Ec2LaunchTemplateNotFound

マネージド型ノードグループの Amazon EC2 起動テンプレートが見つかりませんでした。復旧にはノードグループを再作成する必要があります。

## Ec2LaunchTemplateVersionMismatch

マネージド型ノードグループの Amazon EC2 起動テンプレートのバージョンが、Amazon EKS が作成したバージョンと一致しません。Amazon EKS が作成したバージョンに戻すことで復旧できる場合があります。

## IamInstanceProfileNotFound

マネージド型ノードグループの IAM インスタンスプロファイルが見つかりませんでした。同じ設定でインスタンスプロファイルを再作成して復旧できる場合があります。

## IamNodeRoleNotFound

マネージド型ノードグループの IAM 役割が見つかりませんでした。同じ設定で IAM 役割を再作成して復旧できる場合があります。

## AsgInstanceLaunchFailures

インスタンスの起動中に 自動スケーリング グループに障害が発生しています。

## NodeCreationFailure

起動したインスタンスを Amazon EKS クラスターに登録できません。この障害の一般的な原因は [ノード IAM 役割](#) のアクセス許可が不十分か、ノードのアウトバウンドインターネットアクセスがないことです。ノードは以下の要件のいずれかを満たしている必要があります。

- パブリック IP アドレスを使用してインターネットにアクセス可能です。ノードが存在するサブネットに関連付けられているセキュリティグループが、通信を許可する必要があります。詳細については [the section called “サブネットの要件と考慮事項”](#) および [the section called “セキュリティグループの要件”](#) を参照してください。

- ノードと VPC は「[インターネットアクセスが制限されたプライベートクラスターをデプロイする](#)」に記載される要件を満たしている必要があります。

#### InstanceLimitExceeded

AWS アカウントが、指定されたインスタンスタイプのインスタンスをこれ以上起動できません。Amazon EC2 のインスタンス制限の引き上げをリクエストして復旧できる場合があります。

#### InsufficientFreeAddresses

マネージド型ノードグループに関連付けられている 1 つ以上のサブネットに、新しいノードに使用できる十分な IP アドレスがありません。

#### InternalFailure

これらのエラーは通常、Amazon EKS サーバー側の問題が原因で発生します。

## AccessDenied マネージド型ノードグループエラーを修正する

マネージド型ノードグループで操作を実行する際に AccessDenied エラーが発生する最も一般的な原因は `eks:node-manager`、`ClusterRole` または `ClusterRoleBinding` がないことです。Amazon EKS はマネージド型ノードグループでのオンボーディングの一環として、クラスター内にこれらのリソースを設定します。これらのリソースはノードグループの管理に必要です。

`ClusterRole` は時間の経過とともに変化する可能性があります、次の例のようになります。

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
 name: eks:node-manager
rules:
- apiGroups:
 - ''
 resources:
 - pods
 verbs:
 - get
 - list
 - watch
 - delete
- apiGroups:
 - ''
```

```
resources:
- nodes
verbs:
- get
- list
- watch
- patch
- apiGroups:
 - ''
resources:
- pods/eviction
verbs:
- create
```

ClusterRoleBinding は時間の経過とともに変化する可能性があります、次の例のようになります。

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
 name: eks:node-manager
roleRef:
 apiGroup: rbac.authorization.k8s.io
 kind: ClusterRole
 name: eks:node-manager
subjects:
- apiGroup: rbac.authorization.k8s.io
 kind: User
 name: eks:node-manager
```

eks:node-manager ClusterRole が存在することを確認します。

```
kubectl describe clusterrole eks:node-manager
```

存在する場合は出力を以前の ClusterRole の例と比較します。

eks:node-manager ClusterRoleBinding が存在することを確認します。

```
kubectl describe clusterrolebinding eks:node-manager
```

存在する場合は出力を以前の ClusterRoleBinding の例と比較します。

マネージド型ノードグループの操作のリクエスト中に AccessDenied エラーの原因として ClusterRole や ClusterRoleBinding が紛失または破損していることを発見した場合はそれらを復元できます。*eks-node-manager-role.yaml* という名前のファイルに、以下の内容を保存します。

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
 name: eks:node-manager
rules:
- apiGroups:
 - ''
 resources:
 - pods
 verbs:
 - get
 - list
 - watch
 - delete
- apiGroups:
 - ''
 resources:
 - nodes
 verbs:
 - get
 - list
 - watch
 - patch
- apiGroups:
 - ''
 resources:
 - pods/eviction
 verbs:
 - create

apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
 name: eks:node-manager
roleRef:
 apiGroup: rbac.authorization.k8s.io
 kind: ClusterRole
 name: eks:node-manager
```

```
subjects:
- apiGroup: rbac.authorization.k8s.io
 kind: User
 name: eks:node-manager
```

ファイルを適用します。

```
kubectl apply -f eks-node-manager-role.yaml
```

ノードグループの操作を再試行して、問題が解決したかどうかを確認します。

## Not authorized for images

Not authorized for images エラーメッセージの考えられる原因の 1 つは、プライベート Amazon EKS Windows AMI を使用して Windows マネージドノードグループを起動することです。新しい Windows AMI をリリースすると、AWS は 4 か月以上前の AMI を非公開にし、アクセス不可にします。マネージドノードグループがプライベート Windows AMI を使用している場合は、[Windows マネージドノードグループを更新する](#)ことを検討してください。非公開になった AMI へのアクセスを可能にできるかは保証できませんが、AWS サポートにチケットを提出することでアクセスをリクエストできます。詳細については「Amazon EC2 ユーザーガイド」の「[パッチ](#)」を参照してください。

## ノードが NotReady 状態です

ノードが NotReady 状態になった場合は、そのノードに異常があり、新しい Pod をスケジューリングできない可能性があります。これはノードに CPU、メモリ、利用可能なディスクスペースの十分なリソースがないなど、さまざまな理由で発生します。

Amazon EKS 最適化 Windows AMI の場合、kubelet 設定のデフォルトではコンピューティングリソースの予約は指定されていません。リソースの問題を防ぐために、kubelet に [kube-reserved](#) および/または [system-reserved](#) の設定値を指定することで、システムプロセスのコンピューティングリソースを予約できます。これを行うにはブートストラップスクリプトの `-KubeletExtraArgs` コマンドラインパラメータを使用します。詳細については、Kubernetes ドキュメントの「[Reserve Compute Resources for System Daemons](#)」、およびこのユーザーガイドの「[the section called “ブートストラップスクリプトの設定パラメータ”](#)」を参照してください。

# EKS ログコレクター

Amazon EKS ノードの問題をトラブルシューティングするために、ノード上の `/etc/eks/log-collector-script/eks-log-collector.sh` に構築済みのスクリプトが用意されています。このスクリプトを使用して、サポートケースおよび一般的なトラブルシューティングの診断ログを収集できます。

ノードでスクリプトを実行するには次のコマンドを使用します。

```
sudo bash /etc/eks/log-collector-script/eks-log-collector.sh
```

## Note

その場所にスクリプトがない場合は、手動でスクリプトをダウンロードして実行するには次のコマンドを使用します。

```
curl -O https://amazon-eks.s3.amazonaws.com/support/log-collector-script/linux/eks-log-collector.sh
sudo bash eks-log-collector.sh
```

このスクリプトは次の診断情報を収集します。

```
$ sudo bash /etc/eks/log-collector-script/eks-log-collector.sh
```

```
 This is version 0.7.8. New versions can be found at https://github.com/aws-labs/
amazon-eks-ami/blob/main/log-collector-script/
```

```
Trying to collect common operating system logs...
```

```
Trying to collect kernel logs...
```

```
Trying to collect mount points and volume information...
```

```
...
```

```
...
```

```
Done... your bundled logs are located in /var/log/eks_i-EXAMPLE_2025-03-25_0000-
UTC_0.7.8.tar.gz
```

診断情報が収集され、以下に保存されます。



```
/var/log/eks_i-EXAMPLE_2025-03-25_0000-UTC_0.7.8.tar.gz
```

Bottlerocket ノードのログバンドルを取得するには、「[Bottlerocket Log](#)」で詳細を参照してください。

## コンテナランタイムネットワークの準備ができていません

以下のような Container runtime network not ready エラーと承認エラーが表示される場合があります。

```
4191 kubelet.go:2130] Container runtime network not ready: NetworkReady=false
 reason:NetworkPluginNotReady message:docker: network plugin is not ready: cni config
 uninitialized
4191 reflector.go:205] k8s.io/kubernetes/pkg/kubelet/kubelet.go:452: Failed to list
 *v1.Service: Unauthorized
4191 kubelet_node_status.go:106] Unable to register node
 "ip-10-40-175-122.ec2.internal" with API server: Unauthorized
4191 reflector.go:205] k8s.io/kubernetes/pkg/kubelet/kubelet.go:452: Failed to list
 *v1.Service: Unauthorized
```

これは次のいずれかの理由で発生します。

1. クラスターに aws-auth ConfigMap がないか、ノードに設定した IAM 役割のエントリが含まれていないかのどちらかです。

この問題を解決するには次のコマンドの ##### をクラスターの名前に置き換えて、変更したコマンド `eksctl get iamidentitymapping --cluster my-cluster` を実行することで、ConfigMap 内の現在のエントリを確認します。コマンドからエラーメッセージが表示される場合はクラスターに aws-auth ConfigMap がないことが原因である可能性があります。次のコマンドは ConfigMap にエントリを追加します。ConfigMap が存在しない場合はコマンドによって作成されます。`111122223333` を IAM ロールの AWS ID に、`myAmazonEKSTNodeRole` をノードの役割の名前に置き換えます。

```
eksctl create iamidentitymapping --cluster my-cluster \
 --arn arn:aws:iam::111122223333:role/myAmazonEKSTNodeRole --group
 system:bootstrappers,system:nodes \
 --username system:node:{{EC2PrivateDNSName}}
```

指定する役割の ARN には / 以外の [パス](#) を含めることはできません。例えば、役割の名前が development/apps/my-role の場合、役割の ARN を指定するときに my-role に変更する必要があります。ノード IAM 役割 ARN (インスタンスプロファイルの ARN ではない) を指定していることを確認してください。

- セルフマネージド型ノードが、「[EKS アクセスエントリを使用して Kubernetes へのアクセスを IAM ユーザーに許可する](#)」のトピックの前提条件に記載されている最小バージョンのプラットフォームバージョンのクラスターにあるものの、ノードの IAM 役割のエントリが aws-auth (前の項目を参照) ConfigMap に記載されていないか、役割のアクセスエントリが存在しません。この問題を解決するには次のコマンドの ##### をクラスターの名前に置き換えて、変更したコマンド aws eks list-access-entries --cluster-name *my-cluster* を実行することで、現在のアクセスエントリを確認します。次のコマンドはノードの IAM 役割のアクセスエントリを追加します。*111122223333* を IAM ロールの AWS ID に、*myAmazonEKSTNodeRole* をノードの役割の名前に置き換えます。Windows ノードを使用している場合は *EC2\_LINUX* を EC2\_Windows に置き換えてください。ノード IAM 役割 ARN (インスタンスプロファイルの ARN ではない) を指定していることを確認してください。

```
aws eks create-access-entry --cluster-name my-cluster --principal-arn
arn:aws:iam::111122223333:role/myAmazonEKSTNodeRole --type EC2_LINUX
```

## TLS ハンドシェイクタイムアウト

ノードがパブリック API サーバーエンドポイントへの接続を確立できない場合、次のようなエラーが発生する可能性があります。

```
server.go:233] failed to run Kubelet: could not init cloud provider "aws": error
finding instance i-1111f2222f333e44c: "error listing AWS instances: \"RequestError:
send request failed\\ncaused by: Post net/http: TLS handshake timeout\""
```

kubelet プロセスは API サーバーエンドポイントを継続的に再生成およびテストします。このエラーはコントロールプレーンでクラスターのローリング更新を行う手順 (設定の変更やバージョンの更新など) で一時的に発生することもあります。

この問題を解決するにはルートテーブルとセキュリティグループを確認して、ノードからのトラフィックがパブリックエンドポイントに到達できることを確認します。

## 無効なクライアントトークンID

中国 AWS リージョンでクラスターにデプロイされている Pod または DaemonSet 用サービスアカウントの IAM ロールを使用していて、仕様に `AWS_DEFAULT_REGION` 環境変数を設定していない場合、Pod または DaemonSet に次のエラーが表示されることがあります。

```
An error occurred (InvalidClientTokenId) when calling the GetCallerIdentity operation:
The security token included in the request is invalid
```

この問題を解決するには、次の Pod の仕様例に示すように、`AWS_DEFAULT_REGION` 環境変数を Pod または DaemonSet の仕様に追加する必要があります。

```
apiVersion: v1
kind: Pod
metadata:
 name: envar-demo
 labels:
 purpose: demonstrate-envvars
spec:
 containers:
 - name: envar-demo-container
 image: gcr.io/google-samples/node-hello:1.0
 env:
 - name: AWS_DEFAULT_REGION
 value: "region-code"
```

## コントロールプレーンをアップグレードする前に、ノードグループが Kubernetes バージョンと一致する必要があります

クラスター内の、マネージドノードと Fargate ノードのマイナーバージョンは、コントロールプレーンを新しい Kubernetes バージョンにアップグレードする前に、コントロールプレーンの現在のバージョンと同じにする必要があります。Amazon EKS `update-cluster-version` API はすべての EKS マネージド型ノードが現在のクラスターバージョンにアップグレードされるまで、リクエストを拒否します。Amazon EKS はマネージド型ノードをアップグレードするための API を提供します。マネージドノードグループの Kubernetes バージョンのアップグレードについては、[「the section called “更新”」](#)を参照してください。Fargate ノードのバージョンをアップグレードするには、ノードによって表されるポッドを削除し、コントロールプレーンをアップグレードした後に

ポッドを再デプロイします。詳細については、「[the section called “Kubernetes バージョンを更新する”](#)」を参照してください。

## 多数のノードを起動すると、Too Many Requests エラーが発生します。

多数のノードを同時に起動すると、[Amazon EC2 ユーザーデータ](#)の実行ログに「Too Many Requests」というエラーメッセージが表示される場合があります。これはコントロールプレーンが describeCluster 呼び出しで過負荷になっているために発生する可能性があります。過負荷が原因で、スロットリングが発生し、ノードがブートストラップスクリプトを実行できなくなり、ノードが完全にクラスターに参加できなくなります。

--apiserver-endpoint、--b64-cluster-ca、および --dns-cluster-ip 引数がノードのブートストラップスクリプトに渡されていることを確認してください。これらの引数を含める場合、ブートストラップスクリプトが describeCluster 呼び出しを行う必要がなくなり、コントロールプレーンが過負荷になるのを防ぐのに役立ちます。詳細については、「[the section called “Amazon EKS に最適化された Linux/Bottlerocket AMI に含まれる bootstrap.sh ファイルに引数を渡すためのユーザーデータを提供する”](#)」を参照してください。

## Kubernetes API サーバーリクエストに対する HTTP 401 Unauthorized エラーレスポンス

これらのエラーは、Pod のサービスアカウントトークンがクラスターで期限切れになった場合に表示されます。

Amazon EKS クラスターの Kubernetes API サーバーでは、90 日を超えるトークンによるリクエストが拒否されます。以前の Kubernetes バージョンでは、トークンに有効期限はありませんでした。つまり、これらのトークンに依存しているクライアントは、1 時間以内にトークンを更新する必要があります。無効なトークンが原因で Kubernetes API サーバーによりリクエストが拒否されないようにするには、ワークロードで使用されている [Kubernetes クライアント SDK](#) バージョンを、次のバージョンと同じか、それ以降にする必要があります。

- Go - バージョン 0.15.7 以降
- Python バージョン 12.0.0 以降
- Java バージョン 9.0.0 以降
- JavaScript バージョン 0.10.3 以降

- Ruby master ブランチ
- Haskell バージョン 0.3.0.0
- C# バージョン 7.0.5 以降

古いトークンを使用しているクラスター内の既存の Pod をすべて特定できます。詳細については、「[the section called “サービスアカウントトークン”](#)」を参照してください。

## Amazon EKS プラットフォームのバージョンは現在のプラットフォーム バージョンより 2 つ以上遅れています

これは Amazon EKS がクラスターの [プラットフォームバージョン](#) を自動的に更新できない場合に発生する可能性があります。これには多くの原因がありますが、一般的な原因のいくつかが続きます。これらの問題のいずれかがクラスターに当てはまる場合、それはまだ機能する可能性があります、そのプラットフォームバージョンは Amazon EKS によって更新されません。

### 問題

[クラスター IAM 役割](#) が削除されました - この役割はクラスターの作成時に指定されました。次のコマンドで、どの役割が指定されたかを確認できます。##### の部分は自分のクラスター名に置き換えます。

```
aws eks describe-cluster --name my-cluster --query cluster.roleArn --output text | cut -d / -f 2
```

出力例は次のとおりです。

```
eksClusterRole
```

### ソリューション

同じ名前で新しい [クラスター IAM 役割](#) を作成します。

### 問題

クラスターの作成中に指定されたサブネットが削除されました - クラスターで使用するサブネットはクラスターの作成中に指定されました。次のコマンドで、指定されたサブネットを確認できます。##### の部分は自分のクラスター名に置き換えます。

```
aws eks describe-cluster --name my-cluster --query cluster.resourcesVpcConfig.subnetIds
```

出力例は次のとおりです。

```
[
 "subnet-EXAMPLE1",
 "subnet-EXAMPLE2"
]
```

## ソリューション

アカウントにサブネット ID が存在するかどうかを確認します。

```
vpc_id=$(aws eks describe-cluster --name my-cluster --query
 cluster.resourcesVpcConfig.vpcId --output text)
aws ec2 describe-subnets --filters "Name=vpc-id,Values=$vpc_id" --query
 "Subnets[*].SubnetId"
```

出力例は次のとおりです。

```
[
 "subnet-EXAMPLE3",
 "subnet-EXAMPLE4"
]
```

出力で返されたサブネット ID が、クラスターの作成時に指定されたサブネット ID と一致しない場合、Amazon EKS でクラスターを更新するにはクラスターで使用されるサブネットを変更する必要があります。これはクラスターの作成時に 2 つ以上のサブネットを指定した場合、Amazon EKS は指定したサブネットをランダムに選択して新しいエラスティックネットワークインターフェイスを作成するためです。これらのネットワーク インターフェイスにより、コントロールプレーンはノードと通信できます。Amazon EKS は選択したサブネットが存在しない場合、クラスターを更新しません。Amazon EKS が新しいネットワーク インターフェイスを作成するために選択する、クラスターの作成時に指定したサブネットをコントロールすることはできません。

クラスターの Kubernetes バージョンの更新を開始すると、同じ理由で更新が失敗する可能性があります。

## 問題

クラスターの作成中に指定されたセキュリティグループが削除されました - クラスターの作成中にセキュリティグループを指定した場合は次のコマンドでそれらの ID を確認できます。##### の部分は自分のクラスター名に置き換えます。

```
aws eks describe-cluster --name my-cluster --query
cluster.resourcesVpcConfig.securityGroupIds
```

出力例は次のとおりです。

```
[
 "sg-EXAMPLE1"
]
```

[ ] が返された場合、クラスターの作成時にセキュリティグループが指定されておらず、セキュリティグループの欠落は問題ではありません。セキュリティグループが返された場合はセキュリティグループがアカウントに存在することを確認します。

## ソリューション

これらのセキュリティグループがアカウントに存在するかどうかを確認します。

```
vpc_id=$(aws eks describe-cluster --name my-cluster --query
cluster.resourcesVpcConfig.vpcId --output text)
aws ec2 describe-security-groups --filters "Name=vpc-id,Values=$vpc_id" --query
"SecurityGroups[*].GroupId"
```

出力例は次のとおりです。

```
[
 "sg-EXAMPLE2"
]
```

出力で返されたセキュリティグループ ID が、クラスターの作成時に指定されたセキュリティグループ ID と一致しない場合、Amazon EKS でクラスターを更新するにはクラスターで使用されるセキュリティグループを変更する必要があります。クラスターの作成時に指定されたセキュリティグループ ID が存在しない場合、Amazon EKS はクラスターを更新しません。

クラスターの Kubernetes バージョンの更新を開始すると、同じ理由で更新が失敗する可能性があります。



- クラスターの作成時に指定した各サブネットで、少なくとも 6 つ (ただし、16 をお勧めします) の使用可能な IP アドレスがありません。サブネットに十分な使用可能な IP アドレスがない場合はサブネット内の IP アドレスを解放するか、十分な使用可能な IP アドレスを持つサブネットを使用するように、クラスターで使用されるサブネットを変更する必要があります。
- クラスターの作成時に [シークレット暗号化](#) を有効にしましたが、指定した AWS KMS キーは削除されました。Amazon EKS でクラスターを更新する場合は新しいクラスターを作成する必要があります

## クラスターの正常性に関するよくある質問およびエラーコードと解決パス

Amazon EKS は EKS クラスターとクラスターインフラストラクチャの問題を検出し、それらを EKS クラスターリソースのヘルスオブジェクトに保存します。クラスターヘルスの情報を利用することで、クラスターの問題をより迅速に検出、トラブルシューティング、対処できます。これにより、より安全で最新のアプリケーション環境を構築できます。さらに、必要なインフラストラクチャまたはクラスター設定に問題があるため、Kubernetes の新しいバージョンにアップグレードしたり、Amazon EKS が劣化したクラスターにセキュリティ更新をインストールしたりできない場合があります。Amazon EKS は問題を検出したり、問題が解決されたことを検出したりするまでに 3 時間かかる場合があります。

Amazon EKS クラスターの状態は Amazon EKS とそのユーザー間で共有される責任です。IAM 役割や Amazon VPC サブネットの前提となるインフラストラクチャ、および事前に提供が必要があるその他の必要なインフラストラクチャはお客様の責任となります。Amazon EKS はこのインフラストラクチャとクラスターの設定の変更を検出します。

Amazon EKS コンソールからクラスターのヘルスにアクセスするには、Amazon EKS クラスターの詳細ページからオペレーターバビリティのダッシュボードにアクセスし、[クラスターヘルスの問題] タブの中から [ヘルスの問題] と書かれたテーブルを探します。このデータは EKS API の `DescribeCluster` アクションを、例えば AWS コマンドラインインターフェイス内から呼び出すことでも使用できます。

この機能を使用する理由は何ですか？

Amazon EKS クラスターの状態を把握しやすくなり、問題を迅速に診断して修正できます。デバッグや AWS サポートケースの作成に時間を費やす必要はありません。例えば、Amazon EKS クラスターのサブネットを誤って削除した場合、Amazon EKS はクロスアカウントのネットワークインターフェイスや `kubectl exec` や `kubectl ログ` などの Kubernetes AWS CLI コマ



ンドを作成できなくなります。この場合、次のエラーとともに処理が失敗します: Error from server: error dialing backend: remote error: tls: internal error. そして、次のような Amazon EKS ヘルスの問題が表示されます: subnet-da60e280 was deleted: could not create network interface.

この機能は他の AWS サービスとどのように関連し、連携しますか？

IAM 役割と Amazon VPC サブネットはクラスターヘルスが問題を検出するための前提条件となるインフラストラクチャの 2 つの例です。この機能はこれらのリソースが正しく設定されていない場合に詳細情報を返します。

ヘルスに問題があるクラスターには料金がかかりますか？

はい。Amazon EKS クラスターはすべて標準の Amazon EKS 料金で請求されます。クラスターヘルス機能は追加料金なしで利用できます。

この機能は AWS アウトポスト の Amazon EKS クラスターで動作しますか？

はい。クラスターの問題は AWS アウトポスト の拡張クラスターと AWS アウトポスト のローカルクラスターを含む AWS Cloud 内の EKS クラスターで検出されます。クラスターヘルスでは、Amazon EKS Anywhere や Amazon EKS Distro (EKS-D) の問題は検出されません。

新しい問題が検出されたときに通知を受けることはできますか？

はい。AWS は新しいクラスターのヘルスの問題が検出されると、E メールと Personal Health Dashboard の通知を送信します。

コンソールにはヘルスの問題に関する警告が表示されますか？

はい。ヘルスに問題があるクラスターにはコンソールの上部にバナーが表示されます。

最初の 2 つの列は API レスポンス値に必要なものです。[Health ClusterIssue](#) オブジェクトの 3 番目のフィールドは resourceIds で、返される値は課題タイプによって異なります。

| Code             | メッセージ                                               | ResourceIds | クラスターは回復可能ですか？ |
|------------------|-----------------------------------------------------|-------------|----------------|
| SUBNET_NOT_FOUND | クラスターに現在関連付けられている 1 つ以上のサブネットが見つかりませんでした。Amazon EKS | サブネット ID    | はい             |

| Code                     | メッセージ                                                                                                                                                         | ResourceIds   | クラスターは回復可能ですか？ |
|--------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------|---------------|----------------|
|                          | update-cluster-config API を呼び出して、サブネットを更新します。                                                                                                                 |               |                |
| SECURITY_GROUP_NOT_FOUND | クラスターに現在関連付けられている 1 つ以上のセキュリティグループが見つかりませんでした。Amazon EKS update-cluster-config API を呼び出して、セキュリティグループを更新します                                                    | セキュリティグループ ID | はい             |
| IP_NOT_AVAILABLE         | クラスターに関連付けられている 1 つ以上のサブネットに、Amazon EKS がクラスター管理操作を実行するための十分な IP アドレスがありません。サブネット内のアドレスを解放するか、Amazon EKS update-cluster-config API を使用して別のサブネットをクラスターに関連付けます。 | サブネット ID      | はい             |

| Code                      | メッセージ                                                                                                                                     | ResourceIds  | クラスターは回復可能ですか？ |
|---------------------------|-------------------------------------------------------------------------------------------------------------------------------------------|--------------|----------------|
| VPC_NOT_FOUND             | クラスターに関連付けられている VPC が見つかりませんでした。クラスターを削除して再作成する必要があります。                                                                                   | VPC ID       | いいえ            |
| ASSUME_ROLE_ACCESS_DENIED | クラスターは Amazon EKS サービスにリンクされた役割を使用していません。クラスターに関連する役割を引き受けて、必要な Amazon EKS 管理オペレーションを実行することができませんでした。役割が存在し、必要な信頼ポリシーが適用されていることを確認してください。 | クラスター IAM 役割 | はい             |

| Code                                | メッセージ                                                                                                                                                                | ResourceIds              | クラスターは回復可能ですか？ |
|-------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------|--------------------------|----------------|
| PERMISSION_ACCESS_DENIED            | クラスターは Amazon EKS サービスにリンクされた役割を使用していません。クラスターに関連付けられている役割では、Amazon EKS に必要な管理操作を実行するための十分なアクセス許可が付与されていません。クラスター役割にアタッチされているポリシーを確認し、別の拒否ポリシーが適用されているかどうかを確認してください。 | クラスター IAM 役割             | はい             |
| ASSUME_ROLE_ACCESS_DENIED_USING_SLR | Amazon EKS クラスター管理サービスにリンクされた役割を引き受けることができませんでした。役割が存在し、必要な信頼ポリシーが適用されていることを確認してください。                                                                                | Amazon EKS サービスにリンクされた役割 | はい             |

| Code                               | メッセージ                                                                                                                                 | ResourceIds              | クラスターは回復可能ですか？ |
|------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------|--------------------------|----------------|
| PERMISSION_ACCESS_DENIED_USING_SLR | Amazon EKS クラスター管理サービスにリンクされた役割は Amazon EKS に必要な管理操作を実行するための十分なアクセス許可を付与しません。クラスター役割にアタッチされているポリシーを確認し、別の拒否ポリシーが適用されているかどうかを確認してください。 | Amazon EKS サービスにリンクされた役割 | はい             |
| OPT_IN_REQUIRED                    | アカウントに Amazon EC2 サービスのサブスクリプションがありません。アカウント設定ページでアカウントサブスクリプションを更新してください。                                                            | 該当なし                     | はい             |
| STS_REGIONAL_ENDPOINT_DISABLED     | STS リージョナルエンドポイントは無効になっています。Amazon EKS のエンドポイントで必要なクラスター管理操作を実行できるようにします。                                                             | 該当なし                     | はい             |

| Code              | メッセージ                                                         | ResourceIds | クラスターは回復可能ですか？ |
|-------------------|---------------------------------------------------------------|-------------|----------------|
| KMS_KEY_DISABLED  | クラスターに関連付けられている AWS KMS キーは無効になっています。クラスターを復元するにはキーを再度有効にします。 | KMS Key Arn | はい             |
| KMS_KEY_NOT_FOUND | クラスターに関連付けられている AWS KMS キーが見つかりませんでした。クラスターを削除して再作成する必要があります。 | KMS Key ARN | いいえ            |
| KMS_GRANT_REVOKED | クラスターに関連付けられた AWS KMS キーの付与は取り消されます。クラスターを削除して再作成する必要があります。   | KMS Key Arn | いいえ            |

# Amazon EKS Connector を使用して Kubernetes クラスターを Amazon EKS マネジメントコンソールに接続する

Amazon EKS Connector を使用して、準拠する Kubernetes クラスターを AWS に登録および接続し、Amazon EKS コンソールで視覚化できます。クラスターが接続されると、Amazon EKS コンソールでクラスターのステータス、設定、およびワークロードを確認できます。この機能を使用して Amazon EKS コンソールで接続されたクラスターを表示できますが、管理することはできません。Amazon EKS Connector には、[Github のオープンソースプロジェクト](#)であるエージェントが必要です。よくある質問やトラブルシューティングなどのその他の技術コンテンツについては、「[the section called “EKS Connector をトラブルシューティングする”](#)」を参照してください。

Amazon EKS Connector は、以下のタイプの Kubernetes クラスターを Amazon EKS に接続できます。

- オンプレミス Kubernetes クラスター
- Amazon EC2 で実行されるセルフマネージド型クラスター
- 他のクラウドプロバイダーのマネージド型クラスター

## Amazon EKS Connector の考慮事項

Amazon EKS Connector を使用する前に、次のことを理解してください。

- Amazon EKS にクラスターを接続するためには、Kubernetes クラスターに対する管理者権限が必要です。
- Kubernetes クラスターには、接続する前に Linux 64 ビット (x86) ワーカーノードが存在している必要があります。ARM ワーカーノードはサポートされていません。
- Kubernetes クラスターには、ssm、および ssmmessages、Systems Manager エンドポイントへのアウトバウンドアクセスを持つワーカーノードが必要です。詳細については、AWS 全般のリファレンスの「[Systems Manager エンドポイント](#)」を参照してください。
- デフォルトでは、1 つのリージョンで最大 10 個のクラスターを接続できます。[Service Quotas コンソール](#)を使用して引き上げをリクエストできます。詳細については、「[クォータの引き上げのリクエスト](#)」を参照してください。
- Amazon EKS の `RegisterCluster`、`ListClusters`、`DescribeCluster`、`DeregisterClusterAPI` は外部 Kubernetes クラスターでサポートされています。

- クラスターを登録するには、次のアクセス許可が必要です。
  - eks:RegisterCluster
  - ssm:CreateActivation
  - ssm>DeleteActivation
  - iam:PassRole
- クラスターを登録解除するには、次のアクセス許可が必要です。
  - eks:DeregisterCluster
  - ssm>DeleteActivation
  - SSM: DeregisterManagedInstance

## Amazon EKS コネクターの必要な IAM ロール

Amazon EKS Connector を使用するには、次の 2 つの IAM ロールが必要です。

- 初めてクラスターを登録すると、[Amazon EKS Connector](#) サービスにリンクされたロールが作成されます。
- Amazon EKS Connector エージェントの IAM ロールを作成する必要があります。詳細については、「[the section called “コネクタ IAM ロール”](#)」を参照してください。

[IAM プリンシパル](#)のクラスターおよびワークロードの表示許可を有効にするには、eks-connector および Amazon EKS Connector クラスターのロールをクラスターに適用します。「[Amazon EKS コンソールで Kubernetes クラスターのリソースを表示するためのアクセスを付与する](#)」の手順を行います。

## 外部 Kubernetes クラスターを Amazon EKS マネジメントコンソールに接続する

以下のプロセスで複数のメソッドを使用して、外部の Kubernetes クラスターを Amazon EKS に接続できます。このプロセスには 2 つのステップがあります。Amazon EKS へのクラスターの登録と、クラスターへの eks-connector エージェントのインストールです。



**⚠ Important**

最初のステップを完了してから 3 日以内、つまり登録の有効期限が切れる前に 2 番目のステップを完了する必要があります。

## 考慮事項

エージェントをインストールする際には YAML マニフェストを使用できます。または、AWS Management Console または AWS コマンドラインインターフェイスでクラスターを登録する場合は、Helm を使用できます。ただし、eksctl でクラスターを登録する場合は、Helm を使用してエージェントをインストールすることはできません。

## 前提条件

- Amazon EKS Connector エージェントロールが作成されたことを確認します。[Amazon EKS コネクタエージェントロールの作成](#)のステップに従います。
- クラスターを登録するには、次のアクセス許可が必要です。
  - eks:RegisterCluster
  - ssm:CreateActivation
  - ssm>DeleteActivation
  - iam:PassRole

## ステップ 1: クラスターの登録

クラスターを Amazon EKS コネクタに登録するには、次のいずれかのツールを使用できます。

- [the section called “AWS CLI”](#)
- [the section called “AWS Management Console”](#)
- [the section called “eksctl”](#)

### AWS CLI

1. AWS CLI がインストールされている必要があります。インストールまたはアップグレードするには、「[AWS CLI のインストール](#)」を参照してください。

2. コネクタ設定には、Amazon EKS Connector エージェントの IAM ロールを指定します。詳細については、「[the section called “Amazon EKS コネクターの必要な IAM ロール”](#)」を参照してください。

```
aws eks register-cluster \
 --name my-first-registered-cluster \
 --connector-config roleArn=arn:aws:iam::111122223333:role/
AmazonEKSCoordinatorAgentRole,provider="OTHER" \
 --region aws-region
```

出力例は次のとおりです。

```
{
 "cluster": {
 "name": "my-first-registered-cluster",
 "arn": "arn:aws:eks:region:111122223333:cluster/my-first-registered-cluster",
 "createdAt": 1627669203.531,
 "ConnectorConfig": {
 "activationId": "xxxxxxxxACTIVATION_IDxxxxxxxx",
 "activationCode": "xxxxxxxxACTIVATION_CODExxxxxxxx",
 "activationExpiry": 1627672543.0,
 "provider": "OTHER",
 "roleArn": "arn:aws:iam::111122223333:role/AmazonEKSCoordinatorAgentRole"
 },
 "status": "CREATING"
 }
}
```

次のステップで aws-region、activationId、activationCode の値を使用します。

## AWS Management Console

1. [アマゾン EKS コンソール](#)を開きます。
2. [Add cluster] (クラスターの追加) を選択し、[Register] (登録) を選択して設定ページを表示します。
3. [Configure cluster (クラスターの設定)] セクションで、次のフィールドに入力します。
  - [Name (名前)] – クラスターの一意的な名前。
  - [Provider] (プロバイダー) 選択すると、Kubernetes クラスタープロバイダーのドロップダウンリストが表示されます。特定のプロバイダが不明な場合は、[その他] を選択します。

- EKS Connector のロール — クラスターの接続に使用するロールを選択します。
4. クラスターの登録を選択します。
  5. [Cluster] 概要ページが表示されます。Helm チャートを使用する場合は、`helm install` コマンドをコピーして次のステップに進みます。YAML マニフェストを使用する場合は、[YAML ファイルをダウンロード] を選択して、マニフェストファイルをローカルドライブにダウンロードします。

#### Important

これは、`helm install` コマンドをコピーする、またはこのファイルをダウンロードする唯一の機会です。このページから移動しないでください。リンクにアクセスできないので、クラスターの登録を解除し、最初から手順を開始する必要があります。

コマンドまたはマニフェストファイルは、登録されたクラスターに対して一度だけ使用できません。Kubernetes クラスターからリソースを削除する場合は、クラスターを再登録し、新しいマニフェストファイルを取得する必要があります。

次のステップに進み、Kubernetes クラスターにマニフェストファイルを適用します。

## eksctl

1. `eksctl` バージョン 0.68 以降がインストールされている必要があります。インストール、またはアップグレードをする場合は「[the section called “クラスターを作成する \(eksctl\)”](#)」を参照してください。
2. 名前、プロバイダー、およびリージョンを指定して、クラスターを登録します。

```
eksctl register cluster --name my-cluster --provider my-provider --region region-code
```

出力例:

```
2021-08-19 13:47:26 [#] creating IAM role "eksctl-20210819194112186040"
2021-08-19 13:47:26 [#] registered cluster "<name>" successfully
2021-08-19 13:47:26 [#] wrote file eks-connector.yaml to <current directory>
2021-08-19 13:47:26 [#] wrote file eks-connector-clusterrole.yaml to <current directory>
```

```
2021-08-19 13:47:26 [#] wrote file eks-connector-console-dashboard-full-access-group.yaml to <current directory>
2021-08-19 13:47:26 [!] note: "eks-connector-clusterrole.yaml" and "eks-connector-console-dashboard-full-access-group.yaml" give full EKS Console access to IAM identity "<aws-arn>", edit if required; read https://eksctl.io/usage/eks-connector for more info
2021-08-19 13:47:26 [#] run `kubectl apply -f eks-connector.yaml,eks-connector-clusterrole.yaml,eks-connector-console-dashboard-full-access-group.yaml` before expiry> to connect the cluster
```

これにより、ローカルコンピュータ上にファイルが作成されます。これらのファイルは 3 日以内に外部クラスターに適用する必要があります。そうしないと登録の有効期限が切れます。

3. クラスターにアクセスできるターミナルで、eks-connector-binding.yaml ファイルを適用します。

```
kubectl apply -f eks-connector-binding.yaml
```

## ステップ 2: eks-connector エージェントのインストール

eks-connector エージェントをインストールするには、次のいずれかのツールを使用します。

- [the section called “Helm”](#)
- [the section called “yaml”](#)

### Helm

#### Note

eksctl でクラスターを登録した場合は、Helm チャートの方法の代わりに YAML マニフェストの方法を使用します。

1. 前のステップで AWS CLI を使用した場合は、次のコマンドで ACTIVATION\_CODE と ACTIVATION\_ID をそれぞれ activationId と activationCode の値に置き換えます。aws-region を前のステップで使用した AWS リージョンに置き換えます。その後、次のコマンドを実行して、eks-connector エージェントを登録クラスターにインストールします。

```
$ helm install eks-connector \
 --namespace eks-connector \
 oci://public.ecr.aws/eks-connector/eks-connector-chart \
 --set eks.activationCode=ACTIVATION_CODE \
 --set eks.activationId=ACTIVATION_ID \
 --set eks.agentRegion=aws-region
```

前のステップで AWS Management Console を使用した場合は、前のステップからコピーした、以下の値が入力されたコマンドを使用します。

2. インストールされている `eks-connector` デプロイの正常性をチェックし、Amazon EKS に登録されたクラスターのステータスが `ACTIVE` になるのを待ちます。

## yaml

Kubernetes クラスターに Amazon EKS コネクタマニフェストファイルを適用して、接続を完了します。これを行うには、前述のメソッドを使用する必要があります。マニフェストが 3 日以内に適用されない場合、Amazon EKS Connector の登録の有効期限が切れます。クラスター接続が期限切れになった場合は、クラスターを再度接続する前にクラスターの登録を解除する必要があります。

1. Amazon EKS コネクタ YAML ファイルをダウンロードします。

```
curl -O https://amazon-eks.s3.us-west-2.amazonaws.com/eks-connector/manifests/eks-connector/latest/eks-connector.yaml
```

2. Amazon EKS Connector の YAML ファイルを編集して、`%AWS_REGION%`、`%EKS_ACTIVATION_ID%`、`%EKS_ACTIVATION_CODE%` のすべてのリファレンスを、前のステップの出力から得られた `aws-region`、`activationId`、`activationCode` に置き換えます。

次のコマンド例では、これらの値を置き換えることができます。

```
sed -i "s~%AWS_REGION%~$aws-region~g; s~%EKS_ACTIVATION_ID%~$EKS_ACTIVATION_ID~g; s~%EKS_ACTIVATION_CODE%~$(echo -n $EKS_ACTIVATION_CODE | base64)~g" eks-connector.yaml
```

### Important

アクティベーションコードが base64 形式であることを確認します。

3. クラスターにアクセスできるターミナルで、次のコマンドを実行して、更新されたマニフェストファイルを適用できます。

```
kubectl apply -f eks-connector.yaml
```

4. Amazon EKS Connector のマニフェストとロールバインディング YAML ファイルが Kubernetes クラスターに適用されたら、クラスターが接続されたことを確認します。

```
aws eks describe-cluster \
 --name "my-first-registered-cluster" \
 --region AWS_REGION
```

出力には status=ACTIVE が含まれている必要があります。

5. (オプション) クラスターにタグを追加します。詳細については、「[the section called “リソースのタグ付け”](#)」を参照してください。

## 次のステップ

これらのステップで問題が発生した場合は、「[the section called “EKS Connector をトラブルシューティングする”](#)」を参照してください。

追加の [IAM プリンシパル](#) に Amazon EKS コンソールへのアクセスを許可して、接続されているクラスター内の Kubernetes リソースを表示するには、「[the section called “クラスターに対するアクセスを許可する”](#)」を参照してください。

## Amazon EKS コンソールで Kubernetes クラスターのリソースを表示するためのアクセスを付与する

[IAM プリンシパル](#) に Amazon EKS コンソールへのアクセスを許可して、接続されたクラスターで実行されている Kubernetes リソースに関する情報を表示します。

### 前提条件

AWS Management Console にアクセスするために使用する [IAM プリンシパル](#) は、次の要件を満たしている必要があります。

- eks:AccessKubernetesApi IAM アクセス許可が必要です。

- Amazon EKS Connector サービスアカウントが、クラスター内の IAM プリンシパルを偽装できる。これにより、Amazon EKS Connector は IAM プリンシパルを Kubernetes ユーザーにマッピングできます。

Amazon EKS Connector クラスターロールを作成して適用するには

1. eks-connector クラスターのロールテンプレートをダウンロードします。

```
curl -O https://s3.us-west-2.amazonaws.com/amazon-eks/eks-connector/manifests/eks-connector-console-roles/eks-connector-clusterrole.yaml
```

2. クラスターロールテンプレート YAML ファイルを編集します。%IAM\_ARN% のリファレンスを IAM プリンシパルの Amazon リソースネーム (ARN) に置き換えます。
3. Amazon EKS コネクタクラスターロール YAML を Kubernetes クラスターに適用します。

```
kubectl apply -f eks-connector-clusterrole.yaml
```

IAM プリンシパルが Amazon EKS コンソールで Kubernetes リソースを確認するには、そのプリンシパルが、リソースを読み取るために必要な許可を持つ Kubernetes role または clusterrole に関連付けられている必要があります。詳細については Kubernetes ドキュメントの「[RBAC 認可使用する](#)」を参照してください。

接続されたクラスターにアクセスするように IAM プリンシパルを設定するには

1. 以下のいずれかのサンプルマニフェストファイルをダウンロードして、それぞれ clusterrole および clusterrolebinding、または role および rolebinding を作成できます。

すべての名前空間の Kubernetes リソースを表示する

- クラスターのロール eks-connector-console-dashboard-full-access-clusterrole は、コンソールで視覚化できるすべての名前空間とリソースへのアクセスを提供します。クラスターに適用する前に role、clusterrole、および対応するバインディングの名前を変更することができます。次のコマンドを使用して、サンプルファイルをダウンロードします。

```
curl -O https://s3.us-west-2.amazonaws.com/amazon-eks/eks-connector/manifests/eks-connector-console-roles/eks-connector-console-dashboard-full-access-group.yaml
```

## 特定の名前空間の Kubernetes リソースを表示する

- このファイルの名前空間は default です。別の名前空間を指定する場合は、クラスターに適用する前にファイルを編集します。次のコマンドを使用して、サンプルファイルをダウンロードします。

```
curl -O https://s3.us-west-2.amazonaws.com/amazon-eks/eks-connector/manifests/eks-connector-console-roles/eks-connector-console-dashboard-restricted-access-group.yaml
```

2. フルアクセスまたは制限付きアクセスの YAML ファイルを編集して、%IAM\_ARN% のリファレンスを IAM プリンシパルの Amazon リソースネーム (ARN) に置き換えます。
3. フルアクセスまたは制限付きアクセスの YAML ファイルを Kubernetes クラスターに適用します。YAML ファイルの値は独自の値に置き換えます。

```
kubectl apply -f eks-connector-console-dashboard-full-access-group.yaml
```

接続されたクラスター内の Kubernetes リソースを表示するには、「[the section called “クラスターリソースにアクセスする”](#)」を参照してください。[リソース] タブ上の一部のリソースタイプのデータは、接続されたクラスターでは使用できません。

## Amazon EKS コンソールから Kubernetes クラスターの登録を解除する

接続されたクラスターの使用が終了したら、登録を解除できます。登録解除されると、クラスターは Amazon EKS コンソールに表示されなくなります。

deregisterCluster API を呼び出すには、次のアクセス許可が必要です。

- eks:DeregisterCluster
- ssm:DeleteActivation
- ssm:DeregisterManagedInstance

このプロセスには 2 つのステップがあります。Amazon EKS でのクラスターの登録解除と、クラスター内の eks-connector エージェントのアンインストールです。



## Kubernetes クラスターの登録を解除する

Amazon EKS コネクタからクラスターの登録を解除するには、次のいずれかのツールを使用できます。

- [the section called “AWS CLI”](#)
- [the section called “AWS Management Console”](#)
- [the section called “eksctl”](#)

### AWS CLI

1. AWS CLI がインストールされている必要があります。インストールまたはアップグレードするには、「[AWS CLI のインストール](#)」を参照してください。
2. Amazon EKS Connector エージェントロールが作成されたことを確認します。
3. 接続されているクラスターの登録を解除します。

```
aws eks deregister-cluster \
 --name my-cluster \
 --region region-code
```

### AWS Management Console

1. [アマゾン EKS コンソール](#)を開きます。
2. [クラスター] を選択します。
3. リポジトリの[クラスター]ページで、接続されているクラスターを選択し、[登録解除]を選択します。
4. クラスターの登録を解除することを確認します。

### eksctl

1. eksctl バージョン 0.68 以降をインストールします。インストール、またはアップグレードをする場合は「[the section called “クラスターを作成する \(eksctl\)”](#)」を参照してください。
2. Amazon EKS Connector エージェントロールが作成されたことを確認します。
3. 接続されているクラスターの登録を解除します。

```
eksctl deregister cluster --name my-cluster
```

## Kubernetes クラスターのリソースをクリーンアップする

eks-connector エージェントをアンインストールするには、次のいずれかのツールを使用します。

- [the section called “helm”](#)
- [the section called “yaml”](#)

### helm

次のコマンドを実行して、エージェントをアンインストールします。

```
helm -n eks-connector uninstall eks-connector
```

### yaml

1. Kubernetes クラスターから Amazon EKS コネクタ YAML ファイルを削除します。

```
kubectl delete -f eks-connector.yaml
```

2. 追加の [IAM プリンシパル](#) がクラスターにアクセスするために clusterrole または clusterrolebindings を作成した場合は、それらを Kubernetes クラスターから削除します。

## Amazon EKS Connector の問題をトラブルシューティングする

このトピックでは、Amazon EKS Connector の使用中に発生する可能性があるいくつかの一般的なエラーについて説明します。これには、解決方法および回避策も含まれています。

### 基本的なトラブルシューティング

このセクションでは、Amazon EKS Connector の問題を診断する手順について説明します。

### Amazon EKS Connector の状態を確認する

Amazon EKS Connector の状態を確認するには、次のように入力します。

```
kubectl get pods -n eks-connector
```

## Amazon EKS Connector のログを検査します

Amazon EKS Connector のポッドは 3 つのコンテナで構成されています。これらすべてのコンテナの完全なログを取得して検査できるようにするには、次のコマンドを実行します。

- connector-init

```
kubectl logs eks-connector-0 --container connector-init -n eks-connector
kubectl logs eks-connector-1 --container connector-init -n eks-connector
```

- connector-proxy

```
kubectl logs eks-connector-0 --container connector-proxy -n eks-connector
kubectl logs eks-connector-1 --container connector-proxy -n eks-connector
```

- connector-agent

```
kubectl exec eks-connector-0 --container connector-agent -n eks-connector -- cat /
var/log/amazon/ssm/amazon-ssm-agent.log
kubectl exec eks-connector-1 --container connector-agent -n eks-connector -- cat /
var/log/amazon/ssm/amazon-ssm-agent.log
```

## 有効なクラスター名を取得する

Amazon EKS クラスターは、単一の AWS アカウントおよび AWS リージョン内で `clusterName` により一意に識別されます。Amazon EKS に接続されたクラスターが複数ある場合、現在の Kubernetes クラスターが登録されている Amazon EKS クラスターを確認できます。これを行うには、次のコマンドを入力して、現在のクラスターの `clusterName` を確認します。

```
kubectl exec eks-connector-0 --container connector-agent -n eks-connector \
-- cat /var/log/amazon/ssm/amazon-ssm-agent.log | grep -m1 -oE "eks_c:[a-zA-Z0-9_-]+"
| sed -E "s/^.*eks_c:([a-zA-Z0-9_-]+)[a-zA-Z0-9]+.*$/\1/"
kubectl exec eks-connector-1 --container connector-agent -n eks-connector \
-- cat /var/log/amazon/ssm/amazon-ssm-agent.log | grep -m1 -oE "eks_c:[a-zA-Z0-9_-]+"
| sed -E "s/^.*eks_c:([a-zA-Z0-9_-]+)[a-zA-Z0-9]+.*$/\1/"
```

## その他のコマンド

次のコマンドは、問題のトラブルシューティングに必要な情報の取得に役立ちます。

- Amazon EKS Connector の Pod で使用されるイメージを収集するには、次のコマンドを使用します。

```
kubectl get pods -n eks-connector -o jsonpath="{.items[*].spec.containers[*].image}"
| tr -s '[:space:]' '\n'
```

- Amazon EKS Connector が実行されているノード名を確認するには、次のコマンドを使用します。

```
kubectl get pods -n eks-connector -o jsonpath="{.items[*].spec.nodeName}" | tr -s
'[:space:]' '\n'
```

- Kubernetes クライアントおよびサーバーのバージョンを取得するには、次のコマンドを実行します。

```
kubectl version
```

- ノードに関する情報を取得するには、次のコマンドを実行します。

```
kubectl get nodes -o wide --show-labels
```

## Helm の問題: 403 Forbidden

Helm インストールコマンドの実行中に以下のエラーが表示された場合:

```
Error: INSTALLATION FAILED: unexpected status from HEAD request to https://
public.ecr.aws/v2/eks-connector/eks-connector-chart/manifests/0.0.6: 403 Forbidden
```

以下の行を実行して修正できます。

```
docker logout public.ecr.aws
```

## コンソールのエラー: クラスターが Pending 状態でスタックしています

クラスターの登録後、クラスターが Amazon EKS コンソールで Pending 状態のままスタックしている場合、Amazon EKS Connector がクラスターを AWS に正常に接続しなかったことが原因である可能性があります。登録済みのクラスターの場合、Pending 状態は接続がまだ正常に確立されていないことを意味します。この問題を解決するには、ターゲットの Kubernetes クラスターにマニフェストを適用していることを確認します。クラスターにマニフェストを適用したにもかかわらず、まだ Pending 状態である場合、eks-connector statefulset に異常がある可能性があります。この問題をトラブルシューティングするには、このトピックの [the section called “Amazon EKS Connector Pod がクラッシュループしている”](#) を参照してください。

## コンソールのエラー: ユーザーシステム:serviceaccount:eks-connector:eks-connector can't impersonate resource users in API group at cluster scope (eks-connector はクラスタースコープで API グループのリソースユーザーを偽装できません)

Amazon EKS Connector は、Kubernetes [ユーザー偽装](#)を使用して AWS Management Console から [IAM プリンシパル](#)に代わってアクションを実行します。AWS eks-connector のサービスアカウントから Kubernetes API にアクセスする各プリンシパルについては、Kubernetes ユーザー名として IAM ARN で対応する Kubernetes ユーザーを偽装するために許可を付与する必要があります。次の例では、Kubernetes ユーザーに IAM ARN がマッピングされています。

- AWS アカウント **111122223333** からの IAM ユーザー **john** は、Kubernetes ユーザーにマッピングされています。 [IAM のベストプラクティス](#)では、ユーザーではなくロールに許可を付与することが推奨されています。

```
arn:aws:iam::111122223333:user/john
```

- AWS アカウント **111122223333** からの IAM ロール**###**は、Kubernetes ユーザーにマッピングされています。

```
arn:aws:iam::111122223333:role/admin
```

結果は、AWS STS セッションの ARN ではなく IAM ロールの ARN になります。

マッピングされたユーザーを偽装するアカウント権限を eks-connector サービスに付与するために ClusterRole および ClusterRoleBinding を設定する方法については、「[the section called “クラスターに対するアクセスを許可する”](#)」を参照してください。テンプレートで、%IAM\_ARN% が AWS Management Console IAM プリンシパルの IAM ARN に置き換えられていることを確認してください。

コンソールのエラー: [...] is forbidden: User [...] cannot list resource [...] in API group at the cluster scope ([...] は禁止されています: ユーザー [...] が、クラスタースコープで API グループのリソース [...] をリストできません)

次の問題を検討します。Amazon EKS Connector により、ターゲットの Kubernetes クラスターでリクエストしている AWS Management Console IAM プリンシパルが正常に偽装されました。ただし、偽装されたプリンシパルには Kubernetes API オペレーション用の RBAC 許可がありません。

この問題を解決するには、追加のユーザーに権限を付与する方法が 2 つあります。以前に Helm チャート 経由で eks-connector をインストールしたことがある場合は、以下のコマンドを実行することでユーザーにアクセス権を簡単に付与できます。userARN1 と userARN2 を IAM ロールの ARN のリストに置き換えて、Kubernetes リソース表示のアクセス権を付与します。

```
helm upgrade eks-connector oci://public.ecr.aws/eks-connector/eks-connector-chart \
 --reuse-values \
 --set 'authentication.allowedUserARNs={userARN1,userARN2}'
```

または、クラスターの管理者として個々の Kubernetes ユーザーに適切なレベルの RBAC 権限を付与します。詳細な説明と例については[the section called “クラスターに対するアクセスを許可する”](#)を参照してください。

コンソールのエラー: Amazon EKS can't communicate with your Kubernetes cluster API server. 正常に接続するには、クラスターが ACTIVE 状態である必要があります。数分後にもう一度お試しください。

Amazon EKS サービスがターゲットクラスター内の Amazon EKS Connector と通信できない場合、次のいずれかの原因が考えられます。

- ターゲットクラスターの Amazon EKS Connector に異常があります。
- ターゲットクラスターと AWS リージョン間の接続が悪い、または中断している。

この問題を解決するには、[Amazon EKS Connector ログ](#)を確認してください。Amazon EKS Connector のエラーが表示されない場合、数分後に接続を再試行してください。ターゲットクラスターで高レイテンシーや接続が断続的になる状況が定期的に発生する場合は、近くにある AWS リストにクラスターを再登録することを検討してください。

## Amazon EKS Connector Pod がクラッシュループしている

Amazon EKS Connector Pod が `CrashLoopBackOff` 状態になる原因は数多くあります。この問題は `connector-init` コンテナに関係している可能性があります。Amazon EKS Connector Pod のステータスを確認します。

```
kubectl get pods -n eks-connector
```

出力例は次のとおりです。

| NAME            | READY | STATUS                | RESTARTS | AGE |
|-----------------|-------|-----------------------|----------|-----|
| eks-connector-0 | 0/2   | Init:CrashLoopBackOff | 1        | 7s  |

出力が前の出力と似ている場合、[the section called “Amazon EKS Connector のログを検査します”](#)を参照して問題をトラブルシューティングしてください。

### eks-connector の起動の失敗: InvalidActivation

Amazon EKS Connector を初めて起動する場合、Amazon Web Services に `activationId` および `activationCode` を登録します。登録が失敗することがあり、次のエラーと同様なエラーで `connector-init` コンテナがクラッシュする原因になります。

```
F1116 20:30:47.261469 1 init.go:43] failed to initiate eks-connector:
InvalidActivation:
```

この問題のトラブルシューティングを行うには、次の原因および推奨される修正を検討してください。

- `activationId` および `activationCode` がマニフェストファイルにないために、登録が失敗した可能性があります。このような場合、値が `RegisterCluster` API オペレーションから返された正しいものであり、`activationCode` がマニフェストファイルに含まれていることを確認してください。`activationCode` は Kubernetes シークレットに追加されるため、base64 でエンコードする必要があります。詳細については、「[the section called “ステップ 1: クラスターの登録”](#)」を参照してください。

- アクティベーションの有効期限が切れたため、登録が失敗した可能性があります。これは、セキュリティ上の理由により、クラスターを登録してから 3 日以内に Amazon EKS Connector をアクティブ化する必要があるためです。この問題を解決するには、Amazon EKS Connector のマニフェストが有効期限の日時よりも前にターゲットの Kubernetes クラスターに適用されていることを確認してください。アクティブ化の有効期限を確認するには、DescribeCluster API オペレーションを呼び出します。

```
aws eks describe-cluster --name my-cluster
```

次のレスポンスの例では、有効期限の日時は 2021-11-12T22:28:51.101000-08:00 として記録されています。

```
{
 "cluster": {
 "name": "my-cluster",
 "arn": "arn:aws:eks:region:111122223333:cluster/my-cluster",
 "createdAt": "2021-11-09T22:28:51.449000-08:00",
 "status": "FAILED",
 "tags": {
 },
 "connectorConfig": {
 "activationId": "000000000-0000-0000-0000-000000000000",
 "activationExpiry": "2021-11-12T22:28:51.101000-08:00",
 "provider": "OTHER",
 "roleArn": "arn:aws:iam::111122223333:role/my-connector-role"
 }
 }
}
```

activationExpiry が渡されたら、クラスターの登録を解除し、再度登録します。これにより、新しいアクティベーションが生成されます。

## クラスターノードにアウトバウンド接続がありません

正常に動作するには、Amazon EKS Connector に複数の AWS エンドポイントへのアウトバウンド接続が必要です。ターゲットの AWS リージョンへのアウトバウンド接続が使用可能でないと、プライベートクラスターを接続することはできません。この問題を解決するには、必要なアウトバウンド接続を追加する必要があります。コネクタの要件については、「[the section called “Amazon EKS Connector の考慮事項”](#)」を参照してください。



## Amazon EKS Connector Pod が **ImagePullBackOff** 状態になっている

get pods コマンドを実行しているときに Pod が ImagePullBackOff 状態にある場合、正常に動作しません。Amazon EKS Connector Pod が ImagePullBackOff 状態の場合、正常に作動できません。Amazon EKS Connector Pod の状態を確認してください。

```
kubectl get pods -n eks-connector
```

出力例は次のとおりです。

| NAME            | READY | STATUS                | RESTARTS | AGE |
|-----------------|-------|-----------------------|----------|-----|
| eks-connector-0 | 0/2   | Init:ImagePullBackOff | 0        | 4s  |

デフォルトの Amazon EKS Connector のマニフェストファイルは「[Amazon ECR Public Gallery](#)」からイメージを参照します。ターゲットの Kubernetes クラスターが Amazon ECR Public Gallery からイメージをプルできない可能性があります。Amazon ECR Public Gallery のイメージのプルに関する問題を解決するか、選択したプライベートコンテナレジストリ内のイメージのミラーリングを検討してください。

## AWS Connector に関するよくある質問

Q: Amazon EKS Connector の基盤となるテクノロジーはどのように機能しますか？

A: Amazon EKS Connector は AWS Systems Manager (Systems Manager) エージェントに基づいています。Amazon EKS Connector は Kubernetes クラスター上で StatefulSet として実行されます。これにより接続が確立され、クラスターの API サーバーと Amazon Web Services 間の通信がプロキシされます。これは、AWS からクラスターが切断されるまで Amazon EKS コンソールにクラスターデータを表示しておくためです。Systems Manager エージェントはオープンソースのプロジェクトです。このプロジェクトについての詳細は、「[GitHub project page](#)」を参照してください。

Q: 接続したいオンプレミスの Kubernetes クラスターがあります。接続するにはファイアウォールポートを開く必要がありますか？

A: いいえ。ファイアウォールポートを開く必要はありません。Kubernetes クラスターは、AWS リージョンへのアウトバウンド接続のみを必要とします。AWS のサービスがオンプレミスネットワークのリソースにアクセスすることはありません。Amazon EKS Connector はクラスターで実行され、AWS への接続を開始します。クラスターの登録が完了すると、クラスター上の Kubernetes

API サーバーからの情報を必要とする Amazon EKS コンソールからアクションを開始した後のみ、AWS により Amazon EKS Connector にコマンドが発行されます。

Q: Amazon EKS Connector によってクラスターから AWS に送信されるデータはどのようなものですか？

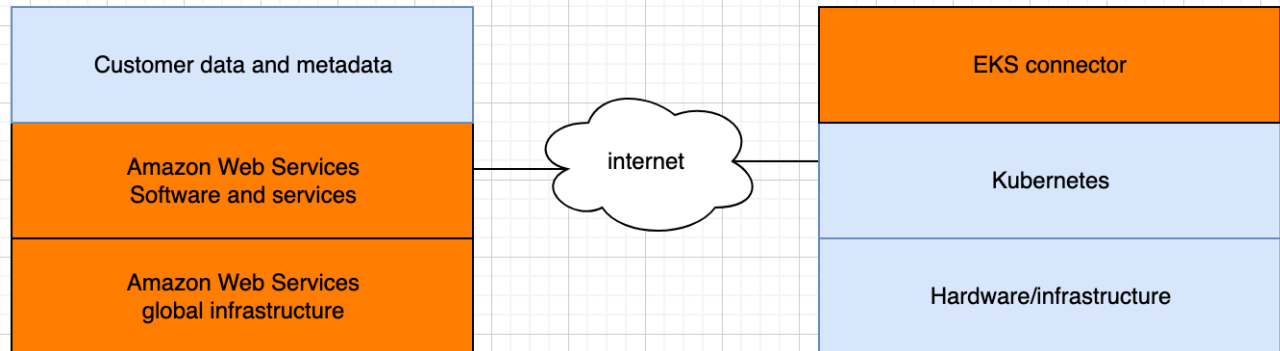
A: Amazon EKS Connector により送信されるのは、AWS でのクラスターの登録に必要な技術情報です。また、お客様がリクエストする Amazon EKS コンソールにおける機能のクラスターとワークロードのメタデータも送信されます。Amazon EKS Connector は、AWS ヘデータの送信を必要とする Amazon EKS コンソールまたは Amazon EKS API からアクションを開始した場合にのみ、このデータを収集または送信します。デフォルトでは、AWS で Kubernetes のバージョン番号以外のデータが保存されることはありません。許可した場合にのみデータを保存します。

Q: AWS リージョンの外部のクラスターを接続できますか？

A: はい、任意の場所からクラスターを Amazon EKS に接続できます。さらに、Amazon EKS サービスはどんな AWS のパブリックの商用 AWS でも配置できます。これは、クラスターからターゲットの AWS リージョンへの有効なネットワーク接続により動作します。UI パフォーマンスを最適化するために、クラスターの場所に最も近い AWS リージョンを選択することをお勧めします。例えば、東京で実行しているクラスターがある場合、レイテンシーを低くするためにクラスターを東京の AWS リージョン (つまり ap-northeast-1 AWS リージョン) に接続します。任意の場所から、中国リージョンまたは GovCloud AWS リージョンを除くパブリックな商用 AWS リージョンの Amazon EKS にクラスターを接続できます。

## Amazon EKS Connector のセキュリティの説明

Amazon EKS Connector は、Kubernetes クラスターで実行されるオープンソースコンポーネントです。このクラスターは、AWS 環境の外部に配置できます。これにより、セキュリティ上の責任に関する追加の考慮事項が作成されます。この設定については、次の図で示されています。オレンジ色は AWS の責任、青色はお客様の責任を表します。



このトピックでは、接続されているクラスターが AWS の外部にある場合の責任モデルの違いについて説明します。

## AWS の責任

- お客様の Kubernetes クラスターで実行され、AWS と通信する [オープンソースコンポーネント](#) である Amazon EKS Connector の保守、構築および提供。
- 接続されている Kubernetes クラスターと AWS のサービス間のトランスポートおよびアプリケーションレイヤーの通信セキュリティの維持。

## お客様の責任

- Kubernetes クラスター固有のセキュリティの具体例:
  - Kubernetes シークレットが適切に暗号化され、保護されている必要があります。
  - eks-connector 名前空間へのアクセスをロックダウンします。
- AWS からの [IAM プリンシパル](#) のアクセスを管理するロールベースのアクセスコントロール (RBAC) 許可の設定。手順については、[the section called “クラスターに対するアクセスを許可する”](#) を参照してください。
- Amazon EKS Connector のインストールおよびアップグレード。
- 接続されている Kubernetes クラスターをサポートするハードウェア、ソフトウェア、およびインフラストラクチャの保守。
- (例えば [ルートユーザーの認証情報](#) を保護することを通じて) AWS アカウントを保護します。

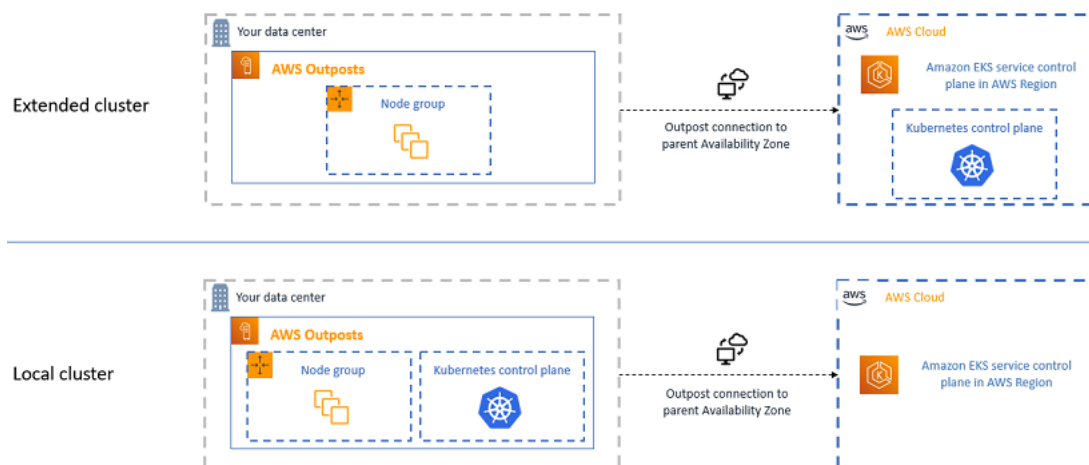
# AWS Outposts を使用して Amazon EKS をオンプレミスにデプロイする

Amazon EKS を使用して、オンプレミス Kubernetes アプリケーションを AWS Outposts で実行できます。次の方法で Amazon EKS を Outposts にデプロイできます。

- 拡張クラスター - Outpost の AWS リージョンとノードで Kubernetes コントロールプレーンを実行します。
- ローカルクラスター - Outpost で Kubernetes コントロールプレーンとノードを実行します。

どちらのデプロイオプションでも、Kubernetes コントロールプレーンは、AWS によって完全に管理されています。クラウドで使用するのと同じ Amazon EKS API、ツール、コンソールを使用して Outposts で Amazon EKS を作成および実行することができます。

以下の図に、これらのデプロイのオプションを示します。



## 各デプロイオプションを使用する時

ローカルクラスターと拡張クラスターはどちらも汎用のデプロイオプションであり、さまざまなアプリケーションに使用できます。

ローカルクラスターでは、Amazon EKS クラスター全体を Outposts でローカルに実行できます。この方法により、クラウドへの一時的なネットワーク切断によって生じる可能性のあるアプリケーションダウンタイムのリスクを軽減できます。このようなネットワーク接続断は、ファイバーの切断や天候によって発生する可能性があります。すべての Amazon EKS クラスター全体が Outposts でローカルに実行されるため、アプリケーションは引き続き使用できます。クラウドへのネットワーク切断中

にクラスターオペレーションを実行できます。詳細については、「[the section called “切断の準備をする”](#)」を参照してください。Outposts から親 AWS リージョンへのネットワーク接続の品質が気になる場合、およびネットワークの切断による高可用性を必要とする場合は、ローカルクラスターデプロイオプションを使用してください。

拡張クラスターでは、Kubernetes コントロールプレーンが親 AWS リージョンで動作するため、Outpost の容量を節約できます。このオプションは、Outpost から AWS リージョンへの信頼性が高く冗長なネットワーク接続に投資できる場合に適している可能性があります。このオプションでは、ネットワーク接続の品質が重要です。Kubernetes が Kubernetes コントロールプレーンとノード間のネットワーク切断を処理する方法によって、アプリケーションのダウンタイムが発生する可能性があります。Kubernetes の動作の詳細については、Kubernetes ドキュメントの「[Scheduling, Preemption, and Eviction](#)」を参照してください。

## デプロイオプションの比較

以下の表は、これら 2 つのオプションの違いを比較したものです。

| 機能                          | 拡張クラスター                                  | ローカルクラスター                                                                                                                                                                                   |
|-----------------------------|------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Kubernetes コントロールプレーンの場所    | AWS リージョン                                | Outpost                                                                                                                                                                                     |
| Kubernetes コントロールプレーンのアカウント | AWS アカウント                                | お客様のアカウント                                                                                                                                                                                   |
| リージョナルな可用性                  | 「 <a href="#">サービスエンドポイント</a> 」を参照してください | 米国東部 (バージニア北部)、米国東部 (オハイオ)、米国西部 (北カリフォルニア)、米国西部 (オレゴン)、アジアパシフィック (ソウル)、アジアパシフィック (シンガポール)、アジアパシフィック (東京)、アジアパシフィック (シドニー)、カナダ (中部)、欧州 (フランクフルト)、欧州 (アイルランド)、欧州 (ロンドン)、中東 (バーレーン)、南米 (サンパウロ) |

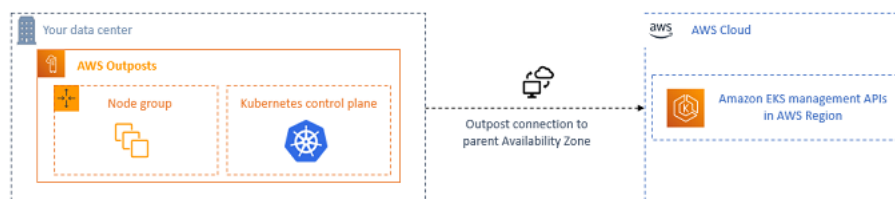
| 機能                    | 拡張クラスター                                                                                                                 | ローカルクラスター                                                                                                                                          |
|-----------------------|-------------------------------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------|
| Kubernetes マイナーバージョン  | <a href="#">サポートされている Amazon EKS バージョン</a> 。                                                                            | <a href="#">サポートされている Amazon EKS バージョン</a> 。                                                                                                       |
| プラットフォームのバージョン        | 「 <a href="#">the section called “プラットフォームのバージョン”</a> 」を参照してください                                                        | 「 <a href="#">the section called “EKS プラットフォームのバージョン”</a> 」を参照してください                                                                               |
| Outpost フォームファクター     | Outpost ラック                                                                                                             | Outpost ラック                                                                                                                                        |
| ユーザーインターフェイス          | AWS Management Console、AWS CLI、Amazon EKS API、eksctl、AWS CloudFormation、および Terraform                                   | AWS Management Console、AWS CLI、Amazon EKS API、eksctl、AWS CloudFormation、および Terraform                                                              |
| マネージドポリシー             | <a href="#">AmazonEKSClusterPolicy</a> および <a href="#">the section called “AWS 管理ポリシー: AmazonEKS ServiceRolePolicy”</a> | <a href="#">AmazonEKSLocalOutpostClusterPolicy</a> および <a href="#">the section called “AWS マネージドポリシー: AmazonEKS LocalOutpostServiceRolePolicy”</a> |
| クラスター VPC およびサブネット    | 「 <a href="#">the section called “VPC およびサブネットの要件”</a> 」を参照してください。                                                      | 「 <a href="#">the section called “VPC とサブネットを作成する”</a> 」を参照してください                                                                                  |
| クラスターエンドポイントのアクセス     | 公共または個人、あるいはその両方                                                                                                        | 非公開のみ                                                                                                                                              |
| Kubernetes API サーバー認証 | AWS Identity and Access Management (IAM) および OIDC                                                                       | IAM および x.509 証明書                                                                                                                                  |
| ノードタイプ                | セルフマネージド型のみ                                                                                                             | セルフマネージド型のみ                                                                                                                                        |
| ノードのコンピュートタイプ         | Amazon EC2 オンデマンド                                                                                                       | Amazon EC2 オンデマンド                                                                                                                                  |

| 機能                                 | 拡張クラスター                                                                                                                          | ローカルクラスター                                                                                                                        |
|------------------------------------|----------------------------------------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------|
| ノードストレージタイプ                        | Amazon EBS gp2 とローカル NVMe SSD                                                                                                    | Amazon EBS gp2 とローカル NVMe SSD                                                                                                    |
| Amazon EKS 最適化 AMI                 | Amazon Linux、Windows、Bottlerocket                                                                                                | Amazon Linux のみ                                                                                                                  |
| IP バージョン                           | IPv4 のみ                                                                                                                          | IPv4 のみ                                                                                                                          |
| アドオン                               | Amazon EKS アドオンまたはセルフマネージドアドオン                                                                                                   | セルフマネージド型アドオンのみ                                                                                                                  |
| デフォルトの Container Network Interface | Amazon VPC CNI plugin for Kubernetes                                                                                             | Amazon VPC CNI plugin for Kubernetes                                                                                             |
| Kubernetes コントロールプレーンのログ           | Amazon CloudWatch Logs                                                                                                           | Amazon CloudWatch Logs                                                                                                           |
| 負荷分散                               | <a href="#">AWS Load Balancer Controller</a> を使用して Application Load Balancer のみをプロビジョニングします (Network Load Balancer はプロビジョニングしない) | <a href="#">AWS Load Balancer Controller</a> を使用して Application Load Balancer のみをプロビジョニングします (Network Load Balancer はプロビジョニングしない) |
| シークレットエンベロープ暗号化                    | 「 <a href="#">the section called “シークレット暗号化を有効にする”</a> 」を参照してください                                                                | サポートされていません                                                                                                                      |
| サービスアカウントの IAM ロール                 | 「 <a href="#">the section called “IRSA による認証情報”</a> 」を参照してください                                                                   | サポートされていません                                                                                                                      |
| トラブルシューティング                        | 「 <a href="#">トラブルシューティング</a> 」を参照してください                                                                                         | 「 <a href="#">the section called “クラスターのトラブルシューティングを行う”</a> 」を参照してください                                                           |



# 高可用性を実現するために AWS Outposts でローカル Amazon EKS クラスターを作成する

ローカルクラスターを使用すると、Amazon EKS クラスターすべてを AWS Outposts でローカルに実行できます。これにより、クラウドへの一時的なネットワーク切断によって生じる可能性のあるアプリケーションダウンタイムのリスクを軽減できます。このような接続は、ファイバーの切断や天候によって発生する可能性があります。Kubernetes クラスター全体が Outposts でローカルに実行されるため、アプリケーションは引き続き使用できます。クラウドへのネットワーク切断中にクラスターオペレーションを実行できます。詳細については、「[the section called “切断の準備をする”](#)」を参照してください。次の図は、ローカルクラスターのデプロイを示しています。



ローカルクラスターは一般的に Outposts ラックで使用できます。

## サポート対象の AWS リージョン

ローカルクラスターを作成できる AWS リージョンは、米国東部 (オハイオ)、米国東部 (バージニア北部)、米国西部 (北カリフォルニア)、米国西部 (オレゴン)、アジアパシフィック (ソウル)、アジアパシフィック (シンガポール)、アジアパシフィック (シドニー)、アジアパシフィック (東京)、カナダ (中部)、欧州 (フランクフルト)、欧州 (アイルランド)、欧州 (ロンドン)、中東 (バーレーン)、南米 (サンパウロ) です。サポートされる機能の詳細については、「[the section called “デプロイオプションの比較”](#)」を参照してください。

## AWS Outposts で Amazon EKS クラスターをデプロイする

このトピックでは、Outpost でローカルクラスターを実行する際に考慮すべき事項について、概要を説明します。また、Outpost でローカルクラスターをデプロイする方法についても説明します。

### ⚠ Important

- これらの考慮事項は、関連する Amazon EKS ドキュメントには記載されていません。Amazon EKS ドキュメントに記載されている他のトピックの情報が、ここに紹介する考慮事項と競合する場合は、こちらの考慮事項を優先してください。



- これらの考慮事項は変更される可能性があり、変更は頻繁に行われる場合があります。そのため、このトピックは定期的に確認することをお勧めします。
  - 考慮事項の多くは、AWS Cloud でクラスターを作成する場合の考慮事項とは異なります。
- 
- ローカルクラスターは Outpost ラックのみをサポートします。単一のローカルクラスターは、単一の論理 Outpost を構成する複数の物理 Outpost ラックにわたって実行できます。単一のローカルクラスターを複数の論理 Outposts にわたって実行することはできません。各論理 Outpost には、単一の Outpost ARN があります。
  - ローカルクラスターは、Outpost のアカウントで Kubernetes コントロールプレーンを実行および管理します。Kubernetes コントロールプレーンインスタンスでワークロードを実行したり、Kubernetes コントロールプレーンのコンポーネントを変更することはできません。これらのノードは Amazon EKS サービスによって管理されます。Kubernetes コントロールプレーンへの変更は、パッチ適用などの自動 Amazon EKS 管理アクションでは存続しません。
  - ローカルクラスターは、セルフマネージド型アドオンとセルフマネージド型 Amazon Linux ノードグループをサポートしています。[Amazon VPC CNI plugin for Kubernetes](#)、[kube-proxy](#)、[CoreDNS](#) アドオンはローカルクラスターに自動的にインストールされます。
  - ローカルクラスターでは、Outposts で Amazon EBS を使用する必要があります。お客様の Outpost では、Kubernetes コントロールプレーンストレージに使用できる Amazon EBS が必要です。
  - ローカルクラスターは、Outposts で Amazon EBS を使用します。お客様の Outpost では、Kubernetes コントロールプレーンストレージに使用できる Amazon EBS が必要です。Outposts は、Amazon EBS gp2 ボリュームのみをサポートします。
  - Amazon EBS のバックアップ対象である Kubernetes PersistentVolumes は、Amazon EBS CSI ドライバーを使用してサポートされています。
  - ローカルクラスターのコントロールプレーンインスタンスは、[可用性の高いスタックされたトポロジ](#)で設定されます。クォーラムを維持するには、3 つのコントロールプレーンインスタンスのうち 2 つが常に正常である必要があります。クォーラムが失われた場合、新しいマネージドインスタンスを有効にするにはいくつかのサービス側のアクションが必要になるため、AWS サポートにお問い合わせください。

## 前提条件

- [Outposts デプロイオプション](#)、[キャパシティに関する考慮事項に基づく AWS Outposts での Amazon EKS クラスターのインスタンスタイプとプレースメントグループの選択](#)、および [VPC 要件と考慮事項](#)に精通していること。
- 既存のアウトポスト。詳細については、「[AWS Outposts とは](#)」を参照してください。
- コンピュータまたは AWS CloudShell に、kubectl コマンドラインツールがインストールされていること。バージョンはご使用のクラスターの Kubernetes バージョンと同じか、1 つ前のマイナーバージョン以前、あるいはそれより新しいバージョンが使用できます。例えば、クラスターのバージョンが 1.29 である場合、kubectl のバージョン 1.28、1.29、または 1.30 が使用できます。kubectl をインストールまたはアップグレードする方法については「[the section called “kubectl と eksctl のセットアップ”](#)」を参照してください。
- ご使用のデバイスまたは AWS クラウドシェル で、バージョン 2.12.3 以降、または AWS コマンドラインインターフェイス (AWS CLI のバージョン 1.27.160 以降がインストールおよび設定されていること。現在のバージョンを確認するには「aws --version | cut -d / -f2 | cut -d ' ' -f1」を参照してください。yum、apt-get、macOS 用の Homebrew などのパッケージマネージャーは、多くの場合 AWS CLI の最新バージョンより数バージョン古くなっています。最新バージョンをインストールするには「AWS コマンドラインインターフェイスユーザーガイド」の「[インストール](#)」および「[aws configure を使用したクイック設定](#)」を参照してください。AWS クラウドシェルにインストールされている AWS CLI バージョンも最新バージョンより数バージョン遅れることがあります。更新するには、「AWS CloudShell ユーザーガイド」の「[ホームディレクトリへの AWS CLI のインストール](#)」を参照してください。
- Amazon EKS クラスターへの create および describe のアクセス許可を持つ IAM プリンシパル (ユーザーまたはロール)。詳細については、「[the section called “Outpost にローカル Kubernetes クラスターを作成する”](#)」および「[the section called “すべてのクラスターの一覧表示または説明”](#)」を参照してください。

ローカルの Amazon EKS クラスターが作成される時点で、クラスターを作成する [IAM プリンシパル](#)が恒久的に追加されています。とりわけ、プリンシパルは管理者として、Kubernetes RBAC 承認テーブルに追加されます。このエンティティには system:masters アクセス許可が付与されています。このエンティティの ID は、クラスター設定には表示されません。したがって、クラスターを作成したエンティティをメモし、削除しないようにすることが重要です。初期状態では、サーバーを作成したプリンシパルのみが、kubectl を使用して Kubernetes API サーバーへの呼び出しを実行できます。コンソールを使用してクラスターを作成する場合は、クラスター上で kubectl コマンドを実行する際、同じ IAM 認証情報が AWS SDK 認証情報チェーンにあることを確認する必要があります。クラスターの作成が完了したら、他の IAM プリンシパルにクラスターへのアクセスを付与できます。

## Amazon EKS ローカルクラスターを作成します。

このページで説明されている以下のツールを使用して、ローカルクラスターを作成できます。

- [the section called “eksctl”](#)
- [the section called “AWS Management Console”](#)

[AWS CLI](#)、[Amazon EKS API](#)、[AWS SDK](#)、[AWS CloudFormation](#) または [Terraform](#) を使用して Outposts にクラスターを作成することもできます。

### eksctl

**eksctl** を使用してローカルクラスターを作成するには

1. デバイスまたは AWS CloudShell に eksctl コマンドラインツールのバージョン 0.210.0 以降をインストールします。eksctl をインストールまたはアップグレードするには eksctl ドキュメントの「[インストール](#)」を参照してください。
2. 次のコンテンツをデバイスにコピーします。次の値を置き換えたら、変更したコマンドを実行して outpost-control-plane.yaml ファイルを作成します。
  - **region-code** は、クラスターを作成する [サポートされている AWS リージョン](#) に置き換えます。
  - **my-cluster** の部分は、自分のクラスター名に置き換えます。この名前には英数字 (大文字と小文字が区別されます) とハイフンのみを使用できます。先頭の文字は英数字である必要があります。また、100 文字より長くすることはできません。名前は、クラスターを作成する AWS リージョンおよび AWS アカウント内で一意である必要があります。名前は、クラスターを作成する AWS リージョンおよび AWS アカウント内で一意である必要があります。
  - **vpc-ExampleID1** と **subnet-ExampleID1** を既存の VPC とサブネットの ID に置き換えます。VPC とサブネットは、[AWS Outposts で Amazon EKS クラスターの VPC とサブネットを作成する](#) 要件を満たす必要があります。
  - **uniqueid** を Outpost の ID に置き換えます。
  - **m5.large** を Outpost で使用可能なインスタンスタイプに置き換えます。インスタンスタイプを選択する前に、「[the section called “容量に関する考慮事項”](#)」を参照してください。3 つのコントロールプレーンインスタンスがデプロイされます。この番号を変更することはできません。

```
cat >outpost-control-plane.yaml <<EOF
apiVersion: eksctl.io/v1alpha5
```

```
kind: ClusterConfig

metadata:
 name: my-cluster
 region: region-code
 version: "1.32"

vpc:
 clusterEndpoints:
 privateAccess: true
 id: "vpc-vpc-ExampleID1"
 subnets:
 private:
 outpost-subnet-1:
 id: "subnet-subnet-ExampleID1"

outpost:
 controlPlaneOutpostARN: arn:aws:outposts:region-code:111122223333:outpost/op-uniqueid
 controlPlaneInstanceType: m5.large
EOF
```

使用可能なすべてのオプションとデフォルトの完全なリストについては、eksctl ドキュメントの「[AWS Outposts サポート](#)」および「[設定ファイルスキーマ](#)」を参照してください。

3. 前のステップで作成した設定ファイルを使用してクラスターを作成します。eksctl は、クラスターをデプロイするために VPC と 1 つのサブネットを Outpost に作成します。

```
eksctl create cluster -f outpost-control-plane.yaml
```

クラスターのプロビジョニングには数分かかります。クラスターの作成中は数行の出力が表示されます。出力の最後の行は次のサンプル行のようになります。

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

#### Tip

eksctl を使用してクラスターを作成するときに指定できるほとんどのオプションを表示するには、eksctl create cluster --help コマンドを使用します。使用可能なオプションをすべて表示するには config ファイルを使用します。詳細については「eksctl

ドキュメント」の「[Using config files](#)」(設定ファイルの使用)と「[設定ファイルのスキーマ](#)」を参照してください。[設定ファイルの例](#)は、GitHub で見つけることができます。

eksctl コマンドはクラスターを作成した IAM プリンシパル (ユーザーまたはロール) の[アクセスエントリ](#)を自動的に作成し、クラスター上の Kubernetes オブジェクトに対するアクセス許可を IAM プリンシパル管理者に付与しました。クラスター作成者にクラスター上の Kubernetes オブジェクトへの管理者アクセス権を付与したくない場合は、前の設定ファイルに次のテキストを追加します: bootstrapClusterCreatorAdminPermissions: false (metadata、vpc、および outpost と同じレベル)。このオプションを追加した場合、クラスターの作成後に少なくとも 1 つの IAM プリンシパルのアクセスエントリを作成する必要があります。そうしないと、IAM プリンシパルはクラスター上の Kubernetes オブジェクトにアクセスできなくなります。

## AWS Management Console

AWS Management Console を使用してクラスターを作成するには

1. Amazon EKS の要件を満たす既存の VPC とサブネットが必要です。詳細については、「[the section called “VPC とサブネットを作成する”](#)」を参照してください。
2. 既にローカルクラスター IAM ロールがある場合、または eksctl を使用してクラスターを作成する場合は、このステップはスキップできます。デフォルトでは eksctl により、ロールが自動的に作成されます。
  - a. IAM 信頼ポリシー用の JSON ファイルを作成するには次のコマンドを実行してください。

```
cat >eks-local-cluster-role-trust-policy.json <<EOF
{
 "Version": "2012-10-17",
 "Statement": [
 {
 "Effect": "Allow",
 "Principal": {
 "Service": "ec2.amazonaws.com"
 },
 "Action": "sts:AssumeRole"
 }
]
}
EOF
```

- b. Amazon EKS クラスターの IAM ロールを作成します。IAM ロールを作成するには、ロールを作成する [IAM プリンシパル](#)に `iam:CreateRole` アクション (許可) を割り当てる必要があります。

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

- c. [AmazonEKSLocalOutpostClusterPolicy](#) という名前の Amazon EKS マネージド型ポリシーをロールにアタッチします。IAM ポリシーを [IAM プリンシパル](#)にアタッチするにはポリシーのアタッチを行っているプリンシパルに、次のいずれかの IAM アクション (許可) を割り当てる必要があります: `iam:AttachUserPolicy` または `iam:AttachRolePolicy`。

```
aws iam attach-role-policy --policy-arn arn:aws:iam::aws:policy/AmazonEKSLocalOutpostClusterPolicy --role-name myAmazonEKSLocalClusterRole
```

3. [Amazon EKS コンソール](#)を開きます。
4. コンソール画面の上部で、[サポートされている AWS リージョン](#) を選択したことを確認します。
5. [クラスターを追加]、[作成] の順にクリックします。
6. [クラスターの設定] ページで、次のフィールドの値を入力または選択します。
- [Kubernetes コントロールプレーンの場所] – AWS Outposts を選択します。
  - [Outpost ID] - コントロールプレーンを作成する Outpost の ID を選択します。
  - [インスタンスタイプ] - インスタンスタイプを選択します。Outpost で使用可能なインスタンスタイプのみが表示されます。ドロップダウンリストでは、各インスタンスタイプにそのインスタンスタイプに推奨されるノード数が示されています。インスタンスタイプを選択する前に、「[the section called “容量に関する考慮事項”](#)」を参照してください。すべてのレプリカは、同じインスタンスタイプを使用してデプロイされます。クラスターの作成後にインスタンスタイプを変更することはできません。3 つのコントロールプレーンインスタンスがデプロイされます。この番号を変更することはできません。
  - [名前] - クラスターの名前。AWS アカウント内で一意にする必要があります。この名前には英数字 (大文字と小文字が区別されます) とハイフンのみを使用できます。先頭の文字は英数字である必要があります。また、100 文字より長くすることはできません。名前は、クラスターを作成する AWS リージョンおよび AWS アカウント内で一意である必要があります。名前は、クラスターを作成する AWS リージョンおよび AWS アカウント内で一意である必要があります。
  - [Kubernetes バージョン] – クラスターで使用する Kubernetes バージョンを選択します。以前のバージョンを使用する必要がない限り、最新バージョンを選択することをお勧めします。



- [クラスターサービスロール] - AWS リソースを管理することを Kubernetes コントロールプレーンに許可するために、前のステップで作成した Amazon EKS クラスター IAM ロールを選択します。
- [Kubernetes クラスター管理者アクセス] — クラスターを作成する IAM プリンシパル (ロールまたはユーザー) にクラスター上の Kubernetes オブジェクトへの管理者アクセス権を付与する場合は、デフォルト (許可) をそのまま使用します。Amazon EKS は IAM プリンシパルのアクセスエントリを作成し、クラスター管理者にそのアクセスエントリに対するアクセス許可を付与します。アクセスエントリの詳細については「[the section called “許可を付与する”](#)」を参照してください。

クラスターを作成するプリンシパルとは異なる IAM プリンシパルに Kubernetes クラスターオブジェクトへの管理者アクセス権を付与したい場合は、拒否のオプションを選択します。クラスターの作成後、アクセスエントリを作成する IAM アクセス許可を持つすべての IAM プリンシパルは、Kubernetes クラスターオブジェクトへのアクセスを必要とする任意の IAM プリンシパルのアクセスエントリを追加できます。必要な IAM アクセス許可の詳細については、サービス認可リファレンスの「[Amazon Elastic Kubernetes Service で定義されるアクション](#)」を参照してください。拒否のオプションを選択し、アクセスエントリを作成しない場合、IAM プリンシパルはクラスター上の Kubernetes オブジェクトにアクセスできなくなります。

- [タグ] - (オプション) クラスターにタグを追加します。詳細については、「[the section called “リソースのタグ付け”](#)」を参照してください。このページを読み終えたら、[次へ] を選択してください。

## 7. [ネットワーキングの指定] ページで、次のフィールドの値を選択してください：

- [VPC] - 既存の VPC を選択します。VPC には、作成するクラスター、ノード、およびその他の Kubernetes リソースで利用できる十分な数の IP アドレスが必要です。VPC は、「[VPC の要件と考慮事項](#)」に記載されている要件を満たしている必要があります。
- [サブネット] - デフォルトで、前のフィールドで指定した VPC 内の利用可能なすべてのサブネットがあらかじめ選択されています。選択するサブネットは、「[サブネットの要件と考慮事項](#)」に記載されている要件を満たしている必要があります。
- [セキュリティグループ] - (オプション) Amazon EKS が作成するネットワークインターフェイスに関連付ける 1 つまたは複数のセキュリティグループを指定します。Amazon EKS は、クラスターと VPC 間の通信を可能にするセキュリティグループを自動的に作成します。Amazon EKS はこのセキュリティグループおよびユーザーが選択したセキュリティグループを、作成するネットワークインターフェイスに関連付けます。Amazon EKS が作成するクラスターセキュリティグループの詳細については「[the section called “セキュリティグループの要件”](#)」を参照してください。Amazon EKS が作成するクラスターセキュリティグループのルールを変更でき

ます。独自のセキュリティグループを追加することを選択した場合、クラスターの作成後に選択したセキュリティグループを変更することはできません。オンプレミスホストがクラスターエンドポイントと通信するためには、クラスターセキュリティグループからのインバウンドトラフィックを許可する必要があります。入出力のインターネット接続がないクラスター (プライベートクラスターとも呼ばれる) の場合は、次のいずれかを行う必要があります。

- 必要となる VPC エンドポイントに関連付けられたセキュリティグループを追加します。必要となるエンドポイントの詳細については、「[AWS サービスへのサブネットアクセス](#)」の「[the section called “インターフェイス VPC エンドポイントの使用”](#)」を参照してください。
- VPC エンドポイントに関連付けられたセキュリティグループからのトラフィックを許可するように、Amazon EKS が作成したセキュリティグループを変更します。このページを読み終えたら、[次へ] を選択してください。

8. [オプザバビリティの設定] ページでは、有効にする [メトリクス] と [コントロールプレーンのロギング] オプションをオプションで選択できます。デフォルトでは、それぞれのログタイプは無効化されています。

- Prometheus メトリクスの詳細については、「[the section called “ステップ 1: Prometheus メトリクスを有効にする”](#)」を参照してください。
- 「コントロールプレーン」に関する詳細に関しては、「[the section called “コントロールプレーンのログ”](#)」を参照してください。このページを読み終えたら、[次へ] を選択してください。

9. [確認と作成] ページで、前のページで入力または選択した情報を確認します。変更する必要がある場合は[編集] を選択してください。そのままよければ、[作成] を選択してください。クラスターがプロビジョニングされている間、[ステータス] フィールドに [作成中] と表示されます。

クラスターのプロビジョニングには数分かかります。

## Amazon EKS ローカルクラスターを表示する

1. クラスターを作成すると、作成された Amazon EC2 コントロールプレーンのインスタンスを表示できます。

```
aws ec2 describe-instances --query 'Reservations[*].Instances[*].{Name:Tags[?Key==`Name`][0].Value}' | grep my-cluster-control-plane
```

出力例は次のとおりです。

```
"Name": "my-cluster-control-plane-id1"
"Name": "my-cluster-control-plane-id2"
```



```
"Name": "my-cluster-control-plane-id3"
```

各インスタンスは `node-role.eks-local.amazonaws.com/control-plane` で汚染されているため、コントロールプレーンインスタンスでワークロードがスケジュールされることはありません。テイントの詳細については、Kubernetes ドキュメントの「[Taints and Tolerations](#)」を参照してください。Amazon EKS はローカルクラスターの状態を継続的に監視します。セキュリティパッチや問題のあるインスタンスの修復などの自動管理アクションを実行します。ローカルクラスターがクラウドから切断されると、再接続時にクラスターが正常な状態に修復されるようアクションが実行されます。

2. `eksctl` を使用してクラスターを作成した場合、このステップはスキップできます。`eksctl` がこのステップを完了します。新しいコンテキストを `kubectl config` ファイルに追加して、`kubectl` がクラスターと通信できるようにします。ファイルを作成し更新する手順については、「[the section called “kubectl を使用してクラスターにアクセスする”](#)」を参照してください。

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

出力例は次のとおりです。

```
Added new context arn:aws:eks:region-code:111122223333:cluster/my-cluster to /home/username/.kube/config
```

3. ローカルクラスターの Kubernetes API サーバーに接続するには、サブネットのローカルゲートウェイにアクセスするか、VPC 内から接続します。Outpost ラックをオンプレミスネットワークに接続する方法の詳細については、「AWS Outposts ユーザーガイド」の「[How local gateways for racks work](#)」(ラックのローカルゲートウェイの仕組み)を参照してください。ダイレクト VPC ルーティングを使用しており、Outpost サブネットにローカルゲートウェイへのルートがある場合、Kubernetes コントロールプレーンインスタンスのプライベート IP アドレスがローカルネットワークを介して自動的にブロードキャストされます。ローカルクラスターの Kubernetes API サーバーエンドポイントは Amazon Route 53 (Route 53) でホストされます。API サービスエンドポイントは、パブリック DNS サーバーによって Kubernetes API サーバーのプライベート IP アドレスに解決されます。

ローカルクラスターの Kubernetes コントロールプレーンインスタンスは、クラスターのライフサイクルを通じて変更されない固定プライベート IP アドレスを持つ静的なエラスティックネットワークインターフェイスで構成されます。Kubernetes API サーバーとやり取りするマシンは、ネットワーク接続が切断されている間は Route 53 に接続できない場合があります。このような場合は、操作の継続性を確保するために、静的プライベート IP アドレスを使用して `/etc/hosts`

を構成することをお勧めします。また、ローカル DNS サーバーを設定して Outpost に接続することもお勧めします。詳細については、「[AWS Outposts ドキュメント](#)」を参照してください。次のコマンドを実行して、クラスターとの通信が確立されていることを確認します。

```
kubectl get svc
```

出力例は次のとおりです。

| NAME       | TYPE      | CLUSTER-IP | EXTERNAL-IP | PORT(S) | AGE |
|------------|-----------|------------|-------------|---------|-----|
| kubernetes | ClusterIP | 10.100.0.1 | <none>      | 443/TCP | 28h |

4. (オプション) AWS Cloud から切断された状態のときにローカルクラスターへの認証をテストします。手順については、「[the section called “切断の準備をする”](#)」を参照してください。

## 内部リソース

Amazon EKS はクラスターで次のリソースを作成します。リソースは、Amazon EKS 内部で使用するのためのものです。クラスターが適切に機能しなくなるため、これらのリソースは編集または変更しないでください。

### • 次のミラーポッド:

- aws-iam-authenticator-*node-hostname*
- eks-certificates-controller-*node-hostname*
- etcd-*node-hostname*
- kube-apiserver-*node-hostname*
- kube-controller-manager-*node-hostname*
- kube-scheduler-*node-hostname*

### • 次のセルフマネージド型アドオン:

- kube-system/coredns
- kube-system/ kube-proxy (最初のノードを追加するまで作成されません)
- kube-system/aws-node (最初のノードを追加するまで作成されません)。ローカルクラスターは、クラスターネットワークに Amazon VPC CNI plugin for Kubernetes プラグインを使用します。コントロールプレーンインスタンス (名前が aws-node-controlplane-\* のポッド) の設定を変更しないでください。プラグインが新しいネットワークインターフェイスを作成したときに、デフォルト値を変更できる設定変数があります。詳細については、GitHub の [ドキュメント](#)を参照してください。

- 次のサービス:
  - default/kubernetes
  - kube-system/kube-dns
- eks.system という名前の PodSecurityPolicy
- eks:system:podsecuritypolicy という名前の ClusterRole
- eks:system という名前の ClusterRoleBinding
- [クラスターセキュリティグループ](#)に加えて、Amazon EKS は eks-local-internal-do-not-use-or-edit-*cluster-name-uniqueid* という名前の AWS アカウントにセキュリティグループを作成します。このセキュリティグループにより、コントロールプレーンインスタンスで実行されている Kubernetes コンポーネント間のトラフィックが自由に流れるようになります。

推奨される次の手順は以下の通りです。

- [クラスターを作成した IAM プリンシパルに、AWS Management Console 内の Kubernetes リソースを表示するために必要な許可を付与する](#)
- [IAM エンティティにクラスターへのアクセスを許可する](#)。エンティティに Amazon EKS コンソールで Kubernetes リソースを表示させる場合は、[必要なアクセス許可](#)をエンティティに付与します。
- [クラスターのログ記録を設定する](#)
- [ネットワークの切断](#)中に何が起こるかをよく理解してください。
- [クラスターにノードを追加する](#)
- etcd のバックアップ計画を設定することを検討してください。Amazon EKS は、ローカルクラスターの etcd の自動バックアップと復元をサポートしていません。詳細については、Kubernetes ドキュメントの「[etcd クラスターのバックアップ](#)」を参照してください。2 つの主なオプションは、etcdctl を使用してスナップショットの作成を自動化する、または Amazon EBS ストレージボリュームのバックアップを使用することです。

## AWS Outposts の Kubernetes および Amazon EKS プラットフォームバージョンについて説明します。

ローカルクラスターのプラットフォームバージョンはAWS アウトポスト 上の Amazon EKS クラスターの機能を表します。バージョンには Kubernetes コントロールプレーン上で実行されるコンポーネントが含まれており、Kubernetes API サーバーフラグが有効になっています。また、現在の Kubernetes パッチバージョンも含まれています。各 Kubernetes マイナーバージョンには、プラッ

プラットフォームバージョンが 1 つ以上関連付けられています。別の Kubernetes マイナーバージョンのプラットフォームバージョンは独立しています。クラウド内のローカルクラスターと Amazon EKS クラスターのプラットフォームバージョンは独立しています。

1.31 など、新しい Kubernetes マイナーバージョンがローカルクラスターで使用可能になると、その Kubernetes マイナーバージョンの最初のプラットフォームバージョンは `eks-local-outposts.1` から始まります。また、Amazon EKS では、新しいプラットフォームバージョンを定期的にリリースすることで、新しい Kubernetes コントロールプレーン設定を有効化し、セキュリティ修正プログラムを提供します。

マイナーバージョンで新しいローカルクラスタープラットフォームのバージョンが使用可能になった場合:

- プラットフォームのバージョン番号がインクリメントされます (`eks-local-outposts.n+1`)。
- Amazon EKS は既存のすべての既存のローカルクラスターを、対応する Kubernetes マイナーバージョン用の最新のプラットフォームバージョンに自動的に更新します。既存のプラットフォームバージョンの自動更新は段階的にロールアウトされます。ロールアウトプロセスは、3 つのインスタンスすべてが新しいインスタンスに置き換えられるまで、Outpost で実行されているマネージド型 Kubernetes コントロールプレーンインスタンスを一度に 1 つずつ置き換えます。
- Kubernetes コントロールプレーンインスタンスの置換プロセスは、サービスの中断のリスクがある場合、進行を停止します。Amazon EKS は、他の 2 つの Kubernetes コントロールプレーンインスタンスが正常で、クラスターノードとしての準備状況条件にすべて合格した場合にのみ、インスタンスの置き換えを試みます。
- プラットフォームバージョンのロールアウトが完了するまでに通常 30 分もかかりません。クラスターが長時間 UPDATING 状態のままである場合は、[「the section called “クラスターのトラブルシューティングを行う”」](#)を参照し、AWS サポートに問い合わせてください。AWS サポートからの指示がない限り、Kubernetes コントロールプレーンインスタンスを手動で終了しないでください。
- Amazon EKS は対応するパッチバージョンを適用して新しいノード AMI を発行する可能性があります。すべてのパッチバージョンは単一の Kubernetes マイナーバージョンに対して、Kubernetes コントロールプレーンとノード AMI との間で互換性があります。

新しいプラットフォームバージョンでは重大な変更が発生したり、サービスが中断されたりすることはありません。

ローカルクラスターは常に、指定された Kubernetes バージョン用に入手可能な最新のプラットフォームバージョン (`eks-local-outposts.n`) で作成されます。

現在および最新の プラットフォームバージョンを以下の表に示します。

この特定のドキュメントページへのすべてのソースファイルの変更の通知を受け取るには、RSS リーダーを使用して次の URL をサブスクライブできます。

```
https://github.com/awsdocs/amazon-eks-user-guide/commits/mainline/latest/ug/outposts/eks-outposts-platform-versions.adoc.atom
```

## Kubernetes バージョン 1.31

以下のアドミSSIONコントローラーは、すべての 1.31 プラットフォームバージョンで有効です: CertificateApproval、CertificateSigning、CertificateSubjectRestriction、ClusterTr

| Kubernetes バージョ<br>ン | Amazon EKS プラッ<br>トフォームバージョ<br>ン | リリースノート                                                                    | リリース日          |
|----------------------|----------------------------------|----------------------------------------------------------------------------|----------------|
| 1.31.6               | eks-local-<br>outposts.1         | アウトポスト のロー<br>カル Amazon EKS ク<br>ラスタ用 Kuberne<br>s バージョン v1.31<br>の初回リリース。 | 2025 年 4 月 9 日 |

## Kubernetes バージョン 1.30

以下のアドミSSIONコントローラーは、すべての 1.30 プラットフォームバージョンで有効です: CertificateApproval、CertificateSigning、CertificateSubjectRestriction、ClusterTr

| Kubernetes バージョ<br>ン | Amazon EKS プラッ<br>トフォームバージョ<br>ン | リリースノート                                                     | リリース日           |
|----------------------|----------------------------------|-------------------------------------------------------------|-----------------|
| 1.30.10              | eks-local-<br>outposts.3         | セキュリティの修<br>正および機能強化<br>が行われた新しい<br>プラットフォーム<br>バージョン。kube- | 2025 年 3 月 27 日 |

| Kubernetes バージョン | Amazon EKS プラットフォームバージョン | リリースノート                                                                                                                                                                                                                      | リリース日 |
|------------------|--------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------|
|                  |                          | proxy は v1.30.10 に更新されました。AWSIAM オーセンティケーター が v0.6.29 に更新されました。Kubernetes 向け Amazon VPC CNI プラグイン は v1.19.2 に更新されました。CoreDNS が v1.11.4 に更新されました。AWS クラウドコントローラマネージャーが v1.30.8 に更新されました。Bottlerocket のバージョンを v1.34.0 に更新しました。 |       |

| Kubernetes バージョン | Amazon EKS プラットフォームバージョン | リリースノート                                                                                                                                                                                              | リリース日            |
|------------------|--------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------------|
| 1.30.7           | eks-local-outposts.2     | セキュリティの修正および機能強化が行われた新しいプラットフォームバージョン。kube-proxy は v1.30.7 に更新されました。AWSIAM オーセンティケーター が v0.6.28 に更新されました。Kubernetes 向け Amazon VPC CNI プラグイン は v1.19.0 に更新されました。Bottlerocket のバージョンを v1.29.0 に更新しました。 | 2025年1月10日       |
| 1.30.5           | eks-local-outposts.1     | アウトポスト のローカル Amazon EKS クラスター用 Kubernetes バージョン v1.30 の初回リリース。                                                                                                                                       | 2024 年 11 月 13 日 |

Kubernetes バージョン 1.29

以下のアドミッションコントローラーは、すべての 1.29 プラットフォームバージョンで有効です:  
CertificateApproval、CertificateSigning、CertificateSubjectRestriction、ClusterTr

| Kubernetes バージョン | Amazon EKS プラットフォームバージョン | リリースノート                                                                                                                                                                                                                           | リリース日           |
|------------------|--------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----------------|
| v1.29.14         | eks-local-outposts.6     | セキュリティの修正および機能強化が行われた新しいプラットフォームバージョン。kube-proxy は v1.29.14 に更新されました。Kubernetes 向け Amazon VPC CNI プラグインは v1.19.2 に更新されました。CoreDNS が v1.11.4 に更新されました。AWS クラウドコントローラマネージャーが v1.29.8 に更新されました。Bottlerocket のバージョンを v1.34.0 に更新しました。 | 2025 年 3 月 27 日 |
| v1.29.11         | eks-local-outposts.5     | セキュリティの修正および機能強化が行われた新しいプラットフォームバージョン。kube-proxy は v1.29.11 に更新されました。Kubernetes 向け Amazon VPC CNI プラグインは v1.19.0 に更新されました。CoreDNS                                                                                                 | 2025年1月10日      |



| Kubernetes バージョン | Amazon EKS プラットフォームバージョン | リリースノート                                                                                                                                        | リリース日            |
|------------------|--------------------------|------------------------------------------------------------------------------------------------------------------------------------------------|------------------|
|                  |                          | イメージを v1.11.3 に更新しました。Bottlerocket のバージョンを v1.29.0 に更新しました。                                                                                    |                  |
| 1.29.9           | eks-local-outposts.4     | セキュリティの修正および機能強化が行われた新しいプラットフォームバージョン。kube-proxy は v1.29.9 に更新されました。AWS IAM オーセンティケーターが v0.6.26 に更新されました。Bottlerocket のバージョンを v1.26.0 に更新しました。 | 2024 年 11 月 8 日  |
| 1.29.6           | eks-local-outposts.3     | セキュリティの修正および機能強化が行われた新しいプラットフォームバージョン。Bottlerocket のバージョンを v1.22.0 に更新しました。                                                                    | 2024 年 10 月 22 日 |

| Kubernetes バージョン | Amazon EKS プラットフォームバージョン | リリースノート                                                                     | リリース日           |
|------------------|--------------------------|-----------------------------------------------------------------------------|-----------------|
| 1.29.6           | eks-local-outposts.2     | セキュリティの修正および機能強化が行われた新しいプラットフォームバージョン。Bottlerocket のバージョンを v1.21.0 に更新しました。 | 2024 年 8 月 27 日 |
| 1.29.6           | eks-local-outposts.1     | アウトポスト のローカル Amazon EKS クラスター用 Kubernetes バージョン v1.29 の初回リリース。              | 2024 年 8 月 20 日 |

## Kubernetes バージョン 1.28

以下のアドミッションコントローラーは、すべての 1.28 プラットフォームバージョンで有効です: CertificateApproval、CertificateSigning、CertificateSubjectRestriction、ClusterTr

| Kubernetes バージョン | Amazon EKS プラットフォームバージョン | リリースノート                                                                              | リリース日           |
|------------------|--------------------------|--------------------------------------------------------------------------------------|-----------------|
| 1.28.15          | eks-local-outposts.13    | セキュリティの修正および機能強化が行われた新しいプラットフォームバージョン。kube-proxy を v1.28.15 ビルドに更新しました。Kubernetes 向け | 2025 年 3 月 27 日 |

| Kubernetes バージョン | Amazon EKS プラットフォームバージョン | リリースノート                                                                                                                                                         | リリース日      |
|------------------|--------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------|------------|
|                  |                          | Amazon VPC CNI プラグインが v1.19.0 に更新されました。AWSクラウドコントローラーマネージャーが v1.28.11 に更新されました。                                                                                 |            |
| 1.28.15          | eks-local-outposts.12    | セキュリティの修正および機能強化が行われた新しいプラットフォームバージョン。kube-proxy は v1.28.15 に更新されました。Kubernetes 向け Amazon VPC CNI プラグインは v1.19.0 に更新されました。Bottlerocket のバージョンを v1.29.0 に更新しました。 | 2025年1月10日 |

| Kubernetes バージョン | Amazon EKS プラットフォームバージョン | リリースノート                                                                                                                                                                        | リリース日            |
|------------------|--------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------------|
| 1.28.14          | eks-local-outposts.11    | セキュリティの修正および機能強化が行われた新しいプラットフォームバージョン。kube-proxy は v1.28.14 に更新されました。AWS IAM オートセンティケーターが v0.6.26 に更新されました。CoreDNS イメージを v1.11.1 に更新しました。Bottlerocket のバージョンを v1.26.0 に更新しました。 | 2024 年 11 月 8 日  |
| 1.28.10          | eks-local-outposts.10    | セキュリティの修正および機能強化が行われた新しいプラットフォームバージョン。Bottlerocket のバージョンを v1.22.0 に更新しました。                                                                                                    | 2024 年 10 月 22 日 |

| Kubernetes バージョン | Amazon EKS プラットフォームバージョン | リリースノート                                                                                                                                         | リリース日           |
|------------------|--------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------|-----------------|
| 1.28.10          | eks-local-outposts.9     | セキュリティの修正および機能強化が行われた新しいプラットフォームバージョン。Bottlerocket のバージョンを v1.21.0 に更新しました。                                                                     | 2024 年 8 月 27 日 |
| 1.28.10          | eks-local-outposts.8     | セキュリティの修正および機能強化が行われた新しいプラットフォームバージョン。                                                                                                          | 2024 年 7 月 30 日 |
| 1.28.10          | eks-local-outposts.6     | セキュリティの修正および機能強化が行われた新しいプラットフォームバージョン。kube-proxy は v1.28.10 に更新されました。AWS IAM オーセンティケーターが v0.6.20 に更新されました。Bottlerocket のバージョンを v1.20.2 に更新しました。 | 2024 年 6 月 19 日 |

| Kubernetes バージョン | Amazon EKS プラットフォームバージョン | リリースノート                                                                                                              | リリース日           |
|------------------|--------------------------|----------------------------------------------------------------------------------------------------------------------|-----------------|
| 1.28.6           | eks-local-outposts.5     | Bottlerocket バージョンを v1.19.3 に更新し、アウトポストでのローカルブートをサポートする最新のバグ修正を追加しました。                                               | 2024 年 4 月 18 日 |
| 1.28.6           | eks-local-outposts.4     | セキュリティの修正および機能強化が行われた新しいプラットフォームバージョン。アウトポストでサポートまたはローカルブートを復元しました。互換性のために Bottlerocket バージョンを v1.15.1 にダウングレードしました。 | 2024 年 4 月 2 日  |
| 1.28.6           | eks-local-outposts.3     | セキュリティの修正および機能強化が行われた新しいプラットフォームバージョン。                                                                               | 2024 年 3 月 22 日 |

| Kubernetes バージョン | Amazon EKS プラットフォームバージョン | リリースノート                                                                                                                                                                                                          | リリース日           |
|------------------|--------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----------------|
| 1.28.6           | eks-local-outposts.2     | セキュリティの修正および機能強化が行われた新しいプラットフォームバージョン。kube-proxy は v1.28.6 に更新されました。AWS IAM オートセンティケーターが v0.6.17 に更新されました。互換性の理由から、Amazon VPC CNI Kubernetes 用プラグインが v1.13.2 にダウングレードされました。BottleRocket のバージョンを v1.19.2 に更新しました。 | 2024 年 3 月 8 日  |
| 1.28.1           | eks-local-outposts.1     | アウトポストのローカル Amazon EKS クラスター用 Kubernetes バージョン v1.28 の初回リリース。                                                                                                                                                    | 2023 年 10 月 4 日 |

## Kubernetes バージョン 1.27

以下のアドミッションコントローラーは、すべての 1.27 プラットフォームバージョンで有効です: CertificateApproval、CertificateSigning、CertificateSubjectRestriction、ClusterTr

| Kubernetes バージョン | Amazon EKS プラットフォームバージョン | リリースノート                                                                                                                          | リリース日           |
|------------------|--------------------------|----------------------------------------------------------------------------------------------------------------------------------|-----------------|
| 1.27.16          | eks-local-outposts.13    | セキュリティの修正および機能強化が行われた新しいプラットフォームバージョン。Kubernetes 向け Amazon VPC CNI プラグインは v1.19.2 に更新されました。Bottlerocket のバージョンを v1.34.0 に更新しました。 | 2025 年 3 月 27 日 |
| 1.27.16          | eks-local-outposts.12    | セキュリティの修正および機能強化が行われた新しいプラットフォームバージョン。Kubernetes 向け Amazon VPC CNI プラグインは v1.19.0 に更新されました。Bottlerocket のバージョンを v1.29.0 に更新しました。 | 2025年1月10日      |
| 1.27.16          | eks-local-outposts.11    | セキュリティの修正および機能強化が行われた新しいプラットフォームバージョン。kube-proxy は v1.27.16 に更新されました。AWSIAM                                                      | 2024 年 11 月 8 日 |



| Kubernetes バージョン | Amazon EKS プラットフォームバージョン | リリースノート                                                                                          | リリース日            |
|------------------|--------------------------|--------------------------------------------------------------------------------------------------|------------------|
|                  |                          | オーセンティケーターが v0.6.26 に更新されました。CoreDNS イメージを v1.11.1 に更新しました。Bottlerocket のバージョンを v1.26.0 に更新しました。 |                  |
| 1.27.14          | eks-local-outposts.10    | セキュリティの修正および機能強化が行われた新しいプラットフォームバージョン。Bottlerocket のバージョンを v1.22.0 に更新しました。                      | 2024 年 10 月 22 日 |
| 1.27.14          | eks-local-outposts.9     | セキュリティの修正および機能強化が行われた新しいプラットフォームバージョン。Bottlerocket のバージョンを v1.21.0 に更新しました。                      | 2024 年 8 月 27 日  |
| 1.27.14          | eks-local-outposts.8     | セキュリティの修正および機能強化が行われた新しいプラットフォームバージョン。                                                           | 2024 年 7 月 30 日  |

| Kubernetes バージョン | Amazon EKS プラットフォームバージョン | リリースノート                                                                                                                                         | リリース日           |
|------------------|--------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------|-----------------|
| 1.27.14          | eks-local-outposts.6     | セキュリティの修正および機能強化が行われた新しいプラットフォームバージョン。kube-proxy は v1.27.14 に更新されました。AWSIAM オーセンティケーター が v0.6.20 に更新されました。Bottlerocket のバージョンを v1.20.2 に更新しました。 | 2024 年 6 月 19 日 |
| 1.27.10          | eks-local-outposts.5     | セキュリティの修正および機能強化が行われた新しいプラットフォームバージョン。                                                                                                          | 2024 年 4 月 2 日  |
| 1.27.10          | eks-local-outposts.4     | セキュリティの修正および機能強化が行われた新しいプラットフォームバージョン。kube-proxy は v1.27.10 に更新されました。AWSIAM オーセンティケーター が v0.6.17 に更新されました。Bottlerocket のバージョンを v1.19.2 に更新しました。 | 2024 年 3 月 22 日 |

| Kubernetes バージョン | Amazon EKS プラットフォームバージョン | リリースノート                                                                                                                   | リリース日           |
|------------------|--------------------------|---------------------------------------------------------------------------------------------------------------------------|-----------------|
| 1.27.3           | eks-local-outposts.3     | セキュリティの修正および機能強化が行われた新しいプラットフォームバージョン。kube-proxy は v1.27.3 に更新されました。Kubernetes 向け Amazon VPC CNI プラグインは v1.13.2 に更新されました。 | 2023 年 7 月 14 日 |
| 1.27.1           | eks-local-outposts.2     | CoreDNS イメージを v1.10.1 に更新しました。                                                                                            | 2023 年 6 月 22 日 |
| 1.27.1           | eks-local-outposts.1     | アウトポストのローカル Amazon EKS クラスター用 Kubernetes バージョン 1.27 の初回リリース。                                                              | 2023 年 5 月 30 日 |

## AWS Outposts で Amazon EKS クラスターの VPC とサブネットを作成する

ローカルクラスターを作成する際には、VPC と、Outposts で実行されるプライベートサブネットを少なくとも 1 つ指定します。このトピックでは、ローカルクラスターの VPC およびサブネットの要件と考慮事項について概要を説明します。

### VPC の要件と考慮事項

ローカルクラスターを作成する際には、指定する VPC が次の要件と考慮事項を満たす必要があります。

- VPC には、作成するローカルクラスター、任意のノード、およびその他の Kubernetes リソースで利用できるだけの十分な数の IP アドレスが必要です。使用する VPC に十分な IP アドレスがない場合は、使用可能な IP アドレスの数を増やしてください。この操作を行うには、VPC への[追加の Classless Inter-Domain Routing \(CIDR\) ブロックの関連付け](#)が必要になります。クラスターの作成前または作成後に、プライベート (RFC 1918) とパブリック (非 RFC 1918) の CIDR ブロックを VPC に関連付けることができます。クラスターで VPC に関連付けた CIDR ブロックが認識されるまでに、最大 5 時間かかることがあります。
- VPC に IP プレフィックスを割り当てたり、IPv6 CIDR ブロックを持たせることはできません。これらの制約があるため、[プレフィックスを使用して Amazon EKS ノードに割り当てる IP アドレスを増やす](#) および [the section called "IPv6"](#) でカバーされた情報は VPC には適用されません。
- VPC では DNS ホスト名と DNS 解決が有効になっています。これらの機能がないと、ローカルクラスターの作成は失敗し、機能を有効にしてクラスターを再作成する必要があります。詳細については、「Amazon VPC ユーザーガイド」の「[DNS attributes for your VPC](#)」(VPC の DNS 属性) を参照してください。
- ローカルネットワーク経由でローカルクラスターにアクセスするには、VPC が Outpost のローカルゲートウェイルートテーブルに関連付けられている必要があります。詳細については、「AWS Outposts ユーザーガイド」の「[VPC アソシエーション](#)」を参照してください。

## サブネットの要件と考慮事項

クラスターを作成する際には、少なくとも 1 つのプライベートサブネットを指定します。複数のサブネットを指定すると、Kubernetes コントロールプレーンインスタンスはサブネット全体に均等に分散されます。1 つ以上のサブネットを指定する場合、サブネットは同じ Outpost 上に存在する必要があります。また、相互に通信するために、サブネットには適切なルートとセキュリティグループ権限も必要です。ローカルクラスターの作成時に、指定するサブネットは次の要件を満たす必要があります。

- サブネットはすべて、同じロジカル Outpost 上に存在すること。
- サブネットには、Kubernetes コントロールプレーンインスタンスに対して、合計で 3 つ以上の使用可能な IP アドレスを用意してください。3 つのサブネットを指定する場合、各サブネットには少なくとも 1 つ以上の使用可能な IP アドレスが必要です。2 つのサブネットを指定する場合、各サブネットには少なくとも 2 つ以上の利用可能な IP アドレスが必要です。1 つのサブネットを指定する場合、サブネットには少なくとも 3 つ以上の利用可能な IP アドレスが必要です。
- ローカルネットワーク経由で Kubernetes API サーバーにアクセスするため、サブネットには Outpost ラックの[ローカルゲートウェイ](#)へのルートがあります。サブネットに Outpost ラックの

ローカルゲートウェイへのルートがない場合は、VPC 内から Kubernetes API サーバーと通信する必要があります。

- サブネットでは IP アドレスベースの命名を使用する必要があります。Amazon EC2 の [リソースベースの命名](#) は、Amazon EKS でサポートされていません。

## AWS サービスへのサブネットアクセス

Outposts 上のローカルクラスターのプライベートサブネットは、リージョンレベルの AWS サービスと通信する必要があります。アウトバウンドインターネットアクセスに [NAT ゲートウェイ](#) を使用するか、VPC 内ですべてのトラフィックをプライベートに保ちたい場合は、[インターフェイス VPC エンドポイント](#) を使用することによって、これを実現できます。

### NAT ゲートウェイの使用

Outposts 上のローカルクラスターのプライベートサブネットには、Outposts の親アベイラビリティゾーンにあるパブリックサブネット内の NAT ゲートウェイへのルートを持つ、関連付けられたルートテーブルが必要です。パブリックサブネットには、[インターネットゲートウェイ](#) へのルートが必要です。NAT ゲートウェイはアウトバウンドのインターネットアクセスを可能にし、インターネットから Outpost のインスタンスへの未承諾なインバウンド接続を防ぎます。

### インターフェイス VPC エンドポイントの使用

Outposts のローカルクラスターのプライベートサブネットにアウトバウンドインターネット接続がない場合、または VPC 内ですべてのトラフィックをプライベートに保ちたい場合は、クラスターを作成する前に、リージョンレベルのサブネットで次のインターフェイス VPC エンドポイントと [ゲートウェイエンドポイント](#) を作成する必要があります。

| エンドポイント                                        | エンドポイントタイプ |
|------------------------------------------------|------------|
| com.amazonaws. <i>region-code</i> .ssm         | インターフェイス   |
| com.amazonaws. <i>region-code</i> .ssmmessages | インターフェイス   |
| com.amazonaws. <i>region-code</i> .ec2messages | インターフェイス   |
| com.amazonaws. <i>region-code</i> .ec2         | インターフェイス   |

| エンドポイント                                                     | エンドポイントタイプ |
|-------------------------------------------------------------|------------|
| com.amazonaws. <i>region-code</i> <i>de</i> .secretsmanager | インターフェイス   |
| com.amazonaws. <i>region-code</i> .logs                     | インターフェイス   |
| com.amazonaws. <i>region-code</i> .sts                      | インターフェイス   |
| com.amazonaws. <i>region-code</i> .ecr.api                  | インターフェイス   |
| com.amazonaws. <i>region-code</i> .ecr.dkr                  | インターフェイス   |
| com.amazonaws. <i>region-code</i> .s3                       | ゲートウェイ     |

エンドポイントは次の要件を満たしている必要があります。

- Outpost の親アベイラビリティゾーンにあるプライベートサブネットに作成されている
- プライベートDNS 名が有効になっている
- プライベート Outpost サブネットの CIDR 範囲からのインバウンド HTTPS トラフィックを許可するセキュリティグループがアタッチされている

エンドポイントを作成すると料金が発生します。詳細については「[AWS プライベートリンク の料金](#)」を参照してください。Pod が他の AWS サービスにアクセスする必要がある場合は、追加のエンドポイントを作成する必要があります。エンドポイントの包括的なリストについては、「[AWS PrivateLink と統合する AWS のサービス](#)」を参照してください。

## 「VPC を作成する」

以下の AWS CloudFormation テンプレートのいずれかを使用して、前の要件を満たす VPC を作成できます。

- [テンプレート 1](#) — このテンプレートは、Outpost に 1 つのプライベートサブネット、また AWS リージョンに 1 つのパブリックサブネットを持つ VPC を作成します。プライベートサブネットには、AWS リージョンのパブリックサブネットにある NAT Gateway 経由でのインターネットへのルートがあります。このテンプレートを使用して、出カインターネットアクセスを持つサブネットにローカルクラスターを作成できます。

- [テンプレート 2](#) — このテンプレートは、Outpost に 1 つのプライベートサブネットと、入力または出力のインターネットアクセスがないサブネット (プライベートサブネットとも呼ばれる) にローカルクラスターを作成するために必要な最小限の VPC エンドポイントセットを含む VPC を作成します。

## AWS Outposts でローカル Amazon EKS クラスターを準備して、ネットワークの切断に備える

ローカルネットワークが AWS Cloud との接続を失った場合でも、Outpost にあるローカル Amazon EKS クラスターは引き続き使用できます。このトピックでは、ネットワークの切断とそれに関連する考慮事項に備えて、ローカルクラスターを準備する方法について説明します。

- ローカルクラスターにより、計画外の一時的なネットワーク切断中でも安定してオペレーションを継続できます。AWS Outposts は、データセンター内の AWS Cloud の拡張機能として機能する完全に接続された製品であることに変わりはありません。Outpost と AWS Cloud 間のネットワークが切断された場合には、接続の復元を試みることをお勧めします。手順については、「AWS Outposts ユーザーガイド」の「[AWS Outposts ラックネットワークのトラブルシューティング チェックリスト](#)」を参照してください。ローカルクラスター上の問題をトラブルシューティングする方法については、「[the section called “クラスターのトラブルシューティングを行う”](#)」を参照してください。
- Outposts は、Outpost の接続状態を監視するために使用できる `ConnectedStatus` 指標を出力します。詳細については、「AWS Outposts ユーザーガイド」の「[Outposts メトリクス](#)」を参照してください。
- ローカルクラスターでは、[AWS Identity and Access Management authenticator for Kubernetes](#) を使用する、デフォルトの認証メカニズムとして IAM を使用します。ネットワークの切断中には、IAM は使用できません。ですから、ローカルクラスターが、ネットワークの切断中にクラスターの接続に使用できる x.509 証明書を使用する代替認証メカニズムをサポートします。クラスターの x.509 証明書を取得して使用する方法については、「[the section called “ネットワーク切断中のローカルクラスターへの認証”](#)」を参照してください。
- ネットワークの切断中に Route 53 にアクセスできない場合は、オンプレミス環境上にあるローカル DNS サーバーを使用することを検討してください。Kubernetes コントロールプレーンインスタンスは静的 IP アドレスを使用します。ローカル DNS サーバーを使用する代わりに、エンドポイントのホスト名と IP アドレスを使用してクラスターに接続するホストを設定できます。詳細については、AWS Outposts ユーザーガイドの [DNS](#) を参照してください。



- ネットワークの切断中にアプリケーショントラフィックの増加が予想する場合は、クラウドに接続したときにクラスターに予備のコンピューティング容量をプロビジョニングできます。Amazon EC2 インスタンスは AWS Outposts の料金に含まれています。そのため、スペアインスタンスを実行しても AWS の使用コストには影響しません。
- ネットワークが切断されている間、ワークロードの作成、更新、スケーリングオペレーションを有効にするには、アプリケーションのコンテナイメージにローカルネットワークを介してアクセスでき、クラスターに十分な容量がある必要があります。ローカルクラスターはコンテナレジストリをホストしません。Pod がそれらのノードで以前に実行されていた場合、コンテナイメージはノードにキャッシュされます。アプリケーションのコンテナイメージをクラウドの Amazon ECR から通常にプルする場合は、ローカルキャッシュまたはレジストリの実行を検討してください。ローカルキャッシュまたはレジストリは、ネットワーク切断中にワークロードリソースの作成、更新、スケーリングオペレーションが必要になった場合に便利です。
- ローカルクラスターは、Amazon EBS を永続ボリュームのデフォルトストレージクラスとして使用し、Amazon EBS 永続ボリュームのライフサイクルを管理するために Amazon EBS CSI ドライバーを使用しています。ネットワークの切断中は、Amazon EBS によってバックアップされる Pod を作成、更新、またはスケーリングすることはできません。これらのオペレーションにはクラウド内の Amazon EBS API への呼び出しが必要だからです。ステートフルワークロードをローカルクラスターにデプロイしていて、ネットワークの切断中に作成、更新、またはスケーリングオペレーションが必要な場合は、別のストレージメカニズムの使用を検討してください。
- AWS Outposts が、関連する AWS リージョン内 API (Amazon EBS または Amazon S3 の API など) にアクセスできない場合、Amazon EBS スナップショットを作成または削除することはできません。
- ALB (Ingress) と AWS Certificate Manager (ACM) を統合すると、証明書がプッシュされ、AWS Outposts ALB コンピューティングインスタンスのメモリに保存されます。現在の TLS ターミネーションは、AWS リージョンとの接続が切断された場合でも引き続き機能します。このコンテキストでの変更操作は失敗します (新しい入力定義、新しい ACM ベースの証明書 API 操作、ALB コンピューティングスケール、証明書のローテーションなど)。詳細については、「AWS Certificate Manager ユーザーガイド」の「[マネージド型の証明書の更新のトラブルシューティング](#)」を参照してください。
- Amazon EKS コントロールプレーンのログは、ネットワークの切断中に Kubernetes コントロールプレーンインスタンスでローカルにキャッシュされます。再接続すると、ログは親 AWS リージョンの CloudWatch Logs に送信されます。[Prometheus](#)、[Grafana](#)、または Amazon EKS パートナーソリューションを使用して、Kubernetes API サーバーのメトリクスエンドポイントを使用して、またはログに Fluent Bit を使用してクラスターをローカルで監視できます。



- アプリケーショントラフィックに Outposts で AWS Load Balancer Controller を使用している場合、AWS Load Balancer Controller が前面にある既存の Pod は、ネットワークの切断中もトラフィックを受信し続けます。ネットワーク切断中に作成された新しい Pod は、Outpost が AWS クラウドに再接続されるまでトラフィックを受信しません。ネットワーク切断中のスケーリングのニーズに対応するために、AWS Cloud に接続している間はアプリケーションのレプリカ数を設定することを検討してください。
- Amazon VPC CNI plugin for Kubernetes は、デフォルトで [セカンダリ IP モード](#) になります。WARM\_ENI\_TARGET=1 で構成されているため、プラグインは使用可能な IP アドレスの「完全な伸縮性ネットワークインターフェイス」を維持できます。接続されていない状態でのスケーリングのニーズに応じて、WARM\_ENI\_TARGET、WARM\_IP\_TARGET、MINIMUM\_IP\_TARGET の値を変更することを検討してください。詳細については、「GitHub」の「プラグインの [readme](#) ファイル」を参照してください。各インスタンスタイプによりサポートされる Pod の最大数のリストについては、GitHub の [eni-max-pods.txt](#) ファイルを参照してください。

## ネットワーク切断中のローカルクラスターへの認証

AWS Identity and Access Management (IAM) は、ネットワークの切断中は使用できません。切断されている間は、IAM 認証情報を使用してローカルクラスターを認証することはできません。しかしながら、切断時に x509 証明書を使用して、ローカルネットワークを介してクラスターに接続できます。切断時に使用するクライアント X509 証明書をダウンロードして保存する必要があります。このトピックでは、証明書を作成して使用し、クラスターが接続されていない状態のときにクラスターを認証する方法を説明します。

### 1. 証明書署名リクエストを作成します。

#### a. 証明書署名リクエストを生成します。

```
openssl req -new -newkey rsa:4096 -nodes -days 365 \
-keyout admin.key -out admin.csr -subj "/CN=admin"
```

#### b. Kubernetes で証明書署名リクエストを作成します。

```
BASE64_CSR=$(cat admin.csr | base64 -w 0)
cat << EOF > admin-csr.yaml
apiVersion: certificates.k8s.io/v1
kind: CertificateSigningRequest
metadata:
 name: admin-csr
spec:
```

```
signerName: kubernetes.io/kube-apiserver-client
request: ${BASE64_CSR}
usages:
- client auth
EOF
```

2. `kubectl` を使用して証明書署名リクエストを作成します。

```
kubectl create -f admin-csr.yaml
```

3. 証明書登録リクエストのステータスを確認します。

```
kubectl get csr admin-csr
```

出力例は次のとおりです。

| NAME      | AGE | REQUESTOR        | CONDITION |
|-----------|-----|------------------|-----------|
| admin-csr | 11m | kubernetes-admin | Pending   |

Kubernetes が証明書署名リクエストを作成しました。

4. 証明書署名リクエストを承認します。

```
kubectl certificate approve admin-csr
```

5. 証明書署名リクエストの承認のステータスを再確認します。

```
kubectl get csr admin-csr
```

出力例は次のとおりです。

| NAME      | AGE | REQUESTOR        | CONDITION |
|-----------|-----|------------------|-----------|
| admin-csr | 11m | kubernetes-admin | Approved  |

6. 証明書を取得して検証します。

- a. 証明書を取得する。

```
kubectl get csr admin-csr -o jsonpath='{.status.certificate}' | base64 --decode >
admin.crt
```

- b. 証明書を検証する。

```
cat admin.crt
```

7. admin ユーザーのクラスターロールバインディングを作成します。

```
kubectl create clusterrolebinding admin --clusterrole=cluster-admin \
 --user=admin --group=system:masters
```

8. 接続されていない状態のユーザスコープの kubeconfig を生成します。

ダウンロードした admin 証明書を使用して kubeconfig ファイルを生成できます。次のコマンドの *my-cluster* と *apiserver-endpoint* を置き換えます。

```
aws eks describe-cluster --name my-cluster \
 --query "cluster.certificateAuthority" \
 --output text | base64 --decode > ca.crt
```

```
kubectl config --kubeconfig admin.kubeconfig set-cluster my-cluster \
 --certificate-authority=ca.crt --server apiserver-endpoint --embed-certs
```

```
kubectl config --kubeconfig admin.kubeconfig set-credentials admin \
 --client-certificate=admin.crt --client-key=admin.key --embed-certs
```

```
kubectl config --kubeconfig admin.kubeconfig set-context admin@my-cluster \
 --cluster my-cluster --user admin
```

```
kubectl config --kubeconfig admin.kubeconfig use-context admin@my-cluster
```

9. kubeconfig ファイルを表示する。

```
kubectl get nodes --kubeconfig admin.kubeconfig
```

10. Outpost で既にサービスが稼働している場合は、この手順をスキップしてください。Outpost で実行されているサービスが Amazon EKS のみで、その Outpost が現在本番環境ではない場合は、ネットワークの切断をシミュレートできます。ローカルクラスターで本番環境に入る前に、切断をシミュレートして、接続されていない状態でもクラスターにアクセスできることを確認できます。

- a. Outpostを AWS リージョンに接続するネットワークデバイスにファイアウォールルールを適用します。これにより、Outpost のサービスリンクが切断されます。新しいインスタンスを作成することはできません。現在実行中のインスタンスは、AWS リージョンおよびインターネットへの接続を失います。
- b. x509 証明書を使用して、切断時にローカルクラスターへの接続をテストできます。kubeconfig を前の手順で作成した admin.kubeconfig に必ず変更してください。my-cluster の部分は、お客様のローカルクラスター名に置き換えます。

```
kubect1 config use-context admin@my-cluster --kubeconfig admin.kubeconfig
```

ローカルクラスターが接続されていない状態で問題が発生した場合は、サポートチケットを開くことをお勧めします。

## キャパシティに関する考慮事項に基づいて、AWS Outposts で Amazon EKS クラスターのインスタンスタイプとプレースメントグループを選択する

このトピックでは、Kubernetes コントロールプレーンインスタンスタイプを選択し、(オプションで) プレースメントグループを使用して Outpost のローカル Amazon EKS クラスターの高可用性要件を満たすためのガイダンスを提供します。

Outposts でローカルクラスターの Kubernetes コントロールプレーンに使用するインスタンスタイプ (m5、c5、r5 など) を選択する前に、Outpost 設定で使用可能なインスタンスタイプを確認します。使用可能なインスタンスタイプを特定したら、ワークロードに必要なノード数に基づいてインスタンスサイズ (large、xlarge、2xlarge など) を選択します。次のテーブルは、インスタンスサイズを選択する際のレコメンデーションを示しています。

### Note

インスタンスサイズは Outposts にスロットされている必要があります。ローカルクラスターの存続期間中、Outposts で使用可能なサイズの 3 つのインスタンス用に十分な容量を確保してください。使用可能な Amazon EC2 インスタンスタイプのリストについては、「[AWS Outposts ラックの特徴](#)」の「コンピューティングとストレージ」セクションを参照してください。

| ノードの数。    | Kubernetes コントロールプレーンのインスタンスサイズ |
|-----------|---------------------------------|
| 1 ~ 20    | large                           |
| 21 ~ 100  | xlarge                          |
| 101 ~ 250 | 2xlarge                         |
| 251 ~ 500 | 4xlarge                         |

Kubernetes コントロールプレーンのストレージには、etcd に要求される IOPS を満たすために、ローカルクラスターごとに 246 GB の Amazon EBS ストレージが必要です。ローカルクラスターの作成時に、Amazon EBS ボリュームは自動的にプロビジョニングされます。

## コントロールプレーンの配置

`OutpostConfig.ControlPlanePlacement.GroupName` プロパティでプレイメントグループを指定しない場合、Kubernetes コントロールプレーン用にプロビジョニングされた Amazon EC2 インスタンスは、Outpost で利用可能な基盤となる容量全体にわたって、特定のハードウェア配置の強制を受けません。

プレイメントグループを使用すると、Outpost のローカル Amazon EKS クラスターの高可用性要件を満たすことができます。クラスターの作成中にプレイメントグループを指定することで、Kubernetes コントロールプレーンインスタンスの配置に影響を与えます。インスタンスは基盤となる独立したハードウェア (ラックまたはホスト) に分散されるため、ハードウェア障害発生時のインスタンス間の影響が最小限に抑えられます。

設定できるスプレッドのタイプは、デプロイの Outpost ラックの数によって異なります。

- 単一の論理的な Outpost に 1 つまたは 2 つの物理ラックを使用するデプロイ - Kubernetes コントロールプレーンインスタンス用に選択したインスタンスタイプで構成されたホストが少なくとも 3 つ必要です。ホストレベルのスプレッドを使用するスプレッドプレイメントグループにより、すべての Kubernetes コントロールプレーンインスタンスが、Outpost デプロイで使用可能な基盤となるラック内の個別のホストで実行されるようになります。
- 単一の論理的な Outpost に 3 つ以上の物理ラックを使用するデプロイ - Kubernetes コントロールプレーンインスタンス用に選択したインスタンスタイプで構成されたホストが少なくとも 3 つ必要です。ラックレベルスプレッドを使用するスプレッドプレイメントグループにより、すべての

Kubernetes コントロールプレーンインスタンスが、Outpost デプロイ内の個別のラックで実行されるようになります。または、前のオプションで説明したように、ホストレベルのスプレッドプレイスメントグループを使用することもできます。

希望するプレイスメントグループの作成はお客様の責任となります。CreateCluster API を呼び出すときにプレイスメントグループを指定します。プレイスメントグループとその作成方法の詳細については、「Amazon EC2 ユーザーガイド」の「[プレイスメントグループ](#)」を参照してください。

- プレイスメントグループを指定する場合、ローカルの Amazon EKS クラスターを正常に作成するには、Outpost に使用可能なスロット容量が必要です。容量は、ホストタイプとラックスプレッドタイプのどちらを使用するかによって異なります。十分な容量がない場合、クラスターは Creating 状態のままになります。[DescribeCluster](#) API レスポンスのヘルスフィールドで Insufficient Capacity Error を確認できます。作成プロセスを進めるには、容量を解放する必要があります。
- Amazon EKS ローカルクラスタープラットフォームとバージョンの更新中に、クラスターの Kubernetes コントロールプレーンインスタンスはローリング更新戦略を使用して新しいインスタンスに置き換えられます。この交換プロセス中に、各コントロールプレーンインスタンスは終了し、それぞれのスロットが解放されます。代わりに新しい更新済みインスタンスがプロビジョニングされます。更新されたインスタンスは、リリースされたスロットに配置される可能性があります。スロットが別の無関係なインスタンスによって消費され、必要なスプレッドポロジータンを満たす容量が残っていない場合、クラスターは Updating 状態のままになります。[DescribeCluster](#) API レスポンスのヘルスフィールドでそれぞれの Insufficient Capacity Error を確認できます。更新プロセスを進めて以前の高可用性レベルを再設定できるように、容量を解放する必要があります。
- AWS リージョンごとにアカウントあたり、最大 500 個のプレイスメントグループを作成できます。詳細については、「Amazon EC2 ユーザーガイド」の「[一般的なルールと制限](#)」を参照してください。

## AWS Outposts でローカル Amazon EKS クラスターをトラブルシューティングする

このトピックでは、ローカルクラスターの使用中に表示される可能性がある一般的なエラーとそのトラブルシューティング方法について説明します。ローカルクラスターはクラウド内の Amazon EKS クラスターと似ていますが、Amazon EKS による管理方法にはいくつかの違いがあります。

### ⚠ Important

AWS サポートから明示的に指示されていない限り、Outpost で実行されているマネージド型 EKS ローカルクラスターの Kubernetes コントロールプレーンインスタンスを終了しないでください。これらのインスタンスを終了すると、複数のインスタンスが同時に終了した場合にローカルクラスターが失われるなど、ローカルクラスターサービスの可用性にリスクが伴います。EKS ローカルクラスターの Kubernetes コントロールプレーンインスタンスは、EC2 インスタンスコンソールでは `eks-local:controlplane-name` タグで識別します。

## API の動作

ローカルクラスターは Amazon EKS API を通して作成されますが、非同期で実行されます。つまり、Amazon EKS API へのリクエストはローカルクラスターに対してすぐに返されます。ただし、これらのリクエストは成功し、入力検証エラーにより迅速に失敗するか、あるいは失敗して説明入りの検証エラーが発生する可能性があります。この動作は、Kubernetes API と同様です。

ローカルクラスターは FAILED ステータスに移行しません。Amazon EKS は、クラスターの状態をユーザーが要求した望ましい状態と一致させようと継続的に試みます。その結果、根本的な問題が解決されるまで、長期間にわたってローカルクラスターが CREATING 状態のままになる可能性があります。

クラスターヘルスフィールドについて説明します

[describe-cluster](#) Amazon EKS AWS CLI コマンドを使用して、ローカルクラスターの問題を検出できます。ローカルクラスターの問題は、`describe-cluster` コマンドの応答の `cluster.health` フィールドによって明らかになります。このフィールドに含まれるメッセージには、エラーコード、説明メッセージ、および関連するリソース ID が含まれます。この情報は、Amazon EKS API および AWS CLI のみから利用可能です。次の例では、*my-cluster* をローカルクラスターの名前に置き換えます。

```
aws eks describe-cluster --name my-cluster --query 'cluster.health'
```

出力例は次のとおりです。

```
{
 "issues": [
 {
```

```

 "code": "ConfigurationConflict",
 "message": "The instance type 'm5.large' is not supported in Outpost 'my-outpost-arn'.",
 "resourceIds": [
 "my-cluster-arn"
]
 }
]
}

```

問題が修復できない場合は、ローカルクラスターを削除して新しいクラスターを作成する必要がある場合があります。たとえば、Outpost で利用できないインスタンスタイプでクラスターをプロビジョニングしようとしている場合などです。下表は、一般的なヘルス関連のエラーを示しています。

| エラーシナリオ                                    | Code                  | メッセージ                                                                                             | ResourceIds       |
|--------------------------------------------|-----------------------|---------------------------------------------------------------------------------------------------|-------------------|
| 指定されたサブネットが見つかりませんでした。                     | ResourceNotFound      | The subnet ID <i>subnet-id</i> does not exist                                                     | 指定されたすべてのサブネット ID |
| 指定されたサブネットが同じ VPC に属していません。                | ConfigurationConflict | Subnets specified must belong to the same VPC                                                     | 指定されたすべてのサブネット ID |
| 指定されたサブネットの一部が、指定された Outpost に属していません。     | ConfigurationConflict | Subnet <i>subnet-id</i> expected to be in <i>outpost-arn</i> , but is in <i>other-outpost-arn</i> | 問題のあるサブネット ID     |
| 指定されたサブネットの中には、どの Outpost にも属していないものがあります。 | ConfigurationConflict | Subnet <i>subnet-id</i> is not part of any Outpost                                                | 問題のあるサブネット ID     |
| 指定されたサブネットの中には、コント                         | ResourceLimitExceeded | The specified subnet does                                                                         | 問題のあるサブネット ID     |



| エラーシナリオ                                                                                                                                                               | Code                  | メッセージ                                                                        | ResourceIds |
|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----------------------|------------------------------------------------------------------------------|-------------|
| ロールプレーンインスタンス用のエラスティックネットワークインターフェイスを作成するだけの十分な空きアドレスがないものがあります。                                                                                                      |                       | not have enough free addresses to satisfy the request.                       |             |
| 指定されたコントロールプレーンインスタンスタイプは、お使いの Outpost でサポートされていません。                                                                                                                  | ConfigurationConflict | The instance type <i>type</i> is not supported in Outpost <i>outpost-arn</i> | クラスター ARN   |
| コントロールプレーン Amazon EC2 インスタンスを終了したか、run-instance が成功し、観察された状態が Terminated に変わりました。これは、Outpost が再接続され、Amazon EBS の内部エラーが原因で Amazon EC2 内部ワークフローが失敗した後、一定期間発生する可能性があります。 | InternalFailure       | EC2 instance state "Terminated" is unexpected                                | クラスター ARN   |

| エラーシナリオ                                                                                                                                                            | Code                  | メッセージ                                                                                                          | ResourceIds          |
|--------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----------------------|----------------------------------------------------------------------------------------------------------------|----------------------|
| Outpost の容量が不足しています。これは、クラスターの作成中に Outpost が AWS リージョンから切断された場合にも発生する可能性があります。                                                                                     | ResourceLimitExceeded | There is not enough capacity on the Outpost to launch or start the instance.                                   | クラスター ARN            |
| アカウントがセキュリティグループの制限を超えています。                                                                                                                                        | ResourceLimitExceeded | Amazon EC2 API から返されたエラーメッセージ                                                                                  | ターゲット VPC ID         |
| アカウントがエラスティックネットワークインターフェイスの制限を超えています。                                                                                                                             | ResourceLimitExceeded | Amazon EC2 API から返されたエラーメッセージ                                                                                  | ターゲットサブネット ID        |
| AWS Systems Manager を介してコントロールプレーンインスタンスにアクセスできません。解決策については、「 <a href="#">the section called “AWS Systems Manager を介してコントロールプレーンインスタンスにアクセスできません。”</a> 」を参照してください。 | ClusterUnreachable    | Amazon EKS コントロールプレーンインスタンスには SSM 経由ではアクセスできません。SSM とネットワーク設定を確認し、EKS on Outposts のトラブルシューティングドキュメントを参照してください。 | Amazon EC2 インスタンス ID |

| エラーシナリオ                                                                                | Code                           | メッセージ                         | ResourceIds                   |
|----------------------------------------------------------------------------------------|--------------------------------|-------------------------------|-------------------------------|
| マネージドセキュリティグループまたはエラスティックネットワークインターフェイスの詳細を取得中に、エラーが発生しました。                            | Amazon EC2 クライアントエラーコードに基づきます。 | Amazon EC2 API から返されたエラーメッセージ | すべてのマネージドセキュリティグループ ID        |
| セキュリティグループのインGRESSルールの承認または取り消す際にエラーが発生しました。これは、クラスターとコントロールプレーンの両方のセキュリティグループに適用されます。 | Amazon EC2 クライアントエラーコードに基づきます。 | Amazon EC2 API から返されたエラーメッセージ | 問題のあるセキュリティグループの ID           |
| コントロールプレーンインスタンスのエラスティックネットワークインターフェイスの削除中にエラーが発生しました。                                 | Amazon EC2 クライアントエラーコードに基づきます。 | Amazon EC2 API から返されたエラーメッセージ | 問題のあるエラスティックネットワークインターフェイス ID |

次の表は、describe-cluster 応答のヘルスフィールドに表示される他の AWS サービスからのエラーを示しています。

| Amazon EC2 エラーコード | クラスターヘルス問題コード | 説明                                                  |
|-------------------|---------------|-----------------------------------------------------|
| AuthFailure       | AccessDenied  | このエラーは、さまざまな理由で発生する可能性があります。最も一般的な理由は、サービスにリンクされたロー |

| Amazon EC2 エラーコード          | クラスターヘルス問題コード    | 説明                                                                                                                                                                  |
|----------------------------|------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|                            |                  | ルポリシーの範囲を狭めるためにサービスが使用するタグが、コントロールプレーンインスタンスから誤って削除された場合に発生するのです。この場合、Amazon EKS はこれらの AWS リソースを管理および監視できなくなります。                                                    |
| UnauthorizedOperation      | AccessDenied     | このエラーは、さまざまな理由で発生する可能性があります。最も一般的な理由は、サービスにリンクされたロールポリシーの範囲を狭めるためにサービスが使用するタグが、コントロールプレーンインスタンスから誤って削除された場合に発生するのです。この場合、Amazon EKS はこれらの AWS リソースを管理および監視できなくなります。 |
| InvalidSubnetID.NotFound   | ResourceNotFound | このエラーは、セキュリティグループのインGRESルールサブネット ID が見つからない場合に発生します。                                                                                                                |
| InvalidPermission.NotFound | ResourceNotFound | このエラーは、セキュリティグループのインGRESルールの権限が正しくない場合に発生します。                                                                                                                       |

| Amazon EC2 エラーコード                  | クラスターヘルス問題コード         | 説明                                                            |
|------------------------------------|-----------------------|---------------------------------------------------------------|
| InvalidGroup.NotFound              | ResourceNotFound      | このエラーは、セキュリティグループのイングレスルールのグループが見つからない場合に発生します。               |
| InvalidNetworkInterfaceID.NotFound | ResourceNotFound      | このエラーは、セキュリティグループのイングレスルールのネットワークインターフェイス ID が見つからない場合に発生します。 |
| InsufficientFreeAddressesInSubnet  | ResourceLimitExceeded | このエラーは、サブネットリソースのクォータを超えたときに発生します。                            |
| InsufficientCapacityOnOutpost      | ResourceLimitExceeded | このエラーは、outpost のクォータを超えたときに発生します。                             |
| NetworkInterfaceLimitExceeded      | ResourceLimitExceeded | このエラーは、エラスティックネットワークインターフェイスのクォータを超えた場合に発生します。                |
| SecurityGroupLimitExceeded         | ResourceLimitExceeded | このエラーは、セキュリティグループのクォータを超えたときに発生します。                           |

| Amazon EC2 エラーコード     | クラスターヘルス問題コード         | 説明                                                                                                                                                                                                                                                                                                                                                                           |
|-----------------------|-----------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| VcpuLimitExceeded     | ResourceLimitExceeded | Amazon EC2 インスタンスを新規アカウントで作成するときに発生します。エラーは次のようなものになります: 「You have requested more vCPU capacity than your current vCPU limit of 32 allows for the instance bucket that the specified instance type belongs to. Please visit <a href="http://aws.amazon.com/contact-us/ec2-request">http://aws.amazon.com/contact-us/ec2-request</a> to request an adjustment to this limit." |
| InvalidParameterValue | ConfigurationConflict | Amazon EC2 は、指定されたインスタンスタイプが Outpost でサポートされていない場合、このエラーコードを返します。                                                                                                                                                                                                                                                                                                            |
| その他のすべての障害            | InternalFailure       | なし                                                                                                                                                                                                                                                                                                                                                                           |

## クラスターを作成または変更できません

ローカルクラスターには、クラウド内にホストされている Amazon EKS クラスターとは異なるアクセス権限とポリシーが必要です。クラスターの作成に失敗し InvalidPermissions エラーが表示される場合は、使用しているクラスターロールに [AmazonEKSLocalOutpostClusterPolicy](#) マネージドポリシーがアタッチされているかどうかをもう一度確認します。その他すべての API 呼び出しには、クラウド内の Amazon EKS クラスターと同じ権限セットが必要です。

## クラスターが **CREATING** の状態でスタックしています

ローカルクラスターの作成にかかる時間は、いくつかの要因によって異なります。ネットワーク設定、Outpost の設定、およびクラスターの設定などが要因として考えられます。通常、ローカルクラスターは 15 ~ 20 分以内に作成され、ACTIVE ステータスに変わります。ローカルクラスターが CREATING 状態を維持する場合は、`describe-cluster` を呼び出すと `cluster.health` 出力フィールドに原因に関する情報が表示されます。

次は、最も一般的な問題を示しています。

- クラスターが Systems Manager のある AWS リージョンからコントロールプレーンインスタンスに接続できません。地域内の踏み台ホストから `aws ssm start-session --target instance-id` を呼び出すことでこれを検証できます。このコマンドがうまくいかない場合は、Systems Manager がコントロールプレーンインスタンスで実行されているかどうかを確認します。または、別の回避策として、クラスターを削除して再度作成することもできます。
- EBS ボリュームの KMS キーのアクセス許可が原因で、コントロールプレーンインスタンスの作成に失敗します。暗号化された EBS ボリュームのユーザーマネージド KMS キーでは、キーにアクセスできない場合、コントロールプレーンインスタンスは終了します。インスタンスが終了した場合、AWS マネージド KMS キーに切り替えるか、ユーザーマネージドキーポリシーがクラスターロールに必要なアクセス許可を付与していることを確認します。
- Systems Manager のコントロールプレーンインスタンスが、インターネットにアクセスできない可能性があります。クラスターの作成時に指定したサブネットに NAT ゲートウェイとインターネットゲートウェイを備えた VPC があるかどうかを確認します。VPC 到達可能性アナライザーを使用して、コントロールプレーンインスタンスがインターネットゲートウェイに到達できることを確認します。詳細については、「[VPC Reachability Analyzer の開始方法](#)」を参照してください。
- 指定したロール ARN にポリシーがありません。[AWS マネージドポリシー: AmazonEKSLocalOutpostClusterPolicy](#) がロールから削除されたかどうかを確認します。これは、AWS CloudFormation スタックの設定が間違っている場合にも発生する可能性があります。
- 指定されているすべてのサブネットが同じ Outpost に関連付けられており、相互に到達できる必要があります。クラスターの作成時に複数のサブネットを指定すると、Amazon EKS はコントロールプレーンインスタンスを複数のサブネットに分散させようとします。
- Amazon EKS マネージド セキュリティグループは、エラスティックネットワークインターフェイスに適用されます。しかしながら、NACL ファイアウォールルールなどの他の設定要素が、エラスティックネットワークインターフェイスのルールと競合する可能性があります。

VPC とサブネット DNS の設定が誤っているか、欠落しています

「[AWS Outposts で Amazon EKS クラスターの VPC とサブネットを作成する](#)」を確認します。

クラスターが **UPDATING** の状態でスタックしています

Amazon EKS は既存のすべてのローカルクラスターを、対応する Kubernetes マイナーバージョン用の最新のプラットフォームバージョンに自動的にアップデートします。プラットフォームバージョンの詳細については、「[the section called “EKS プラットフォームのバージョン”](#)」を参照してください。

プラットフォームバージョンの自動ロールアウト中に、クラスターのステータスが UPDATING に変わります。更新プロセスでは、すべての Kubernetes コントロールプレーンインスタンスを、それぞれの Kubernetes マイナーバージョン用にリリースされた最新のセキュリティパッチとバグ修正を含む新しいインスタンスに置き換えます。一般的に、ローカルクラスタープラットフォームの更新プロセスは 30 分以内に完了し、クラスターは ACTIVE ステータスに戻ります。ローカルクラスターが長時間 UPDATING 状態になる場合は、`describe-cluster` を呼び出すと `cluster.health` 出力フィールドで原因に関する情報を確認できます。

Amazon EKS は、ローカルクラスターの可用性を維持し、サービスの中断を防ぐために、3 つの Kubernetes コントロールプレーンインスタンスのうち少なくとも 2 つが正常で運用可能なクラスターノードであるようにします。ローカルクラスターが UPDATING 状態で停止している場合、通常は、プロセスが続行された場合に 2 つのインスタンスの最小可用性が保証されないインフラストラクチャまたは設定の問題が原因です。したがって、ローカルクラスターサービスの中断を防ぐために、更新プロセスの進行が停止します。

更新プロセスを完了し、3 つの Kubernetes コントロールプレーンインスタンスの高可用性でローカルクラスターを ACTIVE に戻すことができるように、UPDATING 状態で停止しているローカルクラスターをトラブルシューティングし、根本原因に対処することが重要です。

AWS サポートから明示的に指示されていない限り、Outposts でマネージド型 EKS ローカルクラスターの Kubernetes インスタンスを終了しないでください。これは、別のコントロールプレーンノードが完全には正常ではなく、間違ったインスタンスを終了すると、サービスの中断やローカルクラスターのデータ損失のリスクが生じる可能性が高いため、ローカルクラスターが UPDATING 状態で停止した場合、特に重要です。

次は、最も一般的な問題を示しています。

- ローカルクラスターが最初に作成された以降のネットワーク設定の変更により、1 つまたは複数のコントロールプレーンインスタンスが System Manager に接続できません。地域内の踏み台ホストから `aws ssm start-session --target instance-id` を呼び出すことでこれを検証で



きます。このコマンドがうまくいかない場合は、Systems Manager がコントロールプレーンインスタンスで実行されているかどうかを確認します。

- EBS ボリュームの KMS キーのアクセス許可が原因で、新しいコントロールプレーンインスタンスを作成できません。暗号化された EBS ボリュームのユーザーマネージド KMS キーでは、キーにアクセスできない場合、コントロールプレーンインスタンスは終了します。インスタンスが終了した場合、AWS マネージド KMS キーに切り替えるか、ユーザーマネージドキーポリシーがクラスターロールに必要なアクセス許可を付与していることを確認します。
- Systems Manager のコントロールプレーンインスタンスが、インターネットにアクセスできない可能性があります。クラスターの作成時に指定したサブネットに NAT ゲートウェイとインターネットゲートウェイを備えた VPC があるかどうかを確認します。VPC 到達可能性アナライザーを使用して、コントロールプレーンインスタンスがインターネットゲートウェイに到達できることを確認します。詳細については、「[VPC Reachability Analyzer の開始方法](#)」を参照してください。プライベートネットワークにアウトバウンドインターネット接続がない場合は、必要なすべての VPC エンドポイントとゲートウェイエンドポイントがクラスターのリージョンサブネットにまだ存在することを確認してください (「[the section called “AWS サービスへのサブネットアクセス”](#)」を参照)。
- 指定したロール ARN にポリシーがありません。[AWS managed policy: AmazonEKSLocalOutpostClusterPolicy](#) がロールから削除されたかどうかを確認します。
- 新しい Kubernetes コントロールプレーンインスタンスのいずれかで、予期しないブートストラップ障害が発生した可能性があります。この例外的なケースのトラブルシューティングとログ収集に関する詳細なガイダンスについては、[AWS サポートセンター](#)にチケットを提出してください。

## ノードをクラスターに結合できない

- AMI に関する問題:
  - サポートされていない AMI を使用しています。[最適化された Amazon Linux AMI を使用してノードを作成する](#) Amazon EKS 最適化 Amazon Linux には、[v20220620](#) 以降を使用する必要があります。
  - AWS CloudFormation テンプレートを使用してノードを作成した場合、サポートされていない AMI を使用していないことを確認します。
- AWS IAM Authenticator ConfigMap が見つからない - 見つからない場合は、作成する必要があります。詳細については、「[the section called “aws-auth ConfigMap をクラスターに適用する”](#)」を参照してください。
- 間違ったセキュリティグループが使用されている - ワーカーノードのセキュリティグループには、必ず `eks-cluster-sg-cluster-name-uniqueid` を使用してください。スタックが使用され

るたびに、選択したセキュリティグループは AWS CloudFormation によって変更され、新しいセキュリティグループを使用できるようになります。

- 予期しないプライベートリンクの VPC 手順に従う - CA データが間違っている (--b64-cluster-ca) または API エンドポイント (--apiserver-endpoint) が渡されました。

## ログの収集

Outpost が関連付けられている AWS リージョンから切断された場合でも、Kubernetes クラスターは依然として正常に動作し続けるはずです。ただし、クラスターが正常に機能しない場合は、「[AWS Outposts でローカル Amazon EKS クラスターを準備して、ネットワークの切断に備える](#)」にあるトラブルシューティング手順に従ってください。他の問題が発生した場合は、AWS サポートにお問い合わせください。AWS サポートからログ収集ツールをダウンロードして実行する方法を説明します。この方法により、Kubernetes クラスターコントロールプレーンインスタンスからログを収集し、詳細な調査のために AWS サポートに送信できます。

AWS Systems Manager を介してコントロールプレーンインスタンスにアクセスできません。

Amazon EKS コントロールプレーンインスタンスが AWS Systems Manager (Systems Manager) を通じてアクセスできない場合、Amazon EKS はクラスターに対して次のエラーを表示します。

```
Amazon EKS control plane instances are not reachable through SSM. Please verify your SSM and network configuration, and reference the EKS on Outposts troubleshooting documentation.
```

この問題を解決するには、VPC とサブネットが「[AWS Outposts で Amazon EKS クラスターの VPC とサブネットを作成する](#)」の要件を満たしており、AWS Systems Manager ユーザーガイドの「[Session Manager の設定](#)」にある手順が完了していることを確認してください。

## AWS アウトポスト で Amazon Linux ノードを作成する

このトピックは Amazon EKS クラスターに登録されている アウトポスト 上の Amazon Linux ノードの自動スケーリンググループを起動する方法を説明します。クラスターは AWS クラウド または アウトポスト に置くことができます。

- 既存の アウトポスト。詳細については「[AWS アウトポスト とは](#)」を参照してください。
- 既存の Amazon EKS クラスター。AWS クラウド にクラスターをデプロイするには「[the section called “クラスターを作成する”](#)」を参照してください。アウトポスト にクラスターをデプロイするには「[the section called “ローカルクラスターを実行する”](#)」を参照してください。

- AWS クラウド のクラスターにノードを作成しており、AWS アウトポスト、AWS 波長、または AWS ローカルゾーン が有効になっている AWS リージョンにサブネットがあるとしします。この場合、クラスターを作成したときに、これらのサブネットが渡されていない必要があります。アウトポスト 上のクラスターにノードを作成する場合はクラスターの作成時に アウトポスト サブネットを渡しておく必要があります。
- (AWS クラウド上のクラスターに推奨) 必要な IAM ポリシーがアタッチされた独自の IAM ロールで設定された Amazon VPC CNI plugin for Kubernetes アドオン。詳細については、「[the section called “IRSA 用の設定”](#)」を参照してください。ローカルクラスターはサービスアカウントの IAM ロールをサポートしていません。

eksctl または AWS Management Console (AWS クラウドFormation テンプレートを使用) で、セルフマネージド型の Amazon Linux ノードグループを作成できます。Terraform を使用することもできます。

このページで説明されている以下のツールを使用して、ローカルクラスターのセルフマネージド型ノードグループを作成できます。

- [the section called “eksctl”](#)
- [the section called “AWS Management Console”](#)

#### Important

- セルフマネージド型ノードグループにはアカウント内の Amazon EC2 インスタンスが含まれています。これらのインスタンスはお客様または代わりに Amazon EKS がコントローラーのバージョンを更新しても、自動的にアップグレードされません。セルフマネージド型ノードグループは更新が必要であることをコンソールに表示しません。更新が必要なノードを確認するにはクラスターの [概要] タブにある [ノード] のリストからノードを選択して、そこにインストールされている kubelet バージョンを表示します。ノードは手動で更新する必要があります。詳細については、「[the section called “更新方式”](#)」を参照してください。
- セルフマネージド型ノードで kubelet が使用する証明書は、1 年の有効期限で発行されます。デフォルトでは、証明書のローテーションは有効になっていません (<https://kubernetes.io/docs/reference/config-api/kubelet-config.v1beta1/#kubelet-config-k8s-io-v1beta1-KubeletConfiguration> を参照)。つまり、セルフマネージド型ノードが 1 年以上実行されている場合、Kubernetes API に対して認証できなくなります。

- ベストプラクティスとして、お客様は定期的にセルフマネージド型ノードグループを更新して、最新の Amazon EKS 最適化 AMI から CVE およびセキュリティパッチを受け取ることをお勧めします。セルフマネージド型ノードグループで使用される AMI を更新すると、ノードの再作成もトリガーされ、kubelet 証明書の有効期限が切れたことによる問題が発生しないようにします。
- または、セルフマネージド型ノードグループの作成時にクライアント証明書のローテーション (<https://kubernetes.io/docs/tasks/tls/certificate-rotation/> を参照) を有効にして、現在の証明書の有効期限が近づくと kubelet 証明書が更新されるようにすることもできます。

## eksctl

**eksctl** を使用して自己管理型 Linux ノードを起動するには

1. デバイスまたは AWS クラウドシェル にインストールされている **eksctl** コマンドラインツールのバージョン 0.210.0 以降をインストールします。**eksctl** をインストールまたはアップグレードするには **eksctl** ドキュメントの「[インストール](#)」を参照してください。
2. クラスターが AWS クラウド にあり、AmazonEKS\_CNI\_Policy マネージド IAM ポリシーが [Amazon EKS ノードの IAM ロール](#) へアタッチされている場合は代わりに Kubernetes aws-node サービスアカウントに関連付けた IAM ロールに割り当てることをお勧めします。詳細については、「[the section called “IRSA 用の設定”](#)」を参照してください。クラスターが アウトポスト にある場合はポリシーをノードロールにアタッチする必要があります。
3. 次のコマンドは既存のクラスターにノードグループを作成します。クラスターは **eksctl** を使用して作成されている必要があります。**al-nodes** を、ノードグループの名前に置き換えます。ノードグループ名は 63 文字以下である必要があります。先頭は文字または数字でなければなりませんが、残りの文字にはハイフンおよびアンダースコアを含めることもできます。**#####** の部分は自分のクラスター名に置き換えます。この名前には英数字 (大文字と小文字が区別されます) とハイフンのみを使用できます。先頭の文字は英数字である必要があります。また、100 文字より長くすることはできません。名前はクラスターを作成する AWS リージョンおよび AWS アカウント内で一意である必要があります。クラスターが アウトポスト に存在する場合は **id** を アウトポスト サブネットの ID に置き換えます。AWS クラウド にクラスターが存在する場合、**id** を クラスターの作成時に指定しなかったサブネットの ID に置き換えます。**#####** を アウトポストでサポートされているインスタンスタイプに置き換えます。残りの **#####** は独自の値に置き換えます。デフォルトでは、ノードはコントロールプレーンと同じ Kubernetes バージョンで作成されます。

##### を アウトポスト で利用可能なインスタンスタイプに置き換えます。

#### を Amazon EC2 キーペアまたはパブリックキーの名前に置き換えます。このキーは起動後のノードに SSH 接続するために使用されます。Amazon EC2 キーペアをまだ持っていない場合は AWS Management Console で作成できます。詳細については「Amazon EC2 ユーザーガイド」の「[Amazon EC2 キーペア](#)」を参照してください。

次のコマンドを使用して、ノードグループを作成します。

```
eksctl create nodegroup --cluster my-cluster --name al-nodes --node-type instance-type \
 --nodes 3 --nodes-min 1 --nodes-max 4 --managed=false --node-volume-type gp2 --
 subnet-ids subnet-id
```

クラスターを AWS クラウド 上にデプロイしている場合:

- デプロイするノードグループではインスタンスのブロックとは異なる CIDR ブロックから IPv4 アドレスを Pod に割り当てることができます。詳細については、「[the section called “カスタム ネットワーキング”](#)」を参照してください。
- デプロイするノードグループはアウトバウンドインターネットアクセスを必要としません。詳細については「[the section called “プライベートクラスター”](#)」を参照してください。

利用できるすべてのオプションとデフォルトの詳細なリストについてはeksctl ドキュメントの「[AWS アウトポスト サポート](#)」を参照してください。

- ノードがクラスターに参加できない場合は「[Amazon EKS クラスターとノードに関する問題をトラブルシューティングする](#)」の「[the section called “ノードをクラスターに結合できません”](#)」および「[AWS アウトポスト でローカル Amazon EKS クラスターをトラブルシューティングする](#)」の「[the section called “ノードをクラスターに結合できない”](#)」を参照してください。
- 出力例は次のとおりです。ノードの作成中に、複数の行が出力されます。出力の最後の行は次のサンプル行が表示されます。

```
[#] created 1 nodegroup(s) in cluster "my-cluster"
```

4. (オプション) [サンプルアプリケーション](#)をデプロイして、クラスターと Linux ノードをテストします。

# AWS Management Console

ステップ 1: AWS Management Console を使用してセルフマネージド型の リナックス ノードを起動する`

1. AWS クラウドフォーメーション テンプレートの最新バージョンをダウンロードします。

```
curl -O https://s3.us-west-2.amazonaws.com/amazon-eks/cloudformation/2022-12-23/
amazon-eks-nodegroup.yaml
```

2. [AWS クラウドFormation コンソール](#)を開きます。
3. [スタックの作成] を選択し、[新しいリソースを使用 (標準)] を選択してください。
4. [テンプレートの指定] で、[テンプレートファイルのアップロード] を選択し、[ファイルを選択] を選択してください。前のステップでダウンロードした amazon-eks-nodegroup.yaml ファイルを選択し、[次へ] を選択してください。
5. [スタックの詳細の指定] ページで、必要に応じて次のパラメータを入力し、[次へ] を選択してください：
  - スタック名: AWS クラウドFormation スタックのスタック名を選択してください：例えば、**al-nodes** という名前にすることができます。この名前には英数字 (大文字と小文字が区別されます) とハイフンのみを使用できます。先頭の文字は英数字である必要があります。また、100 文字より長くすることはできません。名前はクラスターを作成する AWS リージョンおよび AWS アカウント内で一意である必要があります。
  - [クラスター名]: クラスターの名前を入力してください：この名前が、クラスター名と一致しない場合、ノードはクラスターに参加できません。
  - [クラスター制御プレーンセキュリティグループ]: [VPC](#) の作成時に生成した AWS クラウドフォーメーション 出力の [セキュリティグループ] 値を選択してください：

次のステップでは該当するグループを取得する 1 つのオペレーションを説明します。

- i. [Amazon EKS コンソール](#)を開きます。
  - ii. クラスターの名前を選択してください。
  - iii. [ネットワーキング] タブを選択してください。
  - iv. [クラスター制御プレーンセキュリティグループ] ドロップダウンリストから選択する場合は[追加のセキュリティグループ] の値をリファレンスとして使用します。
- [ノードグループ名]: ノードグループの名前を入力してください。この名前はノードに対して作成される自動スケーリングノードグループを識別するために後で使用できます。



- [ノード自動スケーリンググループ最小サイズ]: ノードの 自動スケーリング グループがスケールインできる最小ノード数を入力してください :
- ノード自動スケーリンググループ希望容量: スタック作成時にスケーリングする必要のあるノード数を入力してください。
- [NodeAutoScalingGroupMaxSize]: ノードの 自動スケーリング グループがスケールアウトできる最大ノード数を入力してください。
- [ノードインスタンス型]: ノードのインスタンスタイプを選択してください : クラスターが AWS クラウド で動作している場合は詳細については「[the section called “Amazon EC2 インスタンスタイプ”](#)」を参照してください。クラスターが アウトポスト で実行されている場合、アウトポスト で使用できるインスタンスタイプのみを選択できます。
- [NodeImageIdSSMParam]: 最新の Amazon EKS 最適化 AMI の Amazon EC2 Systems Manager のパラメータが、可変 Kubernetes バージョン用に事前設定されています。Amazon EKS でサポートされている別の Kubernetes マイナーバージョンを使用するには、**1.XX** を別の[サポートされているバージョン](#)に置き換えます。クラスターと同じ Kubernetes バージョンを指定することをお勧めします。

Amazon EKS 最適化高速 AMI を使用するには **amazon-linux-2** を amazon-linux-2-gpu に置き換えます。Amazon EKS 最適化 Arm AMI を使用するには **amazon-linux-2** を amazon-linux-2-arm64 に置き換えます。

#### Note

Amazon EKS ノード AMI は Amazon リナックス をベースとしています。Amazon Linux のセキュリティまたはプライバシーに関するイベントを、[Amazon Linux セキュリティセンター](#)で追跡できます。そのためには目的のバージョンのタブを選択してください。該当する RSS フィードをサブスクライブすることもできます。セキュリティおよびプライバシーイベントには問題の概要、影響を受けるパッケージ、および問題を修正するためにインスタンスを更新する方法などがあります。

- ノードイメージId: (オプション) (Amazon EKS 最適化 AMI の代わりに) 独自のカスタム AMI を使用している場合はAWS リージョンのノード AMI ID を入力してください。ここで値を指定すると、[ノードイメージIdSSMParam] フィールドの値はすべて上書きされます。
- [ノードボリュームサイズ]: ノードのルートボリュームのサイズを GiB 単位で指定します。
- [ノードボリューム型]: ノードのルートボリュームタイプを指定します。
- [キー名]: 起動後に、SSH を使用してノードに接続するときに使用できる Amazon EC2 SSH キーペアの名前を入力してください。Amazon EC2 キーペアをまだ持っていない場合はAWS

Management Console で作成できます。詳細については「アマゾン EC2 ユーザーガイド」の「[アマゾン EC2 キーペア](#)」を参照してください。

 Note

ここでキーペアを指定しないと、AWS クラウドFormation スタックの作成は失敗します。

- [ブートストラップ引数]: ノードに渡すことができるオプションの引数がいくつかあります : 詳細については、GitHub の [bootstrap script usage information](#) を参照してください。AWS Outposts の Amazon EKS ローカルクラスター (Kubernetes コントロールプレーンインスタンスが AWS Outposts で稼働) にノードを追加し、そのクラスターに入出力のインターネット接続がない (プライベートクラスターとも呼ばれる) 場合、次のブートストラップ引数を 1 行で指定する必要があります。

```
--b64-cluster-ca ${CLUSTER_CA} --apiserver-endpoint https://${APISERVER_ENDPOINT}
--enable-local-outpost true --cluster-id ${CLUSTER_ID}
```

Amazon EKS ローカルクラスターの CLUSTER\_CA、APISERVER\_ENDPOINT、CLUSTER\_ID の値を取得するには、次の AWS CLI コマンドを実行します。cluster-name をクラスターの名前に置き換え、region (us-east-1 など) をクラスターの AWS リージョンに置き換えます。

```
echo "CLUSTER_CA=$(aws eks describe-cluster --name cluster-name --region region --
query cluster.certificateAuthority.data --output text)"

echo "APISERVER_ENDPOINT=$(aws eks describe-cluster --name cluster-name --region
region --query cluster.endpoint --output text)"

echo "CLUSTER_ID=$(aws eks describe-cluster --name cluster-name --region region --
query cluster.id --output text)"
```

- [無効IMDSv1]: 各ノードはデフォルトでインスタンスメタデータサービスバージョン 1 (IMDSv1) および IMDSv2 をサポートします。IMDSv1 は無効にできます。ノードグループ内の将来のノードおよび Pod が IMDSv1 を使用しないようにするには DisableIMDSv1 を true に設定します。IDMS の詳細については「[インスタンスメタデータサービスの設定](#)」を参照してください。ノードでのそれへのアクセス制限について詳しくは[ワーカーノードに割り当てられたインスタンスプロファイルへのアクセスを制限する](#)を参照してください。
- [VpcId]: 作成した [VPC](#) の ID を入力します : VPC を選択する前に、「[VPC の要件と考慮事項](#)」を確認してください。



- [サブネット]: クラスターが アウトポスト にある場合、VPC 内で少なくとも 1 つのプライベートサブネットを選択してください: サブネットを選択する前に、「[サブネットの要件と考慮事項](#)」を確認してください。クラスターの [ネットワーキング] タブから、各サブネットリンクを開き、プライベートのサブネットを確認できます。
6. [スタックオプションの設定] ページで、希望する設定を選択し、[次へ] を選択してください。
  7. [AWS クラウドフォーメーション が IAM リソースを作成する可能性を認識しています] の左にあるチェックボックスを選択して、[スタックの作成] を選択してください。
  8. スタックの作成が完了したら、コンソールで選択し、[出力] を選択してください。
  9. 作成されたノードグループの [ノードインスタンス役割] を記録します。これは Amazon EKS ノードを設定する際、必要になります。

## ステップ 2: ノードを有効にしてクラスターに参加する

1. aws-auth ConfigMap がすでにあるかどうかを確認します。

```
kubectl describe configmap -n kube-system aws-auth
```

2. aws-auth ConfigMap が表示されている場合は必要に応じて更新してください。

- a. 編集する ConfigMap を開きます。

```
kubectl edit -n kube-system configmap/aws-auth
```

- b. 必要に応じて新しい mapRoles エントリを追加します。rolearn 値を、前の手順で記録した [ノードインスタンス役割] 値に設定します。

```
[...]
data:
 mapRoles: |
 - rolearn: <ARN of instance role (not instance profile)>
 username: system:node:{{EC2PrivateDNSName}}
 groups:
 - system:bootstrappers
 - system:nodes
[...]
```

- c. ファイルを保存し、テキストエディタを終了します。
3. 「Error from server (NotFound): configmaps "aws-auth" not found」というエラーが表示されたら、ストック ConfigMap を適用してください。

- a. 設定マップをダウンロードします。

```
curl -O https://s3.us-west-2.amazonaws.com/amazon-eks/cloudformation/2020-10-29/
aws-auth-cm.yaml
```

- b. aws-auth-cm.yaml ファイルで、rolearn を前の手順で記録した [ノードインスタンス役割] 値に設定します。これを行うにはテキストエディタを使用するか、##### を置き換えて次のコマンドを実行してください：

```
sed -i.bak -e 's|<ARN of instance role (not instance profile)>|my-node-instance-
role|' aws-auth-cm.yaml
```

- c. 設定を適用します。このコマンドが完了するまで数分かかることがあります。

```
kubectl apply -f aws-auth-cm.yaml
```

4. ノードのステータスを監視し、Ready ステータスになるまで待機します。

```
kubectl get nodes --watch
```

Ctrl+C を入力して、シェルプロンプトに戻ります。

#### Note

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

ノードがクラスターに参加できない場合は「[Amazon EKS クラスターとノードに関する問題をトラブルシューティングする](#)」の「[the section called “ノードをクラスターに結合できません”](#)」および「[AWS アウトポスト でローカル Amazon EKS クラスターをトラブルシューティングする](#)」の「[the section called “ノードをクラスターに結合できない”](#)」を参照してください。

5. Amazon EBS CSI ドライバーをインストールします。詳細についてはGitHub の [Installation](#) を参照してください。[ドライバーのアクセス許可を設定] セクションでは[IAM インスタンスプロファイルの使用] オプションの指示に従うことを確認します。gp2 ストレージクラスを使用する必要があります。gp3 ストレージクラスはサポートされていません。

クラスターの gp2 ストレージクラスを作成するには以下のステップを実行してください。

- a. 次のコマンドを実行して、gp2-storage-class.yaml ファイルを作成します。

```
cat >gp2-storage-class.yaml <<EOF
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
 annotations:
 storageclass.kubernetes.io/is-default-class: "true"
 name: ebs-sc
provisioner: ebs.csi.aws.com
volumeBindingMode: WaitForFirstConsumer
parameters:
 type: gp2
 encrypted: "true"
allowVolumeExpansion: true
EOF
```

- b. マニフェストをクラスターに適用します。

```
kubectl apply -f gp2-storage-class.yaml
```

6. (GPU ノードのみ) GPU インスタスタイプと Amazon EKS 最適化アクセラレーション AMI を選択した場合はクラスター上の DaemonSet として [NVIDIA device plugin for Kubernetes](#) を適用する必要があります。次のコマンドを実行する前に、**vX.X.X** を必要となる [NVIDIA/k8s-device-plugin](#) バージョンに置き換えます。

```
kubectl apply -f https://raw.githubusercontent.com/NVIDIA/k8s-device-plugin/vX.X.X/
deployments/static/nvidia-device-plugin.yml
```

### ステップ 3: その他のアクション

1. (オプション) [サンプルアプリケーション](#) をデプロイして、クラスターと Linux ノードをテストします。
2. クラスターが アウトポスト にデプロイされている場合はこのステップをスキップしてください。クラスターが AWS クラウド にデプロイされている場合、次の情報はオプションです。AmazonEKS\_CNI\_Policy マネージド IAM ポリシーが [Amazon EKS ノードの IAM ロール](#) にアタッチされている場合は代わりに Kubernetes aws-node サービスアカウントに関連付けた IAM ロールに割り当てることをお勧めします。詳細については、「[the section called “IRSA 用の設定”](#)」を参照してください。

# Amazon EKS での機械学習の概要

Amazon Elastic Kubernetes Service (EKS) は、比類のない柔軟性と制御性によって、組織が AI や機械学習 (ML) のワークロードをデプロイ、管理、スケーリングできるようにするマネージド Kubernetes プラットフォームです。オープンソースの Kubernetes エコシステム上に構築された EKS では、オープンソースのツールや AWS サービスとシームレスに統合しながら、既存の Kubernetes の専門知識を活用できます。

大規模なモデルのトレーニング、リアルタイムのオンライン推論の実行、生成 AI アプリケーションのデプロイなど、どのような用途であっても、EKS は AI/ML プロジェクトが必要とするパフォーマンス、スケーラビリティ、およびコスト効率を提供します。

## AI/ML のために EKS を選択すべき理由

EKS は、複雑な AI/ML ワークロードのデプロイと管理に役立つマネージド Kubernetes プラットフォームです。オープンソースの Kubernetes エコシステム上に構築されており、AWS サービスとの統合によって高度なプロジェクトに必要な制御とスケーラビリティを提供します。AI/ML のデプロイが初めてのチームでも、既存の Kubernetes スキルを直接転用できるため、複数のワークロードを効率的にオーケストレーションできます。

EKS は、オペレーティングシステムのカスタマイズからコンピューティングのスケーリングまで、あらゆることをサポートしており、そのオープンソースの基盤は技術的な柔軟性を促進し、将来のインフラストラクチャに関する意思決定のための選択肢を維持します。プラットフォームは、AI/ML ワークロードに必要なパフォーマンスとチューニングのオプションを提供し、次のような機能をサポートします。

- 隠された抽象化なしにコストと設定を微調整するための完全なクラスター制御
- 本番環境におけるリアルタイム推論ワークロードに対する 1 秒未満のレイテンシー
- マルチインスタンス GPU、マルチクラウド戦略、OS レベルのチューニングなどの高度なカスタマイズ
- AI/ML パイプライン全体で統合オーケストレーターとして EKS を使用してワークロードを一元化する機能

## 主なユースケース

Amazon EKS は、幅広い AI/ML ワークロードに対応した堅牢なプラットフォームを提供し、さまざまなテクノロジーとデプロイパターンをサポートしています。

- リアルタイム (オンライン) 推論: EKS は、Amazon EC2 の [Inf1](#) および [Inf2](#) インスタンス上で [TorchServe](#)、[Triton Inference Server](#)、[KServe](#) などのツールを使用することで、不正検出などの受信データに対する即時予測を 1 秒未満のレイテンシーで実現します。これらのワークロードは、[Karpenter](#) や [KEDA](#) による動的スケーリングの恩恵を受けると同時に、[Amazon EFS](#) を活用して複数のポッド間でモデルのシャーディングを行います。[Amazon ECR プルスルーキャッシュ \(PTC\)](#) によってモデルの更新が高速化され、[Bottlerocket](#) データボリュームと [Amazon EBS](#) 最適化ボリュームの組み合わせによって高速データアクセスが実現されます。
- 一般的なモデルトレーニング: 組織は EKS を活用して、[Amazon EC2 P4d](#) および [Amazon EC2 Trn1](#) インスタンス上で [Kubeflow Training Operator \(KRO\)](#)、[Ray Serve](#)、[Torch Distributed Elastic](#) を使用することで、大規模なデータセットの複雑なモデルを長期間にわたってトレーニングします。これらのワークロードは、[Volcano](#)、[Yunikorn](#)、[Kueue](#) などのツールを使用したバッチスケジューリングによってサポートされています。[Amazon EFS](#) はモデルチェックポイントの共有を可能にし、[Amazon S3](#) はバージョン管理のためのライフサイクルポリシーを使用してモデルのインポート/エクスポートを処理します。
- 検索拡張生成 (RAG) パイプライン: EKS は、検索プロセスと生成プロセスを統合することで、カスタマーサポートのチャットボットや同様のアプリケーションを管理します。これらのワークロードでは、多くの場合、オーケストレーションに [Argo Workflows](#) や [Kubeflow](#) などのツール、[Pinecone](#)、[Weaviate](#)、[Amazon OpenSearch](#) などのベクトルデータベースを使用し、[Application Load Balancer Controller \(LBC\)](#) を介してアプリケーションをユーザーに公開します。[NVIDIA NIM](#) は GPU 使用率を最適化し、[Prometheus](#) と [Grafana](#) はリソース使用率をモニタリングします。
- 生成 AI モデルのデプロイ: 企業は、Amazon [EC2 G5](#) や [Inferentia](#) アクセラレータ上で [Ray Serve](#)、[vLLM](#)、[Triton Inference Server](#) を使用して、EKS 上にテキスト生成や画像生成などのリアルタイムコンテンツ生成サービスをデプロイしています。これらのデプロイは、大規模なモデルにおけるパフォーマンスとメモリ使用率を最適化します。[JupyterHub](#) は反復型開発を可能にし、[Gradio](#) はシンプルな Web インターフェイスを提供し、[S3 Mountpoint CSI ドライバー](#) は大規模なモデルファイルにアクセスするために S3 バケットをファイルシステムとしてマウントすることを可能にします。
- バッチ (オフライン) 推論: 組織は、[AWS Batch](#) や [Volcano](#) を使用したスケジュールジョブを通じて、大規模なデータセットを効率的に処理します。これらのワークロードでは、多くの場合、AWS [Inferentia](#) チップ向けの [Inf1](#) および [Inf2](#) EC2 インスタンス、NVIDIA T4 GPU 向けの

Amazon EC2 [G4dn](#) インスタンス、あるいは [c5](#) および [c6i](#) CPU インスタンスが使用され、分析タスクに対するオフピーク時間中のリソース使用率が最大化されます。[AWS Neuron SDK](#) および NVIDIA GPU ドライバーはパフォーマンスを最適化し、MIG/TS は GPU 共有を可能にします。ストレージソリューションには、[Amazon S3](#)、Amazon [EFS](#)、[FSx for Lustre](#) などがあり、さまざまなストレージクラス用の CSI ドライバーが用意されています。モデル管理は [Kubeflow Pipelines](#)、[Argo Workflows](#)、[Ray クラスタ](#) などのツールを活用して行われ、モニタリングは [Prometheus](#)、[Grafana](#)、およびカスタムのモデルモニタリングツールによって処理されます。

## ケーススタディ

顧客は、以下の導入事例で示されているように、GPU 使用率の最適化や 1 秒未満のレイテンシーでのリアルタイム推論ワークロードの実行など、さまざまな理由で Amazon EKS を選択しています。Amazon EKS のすべての導入事例のリストについては、「[AWS のお客様の成功事例](#)」を参照してください。

- [Unitary](#) は、コンテンツモデレーションのために AI を使用して毎日 2,600 万本の動画を処理しており、高スループットかつ低レイテンシーの推論を必要としています。また、コンテナのブート時間を 80% 削減することで、トラフィックの変動に応じたスケーリングイベントへの迅速な対応を実現しています。
- [Miro](#) は、世界中で 7,000 万人のユーザーをサポートするビジュアルコラボレーションプラットフォームであり、以前のセルフマネージド Kubernetes クラスタと比較してコンピューティングコストが 80% 削減されたと報告しています。
- [Synthesia](#) は、顧客がテキストプロンプトからリアルな動画を作成できる生成 AI 動画作成をサービスとして提供しており、ML モデルのトレーニングスループットにおいて 30 倍の向上を達成しました。
- [Harri](#) は、ホスピタリティ業界向けに HR テクノロジーを提供しており、[AWS Graviton プロセッサ](#)への移行によって、需要の急増に対するスケーリングの 90% の高速化と、コンピューティングコストの 30% の削減を達成しました。
- [Ada Support](#) は、AI を活用したカスタマーサービス自動化企業であり、コンピューティングコストの 15% の削減と、コンピューティング効率の 30% の向上を達成しました。
- [Snorkel AI](#) は、企業が基盤モデルと大規模言語モデルを構築および適応できるように支援しており、GPU リソース用のインテリジェントなスケーリングメカニズムを実装することで、40% 以上のコスト削減を達成しました。



# EKS での機械学習の使用を開始する

AWS クラウド上の EKS で機械学習プラットフォームとワークロードの計画と使用を開始するには [the section called “ML の使用を開始する”](#) セクションに進みます。

## ML の使用を開始する

EKS での機械学習の使用を開始するには、まずこれらの規範的なパターンから選択して、EKS クラスターと ML ソフトウェアおよびハードウェアが ML ワークロードの実行を開始できるように準備します。これらのパターンのほとんどは、[Data on Amazon EKS](#) から入手できる Terraform ブループリントをベースとしています。開始する前に、いくつかの点に留意してください:

- これらの手順を実行するには GPU または Neuron インスタンスが必要です。これらのリソースが使用できない場合、クラスターの作成時またはノードの自動スケーリング中にこれらの手順が失敗する可能性があります。
- Neuron SDK (Trainium および Inferentia ベースのインスタンス) はコストを削減でき、NVIDIA GPU よりも高い可用性を提供します。そのため、ワークロードで許可されている場合は、機械学習ワークロードに Neuron を使用してみることをお勧めします (「[Welcome to AWS Neuron](#)」を参照)。
- ここでの開始方法エクスペリエンスの一部では、独自の [Hugging Face](#) アカウント経由でデータを取得する必要があります。

使用を開始するには、機械学習ワークロードを実行するインフラストラクチャのセットアップを開始できるように設計された次の複数のパターンの中から選択します。

- [JupyterHub on EKS: JupyterHub ブループリント](#) を詳しく見ていきます。Time Slicing および MIG の機能、ならびにプロファイルを使用したマルチテナント設定をご紹介します。これは、EKS で大規模な JupyterHub プラットフォームをデプロイする場合に最適です。
- [AWS Neuron および RayServe での大規模言語モデル](#): Amazon EKS ならびに AWS Trainium および Inferentia アクセラレーターで大規模言語モデル (LLM) AWS を実行するために [AWS Neuron](#) を使用します。推論リクエストを実行するためのプラットフォームをセットアップする手順については、「[Serving LLMs with RayServe and vLLM on AWS Neuron](#)」を参照してください。コンポーネントには次が含まれます:
  - 深層学習のための AWS Neuron SDK ツールキット
  - AWS Inferentia および Trainium アクセラレーター
  - vLLM - 可変長言語モデル ([vLLM](#) ドキュメントサイトを参照)

- RayServe スケーラブルモデルサービングライブラリ (「[Ray Serve: Scalable and Programmable Serving](#)」のサイトを参照)
- 独自の [Hugging Face](#) アカウントを使用する Llama-3 言語モデル。
- AWS CloudWatch と Neuron Monitor を使用したオブザーバビリティ
- WebUI を開く
- [NVIDIA と Triton の大規模言語モデル](#): Amazon EKS と NVIDIA GPU に複数の大規模言語モデル (LLM) をデプロイします。推論リクエストを実行するためのプラットフォームをセットアップする手順については、「[Deploying Multiple Large Language Models with NVIDIA Triton Server and vLLM](#)」を参照してください。コンポーネントには次が含まれます:
  - NVIDIA Triton Inference Server (「[Triton Inference Server](#)」の GitHub サイトを参照)
  - vLLM - 可変長言語モデル ([vLLM](#) ドキュメントサイトを参照)
  - 2 つの言語モデル: mistralai/Mistral-7B-Instruct-v0.2 および meta-llama/Llama-2-7b-chat-hf (独自の [Hugging Face](#) アカウントを使用)。

## ML on EKS の使用を続ける

このページに記載されているブループリントから選択するだけでなく、他にも、必要に応じて ML on EKS ドキュメントを通じて続行する方法があります。例えば、以下のことが可能です。

- ML on EKS のチュートリアルを試す – EKS で独自の機械学習モデルを構築して実行するための他のエンドツーエンドのチュートリアルを実行します。「[the section called “ML on EKS のチュートリアルを試す”](#)」を参照してください。

ML on EKS を使用して作業を改善するには、次を参照してください:

- ML を準備する – カスタム AMI や GPU 予約などの機能を備えた ML on EKS の準備方法について説明します。「[the section called “ML のために準備する”](#)」を参照してください。

## ML クラスターの準備

EKS での機械学習エクスペリエンスを改善するにはいくつかの方法があります。このセクションの以降のページは次に役立ちます:

- ML on EKS を使用するための選択肢を理解する。
- EKS および ML 環境を準備する。



特に、これは次に役立ちます:

- AMI を選択する: AWS は EKS 上で ML ワークロードを実行するための複数のカスタマイズされた AMI を提供します。「[the section called “Linux GPU AMI を実行する”](#)」および「[the section called “Windows GPU AMI を実行する”](#)」を参照してください。
- AMI をカスタマイズする: AWS カスタム AMI をさらに変更して、特定のユースケースに必要な他のソフトウェアやドライバーを追加できます。「[the section called “SMN の GPU を予約する”](#)」を参照してください。
- GPU を予約する: GPU の需要が高いことから、必要な GPU を必要なときに確実に使用できるように、必要な GPU を事前に予約できます。「[the section called “GPU ノードにテイントを適用する”](#)」を参照してください。
- EFA を追加する: Elastic Fabric Adapter を追加して、ノード間クラスター通信のネットワークパフォーマンスを改善します。「[the section called “EFA を使用したクラスターのトレーニング”](#)」を参照してください。
- AWS インフェクエンティア ワークロードを使用する: アマゾン EC2 Inf1 インスタンスを使用して EKS クラスターを作成します。「[the section called “インフェクエンティア クラスターを準備する”](#)」を参照してください。

## トピック

- [GPU アクセラレーションコンテナを実行する \(EC2 での Linux\)](#)
- [GPU アクセラレーションコンテナを実行する \(EC2 G シリーズでの Windows\)](#)
- [Capacity Blocks for ML を使用してマネージドノードグループを作成する](#)
- [Capacity Blocks for ML を使用してセルフマネージドノードを作成する](#)
- [特定のノードで Pod がスケジュールされないようにする](#)
- [Elastic Fabric Adapter を使用して Amazon EKS で機械学習トレーニングを実行する](#)
- [機械学習のための Amazon EKS で AWS Inferentia インスタンスを使用する](#)

## GPU アクセラレーションコンテナを実行する (EC2 での Linux)

Amazon EKS 最適化高速 Amazon Linux AMI は、標準的な Amazon EKS 最適化 Amazon Linux AMI 上に構築されています。これらの AMI の詳細については、「[the section called “Amazon EKS 最適化高速 Amazon Linux AMI”](#)」を参照してください。次のテキストでは、AWS Neuron ベースのワークロードを有効にする方法について説明します。

AWS Neuron (ML アクセラレーター) ベースのワークロードを有効にする

Amazon EKS で Neuron を使用するトレーニングおよび推論ワークロードの詳細については、以下のリファレンスを参照してください。

- [コンテナ - Kubernetes - 開始方法](#) (AWS Neuron ドキュメント)
- GitHub での AWS Neuron EKS サンプルによる [トレーニング](#)
- [Amazon EKS で AWSInferentia を使用して ML 推論ワークロードをデプロイする](#)

次の手順で、Amazon EKS 最適化高速 AMI を使用しながら GPU ベースのインスタンス上でワークロードを実行する方法を説明します。

1. GPU ノードをクラスターに加えた後、[NVIDIA device plugin for Kubernetes](#) をクラスターの DaemonSet として適用する必要があります。次のコマンドを実行する前に、**vX.X.X** を必要となる [NVIDIA/k8s-device-plugin](#) バージョンに置き換えます。

```
kubectl apply -f https://raw.githubusercontent.com/NVIDIA/k8s-device-plugin/vX.X.X/
deployments/static/nvidia-device-plugin.yml
```

2. ノードに割り当て可能な GPU があることは、次のコマンドで確認できます。

```
kubectl get nodes "-o=custom-
columns=NAME:.metadata.name,GPU:.status.allocatable.nvidia\.com/gpu"
```

3. 次の内容で、nvidia-smi.yaml というファイルを作成します。**##**を [nvidia/cuda](#) に必要なタグに置き換えます。このマニフェストでは、ノード上で nvidia-smi を実行する [NVIDIA CUDA](#) コンテナ を起動します。

```
apiVersion: v1
kind: Pod
metadata:
 name: nvidia-smi
spec:
 restartPolicy: OnFailure
 containers:
 - name: nvidia-smi
 image: nvidia/cuda:tag
 args:
 - "nvidia-smi"
 resources:
 limits:
 nvidia.com/gpu: 1
```

4. 次のコマンドを使用してマニフェストを適用します。

```
kubectl apply -f nvidia-smi.yaml
```

5. Pod の実行が終了したら、次のコマンドを使用してそのログを表示します。

```
kubectl logs nvidia-smi
```

出力例は次のとおりです。

```
Mon Aug 6 20:23:31 20XX
+-----+
| NVIDIA-SMI XXX.XX Driver Version: XXX.XX |
+-----+-----+-----+-----+-----+-----+
| GPU Name Persistence-M| Bus-Id Disp.A | Volatile Uncorr. ECC |
| Fan Temp Perf Pwr:Usage/Cap| Memory-Usage | GPU-Util Compute M. |
|=====+=====+=====+=====+=====+=====+
| 0 Tesla V100-SXM2... On | 00000000:00:1C:0 Off | 0 |
| N/A 46C P0 47W / 300W | 0MiB / 16160MiB | 0% Default |
+-----+-----+-----+-----+-----+-----+
+-----+
| Processes: GPU Memory |
| GPU PID Type Process name Usage |
|=====+=====+=====+=====+=====+=====+
| No running processes found |
+-----+-----+-----+-----+-----+-----+-----+
+-----+
```

## GPU アクセラレーションコンテナを実行する (EC2 G シリーズでの Windows)

### Important

TensorWorks による [DirectX 用 Kubernetes Device Plugin](#) は、AWS によって承認、サポート、または保守されていないサードパーティーのツールです。AWS は、このプラグインのセキュリティ、信頼性、またはパフォーマンスについて責任を負いません。

TensorWorks の DirectX 用 Kubernetes デバイスプラグインで NVIDIA GPU を使用して、Amazon EKS (Elastic Kubernetes Service) で GPU アクセラレーション Windows コンテナワークロードを実

行する方法について説明します。詳細については、「[Kubernetes Device Plugin for DirectX](#)」を参照してください。

Windows コンテナの GPU アクセラレーションをセットアップするには、主に 2 つの方法があります。

- オプション 1: 必要な GPU ドライバーがプリインストールされた[カスタム EKS Windows 最適化 AMI を構築](#)します。
  - このアプローチは、GPU アクセラレーションされた Windows コンテナをすぐに実行できる、一貫性のある事前設定済みの環境が必要で、カスタム AMI を構築して維持するために追加の労力を費やすことができる場合に使用します。
- オプション 2: インスタンスの起動後に、必要な GPU ドライバーを EKS ワーカーノードにインストールします。
  - このアプローチは、より簡単なセットアッププロセスが必要で、新しいワーカーノードそれぞれに GPU ドライバーをインストールしたくない場合に使用します。GPU アクセラレーションのワークロードを評価またはプロトタイプ化する開発環境により適しています。

どちらのアプローチも、本ガイドで説明されているステップで活用することができます。

## 考慮事項

本ガイドでは、NVIDIA GPU、NVIDIA GRID ドライバー、および TensorWorks による [DirectX 用 Kubernetes デバイスプラグイン](#) を使用して、Windows コンテナの GPU アクセラレーションをインストールしてセットアップする手順について説明します。これらのステップは、Amazon EKS の Windows コンテナワークロードに GPU アクセラレーションを提供するようにテストおよび検証されています。互換性のあるドライバーとデバイスプラグインの詳細については、「[the section called “既知の制限事項”](#)」を参照してください。先に進める前に、次の点に注意してください。

- [NVIDIA GRID ドライバー](#) を使用する G ファミリーインスタンスタイプのみが、本ガイドで動作することがテストおよび検証されています。他のインスタンスタイプとドライバーの組み合わせでも GPU アクセラレーション Windows コンテナを実行できる場合がありますが、本ガイドでは説明されていない追加の設定手順が必要になる場合があります。
- DirectX ベースのワークロードのみが、本ガイドで動作することがテストおよび検証されています。OpenGL、Vulkan、OpenCL などの他の GPU API でも、GPU アクセラレーションされた Windows コンテナの実行と互換性がある可能性があります。このガイドでは説明されていない追加の設定手順が必要になる場合があります。

- GPU アクセラレーション Windows コンテナを実行する前に注意すべき既知の制限がいくつかあります。詳細については、[the section called “既知の制限事項”](#) のセクションを参照してください。

## 前提条件

Amazon EKS で Windows コンテナの GPU アクセラレーションを有効にするには、手順を進める前に次の要件を準備する必要があります。

- Kubernetes v1.27 以降で Amazon EKS クラスターを起動します。
- Windows Server 2022 以降で Windows ノードをプロビジョニングします。
- [G4](#) や [G5](#) などのインスタンスタイプの G ファミリーで Windows ノードをプロビジョニングします。
- containerd 1.7.x または 2.x.x でコンテナランタイムを使用して Windows ノードをプロビジョニングします。(Amazon EKS 最適化 AMI 内の containerd バージョンを確認するには、「[the section called “バージョン情報を取得する”](#)」を参照してください。)

## 各 Windows ノードに GPU ドライバーをインストールする

EKS ワーカーノードに NVIDIA GRID ドライバーをインストールするには、[Amazon EC2 インスタンスの NVIDIA ドライバー](#)で説明されているステップに従ってください。[インストールオプション - オプション 3: GRID ドライバー](#)に移動し、インストール手順に従ってください。

### Windows Server Core のためのインストール

デスクトップエクスペリエンスのない Windows Server Core の場合は、次のコマンドを使用して NVIDIA GRID ドライバーをサイレントインストールします。

```
$nvidiaInstallerFilePath = nvidia-driver-installer.exe # Replace with path to installer
$installerArguments = "-s -clean -noreboot -noeula"
Start-Process -FilePath $nvidiaInstallerFilePath -ArgumentList $installerArguments -
Wait -NoNewWindow -PassThru
```

### インストールを確認する

次の PowerShell コマンドを実行して、インスタンスの GPU に関する診断情報を表示します。

```
nvidia-smi
```

このコマンドは、NVIDIA ドライバーのバージョンと GPU ハードウェアに関する情報を表示します。このコマンドの出力が、インストールされると想定している NVIDIA GRID ドライバーのバージョンと一致することを確認します。

## 各ノードに GPU デバイスプラグインをデプロイする

GPU リソースの検出と Windows ノード上のコンテナへの公開を有効にするには、デバイスプラグインが必要になります。EKS クラスターで DaemonSet として実行することで、各ワーカーノードに Tensorworks による [DirectX デバイスプラグイン](#) をデプロイします。[README.md](#) で指定されているインストールガイドに従ってください。これには、次のステップが求められます。以下を行うことをお勧めします。

- デバイスプラグインを kube-system 名前空間にデプロイします。
- DaemonSet に適切なリソース制限を設定して、ノードでリソースを過剰に消費しないようにします。

### Note

デバイスプラグイン DaemonSet は、昇格された権限を持つホストプロセスコンテナとしてすべてのノードで実行されます。この DaemonSet へのアクセスを制限する RBAC コントロールを実装して、権限のあるユーザーのみが特権コマンドを実行できるようにすることをお勧めします。

GPU アクセラレーションコンテナを実行する場合、デバイスプラグインは 2 つのモードをサポートします。

- シングルテナンシーモード: このモードは、すべての GPU リソースをインスタンス上の 1 つのコンテナ専用に使います。次のコマンドを使用して、シングルテナンシーをサポートするデバイスプラグインをインストールします。詳細については、「README.md」を参照してください。

```
kubectl apply -f "https://raw.githubusercontent.com/TensorWorks/directx-device-plugins/main/deployments/default-daemonsets.yml"
```

- マルチテナンシーモード: このモードでは、インスタンス上の複数のコンテナ間で GPU リソースを共有できます。次のコマンドを使用して、マルチテナンシーをサポートするデバイスプラグインをインストールします。詳細については、「README.md」を参照してください。

```
kubectl apply -f "https://raw.githubusercontent.com/TensorWorks/directx-device-plugins/main/deployments/multitenancy-inline.yml"
```

または、ConfigMap を使用してマルチテナンシーを指定します。

```
kubectl apply -f "https://raw.githubusercontent.com/TensorWorks/directx-device-plugins/main/deployments/multitenancy-configmap.yml"
```

## デバイスプラグインのデプロイの検証

デバイスプラグインをデプロイしたら、<namespace> を置き換えてから次のコマンドを実行して、DirectX Device Plugin がすべての Windows ノードで正しく動作していることを確認します。

```
kubectl get ds device-plugin-wddm -n <namespace>
```

## コンテナをデプロイする準備が整っていることの検証

デバイスプラグイン DaemonSet が GPU 搭載の Windows ワーカーノードで実行されると、次のコマンドを使用して、各ノードに割り当て可能な GPU があることを確認します。対応する番号は、各ノードの DirectX デバイスの番号と一致する必要があります。

```
kubectl get nodes "-o=custom-columns=NAME:.metadata.name,DirectX:.status.allocatable.directx\.microsoft\.com/display"
```

## GPU アクセラレーションを使用した Windows コンテナの実行

ポッドを起動する前に、`.spec.containers[].resources` でリソース名 `directx.microsoft.com/display` を指定します。これは、コンテナに GPU が対応している機能が必要であることを示し、kube-scheduler は使用可能な GPU リソースを持つ事前設定された Windows ノードにポッドを配置しようとします。

例として、pi の値を推定するモンテカルロ・シミュレーションを実行する Job を起動する以下のサンプル・コマンドをご覧ください。この例は、[DirectX 用の Kubernetes Device Plugins](#) を持つ GitHub リポジトリのものです。このリポジトリには、Windows ノード GPU の機能をテストするために実行できる[複数の例](#)があります。

```
cat <<EOF | kubectl apply -f -
```



```
apiVersion: batch/v1
kind: Job
metadata:
 name: example-cuda-montecarlo-wddm
spec:
 template:
 spec:
 containers:
 - name: example-cuda-montecarlo-wddm
 image: "index.docker.io/tensorworks/example-cuda-montecarlo:0.0.1"
 resources:
 limits:
 directx.microsoft.com/display: 1
 nodeSelector:
 "kubernetes.io/os": windows
 restartPolicy: Never
 backoffLimit: 0
EOF
```

## 既知の制限事項

### すべての GPU が使用可能です

インスタンスのすべての GPU は、特定のコンテナに対して特定の数の GPU をリクエストした場合でも、ホストで実行中の各コンテナで使えるようになります。さらに、デフォルトの動作では、ノードで使用可能な複数の GPU がある場合でも、ホストで実行されているすべてのコンテナがインデックス 0 の GPU を使用します。したがって、マルチ GPU タスクを正しく動作させるには、アプリケーションのコード内で使用される特定の GPU デバイスを明示的に指定する必要があります。

アプリケーションに使用するデバイスを割り当てるための正確な実装は、使用しているプログラミング言語やフレームワークによって変わります。例えば、CUDA プログラミングを使用している場合、特定の GPU を選択するために、関数 [cudaSetDevice\(\)](#) を使用して、アプリケーションコードで使用するデバイスを明示的に指定できます。

デバイスを明示的に指定する必要があるのは、Windows コンテナに影響する既知の問題があるからです。この問題の解決の進捗状況は、[microsoft/Windows-Containers の問題 #333](#) で追跡できます。次の表は、この GPU 割り当て動作の視覚的な表現と実例を示しています。

EC2 インスタンスタイプ g4dn.12xlarge の Windows ノードが 1 つあり、4 つの GPU が搭載されているというシナリオを考えてみましょう。このインスタンスで起動されるポッドが 3 つあるシナリオを考えてみましょう。この表は、各コンテナがリクエストした GPU の数に関係なく、3 つの



ポッドすべてがインスタンス上の 4 つの GPU すべてにアクセスできることを示しており、デフォルトではデバイスインデックス 0 の GPU を利用できます。

| ポッド   | リクエストされた GPU | 実際の GPU アクセス | デフォルトの GPU 使用率 | 使用可能な GPU インデックス | インスタンス GPU の合計 |
|-------|--------------|--------------|----------------|------------------|----------------|
| Pod 1 | 1 GPU        | すべての 4 GPU   | インデックス 0 の GPU | 0、1、2、3          | 4              |
| Pod 2 | GPU 2 個      | すべての 4 GPU   | インデックス 0 の GPU | 0、1、2、3          | 4              |
| Pod 3 | 1 GPU        | すべての 4 GPU   | インデックス 0 の GPU | 0、1、2、3          | 4              |

## Kubernetes デバイスプラグインのサポート

NVIDIA の [Kubernetes デバイスプラグイン](#) の公式実装は Windows をサポートしていません。 [NVIDIA/k8s-device-plugin issue #419](#) で、Windows の公式サポート追加の進行状況を追跡できます。

## GPU コンピューティングインスタンスの制限

AWS アカウント設定によっては、起動できる Amazon EC2 GPU コンピューティングインスタンスの数とタイプにサービス制限がある場合があります。追加の容量が必要な場合は、[クォータの引き上げをリクエスト](#)できます。

## Windows GPU 最適化 AMI を構築する必要があります

Amazon EKS が提供する EKS Windows GPU 最適化 AMI または EC2 Image Builder マネージドコンポーネントはありません。このガイドの手順に従って、必要な GPU ドライバーがプリインストールされたカスタム EKS Windows 最適化 AMI を構築するか、インスタンスの起動後に必要な GPU ドライバーを EKS ワーカーノードにインストールする必要があります。

## Inferentia と Trainium はサポートされていません

AWS [Inferentia](#) および AWS [Trainium](#) ベースのワークロードは Windows ではサポートされていません。

## Capacity Blocks for ML を使用してマネージドノードグループを作成する

機械学習 (ML) のキャパシティブロックを使用すると、短期間の機械学習ワークロードをサポートするために、GPU インスタンスを将来の日付で予約できます。詳細については、「Linux インスタンス用 Amazon EC2 ユーザーガイド」の「[機械学習用のキャパシティブロック](#)」を参照してください。

### 考慮事項

#### Important

- キャパシティブロックは、特定の Amazon EC2 インスタンスタイプおよび AWS リージョンでのみ使用できます。互換性情報については、「Linux インスタンス用 Amazon EC2 ユーザーガイド」の「[キャパシティブロックの操作](#)」を参照してください。
- 詳細については、「Amazon EC2 Auto Scaling ユーザーガイド」の「[Use Capacity Blocks for machine learning workloads](#)」を参照してください。
- キャパシティブロックのあるマネージドノードグループは、カスタム起動テンプレートでのみ作成できます。
- キャパシティブロックを持つマネージドノードグループをアップグレードする場合は、ノードグループの希望するサイズが 0 に設定されていることを確認してください。

## Amazon EC2 キャパシティブロックを持つマネージドノードグループを作成する

Amazon EKS マネージド型ノードグループでキャパシティブロックを使用して、GPU 加速ワーカーノードのプロビジョニングとスケーリングを行うことができます。AWS CloudFormation テンプレートの例は、本番クラスターに必要なすべての側面を網羅しているわけではありません。通常、ブートストラップスクリプトでノードをクラスターに結合し、Amazon EKS 高速 AMI を指定する必要もあります。詳細については、「[the section called “作成”](#)」を参照してください。

1. ワークロードに適した、Amazon EKS マネージド型ノードグループと連携する起動テンプレートを作成します。詳細については、「[the section called “起動テンプレート”](#)」を参照してください。

上記の手順の要件に加えて、LaunchTemplateData に以下が含まれていることを確認してください。

- MarketType が "capacity-block" に設定された InstanceMarketOptions

- CapacityReservationId がキャパシティブロックに設定された CapacityReservationSpecification: CapacityReservationTarget (例: `cr-02168da1478b509e0` )
- キャパシティブロックをサポートするインスタンスタイプに設定された InstanceType (例: `p5.48xlarge` )

キャパシティブロックをターゲットとする起動テンプレートを作成する CloudFormation テンプレートの抜粋を以下に示します。カスタム AMI マネージド型ノードグループを作成するには、ImageId および UserData パラメータを追加することもできます。

```
NodeLaunchTemplate:
 Type: "AWS::EC2::LaunchTemplate"
 Properties:
 LaunchTemplateData:
 InstanceMarketOptions:
 MarketType: "capacity-block"
 CapacityReservationSpecification:
 CapacityReservationTarget:
 CapacityReservationId: "cr-02168da1478b509e0"
 InstanceType: p5.48xlarge
```

## 2. 起動テンプレートを使用して、マネージド型ノードグループを作成します。

キャパシティブロックのノードグループ作成コマンドの例を以下に示します。 *example-values* をクラスターに適用可能なものに置き換えます。

キャパシティブロックのマネージド型ノードグループを作成するときは、以下の操作を行います。

- capacity-type を "CAPACITY\_BLOCK" に設定します。キャパシティタイプが "CAPACITY\_BLOCK" に設定されていない場合、または上記の必須起動テンプレート値のいずれかが欠落している場合、作成リクエストは拒否されます。
- 作成リクエストで subnets を指定する場合は、キャパシティ予約と同じアベイラビリティーゾーンのサブネットのみを指定してください。
- 作成リクエストでゼロ以外の desiredSize を指定すると、Amazon EKS は Auto Scaling グループ (ASG) を作成するときにその値を採用します。ただし、キャパシティ予約がアクティブになる前に作成リクエストが行われた場合は、アクティブになるまで ASG は Amazon EC2 インスタンスを起動できません。その結果、ASG スケーリングアクティビティで起動エラーが発

生します。予約がアクティブになると、インスタンスの起動は成功し、ASG は作成時に指定された `desiredSize` にスケールアップされます。

```
aws eks create-nodegroup \
 --cluster-name my-cluster \
 --nodegroup-name my-mng \
 --node-role node-role-arn \
 --region region-code \
 --subnets subnet-id \
 --scaling-config minSize=node-group-min-size,maxSize=node-group-max-
size,desiredSize=node-group-desired-size \
 --ami-type "AL2023_x86_64_NVIDIA" \
 --capacity-type "CAPACITY_BLOCK" \
 --launch-template id="lt-id",version=1
```

3. スケールアップ後にノードが結合することを確認します。キャパシティブロックを持つマネージドノードグループを使用する Amazon EKS クラスターは、起動されたインスタンスが実際にクラスターに参加して登録されるかの検証を実行しません。
4. 作成時に `desiredSize` を 0 に設定すると、キャパシティ予約がアクティブになったときにノードグループをスケールアップするさまざまなオプションを使用できます。
  - ASG に対して、キャパシティブロックの予約開始時間に合わせて、スケジュールされたスケールリングポリシーを作成します。詳細については、「Amazon EC2 Auto Scaling ユーザーガイド」の「[Amazon EC2 Auto Scaling のスケジュールされたスケールリング](#)」を参照してください。
  - Amazon EKS コンソールまたは `eks update-nodegroup-config` を使用してスケールリング設定を更新し、ノードグループの必要なサイズを設定します。
  - Kubernetes Cluster Autoscaler を使用します。詳細については、「[AWS の Cluster Autoscaler](#)」を参照してください。
5. これで、ノードグループではワークロードと Pod をスケジュールする準備が整いました。
6. 予約が終了する前に Pod を正常にドレインするために、Amazon EKS はスケジュールされたスケールリングポリシーを使用してノードグループサイズを 0 にスケールダウンします。このスケジュールされたスケールリングは、Amazon EKS Node Group Capacity Scaledown Before Reservation End という名前で設定されます。このアクションは編集も削除もしないことをお勧めします。

Amazon EC2 は、予約終了時間の 30 分前にインスタンスのシャットダウンを開始します。その結果、Amazon EKS は、Pod を安全かつ適切にエビクトするために、予約終了の 40 分前に、スケジュールされたスケールダウンをノードグループに設定します。

## Capacity Blocks for ML を使用してセルフマネージドノードを作成する

機械学習 (ML) のキャパシティブロックを使用すると、短期間の機械学習ワークロードをサポートするために、GPU インスタンスを将来の日付で予約できます。詳細については、「Linux インスタンス用 Amazon EC2 ユーザーガイド」の「[機械学習用のキャパシティブロック](#)」を参照してください。

### 考慮事項

#### Important

- キャパシティブロックは、特定の Amazon EC2 インスタンスタイプおよび AWS リージョンでのみ使用できます。互換性情報については、「Linux インスタンス用 Amazon EC2 ユーザーガイド」の「[キャパシティブロックの操作](#)」を参照してください。
- 現在、キャパシティブロックは Karpenter では使用できません。
- キャパシティ予約が有効になる前にセルフマネージド型ノードグループを作成する場合は、希望キャパシティを 0 に設定します。
- ノードを正常にドレインさせるのに十分な時間を確保するために、キャパシティブロックの予約終了時刻の 30 分以上前にスケーリングがゼロになるようにスケーリングをスケジュールすることをお勧めします。
- Pod が正常にドレインされるように、例の手順の説明に従って AWS ノード終了ハンドラーを設定することをお勧めします。

### セルフマネージド型ノードでキャパシティブロックを使用する

Amazon EKS でキャパシティブロックを使用して、セルフマネージド型ノードのプロビジョニングとスケーリングを行うことができます。以下のステップは、一般的な概要の例を示しています。AWS CloudFormation テンプレートの例は、本番ワークロードに必要なすべての側面を網羅しているわけではありません。通常、ブートストラップスクリプトを使用してノードをクラスターに結合し、Amazon EKS 高速 AMI、およびクラスターに参加するための適切なインスタンスプロファイルを指定することも必要になります。詳細については、「[the section called “Amazon Linux”](#)」を参照してください。

1. ワークロードに適用可能な起動テンプレートを作成します。詳細については、「Amazon EC2 Auto Scaling ユーザーガイド」の「[Use Capacity Blocks for machine learning workloads](#)」を参照してください。

LaunchTemplateData に以下が含まれていることを確認します。

- MarketType が "capacity-block" に設定された InstanceMarketOptions
- CapacityReservationId がキャパシティブロックに設定された CapacityReservationSpecification: CapacityReservationTarget (例: `cr-02168da1478b509e0` )
- Arn が適切な `iam-instance-profile-arn` に設定された IamInstanceProfile
- 適切な `image-id` に設定された ImageId
- キャパシティブロックをサポートするインスタンスタイプに設定された InstanceType (例: `p5.48xlarge` )
- 適切な ID に設定された SecurityGroupIds (例: `sg-05b1d815d1EXAMPLE`)
- セルフマネージド型ノードグループの適切な `user-data` に設定された UserData

キャパシティブロックをターゲットとする起動テンプレートを作成する CloudFormation テンプレートの抜粋を以下に示します。

```
NodeLaunchTemplate:
 Type: "aws::EC2::LaunchTemplate"
 Properties:
 LaunchTemplateData:
 InstanceMarketOptions:
 MarketType: "capacity-block"
 CapacityReservationSpecification:
 CapacityReservationTarget:
 CapacityReservationId: "cr-02168da1478b509e0"
 IamInstanceProfile:
 Arn: iam-instance-profile-arn
 ImageId: image-id
 InstanceType: p5.48xlarge
 KeyName: key-name
 SecurityGroupIds:
 - sg-05b1d815d1EXAMPLE
 UserData: user-data
```

キャパシティブロックはゾーン型であるため、予約が行われたアベイラビリティゾーンのサブネットを渡す必要があります。

2. 起動テンプレートを使用して、セルフマネージド型ノードグループを作成します。キャパシティ予約が有効になる前にこれを行う場合は、希望キャパシティを 0 に設定します。ノードグループ

を作成する際には、キャパシティが予約されているアベイラビリティゾーンの当該サブネットのみを指定するようにしてください。

以下は、ワークロードに適用可能な CloudFormation テンプレートを作成するときに参照できる、サンプル CloudFormation テンプレートです。この例では、前の手順で示した `AWS::Amazon::EC2::LaunchTemplate` リソースの `LaunchTemplateId` と `Version` を取得します。また、同じテンプレートの他の場所で宣言されている `DesiredCapacity`、`MaxSize`、`MinSize`、`VPCZoneIdentifier` の値も取得します。

```
NodeGroup:
 Type: "AWS::AutoScaling::AutoScalingGroup"
 Properties:
 DesiredCapacity: !Ref NodeAutoScalingGroupDesiredCapacity
 LaunchTemplate:
 LaunchTemplateId: !Ref NodeLaunchTemplate
 Version: !GetAtt NodeLaunchTemplate.LatestVersionNumber
 MaxSize: !Ref NodeAutoScalingGroupMaxSize
 MinSize: !Ref NodeAutoScalingGroupMinSize
 VPCZoneIdentifier: !Ref Subnets
 Tags:
 - Key: Name
 PropagateAtLaunch: true
 Value: !Sub ${ClusterName}-${NodeGroupName}-Node
 - Key: !Sub kubernetes.io/cluster/${ClusterName}
 PropagateAtLaunch: true
 Value: owned
```

3. ノードグループが正常に作成されたら、作成されたノードグループの `NodeInstanceRole` を必ず記録してください。これは、ノードグループがスケーリングされたときに、新しいノードがクラスターに参加し、Kubernetes がノードを認識できるようにするために必要です。詳細については、「[セルフマネージド Amazon Linux ノードを作成する](#)」の「AWS Management Console」の手順を参照してください。
4. Auto Scaling グループに対して、キャパシティブロックの予約時間に合わせて、スケジュールされたスケーリングポリシーを作成することをお勧めします。詳細については、「Amazon EC2 Auto Scaling ユーザーガイド」の「[Amazon EC2 Auto Scaling のスケジュールされたスケーリング](#)」を参照してください。

予約したすべてのインスタンスを使用できるのは、キャパシティブロックの終了時刻の 30 分前までです。その時点でまだ稼働しているインスタンスでは、終了処理が開始されます。ノードを正常にドレインさせるのに十分な時間を確保するために、キャパシティブロックの予約終了時刻の



30 分以上前にスケーリングがゼロになるようにスケーリングをスケジュールすることをお勧めします。

代わりに、キャパシティ予約が Active になるたびに手動でスケールアップを行う場合は、キャパシティブロック予約の開始時に Auto Scaling グループの希望キャパシティを更新する必要があります。その場合は、キャパシティブロック予約の終了時刻の 30 分以上前に手動でスケールダウンを行う必要もあります。

- これで、ノードグループではワークロードと Pod をスケジューリングする準備が整いました。
- Pod が正常にドレインされるように、AWS ノード終了ハンドラーを設定することをお勧めします。このハンドラーは、EventBridge を使用して Amazon EC2 Auto Scaling からの「ASG スケールイン」ライフサイクルイベントを監視し、インスタンスが使用できなくなる前に Kubernetes コントロールプレーンが必要なアクションを実行できるようにします。そうしないと、Pod と Kubernetes のオブジェクトが保留状態でスタックします。詳細については、GitHub の「[AWS ノード終了ハンドラー](#)」を参照してください。

ノード終了ハンドラーを設定しない場合は、正常にドレインさせる時間を十分に確保できるように、30 分のウィンドウに入る前に Pod のドレインを手動で開始することをお勧めします。

## 特定のノードで Pod がスケジュールされないようにする

GPU などの専門的なプロセッサを搭載したノードは、より標準的なマシンで実行されているノードよりも実行コストが高くなる場合があります。そのため、特別なハードウェアを必要としないワークロードがそれらのノードにデプロイされないように、それらのノードを保護したい場合があります。そのための 1 つの方法はテイントです。

Amazon EKS は、マネージド型ノードグループによる Kubernetes テイントの設定をサポートしています。テイントと容認は連携して、Pod が不適切なノードにスケジュールされないようにします。1 つ以上のテイントをノードに適用できます。つまり、ノードはテイントを容認しない Pod を受け入れるべきではありません。容認は Pod に適用され、テイントが一致するノードに Pod がスケジュールされることを許可しますが、必須ではありません。詳細については、Kubernetes ドキュメントの「[テイントと容認](#)」を参照してください。

AWS Management Consoleまたは Amazon EKS API を使用して、新規または既存のマネージド型ノードグループに Kubernetes ノードのテイントが適用されます。

- ・を使用してテイントが適用されたノードグループを作成するを使用してテイントが適用されたノードグループを作成するを使用してテイントが適用されたノードグループを作成する情報はAWS Management Consolethe section called “作成”、、、、、、、



- AWS CLI を使用してテイントが適用されたノードグループを作成する例は、次のとおりです。

```
aws eks create-nodegroup \
--cli-input-json '
{
 "clusterName": "my-cluster",
 "nodegroupName": "node-taints-example",
 "subnets": [
 "subnet-1234567890abcdef0",
 "subnet-abcdef01234567890",
 "subnet-021345abcdef67890"
],
 "nodeRole": "arn:aws:iam::111122223333:role/AmazonEKSNodeRole",
 "taints": [
 {
 "key": "dedicated",
 "value": "gpuGroup",
 "effect": "NO_SCHEDULE"
 }
]
}'
```

詳細な説明と使用例については、Kubernetes リファレンスドキュメントの「[taint](#)」を参照してください。

#### Note

- テイントは、UpdateNodegroupConfig API を使用してノードグループを作成した後に更新できます。
- テイントのキーは、文字または数字で始まる必要があります。英字、数字、ハイフン (-)、ピリオド (.)、およびアンダースコア (\_) を使用できます。最大 63 文字です
- オプションで、Taint キーは DNS サブドメインプレフィックスと単一の / で始めることができます。このキーを DNS サブドメインプレフィックスで始める場合は、最大 253 文字まで使用できます。
- 値は省略可能で、文字または数字で始める必要があります。英字、数字、ハイフン (-)、ピリオド (.)、およびアンダースコア (\_) を使用できます。最大 63 文字です
- Kubernetes を直接使用するか、AWS Management Consoleを使用する場合、テイント効果は NoSchedule、PreferNoSchedule、または NoExecute でなけ

ればなりません。ただし、AWS CLI または API を使用する場合、テイント効果は NO\_SCHEDULE、PREFER\_NO\_SCHEDULE、または NO\_EXECUTE でなければなりません。

- 1 つのノードグループには、最大 50 個のテイントを使用できます。
- マネージドノードグループを使用して作成されたテイントをノードから手動で削除した場合、Amazon EKS はテイントをノードに戻しません。これは、管理対象ノードグループ設定でテイントが指定されている場合にも当てはまります。

[aws eks update-nodegroup-config](#) AWS CLI コマンドを使用して、マネージドノードグループのテイントを追加、削除、または交換できます。

## Elastic Fabric Adapter を使用して Amazon EKS で機械学習トレーニングを実行する

このトピックでは、Amazon EKS クラスターにデプロイした Pod と Elastic Fabric Adapter (EFA) を統合する方法について説明します。Elastic Fabric Adapter (EFA) は、Amazon EC2 インスタンス向けのネットワークインターフェイスです。これにより、AWS で高レベルのノード間通信を必要とするアプリケーションを実行できます。カスタムビルドされたオペレーティングシステムのバイパスハードウェアインターフェイスにより、インスタンス間通信のパフォーマンスが向上します。これは、これらのアプリケーションのスケールリングに不可欠です。EFA を使用すると、Message Passing Interface (MPI) を使用するハイパフォーマンスコンピューティング (HPC) アプリケーションと NVIDIA Collective Communications Library (NCCL) を使用する Machine Learning (ML) アプリケーションを、数千の CPU または GPU にスケールできます。その結果、AWS クラウドのオンデマンドの伸縮自在性と柔軟性を備えたオンプレミスの HPC クラスターのアプリケーションパフォーマンスを実現できます。EFA と Amazon EKS クラスターで実行中のアプリケーションを統合すると、大規模な分散型トレーニングワークロードを完了する時間を短縮できます。クラスターにインスタンスを追加する必要はありません。EFA の詳細については、「[Elastic Fabric Adapter](#)」を参照してください。

### EFA を使用するインスタンスタイプ

AWS EFA Kubernetes Device Plugin は、EFA を持つすべての Amazon EC2 インスタンスタイプをサポートします。EFA を持つすべてのインスタンスタイプのリストを見るには、「Amazon EC2 ユーザーガイド」の「[サポートされるインスタンスタイプ](#)」を参照してください。ただし、機械学習アプリケーションを迅速に実行するために、EFAに加えて、nVidia GPU、[AWS Inferentia](#) チップ、[AWS Trainium](#) チップなどのハードウェアアクセラレーションチップもインスタンスに搭載しておくことをお勧めします。ハードウェアアクセラレーションチップと EFA を持つインスタンスタイ

プのリストを見るには、「Amazon EC2 ユーザーガイド」の「[高速コンピューティング](#)」を参照してください。

インスタンスタイプを比較して選択する場合、そのインスタンスタイプで利用できる EFA ネットワークカードの数と、アクセラレータカードの数、CPU の量、メモリの量を考慮してください。ネットワークカードごとに EFA を 1 つまで割り当てることができます。EFA はネットワークインターフェイスとしてカウントされます。EFA を持つ各インスタンスタイプで利用できる EFA の数を確認するには、「Amazon EC2 ユーザーガイド」の「[ネットワークカード](#)」のリストを参照してください。

## EFA および EFA 専用インターフェイス

Elastic Fabric Adapter (EFA) は、Elastic Network Adapter (ENA) と OS バイパスインターフェイスの機能を組み合わせたネットワークインターフェイスで、AWS Scalable Reliable Datagram (SRD) プロトコルを搭載しています。EFA 機能を使用すると、アプリケーションはハードウェアと直接通信して低レイテンシーの転送を実現できます。EFA 専用インターフェイスを使用して EFA 機能にのみアクセスすることを選択し、通信を同じアベイラビリティーゾーン内のインターフェイスに制限できます。

EFA 専用インターフェイスを持つノードを作成するには、カスタム EC2 起動テンプレートを使用して InterfaceType を efa-only に設定する必要があります。カスタム起動テンプレートでは、ネットワークカード 0 を EFA 専用インターフェイスに設定することはできません。このネットワークカードが EC2 インスタンスのプライマリネットワークカードおよびネットワークインターフェイスであるためです。EFA 専用インターフェイスには、VPC CNI バージョン 1.18.5 以降が必要です。Amazon Linux 2 を使用している場合、EFA 専用インターフェイスの AMI バージョンは v20240928 以降である必要があります。

次の手順では、nVidia GPU と EFA インターフェイスを持つノードで eksctl を使用して、EKS クラスターを作成する手順を説明します。eksctl を使用して、EFA 専用インターフェイスを使用するノードとノードグループを作成することはできません。

## 前提条件

- 既存の Amazon EKS クラスター。既存のクラスターがない場合は、[はじめに](#) を使用して作成します。クラスターは、ノードをデプロイするのに十分な IP アドレスを持つ、少なくとも 1 つのプライベートサブネットがある VPC にデプロイする必要があります。プライベートサブネットには、NAT ゲートウェイなどの外部のデバイスから提供されるアウトバウンドのインターネットアクセスが必要です。

eksctl を使用してノードグループを作成する予定がある場合、eksctl でクラスターを作成することもできます。

- ご使用のデバイスまたは AWS CloudShell で、バージョン 2.12.3 以降、または AWS コマンドラインインターフェイス (AWS CLI) のバージョン 1.27.160 以降がインストールおよび設定されていること。現在のバージョンを確認するには「aws --version | cut -d / -f2 | cut -d ' ' -f1」を参照してください。yum、apt-get、macOS 用の Homebrew などのパッケージマネージャーは、多くの場合 AWS CLI の最新バージョンより数バージョン古くなっています。最新バージョンをインストールするには「AWS コマンドラインインターフェイスユーザーガイド」の「[インストール](#)」および「[aws configure を使用したクイック設定](#)」を参照してください。AWS クラウドシェル にインストールされている AWS CLI バージョンも最新バージョンより数バージョン遅れることがあります。更新するには「AWS クラウドシェル ユーザーガイド」の「[ホームディレクトリへの AWS CLI のインストール](#)」を参照してください。
- デバイスまたは AWS クラウドシェル に、kubectl コマンドラインツールがインストールされていること。バージョンはご使用のクラスターの Kubernetes バージョンと同じか、1 つ前のマイナーバージョン以前、あるいはそれより新しいバージョンが使用できます。例えば、クラスターのバージョンが 1.29 である場合、kubectl のバージョン 1.28、1.29、または 1.30 が使用できます。kubectl をインストールまたはアップグレードする方法については「[the section called “kubectl と eksctl のセットアップ”](#)」を参照してください。
- p4d や p5 といった複数の Elastic Fabric Adapters をサポートするワーカーノードを起動する前に、Amazon VPC CNI plugin for Kubernetes バージョン 1.7.10 以降をインストールしておく必要があります。Amazon VPC CNI plugin for Kubernetes バージョンの更新の詳細については、「[the section called “Amazon VPC CNI”](#)」を参照してください。

#### Important

Kubernetes で EFA を導入するために必要となる重要な考慮事項は、クラスター内のリソースとして Huge Page を設定および管理することです。詳細については、Kubernetes ドキュメントの「[huge pageを管理する](#)」を参照してください。EFA ドライバーがインストールされた Amazon EC2 インスタンスでは、5128 2MiB Huge Page が事前に割り当てられます。このページは、ジョブの仕様で使用するリソースとしてリクエストできます。

## ノードグループの作成

次の手順は、EFA インターフェイスと GPUDirect RDMA を使用して、p4d.24xlarge がバックアップされたノードグループでノードグループを作成し、EFA を使用したマルチノード NCCL パフォーマンス向けに NVIDIA Collective Communications Library (NCCL) のサンプルテストを実行するのに役立ちます。この例では、EFA を使用した Amazon EKS の分散型深層学習トレーニングのテンプレートを使用できます。

1. ノードをデプロイしたい AWS リージョンで利用可能な、EFA をサポートする Amazon EC2 インスタンスを特定します。**region-code** を、ノードグループをデプロイする AWS リージョンに置き換えます。

```
aws ec2 describe-instance-types --region region-code \
 --filters Name=network-info.efa-supported,Values=true \
 --query "InstanceTypes[*].[InstanceType]" --output text
```

ノードをデプロイする場合、デプロイするインスタンスタイプは、ユーザーのクラスターがある AWS リージョンで利用可能である必要があります。

2. デプロイするインスタンスタイプが利用可能なアベイラビリティゾーンを特定します。このチュートリアルでは、p5.48xlarge インスタンスタイプが使用されているため、前のステップで指定した AWS リージョンの出力が返されているはずです。本番クラスターにノードをデプロイするときは、**p5.48xlarge** を前のステップで返されたいずれかのインスタンスタイプに置き換えます。

```
aws ec2 describe-instance-type-offerings --region region-code \
 --location-type availability-zone --filters Name=instance-
type,Values=p4d.24xlarge,p5.48xlarge \
 --query 'InstanceTypeOfferings[*].Location' --output text
```

出力例は次のとおりです。

```
us-west-2a us-west-2c us-west-2b
```

後のステップで使用するために、返されたアベイラビリティゾーンをメモします。ノードをクラスターにデプロイする場合、VPC には、出力で返されるアベイラビリティゾーンの 1 つに使用可能な IP アドレスを持つサブネットが必要です。

3. eksctl を使用してノードグループを作成します。デバイスまたは AWS CloudShell にインストールされている eksctl コマンドラインツールのバージョン 0.210.0 以降が必要で

す。eksctl をインストールまたはアップグレードするにはeksctl ドキュメントの「[インストール](#)」を参照してください。

- a. 次の内容を *efa-cluster.yaml* という名前のファイルにコピーします。### は実際に使用する値に置き換えます。*p5.48xlarge* を異なるインスタンスで置き換えることができますが、その場合は、availabilityZones の値がステップ 1 でインスタンスタイプに対して返されたアベイラビリティゾーンであることを確認してください。

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

metadata:
 name: my-efa-cluster
 region: region-code
 version: "1.XX"

iam:
 withOIDC: true

availabilityZones: ["us-west-2a", "us-west-2c"]

managedNodeGroups:
- name: my-efa-ng
 instanceType: p5.48xlarge
 minSize: 1
 desiredCapacity: 2
 maxSize: 3
 availabilityZones: ["us-west-2a"]
 volumeSize: 300
 privateNetworking: true
 efaEnabled: true
```

- b. 既存のクラスターにマネージド型ノードグループを作成します。

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

既存のクラスターがない場合は、次のコマンドを実行してクラスターとノードグループを作成できます。

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

 Note

この例で使用されているインスタンスタイプには GPU が含まれているため、Amazon Linux 2 を使用するときは eksctl で NVIDIA Kubernetes デバイスプラグインが各インスタンスに自動的にインストールされます。NVIDIA デバイスプラグインが Bottlerocket の EKS NVIDIA バリエーションに組み込まれているため、このプラグインは Bottlerocket には必要ありません。ノードグループ設定で efaEnabled が true に設定されている場合、eksctl では EFA デバイスプラグインもノードに自動的にデプロイされます。

## EFA で Bottlerocket を使用する

Bottlerocket AMI バージョン 1.28.0 以降には、EFA の公式サポートが含まれています。EFA 対応ノードに Bottlerocket を使用するには、設定で amiFamily: Bottlerocket を指定します。カスタム AMI ID を使用する必要がある場合は、managedNodeGroups の代わりに標準の nodeGroups を使用する必要があります。

設定の例を次に示します。

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

metadata:
 name: my-efa-bottlerocket-cluster
 region: region-code
 version: "1.XX"

iam:
 withOIDC: true

availabilityZones: ["us-west-2a", "us-west-2c"]

managedNodeGroups:
 - name: my-efa-bottlerocket-ng
 instanceType: p5.48xlarge
 minSize: 1
 desiredCapacity: 2
 maxSize: 3
 availabilityZones: ["us-west-2a"]
```

```

volumeSize: 300
privateNetworking: true
efaEnabled: true
amiFamily: Bottlerocket
bottlerocket:
 enableAdminContainer: true
 settings:
 kernel:
 sysctl:
 "vm.nr_hugepages": "3000" # Configures 3000 * 2Mi = 6000Mi hugepages

```

上記の `vm.nr_hugepages` `sysctl` 設定で、2Mi hugepages の数を指定します。この例で、3000 は  $3000 \times 2\text{Mi} = 6000\text{Mi}$  の hugepages を意味します。

### EFA デバイスプラグインのインストールを確認する

`efaEnabled: true` を使用してノードグループを作成すると、`eksctl` によって EFA Kubernetes デバイスプラグインが自動的にデプロイされます。デバイスプラグインが正しくインストールされ、機能していることを確認できます。

#### 1. DaemonSet のステータスを確認します。

```
kubectl get daemonsets -n kube-system
```

#### サンプル出力:

| NAME                                | DESIRED | CURRENT | READY | UP-TO-DATE |
|-------------------------------------|---------|---------|-------|------------|
| aws-efa-k8s-device-plugin-daemonset | 2       | 2       | 2     | 2          |
| AVAILABLE    NODE SELECTOR    AGE   |         |         |       |            |
| <none>                      6m16s   |         |         |       |            |
| ...                                 |         |         |       |            |

ここでは、EFA デバイスプラグインの DaemonSet が 2 つのノードで実行されています。どちらも READY と AVAILABLE です。

#### 2. 次に、DaemonSet によって作成されたポッドを確認します。

```
kubectl get pods -n kube-system -l name=aws-efa-k8s-device-plugin
```

#### サンプル出力:



| NAME                                      | READY | STATUS  | RESTARTS | AGE   |
|-------------------------------------------|-------|---------|----------|-------|
| aws-efa-k8s-device-plugin-daemonset-d68bs | 1/1   | Running | 0        | 6m16s |
| aws-efa-k8s-device-plugin-daemonset-w4l8t | 1/1   | Running | 0        | 6m16s |

EFA デバイスプラグインポッドが実行中状態であり、プラグインが正常にデプロイされ、動作していることを確認します。

### 3. リソース登録を確認します。

`vpc.amazonaws.com/efa` リソースが kubelet に登録されていることを確認するには、ノードを記述します。

```
kubectl describe nodes
```

EFA リソースが正しく登録されている場合、ノードのキャパシティと割り当て可能なリソースの下に一覧表示されます。例:

```
Capacity:
...
vpc.amazonaws.com/efa: 4
Allocatable:
...
vpc.amazonaws.com/efa: 4
```

この出力では、ノードが EFA リソースを認識し、それをリクエストするポッドで 사용할 수 있는ようになっていることが確認されます。

## (オプション) EFA のパフォーマンスをテストする

EFA セットアップをテストすることをお勧めします。GitHub の `aws-samples/awesome-distributed-training` リポジトリで [NCCL テスト](#) を使用できます。[NCCL テスト](#) は、Nvidia Collective Communication Library を使用してネットワークのパフォーマンスを評価します。次のステップでは、Amazon EKS で NCCL テストを送信します。

### 1. Kubeflow MPI Operator のデプロイ:

NCCL テストでは、Kubeflow MPI Operator を適用できます。MPI Operator を使用すると、Kubernetes で AllReduce スタイルの分散型トレーニングを簡単に実行できます。詳細については、GitHub の「[MPI Operator](#)」を参照してください。

## 2. マルチノードの NCCL パフォーマンステストを実行して、GPUDirectRDMA/EFA を検証する:

GPUDirectRDMA over EFA を使用した NCCL パフォーマンスを検証するには、標準の NCCL パフォーマンステストを実施します。詳細については、GitHub で公式の「[NCCL-Tests](#)」(NCCL テスト)を参照してください。

2 ノード NCCL パフォーマンステストを実行するには、次の手順を実行します。サンプルの NCCL テストジョブでは、各ワーカーが 8 つの GPU、5210Mi の hugepages-2Mi、4 つの EFA、8000Mi のメモリをリクエストします。これは事実上、各ワーカーが p5.48xlarge インスタンスのすべてのリソースを使用することを意味します。

### a. MPIJob マニフェストを作成する:

次の JSON を `nccl-tests.yaml` という名前のファイルにコピーします:

```
apiVersion: kubeflow.org/v2beta1
kind: MPIJob
metadata:
 name: nccl-tests
spec:
 runPolicy:
 cleanPodPolicy: Running
 backoffLimit: 20
 slotsPerWorker: 8
 mpiReplicaSpecs:
 Launcher:
 replicas: 1
 template:
 spec:
 restartPolicy: OnFailure
 containers:
 - image: public.ecr.aws/hpc-cloud/nccl-tests:latest
 imagePullPolicy: IfNotPresent
 name: test-nccl-launcher
 env:
 - name: PATH
 value: $PATH:/opt/amazon/efa/bin:/usr/bin
 - name: LD_LIBRARY_PATH
```

```

 value: /opt/amazon/openmpi/lib:/opt/nccl/build/lib:/opt/amazon/efa/
lib:/opt/aws-ofi-nccl/install/lib:/usr/local/nvidia/lib:$LD_LIBRARY_PATH
 - name: NCCL_DEBUG
 value: INFO
 - name: NCCL_BUFFSIZE
 value: '8388608'
 - name: NCCL_P2P_NET_CHUNKSIZE
 value: '524288'
 - name: NCCL_TUNER_PLUGIN
 value: /opt/aws-ofi-nccl/install/lib/libnccl-ofi-tuner.so
 command:
 - /opt/amazon/openmpi/bin/mpirun
 - --allow-run-as-root
 - --tag-output
 - -np
 - "16"
 - -N
 - "8"
 - --bind-to
 - none
 - -x
 - PATH
 - -x
 - LD_LIBRARY_PATH
 - -x
 - NCCL_DEBUG=INFO
 - -x
 - NCCL_BUFFSIZE
 - -x
 - NCCL_P2P_NET_CHUNKSIZE
 - -x
 - NCCL_TUNER_PLUGIN
 - --mca
 - pml
 - ^cm,ucx
 - --mca
 - btl
 - tcp,self
 - --mca
 - btl_tcp_if_exclude
 - lo,docker0,veth_def_agent
 - /opt/nccl-tests/build/all_reduce_perf
 - -b
 - "8"

```

```
- -e
- "16G"
- -f
- "2"
- -g
- "1"
- -c
- "1"
- -n
- "100"

Worker:
 replicas: 2
 template:
 spec:
 nodeSelector:
 node.kubernetes.io/instance-type: "p5.48xlarge"
 containers:
 - image: public.ecr.aws/hpc-cloud/nccl-tests:latest
 imagePullPolicy: IfNotPresent
 name: nccl-tests-worker
 volumeMounts:
 - name: shm
 mountPath: /dev/shm
 resources:
 limits:
 nvidia.com/gpu: 8
 hugepages-2Mi: 5120Mi
 vpc.amazonaws.com/efa: 32
 memory: 32000Mi
 requests:
 nvidia.com/gpu: 8
 hugepages-2Mi: 5120Mi
 vpc.amazonaws.com/efa: 32
 memory: 32000Mi
 volumes:
 - name: shm
 hostPath:
 path: /dev/shm
```

b. NCCL テスト MPIJob を適用する:

マニフェストを適用して MPIJob を送信します。これにより、2 つの p5.48xlarge Amazon EC2 インスタンスが作成されます。

```
kubectl apply -f nccl-tests.yaml
```

出力例は次のとおりです。

```
mpijob.kubeflow.org/nccl-tests created
```

c. ジョブがポッドを開始したことを検証する:

実行中の Pod を表示します。

```
kubectl get pods
```

出力例は次のとおりです。

| NAME                      | READY | STATUS   | RESTARTS | AGE   |
|---------------------------|-------|----------|----------|-------|
| nccl-tests-launcher-nbql9 | 0/1   | Init:0/1 | 0        | 2m49s |
| nccl-tests-worker-0       | 1/1   | Running  | 0        | 2m49s |
| nccl-tests-worker-1       | 1/1   | Running  | 0        | 2m49s |

MPI Operator は、ランチャー Pod と 2 つのワーカー Pod (ノードごとに 1 つ) を作成します。

d. ログを使用してジョブが正常に実行されていることを検証する:

nccl-tests-launcher Pod のログを表示します。**nbql9** を出力の値で置き換えます。

```
kubectl logs -f nccl-tests-launcher-nbql9
```

テストが正常に完了すると、Nvidia Collective Communication Library を使用するアプリケーションをデプロイできます。

## 機械学習のための Amazon EKS で AWS Inferentia インスタンスを使用する

このトピックでは[Amazon EC2 Inf1](#) インスタンスを実行中のノードで Amazon EKS クラスターを作成する方法、(オプションで) サンプルアプリケーションをデプロイする方法について説明します。Amazon EC2 Inf1 インスタンスは [AWS インフエクエンティア](#) チップを利用しています。Inf1 チップはクラウド内で高パフォーマンスと低コストの推論を提供するために、AWS によってカスタムビルドされたものです。機械学習モデルは[AWS Neuron](#) を使用してコンテナにデプロイされま

す。Neuron はコンパイラー、ランタイム、および インフェクエンティア チップの機械学習推論パフォーマンスを最適化するプロファイリングツールで構成される software development kit (SDK です。AWSNeuron はTensorFlow、PyTorch、MXNet などの一般的な機械学習フレームワークをサポートしています。

#### Note

Neuron デバイスの論理 ID は連続している必要があります。複数の Neuron デバイスをリクエストする Pod が、(複数の Neuron デバイスを持つ) `inf1.6xlarge` または `inf1.24xlarge` インスタンスタイプでスケジュールされている場合、Kubernetes スケジューラが連続していないデバイス ID を選択すると、その Pod は起動に失敗します。詳細については、GitHub の「[Device logical IDs must be contiguous](#)」を参照してください。

## 前提条件

- コンピュータに `eksctl` がインストールされている。インストールされていない場合は `eksctl` ドキュメントの「[インストール](#)」を参照してください。
- コンピュータに `kubectl` がインストールされている。詳細については「[the section called “kubectl と eksctl のセットアップ”](#)」を参照してください。
- (オプション) コンピュータに `python3` がインストールされている。インストールされていない場合は「[Python ダウンロード](#)」でインストール手順を参照してください。

## クラスターを作成する

1. `Inf1` Amazon EC2 インスタンスを実行するノードを持つクラスターを作成しま

す。`inf1.2xlarge` を任意の [Inf1 インスタンスタイプ](#) に置き換えることができます。`eksctl` ユーティリティは `Inf1` インスタンスタイプでノードグループを起動していることを検出し、アクセラレーションされた Amazon EKS 最適化 Amazon Linux AMI のいずれかを使用してノードを起動します。

#### Note

TensorFlow Serving で [サービスアカウント用の IAM 役割](#) を使用することはできません。

```
eksctl create cluster \
```

```
--name inferentia \
--region region-code \
--nodegroup-name ng-inf1 \
--node-type inf1.2xlarge \
--nodes 2 \
--nodes-min 1 \
--nodes-max 4 \
--ssh-access \
--ssh-public-key your-key \
--with-oidc
```

### Note

出力の次の行の値をメモします。これは後の (オプションの) ステップで使用されます。

```
[9] adding identity "arn:aws:iam::111122223333:role/eksctl-inferentia-nodegroup-ng-
in-NodeInstanceRole-FI7HIYS3BS09" to auth ConfigMap
```

Inf1 インスタンスでノードグループを起動すると、eksctl によって AWS Neuron Kubernetes デバイスプラグインが自動的にインストールされます。このプラグインは、Neuron デバイスをシステムリソースとして Kubernetes スケジューラにアドバタイズします。このスケジューラはコンテナによってリクエストできます。デフォルトの Amazon EKS ノードの IAM ポリシーに加えて、Amazon S3 読み取り専用のアクセスポリシーが追加されます。これにより、後のステップで説明するサンプルアプリケーションで Amazon S3 からトレーニングされたモデルをロードできるようになります。

## 2. すべての Pod が正しく起動していることを確認します。

```
kubectl get pods -n kube-system
```

簡略化された出力:

| NAME                                 | READY | STATUS  | RESTARTS | AGE |
|--------------------------------------|-------|---------|----------|-----|
| [...]                                |       |         |          |     |
| neuron-device-plugin-daemonset-6djhp | 1/1   | Running | 0        | 5m  |
| neuron-device-plugin-daemonset-hwjsj | 1/1   | Running | 0        | 5m  |

## (オプション) TensorFlow Serving アプリケーションイメージのデプロイ

トレーニング済みモデルはインフェクエンティア インスタンスにデプロイする前に、インフェクエンティア ターゲットにコンパイルする必要があります。続行するにはAmazon S3 に保存されている [Neuron optimized TensorFlow](#) モデルが必要になります。SavedModel がまだない場合は[Neuron 互換の ResNet50 モデルの作成](#)についてのチュートリアルに従って、結果の SavedModel を S3 にアップロードします。ResNet-50 は画像認識のタスクに使用される一般的な機械学習モデルです。Neuron モデルのコンパイルについての詳細は「AWS ディープラーニング AMI デベロッパーガイド」の「[AWS インフェクエンティア チップと DLAMI](#)」を参照してください。

サンプルのデプロイメントマニフェストではAWS ディープラーニング Containers によって提供される TensorFlow の事前構築済みの推論サービングコンテナが管理されます。コンテナ内には、AWS Neuron ランタイムと TensorFlow Serving アプリケーションがあります。Neuron 用に最適化された事前構築済みの Deep Learning Containers の完全なリストは、GitHub で[利用可能なイメージ](#)に保持されます。起動時に、DLC は Amazon S3 からモデルを取得し、保存したモデルで Neuron TensorFlow Serving を起動して、予測リクエストのために待機します。

サービングアプリケーションに割り当てられる Neuron デバイスの数はデプロイメント yaml で `aws.amazon.com/neuron` リソースを変更することで調整できます。TensorFlow Serving と Neuron ランタイム間の通信は GRPC 経由で行われることに注意してください。GRPC ではIPC\_LOCK 機能をコンテナに追加する必要があります。

1. 「[クラスターを作成する](#)」のステップ 1 で作成したノードのインスタンス役割に AmazonS3ReadOnlyAccess IAM ポリシーを追加します。これはサンプルアプリケーションが Amazon S3 からトレーニングされたモデルをロードできるようにするために必要です。

```
aws iam attach-role-policy \
 --policy-arn arn:aws:iam::aws:policy/AmazonS3ReadOnlyAccess \
 --role-name eksctl-inferentia-nodegroup-ng-in-NodeInstanceRole-FI7HIYS3BS09
```

2. 次の内容で、`rn50_deployment.yaml` という名前のファイルを作成します。希望の設定に合わせて、リージョンコードとモデルパスを更新します。モデル名はクライアントが TensorFlow サーバーにリクエストを行う際の識別を目的としています。この例ではモデル名を使用してサンプルの ResNet50 のクライアントスクリプトと照合します。このスクリプトは後のステップで予測リクエストを送信する際に使用されます。

```
aws ecr list-images --repository-name neuron-rtd --registry-id 790709498068 --region
us-west-2
```



```
kind: Deployment
apiVersion: apps/v1
metadata:
 name: eks-neuron-test
 labels:
 app: eks-neuron-test
 role: master
spec:
 replicas: 2
 selector:
 matchLabels:
 app: eks-neuron-test
 role: master
 template:
 metadata:
 labels:
 app: eks-neuron-test
 role: master
 spec:
 containers:
 - name: eks-neuron-test
 image: 763104351884.dkr.ecr.us-east-1.amazonaws.com/tensorflow-inference-
neuron:1.15.4-neuron-py37-ubuntu18.04
 command:
 - /usr/local/bin/entrypoint.sh
 args:
 - --port=8500
 - --rest_api_port=9000
 - --model_name=resnet50_neuron
 - --model_base_path=s3://${your-bucket-of-models}/resnet50_neuron/
 ports:
 - containerPort: 8500
 - containerPort: 9000
 imagePullPolicy: IfNotPresent
 env:
 - name: AWS_REGION
 value: "us-east-1"
 - name: S3_USE_HTTPS
 value: "1"
 - name: S3_VERIFY_SSL
 value: "0"
 - name: S3_ENDPOINT
 value: s3.us-east-1.amazonaws.com
```

```
- name: AWS_LOG_LEVEL
 value: "3"
resources:
 limits:
 cpu: 4
 memory: 4Gi
 aws.amazon.com/neuron: 1
 requests:
 cpu: "1"
 memory: 1Gi
securityContext:
 capabilities:
 add:
 - IPC_LOCK
```

### 3. モデルをデプロイします。

```
kubectl apply -f rn50_deployment.yaml
```

### 4. 次の内容で、rn50\_service.yaml というファイルを作成します。予測リクエストを受け入れるために HTTP ポートと gRPC ポートが開かれます。

```
kind: Service
apiVersion: v1
metadata:
 name: eks-neuron-test
 labels:
 app: eks-neuron-test
spec:
 type: ClusterIP
 ports:
 - name: http-tf-serving
 port: 8500
 targetPort: 8500
 - name: grpc-tf-serving
 port: 9000
 targetPort: 9000
 selector:
 app: eks-neuron-test
 role: master
```

### 5. TensorFlow モデル提供アプリケーション用の Kubernetes サービスを作成します。

```
kubectl apply -f rn50_service.yaml
```

## (オプション) TensorFlow Serving サービスに対する予測を行う

1. ローカルでテストするにはgRPC ポートを eks-neuron-test サービスに転送します。

```
kubectl port-forward service/eks-neuron-test 8500:8500 &
```

2. 次の内容で tensorflow-model-server-infer.py という Python スクリプトを作成します。  
このスクリプトはサービスフレームワークである gRPC を介して推論を実行してください。

```
import numpy as np
import grpc
import tensorflow as tf
from tensorflow.keras.preprocessing import image
from tensorflow.keras.applications.resnet50 import preprocess_input
from tensorflow_serving.apis import predict_pb2
from tensorflow_serving.apis import prediction_service_pb2_grpc
from tensorflow.keras.applications.resnet50 import decode_predictions

if __name__ == '__main__':
 channel = grpc.insecure_channel('localhost:8500')
 stub = prediction_service_pb2_grpc.PredictionServiceStub(channel)
 img_file = tf.keras.utils.get_file(
 "./kitten_small.jpg",
 "https://raw.githubusercontent.com/aws-labs/mxnet-model-server/master/docs/
images/kitten_small.jpg")
 img = image.load_img(img_file, target_size=(224, 224))
 img_array = preprocess_input(image.img_to_array(img)[None, ...])
 request = predict_pb2.PredictRequest()
 request.model_spec.name = 'resnet50_inf1'
 request.inputs['input'].CopyFrom(
 tf.make_tensor_proto(img_array, shape=img_array.shape))
 result = stub.Predict(request)
 prediction = tf.make_ndarray(result.outputs['output'])
 print(decode_predictions(prediction))
```

3. スクリプトを実行して、予測をサービスに送信します。

```
python3 tensorflow-model-server-infer.py
```

出力例は次のとおりです。

```
[[('n02123045', 'tabby', 0.68817204), ('n02127052', 'lynx', 0.12701613),
 ('n02123159', 'tiger_cat', 0.08736559), ('n02124075', 'Egyptian_cat',
 0.063844085), ('n02128757', 'snow_leopard', 0.009240591)]]
```

## EKS に機械学習ワークロードをデプロイするためのチュートリアルを試す

EKS での機械学習プラットフォームとフレームワークのセットアップに関心がある場合はこのページに記載されているチュートリアルを詳しくご覧ください。これらのチュートリアルではGPU プロセッサを最大限に活用するためのパターンから、モデリングツールの選択、専門的な業界向けのフレームワークの構築まで、あらゆる事項について説明します。

### EKS で生成 AI プラットフォームを構築する

- [Deploy Generative AI Models on アマゾン EKS](#)
- [Building multi-tenant JupyterHub Platforms on アマゾン EKS](#)

### EKS で専門的な生成 AI フレームワークを実行する

- [Accelerate your generative AI distributed training workloads with the NVIDIA NeMo Framework on アマゾン EKS](#)
- [Running TorchServe on アマゾン エラスティックKubernetesサービス](#)

### ML on EKS の NVIDIA GPU パフォーマンスを最大化する

- EKS クラスターで NVIDIA GPU を効率的に使用するために、GPU 共有を実装します:

[GPU sharing on アマゾン EKS with NVIDIA time-slicing and accelerated EC2 instances](#)

- EKS クラスターで GPU ごとにより多くのポッドを実行するために、マルチインスタンス GPU (MIG) と NIM マイクロサービスを使用します:

[Maximizing GPU utilization with NVIDIA's Multi-Instance GPU \(MIG\) on アマゾン EKS: Running more pods per GPU for enhanced performance](#)

- [Build and deploy a scalable machine learning system on Kubernetes with Kubeflow on AWS](#)

## EKS で動画エンコーディングワークロードを実行する

- [Delivering video content with fractional GPUs in containers on アマゾン EKS](#)

## 推論ワークロードのイメージロードを高速化する

- [How H2O.ai optimized and secured their AI/ML infrastructure with Karpenter and Bottlerocket](#)

## ML ワークロードのモニタリング

- [Monitoring GPU workloads on アマゾン EKS using AWS managed open-source services](#)
- [Enable pod-based GPU metrics in アマゾン CloudWatch](#)

## ML on EKS に関するお知らせ

- [Bottlerocket が NVIDIA GPU をサポートしました](#)
- [Utilizing NVIDIA Multi-Instance GPU \(MIG\) in アマゾン EC2 P4d Instances on アマゾン エラスティックKubernetesサービス](#)
- [Deploying managed P4d Instances in アマゾン エラスティックKubernetesサービス with NVIDIA GPUDirectRDMA](#)

# オープンソースプロジェクトで Amazon EKS の機能を拡張する

これらのオープンソースプロジェクトは、Amazon EKS によって管理されているクラスターを含め、AWS 上や外部で実行されている Kubernetes クラスターの機能を拡張します。

## EKS にデプロイされたソフトウェアのサポート

Amazon EKS のドキュメントを確認すると、手順や例を使用した、さまざまなオープンソースのツールやソフトウェアへの参照があります。これらのツールには、[Kubernetes メトリクスサーバー](#)や [Cert Manager](#) も含まれています。

デプロイするサードパーティーソフトウェアまたはオープンソースソフトウェアは、AWS サポート契約の範囲外であることに注意してください。Kubernetes を使用する利点は、アクティブなオープンソースコミュニティであるということです。関連するオープンソースコミュニティやプロジェクトの管理者と直接連携し、そのようなコンポーネントに適切なサポートチャネルを確立することをお勧めします。詳細については、Cloud Native Computing Foundation (CNCF) に関連付けられた [Graduated Projects](#) および [Incubating Projects](#) を参照してください。

Kubernetes エコシステムには、さまざまなレベルのコミュニティサポート、応答時間、および目的のユースケースを備えたプロジェクトやコンポーネントが多数含まれています。これらのテクノロジーを EKS と一緒に実装する場合は、各コンポーネントに対するサポートマトリックスについて確実に理解しておいてください。

AWS では、EKS コントロールプレーンに統合するオープンソースコンポーネントを管理しています。これには、配布するすべてのコンテナイメージとバイナリのビルド検証、脆弱性スキャン、検証テスト、パッチ管理を網羅する包括的なセキュリティのパイプラインが含まれます。例えば、AWS は [Kubernetes API サーバー](#) に対する責任を負います。したがって、Kubernetes API サーバーは [Amazon EKS サービスレベルアグリーメント](#) の対象となります。[Amazon Web Services サポートプラン](#) を使用して、Kubernetes API サーバーの問題を解決したり、一般的なガイダンスを取得したりできます。

さまざまな Amazon EKS アドオンで提供されているサポートを慎重に確認する必要があります。AWS のアドオンは、AWS で完全にサポートされている Amazon EKS アドオンの唯一のタイプです。AWS マーケットプレイスのアドオンは、主に AWS パートナーによってサポートされています。コミュニティのアドオンは、AWS から基本的なライフサイクルサポートを受けます。詳細については、「[アドオンのサポート](#)」を参照してください。

マーケットプレイスのアドオンなど、アドオンのタイプに関係なくすべての EKS アドオンが EKS から基本的なライフサイクルサポートを受けます。基本的なライフサイクルサポートには、アドオンのインストールとアンインストールが含まれます。使用可能な Amazon EKS アドオンのタイプと関連するサポートレベルの詳細については、「[Amazon EKS アドオンのサポート範囲](#)」を参照してください。AWS で完全にサポートされているアドオンを表示するには、「[Amazon Web Services のアドオン](#)」を参照してください。

- セキュリティプラクティスとサポート境界の詳細については、「[Amazon EKS のセキュリティ](#)」を参照してください。
- Amazon EKS アドオンで利用可能なコミュニティのアドオンと AWS のマーケットプレイスのアドオンの詳細については、「[EKS アドオンのサポート](#)」を参照してください。

## 管理ツール

Amazon EKS および Kubernetes クラスターに関連する管理ツール。

### eksctl

eksctl は、Amazon EKS 上にクラスターを作成するためのシンプルな CLI ツールです。

- [プロジェクト URL](#)
- [プロジェクトドキュメント](#)
- AWS オープンソースブログ: 「[eksctl: 1 つのコマンドによる Amazon EKS クラスター](#)」

## Kubernetes 用の AWS コントローラー

Kubernetes 用 AWS コントローラーを使用すると、Kubernetes クラスターから、直接 AWS リソースを作成および管理できます。

- [プロジェクト URL](#)
- AWS オープンソースブログ: [AWS Service Operator for Kubernetes Now Available \(AWS Service Operator for Kubernetes の使用開始\)](#)

## Flux CD

Flux は Git を使用して、クラスター設定の管理に使用できるツールです。クラスター内のオペレータを使用して、Kubernetes 内のデプロイをトリガーします。オペレーターの詳細については、GitHub の「[OperatorHub.io](https://operatorhub.io)」を参照してください。

- [プロジェクト URL](#)
- [プロジェクトドキュメント](#)

## Kubernetes 用 CDK

Kubernetes 用 CDK (cdk8s) を使用すると、使い慣れたプログラミング言語を使用して Kubernetes アプリとコンポーネントを定義できます。cdk8s アプリは標準の Kubernetes マニフェストに合成され、どの Kubernetes クラスターにも適用できます。

- [プロジェクト URL](#)
- [プロジェクトドキュメント](#)
- AWS コンテナのブログ: 「[cdk8s+ の紹介: Kubernetes オブジェクト用のインテント駆動型 API](#)」

## ネットワーク

Amazon EKS および Kubernetes クラスターに関連するネットワークプロジェクト。

### Amazon VPC CNI Plugin for Kubernetes

Amazon EKS は、Amazon VPC CNI Plugin for Kubernetes を使用した、従来の VPC ネットワークをサポートしています。プラグインは、VPC から各 Pod に IP アドレスを割り当てます。

- [プロジェクト URL](#)
- [プロジェクトドキュメント](#)

### AWS Load Balancer Controller for Kubernetes

AWS Load Balancer Controller は、Kubernetes クラスターの AWS Elastic Load Balancers の管理に役立ちます。AWS Application Load Balancers をプロビジョニングすることで、Kubernetes Ingress リソースを満たします。AWS Network Load Balancers をプロビジョニングすることで、Kubernetes Service リソースを満たします。



- [プロジェクト URL](#)
- [プロジェクトドキュメント](#)

## ExternalDNS

ExternalDNS は、公開された Kubernetes サービスとそのインGRESを、Amazon Route 53 や AWS Service Discovery を含む DNS プロバイダーと同期させます。

- [プロジェクト URL](#)
- [プロジェクトドキュメント](#)

## 機械学習

Amazon EKS および Kubernetes クラスターに関連する機械学習プロジェクト。

## Kubeflow

Kubernetes 用の機械学習ツールキット。

- [プロジェクト URL](#)
- [プロジェクトドキュメント](#)
- AWS オープンソースブログ: 「[Amazon EKS での Kubeflow](#)」

## Auto Scaling

Amazon EKS および Kubernetes クラスターに関連する自動スケーリングプロジェクト。

## Cluster Autoscaler

Cluster Autoscaler は、CPU とメモリーの負荷に基づいて Kubernetes クラスターのサイズを自動的に調整するツールです。

- [プロジェクト URL](#)
- [プロジェクトドキュメント](#)
- Amazon EKS ワークショップ: [Cluster Autoscaler](#)

## Karpenter

Karpenter は、柔軟性、パフォーマンス、シンプルさを実現するために構築された Kubernetes Node Autoscaler です。

- [プロジェクト URL](#)
- [プロジェクトドキュメント](#)
- Amazon EKS ワークショップ: [Karpenter](#)

## Escalator

Escalator は、Kubernetes 用のバッチまたはジョブ最適化水平オートスケーラーです。

- [プロジェクト URL](#)
- [プロジェクトドキュメント](#)

## モニタリング

Amazon EKS および Kubernetes クラスターに関連するモニタリングプロジェクト。

## Prometheus

Prometheus はオープンソースのシステムモニタリングおよび警告ツールキットです。

- [プロジェクト URL](#)
- [プロジェクトドキュメント](#)
- Amazon EKS ワークショップ: [https://eksworkshop.com/intermediate/240\\_monitoring/](https://eksworkshop.com/intermediate/240_monitoring/)

## 継続的インテグレーション/継続的デプロイメント

Amazon EKS および Kubernetes クラスターに関連する CI/CD プロジェクト。

## Jenkins X

Amazon EKS および Kubernetes クラスター上の最新のクラウドアプリケーション用 CI/CD ソリューション。

- [プロジェクト URL](#)
- [プロジェクトドキュメント](#)

# Amazon EKS 新機能とロードマップの詳細

Amazon EKS の新機能については、[AWS の最新情報](#) ページの新機能フィードをスクロールし、確認してください。また、GitHub の [roadmap \(ロードマップ\)](#) を参照してください。そうすることで今後の機能と優先度について知り、今後の Amazon EKS の使用方法についての計画を立てることができます。ロードマップの優先度について、直接フィードバックをお寄せください。

## ドキュメント履歴

次の表は、Amazon EKS ユーザーガイドの主な更新と新機能を示しています。このテーブルが新しいエントリを取得したときに通知を受け取るには、RSS リーダーを使用して次の URL をサブスクライブできます。

<https://docs.aws.amazon.com/eks/latest/userguide/doc-history.rss>

| 変更                                                                  | 説明                                                                                                                                                                                                                      | 日付              |
|---------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----------------|
| <a href="#">Amazon EKS Auto Mode の NodeClass への更新</a>               | Auto Mode ノードの NodeClass テンプレートに、個別のポッドサブネットの設定が追加されました。これにより、オプションのキー `podSubnetSelectorTerms` と `podSecurityGroupSelectorTerms` が追加され、ポッドのサブネットとセキュリティグループが設定されます。                                                    | 2025 年 6 月 13 日 |
| <a href="#">EKS Pod Identity を使用してセカンダリロールとクロスアカウントロールをターゲットにする</a> | Amazon EKS は、ターゲット IAM ロールを EKS Pod Identity に追加して、ロールの連鎖を自動化します。これを使用して、別のアカウントのロールを自動的に引き受けると、EKS Pod Identity は一時的な認証情報をローテーションします。各 Pod Identity 関連付けには、最初に引き受ける同じアカウントに IAM ロールが必要です。次に、そのロールを使用してターゲットロールを引き受けます。 | 2025 年 6 月 11 日 |

### [Amazon EKS プラットフォームのバージョンの更新](#)

これはセキュリティの修正および機能強化が行われた新しいプラットフォームバージョンです。これらのプラットフォームバージョンには、Kubernetes の新しいパッチのバージョンはありません。

2025 年 6 月 11 日

### [Amazon EKS AWS リージョンの拡張](#)

Amazon EKS がアジアパシフィック (台北) (ap-east-2) AWS リージョンで利用可能になりました。

2025 年 6 月 6 日

### [新しい IPv6 クラスターのデュアルスタックパブリックエンドポイントの IPv6 アクセスコントロール](#)

2025 年 6 月 5 日

### [Amazon EKS プラットフォームのバージョンの更新](#)

これはセキュリティの修正および機能強化が行われた新しいプラットフォームバージョンです。これにはKubernetes 1.32.5、1.31.9、1.30.13の新しいパッチバージョンが含まれます。

2025 年 5 月 30 日

### [EKS Hybrid Nodes の新しいクラスターインサイト](#)

Amazon EKS は、ハイブリッドノードの設定を確認する新しいクラスターインサイトを追加します。これらのインサイトチェックでは、オンプレミスノード、ポッド、およびクラスターのリモートネットワーク設定に関する問題について警告します。

2025 年 5 月 29 日

|                                                      |                                                                                                |                 |
|------------------------------------------------------|------------------------------------------------------------------------------------------------|-----------------|
| <a href="#">Kubernetes バージョン 1.33</a>                | Kubernetes バージョン 1.33 の新しいクラスターとバージョンアップグレードのサポートが追加されました。                                     | 2025 年 5 月 29 日 |
| <a href="#">Amazon FSx CSI ドライバーのアドオンサポート</a>        | AWS マネジメントコンソール、AWS CLI、API を使用して Amazon FSx CSI ドライバーを管理できるようになりました。                          | 2025 年 5 月 23 日 |
| <a href="#">コンソールで Prometheus スクレイパーを編集する</a>        | Amazon EKS コンソールで Amazon Managed Service for Prometheus スクレイパーを編集できるようになりました。                  | 2025 年 5 月 22 日 |
| <a href="#">マネージドポリシーの更新</a>                         | AmazonEKSDashboard ServiceRolePolicy ポリシーを追加しました。                                              | 2025 年 5 月 21 日 |
| <a href="#">Amazon EKS プラットフォームのバージョンの更新</a>         | これはセキュリティの修正および機能強化が行われた新しいプラットフォームバージョンです。これらのプラットフォームバージョンには、Kubernetes の新しいパッチのバージョンはありません。 | 2025 年 5 月 16 日 |
| <a href="#">Amazon FSx for Lustre のパフォーマンスの新規ページ</a> | Amazon FSx for Lustre のパフォーマンスの最適化に関する詳細を記載した、新しいトピックが追加されました。                                 | 2025 年 5 月 2 日  |

[Amazon EKS Auto Mode の  
NodeClass への更新](#)

Auto Mode ノードの NodeClass テンプレートに、フォワードネットワークプロキシの設定が追加されました。これにより、HTTP S プロキシを設定するためのオプションキー `advancedNetworking` が追加されます。

2025 年 4 月 30 日

[ハイブリッドノードの  
Bottlerocket](#)

Bottlerocket が EKS Hybrid Nodes で利用できるようになりました。

2025 年 4 月 29 日

[Amazon EKS プラットフォー  
ムのバージョンの更新](#)

これはセキュリティの修正および機能強化が行われた新しいプラットフォームバージョンです。これらのプラットフォームバージョンには、Kubernetes の新しいパッチのバージョンはありません。

2025 年 4 月 29 日

[ハイブリッドネットワークの  
新しいコンセプトページ](#)

EKS Hybrid Nodes の概念に関するページを追加しました。オンプレミスとクラウドのネットワークについて、図を使いながら詳細に解説しています。

2025 年 4 月 18 日



## [AWS マネージドポリシーの更新](#)

AmazonEKSClusterPolicy に権限を追加して、VPC CNI によって作成された Elastic Network Interface に Amazon EKS がアクセスできるようにしました。こうした権限付与が必要なのは、VPC CNI が予期せず終了した場合に、残った Elastic Network Interface を EKS によってクリーンアップするためです。

2025 年 4 月 16 日

## [AWS マネージドポリシーの更新](#)

AmazonEKSServiceRolePolicy にアクセス権限を追加して、EKS AI/ML のお客様がデフォルトの EKS クラスターセキュリティグループに Egress ルールを追加できるようにしました。

2025 年 4 月 14 日

## [EKS Hybrid Nodes のノードのヘルス](#)

1.2.0-eksbuild.1 バージョン以降は、eks-node-monitoring-agent をハイブリッドノードで使用できます。eks-node-monitoring-agent を Amazon EKS アドオンとして実行することで、健全性に関する問題を検出し表示できます。

2025 年 3 月 31 日

## [既存のクラスターの EKS Hybrid Nodes](#)

既存のクラスターのハイブリッドノード設定を追加、変更、削除できるようになりました。以前は、クラスターを作成する際にしか、ハイブリッドノード設定を新しいクラスターに追加できませんでした。Amazon EKS Hybrid Nodes ではオンプレミスおよびエッジインフラストラクチャを Amazon EKS クラスターのノードとして使用できます。AWS は Amazon EKS クラスターの AWS ホスト型 Kubernetes コントロールプレーンを管理し、オンプレミス環境またはエッジ環境で実行されるハイブリッドノードを管理します。

2025 年 3 月 31 日

## [ロールバック: クラスターインサイトによる偶発的なアップグレードの防止](#)

Amazon EKS では、特定のクラスターインサイトの問題が発生した際に `--force` フラグでのクラスターアップグレードが必要となる機能が一時的にロールバックされています。詳細については、Git Hub の「[Temporary rollback of enforcing upgrade insights on update cluster version](#)」を参照してください。

2025 年 3 月 28 日

## [Bottlerocket FIPS AMI](#)

Bottlerocket FIPS AMI が、標準のマネージド型ノードグループで利用可能になりました。

2025 年 3 月 27 日

## [AWS マネージドポリシーの更新](#)

Amazon EKS が Auto Mode で作成された EC2 インスタンスを終了できるよう、AmazonEKSServiceRolePolicy へのアクセス許可を追加しました。Amazon EKS が Auto Mode で作成された EC2 インスタンスを終了できるよう、AmazonEKS ServiceRolePolicy へのアクセス許可を追加しました。

2025 年 2 月 28 日

## [マネージド型ノードグループの戦略の更新](#)

更新戦略を使用して、マネージド型ノードグループのバージョン更新プロセスを設定できるようになりました。新しいノードを作成する前にノードを終了する最小限の更新戦略は、容量に制約のある環境で役立ちます。デフォルトの更新戦略では、既存の動作が継続されます。

2025 年 1 月 27 日

## [Kubernetes バージョン 1.32](#)

Kubernetes バージョン 1.32 の新しいクラスターとバージョンアップグレードのサポートが追加されました。

2025年1月23日

## [Amazon EKS AWS リージョ ンの拡張](#)

Amazon EKS が、アジアパシ  
フィック (タイ) リージョン  
( ) およびメキシコ (中部ap-  
southeast-7 ) (mx-centra  
l-1 ) AWSリージョンで利用  
可能になりました。EKS API  
の EKS 自動モードと VPC エ  
ンドポイントはどちらのリー  
ジョンでも利用できません。

2025年1月14日

## [AWS マネージドポリシーの更 新](#)

Amazon EBS CSI ド  
ライバーがすべての  
スナップショットを復  
元AmazonEBSCSIDriver  
Policy し、EBS ボリューム  
で高速スナップショット復元  
(FSR) 有効にし、ボリューム  
のタグを変更できるようにす  
る複数のアクセス許可を に追  
加しました。

2025年1月13日

## [AWS マネージドポリシーの更 新](#)

許可を AmazonEKS  
LoadBalancingPolicy  
に追加しました。

2024 年 12 月 26 日

## [クラスターインサイトの更新](#)

Amazon EKS のアップグレー  
ドインサイトはクラスターの  
健全性とバージョンの互換性  
に関するより多くの問題に  
ついて警告するようになります。  
kubelet、kube-prox  
y、Amazon EKS アドオンな  
ど、さまざまな Kubernetes と  
Amazon EKS コンポーネント  
間の問題を検出できます。

2024 年 12 月 20 日

## [ノードモニタリングエージェントと自動修復](#)

新しい eks-node-monitoring-agent を Amazon EKS アドオンとして使用して、健全性に関する問題を検出および表示できます。ノードの自動修復を有効にして、問題が検出されたときノードを自動的に置き換えることもできます。

2024 年 12 月 16 日

## [Amazon EKS Hybrid Nodes](#)

Amazon EKS クラスターに接続されたノードをオンプレミスで実行できるようになりました。Amazon EKS Hybrid Nodes ではオンプレミスおよびエッジインフラストラクチャを Amazon EKS クラスターのノードとして使用できます。AWS は Amazon EKS クラスターの AWS ホスト型 Kubernetes コントロールプレーンを管理し、オンプレミス環境またはエッジ環境で実行されるハイブリッドノードを管理します。

2024 年 12 月 1 日

## [Amazon EKS 自動モードI](#)

Amazon EKS 自動モードI は AWS でのコンピューティング、ストレージ、ネットワーキングのための Kubernetes クラスターインフラストラクチャ管理を完全に自動化します。インフラストラクチャの自動プロビジョニング、最適なコンピューティングインスタンスの選択、リソースの動的なスケーリング、継続的なコストの最適化、オペレーティングシステムへのパッチ適用、AWS セキュリティサービスとの統合により、Kubernetes の管理を簡素化します。

2024 年 12 月 1 日

## [Amazon EKS プラットフォームバージョンの更新](#)

これはセキュリティの修正および機能強化が行われた新しいプラットフォームバージョンです。これにはKubernetes 1.31.2、1.30.6、1.29.10、1.28.15 の新しいパッチバージョンが含まれます。

2024 年 11 月 22 日

## [AWS マネージドポリシーの更新](#)

AWSServiceRoleForAmazonEKSNodegroup が中国リージョンとの互換性のために更新されました。

2024 年 11 月 22 日

## [Kubernetes バージョン 1.30 が AWS アウトポストのローカルクラスターで利用可能に](#)

Kubernetes バージョン 1.30 を使用して AWS アウトポストに Amazon EKS ローカルクラスターを作成できるようになりました。

2024 年 11 月 21 日

### [AWS マネージドポリシーの更新](#)

EKS が AWS マネージドポリシー AmazonEKSLocalOutpostClusterPolicy を更新しました。クラスターコントロールプレーンでの AWS クラウド Controller Manager が各ノードが存在するアベイラビリティゾーンを識別できるように、ec2:DescribeAvailabilityZones 許可を追加しました。

2024 年 11 月 21 日

### [FIPS 140-3 を使用する Bottlerocket AMI](#)

Bottlerocket AMI は FIPS 140-3 検証済み暗号化モジュールを使用するように事前設定されています。これには Amazon Linux 2023 Kernel Crypto API 暗号化モジュールや AWS-LC 暗号化モジュールなどがあります。

2024 年 11 月 20 日

### [AWS マネージドポリシーの更新](#)

Amazon EKS マネージドノードグループによって作成されたインスタンスのために ec2:RebootInstances を許可するように AWSServiceRoleForAmazonEKSNodegroup ポリシーを更新しました。Amazon EC2 リソースについての ec2:CreateTags 許可を制限しました。

2024 年 11 月 20 日

## [オブザーバビリティダッシュボード](#)

オブザーバビリティダッシュボードは問題を迅速に検出、トラブルシューティング、是正するのに役立ちます。また、AWS/EKS 名前空間では新しい [クラウドWatch Vended メトリクス](#)を使用することもできます。

2024 年 11 月 18 日

## [AWS マネージドポリシーの更新](#)

EKS が AWS マネージドポリシー AmazonEKSServiceRolePolicy を更新しました。EKS アクセスポリシー、ロードバランサーの管理、クラスターリソースの自動クリーンアップの許可を追加しました。

2024 年 11 月 16 日

## [EKS Pod Identity をサポートするアドオンのための、コンソールでの新しいロールの作成](#)

コンソールを使用して EKS ポッドアイデンティティをサポートするアドオンを作成または更新する場合、新しいステップがあります。このステップではアドオンのために適切な名前、ロールポリシー、信頼ポリシーを使用して IAM ロールを自動的に生成できます。

2024 年 11 月 15 日

## [AWS ローカルゾーン のマネージドノードグループ](#)

マネージドノードグループをAWS ローカルゾーンで作成できるようになりました。

2024 年 11 月 15 日

## [新しいメトリクスが使用可能に](#)

API グループ `metrics.eks.amazonaws.com` の下に使用可能な新しいメトリクスがあります。

2024 年 11 月 11 日



[AWS マネージドポリシーの更新](#)

EKS が AWS マネージドポリシー AmazonEKSComputePolicy を更新しました。iam:AddRoleToInstanceProfile アクションのリソース許可を更新しました。

2024 年 11 月 7 日

[AWS マネージドポリシーの更新](#)

EKS に、新しい AWS マネージドポリシー が追加されました : AmazonEKSComputePolicy

2024 年 11 月 1 日

[AWS マネージドポリシーの更新](#)

許可を AmazonEKSClusterPolicy に追加しました。Amazon EKS がトポロジ情報をラベルとしてノードにアタッチできるようにする ec2:DescribeInstanceTopology 許可を追加しました。

2024 年 11 月 1 日

[AWS マネージドポリシーの更新](#)

EKS に、新しい AWS マネージドポリシー が追加されました : AmazonEKSBlockStoragePolicy

2024 年 10 月 30 日

[AWS マネージドポリシーの更新](#)

EKS に、新しい AWS マネージドポリシー が追加されました : AmazonEKSLoadBalancingPolicy

2024 年 10 月 30 日

## [AWS マネージドポリシーの更新](#)

Amazon EKS が Amazon CloudWatch にメトリクスを発行できるようにする `cloudwatch:PutMetricData` 許可を AmazonEKS `ServiceRolePolicy` に追加しました。

2024 年 10 月 29 日

## [AWS マネージドポリシーの更新](#)

EKS に、新しい AWS マネージドポリシー が追加されました : `AmazonEKSNetworkingPolicy`

2024 年 10 月 28 日

## [新しい IPv6 クラスターのデュアルスタックエンドポイント](#)

デュアルスタックの `eks-cluster. region.api.aws` エンドポイントを使用して新しい IPv6 クラスターに接続します。このエンドポイントはこれらのクラスターを記述するときに返されます。IPv4、IPv6、またはデュアルスタック環境の `kubectl` およびその他の Kubernetes API クライアントはパブリッククラスターまたはプライベートクラスターのこれらのエンドポイントを解決して接続できます。

2024 年 10 月 21 日

## [AWS マネージドポリシーの更新](#)

`autoscaling:ResumeProcesses`、`autoscaling:SuspendProcesses` および関連するアクセス許可を中国リージョンの `AWSServiceRoleForAmazonEKSNodegroup` に追加し、EKS 用の Amazon Application Recovery Controller と統合しました。他のリージョンに変更はありません。

2024 年 10 月 21 日

## [AL2023 高速 AMI](#)

AL2023 に基づいて高速 NVIDIA と AWS Neuron インスタンスを AMI に使用できるようになりました。

2024 年 10 月 11 日

## [新しいソース形式](#)

レイアウトが変更された新しいソース形式に切り替えました。対処中の一時的で軽微なフォーマットの問題があります。

2024 年 10 月 10 日

## [AWS マネージドポリシーの更新](#)

`AmazonEKSServicePolicy` および `AmazonEKSServiceRolePolicy` にアクセス許可を追加しました。EKS がセキュリティグループ情報を読み取り、関連するタグを更新できるように、`ec2:GetSecurityGroupsForVpc` および関連するタグの許可を追加しました。

2024 年 10 月 10 日

|                                                                    |                                                                                                                                                                                                                                             |                 |
|--------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----------------|
| <a href="#">AWS マネージドポリシーの更新 – 新しいポリシー</a>                         | EKS に、新しい AWS マネージドポリシーが追加されました。                                                                                                                                                                                                            | 2024 年 10 月 3 日 |
| <a href="#">Kubernetes バージョン 1.31</a>                              | Kubernetes バージョン 1.31 の新しいクラスターとバージョンアップグレードのサポートが追加されました。                                                                                                                                                                                  | 2024 年 9 月 24 日 |
| <a href="#">AWS マネージドポリシーの更新 – 既存のポリシーへの更新</a>                     | Amazon EKS において、既存の AWS マネージドポリシーの更新を行いました。                                                                                                                                                                                                 | 2024 年 8 月 21 日 |
| <a href="#">Kubernetes バージョン 1.29 が AWS アウトポストのローカルクラスターで利用可能に</a> | Kubernetes バージョン 1.29 を使用して AWS アウトポストに Amazon EKS ローカルクラスターを作成できるようになりました。                                                                                                                                                                 | 2024 年 8 月 20 日 |
| <a href="#">AWS Govクラウド (米国) での EKS Pod Identity</a>               | Amazon EKS Pod Identity は IAM ロールを Kubernetes サービスアカウントに関連付けます。この機能を使用すると、ノード IAM ロールに拡張されたアクセス許可を提供する必要がなくなります。このようにして、そのノード上の Pod は AWS API を呼び出すことができます。サービスアカウントの IAM ロールとは異なり、EKS Pod アイデンティティは完全に EKS 内部にあるため、OIDC ID プロバイダーは必要ありません。 | 2024 年 8 月 14 日 |

### [シナリオ駆動型コンテンツの更新](#)

ガイド全体でよりシナリオ駆動型になるようにトピックの名前を変更および更新しました。

2024 年 8 月 9 日

### [Amazon EKS のデュアルスタック VPC インターフェイスエンドポイント](#)

Amazon EKS のデュアルスタック VPC インターフェイスエンドポイントを、IPv4 および IPv6 IP アドレスと DNS 名の両方で作成できるようになりました。

2024 年 8 月 7 日

### [IPv6 アドレスを持つ Amazon EKS API の新しいデュアルスタックエンドポイント](#)

クラスターを作成および管理するための EKS API、およびクラスターの OIDC 発行者 URL には新しいデュアルスタックエンドポイントがあります。Amazon EKS API の新しい DNS 名は `eks.region.api.aws` であり、これは IPv4 アドレスと IPv6 アドレスに解決されます。新しいクラスターには新しいデュアルスタック OIDC 発行者 URL (`oidc-eks.region.api.aws`) があります。

2024 年 8 月 1 日

### [マネージド型ノードグループのキャパシティブロック](#)

マネージド型ノードグループにキャパシティブロックを使用できるようになりました。

2024 年 7 月 1 日

### [デフォルトで 自動スケーリング グループのメトリクス収集を有効化](#)

Amazon EKS マネージドノードグループでは追加料金なしで Amazon EC2 Auto Scaling グループのメトリクスがデフォルトで有効になりました。以前はこの機能を有効にするためにいくつかのステップを実行する必要がありました。

2024 年 6 月 28 日

### [AWS マネージドポリシーの更新 – 既存のポリシーへの更新](#)

Amazon EKS において、既存の AWS マネージドポリシーの更新を行いました。

2024 年 6 月 27 日

### [Kubernetes バージョン 1.26](#)

Kubernetes バージョン 1.26 が延長サポート対象になりました。

2024 年 6 月 12 日

### [AMI 情報リファレンスの改善](#)

AMI 情報リファレンス、特に Bottlerocket が改善されました。

2024 年 6 月 12 日

### [Kubernetes バージョン 1.30](#)

Kubernetes バージョン 1.30 の新しいクラスターとバージョンアップグレードのサポートが追加されました。

2024 年 5 月 23 日

### [CoreDNS のオートスケーリング](#)

CoreDNS オートスケーラーはノードと CPU コアの数に基づいて、EKS クラスター内の CoreDNS デプロイのレプリカ数を動的に調整します。この機能は CoreDNS v1.9 および EKS リリースバージョン 1.25 以降の最新プラットフォームバージョンで機能します。

2024 年 5 月 14 日

[Amazon EKS プラットフォームバージョンの更新](#)

これはセキュリティの修正および機能強化が行われた新しいプラットフォームバージョンです。これにはKubernetes 1.29.4、1.28.9、1.27.13の新しいパッチバージョンが含まれます。

2024 年 5 月 14 日

[Windows での CloudWatch Container Insights のサポート](#)

Amazon CloudWatch Observability Operator アドオンはクラスター内の Windows ワーカーノードでも Container Insights を許可するようになりました。

2024 年 4 月 10 日

[Kubernetes の概念](#)

Kubernetes コンセプトのトピックが新たに追加されました。

2024 年 4 月 5 日

[アクセスと IAM コンテンツを再構築する](#)

認証設定マップ、アクセスエントリ、ポッド ID、IRSA など、アクセスと IAM トピックに関連する既存のページが新しいセクションに移動します。概要の内容を改訂しました。

2024 年 4 月 2 日

[Amazon S3 CSI ドライバーの Bottlerocket OS サポート](#)

Mountpoint for Amazon S3 CSI ドライバーが Bottlerocket と互換性を持つようになりました。

2024 年 3 月 13 日

[AWS マネージドポリシーの更新 – 既存のポリシーへの更新](#)

Amazon EKS において、既存の AWS マネージドポリシーの更新を行いました。

2024 年 3 月 4 日

[Amazon Linux 2023](#)

Amazon Linux 2023 (AL2023) はクラウドアプリケーションに安全かつ安定した高パフォーマンス環境を提供するように設計された、新しい Linux ベースのオペレーティングシステムです。

2024 年 2 月 29 日

[Kubernetes 1.29 で EKS Pod Identity と IRSA がサイドカーをサポート](#)

Kubernetes 1.29 ではサイドカーコンテナは Amazon EKS クラスターで使用できます。サイドカーコンテナはサービスアカウントまたは EKS Pod Identity の IAM ロールでサポートされています。サイドカーの詳細については、Kubernetes ドキュメントの「[サイドカーコンテナ](#)」を参照してください。

2024 年 2 月 26 日

[Kubernetes バージョン 1.29](#)

Kubernetes バージョン 1.29 の新しいクラスターとバージョンアップグレードのサポートが追加されました。

2024 年 1 月 23 日

[フルリリース: Kubernetes バージョンに対する Amazon EKS の延長サポート](#)

Kubernetes バージョンに対する延長サポートにより、特定の Kubernetes バージョンを 14 か月よりも長く使用することができます。

2024 年 1 月 16 日



## [AWS クラウドでの Amazon EKS クラスターヘルスの検出](#)

Amazon EKS はクラスターヘルス内の、Amazon EKS クラスターとクラスターの前提条件のインフラストラクチャの問題を検出します。EKS API ではAWS Management Console 内およびクラスターの health 内の EKS クラスターの問題が確認できます。これらの問題はコンソールによって検出され表示される問題に追加されます。以前はクラスターヘルスは AWS アウトポスト 上のローカルクラスターのみで利用可能でした。

2023 年 12 月 28 日

## [クラスターのインサイト](#)

定期的なチェックに基づいて、クラスターに関する推奨事項を取得できるようになりました。

2023 年 12 月 20 日

## [Amazon EKS AWS リージョンの拡張](#)

Amazon EKS がカナダ西部 (カルガリー) (ca-west-1 ) AWS リージョンで利用可能になりました。

2023 年 12 月 20 日

### [IAM ロールとユーザーに、アクセスエントリを使用してクラスターへのアクセス権を付与できるようになりました。](#)

アクセスエントリを導入する前はaws-auth ConfigMap にエントリを追加して IAM ロールとユーザーにクラスターへのアクセス権を付与していました。これで、各クラスターにアクセスモードが用意され、スケジュールに従ってアクセスエントリを使用するように切り替えることができるようになりました。モードを切り替えたら、AWS CLI、AWS クラウドFormation、AWS SDK にアクセスエントリを追加してユーザーを追加できます。

2023 年 12 月 18 日

### [Amazon EKS プラットフォームバージョンの更新](#)

これはセキュリティの修正および機能強化が行われた新しいプラットフォームバージョンです。これにはKubernetes 1.28.4、1.27.8、1.26.11、1.25.16 の新しいパッチバージョンが含まれます。

2023 年 12 月 12 日

### [Mountpoint for Amazon S3 CSI ドライバー](#)

Amazon EKS クラスターで Mountpoint for Amazon S3 CSI ドライバーをインストールできるようになりました。

2023 年 11 月 27 日

### [クラスターの作成時に Prometheus メトリクスを有効にする](#)

AWS Management Console ではクラスターの作成時に Prometheus メトリクスをオンにできるようになりました。Prometheus スクレイパーの詳細は [オブザーバビリティ] タブでも表示できます。

2023 年 11 月 26 日

[Amazon EKS Pod Identity](#)

Amazon EKS Pod Identity は IAM ロールを Kubernetes サービスアカウントに関連付けます。この機能を使用すると、ノード IAM ロールに拡張されたアクセス許可を提供する必要がなくなります。このようにして、そのノード上の Pod は AWS API を呼び出すことができます。サービスアカウントの IAM ロールとは異なり、EKS Pod アイデンティティは完全に EKS 内部にあるため、OIDC ID プロバイダーは必要ありません。

2023 年 11 月 26 日

[AWS 管理ポリシーの更新 – 既存のポリシーへの更新](#)

Amazon EKS において、既存の AWS マネージドポリシーの更新を行いました。

2023 年 11 月 26 日

[CSI スナップショットコントローラー](#)

CSI スナップショットコントローラーをインストールして、Amazon EBS CSI ドライバーなどの互換性のある CSI ドライバーで使えるようになりました。

2023 年 11 月 17 日

[ADOT オペレーターピックリライト](#)

ADOT Operator セクションの Amazon EKS アドオンサポートは AWS Distro for OpenTelemetry のドキュメントと重複していました。古い情報や一貫性のない情報を減らすために、残りの重要な情報をそのリソースに移行しました。

2023 年 11 月 14 日

### [Prometheus メトリクスの CoreDNS EKS アドオンサ ポート](#)

CoreDNS 用の EKS アドオンの v1.10.1-eksbuild.5 、v1.9.3-eksbuild.9 、v1.8.7-eksbuild.8 バージョンでは、CoreDNS がメトリクスを公開したポートが kube-dns サービス内で公開されます。これにより、モニタリングシステムに CoreDNS メトリクスを組み込むことが容易になります。

2023 年 11 月 10 日

### [Amazon EKS CloudWatch オ ブザーバビリティオペレー ターアドオン](#)

Amazon EKS クラウドWatch オブザーバビリティオペレーターページを追加しました。

2023 年 11 月 6 日

### [米国東部 \(オハイオ\) のセルフ マネージド型 P5 インスタン スのキャパシティブロック](#)

米国東部 (オハイオ) では米国東部 (オハイオ) のセルフマネージド型 P5 インスタンスにキャパシティブロックを使用できるようになりました。

2023 年 10 月 31 日

### [クラスターでのサブネットおよびセキュリティグループの変更のサポート](#)

クラスターを更新して、クラスターが使用するサブネットとセキュリティグループを変更できます。AWS Management Console、AWS CLI の最新バージョン、AWS クラウドFormation、および eksctl バージョン v0.164.0-rc.0 以降から更新できます。クラスターバージョンを正常にアップグレードするにはこれを実行して、サブネットに利用可能な IP アドレスを増やす必要がある場合があります。

2023 年 10 月 24 日

### [クラスターロールおよびマネージドノードグループロールでのカスターマネージド AWS アイデンティティとアクセス管理 ポリシーのサポート](#)

クラスターロールでは [AmazonEKSClusterPolicy](#) AWS マネージドポリシーの代わりにカスタム IAM ポリシーを使用できます。また、マネージドノードグループのノードロールでは [AmazonEKSWorkerNodePolicy](#) AWS マネージドポリシーの代わりにカスタム IAM ポリシーを使用できます。これにより、厳格なコンプライアンス要件を満たす最小特権でポリシーを作成できます。

2023 年 10 月 23 日

### [eksctl インストールへのリンクの修正](#)

ページが移動された後の eksctl のインストールリンクを修正しました。

2023 年 10 月 6 日

[プレビューリリース:  
Kubernetes バージョンに対す  
る Amazon EKS の延長サポー  
ト](#)

Kubernetes バージョンに対す  
る延長サポートにより、特定  
の Kubernetes バージョンを  
14 か月よりも長く使用するこ  
とができます。

2023 年 10 月 4 日

[AWS App Mesh 統合へのリ  
ファレンスを削除](#)

Amazon EKS と AWS App  
Mesh の統合は引き続き App  
Mesh の既存のお客様のみを対  
象としています。

2023 年 9 月 29 日

[Kubernetes バージョン 1.28](#)

Kubernetes バージョン 1.28  
の新しいクラスターとバー  
ジョンアップグレードのサ  
ポートが追加されました。

2023 年 9 月 26 日

[既存のクラスターが、Amazon  
VPC CNI plugin for Kubernete  
s で Kubernetes ネットワーク  
ポリシーの適用をサポート](#)

サードパーティーのソリユー  
ションの代わりに、既存の  
クラスターにある Kubernete  
s のネットワークポリシーと  
Amazon VPC CNI plugin for  
Kubernetes を使用できます。  
サードパーティーのソリユー  
ションの代わりに、既存の  
クラスターにある Kubernete  
s のネットワークポリシーと  
Amazon VPC CNI plugin for  
Kubernetes を使用できます。

2023 年 9 月 15 日

[CoreDNS Amazon EKS アド  
オンが PDB の変更をサポート](#)

バージョン v1.9.3-ek  
sbuild.7 以降および  
v1.10.1-eksbuild.4  
以降では CoreDNS の EKS  
アドオンの PodDisrup  
tionBudget を変更できま  
す。

2023 年 9 月 15 日

|                                                                                  |                                                                                                                                                                                        |                 |
|----------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----------------|
| <a href="#">Amazon EKS での共有サブネットのサポート</a>                                        | 共有サブネット内に Amazon EKS クラスターを作成するための、新しい「 <a href="#">共有サブネットの要件と考慮事項</a> 」。                                                                                                             | 2023 年 9 月 7 日  |
| <a href="#">「Amazon EKS とは」の更新</a>                                               | 新規トピック「 <a href="#">一般的なユースケース</a> 」と「 <a href="#">アーキテクチャ</a> 」の追加。その他に複数のトピックの更新。                                                                                                    | 2023 年 9 月 6 日  |
| <a href="#">Amazon VPC CNI plugin for Kubernetes での Kubernetes ネットワークポリシーの適用</a> | サードパーティーのソリューションの代わりに、Kubernetes のネットワークポリシーと Amazon VPC CNI plugin for Kubernetes を使用できます。サードパーティーのソリューションの代わりに、Kubernetes のネットワークポリシーと Amazon VPC CNI plugin for Kubernetes を使用できます。 | 2023 年 8 月 29 日 |
| <a href="#">Amazon EKS AWS リージョンの拡張</a>                                          | Amazon EKS がイスラエル (テルアビブ) (il-central-1 ) AWS リージョンで利用可能になりました。                                                                                                                        | 2023 年 8 月 1 日  |
| <a href="#">設定可能な Fargate 用エフェメラルストレージ</a>                                       | Amazon EKS Fargate で実行される各 Pod 用エフェメラルストレージの総量を増やすことができます。                                                                                                                             | 2023 年 7 月 31 日 |
| <a href="#">Amazon EFS CSI ドライバーのアドオンサポート</a>                                    | AWS Management Console、AWS CLI、API を使用して Amazon EFS CSI ドライバーを管理できるようになりました。                                                                                                           | 2023 年 7 月 26 日 |

|                                                                                            |                                                                                               |                 |
|--------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------|-----------------|
| <a href="#">AWS 管理ポリシーの更新 – 新しいポリシー</a>                                                    | Amazon EKS に、新しい AWS 管理ポリシーが追加されました。                                                          | 2023 年 7 月 26 日 |
| <a href="#">1.27、1.26、1.25、1.24 向けの Kubernetes バージョンアップデートが、AWS アウトポストのローカルクラスターで利用可能に</a> | 1.27.3、1.26.6、1.25.11、1.24.15 向けの Kubernetes バージョンアップデートが、AWS アウトポストのローカルクラスターで利用できるようになりました。 | 2023 年 7 月 20 日 |
| <a href="#">Windows ノードの IP プレフィックスサポート</a>                                                | ノードに IP プレフィックスを割り当てると、個別のセカンダリ IP アドレスをノードに割り当てる場合よりもはるかに多くの数の Pod をノードでホストできます。             | 2023 年 7 月 6 日  |
| <a href="#">Amazon FSx for OpenZFS CSI ドライバー</a>                                           | Amazon EKS クラスターで Amazon FSx for OpenZFS CSI ドライバーをインストールできるようになりました。                         | 2023 年 6 月 30 日 |
| <a href="#">IPv4 クラスター内の Linux ノード上の Pod が、IPv6 エンドポイントと通信できるようになりました。</a>                 | IPv6 アドレスがノードに割り当てられると、Pod の IPv4 アドレスは実行されているノードの IPv6 アドレスにネットワークアドレス変換されます。                | 2023 年 6 月 19 日 |
| <a href="#">AWS GovCloud (米国) リージョンの Windows マネージド型ノードグループ</a>                             | AWS GovCloud (US) リージョンでは、Amazon EKS マネージドノード型グループが Windows コンテナを実行できるようになりました。               | 2023 年 5 月 30 日 |



|                                                                    |                                                                                 |                 |
|--------------------------------------------------------------------|---------------------------------------------------------------------------------|-----------------|
| <a href="#">Kubernetes バージョン 1.27</a>                              | Kubernetes バージョン 1.27 の新しいクラスターとバージョンアップグレードのサポートが追加されました。                      | 2023 年 5 月 24 日 |
| <a href="#">Kubernetes バージョン 1.26</a>                              | Kubernetes バージョン 1.26 の新しいクラスターとバージョンアップグレードのサポートが追加されました。                      | 2023 年 4 月 11 日 |
| <a href="#">ドメインレス gMSA</a>                                        | Windows Pod でドメインレス gMSA を使用できるようになりました。                                        | 2023 年 3 月 27 日 |
| <a href="#">Amazon EKS AWS リージョンの拡張</a>                            | Amazon EKS がアジアパシフィック (メルボルン) (ap-southeast-4 ) AWS リージョンで利用可能になりました。           | 2023 年 3 月 10 日 |
| <a href="#">Amazon File Cache CSI ドライバー</a>                        | Amazon EKS クラスターに Amazon File Cache CSI ドライバーをインストールできるようになりました。                | 2023 年 3 月 3 日  |
| <a href="#">Kubernetes バージョン 1.25 が AWS アウトポストのローカルクラスターで利用可能に</a> | Kubernetes バージョン 1.22 ~ 1.25 を使用して、アウトポスト に Amazon EKS ローカルクラスターを作成できるようになりました。 | 2023 年 3 月 1 日  |
| <a href="#">Kubernetes バージョン 1.25</a>                              | Kubernetes バージョン 1.25 の新しいクラスターとバージョンアップグレードのサポートが追加されました。                      | 2023 年 2 月 22 日 |

[AWS 管理ポリシーの更新 – 既存のポリシーへの更新](#)

Amazon EKS において、既存の AWS マネージドポリシーの更新を行いました。

2023 年 2 月 7 日

[Amazon EKS AWS リージョンの拡張](#)

Amazon EKS は現在、アジアパシフィック (ハイデラバード) (ap-south-2)、欧州 (チューリッヒ) (eu-central-2)、欧州 (スペイン) (eu-south-2) の AWS リージョンで利用可能になりました。

2023 年 2 月 6 日

[AWS アウトポスト上のローカルクラスターで Kubernetes バージョン 1.21~1.24 が利用可能に。](#)

Kubernetes バージョン 1.21~1.24 を使用して、アウトポストに Amazon EKS ローカルクラスターを作成できるようになりました。以前は 1.21 バージョンのみが使用可能でした。

2023 年 1 月 17 日

[Amazon EKS が AWS プライベートリンクをサポート](#)

AWS プライベートリンクを使用して、VPC と Amazon EKS 間にプライベート接続を作成できます。

2022 年 12 月 16 日

[マネージド型ノードグループの Windows サポート](#)

Amazon EKS マネージド型ノードグループに、Windows を使用できるようになりました。

2022 年 12 月 15 日

[AWS マーケットプレイスで独立系ソフトウェアベンダーの Amazon EKS アドオンの提供を開始](#)

AWS マーケットプレイスを通じて、独立系ソフトウェアベンダーの Amazon EKS アドオンを閲覧し、サブスクライブできるようになりました。

2022 年 11 月 28 日

|                                             |                                                            |                  |
|---------------------------------------------|------------------------------------------------------------|------------------|
| <a href="#">AWS 管理ポリシーの更新 – 既存のポリシーへの更新</a> | Amazon EKS において、既存の AWS 管理ポリシーの更新を行いました。                   | 2022 年 11 月 17 日 |
| <a href="#">Kubernetes バージョン 1.24</a>       | Kubernetes バージョン 1.24 の新しいクラスターとバージョンアップグレードのサポートが追加されました。 | 2022 年 11 月 15 日 |
| <a href="#">Amazon EKS AWS リージョンの拡張</a>     | Amazon EKS が中東 (UAE) (me-central-1 ) AWS リージョンで利用可能になりました。 | 2022 年 11 月 3 日  |
| <a href="#">AWS 管理ポリシーの更新 – 既存のポリシーへの更新</a> | Amazon EKS において、既存の AWS 管理ポリシーの更新を行いました。                   | 2022 年 10 月 24 日 |
| <a href="#">AWS 管理ポリシーの更新 – 既存のポリシーへの更新</a> | Amazon EKS において、既存の AWS 管理ポリシーの更新を行いました。                   | 2022 年 10 月 20 日 |
| <a href="#">AWS アウトポスト のローカルクラスターが利用可能</a>  | アウトポスト に Amazon EKS ローカルクラスターを作成できるようになりました。               | 2022 年 9 月 19 日  |
| <a href="#">Fargate vCPU ベースのクォータ</a>       | Fargate は、Pod ベースのクォータから vCPU ベースのクォータに移行しています。            | 2022 年 9 月 8 日   |
| <a href="#">AWS 管理ポリシーの更新 – 既存のポリシーへの更新</a> | Amazon EKS において、既存の AWS 管理ポリシーの更新を行いました。                   | 2022 年 8 月 31 日  |

|                                                               |                                                                                                        |                 |
|---------------------------------------------------------------|--------------------------------------------------------------------------------------------------------|-----------------|
| <a href="#">コストモニタリング</a>                                     | Amazon EKS が Kubecost をサポートするようになりました。これにより、Pod、ノード、名前空間、ラベルなどの Kubernetes リソースごとにコストを分類して監視することができます。 | 2022 年 8 月 24 日 |
| <a href="#">AWS 管理ポリシーの更新 – 新しいポリシー</a>                       | Amazon EKS に、新しい AWS 管理ポリシーが追加されました。                                                                   | 2022 年 8 月 24 日 |
| <a href="#">AWS 管理ポリシーの更新 – 新しいポリシー</a>                       | Amazon EKS に、新しい AWS 管理ポリシーが追加されました。                                                                   | 2022 年 8 月 23 日 |
| <a href="#">請求用のタグリソース</a>                                    | すべてのクラスターで、aws:eks:cluster-name 生成のコスト配分タグがサポートされるようになりました。                                            | 2022 年 8 月 16 日 |
| <a href="#">Fargate プロファイルのワイルドカード</a>                        | 名前空間、ラベルキー、およびラベル値のセクター基準で、Fargate プロファイルのワイルドカードがサポートされるようになりました。                                     | 2022 年 8 月 16 日 |
| <a href="#">Kubernetes バージョン 1.23</a>                         | Kubernetes バージョン 1.23 の新しいクラスターとバージョンアップグレードのサポートが追加されました。                                             | 2022 年 8 月 11 日 |
| <a href="#">AWS Management Console に Kubernetes リソースを表示する</a> | AWS Management Console を使用してクラスターにデプロイされた Kubernetes リソースの情報を表示できるようになりました。                            | 2022 年 5 月 3 日  |

|                                                       |                                                                                                                    |                 |
|-------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------|-----------------|
| <a href="#">Amazon EKS AWS リージョンの拡張</a>               | Amazon EKS がアジアパシフィック (ジャカルタ) (ap-southeast-3 ) AWS リージョンで利用可能になりました。                                              | 2022 年 5 月 2 日  |
| <a href="#">観測可能性ページと ADOT アドオンのサポート</a>              | [Observability] (可観測性) ページと、AWS Distro for OpenTelemetry (ADOT) を追加。                                               | 2022 年 4 月 21 日 |
| <a href="#">Kubernetes バージョン 1.22</a>                 | Kubernetes バージョン 1.22 の新しいクラスターとバージョンアップグレードのサポートが追加されました。                                                         | 2022 年 4 月 4 日  |
| <a href="#">AWS 管理ポリシーの更新 – 新しいポリシー</a>               | Amazon EKS に、新しい AWS 管理ポリシーが追加されました。                                                                               | 2022 年 4 月 4 日  |
| <a href="#">Fargate Pod のパッチ適用の詳細を追加</a>              | Fargate Pod をアップグレードする場合、Amazon EKS はまず Pod の中断予算に基づいて Pod を削除しようとしません。Pod が削除される前に、失敗したエビクションに対応するイベントルールを作成できます。 | 2022 年 4 月 1 日  |
| <a href="#">フルリリース: Amazon EBS CSI ドライバーのアドオンサポート</a> | AWS Management Console、AWS CLI、API を使用して Amazon EBS CSI ドライバーを管理できるようになりました。                                       | 2022 年 3 月 31 日 |
| <a href="#">AWS アウトポスト コンテンツの更新</a>                   | AWS アウトポスト で Amazon EKS クラスターをデプロイする手順。                                                                            | 2022 年 3 月 22 日 |

|                                                                 |                                                                                  |                  |
|-----------------------------------------------------------------|----------------------------------------------------------------------------------|------------------|
| <a href="#">AWS 管理ポリシーの更新 – 既存のポリシーへの更新</a>                     | Amazon EKS において、既存の AWS 管理ポリシーの更新を行いました。                                         | 2022 年 3 月 21 日  |
| <a href="#">Windows containerd サポート</a>                         | Windows ノードの containerd ランタイムを選択できるようになりました。                                     | 2022 年 3 月 14 日  |
| <a href="#">Amazon EKS コネクタの考慮事項をセキュリティドキュメントに追加</a>            | 接続されたクラスターに関連する責任共有モデルについて説明しています。                                               | 2022 年 2 月 25 日  |
| <a href="#">Pod とサービスへの IPv6 アドレスの割り当て</a>                      | Pod とサービスに IPv6 アドレスを割り当てる 1.21 以降のクラスターを作成できるようになりました。                          | 2022 年 1 月 6 日   |
| <a href="#">AWS 管理ポリシーの更新 – 既存のポリシーへの更新</a>                     | Amazon EKS において、既存の AWS 管理ポリシーの更新を行いました。                                         | 2021 年 12 月 13 日 |
| <a href="#">プレビューリリース: Amazon EBS CSI ドライバーのアドオンサポート</a>        | AWS Management Console、AWS CLI、および API を使用して Amazon EBS CSI ドライバーを管理できるようになりました。 | 2021 年 12 月 9 日  |
| <a href="#">Karpenter オートスケーラーのサポート</a>                         | Karpenter オープンソースプロジェクトを使用してノードを自動スケールできるようになりました。                               | 2021 年 11 月 29 日 |
| <a href="#">Fargate ログ記録での Fluent Bit Kubernetes フィルターのサポート</a> | Fargate ログ記録で Fluent Bit Kubernetes フィルターを使用できるようになりました。                         | 2021 年 11 月 10 日 |

### [コントロールプレーンで利用可能な Windows サポート](#)

Windows サポートがコントロールプレーンで利用可能になりました。データプレーンで有効にする必要はなくなりました。

2021 年 11 月 9 日

### [マネージド型ノードグループの AMI タイプとして追加された Bottlerocket](#)

以前は、Bottlerocket はセルフマネージド型のノードオプションとしてのみ利用可能でした。マネージドノードグループとして設定できるようになったため、ノードのコンプライアンス要件を満たすために必要な労力が軽減されます。

2021 年 10 月 28 日

### [DL1 ドライバーのサポート](#)

カスタム Amazon Linux AMI が、Amazon Linux 2 の深層学習ワークロードをサポートするようになりました。この有効化により、一般的なオンプレミスまたはクラウドのベースライン設定が可能になります。

2021 年 10 月 25 日

### [VT1 ビデオのサポート](#)

カスタム Amazon Linux AMI では一部のディストリビューションで VT1 がサポートされるようになりました。この有効化により、Amazon EKS クラスター上の Xilinx U30 デバイスがアドバタイズされます。

2021 年 9 月 13 日

[Amazon EKS コネクトor が利用可能](#)

Amazon EKS Connector を使用して、準拠する Kubernetes クラスターを AWS に登録および接続し、Amazon EKS コンソールで視覚化できます。

2021 年 9 月 8 日

[Amazon EKS Anywhere が利用可能](#)

Amazon EKS Anywhere はオンプレミスの Kubernetes クラスターの作成と運用に使用できる Amazon EKS の新しいデプロイオプションです。

2021 年 9 月 8 日

[Amazon FSx for NetApp ONTAP CSI ドライバ](#)

Amazon FSx for NetApp ONTAP CSI ドライバーの概要と、その他のリファレンスへのリンクを提供するトピックを追加しました。

2021 年 9 月 2 日

[マネージド型ノードグループは、ノードの Amazon EKS 推奨最大 Pod を自動計算するようになりました](#)

マネージド型ノードグループは、起動テンプレートなしで、または AMI ID を指定していない起動テンプレートを使用して、デプロイするノードの Amazon EKS 最大 Pod を自動的に計算するようになりました。

2021 年 8 月 30 日

[Amazon EKS アドオンソフトウェアを削除せずに、アドオン設定の Amazon EKS 管理を削除してください。](#)

クラスターからアドオンソフトウェアを削除せずに Amazon EKS アドオンを削除できるようになりました。

2021 年 8 月 20 日

[Multus を使用したマルチホーム Pod の作成](#)

ポッドに対し、Multus を使用しながら複数のネットワークインターフェースを追加できるようになりました。

2021 年 8 月 2 日



### [Linux Amazon EC2 ノードに IP アドレスを追加する](#)

Linux Amazon EC2 ノードに大幅に多くの IP アドレスを追加できるようになりました。つまり、各ノードでより高い密度の Pod を実行できるということです。Linux Amazon EC2 ノードに大幅に多くの IP アドレスを追加できるようになりました。つまり、各ノードでより高い密度の Pod を実行できるということです。

2021 年 7 月 27 日

### [containerd ランタイム ブートストラップ](#)

Amazon EKS に最適化されアクセラレートされた Amazon Linux Amazon マシンイメージ (AMI) に、Amazon EKS に最適化された Bottlerocket AMI で containerd ランタイムの有効化に使用できる、ブートストラップフラグが含まれるようになりました。このフラグは、Kubernetes でサポートされているすべての AMI のバージョンで使用できます。

2021 年 7 月 19 日

### [Kubernetes バージョン 1.21](#)

Kubernetes バージョン 1.21 のサポートを追加しました。

2021 年 7 月 19 日

### [管理ポリシーのトピックを追加](#)

Amazon EKS IAM 管理ポリシーと、2021 年 6 月 17 日以降に行われたすべての変更のリスト。

2021 年 6 月 17 日

### [Fargate での Pod のセキュリティグループの使用](#)

Amazon EC2 ノードに加えて、Fargate でも Pod のセキュリティグループを使用できるようになりました。

2021 年 6 月 1 日

[CoreDNS と kube-proxy  
Amazon EKS アドオンの追加](#)

Amazon EKS は、クラスターの CoreDNS および kube-proxy Amazon EKS アドオンの管理もお手伝いできるようになりました。

2021 年 5 月 19 日

[Kubernetes バージョン 1.20](#)

Kubernetes バージョン 1.20 の新しいクラスターとバージョンアップグレードのサポートが追加されました。

2021 年 5 月 18 日

[AWS Load Balancer コン  
トローラー 2.2.0 のリリース](#)

AWS ロードバランサーコントローラーを使用して Elastic Load Balancer を作成できるようになりました。インスタンスまたは IP ターゲットを使用します。

2021 年 5 月 14 日

[マネージド型ノードグループ  
のノードテイント](#)

Amazon EKS ではマネージド型ノードグループへのノードテイントの追加がサポートされるようになりました。

2021 年 5 月 11 日

[既存クラスターのシークレ  
ット暗号化](#)

Amazon EKS では既存クラスターへの[シークレット暗号化](#)の追加がサポートされるようになりました。

2021 年 2 月 26 日

[Kubernetes バージョン 1.19](#)

Kubernetes バージョン 1.19 の新しいクラスターとバージョンアップグレードのサポートが追加されました。

2021 年 2 月 16 日

|                                                                                                                      |                                                                                                                     |                  |
|----------------------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------|------------------|
| <a href="#">Amazon EKS では、バージョン 1.16 以降のクラスターに対してユーザーを認証する方法として OpenID Connect (OIDC) ID プロバイダーがサポートされるようになりました。</a> | OIDC ID プロバイダーはAWS アイデンティティとアクセス管理 (IAM) と併用、または代替として使用できます。                                                        | 2021 年 2 月 12 日  |
| <a href="#">AWS Management Console でのノードとワークロードのリソースの表示</a>                                                          | AWS Management Console で、マネージド型ノード、セルフマネージド型ノード、Fargate ノード、およびデプロイされた Kubernetes ワークロードに関する詳細を表示できるようになりました。       | 2020 年 12 月 1 日  |
| <a href="#">マネージド型ノードグループへのスポットインスタンスタイプのデプロイ</a>                                                                    | 複数のスポットまたはオンデマンドインスタンスタイプを、マネージド型ノードグループにデプロイできるようになりました。                                                           | 2020 年 12 月 1 日  |
| <a href="#">Amazon EKS がクラスターの特定のアドオンを管理できるようになりました</a>                                                              | アドオンをお客様自身で管理することもできますが、Amazon EKS API を介してアドオンの起動とバージョンを Amazon EKS に制御させることもできます。                                 | 2020 年 12 月 1 日  |
| <a href="#">複数の Ingress 間で ALB を共有する</a>                                                                             | AWS Application Load Balancer (ALB) を複数の Kubernetes Ingress 間で共有できるようになりました。以前は、各 Ingress に個別の ALB をデプロイする必要がありました。 | 2020 年 10 月 23 日 |

|                                                                    |                                                                                                                                                                  |                  |
|--------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------------|
| <a href="#">NLB IP ターゲットのサポート</a>                                  | IP ターゲットを使用し、Network Load Balancer (NLB) をデプロイできるようになりました。これにより、NLB を使用して、Fargate Pod へのネットワークトラフィック、および Amazon EC2 ノードで実行されている Pod への直接のネットワークトラフィックの負荷分散ができます。 | 2020 年 10 月 23 日 |
| <a href="#">Kubernetes バージョン 1.18</a>                              | Kubernetes バージョン 1.18 の新しいクラスターとバージョンアップグレードのサポートが追加されました。                                                                                                       | 2020 年 10 月 13 日 |
| <a href="#">Kubernetes サービス IP アドレス割り当て用のカスタム CIDR ブロックを指定します。</a> | Kubernetes がサービス IP アドレスの割り当てるカスタム CIDR ブロックを指定できるようになりました。                                                                                                      | 2020 年 9 月 29 日  |
| <a href="#">セキュリティグループを個別の Pod に割り当てる</a>                          | 多くの Amazon EC2 インスタンスタイプで実行されている個別の Pod の一部に、異なるセキュリティグループを関連付けることができるようになりました。                                                                                  | 2020 年 9 月 9 日   |
| <a href="#">ノードに Bottlerocket をデプロイする</a>                          | <a href="#">Bottlerocket</a> を実行中のノードをデプロイできるようになりました。                                                                                                           | 2020 年 8 月 31 日  |

### [Arm ノードを起動する機能は一般的に利用可能です](#)

マネージド型ノードグループ、およびセルフマネージド型ノードグループで、Arm ノードを起動できるようになりました。

2020 年 8 月 17 日

### [マネージド型ノードグループの起動テンプレートとカスタム AMI](#)

Amazon EC2 起動テンプレートを使用して、マネージド型ノードグループをデプロイできるようになりました。起動テンプレートで、カスタム AMI を指定できます。

2020 年 8 月 17 日

### [AWS Fargate の EFS サポート](#)

AWS Fargate で Amazon EFS を使用できるようになりました。

2020 年 8 月 17 日

### [Amazon EKS プラットフォームのバージョンの更新](#)

これはセキュリティの修正および機能強化が行われた新しいプラットフォームバージョンです。これにはバージョン 1.15 以降で Network Load Balancer を使用する場合、LoadBalancer タイプのサービスに対する UDP サポートが含まれます。詳細については、GitHub で「[Allow UDP for AWS Network Load Balancer](#)」( Network Load Balancer に UDP を許可する) という問題を参照してください。

2020 年 8 月 12 日

[Amazon EKS AWS リージョ  
ンの拡張](#)

Amazon EKS がアフリカ (ケープタウン) (af-south-1 ) および欧州 (ミラノ) (eu-south-1 ) AWS リージョンで利用可能になりました。

2020 年 8 月 6 日

[Fargate 使用状況メトリクス](#)

AWS Fargate は Fargate オンデマンドリソースのアカウント使用状況を可視化する クラウドWatch 使用状況メトリクスを提供します。

2020 年 8 月 3 日

[Kubernetes バージョン 1.17](#)

Kubernetes バージョン 1.17 の新しいクラスターとバージョンアップグレードのサポートが追加されました。

2020 年 7 月 10 日

[Kubernetes 用 App Mesh コ  
ントローラーを使用して、Kuber  
netes 内から App Mesh リ  
ソースを作成し管理する](#)

Kubernetes 内から App Mesh リソースを作成し管理できます。また、コントローラーは Envoy プロキシと init コンテナを、デプロイする Pod に自動的に挿入します。

2020 年 6 月 18 日

[Amazon EKS が Amazon EC2  
Inf1 ノードをサポートするよ  
うになりました](#)

Amazon EC2 Inf1 ノードをクラスターに追加できます。

2020 年 6 月 4 日

[Amazon EKS AWS リージョ  
ンの拡張](#)

Amazon EKS が AWS Govクラウド (米国東部) (us-gov-east-1 ) および AWS Govクラウド (米国西部) (us-gov-west-1 ) AWS リージョンで利用可能になりました。

2020 年 5 月 13 日

|                                                                            |                                                                                                        |                  |
|----------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------|------------------|
| <a href="#">Kubernetes 1.12 は Amazon EKS でサポートされなくなりました</a>                | Kubernetes バージョン 1.12 は Amazon EKS でサポートされなくなりました。サービスの中断を防ぐため、すべての 1.12 クラスターをバージョン 1.13 以降に更新してください。 | 2020 年 5 月 12 日  |
| <a href="#">Kubernetes バージョン 1.16</a>                                      | Kubernetes バージョン 1.16 の新しいクラスターとバージョンアップグレードのサポートが追加されました。                                             | 2020 年 4 月 30 日  |
| <a href="#">サービスにリンクされたロール <code>AWSServiceRoleForAmazonEKS</code> を追加</a> | <code>AWSServiceRoleForAmazonEKS</code> サービスリンクロールを追加しました。                                             | 2020 年 4 月 16 日  |
| <a href="#">Kubernetes バージョン 1.15</a>                                      | Kubernetes バージョン 1.15 の新しいクラスターとバージョンアップグレードのサポートが追加されました。                                             | 2020 年 3 月 10 日  |
| <a href="#">Amazon EKS AWS リージョンの拡張</a>                                    | Amazon EKS が北京 (cn-north-1 ) および寧夏 (cn-northwest-1 ) AWS リージョンで利用可能になりました。                             | 2020 年 2 月 26 日  |
| <a href="#">FSx for Lustre CSI ドライバー</a>                                   | Kubernetes 1.14 Amazon EKS クラスターに FSx for Lustre CSI ドライバーをインストールするトピックを追加しました。                        | 2019 年 12 月 23 日 |

### [クラスターのパブリックアクセスエンドポイントへのネットワークアクセスを制限](#)

今回の更新では、Amazon EKS を使用して、Kubernetes API サーバーのパブリックアクセスエンドポイントと通信できる CIDR 範囲を制限できるようになりました。

2019 年 12 月 20 日

### [VPC 外部からクラスターのプライベートアクセスエンドポイントアドレスを解決](#)

今回の更新では、Amazon EKS を使用して、VPC 外から Kubernetes API サーバーのプライベートアクセスエンドポイントを解決できるようになりました。

2019 年 12 月 13 日

### [\(ベータ\) Amazon EC2 A1 Amazon EC2 インスタンスノード](#)

Amazon EKS クラスターに登録する [Amazon EC2 A1](#) Amazon EC2 インスタンスノードを起動します。

2019 年 12 月 4 日

### [AWS アウトポスト でのクラスターの作成](#)

Amazon EKS ではAWS アウトポスト でのクラスターの作成がサポートされるようになりました。

2019 年 12 月 3 日

### [Amazon EKS の AWS Fargate](#)

Amazon EKS Kubernetes クラスターで、Fargate での Pod の実行がサポートされるようになりました。

2019 年 12 月 3 日

### [Amazon EKS AWS リージョンの拡張](#)

Amazon EKS がカナダ (中部) (ca-central-1 ) AWS リージョンで使用可能になりました。

2019 年 11 月 21 日



|                                                             |                                                                                                           |                  |
|-------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------|------------------|
| <a href="#">マネージドノードグループ</a>                                | Amazon EKS マネージド型ノードグループは、Amazon EKS Kubernetes クラスターのノード (Amazon EC2 インスタンス) のプロビジョニングとライフサイクル管理を自動化します。 | 2019 年 11 月 18 日 |
| <a href="#">Amazon EKS プラットフォームのバージョンの更新</a>                | <a href="#">CVE-2019-11253</a> に対応する新しいプラットフォームバージョン                                                      | 2019 年 11 月 6 日  |
| <a href="#">Kubernetes 1.11 は Amazon EKS でサポートされなくなりました</a> | Kubernetes バージョン 1.11 は Amazon EKS でサポートされなくなりました。サービスの中断を防ぐため、1.11 クラスターはバージョン 1.12 以降に更新してください。         | 2019 年 11 月 4 日  |
| <a href="#">Amazon EKS AWS リージョンの拡張</a>                     | Amazon EKS が南米 (サンパウロ) (sa-east-1 ) AWS リージョンで利用可能になりました。                                                 | 2019 年 10 月 16 日 |
| <a href="#">Windows サポート</a>                                | Kubernetes バージョン 1.14 を実行する Amazon EKS クラスターが Windows ワークロードをサポートするようになりました。                              | 2019 年 10 月 7 日  |
| <a href="#">自動スケーリング</a>                                    | Amazon EKS クラスターでサポートされる異なるタイプの Kubernetes autoscaling に関する章を追加しました。                                      | 2019 年 9 月 30 日  |

[Kubernetes ダッシュボードの更新](#)

ベータ 2.0 バージョンを使用するため、Amazon EKS クラスターに Kubernetes ダッシュボードをインストールするトピックを更新しました。

2019 年 9 月 28 日

[Amazon EFS CSI ドライバー](#)

Kubernetes 1.14 Amazon EKS クラスターに Amazon EFS CSI ドライバーをインストールするトピックを追加しました。

2019 年 9 月 19 日

[Amazon EKS 最適化された AMI ID の Amazon EC2 システムマネージャー パラメータ](#)

Amazon EC2 システムマネージャー パラメータを使用して、Amazon EKS 最適化された AMI ID を取得するトピックを追加しました。このパラメータを使用すると、AMI ID を検索する必要がなくなります。

2019 年 9 月 18 日

[Amazon EKS リソースのタグ付け](#)

Amazon EKS クラスターのタグ付けを管理できます。

2019 年 9 月 16 日

[Amazon EBS CSI ドライバー](#)

Kubernetes 1.14 Amazon EKS クラスターに Amazon EBS CSI ドライバーをインストールするトピックを追加しました。

2019 年 9 月 9 日

[新しい Amazon EKS 最適化 AMI に CVE-2019-9512 および CVE-2019-9514 用のパッチを適用](#)

Amazon EKS は [CVE-2019-9512 および CVE-2019-9514](#) に対応するため、Amazon EKS 最適化された AMI を更新しました。

2019 年 9 月 6 日

## [Amazon EKS の Kubernetes 1.11 の廃止を発表](#)

Amazon EKS は 2019 年 11 月 4 日に Kubernetes バージョン 1.11 のサポートを終了しました。

2019 年 9 月 4 日

## [Kubernetes バージョン 1.14](#)

Kubernetes バージョン 1.14 の新しいクラスターとバージョンアップグレードのサポートが追加されました。

2019 年 9 月 3 日

## [サービスアカウントの IAM ロール](#)

Amazon EKS クラスターのサービスアカウントの IAM ロールを使用すると、IAM ロールを Kubernetes サービスアカウントに関連付けることができます。この機能を使用すると、ノード IAM ロールに拡張されたアクセス許可を提供する必要がなくなります。このようにして、そのノード上の Pod は AWS API を呼び出すことができます。

2019 年 9 月 3 日

## [Amazon EKS AWS リージョンの拡張](#)

Amazon EKS が中東 (バーレーン) (me-south-1 ) AWS リージョンで利用可能になりました。

2019 年 8 月 29 日

## [Amazon EKS プラットフォームのバージョンの更新](#)

[CVE-2019-9512](#) および [CVE-2019-9514](#) に対応する新しいプラットフォームバージョン。

2019 年 8 月 28 日

## [Amazon EKS プラットフォームのバージョンの更新](#)

[CVE-2019-11247](#) および [CVE-2019-11249](#) に対応する新しいプラットフォームバージョン。

2019 年 8 月 5 日

|                                                                |                                                                                                                                         |                 |
|----------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------|-----------------|
| <a href="#">Amazon EKS リージョンの拡張</a>                            | Amazon EKS がアジアパシフィック (香港) (ap-east-1) AWS リージョンで利用可能になりました。                                                                            | 2019 年 7 月 31 日 |
| <a href="#">Kubernetes 1.10 は Amazon EKS でサポートされなくなりました</a>    | Kubernetes バージョン 1.10 は Amazon EKS でサポートされなくなりました。サービスの中断を防ぐため、1.10 クラスターはバージョン 1.11 以降に更新してください。                                       | 2019 年 7 月 30 日 |
| <a href="#">ALB イングレスコントローラーに関するトピックの追加</a>                    | AWS ALB Ingress Controller for Kubernetes は Ingress リソースが作成されたときに ALB を作成するコントローラーです。                                                   | 2019 年 7 月 11 日 |
| <a href="#">新しい Amazon EKS 最適化された AMI</a>                      | 不要な kubect1 バイナリを AMI から削除します。                                                                                                          | 2019 年 7 月 3 日  |
| <a href="#">Kubernetes バージョン 1.13</a>                          | Kubernetes バージョン 1.13 の新しいクラスターとバージョンアップグレードのサポートが追加されました。                                                                              | 2019 年 6 月 18 日 |
| <a href="#">新しい Amazon EKS 最適化 AMI に AWS-2019-005 用のパッチを適用</a> | Amazon EKS は Amazon EKS 最適化 AMI を更新して、link:security/security-bulletins/AWS-2019-005/[AWS-2019-005、type="marketing"] で説明されている脆弱性に対処しました： | 2019 年 6 月 17 日 |
| <a href="#">Amazon EKS での Kubernetes 1.10 のサポートの停止を発表</a>      | Amazon EKS は、2019 年 7 月 22 日に Kubernetes バージョン 1.10 のサポートを停止しました。                                                                       | 2019 年 5 月 21 日 |

## [Amazon EKS プラットフォームのバージョンの更新](#)

kubelet 証明書のカスタム DNS 名をサポートし、etcd パフォーマンスを向上させる Kubernetes 1.11 および 1.10 クラスター用の新しいプラットフォームバージョン。

2019 年 5 月 21 日

## [eksctl の開始方法](#)

この入門ガイドは、eksctl を使用して Amazon EKS を開始するために必要な、すべてのリソースの作成に役立ちます。これは、Amazon EKS で Kubernetes クラスターを作成および管理するためのシンプルなコマンドラインユーティリティです。

2019 年 5 月 10 日

## [AWS CLI の get-token コマンド](#)

aws eks get-token コマンドが AWS CLI に追加されました。クラスター API サーバー通信用のクライアントセキュリティトークンを作成するために、AWS IAM Authenticator for Kubernetes をインストールする必要がなくなりました。この新しい機能を使用するにはインストール済みの AWS CLI を最新バージョンにアップグレードしてください。詳細については AWS コマンドラインインターフェイスユーザーガイドの「[AWS コマンドラインインターフェイスのインストール](#)」を参照してください。

2019 年 5 月 10 日

[Amazon EKS プラットフォームのバージョンの更新](#)

kubelet 証明書のカスタム DNS 名をサポートし、etcd のパフォーマンスを向上させる Kubernetes 1.12 クラスター用の新しいプラットフォームバージョン。これにより、ノードの kubelet デーモンによって、数秒ごとに新しい証明書が要求されるバグが修正されました。

2019 年 5 月 8 日

[Prometheus チュートリアル](#)

Amazon EKS クラスターへの Prometheus のデプロイのトピックが追加されました。

2019 年 4 月 5 日

[Amazon EKS コントロールプレーンのログ記録](#)

今回の更新では監査および診断ログは Amazon EKS コントロールプレーンから直接取得できるようになりました。これらの クラウドWatch ログはクラスターをセキュリティで保護および実行するための参照としてアカウントで使用できます。

2019 年 4 月 4 日

[Kubernetes バージョン 1.12](#)

Kubernetes バージョン 1.12 の新しいクラスターとバージョンアップグレードのサポートが追加されました。

2019 年 3 月 28 日

[App Mesh 入門ガイドの追加](#)

App Mesh と Kubernetes の入門用ドキュメントが追加されました。

2019 年 3 月 27 日

[Amazon EKS API サーバーエンドポイントのプライベートアクセス](#)

Amazon EKS クラスターの Kubernetes API サーバーエンドポイントのパブリックアクセスを無効にする方法に関するドキュメントが追加されました。

2019 年 3 月 19 日

[Kubernetes メトリクスサーバーのインストールに関するトピックの追加](#)

Kubernetes メトリクスサーバーは、クラスター内のリソース使用状況データを集約します。

2019 年 3 月 18 日

[関連するオープンソースプロジェクトのリストの追加](#)

これらのオープンソースプロジェクトは、Amazon EKS によって管理されているクラスターを含め、AWS で実行されている Kubernetes クラスターの機能を拡張します。

2019 年 3 月 15 日

[Helm のローカルインストールに関するトピックの追加](#)

Kubernetes 用の helm パッケージマネージャーを使用すると、Kubernetes クラスターにアプリケーションをインストールし、管理できます。このトピックではhelmおよびtillerバイナリをローカルにインストールして実行する方法について説明します。これにより、ローカルシステムでHelm CLIを使用してチャートをインストールおよび管理することができます。

2019 年 3 月 11 日

## [Amazon EKS プラットフォームのバージョンの更新](#)

[CVE-2019-1002100](#) に対応するために Amazon EKS Kubernetes 1.11 クラスターをパッチレベル 1.11.8 に更新する新しいプラットフォームバージョン。

2019 年 3 月 8 日

## [クラスターの上限の引き上げ](#)

Amazon EKS ではAWS リージョン内で作成できるクラスターの数が増えました。

2019 年 2 月 13 日

## [Amazon EKS AWS リージョンの拡張](#)

Amazon EKS が欧州 (ロンドン) (eu-west-2)、欧州 (パリ) (eu-west-3)、アジアパシフィック (ムンバイ) (ap-south-1) AWS リージョンで利用可能になりました。

2019 年 2 月 13 日

## [新しい Amazon EKS 最適化 AMI に ALAS-2019-1156 用のパッチを適用](#)

Amazon EKS は[ALAS-2019-1156](#) で説明されている脆弱性に対応するため、Amazon EKS 最適化 AMI を更新しました。

2019 年 2 月 11 日

## [新しい Amazon EKS 最適化 AMI に ALAS2-2019-1141 用のパッチを適用](#)

Amazon EKSは[ALAS2-2019-1141](#) で参照されている CVE に対応するため、Amazon EKS 最適化された AMI を更新しました。

2019 年 1 月 9 日

## [Amazon EKS AWS リージョンの拡張](#)

Amazon EKS がアジアパシフィック (ソウル) (ap-northeast-2) AWS リージョンで利用可能になりました。

2019 年 1 月 9 日



## [Amazon EKS リージョンの拡張](#)

Amazon EKS が、欧州 (フランクフルト) (eu-central-1)、アジアパシフィック (東京) (ap-northeast-1)、アジアパシフィック (シンガポール) (ap-southeast-1)、およびアジアパシフィック (シドニー) (ap-southeast-2) の AWS リージョンでも新たに利用可能になりました :

2018 年 12 月 19 日

## [Amazon EKS クラスターの更新](#)

Amazon EKS [クラスターの更新](#)および[ノード置換](#)に関するドキュメントが追加されました。

2018 年 12 月 12 日

## [Amazon EKS AWS リージョンの拡張](#)

Amazon EKS が欧州 (ストックホルム) (eu-north-1) AWS リージョンで利用可能になりました。

2018 年 12 月 11 日

## [Amazon EKS プラットフォームのバージョンの更新](#)

[link:security/security-bulletins/AWS-2018-020/\[CVE-2018-1002105,type="marketing"\]](#)に対応するために、新しいプラットフォームバージョンをパッチレベル 1.10.11 に更新しました。

2018 年 12 月 4 日

## [ALB イングレスコントローラーのバージョン 1.0.0 のサポートを追加](#)

ALB イングレスコントローラーで、AWS の公式サポートを備えたバージョン 1.0.0 がリリースされました。

2018 年 11 月 20 日

### [CNI ネットワーク設定のサポートを追加](#)

Kubernetes バージョン 1.2.1 用の Amazon VPC CNI プラグインが、セカンダリ Pod ネットワークインターフェイスのカスタムネットワーク設定をサポートするようになりました。

### [MutatingAdmissionWebhook と ValidatingAdmissionWebhook のサポートを追加](#)

Amazon EKS プラットフォームバージョン 1.10-eks.2 が MutatingAdmissionWebhook および ValidatingAdmissionWebhook アドミッションコントローラーをサポートするようになりました。

### [パートナー AMI 情報の追加](#)

Canonical は Amazon EKS と提携し、クラスターで使用できるノード AMI を作成しました。

### [AWS CLI の update-kubeconfig コマンドの手順を追加](#)

Amazon EKS で update-kubeconfig が AWS CLI に追加され、クラスターにアクセスするための kubeconfig ファイルの作成プロセスが簡単になりました。

### [新しい Amazon EKS 最適化された AMI](#)

Amazon EKS はさまざまなセキュリティ修正と AMI 最適化を提供する Amazon EKS 最適化された AMI (GPU サポートあり/なし) 更新しました。

|                                                                   |                                                                                                                                                                                   |                 |
|-------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----------------|
| <a href="#">Amazon EKS AWS リージョンの拡張</a>                           | Amazon EKS が欧州 (アイルランド) (eu-west-1 ) リージョンで利用可能になりました。                                                                                                                            | 2018 年 9 月 5 日  |
| <a href="#">Amazon EKS プラットフォームのバージョンの更新</a>                      | Kubernetes <a href="#">アグリゲーションレイヤ</a> と <a href="#">水平ポッドオートスケーラー</a> (HPA) をサポートする新しいプラットフォームバージョン。                                                                             | 2018 年 8 月 31 日 |
| <a href="#">新しい Amazon EKS 最適化された AMI と GPU サポート</a>              | Amazon EKS は新しい AWS クラウドフォーメーション ノードテンプレートと <a href="#">ブートストラップスクリプト</a> を使用するため、Amazon EKS 最適化 AMI を更新しました。さらに、新しい <a href="#">GPU サポートを備えた Amazon EKS 最適化 AMI</a> が利用可能になりました。 | 2018 年 8 月 22 日 |
| <a href="#">新しい Amazon EKS 最適化 AMI に ALAS2-2018-1058 用のパッチを適用</a> | Amazon EKS は <a href="#">ALAS2-2018-1058</a> で参照されている CVE に対応するため、Amazon EKS 最適化 AMI を更新しました。                                                                                     | 2018 年 8 月 14 日 |
| <a href="#">Amazon EKS 最適化された AMI のビルドスクリプト</a>                   | Amazon EKS で、Amazon EKS 最適化された AMI のビルドに使用されているビルドスクリプトをオープンソース化されました。これらのビルドスクリプトは、現在 GitHub で入手できます。                                                                            | 2018 年 7 月 10 日 |
| <a href="#">Amazon EKS 初回リリース</a>                                 | サービス開始時の初版ドキュメント                                                                                                                                                                  | 2018 年 5 月 6 日  |

# EKS ユーザーガイドに貢献する

AWS は「EKS ユーザーガイド」の貢献エクスペリエンスを改善しています。

EKS ユーザーガイドのソースを GitHub 上で直接編集できるようになりました。

更新されたエクスペリエンスではマークダウンと同様の強力なオーサリング言語である AsciiDoc が使用されます。AsciiDoc はシンプルな構文と、高度な書式設定、相互参照、セキュリティコントロールなどのエンタープライズドキュメント機能を組み合わせます。

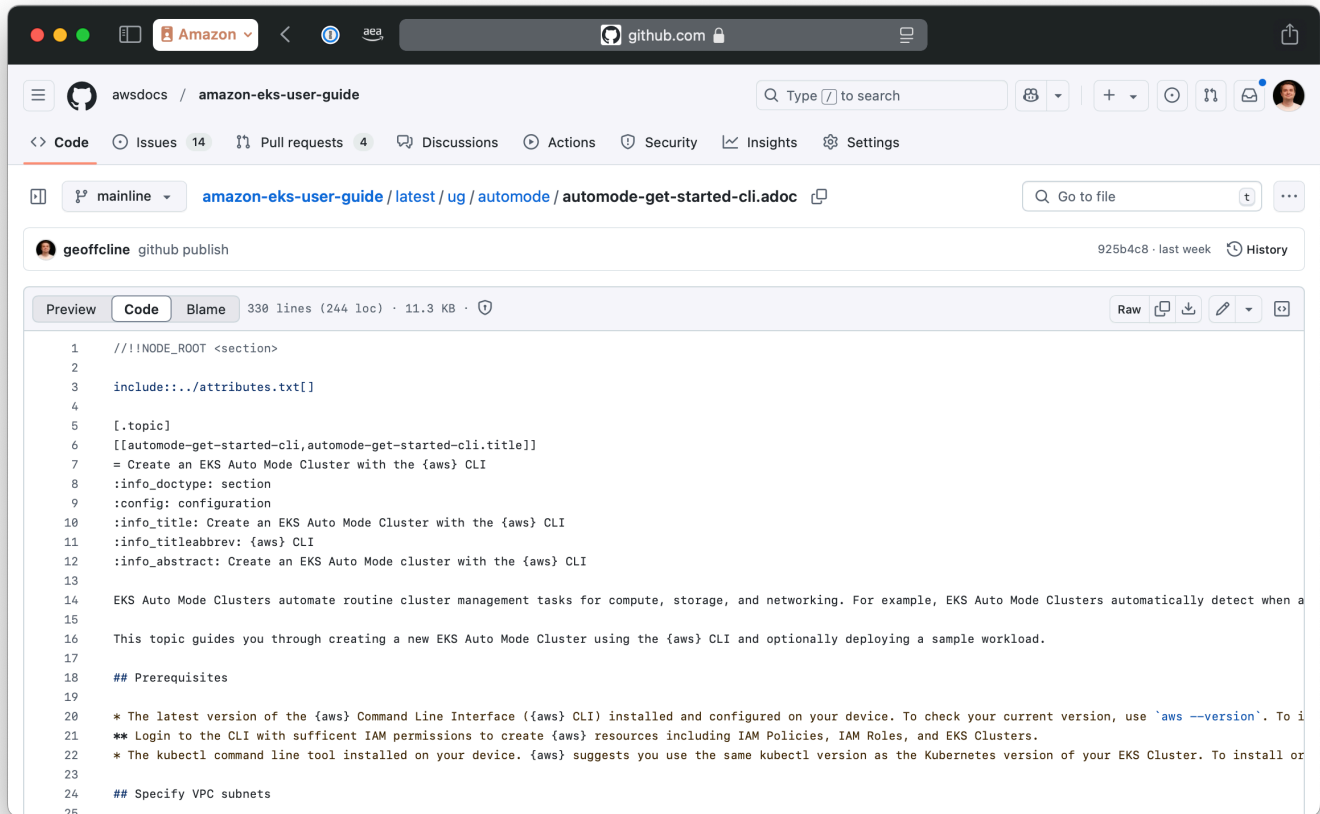
EKS Docs を GitHub 上で直接編集できるようになりました。合理化されたプロセスには次が含まれます:

- プルリクエストの処理の高速化
- 手動ステップの削減
- 自動コンテンツ品質チェック

お客様の貢献をお待ちしています。

## ウェブブラウザから単一ページを編集する

EKS ユーザーガイドの 1 ページはウェブブラウザから直接簡単に編集できます。



ウェブブラウザから複数のページを編集する場合は「」を参照してください[the section called “GitHub でファイルを編集する”](#)。

## 前提条件

- 変更するドキュメント ページが Web ブラウザで開かれていること。
- GitHub にサインインしていること。

## 手順

1. Amazon EKS User Guide で編集したいページに移動します。
2. 右側のペインで、[GitHub でこのページを編集する] を選択します。
3. GitHub で、次のいずれかの方法でエディタを開きます：
  - キーボードの e キーを押します。
  - 鉛筆アイコンをクリックし、ドロップダウンメニューから[インプレース編集] を選択します。

- 編集するオプションがない場合はGitHub にログインする必要があります。GitHub アカウントでは変更を提案するための特別なアクセス許可は必要ありません。ただし、内部 Amazon 寄稿者は GitHub プロファイルをリンクする必要があります。
4. GitHub エディタでコンテンツに必要な変更を加えます。
    - エディタは構文の強調表示とプレビュー機能を提供します。
    - AsciiDoc マークアップを使用して変更をフォーマットすることができます。
    - `ctrl-f` を使用すると、find/replaceインターフェースを開くことができます。
  5. (オプション) 変更内容をプレビューします。
    - `preview` タブを使用して、豊富な書式で変更内容をプレビューします。
    - 変更されたセクションを強調表示するには、`show diff` オプションを使用します。削除されたセクションの左余白には赤色のインジケータがあります。新しいセクションでは左余白に緑色のインジケータが表示されます。
  6. 編集が完了したら、エディターの上部にあるCommit changes...ボタンをクリックします。
  7. コミットダイアログで、次の操作を行います：
    - メールアドレスが正しいことを確認します。
    - 変更点を説明する簡潔でわかりやすいコミットメッセージを追加します。
    - 必要に応じて、より長い説明を追加します。
    - 新しいブランチとプルリクエストを作成することを選択します。

提案された変更を含むプル リクエストを作成しました。

## プルリクエストの概要

PR を作成する場合：

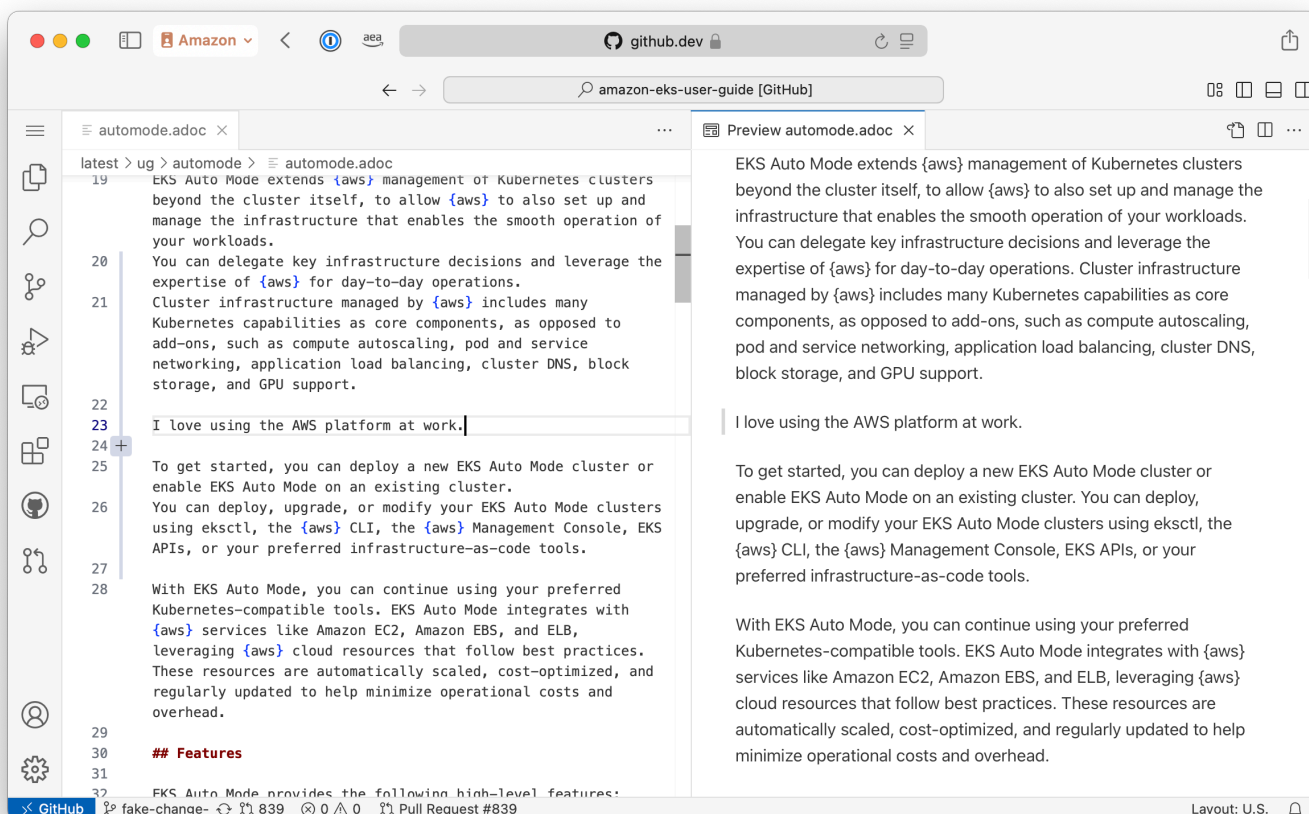
- 変更内容はリポジトリ管理者によるレビューのために送信されます。
- レビュー担当者は変更内容にコメントしたり、修正をリクエストしたりできます。
- 変更を検証するために自動テストが実行される場合があります。
- 承認されると、変更はメインリポジトリにマージされます。

プルリクエストは品質を確保し、変更が統合される前に変更について話し合う方法を提供します。

[GitHub Docs でプル リクエストがどのようにレビューされ、承認されるかを学びます。](#)

# GitHub Web Editorを使用してWebブラウザから複数のファイルを編集する

複数のページへの変更を提案したり、新しいドキュメントページを作成したりする場合はGitHub.dev ウェブエディタを使用します。このウェブエディタは一般的な Visual Studio Code テキストエディタに基づいています。



## 前提条件

- GitHubにログインしました
- Visual Studio Code エディターに精通していること
- Git の知識

## 手順

### Note

EKS Docs チームはテキストラップや AsciiDoc 構文の強調表示など、エディタの推奨設定を含むワークスペースファイルを作成しました。このワークスペースファイルをロードすることをお勧めします。

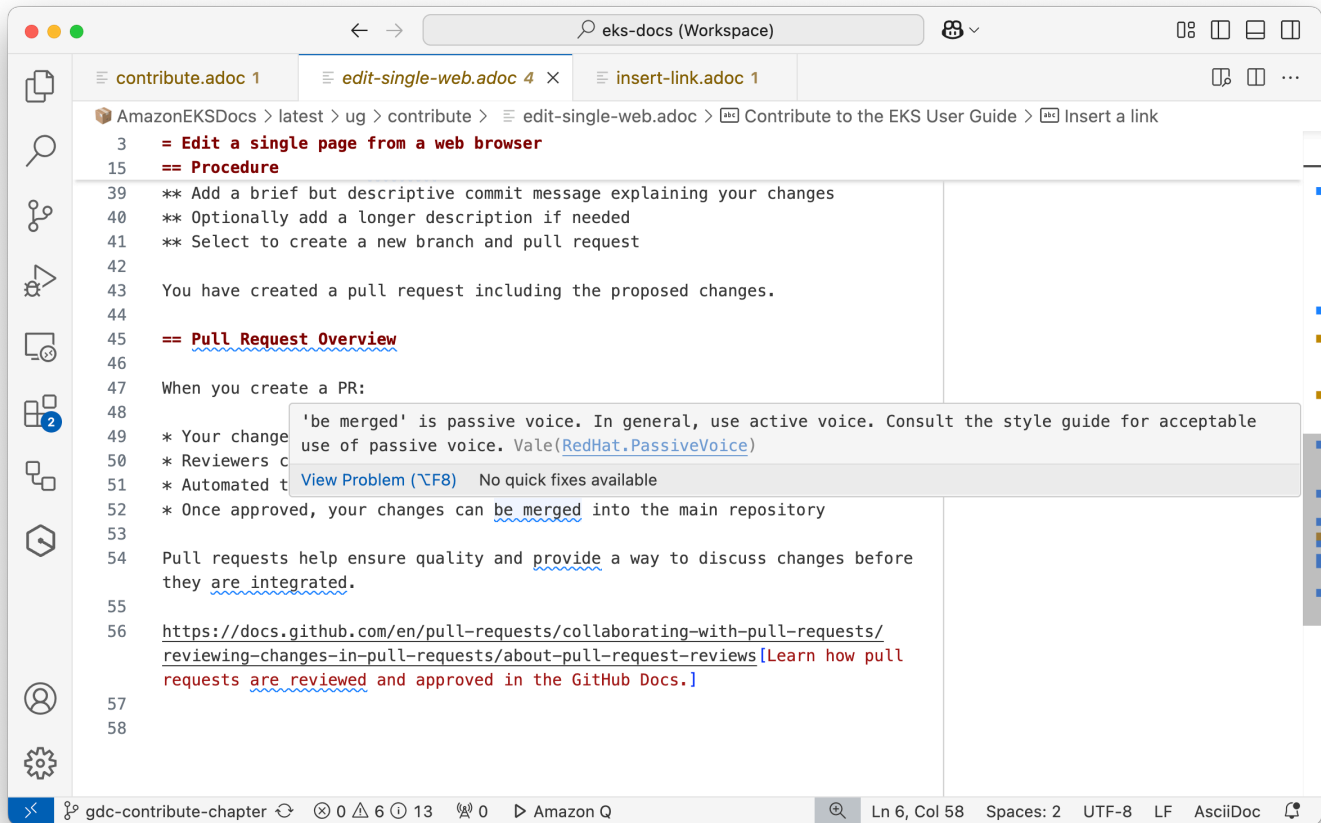
1. GitHub.dev で [ワークスペース](https://github.dev/awsdocs/amazon-eks-user-guide/blob/mainline/eks-docs.code-workspace?workspace=true)を開く
  - URL をブックマークできます <https://github.dev/awsdocs/amazon-eks-user-guide/blob/mainline/eks-docs.code-workspace?workspace=true>
2. (初回セットアップのみ) 自分の GitHub アカウントでリポジトリのフォークを作成するように求められる場合があります。このプロンプトを受け入れます。詳細については「GitHub Docs」の「[Fork a repo](#)」(レポジトリのフォーク) 参照してください。
3. (初回セットアップ時のみ) 右下のプロンプトを受け入れて、AsciiDoc 拡張機能をインストールします。
4. latest/ug のドキュメントコンテンツに移動します。
  - ドキュメントファイルは最上位のセクション別に整理されています。たとえば、「セキュリティ」章のページには「セキュリティ/」ディレクトリの下にソースファイルがあります。
5. ドキュメントページのプレビューを表示するには右上の「サイドへのプレビューを開く」ボタンを使用します。アイコンには小さな拡大鏡が含まれています。
6. 左側のSource Controlタブを使用して、変更をコミットします。詳細についてはVisual Studio Code のドキュメントを参照してください：
  - [Commit changes](#)
  - [プルリクエストの作成](#)

プル リクエストを作成すると、ドキュメント チームによってレビューされます。

## Vale をローカルにインストールして、入力時にスタイルフィードバックを表示する

入力するとスタイルフィードバックが表示されます。これにより、ごこちない書き方やタイプミスを特定するのに役立ちます。





## 概要：

- Vale CLI はスタイル ガイドを読み込み、ソース ファイルに対して実行します。
- EKS Docs リポジトリにはスタイルガイドとローカルルールをロードする Vale 設定ファイルが含まれています。
- Visual Studio (VS) Code の Vale 拡張機能はエディタ内に Vale フィードバックを表示します。

## Vale をインストールする

Vale CLI ドキュメントの指示に従って、[パッケージマネージャーで Vale をインストールします](#)。

## VS Code Vale 拡張機能をインストールする

1. VS Code を開きます。
2. アクティビティバーの拡張機能アイコンをクリックします (または Ctrl+Shift+X キーを押します)。

3. [Vale] を検索します。
4. Chris Chinchilla による「Vale VSCode」拡張機能のインストールをクリックします。
5. プロンプトが表示されたら VS Code を再ロードします。

## Vale を同期する

Vale はプロジェクトルートの `.vale.ini` 設定ファイルを使用して、適用するスタイルルールを決定します。

1. VS Code を開きます。
2. [表示] > [ターミナル]をクリックします (またはCtrl+ を押します)。
3. 必要に応じてプロジェクトのルートディレクトリに移動します。
4. コマンドを実行します。

```
vale sync
```

5. Vale がスタイルルールのダウンロードと同期を完了するのを待ちます。

## VS Code でスタイルフィードバックを表示する

1. VS Code で Markdown または AsciiDoc ファイルを開きます。
2. Vale 拡張機能はスタイルルールと照合してテキストを自動的にチェックします。
3. スタイルの問題はエディターに下線付きで表示されます。
4. 下線付きのテキストにマウスを合わせると、特定のスタイルの提案が表示されます。
5. 提案に従うか、スタイルガイドを参照して問題を修正します。

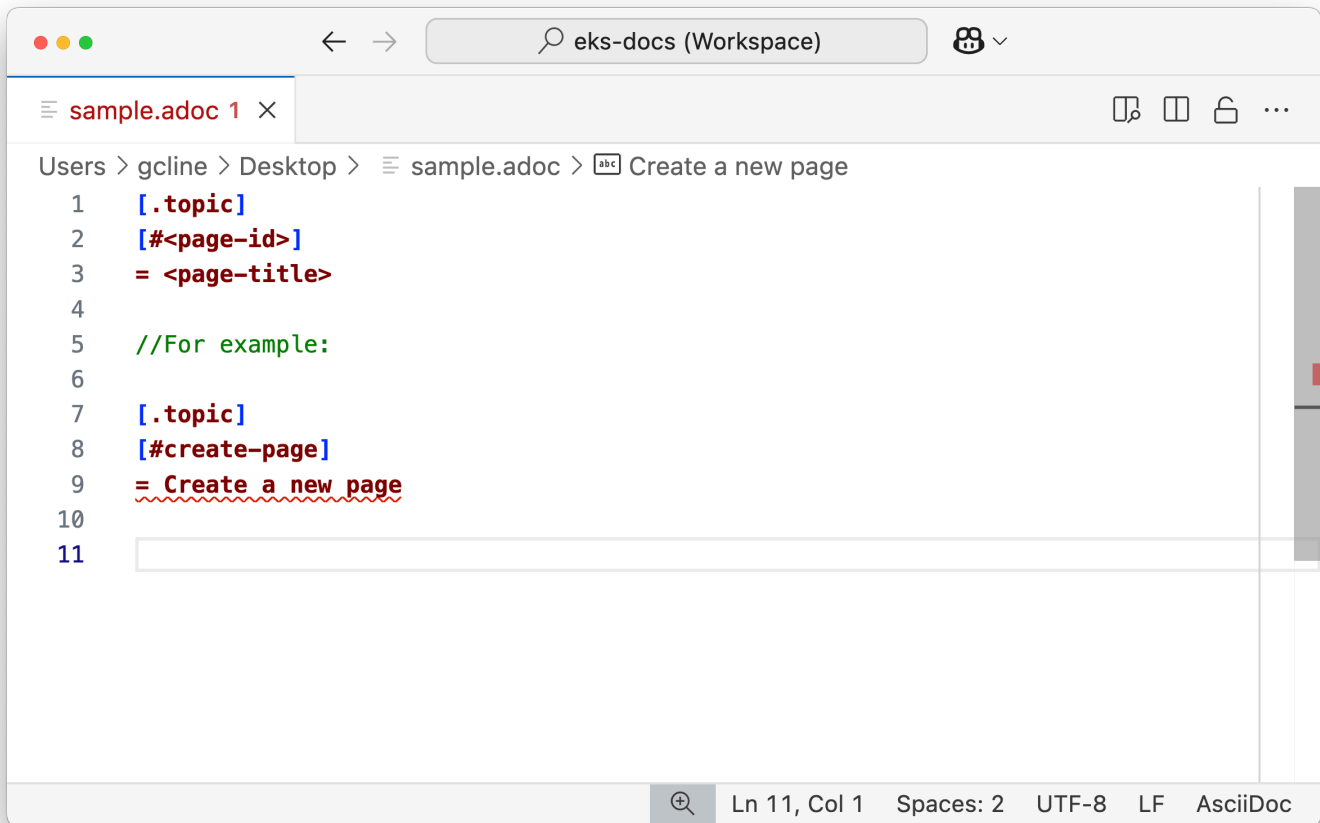
## 新しいページを作成する

新しいドキュメント ページを作成する方法を学びます。このトピックでは最初のページメタデータを作成し、ガイドの目次にページを追加する手順について説明します。

### ページを作成

1. 章ディレクトリに移動します。例えば、「セキュリティ」セクションに新しいページを作成する場合は `latest/ug/security` ディレクトリに移動します。

2. ページ ID を決定します。慣例により、ページ ID はすべて小文字で、- で区切られます。このページの ID は create-page です。
3. ページ ID と adoc 拡張子を持つ新しいファイルを作成します。例えば、create-page.adoc と指定します。
4. このテンプレートを使用してページメタデータを挿入します。



## ナビゲーションにページを追加

1. 親ページに移動します。最上位セクションの親ページは ですbook.adoc。
2. 親ページの下部に、子ページを含めます。

```
include::${filename}[leveloffset=+1]
```

例：

```
include::create-page.adoc[leveloffset=+1]
```

## リンクを挿入する

AsciiDoc は複数の種類のリンクをサポートしています。さまざまな環境でリンクが適切に機能するように、適切なリンク タイプを使用することが重要です。

### EKS User Guideのページまたはセクションへのリンク

「EKS ユーザーガイド」など、同じドキュメントサイト内のページまたはセクション間をリンクするには、クロスリファレンス (xref) を使用します。対象セクションが移動または名前変更されると、自動的に更新されます。

#### カスタムリンクテキストを定義する

```
xref :section-id[${link-text}]
```

#### ページタイトルをリンクテキストとして使用する

山括弧で囲まれたセクション ID を使用します。例: << page-id>>。

### AWS ドキュメント内の別のページへのリンク

1. AWS Docs ページへのリンクを見つける
2. <https://docs.aws.amazon.com/> プレフィックスを削除し、パスのみを残します。パスは a ~ z の文字で始まる必要があります。
3. 以下のようにリンクを作成します

```
link :AmazonS3/latest/userguide/create-bucket-overview.html["Create a bucket", type="documentation"]
```

#### インターネットにリンクする

この形式ではインターネットへの標準リンクが作成されます。これは Amazon 以外のコンテンツまたは GitHub のコンテンツに使用します。

```
link :https://example.com[Visit Example Site]
```

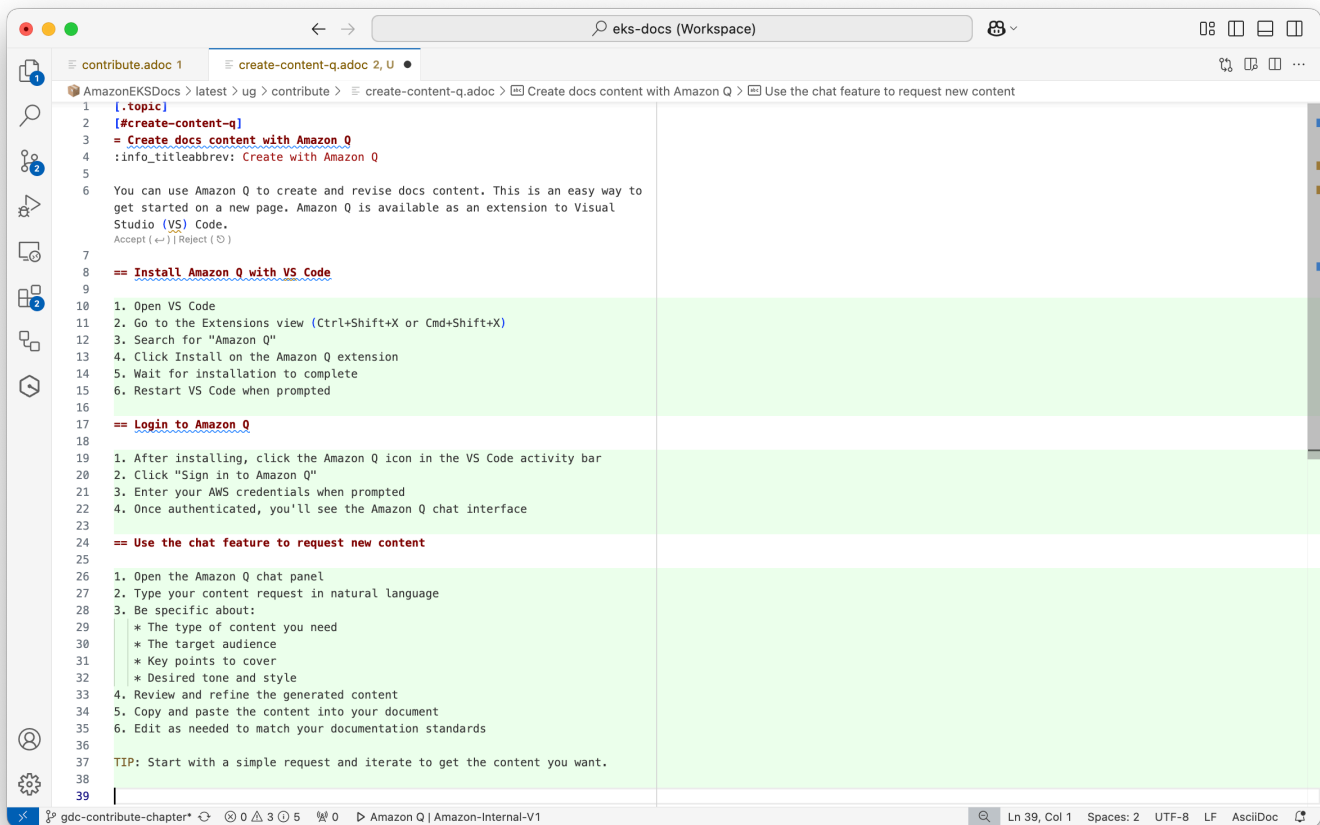
**Note**

外部ドメインを許可リストに登録します。許可リストは `vale/styles/EksDocs/ExternalDomains.yml` にあります

## Amazon Q でドキュメントコンテンツを作成する

Amazon Q を使用して、ドキュメントコンテンツを作成および修正できます。これは新しいページを簡単に開始する方法です。Amazon Q は Visual Studio Code および JetBrains の拡張機能として利用できます。

次の画像では Amazon Q が緑色でマークされた行を生成しました。



## VS Code で Amazon Q をインストールする

### 1. Open VS Code

2. [拡張機能] ビューに移動します (Ctrl+Shift+X または Cmd+Shift+X)
3. 「Amazon Q」を検索します。
4. に Amazon Q IDE 拡張機能をインストールします。
5. インストールが完了するまでお待ちください
6. プロンプトが表示されたらVS Codeを再起動します。

## Amazon Q にログインする

1. インストール後、VS Code アクティビティバーの Amazon Q アイコンを選択します。
2. [Amazon Q にサインイン] を選択します。
3. プロンプトが表示されたら、AWS 認証情報を入力します。
4. 認証されると、Amazon Q チャットインターフェイスが表示されます。

## Amazon Q を使用してコンテンツを作成する

1. VS Code で編集したいファイルを開きます。
2. 修正するテキストまたは新しいコンテンツの場所を選択します。
3. [Ctrl+I] または [Cmd+I] を押します。
4. プロンプトでは以下について具体的に説明します。
  - 必要なコンテンツの種類。
  - 対象者。
  - カバーする重要なポイント。
  - 望ましいトーンとスタイル。
5. 生成されたコンテンツをインラインプレビューで確認します。
6. 変更を承認するには Enter キーを押し、変更を拒否するには Esc キーを押します。
7. 必要に応じてさらに編集します。

## ヒント

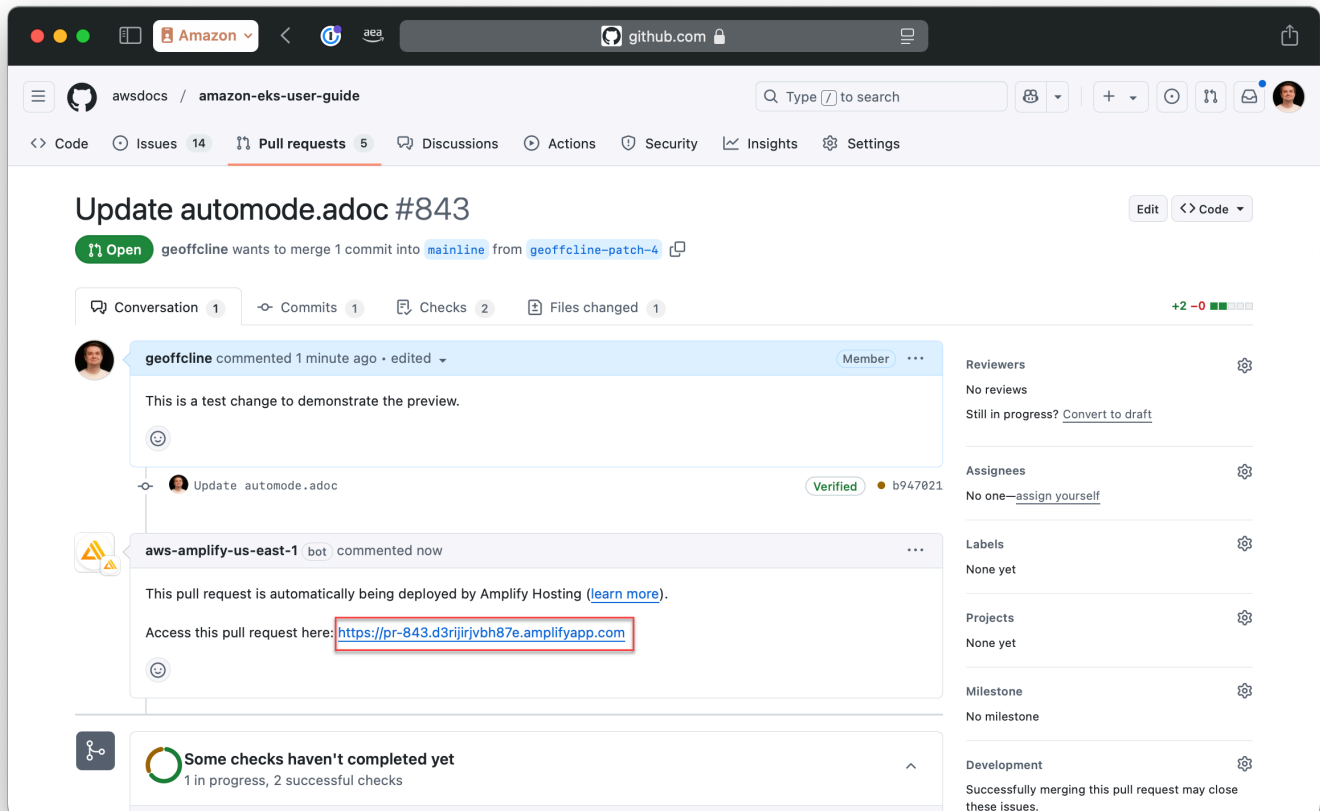
- 簡単なリクエストから始めて、繰り返し実行して必要なコンテンツを取得します。
- ページの見出しの最初のドラフトを作成し、質問して埋めていきます。

- Amazon Q は Markdown を出力する場合があります。これは問題ありません。AsciiDoc ツールではほとんどのマークダウン構文を理解できます。

Amazon Q Developer の詳細については、「IDE での [Amazon Q Developer の使用](#)」を参照してください。

## プルリクエストのコンテンツのプレビューを表示する

Amazon EKS ユーザーガイド GitHub は、ドキュメントサイトのプレビューを構築して生成するように設定されています。このプレビューには完全な AWS テーマは含まれていませんが、コンテンツが適切にビルドされ、リンクが機能することがチェックされます。



このプレビューは、AWS Amplify によって一時的な URL でホストされます。

## プレビューを見る

プルリクエストを送信すると、AWS Amplify はコンテンツのプレビューを構築してデプロイしようとしています。

ビルドが正常に完了すると、aws-amplify-us-east-1 はプレビューへのリンクが張られたコメントをプルリクエストに追加します。「Access this pull request here」(スクリーンショットの赤の枠線で囲まれた部分) の右側にあるリンクを選択します。

ビルドが失敗した場合、リポジトリ管理者はログを確認してフィードバックを提供できます。

### Note

以前に寄稿したことがない場合はプロジェクト保守者がビルドの実行を承認する必要がある場合があります。

## プレビューの制限事項

プレビューは、単一の大きな HTML ファイルとして構築されます。公開すると複数のページとして表示されます。

何が効果的か:

- 相互参照 (xref )
- インターネットへのリンク
- イメージ
- samples/ からホストされているコンテンツ

機能しないもの:

- type="documentation" を使用した、他の AWS コンテンツへのリンク。というのも、プレビュー環境にはこのコンテンツが存在しないためです。
- 属性 {aws} は正しく表示されません。変数 <1> は正しく表示されません。

## AsciiDoc Syntax Reference

このページでは、AsciiDoc 構文の簡単な概要について説明します。



AsciiDoc 構文を使用することをお勧めします。AsciiDoc ツールでは見出しやリストなど、Markdown 構文が部分的にサポートされています。

高度な構文については、[AsciiDoc のウェブサイト](#)を参照してください。

AWS ドキュメントのリンク構文を正しく理解することが重要です。[the section called “リンクを挿入”](#) のやり方を復習します。リンクには Markdown 構文を使用できません。

## 基本的なフォーマット

```
bold text
italic text
`monospace text`
```

## ヘッダー

```
= Document Title (Header 1)
== Header 2
=== Header 3
==== Header 4
===== Header 5
===== Header 6
```

## Lists

順序付けられていないリスト:

```
- Item 1
- Item 2
-- Subitem 2.1
-- Subitem 2.2
- Item 3
```

順序付けられたリスト:

```
. First item
. Second item
.. Subitem 2.1
.. Subitem 2.2
. Third item
```

## リンク

```
External link: https://example.com[Link text]
Internal link: <<page-id>>
Internal link: <<page-id,Link text>>
```

## イメージ

```
image::image-file.jpg[Alt text]
```

## コードブロック

```
[source,python]

def hello_world():
 print("Hello, World!")

```

## テーブル

[基本的なテーブルの構築方法はこちらをご覧ください。](#)

```
[%header,cols="2"]
|===
|Cell in column 1, header row
|Cell in column 2, header row

|Cell in column 1, body row 1
|Cell in column 2, body row 1

|Cell in column 1, body row 2
|Cell in column 2, body row 2

|===
```

## 警告

NOTE: This is a note admonition.

WARNING: This is a warning admonition.

TIP: This is a tip admonition.

IMPORTANT: This is an important admonition.

CAUTION: This is a caution admonition.

プレビュー:

 Note

これはメモの警告です。

## 含まれるもの

```
include::filename.adoc[]
```