Amazon EKS

Eksctl ユーザーガイド

Eksctl ユーザーガイド: Amazon EKS

Table of Contents

Eksctl とは

eksctl は、Amazon Elastic Kubernetes Service (Amazon EKS) クラスターの作成、管理、運用のプロセスを自動化および簡素化するコマンドラインユーティリティツールです。Go で記述された eksctl は、YAML 設定と CLI コマンドを使用して宣言構文を提供し、複雑な EKS クラスターオペレーションを処理します。複雑な EKS クラスターオペレーションでは、異なる AWS サービス間で複数の手動ステップが必要になります。

eksctl は、EKS クラスターを大規模に一貫してデプロイおよび管理する必要がある DevOps エンジニア、プラットフォームチーム、Kubernetes 管理者にとって特に重要です。これは、セルフマネージド Kubernetes から EKS に移行する組織や、Infrastructure as Code (IaC) プラクティスを実装する組織にとって特に便利です。既存の CI/CD パイプラインや自動化ワークフローに統合できるためです。このツールは、VPC 設定、IAM ロールの作成、セキュリティグループ管理など、EKS クラスターのセットアップに必要な AWS サービス間の複雑なやり取りの多くを抽象化します。

eksctl の主な機能には、1 つのコマンドで完全に機能する EKS クラスターを作成する機能、カスタムネットワーク設定のサポート、ノードグループの自動管理、GitOps ワークフロー統合などがあります。このツールは、宣言的なアプローチでクラスターのアップグレードを管理し、ノードグループをスケーリングし、アドオン管理を処理します。また、eksctl は、ネイティブ AWS SDK 統合を通じて他の AWS ツールやサービスとの互換性を維持しながら、Fargate プロファイル設定、マネージド型ノードグループのカスタマイズ、スポットインスタンス統合などの高度な機能も提供します。

機能

現在実装されている機能は次のとおりです。

- クラスターの作成、取得、一覧表示、削除
- ノードグループの作成、ドレイン、削除
- ノードグループのスケール
- クラスターを更新する
- カスタム AMIs
- VPC ネットワーキングを設定する
- API エンドポイントへのアクセスを設定する
- GPU ノードグループのサポート
- スポットインスタンスと混合インスタンス

- IAM 管理ポリシーとアドオンポリシー
- クラスター Cloudformation スタックを一覧表示する
- coredns をインストールする
- クラスターの kubeconfig ファイルの書き込み

Eksctl に関するよくある質問

General

eksctlを使用して、 によって作成されていないクラスターを管理できますかeksctl?

はい。バージョンから0.40.0、 によって作成されたかどうかにかかわらず、任意のクラスターeksctlに対して を実行できますeksctl。詳細については、「the section called “eksctl で作成されていないクラスター”」を参照してください。

ノードグループ

ノードグループのインスタンスタイプを変更するにはどうすればよいですか?

の観点からはeksctl、ノードグループはイミュータブルです。つまり、 を作成したら、ノードグループをスケールアップまたはeksctlスケールダウンするだけです。

インスタンスタイプを変更するには、目的のインスタンスタイプで新しいノードグループを作成し、ワークロードが新しいインスタンスタイプに移動するようにドレインします。このステップが完了したら、古いノードグループを削除できます。

ノードグループの生成されたユーザーデータを表示するにはどうすればよいですか?

まず、ノードグループを管理する Cloudformation スタックの名前が必要です。

```
eksctl utils describe-stacks --region=us-west-2 --cluster NAME
```

のような名前が表示されますeksctl-CLUSTER_NAME-nodegroup-NODEGROUP_NAME。

以下を実行して、ユーザーデータを取得できます。base64 からデコードし、ギップされたデータを解凍する最後の行に注意してください。

```
NG_STACK=eksctl-scrumptious-monster-1595247364-nodegroup-ng-29b8862f # your stack here
LAUNCH_TEMPLATE_ID=$(aws cloudformation describe-stack-resources --stack-name $NG_STACK
 \
```

```
| jq -r '.StackResources | map(select(.LogicalResourceId == "NodeGroupLaunchTemplate")
 \
| .PhysicalResourceId)[0]')
aws ec2 describe-launch-template-versions --launch-template-id $LAUNCH_TEMPLATE_ID \
| jq -r '.LaunchTemplateVersions[0].LaunchTemplateData.UserData' \
| base64 -d | gunzip
```

Ingress

で Ingress をセットアップするにはどうすればよいですか**eksctl**?

AWS Load Balancer Controller を使用することをお勧めします。コントローラーをクラスターにデプロイする方法と、古い ALB Ingress Controller から移行する方法については、こちらを参照してください。

Nginx Ingress Controller の場合、セットアップは他の Kubernetes クラスターの と同じになります。

Kubectl

HTTPS プロキシを使用していて、クラスター証明書の検証が失敗した場合、システム CAs はどのように使用できますか?

環境変数を設定KUBECONFIG_USE_SYSTEM_CAして、システム認証機関kubeconfigを尊重します。

ドライラン

ドライラン機能を使用すると、ノードグループの作成に進む前に、インスタンスセレクタに一致するインスタンスを検査および変更できます。

インスタンスセレクタオプションと を使用して eksctl create clusterが呼び出されると--dry-run、eksctl は CLI オプションを表すノードグループと、インスタンスセレクタリソース条件に一致するインスタンスに設定されたインスタンスタイプを含む ClusterConfig ファイルを出力します。

```
eksctl create cluster --name development --dry-run


apiVersion: eksctl.io/v1alpha5
cloudWatch:
  clusterLogging: {}
```

```
iam:
  vpcResourceControllerPolicy: true
  withOIDC: false
kind: ClusterConfig
managedNodeGroups:
- amiFamily: AmazonLinux2
  desiredCapacity: 2
  disableIMDSv1: true
  disablePodIMDS: false
  iam:
    withAddonPolicies:
      albIngress: false
      appMesh: false
      appMeshPreview: false
      autoScaler: false
      certManager: false
      cloudWatch: false
      ebs: false
      efs: false
      externalDNS: false
      fsx: false
      imageBuilder: false
      xRay: false
  instanceSelector: {}
  instanceType: m5.large
  labels:
    alpha.eksctl.io/cluster-name: development
    alpha.eksctl.io/nodegroup-name: ng-4aba8a47
  maxSize: 2
  minSize: 2
  name: ng-4aba8a47
  privateNetworking: false
  securityGroups:
    withLocal: null
    withShared: null
  ssh:
    allow: false
    enableSsm: false
    publicKeyPath: ""
  tags:
    alpha.eksctl.io/nodegroup-name: ng-4aba8a47
    alpha.eksctl.io/nodegroup-type: managed
  volumeIOPS: 3000
  volumeSize: 80
```

```
  volumeThroughput: 125
  volumeType: gp3
metadata:
  name: development
  region: us-west-2
  version: "1.24"
privateCluster:
  enabled: false
vpc:
  autoAllocateIPv6: false
  cidr: 192.168.0.0/16
  clusterEndpoints:
    privateAccess: false
    publicAccess: true
  manageSharedNodeSecurityGroupRules: true
  nat:
    gateway: Single
```

その後、生成された ClusterConfig を に渡すことができますeksctl create cluster。

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

ClusterConfig ファイルが で渡されると--dry-run、eksctl はファイルに設定された値を含む ClusterConfig ファイルを出力します。

eksctl の 1 回限りのオプション

など、 ClusterConfig ファイルで表現できない特定の 1 回限りのオプションがあります--install-vpc-controllers。

次のことが期待されます。

```
eksctl create cluster --<options...> --dry-run > config.yaml
```

次に、次の操作を行います。

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

は、 なしで最初のコマンドを実行するのと同じです--dry-run。

したがって、eksctl は、 が渡されたときに設定ファイルで表現できない渡すオプションを禁止--dry-runします。

> **Important**
>
> AWS プロファイルを渡す必要がある場合は、CLI `--profile` オプションを渡す代わりに、`AWS_PROFILE`環境変数を設定します。

チュートリアル

このトピックでは、eksctl をインストールして設定し、それを使用して Amazon EKS クラスターを作成する方法について説明します。

ステップ 1: eksctl をインストールする

Linux または macOS デバイスに最新バージョンの eksctl をダウンロードしてインストールするには、次の手順を実行します。

Homebrew で eksctl をインストールするには

1. (前提条件) Homebrew をインストールします。
2. AWS タップを追加します。

```
brew tap aws/tap
```

3. eksctl をインストールする

```
brew install aws/tap/eksctl
```

eksctl を使用する前に、以下の設定ステップを完了します。

1. インストールの前提条件:
   - AWS CLI バージョン 2.x 以降をインストールします。
   - Homebrew を使用して kubectl をインストールします。

     ```
     brew install kubernetes-cli
     ```

2. 環境で AWS 認証情報を設定します。

```
aws configure
```

3. AWS CLI の設定を確認します。

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

ステップ 2: クラスター設定ファイルを作成する

以下の手順を使用して、クラスター設定ファイルを作成します。

1. という名前の新しいファイルを作成しますcluster.yaml。

```
touch cluster.yaml
```

2. 次の基本的なクラスター設定を追加します。

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

metadata:
   name: basic-cluster
   region: us-west-2

nodeGroups:
   - name: ng-1
     instanceType: m5.large
     desiredCapacity: 2
     minSize: 1
     maxSize: 3
     ssh:
        allow: false
```

3. 設定をカスタマイズします。
   - 必要な AWS リージョンregionに合わせて を更新します。
   - ワークロードの要件instanceTypeに基づいて を変更します。
   - 必要に応じて、desiredCapacity、minSize、および maxSizeを調整します。
4. 設定ファイルを検証します。

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

ステップ 3: クラスターを作成する

EKS クラスターを作成するには、次の手順に従います。

1. 設定ファイルを使用してクラスターを作成します。

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

2. クラスターの作成を待ちます (通常は 15～20 分かかります) 。
3. クラスターの作成を確認します。

```
eksctl get cluster
```

4. 新しいクラスターを使用するように kubectl を設定します。

```
aws eks update-kubeconfig --name basic-cluster --region us-west-2
```

5. クラスターの接続を確認します。

```
kubectl get nodes
```

これで、クラスターを使用する準備ができました。

オプション: クラスターを削除する

不要な料金が発生しないように、完了したらクラスターを削除してください。

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

> **Note**
>
> クラスターの作成には AWS の料金が発生する場合があります。クラスターを作成する前に、Amazon EKS の料金を確認してください。

次のステップ

- クラスターに接続するように Kubectl を設定する
- サンプルアプリケーションをデプロイする

Eksctl のインストールオプション

eksctl は、以下に説明するように、公式リリースからインストールできます。公式の GitHub リリースeksctlからのみインストールすることをお勧めします。サードパーティーのインストーラを使用することを選択できますが、AWS はこれらのインストール方法を維持またはサポートしていないことに注意してください。独自の判断で使用してください。

前提条件

AWS API 認証情報を設定する必要があります。AWS CLI やその他のツール (kops、Terraform など) で機能するもので十分です。~/.aws/credentials ファイル変数または環境変数を使用できます。詳細については、AWS CLI リファレンスを参照してください。

また、 には AWS IAM Authenticator for Kubernetes コマンド (aws-iam-authenticatorまたは aws eks get-token (AWS CLI のバージョン 1.16.156 以降で利用可能) も必要ですPATH。

EKS クラスターの作成に使用される IAM アカウントには、これらの最小限のアクセスレベルが必要です。

AWS サービス	アクセスレベル
CloudFormation	フルアクセス
EC2	Full: Tagging Limited: List、Read、Write
EC2 オートスケーリング	制限: 一覧表示、書き込み
EKS	フルアクセス
IAM	制限: リスト、読み取り、書き込み、アクセス許可管理
Systems Manager	制限: List、Read

Unix の場合

最新リリースをダウンロードするには、以下を実行します。

```
# for ARM systems, set ARCH to: `arm64`, `armv6` or `armv7`
ARCH=amd64
PLATFORM=$(uname -s)_$ARCH

curl -sLO "https://github.com/eksctl-io/eksctl/releases/latest/download/eksctl_
$PLATFORM.tar.gz"

# (Optional) Verify checksum
curl -sL "https://github.com/eksctl-io/eksctl/releases/latest/download/
eksctl_checksums.txt" | grep $PLATFORM | sha256sum --check

tar -xzf eksctl_$PLATFORM.tar.gz -C /tmp && rm eksctl_$PLATFORM.tar.gz

sudo install -m 0755 /tmp/eksctl /usr/local/bin && rm /tmp/eksctl
```

Windows の場合

直接ダウンロード (最新リリース):

- AMD64/x86_64
- ARMv6
- ARMv7
- ARM64

アーカイブを PATH変数のフォルダに解凍してください。

必要に応じて、チェックサムを確認します。

1. チェックサムファイルのダウンロード: 最新
2. コマンドプロンプトを使用して、 CertUtilの出力とダウンロードしたチェックサムファイルを手動で比較します。

   ```
   REM Replace amd64 with armv6, armv7 or arm64
   CertUtil -hashfile eksctl_Windows_amd64.zip SHA256
   ```

3. PowerShell を使用して検証を自動化するには、 -eq演算子を使用して Trueまたは Falseの結果を取得します。

   ```
   # Replace amd64 with armv6, armv7 or arm64
   ```

```
 (Get-FileHash -Algorithm SHA256 .\eksctl_Windows_amd64.zip).Hash -eq ((Get-Content .
\eksctl_checksums.txt) -match 'eksctl_Windows_amd64.zip' -split ' ')[0]
```

Git Bash の使用:

```
# for ARM systems, set ARCH to: `arm64`, `armv6` or `armv7`
ARCH=amd64
PLATFORM=windows_$ARCH

curl -sLO "https://github.com/eksctl-io/eksctl/releases/latest/download/eksctl_
$PLATFORM.zip"

# (Optional) Verify checksum
curl -sL "https://github.com/eksctl-io/eksctl/releases/latest/download/
eksctl_checksums.txt" | grep $PLATFORM | sha256sum --check

unzip eksctl_$PLATFORM.zip -d $HOME/bin

rm eksctl_$PLATFORM.zip
```

実行可能ファイルは$HOME/bin、Git Bash eksctl $PATHから にある に配置されます。

Homebrew

Homebrew を使用して、MacOS および Linux にソフトウェアをインストールできます。

AWS は、eksctl を含む Homebrew タップを維持します。

Homebrew タップの詳細については、Github の プロジェクトと eksctl の Homebrew 式を参照してください。

Homebrew で eksctl をインストールするには

1. (前提条件) Homebrew をインストールする
2. AWS タップを追加する

```
brew tap aws/tap
```

3. eksctl をインストールする

```
brew install aws/tap/eksctl
```

Docker

リリースと RC ごとに、コンテナイメージが ECR リポジトリ にプッシュされますpublic.ecr.aws/eksctl/eksctl。ECR Public Gallery - eksctl の使用状況の詳細をご覧ください。例えば、

```
docker run --rm -it public.ecr.aws/eksctl/eksctl version
```

シェルの完了

Bash

bash 補完を有効にするには、以下を実行するか、 ~/.bashrc または に配置します~/.profile。

```
. <(eksctl completion bash)
```

Zsh

zsh を完了するには、以下を実行してください。

```
mkdir -p ~/.zsh/completion/
eksctl completion zsh > ~/.zsh/completion/_eksctl
```

および を に配置します~/.zshrc。

```
fpath=($fpath ~/.zsh/completion)
```

oh-my-zsh のようなディストリビューションを実行していない場合は、まず自動補完を有効にする必要があります (永続化~/.zshrcするために を挿入します) 。

```
autoload -U compinit
compinit
```

魚

次のコマンドは、魚の自動補完に使用できます。

```
mkdir -p ~/.config/fish/completions
eksctl completion fish > ~/.config/fish/completions/eksctl.fish
```

PowerShell

以下のコマンドは、セットアップのために参照できます。パスは、システム設定によって異なる場合があります。

```
eksctl completion powershell > C:\Users\Documents\WindowsPowerShell\Scripts\eksctl.ps1
```

更新

> **Important**
>
> eksctl を直接ダウンロードしてインストールする場合 (パッケージマネージャーを使用しない) 、手動で更新する必要があります。

クラスター

この章では、eksctl を使用した EKS クラスターの作成と設定について説明します。また、アドオンと EKS Auto Mode も含まれています。

トピック:

- the section called “EKS アクセスエントリ”
  - aws-auth ConfigMap を EKS アクセスエントリに置き換えて Kubernetes RBAC 管理を簡素化する
  - 既存の IAM ID マッピングを aws-auth ConfigMap からアクセスエントリに移行する
  - クラスター認証モードを設定し、クラスター作成者の管理者アクセス許可を制御する
- the section called “デフォルトのアドオンの更新”
  - 古いクラスターでデフォルトの EKS アドオンを更新してクラスターを安全に保つ
- the section called “アドオン”
  - アドオンのインストール、更新、削除のルーチンタスクを自動化します。
  - Amazon EKS アドオンには、AWS アドオン、オープンソースコミュニティアドオン、マーケットプレイスアドオンが含まれます。
- the section called “EKS 自動モード”
  - AWS が EKS インフラストラクチャを管理できるようにすることで、運用オーバーヘッドを削減する
  - デフォルトの汎用プールとシステムプールの代わりにカスタムノードプールを設定する
  - Auto Mode を使用するように既存の EKS クラスターを変換する
- the section called “CloudWatch ロギング”
  - 特定の EKS コントロールプレーンコンポーネントのログを有効にしてクラスターの問題をトラブルシューティングする
  - EKS クラスターログのログ保持期間を設定する
  - eksctl コマンドを使用して既存のクラスターログ記録設定を変更する
- the section called “クラスターのアップグレード”
  - EKS コントロールプレーンバージョンを安全にアップグレードしてセキュリティと安定性を維持する

- 古いグループを新しいグループに置き換えて、ノードグループ間でアップグレードをロールアウトする
- デフォルトのクラスターアドオンを更新する

- the section called “クラスターの作成と管理”
  - デフォルトのマネージド型ノードグループを使用して基本的な EKS クラスターをすばやく開始する
  - 特定の設定で設定ファイルを使用してカスタマイズされたクラスターを作成する
  - プライベートネットワークとカスタム IAM ポリシーを使用して既存の VPCs にクラスターをデプロイする
- the section called “kubelet を設定する”
  - kubelet とシステムデーモンの予約を設定してノードリソース不足を防ぐ
  - メモリとファイルシステムの可用性のエビクションしきい値をカスタマイズする
  - ノードグループ間で特定の kubelet 機能ゲートを有効または無効にする
- the section called “EKS コネクタ”
  - EKS コンソールを使用してハイブリッド Kubernetes デプロイの管理を一元化する
  - 外部クラスターアクセスの IAM ロールとアクセス許可を設定する
  - 外部クラスターを削除し、関連する AWS リソースをクリーンアップする
- the section called “EKS 完全プライベートクラスター”
  - アウトバウンドインターネットアクセスを持たない完全プライベート EKS クラスターでセキュリティ要件を満たす
  - VPC エンドポイントを介して AWS サービスへのプライベートアクセスを設定する
  - 明示的なネットワーク設定によるプライベートノードグループの作成と管理
- the section called “Karpenter サポート”
  - ノードのプロビジョニングを自動化する
  - カスタム Karpenter プロビジョナー設定を作成する
  - スポットインスタンスの中断処理を使用して Karpenter をセットアップする
- the section called “Amazon EMR”
  - EMR クラスターと EKS クラスター間の IAM ID マッピングを作成する
- the section called “EKS Fargate のサポート”
  - ポッドスケジューリングのカスタム Fargate プロファイルを定義する
  - 作成と設定の更新による Fargate プロファイルの管理

- the section called “eksctl で作成されていないクラスター”
  - eksctl の外部で作成されたクラスターの管理を標準化する
  - 既存の eksctl 以外のクラスターで eksctl コマンドを使用する
- the section called “ゾーンシフトを有効にする”
  - ラピッドゾーンフェイルオーバー機能を有効にしてアプリケーションの可用性を向上させる
  - 新しい EKS クラスターデプロイでゾーンシフトを設定する
  - 既存の EKS クラスターでゾーンシフト機能を有効にする

クラスターの作成と管理

このトピックでは、Eksctl を使用して EKS クラスターを作成および削除する方法について説明します。クラスターを作成するには、CLI コマンドを使用するか、クラスター設定 YAML ファイルを作成します。

シンプルなクラスターの作成

次のコマンドを使用して、シンプルなクラスターを作成します。

```
eksctl create cluster
```

これにより、2 つの m5.large ノードを含む 1 つのマネージド型ノードグループを使用して、デフォルトのリージョン (AWS CLI 設定で指定) に EKS クラスターが作成されます。

eksctl は、設定ファイルが使用されていないときに、デフォルトでマネージド型ノードグループを作成するようになりました。セルフマネージド型ノードグループを作成するには、 `eksctl create cluster`または `--managed=false`に渡します`eksctl create nodegroup`。

考慮事項

- でクラスターを作成するときに`us-east-1`、 が発生することがありますUnsupportedAavailabilityZoneException。この場合、提案されたゾーンをコピーし、 `--zones`フラグを渡します。例: `eksctl create cluster --region=us-east-1 --zones=us-east-1a,us-east-1b,us-east-1d`。この問題は他のリージョンで発生する可能性がありますが、あまり一般的ではありません。ほとんどの場合、 `--zone`フラグを使用する必要はありません。

設定ファイルを使用してクラスターを作成する

フラグの代わりに設定ファイルを使用してクラスターを作成できます。

まず、 cluster.yaml ファイルを作成します。

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

metadata:
  name: basic-cluster
  region: eu-north-1

nodeGroups:
  - name: ng-1
    instanceType: m5.large
    desiredCapacity: 10
    volumeSize: 80
    ssh:
      allow: true # will use ~/.ssh/id_rsa.pub as the default ssh key
  - name: ng-2
    instanceType: m5.xlarge
    desiredCapacity: 2
    volumeSize: 100
    ssh:
      publicKeyPath: ~/.ssh/ec2_id_rsa.pub
```

次に、次のコマンドを実行します。

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

これにより、説明に従ってクラスターが作成されます。

既存の VPC を使用する必要がある場合は、次のような設定ファイルを使用できます。

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

metadata:
  name: cluster-in-existing-vpc
  region: eu-north-1
```

```
vpc:
  subnets:
    private:
      eu-north-1a: { id: subnet-0ff156e0c4a6d300c }
      eu-north-1b: { id: subnet-0549cdab573695c03 }
      eu-north-1c: { id: subnet-0426fb4a607393184 }

nodeGroups:
  - name: ng-1-workers
    labels: { role: workers }
    instanceType: m5.xlarge
    desiredCapacity: 10
    privateNetworking: true
  - name: ng-2-builders
    labels: { role: builders }
    instanceType: m5.2xlarge
    desiredCapacity: 2
    privateNetworking: true
    iam:
      withAddonPolicies:
        imageBuilder: true
```

クラスター名またはノードグループ名には、英数字 (大文字と小文字を区別) とハイフンのみを含める必要があります。アルファベット文字で始まり、128 文字を超えることはできません。そうしないと、検証エラーが発生します。詳細については、AWS CloudFormation コンソールからスタックを作成する」を参照してください。 CLoudFormation

新しいクラスターの kubeconfig を更新する

クラスターが作成されると、適切な kubernetes 設定が kubeconfig ファイルに追加されます。これは、環境変数または`KUBECONFIG`~/.kube/configデフォルトで設定したファイルです。kubeconfig ファイルへのパスは、 `--kubeconfig`フラグを使用して上書きできます。

kubeconfig ファイルの書き込み方法を変更できるその他のフラグ:

フラグ	型	ファイル	デフォルト値
--kubeconfig	string	kubeconfig を書き込むパス (--auto-kubeconfig と互換性がありません)	$KUBECONFIG または ~/.kube/config

フラグ	型	ファイル	デフォルト値
--set-kubeconfig-context	ブール	true の場合、現在のコンテキストは kubeconfig で設定されます。コンテキストがすでに設定されている場合、上書きされます。	true
--auto-kubeconfig	ブール	kubeconfig ファイルをクラスター名で保存する	true
--write-kubeconfig	ブール	kubeconfig の書き込みを切り替える	true

クラスターの削除

このクラスターを削除するには、以下を実行します。

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

> **Warning**
>
> 削除エラーが適切に報告されるようにするには、削除オペレーションで `--wait`フラグを使用します。

`--wait` フラグがない場合、eksctl はクラスターの CloudFormation スタックにのみ削除オペレーションを発行し、削除を待機しません。場合によっては、クラスターまたはその VPC を使用する AWS リソースによって、クラスターの削除が失敗することがあります。削除が失敗した場合、または待機フラグを忘れた場合は、CloudFormation GUI に移動し、そこから eks スタックを削除する必要があります。

> **Warning**
>
> PDB ポリシーは、クラスターの削除中にノードの削除をブロックすることがあります。

ノードグループを使用してクラスターを削除すると、Pod Disruption Budget (PDB) ポリシーによってノードが正常に削除されない可能性があります。たとえば、 `aws-ebs-csi-driver`がインストールされているクラスターには通常、PDB ポリシーを持つ 2 つのポッドがあり、一度に 1 つのポッドしか使用できないため、削除中に他のポッドを削除できなくなります。これらのシナリオでクラスターを正常に削除するには、 `disable-nodegroup-eviction`フラグを使用して PDB ポリシーチェックをバイパスします。

```
eksctl delete cluster -f cluster.yaml --disable-nodegroup-eviction
```

その他のサンプル設定ファイルについては、eksctl GitHub リポジトリの `examples/` ディレクトリを参照してください。

ドライラン

ドライラン機能を使用すると、クラスターの作成をスキップする ClusterConfig ファイルを生成し、提供された CLI オプションを表し、eksctl によって設定されたデフォルト値を含む ClusterConfig ファイルを出力できます。

詳細については、「ドライラン」ページを参照してください。

EKS 自動モード

eksctl 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する機能である EKS Auto Mode をサポートしています。これにより、AWS は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もセットアップおよび管理できます。これにより、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day-to-day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ク、アプリケーションの負荷分散、クラスター DNS、ブロックストレージ、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 機能が含まれています。

自動モードを有効にした EKS クラスターの作成

eksctl は、自動モードを有効にして設定するための新しいautoModeConfigフィールドを追加しました。autoModeConfig フィールドの形状は です。

```
autoModeConfig:
    # defaults to false
    enabled: boolean
    # optional, defaults to [general-purpose, system].
    # 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
```

autoModeConfig.enabled が true の場合、eksctl は computeConfig.enabled: true、、および を EKS API storageConfig.blockStorage.enabled: trueに渡して EKS クラスターを作成し、コンピューティングkubernetesNetworkConfig.elasticLoadBalancing.enabled: true、ストレージ、ネットワークなどのデータプレーンコンポーネントを管理できるようにします。

自動モードを有効にして EKS クラスターを作成するには、autoModeConfig.enabled: true「」のように を設定します。

```
# auto-mode-cluster.yaml
apiVersion: eksctl.io/v1alpha5
kind: ClusterConfig
metadata:
    name: auto-mode-cluster
    region: us-west-2

autoModeConfig:
    enabled: true
```

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

eksctl は、Auto Mode によって起動されたノードに使用するノードロールを作成します。また、eksctl は general-purposeおよび systemノードプールを作成します。デフォルトのノードプールの作成を無効にするには、たとえば、異なるサブネットセットを使用する独自のノードプールを設定するにはnodePools: []、「」のように を設定します。

```
apiVersion: eksctl.io/v1alpha5
kind: ClusterConfig
metadata:
    name: auto-mode-cluster
    region: us-west-2

autoModeConfig:
    enabled: true
    nodePools: [] # disables creation of default node pools.
```

自動モードを使用するように EKS クラスターを更新する

Auto Mode を使用するように既存の EKS クラスターを更新するには、 を実行します。

```
# cluster.yaml
apiVersion: eksctl.io/v1alpha5
kind: ClusterConfig
metadata:
    name: cluster
    region: us-west-2

autoModeConfig:
    enabled: true
```

```
eksctl update auto-mode-config -f cluster.yaml
```

> **Note**
>
> クラスターが eksctl によって作成され、パブリックサブネットをクラスターサブネットとして使用している場合、Auto Mode はパブリックサブネットでノードを起動します。Auto Mode によって起動されたワーカーノードにプライベートサブネットを使用するには、プライベートサブネットを使用するようにクラスターを更新します。

自動モードの無効化

自動モードを無効にするには、 を設定`autoModeConfig.enabled: false`して実行します。

```
# cluster.yaml
```

```
apiVersion: eksctl.io/v1alpha5
kind: ClusterConfig
metadata:
    name: auto-mode-cluster
    region: us-west-2

autoModeConfig:
    enabled: false
```

```
eksctl update auto-mode-config -f cluster.yaml
```

詳細情報

- EKS Auto Mode

EKS アクセスエントリ

eksctl を使用して EKS アクセスエントリを管理できます。アクセスエントリを使用して、AWS IAM ID に Kubernetes アクセス許可を付与します。たとえば、クラスター内の Kubernetes リソースを読み取るアクセス許可を開発者ロールに付与できます。

このトピックでは、eksctl を使用してアクセスエントリを管理する方法について説明します。アクセスエントリの一般的な情報については、「EKS アクセスエントリを使用して Kubernetes へのアクセス権を IAM ユーザーに付与する」を参照してください。

AWS で定義された Kubernetes アクセスポリシーをアタッチすることも、IAM アイデンティティを Kubernetes グループと関連付けることもできます。

使用可能な事前定義ポリシーの詳細については、「アクセスポリシーをアクセスエントリに関連付ける」を参照してください。

顧客の Kubernetes ポリシーを定義する必要がある場合は、IAM Identity を Kubernetes グループに関連付け、そのグループにアクセス許可を付与します。

クラスター認証モード

アクセスエントリは、クラスターの認証モードで許可されている場合にのみ使用できます。

詳細については、「クラスター認証モードの設定」を参照してください。

YAML ファイルを使用して認証モードを設定する

eksctl は ClusterConfig に新しいaccessConfig.authenticationModeフィールドを追加しました。これは、次の 3 つの値のいずれかに設定できます。

- CONFIG_MAP - EKS API のデフォルト - aws-auth ConfigMap のみが使用されます
- API - アクセスエントリ API のみが使用されます
- API_AND_CONFIG_MAP - default in eksctl - aws-auth ConfigMap とアクセスエントリ API の両方を使用できます

ClusterConfig YAML で認証モードを設定します。

```
accessConfig:
  authenticationMode: <>
```

コマンドを使用して認証モードを更新する

既存の eksctl 以外の作成済みクラスターでアクセスエントリを使用する場合は、 CONFIG_MAPオプションを使用するクラスターで、ユーザーはまず authenticationMode を に設定する必要がありますAPI_AND_CONFIG_MAP。そのために、 eksctl はクラスター認証モードを更新するための新しいコマンドを導入しました。これは、 などの CLI フラグの両方で機能します。

```
eksctl utils update-authentication-mode --cluster my-cluster --authentication-mode
 API_AND_CONFIG_MAP
```

アクセスエントリリソース

アクセスエントリには、 STANDARDや などのタイプがありますEC2_LINUX。タイプは、アクセスエントリの使用方法によって異なります。

- standard タイプは、IAM ユーザーと IAM ロールに Kubernetes アクセス許可を付与するためのものです。
  - たとえば、コンソールへのアクセスに使用するロールまたはユーザーにアクセスポリシーをアタッチすることで、AWS コンソールで Kubernetes リソースを表示できます。
- EC2_LINUX および EC2_WINDOWSタイプは、EC2 インスタンスに Kubernetes アクセス許可を付与するためのものです。インスタンスは、これらのアクセス許可を使用してクラスターに参加します。

アクセスエントリのタイプの詳細については、「アクセスエントリの作成」を参照してください。

IAM エンティティ

アクセスエントリを使用して、IAM ユーザーや IAM ロールなどの IAM ID に Kubernetes アクセス許可を付与できます。

`accessConfig.accessEntries` フィールドを使用して、IAM リソースの ARN を Access Entries EKS API に関連付けます。例えば、次のようになります。

```
accessConfig:
  authenticationMode: API_AND_CONFIG_MAP
  accessEntries:
    - principalARN: arn:aws:iam::111122223333:user/my-user-name
      type: STANDARD
      kubernetesGroups: # optional Kubernetes groups
        - group1 # groups can used to give permissions via RBAC
        - group2

    - principalARN: arn:aws:iam::111122223333:role/role-name-1
      accessPolicies: # optional access polices
        - policyARN: arn:aws:eks::aws:cluster-access-policy/AmazonEKSViewPolicy
          accessScope:
            type: namespace
            namespaces:
              - default
              - my-namespace
              - dev-*

    - principalARN: arn:aws:iam::111122223333:role/admin-role
      accessPolicies: # optional access polices
        - policyARN: arn:aws:eks::aws:cluster-access-policy/AmazonEKSClusterAdminPolicy
          accessScope:
            type: cluster

    - principalARN: arn:aws:iam::111122223333:role/role-name-2
      type: EC2_LINUX
```

EKS ポリシーの関連付けに加えて、IAM エンティティが属する Kubernetes グループを指定して、RBAC 経由でアクセス許可を付与することもできます。

マネージド型ノードグループと Fargate

これらのリソースのアクセスエントリとの統合は、EKS API によってバックグラウンドで実現されます。新しく作成されたマネージド型ノードグループと Fargate ポッドは、プリロードされた RBAC リソースを使用するのではなく、API アクセスエントリを作成します。既存のノードグループと Fargate ポッドは変更されず、aws-auth 設定マップのエントリに引き続き依存します。

セルフマネージド型ノードグループ

各アクセスエントリにはタイプがあります。セルフマネージド型ノードグループを承認するために、 `eksctl`は、プリンシパル ARN をノードロール ARN に設定し、タイプをノードグループの amiFamily `EC2_WINDOWS`に応じて `EC2_LINUX`または に設定して、各ノードグループに一意のアクセスエントリを作成します。

独自のアクセスエントリを作成するときは、 `EC2_LINUX` (Linux または Bottlerocket セルフマネージドノードで使用される IAM ロールの場合) 、 `EC2_WINDOWS` (Windows セルフマネージドノードで使用される IAM ロールの場合) 、 `FARGATE_LINUX` (AWS Fargate (Fargate) で使用される IAM ロールの場合) 、またはタイプ`STANDARD`として を指定することもできます。タイプを指定しない場合、デフォルトのタイプは に設定されます`STANDARD`。

> **Note**
>
> 既存の で作成されたノードグループを削除する場合`instanceRoleARN`、ノードグループがそれ以上関連付けられていない場合、対応するアクセスエントリを削除するのはユーザーの責任です。これは、eksctl は複雑なプロセスであるため、eksctl が作成した以外のセルフマネージド型ノードグループでアクセスエントリがまだ使用されているかどうかを検出しようとしないためです。

アクセスエントリを作成する

これは、クラスターの作成時に、設定ファイルの一部として目的のアクセスエントリを指定して実行するという 2 つの異なる方法で実行できます。

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

以下を実行して、クラスターの作成後に または を実行します。

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

アクセスエントリを作成するための設定ファイルの例については、eksctl GitHub リポジトリの「40-access-entries.yaml」を参照してください。

アクセスエントリの取得

ユーザーは、次のいずれかを実行して、特定のクラスターに関連付けられているすべてのアクセスエントリを取得できます。

```
eksctl get accessentry -f config.yaml
```

OR

```
eksctl get accessentry --cluster my-cluster
```

または、特定の IAM エンティティに対応するアクセスエントリのみを取得するには、 `--principal-arn`フラグを使用します。例:

```
eksctl get accessentry --cluster my-cluster --principal-arn
 arn:aws:iam::111122223333:user/admin
```

アクセスエントリを削除する

一度に 1 つのアクセスエントリを削除するには、以下を使用します。

```
eksctl delete accessentry --cluster my-cluster --principal-arn
 arn:aws:iam::111122223333:user/admin
```

複数のアクセスエントリを削除するには、 `--config-file`フラグを使用して、アクセスエントリ`principalARN's`に対応するすべての を最上位`accessEntry`フィールドで指定します。例:

```
...
accessEntry:
  - principalARN: arn:aws:iam::111122223333:user/my-user-name
  - principalARN: arn:aws:iam::111122223333:role/role-name-1
  - principalARN: arn:aws:iam::111122223333:role/admin-role
```

```
eksctl delete accessentry -f config.yaml
```

aws-auth ConfigMap から移行する

ユーザーは、以下を実行して、既存の IAM ID を aws-auth configmap からアクセスエントリに移行できます。

```
eksctl utils migrate-to-access-entry --cluster my-cluster --target-authentication-mode
 <API or API_AND_CONFIG_MAP>
```

--target-authentication-mode フラグが に設定されている場合API、認証モードは API モードに切り替えられ (既に API モードになっている場合はスキップされます) 、IAM ID マッピングはアクセスエントリに移行され、aws-auth設定マップはクラスターから削除されます。

--target-authentication-mode フラグが に設定されている場合API_AND_CONFIG_MAP、認証モードは API_AND_CONFIG_MAP モード (既に API_AND_CONFIG_MAP モードの場合はスキップ) aws-auth に切り替わり、IAM ID マッピングはアクセスエントリに移行されますが、設定マップは保持されます。

> **Note**
>
> --target-authentication-mode フラグが に設定されている場合API、configmap aws-auth に次のいずれかの制約がある場合、このコマンドは認証モードを API モードに更新しません。

- アカウントレベルの ID マッピングがあります。
- 1 つ以上のロール/ユーザーは、プレフィックス (、 system:bootstrapperssystem:nodesなどの EKS 固有のグループsystem:を除く) で始まる kubernetes system:mastersグループにマッピングされます。
- 1 つ以上の IAM ID マッピング (複数可) が [サービスにリンクされたロール](link:IAM/latest/UserGuide/using-service-linked-roles.html) 用です。

クラスター作成者管理者のアクセス許可を無効にする

eksctl はaccessConfig.bootstrapClusterCreatorAdminPermissions: boolean、false に設定すると、クラスターを作成する IAM ID へのクラスター管理者アクセス許可の付与を無効にする新しいフィールドを追加しました。つまり、

設定ファイルに オプションを追加します。

```
accessConfig:
  bootstrapClusterCreatorAdminPermissions: false
```

と を実行します。

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

eksctl で作成されていないクラスター

によって作成されていないクラスターに対してeksctlコマンドを実行できますeksctl。

> **Note**
>
> Eksctl は、AWS CloudFormation と互換性のある名前を持つ非所有クラスターのみをサポートできます。これと一致しないクラスター名は、CloudFormation API 検証チェックに失敗します。

サポートされているコマンド

以下のコマンドは、 以外の方法で作成されたクラスターに対して使用できますeksctl。コマンド、フラグ、設定ファイルオプションは、まったく同じ方法で使用できます。

一部の機能を見逃した場合は、お知らせください。

- ✓ 作成:
  - eksctl create nodegroup ✓ (以下の注意を参照)
  - ✓ eksctl create fargateprofile
  - ✓ eksctl create iamserviceaccount
  - ✓ eksctl create iamidentitymapping
- ✓ 取得:
  - ✓ eksctl get clusters/cluster
  - ✓ eksctl get fargateprofile
  - ✓ eksctl get nodegroup

- ✓ eksctl get labels
- ✓ 削除:
  - ✓ eksctl delete cluster
  - ✓ eksctl delete nodegroup
  - ✓ eksctl delete fargateprofile
  - ✓ eksctl delete iamserviceaccount
  - ✓ eksctl delete iamidentitymapping
- ✓ アップグレード:
  - ✓ eksctl upgrade cluster
  - ✓ eksctl upgrade nodegroup
- ✓ 設定/設定解除:
  - ✓ eksctl set labels
  - ✓ eksctl unset labels
- ✓ スケール:
  - ✓ eksctl scale nodegroup
- ✓ ドレイン:
  - ✓ eksctl drain nodegroup
- ✓ 有効にする:
  - ✓ eksctl enable profile
  - ✓ eksctl enable repo
- ✓ ユーティリティ:
  - ✓ eksctl utils associate-iam-oidc-provider
  - ✓ eksctl utils describe-stacks
  - ✓ eksctl utils install-vpc-controllers
  - ✓ eksctl utils nodegroup-health
  - ✓ eksctl utils set-public-access-cidrs
  - ✓ eksctl utils update-cluster-endpoints
  - ✓ eksctl utils update-cluster-logging
  - ✓ eksctl utils write-kubeconfig
  - ✓ eksctl utils update-coredns

- ✓ eksctl utils update-aws-node
- ✓ eksctl utils update-kube-proxy

ノードグループの作成

eksctl create nodegroup は、ユーザーからの特定の入力を必要とする唯一のコマンドです。

ユーザーは任意のネットワーク設定でクラスターを作成できるため、時間を考慮して、 eksctlはこれらの値の取得や推測を試みません。これは、eksctl で作成されていないクラスターでユーザーがこのコマンドをどのように使用しているかを知るにつれて、将来変更される可能性があります。

つまり、 によって作成されていないクラスターにノードグループまたはマネージド型ノードグループを作成するにはeksctl、VPC の詳細を含む設定ファイルを指定する必要があります。少なくとも:

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

metadata:
  name: non-eksctl-created-cluster
  region: us-west-2

vpc:
  id: "vpc-12345"
  securityGroup: "sg-12345"    # this is the ControlPlaneSecurityGroup
  subnets:
    private:
      private1:
          id: "subnet-12345"
      private2:
          id: "subnet-67890"
    public:
      public1:
          id: "subnet-12345"
      public2:
          id: "subnet-67890"

...
```

VPC 設定オプションの詳細については、「ネットワーク」を参照してください。

EKS Connector での非 EKS クラスターの登録

EKS Connector を使用して、AWS の外部にあるクラスターを EKS コンソールで表示できます。このプロセスでは、クラスターを EKS に登録し、外部 Kubernetes クラスターで EKS Connector エージェントを実行する必要があります。

eksctl は、必要な AWS リソースを作成し、EKS Connector の Kubernetes マニフェストを生成して外部クラスターに適用することで、EKS 以外のクラスターの登録を簡素化します。

クラスターを登録する

EKS 以外の Kubernetes クラスターを登録または接続するには、 を実行します。

```
eksctl register cluster --name <name> --provider <provider>
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
```

このコマンドは、クラスターを登録し、登録の有効期限が切れる前に外部クラスターに適用する必要がある EKS Connector の Kubernetes マニフェストを含む 3 つのファイルを書き込みます。

> **Note**
>
> eks-connector-clusterrole.yaml および は、すべての名前空間の Kubernetes リソースの getおよび アクセスlist許可を呼び出し元の IAM ID eks-connector-console-dashboard-full-access-clusterrole.yamlに付与します。クラスターに適用する前に、必要に応じて編集する必要があります。より制限されたアクセスを設定するには、「クラスターを表示するためのアクセスをユーザーに付与する」を参照してください。

EKS Connector に使用する既存の IAM ロールを提供するには、次のように 経由で渡`--role-arn`します。

```
eksctl register cluster --name <name> --provider <provider> --role-arn=<role-arn>
```

クラスターが既に存在する場合、eksctl はエラーを返します。

クラスターの登録解除

登録済みクラスターを登録解除または切断するには、 を実行します。

```
eksctl deregister cluster --name <name>
2021-08-19 16:04:09 [#]  unregistered cluster "<name>" successfully
2021-08-19 16:04:09 [#]  run `kubectl delete namespace eks-connector` and `kubectl
 delete -f eks-connector-binding.yaml` on your cluster to remove EKS Connector
 resources
```

このコマンドは外部クラスターの登録を解除し、関連する AWS リソースを削除しますが、クラスターから EKS コネクタ Kubernetes リソースを削除する必要があります。

詳細情報

- EKS コネクタ

kubelet 設定のカスタマイズ

システムリソースは、kubelet の設定を通じて予約できます。リソース不足の場合、kubelet はポッドを削除できず、最終的にノードが になる可能性があるため、これは推奨されます`NotReady`。これを行うために、設定ファイルには、 に埋め込まれる無料のフォーム yaml を受け入れる`kubeletExtraConfig`フィールドを含めることができます`kubelet.yaml`。

の一部のフィールド`kubelet.yaml`は eksctl によって設定されるため`address`、、、、、、などclusterDomainauthenticationauthorization、上書きできませんserverTLSBootstrap。

次の設定ファイルの例では、kubelet `300m` の vCPU、`300Mi`メモリ、`1Gi`エフェメラルストレージの vCPU、OS システムデーモン`1Gi`のメモリ、エフェメラルストレージ`300Mi`の `300m`vCPU を予約するノードグループを作成し、使用可能な`200Mi`メモリが 未満であるか、ルートファイルシステムの 10% 未満である場合にポッドの削除を開始します。

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

metadata:
  name: dev-cluster-1
  region: eu-north-1

nodeGroups:
  - name: ng-1
    instanceType: m5a.xlarge
    desiredCapacity: 1
    kubeletExtraConfig:
        kubeReserved:
            cpu: "300m"
            memory: "300Mi"
            ephemeral-storage: "1Gi"
        kubeReservedCgroup: "/kube-reserved"
        systemReserved:
            cpu: "300m"
            memory: "300Mi"
            ephemeral-storage: "1Gi"
        evictionHard:
            memory.available:  "200Mi"
            nodefs.available: "10%"
        featureGates:
            RotateKubeletServerCertificate: true # has to be enabled, otherwise it will
 be disabled
```

この例では、4 つの vCPUs と 16GiB のメモリm5a.xlargeを持つ タイプのインスタンスの場合、CPUs のAllocatable量は 3.4 と 15.4 GiB のメモリになります。のフィールドの設定ファイルで指定された値はkubeletExtraconfig、eksctl で指定されたデフォルト値を完全に上書きすることに注意してください。ただし、1 つ以上のkubeReservedパラメータを省略すると、欠落しているパラメータは、使用されている aws インスタンスタイプに基づいてデフォルトで正常な値になります。

kubeReserved 計算

通常、同じ CPU および RAM 設定のインスタンスを使用するように混合インスタンス NodeGroup を設定することをお勧めしますが、これは厳密な要件ではありません。したがって、kubeReserved計算では InstanceDistribution.InstanceTypesフィールドで最小のインスタンスが使用されます。これにより、異なるインスタンスタイプの NodeGroups は、最小のイン

スタンスであまり多くのリソースを予約しません。ただし、これにより、最大インスタンスタイプに対して小さすぎる予約が発生する可能性があります。

> **Warning**
>
> デフォルトでは eksctlが設定されます がfeatureGates.RotateKubeletServerCertificate=true、カスタム featureGatesが指定されている場合、設定は解除されます。無効にする必要がない限りfeatureGates.RotateKubeletServerCertificate=true、常に を含める必要があります。

CloudWatch ロギング

このトピックでは、EKS クラスターのコントロールプレーンコンポーネントの Amazon CloudWatch ログ記録を設定する方法について説明します。CloudWatch ログ記録は、クラスターのコントロールプレーンオペレーションを可視化します。これは、問題のトラブルシューティング、クラスターアクティビティの監査、Kubernetes コンポーネントのヘルスのモニタリングに不可欠です。

CloudWatch ログ記録の有効化

EKS コントロールプレーンの CloudWatch ログ記録は、データの取り込みとストレージのコストのため、デフォルトでは有効になっていません。

クラスターの作成時にコントロールプレーンのログ記録を有効にするには、 で **cloudWatch.clusterLogging.enableTypes**設定を定義する必要があります `ClusterConfig` (例については、以下を参照してください) 。

したがって、正しい**cloudWatch.clusterLogging.enableTypes**設定の 設定ファイルがある場合は、 を使用してクラスターを作成できますeksctl create cluster --config-file=<path>。

クラスターを既に作成している場合は、 を使用できますeksctl utils update-cluster-logging。

> **Note**
>
> このコマンドはデフォルトでプランモードで実行されます。クラスターに変更を適用するには、 --approveフラグを指定する必要があります。

設定ファイルを使用している場合は、以下を実行します。

```
eksctl utils update-cluster-logging --config-file=<path>
```

または、CLI フラグを使用することもできます。

すべてのタイプのログを有効にするには、以下を実行します。

```
eksctl utils update-cluster-logging --enable-types all
```

audit ログを有効にするには、以下を実行します。

```
eksctl utils update-cluster-logging --enable-types audit
```

ログを除くすべてのcontrollerManagerログを有効にするには、以下を実行します。

```
eksctl utils update-cluster-logging --enable-types=all --disable-
types=controllerManager
```

api ログタイプと schedulerログタイプがすでに有効になっている場合
は、controllerManager同時に を有効schedulerまたは無効にするには、以下を実行します。

```
eksctl utils update-cluster-logging --enable-types=controllerManager --disable-
types=scheduler
```

これにより、 apiと のみが有効なログタイプcontrollerManagerとして残ります。

すべてのタイプのログを無効にするには、以下を実行します。

```
eksctl utils update-cluster-logging --disable-types all
```

ClusterConfig の例

EKS クラスターでは、 の enableTypesフィールドで可能な値のリストを取得して、コントロールプレーンコンポーネントのさまざまなタイプのログを有効にclusterLoggingできます。

取り得る値には以下のものがあります。

- `api`: Kubernetes API サーバーログを有効にします。
- `audit`: Kubernetes 監査ログを有効にします。
- `authenticator`: 認証ログを有効にします。
- `controllerManager`: Kubernetes コントローラーマネージャーログを有効にします。
- `scheduler`: Kubernetes スケジューラログを有効にします。

詳細については、「EKS ドキュメント」を参照してください。

すべてのログを無効にする

すべてのタイプを無効にするには、 `cloudWatch`セクションを完全に使用`[]`または削除します。

すべてのログを有効にする

`"*"` または を使用して、すべてのタイプを有効にできます`"all"`。例えば、次のようになります。

```
cloudWatch:
  clusterLogging:
    enableTypes: ["*"]
```

1 つ以上のログを有効にする

有効にするタイプを一覧表示することで、タイプのサブセットを有効にできます。例えば、次のようになります。

```
cloudWatch:
  clusterLogging:
    enableTypes:
      - "audit"
      - "authenticator"
```

ログの保持期間

デフォルトでは、ログは CloudWatch Logs に無期限に保存されます。CloudWatch Logs でコントロールプレーンログを保持する日数を指定できます。次の例では、ログを 7 日間保持します。

```
cloudWatch:
  clusterLogging:
```

```
    logRetentionInDays: 7
```

完全な例

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

metadata:
  name: cluster-11
  region: eu-west-2

nodeGroups:
  - name: ng-1
    instanceType: m5.large
    desiredCapacity: 1

cloudWatch:
  clusterLogging:
    enableTypes: ["audit", "authenticator"]
    logRetentionInDays: 7
```

EKS 完全プライベートクラスター

eksctl は、アウトバウンドインターネットアクセスがなく、プライベートサブネットのみを持つ完全プライベートクラスターの作成をサポートしています。VPC エンドポイントは、AWS サービスへのプライベートアクセスを有効にするために使用されます。

このガイドでは、アウトバウンドインターネットアクセスなしでプライベートクラスターを作成する方法について説明します。

完全プライベートクラスターの作成

完全プライベートクラスターを作成するために必要なフィールドは のみで すprivateCluster.enabled。

```
privateCluster:
  enabled: true
```

クラスターの作成後、Kubernetes API サーバーへのアクセスを必要とする eksctl コマンドは、クラスターの VPC 内、ピア接続された VPC、または AWS Direct Connect などの他の方法を使用し

て実行する必要があります。EKS APIs へのアクセスを必要とする eksctl コマンドは、クラスターの VPC 内から実行されている場合、機能しません。これを修正するには、Amazon EKS のインターフェイスエンドポイントを作成して、Amazon Virtual Private Cloud (VPC) から Amazon Elastic Kubernetes Service (Amazon EKS) 管理 APIs にプライベートにアクセスします。今後のリリースでは、eksctl はこのエンドポイントを作成するためのサポートを追加するため、手動で作成する必要はありません。OpenID Connect プロバイダー URL へのアクセスを必要とするコマンドは、Amazon EKS の AWS PrivateLink を有効にすると、クラスターの VPC の外部から実行する必要があります。

マネージド型ノードグループの作成は引き続き機能し、クラスターの VPC 内、ピア接続された VPC、または AWS Direct Connect などの他の方法を使用してコマンドを実行する場合、EKS インターフェイスエンドポイント経由で API サーバーにアクセスする必要があるため、セルフマネージド型ノードグループの作成は機能します。

> **Note**
>
> VPC エンドポイントは、使用量に基づいて時間単位で課金されます。料金の詳細については、AWS PrivateLink の料金を参照してください。

> **Warning**
>
> 完全プライベートクラスターは、 ではサポートされていませんeu-south-1。

追加の AWS サービスへのプライベートアクセスの設定

ワーカーノードが AWS のサービスにプライベートにアクセスできるようにするには、eksctl は次のサービスの VPC エンドポイントを作成します。

- コンテナイメージをプルするための ECR (`ecr.api`と の両方`ecr.dkr`) のインターフェイスエンドポイント (AWS CNI プラグインなど)
- S3 が実際のイメージレイヤーをプルするためのゲートウェイエンドポイント
- `aws-cloud-provider` 統合に必要な EC2 のインターフェイスエンドポイント
- サービスアカウントの Fargate ロールと IAM ロール (IRSA) をサポートする STS のインターフェイスエンドポイント
- CloudWatch ログ記録が有効になっている場合の CloudWatch ログ記録 (`logs`) のインターフェイスエンドポイント

これらの VPC エンドポイントは機能するプライベートクラスターに不可欠であるため、eksctl はそれらの設定または無効化をサポートしていません。ただし、クラスターには他の AWS のサービスへのプライベートアクセスが必要になる場合があります (Cluster Autoscaler に必要な Autoscaling など) 。これらのサービスは で指定できます。これによりprivateCluster.additionalEndpointServices、eksctl は各サービスに VPC エンドポイントを作成するように指示されます。

たとえば、Autoscaling と CloudWatch ログ記録へのプライベートアクセスを許可するには:

```
privateCluster:
  enabled: true
  additionalEndpointServices:
  # For Cluster Autoscaler
  - "autoscaling"
  # CloudWatch logging
  - "logs"
```

でサポートされているエンドポイントadditionalEndpointServicesは、autoscaling、、cloudformationおよび ですlogs。

エンドポイント作成のスキップ

必要な AWS エンドポイントがセットアップされ、EKS ドキュメントで説明されているサブネットにリンクされた VPC がすでに作成されている場合、 eksctlはskipEndpointCreation次のようなオプションを指定することで作成をスキップできます。

```
privateCluster:
  enabled: true
  skipEndpointCreation: true
```

この設定は、 と一緒に使用することはできませんadditionalEndpointServices。エンドポイントの作成はすべてスキップされます。また、この設定は、エンドポイント<#サブネットトポロジが正しく設定されている場合にのみ推奨されます。サブネット ID が正しい場合、vpceルーティングはプレフィックスアドレスで設定され、必要なすべての EKS エンドポイントが作成され、提供された VPC にリンクされます。 eksctl はこれらのリソースを変更しません。

ノードグループ

クラスターの VPC はパブリックサブネットなしで作成されるため、完全プライベートクラスターではプライベートノードグループ (マネージド型とセルフマネージド型の両方) のみがサポー

トされます。privateNetworking フィールド (nodeGroup**[].privateNetworking** と managedNodeGroup[) は明示的に設定する必要があります。フルプライベートクラスターで未privateNetworking設定のままにすることはエラーです。

```
nodeGroups:
- name: ng1
  instanceType: m5.large
  desiredCapacity: 2
  # privateNetworking must be explicitly set for a fully-private cluster
  # Rather than defaulting this field to `true`,
  # we require users to explicitly set it to make the behaviour
  # explicit and avoid confusion.
  privateNetworking: true

managedNodeGroups:
- name: m1
  instanceType: m5.large
  desiredCapacity: 2
  privateNetworking: true
```

クラスターエンドポイントのアクセス

完全プライベートクラスターは、クラスター作成clusterEndpointAccess時の変更をサポートしていません。完全プライベートクラスターはプライベートアクセスのみを持つことができclusterEndpoints.privateAccess、これらのフィールドの変更を許可するとクラスターが壊れる可能性があるため、 clusterEndpoints.publicAccessまたは のいずれかを設定するのはエラーです。

ユーザー提供の VPC とサブネット

eksctl は、既存の VPC とサブネットを使用した完全プライベートクラスターの作成をサポートしています。プライベートサブネットのみを指定でき、 でサブネットを指定するのはエラーですvpc.subnets.public。

eksctl は、指定された VPC に VPC エンドポイントを作成し、指定されたサブネットのルートテーブルを変更します。eksctl はメインルートテーブルを変更しないため、各サブネットには明示的なルートテーブルが関連付けられている必要があります。

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

```
metadata:
  name: private-cluster
  region: us-west-2

privateCluster:
  enabled: true
  additionalEndpointServices:
  - "autoscaling"

vpc:
  subnets:
    private:
      us-west-2b:
        id: subnet-0818beec303f8419b
      us-west-2c:
        id: subnet-0d42ef09490805e2a
      us-west-2d:
        id: subnet-0da7418077077c5f9


nodeGroups:
- name: ng1
  instanceType: m5.large
  desiredCapacity: 2
  # privateNetworking must be explicitly set for a fully-private cluster
  # Rather than defaulting this field to true for a fully-private cluster, we require
 users to explicitly set it
  # to make the behaviour explicit and avoid confusion.
  privateNetworking: true

managedNodeGroups:
- name: m1
  instanceType: m5.large
  desiredCapacity: 2
  privateNetworking: true
```

完全プライベートクラスターの管理

クラスターの作成後にすべてのコマンドを使用するには、eksctl に EKS API サーバーエンドポイントへのプライベートアクセスとアウトバウンドインターネットアクセス (の場合) が必要ですEKS:DescribeCluster。API サーバーへのアクセスを必要としないコマンドは、eksctl にアウトバウンドインターネットアクセスがある場合にサポートされます。

完全プライベートクラスターを強制削除する

eksctl はクラスターのすべてのリソースに自動的にアクセスできないため、eksctl を介して完全プライベートクラスターを削除するとエラーが発生する可能性があります。 はこれを解決するために`--force`存在します。クラスターは強制的に削除され、エラーが発生したときに続行されます。

制限事項

現在の実装の制限は、eksctl が最初にパブリックエンドポイントアクセスとプライベートエンドポイントアクセスの両方を有効にしてクラスターを作成し、すべてのオペレーションが完了した後にパブリックエンドポイントアクセスを無効にすることです。これは、eksctl が自己管理型ノードをクラスターに参加させ、GitOps と Fargate をサポートするために、Kubernetes API サーバーにアクセスする必要があるために必要です。これらのオペレーションが完了すると、eksctl はクラスターエンドポイントへのアクセスをプライベートのみに切り替えます。この追加更新は、完全プライベートクラスターの作成に標準クラスターよりも時間がかかることを意味します。将来、eksctl は VPC 対応の Lambda 関数に切り替えて、これらの API オペレーションを実行する場合があります。

HTTP プロキシサーバーを介したアウトバウンドアクセス

eksctl は、設定された HTTP(S) プロキシサーバーを介して AWS APIs と通信できますが、プロキシ除外リストを正しく設定する必要があります。

通常、値 を含む適切な`no_proxy`環境変数を設定することで、クラスターの VPC エンドポイントに対するリクエストがプロキシ経由でルーティングされないようにする必要があります`.eks.amazonaws.com`。

プロキシサーバーが「SSL 傍受」を実行し、サービスアカウント (IRSA) の IAM ロールを使用している場合は、ドメイン の SSL Man-in-the-Middle を明示的にバイパスする必要がありますoidc.<region>.amazonaws.com。そうしないと、eksctl が OIDC プロバイダーの誤ったルート証明書サムプリントを取得し、IAM 認証情報を取得できないために AWS VPC CNI プラグインが起動できず、クラスターが動作しなくなります。

詳細情報

- EKS プライベートクラスター

アドオン

このトピックでは、eksctl を使用して Amazon EKS クラスターの Amazon EKS アドオンを管理する方法について説明します。EKS アドオンは、EKS API を使用して Kubernetes 運用ソフトウェアを有効化および管理できる機能で、クラスターアドオンのインストール、設定、更新のプロセスを簡素化します。

> **Warning**
>
> eksctl は、セルフマネージドアドオンではなく EKS アドオンとしてデフォルトのアドオン (vpc-cni、coredns、kube-proxy) をインストールするようになりました。つまり、eksctl v0.184.0 以降で作成されたクラスターでは、`eksctl utils update-*` コマンド`eksctl update addon`の代わりに を使用する必要があります。

Cilium や Calico などの代替 CNI プラグインを使用する場合は、デフォルトのネットワークアドオンなしでクラスターを作成できます。

EKS アドオンは、EKS Pod Identity Associations を介した IAM アクセス許可の受信をサポートし、クラスター外の AWS サービスに接続できるようになりました。

アドオンの作成

Eksctl は、クラスターアドオンをより柔軟に管理できます。

設定ファイルで、必要なアドオンと、それらにアタッチするロールまたはポリシー (必要な場合) を指定できます。

```
apiVersion: eksctl.io/v1alpha5
kind: ClusterConfig
metadata:
  name: example-cluster
  region: us-west-2

iam:
  withOIDC: true

addons:
- name: vpc-cni
  # all below properties are optional
```

```
  version: 1.7.5
  tags:
    team: eks
  # you can specify at most one of:
  attachPolicyARNs:
  - arn:aws:iam::account:policy/AmazonEKS_CNI_Policy
  # or
  serviceAccountRoleARN: arn:aws:iam::account:role/AmazonEKSCNIAccess
  # or
  attachPolicy:
    Statement:
    - Effect: Allow
      Action:
      - ec2:AssignPrivateIpAddresses
      - ec2:AttachNetworkInterface
      - ec2:CreateNetworkInterface
      - ec2:DeleteNetworkInterface
      - ec2:DescribeInstances
      - ec2:DescribeTags
      - ec2:DescribeNetworkInterfaces
      - ec2:DescribeInstanceTypes
      - ec2:DetachNetworkInterface
      - ec2:ModifyNetworkInterfaceAttribute
      - ec2:UnassignPrivateIpAddresses
      Resource: '*'
```

attachPolicy、、 attachPolicyARNs のうち最大 1 つを指定できますserviceAccountRoleARN。

これらが指定されていない場合、アドオンはすべての推奨ポリシーがアタッチされたロールで作成されます。

> **Note**
>
> アドオンにポリシーをアタッチするには、クラスターで OIDCが有効になっている必要があります。有効になっていない場合、アタッチされたポリシーは無視されます。

その後、クラスター作成プロセス中にこれらのアドオンを作成できます。

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

または、設定ファイルまたは CLI フラグを使用して、クラスターの作成後に明示的にアドオンを作成します。

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

```
eksctl create addon --name vpc-cni --version 1.7.5 --service-account-role-arn <role-arn>
```

```
eksctl create addon --name aws-ebs-csi-driver --namespace-config 'namespace=custom-namespace'
```

> **Tip**
>
> --namespace-config フラグを使用して、デフォルトの名前空間ではなくカスタム名前空間にアドオンをデプロイします。

アドオンの作成中に、アドオンのセルフマネージドバージョンがクラスターに既に存在する場合は、設定ファイルを介して resolveConflictsオプションを設定することで、潜在的なconfigMap競合を解決する方法を選択できます。例:

```
addons:
- name: vpc-cni
  attachPolicyARNs:
    - arn:aws:iam::aws:policy/AmazonEKS_CNI_Policy
  resolveConflicts: overwrite
```

アドオン作成の場合、 resolveConflictsフィールドは 3 つの異なる値をサポートします。

- none - EKS は値を変更しません。作成が失敗する可能性があります。
- overwrite - EKS は、設定の変更を EKS のデフォルト値に上書きします。
- preserve - EKS は値を変更しません。作成が失敗する可能性があります。(に似ていますがnone、preserveアドオンの更新とは異なります)。

有効なアドオンの一覧表示

クラスターで有効になっているアドオンを確認するには、以下を実行します。

```
eksctl get addons --cluster <cluster-name>
```

または

```
eksctl get addons -f config.yaml
```

アドオンのバージョンの設定

アドオンのバージョンの設定はオプションです。version フィールドを空のままにするとeksctl、アドオンのデフォルトバージョンが解決されます。特定のアドオンのデフォルトバージョンであるバージョンの詳細については、EKS に関する AWS ドキュメントを参照してください。デフォルトバージョンは、必ずしも利用可能な最新バージョンではない場合があります。

アドオンバージョンは に設定できますlatest。または、 バージョンは、 v1.7.5-eksbuild.1やなどの EKS ビルドタグを指定して設定できますv1.7.5-eksbuild.2。また、 v1.7.5や などのアドオンのリリースバージョンに設定することもでき1.7.5、eksbuildサフィックスタグが検出されて設定されます。

利用可能なアドオンとそのバージョンを検出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。

アドオンの検出

クラスターにインストールできるアドオンを確認するには、以下を実行します。

```
eksctl utils describe-addon-versions --cluster <cluster-name>
```

これにより、クラスターの kubernetes バージョンが検出され、そのバージョンがフィルタリングされます。または、特定の kubernetes バージョンで使用できるアドオンを確認したい場合は、以下を実行します。

```
eksctl utils describe-addon-versions --kubernetes-version <version>
```

アドオンはtype、、ownerおよび/または でフィルタリングして検出することもできますpublisher。例えば、特定の所有者とタイプのアドオンを表示するには、以下を実行できます。

```
eksctl utils describe-addon-versions --kubernetes-version 1.22 --types "infra-
management, policy-management" --owners "aws-marketplace"
```

types、、 ownerspublishersフラグはオプションであり、結果をフィルタリングするために一緒にまたは個別に指定できます。

アドオンの設定スキーマの検出

アドオンとバージョンを検出したら、JSON 設定スキーマを取得してカスタマイズオプションを表示できます。

```
eksctl utils describe-addon-configuration --name vpc-cni --version v1.12.0-eksbuild.1
```

これにより、このアドオンで使用できるさまざまなオプションの JSON スキーマが返されます。

設定値の操作

ConfigurationValues は、アドオンの作成時または更新時に設定ファイルで提供できます。JSON 形式と YAML 形式のみがサポートされています。

例:

```
addons:
- name: coredns
  configurationValues: |-
    replicaCount: 2
```

```
addons:
- name: coredns
  version: latest
  configurationValues: "{\"replicaCount\":3}"
  resolveConflicts: overwrite
```

> **Note**
>
> アドオン設定値を変更すると、設定の競合が発生することに注意してください。

```
Thus, we need to specify how to deal with those by setting the `resolveConflicts` field
 accordingly.
As in this scenario we want to modify these values, we'd set `resolveConflicts:
 overwrite`.
```

さらに、get コマンドはアドオンConfigurationValuesの も取得するようになりました。例:

```
eksctl get addon --cluster my-cluster --output yaml
```

```
- ConfigurationValues: '{"replicaCount":3}'
  IAMRole: ""
  Issues: null
  Name: coredns
  NewerVersion: ""
  Status: ACTIVE
  Version: v1.8.7-eksbuild.3
```

カスタム名前空間の使用

カスタム名前空間は、アドオンの作成時に設定ファイルで指定できます。アドオンが作成されると、名前空間を更新することはできません。

設定ファイルの使用

```
addons:
  - name: aws-ebs-csi-driver
    version: latest
    namespaceConfig:
      namespace: custom-namespace
```

CLI フラグの使用

または、 --namespace-configフラグを使用してカスタム名前空間を指定することもできます。

```
eksctl create addon --cluster my-cluster --name aws-ebs-csi-driver --namespace-config
 'namespace=custom-namespace'
```

get コマンドはアドオンの名前空間値も取得します。

```
- ConfigurationValues: ""
  IAMRole: ""
  Issues: null
  Name: aws-ebs-csi-driver
  NamespaceConfig:
```

```
    namespace: custom-namespace
  NewerVersion: ""
  PodIdentityAssociations: null
  Status: ACTIVE
  Version: v1.47.0-eksbuild.1
```

アドオンの更新

アドオンを新しいバージョンに更新し、以下を実行してアタッチするポリシーを変更できます。

```
eksctl update addon -f config.yaml
```

```
eksctl update addon --name vpc-cni --version 1.8.0 --service-account-role-arn <new-
role>
```

> **Note**
>
> アドオンが作成されると、名前空間設定を更新することはできません。--namespace-config フラグはアドオンの作成時にのみ使用できます。

アドオンの作成と同様に、アドオンを更新すると、そのアドオンの に以前に適用した設定変更を完全に制御できますconfigMap。具体的には、それらを保存または上書きできます。このオプション機能は、同じ設定ファイルフィールド を介して使用できます。resolveConflicts例:

```
addons:
- name: vpc-cni
  attachPolicyARNs:
    - arn:aws:iam::aws:policy/AmazonEKS_CNI_Policy
  resolveConflicts: preserve
```

アドオンの更新の場合、 resolveConflictsフィールドは 3 つの異なる値を受け入れます。

- none - EKS は値を変更しません。更新が失敗する可能性があります。
- overwrite - EKS は、設定の変更を EKS のデフォルト値に上書きします。
- preserve - EKS は 値を保持します。このオプションを選択した場合は、本番クラスターのアドオンを更新する前に、非本番クラスターでフィールドと値の変更をテストすることをお勧めします。

アドオンの削除

アドオンを削除するには、以下を実行します。

```
eksctl delete addon --cluster <cluster-name> --name <addon-name>
```

これにより、アドオンとそれに関連付けられた IAM ロールが削除されます。

クラスターを削除すると、アドオンに関連付けられたすべての IAM ロールも削除されます。

デフォルトのネットワーキングアドオンのクラスター作成の柔軟性

クラスターが作成されると、EKS は VPC CNI、CoreDNS、kube-proxy をセルフマネージドアドオンとして自動的にインストールします。Cilium や Calico などの他の CNI プラグインを使用するためにこの動作を無効にするために、eksctl はデフォルトのネットワークアドオンなしでクラスターの作成をサポートするようになりました。このようなクラスターを作成するには、次のように を設定しますaddonsConfig.disableDefaultAddons。

```
addonsConfig:
  disableDefaultAddons: true
```

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

CoreDNS と kube-proxy のみを使用してクラスターを作成し、VPC CNI を使用してクラスターを作成するには、 で明示的にアドオンを指定addonsしaddonsConfig.disableDefaultAddons、次のように を設定します。

```
addonsConfig:
  disableDefaultAddons: true
addons:
  - name: kube-proxy
  - name: coredns
```

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

この変更の一環として、 が明示的に true に設定されていない場合、eksctl addonsConfig.disableDefaultAddonsはクラスターの作成中にセルフマネージド型アドオンではなく EKS アドオンとしてデフォルトアドオンをインストールするようになりました。そのた

め、eksctl utils update-*コマンドは eksctl v0.184.0 以降で作成されたクラスターのアドオンの更新に使用されなくなりました。

- eksctl utils update-aws-node
- eksctl utils update-coredns
- eksctl utils update-kube-proxy

代わりに、 を今すぐ使用eksctl update addonする必要があります。

詳細については、「Amazon EKS がネットワークアドオンにクラスター作成の柔軟性を導入する」を参照してください。

Amazon EMR へのアクセスの有効化

EMR が Kubernetes API でオペレーションを実行できるようにするには、その SLR に必要な RBAC アクセス許可を付与する必要があります。eksctl は、EMR に必要な RBAC リソースを作成するコマンドを提供し、ConfigMap aws-auth を更新してロールを EMR の SLR にバインドします。

```
eksctl create iamidentitymapping --cluster dev --service-name emr-containers --
namespace default
```

EKS Fargate のサポート

AWS Fargate は、コンテナを実行できる Amazon ECS のマネージドコンピューティングエンジンです。Fargate では、サーバーやクラスターを管理する必要はありません。

Amazon EKS が AWS Fargate でポッドを起動できるようになりました。これにより、ポッドのインフラストラクチャをプロビジョニングまたは管理する方法について心配する必要がなくなり、AWS での高性能で可用性の高い Kubernetes アプリケーションの構築と実行が容易になります。

Fargate をサポートするクラスターの作成

Fargate をサポートするクラスターは、以下を使用して追加できます。

```
eksctl create cluster --fargate
[#]  eksctl version 0.11.0
[#]  using region ap-northeast-1
[#]  setting availability zones to [ap-northeast-1a ap-northeast-1d ap-northeast-1c]
[#]  subnets for ap-northeast-1a - public:192.168.0.0/19 private:192.168.96.0/19
```

```
[#]  subnets for ap-northeast-1d - public:192.168.32.0/19 private:192.168.128.0/19
[#]  subnets for ap-northeast-1c - public:192.168.64.0/19 private:192.168.160.0/19
[#]  nodegroup "ng-dba9d731" will use "ami-02e124a380df41614" [AmazonLinux2/1.14]
[#]  using Kubernetes version 1.14
[#]  creating EKS cluster "ridiculous-painting-1574859263" in "ap-northeast-1" region
[#]  will create 2 separate CloudFormation stacks for cluster itself and the initial
 nodegroup
[#]  if you encounter any issues, check CloudFormation console or try 'eksctl utils
 describe-stacks --region=ap-northeast-1 --cluster=ridiculous-painting-1574859263'
[#]  CloudWatch logging will not be enabled for cluster "ridiculous-
painting-1574859263" in "ap-northeast-1"
[#]  you can enable it with 'eksctl utils update-cluster-logging --enable-
types={SPECIFY-YOUR-LOG-TYPES-HERE (e.g. all)} --region=ap-northeast-1 --
cluster=ridiculous-painting-1574859263'
[#]  Kubernetes API endpoint access will use default of {publicAccess=true,
 privateAccess=false} for cluster "ridiculous-painting-1574859263" in "ap-northeast-1"
[#]  2 sequential tasks: { create cluster control plane "ridiculous-
painting-1574859263", create nodegroup "ng-dba9d731" }
[#]  building cluster stack "eksctl-ridiculous-painting-1574859263-cluster"
[#]  deploying stack "eksctl-ridiculous-painting-1574859263-cluster"
[#]  building nodegroup stack "eksctl-ridiculous-painting-1574859263-nodegroup-ng-
dba9d731"
[#]  --nodes-min=2 was set automatically for nodegroup ng-dba9d731
[#]  --nodes-max=2 was set automatically for nodegroup ng-dba9d731
[#]  deploying stack "eksctl-ridiculous-painting-1574859263-nodegroup-ng-dba9d731"
[#]  all EKS cluster resources for "ridiculous-painting-1574859263" have been created
[#]  saved kubeconfig as "/Users/marc/.kube/config"
[#]  adding identity "arn:aws:iam::123456789012:role/eksctl-ridiculous-painting-157485-
NodeInstanceRole-104DXUJOFDPO5" to auth ConfigMap
[#]  nodegroup "ng-dba9d731" has 0 node(s)
[#]  waiting for at least 2 node(s) to become ready in "ng-dba9d731"
[#]  nodegroup "ng-dba9d731" has 2 node(s)
[#]  node "ip-192-168-27-156.ap-northeast-1.compute.internal" is ready
[#]  node "ip-192-168-95-177.ap-northeast-1.compute.internal" is ready
[#]  creating Fargate profile "default" on EKS cluster "ridiculous-painting-1574859263"
[#]  created Fargate profile "default" on EKS cluster "ridiculous-painting-1574859263"
[#]  kubectl command should work with "/Users/marc/.kube/config", try 'kubectl get
 nodes'
[#]  EKS cluster "ridiculous-painting-1574859263" in "ap-northeast-1" region is ready
```

このコマンドは、クラスターと Fargate プロファイルを作成します。このプロファイルには、Fargate でポッドをインスタンス化するために AWS が必要とする特定の情報が含まれています。次のようなものがあります。

- ポッドを実行するために必要なアクセス許可 と、ポッドを実行するネットワークの場所 (サブネット) を定義する ポッド実行ロール。これにより、同じネットワークとセキュリティのアクセス許可を複数の Fargate ポッドに適用でき、クラスター上の既存のポッドを Fargate に移行することが容易になります。
- Fargate で実行するポッドを定義するセレクタ。これは、 namespaceと で構成されますlabels。

プロファイルが指定されていないが、デフォルトの Fargate プロファイルで Fargate のサポートが有効になってい--fargateる場合。このプロファイルは、 defaultおよび kube-system名前空間をターゲットとしているため、それらの名前空間のポッドは Fargate で実行されます。

作成された Fargate プロファイルは、次のコマンドで確認できます。

```
eksctl get fargateprofile --cluster ridiculous-painting-1574859263 -o yaml
- name: fp-default
  podExecutionRoleARN: arn:aws:iam::123456789012:role/eksctl-ridiculous-
painting-1574859263-ServiceRole-EIFQOH0S1GE7
  selectors:
  - namespace: default
  - namespace: kube-system
  subnets:
  - subnet-0b3a5522f3b48a742
  - subnet-0c35f1497067363f3
  - subnet-0a29aa00b25082021
```

セレクタの詳細については、「Fargate プロファイルの設計」を参照してください。

設定ファイルを使用した Fargate サポートによるクラスターの作成

次の設定ファイルは、1 つの EC2 m5.largeインスタンスと 2 つの Fargate プロファイルで構成されるノードグループの両方を持つ EKS クラスターを宣言します。default および kube-system名前空間で定義されたすべてのポッドは Fargate で実行されます。ラベルも付いているdev名前空間内のすべてのポッドdev=passedも Fargate で実行されます。その他のポッドは、 のノードでスケジュールされますng-1。

```
# An example of ClusterConfig with a normal nodegroup and a Fargate profile.
---
apiVersion: eksctl.io/v1alpha5
kind: ClusterConfig
```

```
metadata:
  name: fargate-cluster
  region: ap-northeast-1

nodeGroups:
  - name: ng-1
    instanceType: m5.large
    desiredCapacity: 1

fargateProfiles:
  - name: fp-default
    selectors:
      # All workloads in the "default" Kubernetes namespace will be
      # scheduled onto Fargate:
      - namespace: default
      # All workloads in the "kube-system" Kubernetes namespace will be
      # scheduled onto Fargate:
      - namespace: kube-system
  - name: fp-dev
    selectors:
      # All workloads in the "dev" Kubernetes namespace matching the following
      # label selectors will be scheduled onto Fargate:
      - namespace: dev
        labels:
          env: dev
          checks: passed
```

```
eksctl create cluster -f cluster-fargate.yaml
[#]  eksctl version 0.11.0
[#]  using region ap-northeast-1
[#]  setting availability zones to [ap-northeast-1c ap-northeast-1a ap-northeast-1d]
[#]  subnets for ap-northeast-1c - public:192.168.0.0/19 private:192.168.96.0/19
[#]  subnets for ap-northeast-1a - public:192.168.32.0/19 private:192.168.128.0/19
[#]  subnets for ap-northeast-1d - public:192.168.64.0/19 private:192.168.160.0/19
[#]  nodegroup "ng-1" will use "ami-02e124a380df41614" [AmazonLinux2/1.14]
[#]  using Kubernetes version 1.14
[#]  creating EKS cluster "fargate-cluster" in "ap-northeast-1" region with Fargate
 profile and un-managed nodes
[#]  1 nodegroup (ng-1) was included (based on the include/exclude rules)
[#]  will create a CloudFormation stack for cluster itself and 1 nodegroup stack(s)
[#]  will create a CloudFormation stack for cluster itself and 0 managed nodegroup
 stack(s)
```

```
[#]  if you encounter any issues, check CloudFormation console or try 'eksctl utils
 describe-stacks --region=ap-northeast-1 --cluster=fargate-cluster'
[#]  CloudWatch logging will not be enabled for cluster "fargate-cluster" in "ap-
northeast-1"
[#]  you can enable it with 'eksctl utils update-cluster-logging --enable-
types={SPECIFY-YOUR-LOG-TYPES-HERE (e.g. all)} --region=ap-northeast-1 --
cluster=fargate-cluster'
[#]  Kubernetes API endpoint access will use default of {publicAccess=true,
 privateAccess=false} for cluster "fargate-cluster" in "ap-northeast-1"
[#]  2 sequential tasks: { create cluster control plane "fargate-cluster", create
 nodegroup "ng-1" }
[#]  building cluster stack "eksctl-fargate-cluster-cluster"
[#]  deploying stack "eksctl-fargate-cluster-cluster"
[#]  building nodegroup stack "eksctl-fargate-cluster-nodegroup-ng-1"
[#]  --nodes-min=1 was set automatically for nodegroup ng-1
[#]  --nodes-max=1 was set automatically for nodegroup ng-1
[#]  deploying stack "eksctl-fargate-cluster-nodegroup-ng-1"
[#]  all EKS cluster resources for "fargate-cluster" have been created
[#]  saved kubeconfig as "/home/user1/.kube/config"
[#]  adding identity "arn:aws:iam::123456789012:role/eksctl-fargate-cluster-nod-
NodeInstanceRole-42Q80B2Z147I" to auth ConfigMap
[#]  nodegroup "ng-1" has 0 node(s)
[#]  waiting for at least 1 node(s) to become ready in "ng-1"
[#]  nodegroup "ng-1" has 1 node(s)
[#]  node "ip-192-168-71-83.ap-northeast-1.compute.internal" is ready
[#]  creating Fargate profile "fp-default" on EKS cluster "fargate-cluster"
[#]  created Fargate profile "fp-default" on EKS cluster "fargate-cluster"
[#]  creating Fargate profile "fp-dev" on EKS cluster "fargate-cluster"
[#]  created Fargate profile "fp-dev" on EKS cluster "fargate-cluster"
[#]  "coredns" is now schedulable onto Fargate
[#]  "coredns" is now scheduled onto Fargate
[#]  "coredns" is now scheduled onto Fargate
[#]  "coredns" pods are now scheduled onto Fargate
[#]  kubectl command should work with "/home/user1/.kube/config", try 'kubectl get
 nodes'
[#]  EKS cluster "fargate-cluster" in "ap-northeast-1" region is ready
```

Fargate プロファイルの設計

各セレクタエントリには、名前空間とキーと値のペアのリストという最大 2 つのコンポーネントがあります。セレクタエントリを作成するには、名前空間コンポーネントのみが必要です。セレクタエントリと一致するには、すべてのルール (名前空間、キーと値のペア) がポッドに適用されている必要があります。ポッドは、プロファイルで実行するセレクタエントリを 1 つだけ一致させる必

要があります。セレクタフィールドのすべての条件に一致するポッドは、Fargate で実行されるようにスケジュールされます。ホワイトリストに登録されている 名前空間のいずれとも一致しないポッドでも、ユーザーが手動でス ケジューラを設定すると、Fargate での実行が許可されていないため、 fargate-scheduler filed は保留状態のままになります。

プロファイルは、次の要件を満たしている必要があります。

- プロファイルごとに 1 つのセレクタが必須です
- 各セレクタには名前空間を含める必要があります。ラベルはオプションです

例: Fargate でのワークロードのスケジューリング

上記の例の Fargate でポッドをスケジュールするには、例えば、 という名前空間を作成し、そこにワークロードをdevデプロイします。

```
kubectl create namespace dev
namespace/dev created

kubectl run nginx --image=nginx --restart=Never --namespace dev
pod/nginx created

kubectl get pods --all-namespaces --output wide
NAMESPACE     NAME                       READY   STATUS    AGE   IP                NODE
dev           nginx                      1/1     Running   75s   192.168.183.140
 fargate-ip-192-168-183-140.ap-northeast-1.compute.internal
kube-system   aws-node-44qst             1/1     Running   21m   192.168.70.246
 ip-192-168-70-246.ap-northeast-1.compute.internal
kube-system   aws-node-4vr66             1/1     Running   21m   192.168.23.122
 ip-192-168-23-122.ap-northeast-1.compute.internal
kube-system   coredns-699bb99bf8-84x74   1/1     Running   26m   192.168.2.95
 ip-192-168-23-122.ap-northeast-1.compute.internal
kube-system   coredns-699bb99bf8-f6x6n   1/1     Running   26m   192.168.90.73
 ip-192-168-70-246.ap-northeast-1.compute.internal
kube-system   kube-proxy-brxhg           1/1     Running   21m   192.168.23.122
 ip-192-168-23-122.ap-northeast-1.compute.internal
kube-system   kube-proxy-zd7s8           1/1     Running   21m   192.168.70.246
 ip-192-168-70-246.ap-northeast-1.compute.internal
```

最後のkubectl get podsコマンドの出力から、ポッドが nginx というノードにデプロイされていることを確認できますfargate-ip-192-168-183-140.ap-northeast-1.compute.internal。

Fargate プロファイルの管理

Fargate に Kubernetes ワークロードをデプロイするには、EKS に Fargate プロファイルが必要です。上記の例のようなクラスターを作成する場合、 eksctl はデフォルトのプロファイルを作成してこれを処理します。既存のクラスターがある場合、 eksctl create fargateprofile コマンドを使用して Fargate プロファイルを作成することもできます。

> **Note**
>
> このオペレーションは、EKS プラットフォームバージョン eks.5以降で実行されるクラスターでのみサポートされます。

> **Note**
>
> 既存の が 0.11.0 eksctlより前のバージョンで作成されている場合は、Fargate プロファイルを作成するeksctl upgrade cluster前に を実行する必要があります。

```
eksctl create fargateprofile --namespace dev --cluster fargate-example-cluster
[#]  creating Fargate profile "fp-9bfc77ad" on EKS cluster "fargate-example-cluster"
[#]  created Fargate profile "fp-9bfc77ad" on EKS cluster "fargate-example-cluster"
```

作成する Fargate プロファイルの名前を指定することもできます。この名前はプレフィックス で始まることはできませんeks-。

```
eksctl create fargateprofile --namespace dev --cluster fargate-example-cluster --name
 fp-development
[#]  created Fargate profile "fp-development" on EKS cluster "fargate-example-cluster"
```

CLI フラグ eksctl でこのコマンドを使用すると、シンプルなセレクタで単一の Fargate プロファイルのみを作成できます。名前空間が多いなど、より複雑なセレクタの場合、eksctl は設定ファイルの使用をサポートします。

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

metadata:
```

```
  name: fargate-example-cluster
  region: ap-northeast-1

fargateProfiles:
  - name: fp-default
    selectors:
      # All workloads in the "default" Kubernetes namespace will be
      # scheduled onto Fargate:
      - namespace: default
      # All workloads in the "kube-system" Kubernetes namespace will be
      # scheduled onto Fargate:
      - namespace: kube-system
  - name: fp-dev
    selectors:
      # All workloads in the "dev" Kubernetes namespace matching the following
      # label selectors will be scheduled onto Fargate:
      - namespace: dev
        labels:
          env: dev
          checks: passed
```

```
eksctl create fargateprofile -f fargate-example-cluster.yaml
[#]  creating Fargate profile "fp-default" on EKS cluster "fargate-example-cluster"
[#]  created Fargate profile "fp-default" on EKS cluster "fargate-example-cluster"
[#]  creating Fargate profile "fp-dev" on EKS cluster "fargate-example-cluster"
[#]  created Fargate profile "fp-dev" on EKS cluster "fargate-example-cluster"
[#]  "coredns" is now scheduled onto Fargate
[#]  "coredns" pods are now scheduled onto Fargate
```

クラスター内の既存の Fargate プロファイルを表示するには:

```
eksctl get fargateprofile --cluster fargate-example-cluster
NAME         SELECTOR_NAMESPACE  SELECTOR_LABELS  POD_EXECUTION_ROLE_ARN
                                                    SUBNETS
fp-9bfc77ad  dev                 <none>           arn:aws:iam::123456789012:role/
eksctl-fargate-example-cluster-ServiceRole-1T5F78E5FSH79
 subnet-00adf1d8c99f83381,subnet-04affb163ffab17d4,subnet-035b34379d5ef5473
```

また、 yaml形式で表示するには:

```
eksctl get fargateprofile --cluster fargate-example-cluster -o yaml
- name: fp-9bfc77ad
```

```
  podExecutionRoleARN: arn:aws:iam::123456789012:role/eksctl-fargate-example-cluster-
ServiceRole-1T5F78E5FSH79
  selectors:
  - namespace: dev
  subnets:
  - subnet-00adf1d8c99f83381
  - subnet-04affb163ffab17d4
  - subnet-035b34379d5ef5473
```

または json形式:

```
eksctl get fargateprofile --cluster fargate-example-cluster -o json
[
    {
        "name": "fp-9bfc77ad",
        "podExecutionRoleARN": "arn:aws:iam::123456789012:role/eksctl-fargate-example-
cluster-ServiceRole-1T5F78E5FSH79",
        "selectors": [
            {
                "namespace": "dev"
            }
        ],
        "subnets": [
            "subnet-00adf1d8c99f83381",
            "subnet-04affb163ffab17d4",
            "subnet-035b34379d5ef5473"
        ]
    }
]
```

Fargate プロファイルは設計上イミュータブルです。何かを変更するには、必要な変更を含む新しい Fargate プロファイルを作成し、次の例のように `eksctl delete fargateprofile` コマンドを使用して古いプロファイルを削除します。

```
eksctl delete fargateprofile --cluster fargate-example-cluster --name fp-9bfc77ad --
wait
2019-11-27T19:04:26+09:00 [#]  deleting Fargate profile "fp-9bfc77ad"
  ClusterName: "fargate-example-cluster",
  FargateProfileName: "fp-9bfc77ad"
}
```

プロファイルの削除は、数分かかる場合があるプロセスであることに注意してください。--wait フラグが指定されていない場合、 はプロファイルが削除されることをeksctl楽観的に想定し、AWS API リクエストが送信されるとすぐに を返します。プロファイルが正常に削除されるまでeksctl待機するには、上記の例--waitのように を使用します。

詳細情報

- AWS Fargate
- Amazon EKS が AWS Fargate でポッドを起動できるようになりました

クラスターのアップグレード

「eksctl」マネージドクラスターは、3 つの簡単なステップでアップグレードできます。

1. を使用したコントロールプレーンバージョンのアップグレード `eksctl upgrade cluster`
2. ノードグループのアップグレード
3. デフォルトのネットワークアドオンを更新します (詳細については、「」を参照してくださいthe section called “デフォルトのアドオンの更新”)。

クラスターのアップグレードに関連するリソースを慎重に確認します。

- 「Amazon EKS ユーザーガイド」の「既存のクラスターを新しい Kubernetes バージョンに更新する」
- 「EKS ベストプラクティスガイド」の「クラスターのアップグレードのベストプラクティス」

> **Note**
>
> 古い `eksctl update cluster`は廃止されます。代わりに `eksctl upgrade cluster` を使用します。

コントロールプレーンバージョンの更新

コントロールプレーンのバージョンのアップグレードは、一度に 1 つのマイナーバージョンに対して行う必要があります。

コントロールプレーンを次に使用可能なバージョン実行にアップグレードするには:

```
eksctl upgrade cluster --name=<clusterName>
```

このコマンドはすぐに変更を適用しません。変更--approveを適用するには、 で再実行する必要があります。

クラスターアップグレードのターゲットバージョンは、 CLI フラグの両方を使用して指定できます。

```
eksctl upgrade cluster --name=<clusterName> --version=1.16
```

または と 設定ファイル

```
cat cluster1.yaml
---
apiVersion: eksctl.io/v1alpha5
kind: ClusterConfig

metadata:
  name: cluster-1
  region: eu-north-1
  version: "1.16"

eksctl upgrade cluster --config-file cluster1.yaml
```

> **Warning**
>
> 引数--versionと metadata.version引数に使用できる値は、クラスターの現在のバージョンまたは 1 つ上のバージョンのみです。複数の Kubernetes バージョンのアップグレードはサポートされていません。

デフォルトのアドオンの更新

このトピックでは、EKS クラスターに含まれるデフォルトのプリインストールされたアドオンを更新する方法について説明します。

> **Warning**
>
> eksctl は、セルフマネージドアドオンではなく EKS アドオンとしてデフォルトのアドオンをインストールするようになりました。デフォルトのネットワーキングアドオンのクラスター作成の柔軟性における影響について詳しく説明します。
> アドオンを更新する場合、eksctl v0.184.0 以降で作成されたクラスターに `eksctl utils update-<addon>` を使用することはできません。このガイドは、この変更前に作成されたクラスターに対してのみ有効です。

各 EKS クラスターに含まれるデフォルトのアドオンは 3 つあります。

- `kube-proxy`
- `aws-node`
- `coredns`

プリインストールされたアドオンを更新する

クラスターの作成時`eksctl create addons`または作成時に手動で作成される公式 EKS アドオンの場合、それらを管理する方法は ですeksctl create/get/update/delete addon。このような場合は、EKS アドオンに関するドキュメントを参照してください。

それぞれを更新するプロセスは異なるため、実行する必要があるコマンドは 3 つあります。以下のコマンドはすべて を受け入れます`--config-file`。デフォルトでは、これらの各コマンドはプランモードで実行されます。提案された変更に満足している場合は、 で再実行してください`--approve`。

を更新するには`kube-proxy`、以下を実行します。

```
eksctl utils update-kube-proxy --cluster=<clusterName>
```

を更新するには`aws-node`、以下を実行します。

```
eksctl utils update-aws-node --cluster=<clusterName>
```

を更新するには`coredns`、以下を実行します。

```
eksctl utils update-coredns --cluster=<clusterName>
```

アップグレードしたら、 を実行しkubectl get pods -n kube-system、すべてのアドオンポッドが準備完了状態になっているかどうかを確認してください。次のようになります。

```
NAME                       READY   STATUS    RESTARTS   AGE
aws-node-g5ghn             1/1     Running   0          2m
aws-node-zfc9s             1/1     Running   0          2m
coredns-7bcbfc4774-g6gg8   1/1     Running   0          1m
coredns-7bcbfc4774-hftng   1/1     Running   0          1m
kube-proxy-djkp7           1/1     Running   0          3m
kube-proxy-mpdsp           1/1     Running   0          3m
```

EKS クラスターでのゾーンシフトのサポート

EKS は、マルチ AZ クラスター環境の耐障害性を強化する Amazon Application Recovery Controller (ARC) ゾーンシフトとゾーンオートシフトをサポートするようになりました。AWS ゾーンシフトを使用すると、クラスター内トラフィックを障害のあるアベイラビリティーゾーンから移行できるため、新しい Kubernetes ポッドとノードは正常なアベイラビリティーゾーンでのみ起動されます。

ゾーンシフトを有効にしたクラスターの作成

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

metadata:
  name: highly-available-cluster
  region: us-west-2


zonalShiftConfig:
  enabled: true
```

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

既存のクラスターでゾーンシフトを有効にする

既存のクラスターでゾーンシフトを有効または無効にするには、 を実行します。

```
eksctl utils update-zonal-shift-config -f zonal-shift-cluster.yaml
```

設定ファイルがない場合:

```
eksctl utils update-zonal-shift-config --cluster=zonal-shift-cluster --enabled
```

詳細情報

- EKS ゾーンシフト

Karpenter サポート

`eksctl` は、新しく作成されたクラスターに Karpenter を追加するためのサポートを提供します。これは、Helm を使用して Karpenter 自体をインストールするなど、Karpenter の開始方法セクションで説明されているすべての必要な前提条件を作成します。現在、バージョン のインストールをサポートしています`0.28.0+`。詳細については、Karpenter の互換性セクションを参照してください。

次のクラスター設定は、一般的な Karpenter のインストールの概要を示しています。

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

metadata:
  name: cluster-with-karpenter
  region: us-west-2
  version: '1.32' # requires a version of Kubernetes compatible with Karpenter
  tags:
    karpenter.sh/discovery: cluster-with-karpenter # here, it is set to the cluster
 name
iam:
  withOIDC: true # required

karpenter:
  version: '1.2.1' # Exact version should be specified according to the Karpenter
 compatibility matrix

managedNodeGroups:
  - name: managed-ng-1
    minSize: 1
    maxSize: 2
```

```
    desiredCapacity: 1
```

バージョンは、Helm リポジトリにある Karpenter のバージョンです。以下のオプションを設定することもできます。

```
karpenter:
  version: '1.2.1'
  createServiceAccount: true # default is false
  defaultInstanceProfile: 'KarpenterNodeInstanceProfile' # default is to use the IAM
 instance profile created by eksctl
  withSpotInterruptionQueue: true # adds all required policies and rules for supporting
 Spot Interruption Queue, default is false
```

Karpenter をインストールするには、OIDC を定義する必要があります。

Karpenter が正常にインストールされたら、NodePool (複数可) と NodeClass (複数可) を追加して、Karpenter がクラスターへのノードの追加を開始できるようにします。

NodePool の nodeClassRefセクションは、 の名前と一致する必要がありますEC2NodeClass。例えば、次のようになります。

```
apiVersion: karpenter.sh/v1
kind: NodePool
metadata:
  name: example
  annotations:
    kubernetes.io/description: "Example NodePool"
spec:
  template:
    spec:
      requirements:
        - key: kubernetes.io/arch
          operator: In
          values: ["amd64"]
        - key: kubernetes.io/os
          operator: In
          values: ["linux"]
        - key: karpenter.sh/capacity-type
          operator: In
          values: ["on-demand"]
        - key: karpenter.k8s.aws/instance-category
          operator: In
```

```
          values: ["c", "m", "r"]
        - key: karpenter.k8s.aws/instance-generation
          operator: Gt
          values: ["2"]
      nodeClassRef:
        group: karpenter.k8s.aws
        kind: EC2NodeClass
        name: example # must match the name of an EC2NodeClass
```

```
apiVersion: karpenter.k8s.aws/v1
kind: EC2NodeClass
metadata:
  name: example
  annotations:
    kubernetes.io/description: "Example EC2NodeClass"
spec:
  role: "eksctl-KarpenterNodeRole-${CLUSTER_NAME}" # replace with your cluster name
  subnetSelectorTerms:
    - tags:
        karpenter.sh/discovery: "${CLUSTER_NAME}" # replace with your cluster name
  securityGroupSelectorTerms:
    - tags:
        karpenter.sh/discovery: "${CLUSTER_NAME}" # replace with your cluster name
  amiSelectorTerms:
    - alias: al2023@latest # Amazon Linux 2023
```

ラウチノードには roleまたは instanceProfile のいずれかを指定する必要があることに注意してください。によって作成されたプロファイルinstanceProfileの名前を使用する場合は、パターン eksctlに従いますeksctl-KarpenterNodeInstanceProfile-<cluster-name>。

セキュリティグループの自動タグ付け

eksctl Karpenter が有効 (karpenter.version指定) で タグが に存在するkarpenter.sh/discovery場合、クラスターの共有ノードセキュリティグループには が自動的にkarpenter.sh/discoveryタグ付けされますmetadata.tags。これにより、AWS Load Balancer Controller の互換性が有効になります。

karpenter 0.32.0 以降では、プロビジョニングは廃止され、NodePool に置き換えられました。

クラスター設定スキーマ

> **Note**
>
> スキーマの場所は現在移行中です。

yaml ファイルを使用してクラスターを作成できます。スキーマリファレンスを表示します。

例えば、次のようになります。

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

このファイルのスキーマリファレンスは GitHub で入手できます。

ファイルの使用の詳細については、「」を参照してくださいthe section called “クラスターの作成と管理”。

ノードグループ

この章には、Eksctl で Nodegroups を作成および設定する方法に関する情報が含まれています。ノードグループは、EKS クラスターにアタッチされた EC2 インスタンスのグループです。

トピック:

- EKS Managed Nodegroups のノード修復を有効にして、異常なワーカーノードを自動的にモニタリングおよび置換または再起動する

- the section called “ARM サポート”
  - ARM ベースの Graviton インスタンスを使用して EKS クラスターを作成し、パフォーマンスとコスト効率を向上させる
- the section called “テイント”
  - Kubernetes クラスター内の特定のノードグループにテイントを適用する
  - テイントキー、値、効果に基づいてポッドのスケジュールと削除を制御する
- the section called “起動テンプレートのサポート”
  - 提供された EC2 起動テンプレートを使用したマネージド型ノードグループの起動
  - 別のバージョンの起動テンプレートを使用するようにマネージド型ノードグループをアップグレードする
  - マネージド型ノードグループでカスタム AMIs と起動テンプレートを使用する場合の制限と考慮事項を理解する
- the section called “ノードグループを操作する”
  - ノードグループ内の EC2 インスタンスへの SSH アクセスを有効にする
  - ノードグループ内のノード数をスケールアップまたはスケールダウンする
- the section called “カスタムサブネット”
  - 既存の VPC を新しいサブネットで拡張し、そのサブネットに Nodegroup を追加する
- the section called “ノードブートストラップ”
  - AmazonLinux2023 で導入された新しいノード初期化プロセス (nodeadm) を理解する
  - eksctl によってセルフマネージドノードと EKS マネージドノードに適用されるデフォルトの NodeConfig 設定について説明します。
  - カスタム NodeConfig で overrideBootstrapCommand を指定してノードブートストラッププロセスをカスタマイズする
- the section called “アンマネージド型ノードグループ”
  - EKS クラスターでアンマネージド型ノードグループを作成または更新する
  - kube-proxy、aws-node、CoreDNS などのデフォルトの Kubernetes アドオンを更新する
- the section called “GPU サポート”
  - Eksctl はノードグループの GPU インスタンスタイプの選択をサポートしているため、EKS クラスターで GPU アクセラレーションワークロードを使用できます。

  - Eksctl は、GPU 対応インスタンスタイプが選択されると NVIDIA Kubernetes デバイスプラグインを自動的にインストールするため、クラスター内の GPU リソースの使用が容易になります。
  - ユーザーは、提供されたコマンドを使用して、自動プラグインのインストールを無効にし、特定のバージョンの NVIDIA Kubernetes デバイスプラグインを手動でインストールできます。
- the section called “インスタンスセレクタ”
  - vCPUs、メモリ、GPUs、適切な EC2 インスタンスタイプのリストを自動的に生成する
  - 指定されたインスタンスセレクタ条件に一致するインスタンスタイプを使用してクラスターとノードグループを作成する
  - ノードグループを作成する前に、ドライランを実行して、インスタンスセレクタに一致するインスタンスタイプを検査および変更します。
- the section called “追加のボリュームマッピング”
  - EKS クラスター内のマネージドノードグループの追加のボリュームマッピングを設定する
  - 追加ボリュームのサイズ、タイプ、暗号化、IOPS、スループットなどのボリュームプロパティをカスタマイズする
  - 既存の EBS スナップショットを追加ボリュームとしてノードグループにアタッチする
- the section called “Windows ワーカーノード”
  - Windows ノードグループを既存の Linux Kubernetes クラスターに追加して、Windows ワークロードの実行を有効にする
  - `kubernetes.io/os` および `kubernetes.io/arch`ラベルに基づくノードセレクタを使用して、適切なオペレーティングシステム (Windows または Linux) でワークロードをスケジュールする
- the section called “カスタム AMI サポート”
  - `--node-ami` フラグを使用してノードグループのカスタム AMI を指定するか、最新の EKS 最適化 AMI を AWS にクエリするか、AWS Systems Manager パラメータストアを使用して AMI を検索します。
  - `--node-ami-family` フラグを設定して、 AmazonLinux2, Ubuntu2204, WindowsServer2022CoreContainer などのノードグループ AMI のオペレーティングシステムファミリーを指定します。
  - Windows ノードグループの場合は、カスタム AMI を指定し、 経由で PowerShell ブートストラップスクリプトを指定しますoverrideBootstrapCommand。
- the section called “カスタム DNS”
  - 内部および外部 DNS ルックアップに使用される DNS サーバーの IP アドレスを上書きする

ノードグループを操作する

ノードグループの作成

クラスターとともに作成された最初のノードグループに加えて、1 つ以上のノードグループを追加できます。

追加のノードグループを作成するには、以下を使用します。

```
eksctl create nodegroup --cluster=<clusterName> [--name=<nodegroupName>]
```

> **Note**
>
> --version フラグはマネージド型ノードグループではサポートされていません。常にコントロールプレーンからバージョンを継承します。

デフォルトでは、新しいアンマネージド型ノードグループはコントロールプレーン (--version=auto) からバージョンを継承しますが、別のバージョンを指定することもできます。また、 --version=latestを使用して最新バージョンのどちらかを強制的に使用できます。

さらに、 と同じ設定ファイルを使用できますeksctl create cluster。

```
eksctl create nodegroup --config-file=<path>
```

設定ファイルからのノードグループの作成

Nodegroups は、クラスター定義または設定ファイルを使用して作成することもできます。次の設定ファイルの例と、 という既存のクラスターがあるとしますdev-cluster。

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

metadata:
  name: dev-cluster
  region: eu-north-1

managedNodeGroups:
```

```
  - name: ng-1-workers
    labels: { role: workers }
    instanceType: m5.xlarge
    desiredCapacity: 10
    volumeSize: 80
    privateNetworking: true
  - name: ng-2-builders
    labels: { role: builders }
    instanceType: m5.2xlarge
    desiredCapacity: 2
    volumeSize: 100
    privateNetworking: true
```

ノードグループng-1-workersと は、次のコマンドで作成ng-2-buildersできます。

```
eksctl create nodegroup --config-file=dev-cluster.yaml
```

ロードバランシング

既存のクラシックロードバランサーまたは/およびターゲットグループをノードグループにアタッチする準備ができている場合は、設定ファイルでこれらを指定できます。クラシックロードバランサーまたは/およびターゲットグループは、ノードグループの作成時に自動的に ASG に関連付けられます。これは、 nodeGroupsフィールドで定義されたセルフマネージド型ノードグループでのみサポートされます。

```
# dev-cluster-with-lb.yaml
apiVersion: eksctl.io/v1alpha5
kind: ClusterConfig

metadata:
  name: dev-cluster
  region: eu-north-1

nodeGroups:
  - name: ng-1-web
    labels: { role: web }
    instanceType: m5.xlarge
    desiredCapacity: 10
    privateNetworking: true
    classicLoadBalancerNames:
      - dev-clb-1
      - dev-clb-2
```

```
    asgMetricsCollection:
      - granularity: 1Minute
        metrics:
          - GroupMinSize
          - GroupMaxSize
          - GroupDesiredCapacity
          - GroupInServiceInstances
          - GroupPendingInstances
          - GroupStandbyInstances
          - GroupTerminatingInstances
          - GroupTotalInstances
  - name: ng-2-api
    labels: { role: api }
    instanceType: m5.2xlarge
    desiredCapacity: 2
    privateNetworking: true
    targetGroupARNs:
      - arn:aws:elasticloadbalancing:eu-north-1:01234567890:targetgroup/dev-target-
group-1/abcdef0123456789
```

設定ファイルでのノードグループの選択

設定ファイルで指定されたノードグループのサブセットに対してのみ `create` または `delete`オペレーションを実行するには、glob のリストを受け入れる CLI フラグ`0`が 2 つあります。1次に例を示します。

```
eksctl create nodegroup --config-file=<path> --include='ng-prod-*-??' --exclude='ng-
test-1-ml-a,ng-test-2-?'
```

上記の設定ファイルの例を使用すると、次のコマンドでワーカーを除いたすべてのワーカーノードグループを作成できます。

```
eksctl create nodegroup --config-file=dev-cluster.yaml --exclude=ng-1-workers
```

または、以下を使用してビルダーノードグループを削除することもできます。

```
eksctl delete nodegroup --config-file=dev-cluster.yaml --include=ng-2-builders --
approve
```

この場合、ノードグループを実際に削除する`--approve`には コマンドも指定する必要があります。

ルールを含めるおよび除外する

- `--include` または `--exclude` が指定されていない場合は、すべてが含まれます。
- `--include` のみを指定した場合、それらの glob に一致するノードグループのみが含まれます。
- `--exclude` のみを指定した場合、それらの glob と一致しないすべてのノードグループが含まれます。
- 両方が指定されている場合、`--exclude`ルールが優先されます `--include` (つまり、両方のグループのルールに一致するノードグループは除外されます) 。

ノードグループの一覧表示

ノードグループまたはすべてのノードグループの詳細を一覧表示するには、以下を使用します。

```
eksctl get nodegroup --cluster=<clusterName> [--name=<nodegroupName>]
```

デフォルトのログテーブルよりも多くの情報を出力する YAML または JSON 形式で 1 つ以上のノードグループを一覧表示するには、以下を使用します。

```
# YAML format
eksctl get nodegroup --cluster=<clusterName> [--name=<nodegroupName>] --output=yaml

# JSON format
eksctl get nodegroup --cluster=<clusterName> [--name=<nodegroupName>] --output=json
```

ノードグループのイミュータビリティ

設計上、ノードグループはイミュータブルです。つまり、AMI やノードグループのインスタンスタイプなどの何かを変更 (スケーリング以外) する必要がある場合は、必要な変更を加えて新しいノードグループを作成し、ロードを移動して古いノードグループを削除する必要があります。「ノードグループの削除とドレイン」セクションを参照してください。

ノードグループのスケーリング

Nodegroup スケーリングは、最大数分かかるプロセスです。`--wait` フラグが指定されていない場合、 はノードグループのスケーリングを`eksctl`楽観的に想定し、AWS API リクエストが送信されるとすぐに を返します。ノードが使用可能になるまで`eksctl`待機するには、以下の例のような`--wait`フラグを追加します。

> **Note**
>
> ノードグループをダウン/イン (ノード数を減らす) にスケールすると、ASG への純粋な変更に依存するため、エラーが発生する可能性があります。つまり、削除/終了されるノード (複数可) は明示的にドレインされません。これは将来改善の余地がある分野かもしれません。

マネージド型ノードグループのスケーリングは、マネージド型ノードグループ設定を更新する EKS API を直接呼び出すことで実現されます。

単一のノードグループのスケーリング

ノードグループは、 `eksctl scale nodegroup` コマンドを使用してスケーリングできます。

```
eksctl scale nodegroup --cluster=<clusterName> --nodes=<desiredCount> --
name=<nodegroupName> [ --nodes-min=<minSize> ] [ --nodes-max=<maxSize> ] --wait
```

たとえば、ノードグループ`cluster-1`を 5 つのノード`ng-a345f4e1`にスケールするには、以下を実行します。

```
eksctl scale nodegroup --cluster=cluster-1 --nodes=5 ng-a345f4e1
```

ノードグループは、 に渡される設定ファイルを使用して、 `--config-file` でスケールするノードグループの名前を指定することでスケールすることもできます`--name`。Eksctl は設定ファイルを検索し、そのノードグループとそのスケーリング設定値を検出します。

必要なノード数が現在の最小ノード数と現在の最大ノード数の範囲内NOTにある場合、1 つの特定のエラーが表示されます。これらの値は、`--nodes-max`それぞれフラグ `--nodes-min`と で渡すこともできます。

複数のノードグループのスケーリング

Eksctl は、 で渡される設定ファイルにあるすべてのノードグループを検出してスケーリングできます`--config-file`。

単一のノードグループのスケーリングと同様に、同じ検証セットが各ノードグループに適用されます。たとえば、必要なノード数は、ノードの最小数と最大数の範囲内である必要があります。

ノードグループの削除とドレイン

ノードグループを削除するには、以下を実行します。

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

包含ルールと除外ルールは、このコマンドでも使用できます。

> **Note**
>
> これにより、インスタンスが削除される前に、そのノードグループからすべてのポッドがドレインされます。

ドレインプロセス中にエビクションルールをスキップするには、以下を実行します。

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

すべてのノードは接続され、すべてのポッドは削除時にノードグループから削除されますが、ノードグループを削除せずにドレインする必要がある場合は、以下を実行します。

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

ノードグループのコードを解除するには、以下を実行します。

```
eksctl drain nodegroup --cluster=<clusterName> --name=<nodegroupName> --undo
```

PodDisruptionBudget 設定などの削除ルールを無視するには、以下を実行します。

```
eksctl drain nodegroup --cluster=<clusterName> --name=<nodegroupName> --disable-
eviction
```

ドレインプロセスを高速化するには、並列でドレインするノードの数`--parallel <value>`に を指定できます。

その他の機能

ノードグループの SSH、ASG アクセス、およびその他の機能を有効にすることもできます。次に例を示します。

```
eksctl create nodegroup --cluster=cluster-1 --node-
labels="autoscaling=enabled,purpose=ci-worker" --asg-access --full-ecr-access --ssh-
access
```

ラベルを更新する

ノードグループのラベルeksctlを更新する特定のコマンドは にはありませんが、 kubectlを使用して簡単に実行できます。例:

```
kubectl label nodes -l alpha.eksctl.io/nodegroup-name=ng-1 new-label=foo
```

SSH アクセス

ノードグループの SSH アクセスを有効にするには、ノードグループ設定publicKeyPathで publicKeyName、 publicKeyのいずれかを設定します。または、 でノードグループを設定することで、AWS Systems Manager (SSM) を使用してノードに SSH 接続することもできますenableSsm。

```
managedNodeGroups:
  - name: ng-1
    instanceType: m5.large
    desiredCapacity: 1
    ssh: # import public key from file
      publicKeyPath: ~/.ssh/id_rsa_tests.pub
  - name: ng-2
    instanceType: m5.large
    desiredCapacity: 1
    ssh: # use existing EC2 key
      publicKeyName: ec2_dev_key
  - name: ng-3
    instanceType: m5.large
    desiredCapacity: 1
    ssh: # import inline public key
      publicKey: "ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAABAQDqZEdzvHnK/GVP8nLngRHu/
GDi/3PeES7+Bx6l3koXn/Oi/UmM9/jcW5XGziZ/
oe1cPJ777eZV7muEvXg5ZMQBrYxUtYCdvd8Rt6DIoSqDLsIPqbuuNlQoBHq/PU2IjpWnp/
```

```
wrJQXMk94IIrGjY8QHfCnpuMENCucVaifgAhwyeyuO5KiqUmD8E0RmcsotHKBV9X8H5eqLXd8zMQaPl
+Ub7j5PG+9KftQu0F/QhdFvpSLsHaxvBzA5nhIltjkaFcwGQnD1rpCM3+UnQE7Izoa5Yt1xoUWRwnF
+L2TKovW7+bYQ1kxsuuiX149jXTCJDVjkYCqi7HkrXYqcC1sbsror someuser@hostname"
  - name: ng-4
    instanceType: m5.large
    desiredCapacity: 1
    ssh: # enable SSH using SSM
      enableSsm: true
```

アンマネージド型ノードグループ

でeksctl、 nodeGroupsフィールドを設定--managed=falseまたは使用すると、アンマネージド型ノードグループが作成されます。アンマネージド型ノードグループは EKS コンソールに表示されません。これは、原則として EKS マネージド型ノードグループについてのみ認識されます。

を実行した後でのみノードグループをアップグレードする必要がありますeksctl upgrade cluster。(「クラスターのアップグレード」を参照してください。)

初期ノードグループのみを持つシンプルなクラスター (で作成eksctl create cluster) がある場合、プロセスは非常に簡単です。

1. 古いノードグループの名前を取得します。

```
eksctl get nodegroups --cluster=<clusterName> --region=<region>
```

> **Note**
>
> ```
> You should see only one nodegroup here, if you see more - read the next
>  section.
> ```

2. 新しいノードグループを作成します。

```
 eksctl create nodegroup --cluster=<clusterName> --region=<region> --
name=<newNodeGroupName> --managed=false
```

3. 古いノードグループを削除します。

```
 eksctl delete nodegroup --cluster=<clusterName> --region=<region> --
name=<oldNodeGroupName>
```

> **Note**
>
> ```
> This will drain all pods from that nodegroup before the instances are deleted.
>  In some scenarios, Pod Disruption Budget (PDB) policies can prevent pods to
>  be evicted. To delete the nodegroup regardless of PDB, one should use the `--
> disable-eviction` flag, will bypass checking PDB policies.
> ```

複数のノードグループの更新

複数のノードグループがある場合は、それぞれの設定方法を追跡する責任があります。これを行うには、設定ファイルを使用しますが、まだ使用していない場合は、クラスターを調べて各ノードグループの設定を確認する必要があります。

一般的に、次のことを検討しています。

- 所有しているノードグループと、新しいバージョンで削除または置き換えることができるノードグループを確認する
- 各ノードグループの設定をメモし、次回アップグレードを容易にするために設定ファイルの使用を検討してください

設定ファイルを使用した更新

設定ファイルを使用している場合は、以下を実行する必要があります。

設定ファイルを編集して新しいノードグループを追加し、古いノードグループを削除します。ノードグループをアップグレードして同じ設定を維持するだけの場合は、名前-v2に を追加するなど、ノードグループ名を変更することができます。

設定ファイルで定義されているすべての新しいノードグループを作成するには、以下を実行します。

```
eksctl create nodegroup --config-file=<path>
```

新しいノードグループを設定したら、古いノードグループを削除できます。

```
eksctl delete nodegroup --config-file=<path> --only-missing
```

> **Note**
>
> 最初の実行はプランモードです。提案された変更に満足している場合は、 で再実行します`--approve`。

デフォルトのアドオンの更新

クラスターにインストールされているネットワークアドオンを更新する必要がある場合があります。詳細については、「the section called “デフォルトのアドオンの更新”」を参照してください。

EKS マネージド型ノードグループ

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 (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する機能です。お客様はクラスター用に最適化されたノードグループをプロビジョニングでき、EKS はノードを最新の Kubernetes およびホスト OS バージョンで最新の状態に保ちます。

EKS マネージド型ノードグループは、Amazon EKS クラスターの AWS によって管理される自動スケーリンググループおよび関連する EC2 インスタンスです。各ノードグループは、Amazon EKS 最適化 Amazon Linux 2 AMI を使用します。Amazon EKS を使用すると、ノードにバグ修正とセキュリティパッチを簡単に適用し、最新の Kubernetes バージョンに更新できます。各ノードグループは、クラスターの Auto Scaling グループを起動します。このグループは、高可用性のために複数の AWS VPC アベイラビリティーゾーンとサブネットにまたがることができます。

マネージド型ノードグループの新しい起動テンプレートのサポート ???

> **Note**
>
> 「アンマネージド型ノードグループ」という用語は、eksctl が最初からサポートしているノードグループを参照するために使用されています (`nodeGroups`フィールドで表されます) 。`ClusterConfig` ファイルは引き続き `nodeGroups`フィールドを使用してアンマネージド型ノードグループを定義し、マネージド型ノードグループは `managedNodeGroups`フィールドで定義されます。

マネージド型ノードグループの作成

```
$ eksctl create nodegroup
```

新しいクラスター

マネージド型ノードグループを使用して新しいクラスターを作成するには、 を実行します。

```
eksctl create cluster
```

複数のマネージド型ノードグループを作成し、設定をより詳細に制御するには、設定ファイルを使用できます。

> **Note**
>
> マネージド型ノードグループには、アンマネージド型ノードグループとの完全な機能パリティはありません。

```
# cluster.yaml
# A cluster with two managed nodegroups
---
apiVersion: eksctl.io/v1alpha5
kind: ClusterConfig

metadata:
  name: managed-cluster
  region: us-west-2

managedNodeGroups:
  - name: managed-ng-1
    minSize: 2
    maxSize: 4
    desiredCapacity: 3
    volumeSize: 20
    ssh:
      allow: true
      publicKeyPath: ~/.ssh/ec2_id_rsa.pub
      # new feature for restricting SSH access to certain AWS security group IDs
      sourceSecurityGroupIds: ["sg-00241fbb12c607007"]
    labels: {role: worker}
```

```
    tags:
      nodegroup-role: worker
    iam:
      withAddonPolicies:
        externalDNS: true
        certManager: true

  - name: managed-ng-2
    instanceType: t2.large
    minSize: 2
    maxSize: 3
```

マネージド型ノードグループを作成するための設定ファイルの別の例を以下に示します。

マネージド型ノードグループとアンマネージド型ノードグループの両方を持つクラスターを持つことができます。アンマネージド型ノードグループは AWS EKS コンソールには表示されませんが、両方のタイプのノードグループが一覧表示eksctl get nodegroupされます。

```
# cluster.yaml
# A cluster with an unmanaged nodegroup and two managed nodegroups.
---
apiVersion: eksctl.io/v1alpha5
kind: ClusterConfig

metadata:
  name: managed-cluster
  region: us-west-2

nodeGroups:
  - name: ng-1
    minSize: 2

managedNodeGroups:
  - name: managed-ng-1
    minSize: 2
    maxSize: 4
    desiredCapacity: 3
    volumeSize: 20
    ssh:
      allow: true
      publicKeyPath: ~/.ssh/ec2_id_rsa.pub
      # new feature for restricting SSH access to certain AWS security group IDs
      sourceSecurityGroupIds: ["sg-00241fbb12c607007"]
```

```
    labels: {role: worker}
    tags:
      nodegroup-role: worker
    iam:
      withAddonPolicies:
        externalDNS: true
        certManager: true

  - name: managed-ng-2
    instanceType: t2.large
    privateNetworking: true
    minSize: 2
    maxSize: 3
```

カスタム AMI、セキュリティグループ、instancePrefix、、、instanceNameebsOptimized、volumeType、volumeName、volumeEnc の新しいサポート disableIMDSv1

```
# cluster.yaml
# A cluster with a managed nodegroup with customization.
---
apiVersion: eksctl.io/v1alpha5
kind: ClusterConfig

metadata:
  name: managed-cluster
  region: us-west-2

managedNodeGroups:
  - name: custom-ng
    ami: ami-0e124de4755b2734d
    securityGroups:
      attachIDs: ["sg-1234"]
    maxPodsPerNode: 80
    ssh:
      allow: true
    volumeSize: 100
    volumeName: /dev/xvda
    volumeEncrypted: true
    # defaults to true, which enforces the use of IMDSv2 tokens
    disableIMDSv1: false
    overrideBootstrapCommand: |
      #!/bin/bash
```

```
      /etc/eks/bootstrap.sh managed-cluster --kubelet-extra-args '--node-
labels=eks.amazonaws.com/nodegroup=custom-ng,eks.amazonaws.com/nodegroup-
image=ami-0e124de4755b2734d'
```

1 つのゾーンでのみ使用可能なインスタンスタイプをリクエストする場合 (eksctl 設定には 2 つの仕様が必要です)、ノードグループリクエストにアベイラビリティーゾーンを追加してください。

```
# cluster.yaml
# A cluster with a managed nodegroup with "availabilityZones"
---

apiVersion: eksctl.io/v1alpha5
kind: ClusterConfig

metadata:
  name: flux-cluster
  region: us-east-2
  version: "1.23"

availabilityZones: ["us-east-2b", "us-east-2c"]
managedNodeGroups:
  - name: workers
    instanceType: hpc6a.48xlarge
    minSize: 64
    maxSize: 64
    labels: { "fluxoperator": "true" }
    availabilityZones: ["us-east-2b"]
    efaEnabled: true
    placement:
      groupName: eks-efa-testing
```

これは、1 つのゾーンでのみ利用可能な Hpc6 ファミリーなどのインスタンスタイプに当てはまります。

既存のクラスター

```
eksctl create nodegroup --managed
```

ヒント: `ClusterConfig` ファイルを使用してクラスター全体を記述する場合は、`managedNodeGroups`フィールドに新しいマネージドノードグループを記述し、以下を実行します。

```
eksctl create nodegroup --config-file=YOUR_CLUSTER.yaml
```

マネージド型ノードグループのアップグレード

ノードグループは、使用している AMI タイプの最新の EKS 最適化 AMI リリースバージョンにいつでも更新できます。

ノードグループがクラスターと同じ Kubernetes バージョンである場合は、使用している AMI タイプのその Kubernetes バージョンの最新の AMI リリースバージョンに更新できます。ノードグループがクラスターの Kubernetes バージョンから以前の Kubernetes バージョンである場合は、ノードグループをノードグループの Kubernetes バージョンに一致する最新の AMI リリースバージョンに更新するか、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に更新できます。ノードグループを以前の Kubernetes バージョンにロールバックすることはできません。

マネージド型ノードグループを最新の AMI リリースバージョンにアップグレードするには:

```
eksctl upgrade nodegroup --name=managed-ng-1 --cluster=managed-cluster
```

ノードグループは、以下を使用して、指定された Kubernetes バージョンの最新の AMI リリースにアップグレードできます。

```
eksctl upgrade nodegroup --name=managed-ng-1 --cluster=managed-cluster --kubernetes-
version=<kubernetes-version>
```

最新バージョンではなく特定の AMI リリースバージョンにアップグレードするには、 を渡します`--release-version`。

```
eksctl upgrade nodegroup --name=managed-ng-1 --cluster=managed-cluster --release-
version=1.19.6-20210310
```

> **Note**
>
> マネージドノードがカスタム AMIs を使用してデプロイされる場合、カスタム AMI の新しいバージョンをデプロイするには、次のワークフローに従う必要があります。

- ノードグループの初期デプロイは、起動テンプレートを使用して実行する必要があります。例:

```
managedNodeGroups:
  - name: launch-template-ng
    launchTemplate:
      id: lt-1234
      version: "2" #optional (uses the default version of the launch template if
 unspecified)
```

- カスタム AMI の新しいバージョンを作成します (AWS EKS コンソールを使用) 。
- 新しい AMI ID を使用して新しい起動テンプレートバージョンを作成します (AWS EKS コンソールを使用) 。
- ノードを起動テンプレートの新しいバージョンにアップグレードします。例:

```
eksctl upgrade nodegroup --name nodegroup-name --cluster cluster-name --launch-
template-version new-template-version
```

ノードの並列アップグレードの処理

複数のマネージドノードを同時にアップグレードできます。並列アップグレードを設定するには、updateConfigノードグループの作成時にノードグループの を定義します。例についてはupdateConfig、こちらを参照してください。

一度に複数のノードをアップグレードすることによるワークロードのダウンタイムを回避するには、 の maxUnavailableフィールドにこれを指定することで、アップグレード中に使用できなくなるノードの数を制限できますupdateConfig。または、 を使用します。これはmaxUnavailablePercentage、使用できないノードの最大数をノードの合計数に対する割合として定義します。

を より大きくmaxUnavailableすることはできませんmaxSize。また、 maxUnavailableとmaxUnavailablePercentage を同時に使用することはできません。

この機能はマネージドノードでのみ使用できます。

マネージド型ノードグループの更新

eksctl では、マネージド型ノードグループの UpdateConfig セクションを更新できます。このセクションでは、 MaxUnavailableと の 2 つのフィールドを定義しますMaxUnavailablePercentage。更新中、ノードグループは影響を受けないため、ダウンタイムは予想されません。

コマンドは、 `--config-file`フラグを使用する設定ファイルとともに使用`update nodegroup`する必要があります。ノードグループには `nodeGroup.updateConfig`セクションが含まれている必要があります。詳細については、こちらを参照してください。

Nodegroup のヘルスの問題

EKS Managed Nodegroups は、ノードグループとノードの正常性の問題を自動的にチェックし、EKS API とコンソールを通じて報告します。ノードグループのヘルス問題を表示するには:

```
eksctl utils nodegroup-health --name=managed-ng-1 --cluster=managed-cluster
```

ラベルの管理

EKS Managed Nodegroups は、ノードグループの Kubernetes ノードに適用されるラベルのアタッチをサポートしています。これは、クラスターまたはノードグループの作成時に eksctl の `labels`フィールドを介して指定されます。

ノードグループで新しいラベルを設定したり、既存のラベルを更新したりするには:

```
eksctl set labels --cluster managed-cluster --nodegroup managed-ng-1 --labels
 kubernetes.io/managed-by=eks,kubernetes.io/role=worker
```

ノードグループからラベルを設定解除または削除するには:

```
eksctl unset labels --cluster managed-cluster --nodegroup managed-ng-1 --labels
 kubernetes.io/managed-by,kubernetes.io/role
```

ノードグループに設定されたすべてのラベルを表示するには:

```
eksctl get labels --cluster managed-cluster --nodegroup managed-ng-1
```

マネージド型ノードグループのスケーリング

`eksctl scale nodegroup` はマネージド型ノードグループもサポートしています。マネージド型ノードグループまたはアンマネージド型ノードグループをスケーリングする構文は同じです。

```
eksctl scale nodegroup --name=managed-ng-1 --cluster=managed-cluster --nodes=4 --nodes-
min=3 --nodes-max=5
```

詳細情報

- EKS マネージド型ノードグループ

ノードブートストラップ

AmazonLinux2023

AL2023 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 を導入し、/etc/eks/bootstrap.shスクリプトの使用を削除しました。

> **Note**
>
> Kubernetes バージョン 1.30 以降では、Amazon Linux 2023 がデフォルトの OS です。

AL2 のデフォルト設定

カスタム AMIs、 eksctlはデフォルト、最小、NodeConfigおよびノードグループの起動テンプレートのユーザーデータに自動的に挿入します。つまり、

```
MIME-Version: 1.0
Content-Type: multipart/mixed; boundary=//

--//
Content-Type: application/node.eks.aws

apiVersion: node.eks.aws/v1alpha1
kind: NodeConfig
spec:
  cluster:
    apiServerEndpoint: https://XXXX.us-west-2.eks.amazonaws.com
    certificateAuthority: XXXX
    cidr: 10.100.0.0/16
    name: my-cluster
  kubelet:
    config:
      clusterDNS:
      - 10.100.0.10
    flags:
```

```
    - --node-labels=alpha.eksctl.io/cluster-name=my-cluster,alpha.eksctl.io/nodegroup-
name=my-nodegroup
    - --register-with-taints=special=true:NoSchedule

--//--
```

ネイティブ AMIs に基づく EKS マネージドノードの場合、デフォルトNodeConfigは EKS MNG によって内部に追加され、EC2 のユーザーデータに直接追加されます。したがって、このシナリオでは、 eksctl は起動テンプレートに含める必要はありません。

ブートストラッププロセスの設定

の高度なプロパティを設定したりNodeConfig、単にデフォルト値を上書きしたりするために、eksctl では nodeGroup.overrideBootstrapCommandや managedNodeGroup.overrideBootstrapCommand などNodeConfigを使用してカスタムを指定できます。

```
managedNodeGroups:
  - name: mng-1
    amiFamily: AmazonLinux2023
    ami: ami-0253856dd7ab7dbc8
    overrideBootstrapCommand: |
      apiVersion: node.eks.aws/v1alpha1
      kind: NodeConfig
      spec:
        instance:
          localStorage:
            strategy: RAID0
```

このカスタム設定は、eksctl によってユーザーデータに付加され、 によってデフォルトの設定nodeadmとマージされます。複数の設定オブジェクトをマージする nodeadmの機能の詳細については、こちらを参照してください。

Managed Nodegroups の起動テンプレートのサポート

eksctl は、提供された EC2 起動テンプレートを使用したマネージド型ノードグループの起動をサポートしています。これにより、カスタム AMIs とセキュリティグループの提供、ノードブートストラップ用のユーザーデータの受け渡しなど、ノードグループの複数のカスタマイズオプションが可能になります。

提供された起動テンプレートを使用したマネージド型ノードグループの作成

```
# managed-cluster.yaml
# A cluster with two managed nodegroups
---
apiVersion: eksctl.io/v1alpha5
kind: ClusterConfig

metadata:
  name: managed-cluster
  region: us-west-2

managedNodeGroups:
  - name: managed-ng-1
    launchTemplate:
      id: lt-12345
      version: "2" # optional (uses the default launch template version if unspecified)

  - name: managed-ng-2
    minSize: 2
    desiredCapacity: 2
    maxSize: 4
    labels:
      role: worker
    tags:
      nodegroup-name: managed-ng-2
    privateNetworking: true
    launchTemplate:
      id: lt-12345
```

別の起動テンプレートバージョンを使用するようにマネージド型ノードグループをアップグレードする

```
eksctl upgrade nodegroup --name=managed-ng-1 --cluster=managed-cluster --launch-
template-version=3
```

> **Note**
>
> 起動テンプレートがカスタム AMI を使用している場合、新しいバージョンもカスタム AMI を使用する必要があります。そうしないと、アップグレードオペレーションは失敗します。

起動テンプレートがカスタム AMI を使用していない場合は、アップグレード先の Kubernetes バージョンも指定できます。

```
eksctl upgrade nodegroup --name=managed-ng-1 --cluster=managed-cluster --launch-
template-version=3 --kubernetes-version=1.17
```

カスタム AMI と起動テンプレートのサポートに関する注意事項

- 起動テンプレートが指定されている場合、次のフィールドはサポートされていません: `instanceType`、、`ami`、`ssh.allowssh.sourceSecurityGroupIds`、、`securityGroups`、`insta` `overrideBootstrapCommand` `disableIMDSv1`
- カスタム AMI (`ami`) を使用する場合は、ブートストラップを実行するように も設定`overrideBootstrapCommand`する必要があります。
- `overrideBootstrapCommand` は、カスタム AMI を使用する場合にのみ設定できます。
- 起動テンプレートを指定すると、ノードグループ設定で指定されたタグは EKS Nodegroup リソースにのみ適用され、EC2 インスタンスには伝達されません。

カスタムサブネット

既存の VPC を新しいサブネットで拡張し、そのサブネットに Nodegroup を追加できます。

その理由

クラスターで事前設定された IPs が不足した場合、新しい CIDR を使用して既存の VPC のサイズを変更して、新しいサブネットを追加できます。これを行う方法については、AWS VPCs の拡張に関するこのガイドを参照してください。

TL;DR

VPC の設定に移動し、Actions->Edit CIDRs をクリックして新しい範囲を追加します。例えば、次のようになります。

```
192.168.0.0/19 -> existing CIDR
+ 192.169.0.0/19 -> new CIDR
```

次に、新しいサブネットを追加する必要があります。新しいプライベートサブネットかパブリックサブネットかに応じて、それぞれプライベートサブネットまたはパブリックサブネットからルーティング情報をコピーする必要があります。

サブネットが作成されたら、ルーティングを追加し、VPC 内の別のサブネットから NAT ゲートウェイ ID またはインターネットゲートウェイをコピーします。パブリックサブネットの場合は、自動 IP 割り当てを有効にするように注意してください。Actions->Modify auto-assign IP settings->Enable auto-assign public IPv4 address。

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設定に応じて、既存のサブネットの TAGS もコピーすることを忘れないでください。これは重要です。そうしないと、サブネットはクラスターの一部ではなく、サブネット内のインスタンスは参加できなくなります。

完了したら、新しいサブネットの ID をコピーします。必要に応じて繰り返します。

その方法は?

作成したサブネットにノードグループを作成するには (複数可) 、次のコマンドを実行します。

```
eksctl create nodegroup --cluster <cluster-name> --name my-new-subnet --subnet-ids
 subnet-0edeb3a04bec27141,subnet-0edeb3a04bec27142,subnet-0edeb3a04bec27143
# or for a single subnet id
eksctl create nodegroup --cluster <cluster-name> --name my-new-subnet --subnet-ids
 subnet-0edeb3a04bec27141
```

または、次のように設定を使用します。

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

このような設定では、次のようになります。

```
# A simple example of ClusterConfig object with two nodegroups:
---
apiVersion: eksctl.io/v1alpha5
kind: ClusterConfig

metadata:
```

```
  name: cluster-3
  region: eu-north-1

nodeGroups:
  - name: new-subnet-nodegroup
    instanceType: m5.large
    desiredCapacity: 1
    subnets:
      - subnet-id1
      - subnet-id2
```

ノードグループが作成され、新しいインスタンスにサブネットの新しい IP 範囲 (複数可) が設定されるまで待ちます。

クラスターの削除

新しい追加により、CloudFormation スタックの外部に依存関係を追加して既存の VPC が変更されたため、CloudFormation はクラスターを削除できなくなります。

クラスターを削除する前に、作成した追加のサブネットをすべて手動で削除し、 を呼び出して続行しますeksctl。

```
eksctl delete cluster -n <cluster-name> --wait
```

カスタム DNS

すべての内部および外部 DNS ルックアップに使用される DNS サーバーの IP アドレスを上書きするには、2 つの方法があります。これは、 の --cluster-dnsフラグと同等ですkubelet。

1 つ目は、 clusterDNSフィールド経由です。Config ファイルは、使用する DNS サーバーの IP アドレスclusterDNSを持つ というstringフィールドを受け入れます。これは に渡され、 kubelet ファイルを介してポッドに渡されます/etc/resolv.conf。詳細については、設定ファイルのスキーマを参照してください。

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

metadata:
  name: cluster-1
  region: eu-north-1
```

```
nodeGroups:
- name: ng-1
  clusterDNS: 169.254.20.10
```

この設定は 1 つの IP アドレスのみを受け入れることに注意してください。複数のアドレスを指定するには、 kubeletExtraConfigパラメータを使用します。

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

metadata:
  name: cluster-1
  region: eu-north-1

nodeGroups:
  - name: ng-1
    kubeletExtraConfig:
      clusterDNS: ["169.254.20.10","172.20.0.10"]
```

テイント

テイントを特定のノードグループに適用するには、次のように taints config セクションを使用します。

```
    taints:
      - key: your.domain.com/db
        value: "true"
        effect: NoSchedule
      - key: your.domain.com/production
        value: "true"
        effect: NoExecute
```

完全な例については、こちらを参照してください。

インスタンスセレクタ

eksctl 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グループに複数のインスタンスタイプの指定をサポートしていますが、270 を超える EC2 インスタンスタイプでは、ユーザーはノードグループに適したインスタンスタイプを見つけるために時間を費やす必要がありま

す。Cluster Autoscaler と連携する一連のインスタンスを選択する必要があるため、スポットインスタンスを使用するとさらに難しくなります。

eksctl は EC2 インスタンスセレクタと統合されるようになりました。これにより、vCPUs、メモリ、GPUsCPU アーキテクチャなどのリソース基準に基づいてインスタンスタイプのリストを生成することで、この問題に対処できます。インスタンスセレクタ条件に合格すると、eksctl は指定された条件に一致するインスタンスタイプに設定されたインスタンスタイプを持つノードグループを作成します。

クラスターとノードグループを作成する

eksctl に渡されたインスタンスセレクタリソース条件に一致するインスタンスタイプを使用する単一のノードグループを持つクラスターを作成するには、 を実行します。

```
eksctl create cluster --instance-selector-vcpus=2 --instance-selector-memory=4
```

これにより、 instanceTypesフィールドが に設定されているクラスターとマネージド型ノードグループが作成されます [c5.large, c5a.large, c5ad.large, c5d.large, t2.medium, t3.medium, t3a.medium] (返されるインスタンスタイプのセットは変更される可能性があります)。

アンマネージド型ノードグループの場合、 instancesDistribution.instanceTypesフィールドが設定されます。

```
eksctl create cluster --managed=false --instance-selector-vcpus=2 --instance-selector-
memory=4
```

インスタンスセレクタ条件は ClusterConfig で指定することもできます。

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

metadata:
  name: cluster
  region: us-west-2

nodeGroups:
- name: ng
  instanceSelector:
```

```
    vCPUs: 2
    memory: "4" # 4 GiB, unit defaults to GiB

managedNodeGroups:
- name: mng
  instanceSelector:
    vCPUs: 2
    memory: 2GiB #
    cpuArchitecture: x86_64 # default value
```

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

次のインスタンスセレクタ CLI オプションは、 eksctl create clusterおよび でサポートされていますeksctl create nodegroup。

--instance-selector-vcpus、--instance-selector-memory、--instance-selector-gpus および instance-selector-cpu-architecture

ファイルの例については、こちらを参照してください。

ドライラン

ドライラン機能を使用すると、ノードグループの作成に進む前に、インスタンスセレクタに一致するインスタンスを検査および変更できます。

```
eksctl create cluster --name development --instance-selector-vcpus=2 --instance-
selector-memory=4 --dry-run

apiVersion: eksctl.io/v1alpha5
kind: ClusterConfig
# ...
managedNodeGroups:
- amiFamily: AmazonLinux2
  instanceSelector:
    memory: "4"
    vCPUs: 2
  instanceTypes:
  - c5.large
  - c5a.large
  - c5ad.large
  - c5d.large
  - t2.medium
```

```
  - t3.medium
  - t3a.medium
...
# other config
```

その後、生成された ClusterConfig を に渡すことができますeksctl create cluster。

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

CLI オプションを表す instanceSelectorフィールドは、可視性とドキュメント化の目的で ClusterConfig ファイルにも追加されます。--dry-run を省略すると、このフィールドは無視され、 instanceTypesフィールドが使用されます。それ以外の場合、 への変更は eksctl によって上書きinstanceTypesされます。

ClusterConfig ファイルを で渡すと--dry-run、eksctl は各ノードグループのインスタンスセレクタリソース条件を展開した後、同じノードグループのセットを含む ClusterConfig ファイルを出力します。

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

metadata:
  name: cluster
  region: us-west-2

nodeGroups:
- name: ng
  instanceSelector:
    vCPUs: 2
    memory: 4 # 4 GiB, unit defaults to GiB

managedNodeGroups:
- name: mng
  instanceSelector:
    vCPUs: 2
    memory: 2GiB #
    cpuArchitecture: x86_64 # default value
```

```
eksctl create cluster -f instance-selector-cluster.yaml --dry-run
```

```
apiVersion: eksctl.io/v1alpha5
kind: ClusterConfig
# ...
managedNodeGroups:
- amiFamily: AmazonLinux2
  # ...
  instanceSelector:
    cpuArchitecture: x86_64
    memory: 2GiB
    vCPUs: 2
  instanceTypes:
  - t3.small
  - t3a.small
nodeGroups:
- amiFamily: AmazonLinux2
  # ...
  instanceSelector:
    memory: "4"
    vCPUs: 2
  instanceType: mixed
  instancesDistribution:
    capacityRebalance: false
    instanceTypes:
    - c5.large
    - c5a.large
    - c5ad.large
    - c5d.large
    - t2.medium
    - t3.medium
    - t3a.medium
# ...
```

スポットインスタンス

マネージド型ノードグループ

eksctl は、EKS Managed Nodegroups を使用してスポットワーカーノードをサポートします。これは、耐障害性アプリケーションを持つ EKS のお客様が EKS クラスターの EC2 スポットインスタンスを簡単にプロビジョニングおよび管理できるようにする機能です。EKS Managed Nodegroup は、スポットのベストプラクティスに従ってスポットインスタンスの EC2 Autoscaling グループを設定して起動し、インスタンスが AWS によって中断される前にスポットワーカーノードを自動的にド

レインします。この機能を使用するための増分料金はかかりません。また、お客様は EC2 スポットインスタンスや EBS ボリュームなどの AWS リソースの使用に対してのみ料金が発生します。

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持つクラスターを作成するには、--spotフラグとインスタンスタイプのオプションリストを渡します。

```
eksctl create cluster --spot --instance-types=c3.large,c4.large,c5.large
```

既存のクラスターで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作成するには:

```
eksctl create nodegroup --cluster=<clusterName> --spot --instance-
types=c3.large,c4.large,c5.large
```

設定ファイルを介してマネージド型ノードグループを使用してスポットインスタンスを作成するには:

```
# spot-cluster.yaml

apiVersion: eksctl.io/v1alpha5
kind: ClusterConfig

metadata:
  name: spot-cluster
  region: us-west-2

managedNodeGroups:
- name: spot
  instanceTypes: ["c3.large","c4.large","c5.large","c5d.large","c5n.large","c5a.large"]
  spot: true


# `instanceTypes` defaults to [`m5.large`]
- name: spot-2
  spot: true

# On-Demand instances
- name: on-demand
  instanceTypes: ["c3.large", "c4.large", "c5.large"]
```

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

> **Note**
>
> アンマネージド型ノードグループは spotおよび instanceTypesフィールドをサポートしておらず、代わりに instancesDistributionフィールドを使用してスポットインスタンスを設定します。以下を参照

詳細情報

- EKS スポットノードグループ
- EKS マネージド型ノードグループのキャパシティタイプ

アンマネージド型ノードグループ

eksctl は、Auto Scaling グループの MixedInstancesPolicy を通じてスポットインスタンスをサポートしています。

50% のスポットインスタンスと 50% のオンデマンドインスタンスを使用するノードグループの例を次に示します。

```
nodeGroups:
  - name: ng-1
    minSize: 2
    maxSize: 5
    instancesDistribution:
      maxPrice: 0.017
      instanceTypes: ["t3.small", "t3.medium"] # At least one instance type should be
 specified
      onDemandBaseCapacity: 0
      onDemandPercentageAboveBaseCapacity: 50
      spotInstancePools: 2
```

nodeGroups.X.instanceType フィールドを使用する場合は、instancesDistributionフィールドを設定しないでください。

この例では、GPU インスタンスを使用します。

```
nodeGroups:
  - name: ng-gpu
```

```
    instanceType: mixed
    desiredCapacity: 1
    instancesDistribution:
      instanceTypes:
        - p2.xlarge
        - p2.8xlarge
        - p2.16xlarge
      maxPrice: 0.50
```

この例では、容量最適化スポット配分戦略を使用します。

```
nodeGroups:
  - name: ng-capacity-optimized
    minSize: 2
    maxSize: 5
    instancesDistribution:
      maxPrice: 0.017
      instanceTypes: ["t3.small", "t3.medium"] # At least one instance type should be
 specified
      onDemandBaseCapacity: 0
      onDemandPercentageAboveBaseCapacity: 50
      spotAllocationStrategy: "capacity-optimized"
```

この例では、capacity-optimized-prioritizedスポット配分戦略を使用しています。

```
nodeGroups:
  - name: ng-capacity-optimized-prioritized
    minSize: 2
    maxSize: 5
    instancesDistribution:
      maxPrice: 0.017
      instanceTypes: ["t3a.small", "t3.small"] # At least two instance types should be
 specified
      onDemandBaseCapacity: 0
      onDemandPercentageAboveBaseCapacity: 0
      spotAllocationStrategy: "capacity-optimized-prioritized"
```

capacity-optimized-prioritized 配分戦略を使用して、起動テンプレートのオーバーライドのリストのインスタンスタイプの順序を、優先度が最も高いものから低いもの (リストの最初から最後のもの) に設定します。Amazon EC2 Auto Scaling は、ベストエフォートベースでインスタンスタイプの優先順位を尊重しますが、最初に容量を最適化します。これは、中断の可能性を最小限に抑え

る必要があるワークロードに適していますが、特定のインスタンスタイプの設定も重要です。詳細については、「ASG 購入オプション」を参照してください。

spotInstancePools フィールドを使用する場合は、 spotAllocationStrategyフィールドを設定しないでください。が指定されspotAllocationStrategyていない場合、EC2 はデフォルトで lowest-price戦略を使用します。

最小限の例を次に示します。

```
nodeGroups:
  - name: ng-1
    instancesDistribution:
      instanceTypes: ["t3.small", "t3.medium"] # At least one instance type should be
 specified
```

スポットインスタンスとオンデマンドインスタンス間でノードを区別するには、kubernetes ラベルを使用します。node-lifecycleこのラベルには spotまたは on-demand の値があります。

instancesDistribution のパラメータ

詳細については、クラスター設定スキーマを参照してください。

GPU サポート

Eksctl は、ノードグループの GPU インスタンスタイプの選択をサポートしています。互換性のあるインスタンスタイプを create コマンドに指定するか、 設定ファイルを介して指定します。

```
eksctl create cluster --node-type=p2.xlarge
```

> **Note**
>
> EKS での GPU サポートのために Marketplace AMI にサブスクライブする必要がなくなりました。

AMI リゾルバー (auto および auto-ssm) は、GPU インスタンスタイプを使用することを確認し、適切な EKS 最適化高速 AMI を選択します。

Eksctl は、GPU 対応インスタンスタイプの AMI が選択されていることを検出し、NVIDIA Kubernetes デバイスプラグインを自動的にインストールします。

> **Note**
>
> Windows および Ubuntu AMIs には GPU ドライバーがインストールされていないため、GPU アクセラレーションワークロードの実行はすぐには機能しません。

自動プラグインのインストールを無効にし、特定のバージョンを手動でインストールするには、create コマンド`--install-nvidia-plugin=false`で を使用します。例えば、次のようになります。

```
eksctl create cluster --node-type=p2.xlarge --install-nvidia-plugin=false
```

バージョン 0.15.0 以降では、

```
kubectl create -f https://raw.githubusercontent.com/NVIDIA/k8s-device-plugin/<VERSION>/
deployments/static/nvidia-device-plugin.yml
```

または、古いバージョンの場合は、

```
kubectl create -f https://raw.githubusercontent.com/NVIDIA/k8s-device-plugin/<VERSION>/
nvidia-device-plugin.yml
```

Bottlerocket がデバイスプラグインの実行をすでに処理しているため、クラスターに Bottlerocket ノードグループのみが含まれている場合、NVIDIA Kubernetes デバイスプラグインのインストールはスキップされます。クラスターの設定で異なる AMI ファミリーを使用する場合は、テイントと許容範囲を使用して、デバイスプラグインが Bottlerocket ノードで実行されないようにする必要がある場合があります。

ARM サポート

このトピックでは、ARM ノードグループを使用してクラスターを作成する方法と、ARM ノードグループを既存のクラスターに追加する方法について説明します。

EKS は、Graviton プロセッサを搭載した 64 ビット ARM アーキテクチャをサポートしています。クラスターを作成するには、Graviton ベースのインスタンスタイプ (a1、、t4g、m6gm7g、、、m6gd、、c6gc7gc6gdr6g、、、c8g) r7g r6gd m8gのいずれかを選択しr8g、以下を実行します。

```
eksctl create cluster --node-type=a1.large
```

または 設定ファイルを使用します。

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

metadata:
  name: cluster-arm-1
  region: us-west-2


nodeGroups:
  - name: ng-arm-1
    instanceType: m6g.medium
    desiredCapacity: 1
```

```
eksctl create cluster -f cluster-arm-1.yaml
```

ARM はマネージド型ノードグループでもサポートされています。

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

metadata:
  name: cluster-arm-2
  region: us-west-2

managedNodeGroups:
  - name: mng-arm-1
    instanceType: m6g.medium
    desiredCapacity: 1
```

```
eksctl create cluster -f cluster-arm-2.yaml
```

AMI リゾルバー `auto`および `auto-ssm`は、ARM インスタンスタイプに基づいて正しい AMI を推測します。AmazonLinux2023, AmazonLinux2、Bottlerocket ファミリーのみが ARM 用に EKS 最適化 AMIs を持っています。

> **Note**
>
> ARM は、バージョン 1.15 以降のクラスターでサポートされています。

Auto Scaling

Auto Scaling を有効にする

クラスターオートスケーラーの使用を許可する IAM ロールを使用して、クラスター (または既存のクラスター内のノードグループ) を作成できます。

```
eksctl create cluster --asg-access
```

このフラグは k8s.io/cluster-autoscaler/enabledと k8s.io/cluster-autoscaler/<clusterName> タグも設定するため、ノードグループ検出は機能します。

クラスターが実行されたら、Cluster Autoscaler 自体をインストールする必要があります。

また、マネージド型またはアンマネージド型のノードグループ定義 (複数可) に以下を追加して、Cluster Autoscaler がノードグループをスケールするために必要なタグを追加する必要があります。

```
nodeGroups:
  - name: ng1-public
    iam:
      withAddonPolicies:
        autoScaler: true
```

0 からスケールアップする

ノードグループを 0 からスケールアップでき、ノードグループにラベルやテイントが定義されている場合は、Auto Scaling Groups (ASGs) にタグとして伝達する必要があります。

これを行う 1 つの方法は、ノードグループ定義の tagsフィールドに ASG タグを設定することです。たとえば、次のラベルとテイントを持つノードグループがあるとします。

```
nodeGroups:
  - name: ng1-public
```

```
    ...
    labels:
      my-cool-label: pizza
    taints:
      key: feaster
      value: "true"
      effect: NoSchedule
```

次の ASG タグを追加する必要があります。

```
nodeGroups:
  - name: ng1-public
    ...
    labels:
      my-cool-label: pizza
    taints:
      feaster: "true:NoSchedule"
    tags:
      k8s.io/cluster-autoscaler/node-template/label/my-cool-label: pizza
      k8s.io/cluster-autoscaler/node-template/taint/feaster: "true:NoSchedule"
```

マネージド型ノードグループとアンマネージド型ノードグループの両方で、これは `propagateASGTags`を に設定することで自動的に実行できます。これにより`true`、ラベルとテイントがタグとして Auto Scaling グループに追加されます。

```
nodeGroups:
  - name: ng1-public
    ...
    labels:
      my-cool-label: pizza
    taints:
      feaster: "true:NoSchedule"
    propagateASGTags: true
```

ゾーン対応の Auto Scaling

ワークロードがゾーン固有である場合は、ゾーンごとに個別のノードグループを作成する必要があります。これは、 がグループ内のすべてのノードが完全に同等であることを`cluster-autoscaler`前提としているためです。したがって、例えば、ゾーン固有の PVC (EBS ボリュームなど) を必要とするポッドによってスケールアップイベントがトリガーされた場合、新しいノードが間違った AZ でスケジュールされ、ポッドの起動が失敗する可能性があります。

環境が次の基準を満たしている場合、AZ ごとに個別のノードグループは必要ありません。

- ゾーン固有のストレージ要件はありません。
- 必須の podAffinity は、ホスト以外のトポロジでは必要ありません。
- ゾーンラベルに nodeAffinity は必要ありません。
- ゾーンラベルに nodeSelector がありません。

(詳細については、こちらとこちらを参照してください)。

上記のすべての要件 (および場合によっては他の要件) を満たしている場合は、複数の AZs にまたがる単一のノードグループで安全である必要があります。それ以外の場合は、個別の単一 AZ ノードグループを作成します。

前:

```
nodeGroups:
  - name: ng1-public
    instanceType: m5.xlarge
    # availabilityZones: ["eu-west-2a", "eu-west-2b"]
```

後:

```
nodeGroups:
  - name: ng1-public-2a
    instanceType: m5.xlarge
    availabilityZones: ["eu-west-2a"]
  - name: ng1-public-2b
    instanceType: m5.xlarge
    availabilityZones: ["eu-west-2b"]
```

カスタム AMI サポート

ノード AMI ID の設定

`--node-ami` フラグを使用すると、カスタム AMI の使用や、使用する AMI のリアルタイムクエリなど、さまざまな高度なユースケースが可能になります。フラグは、非 GPU イメージと GPU イメージの両方に使用できます。

フラグは、明示的に使用するイメージの AMI イメージ ID を取得できます。また、次の「特別な」キーワードを使用することもできます。

キーワード	説明
自動	ノードに使用する AMI が AWS EC2 をクエリして検索されることを示します。これは自動リゾルバーに関連しています。
auto-ssm	ノードに使用する AMI が AWS SSM パラメータストアにクエリを実行して検索されることを示します。

> **Note**
>
> 現時点では、EKS マネージド型ノードグループは、カスタム AMI を使用する場合、次の AMIs ファミリーのみをサポートします: `AmazonLinux2023`、`AmazonLinux2`、`Bottlerocket`、`Ubuntu2004`、`UbuntuPro2004`、`Ubuntu` および `Ubuntu2404`

`--node-ami` を ID 文字列に設定すると、 `eksctl` はカスタム AMI がリクエストされたことを前提としています。EKS マネージドノードとセルフマネージドノードの両方である AmazonLinux2 ノードと Ubuntu ノードの場合、これは `overrideBootstrapCommand` が必要であることを意味します。AmazonLinux2023 では、ノードブートストラップの`/etc/eks/bootstrap.sh`スクリプトの使用を停止するため、nodeadm 初期化プロセス (詳細については、ノードブートストラップドキュメントを参照してください) の代わりに、 `overrideBootstrapCommand`はサポートされていません。

CLI フラグの例:

```
eksctl create cluster --node-ami=auto

# with a custom ami id
eksctl create cluster --node-ami=ami-custom1234
```

Config ファイルの例:

```
nodeGroups:
  - name: ng1
    instanceType: p2.xlarge
    amiFamily: AmazonLinux2
    ami: auto
  - name: ng2
    instanceType: m5.large
    amiFamily: AmazonLinux2
    ami: ami-custom1234
managedNodeGroups:
  - name: m-ng-2
    amiFamily: AmazonLinux2
    ami: ami-custom1234
    instanceType: m5.large
    overrideBootstrapCommand: |
      #!/bin/bash
      /etc/eks/bootstrap.sh <cluster-name>
```

--node-ami フラグは でも使用できますeksctl create nodegroup。

ノード AMI ファミリーの設定

--node-ami-family では、次のキーワードを使用できます。

キーワード	説明
AmazonLinux2	Amazon Linux 2 に基づく EKS AMI イメージを使用することを示します (デフォルト)。
AmazonLinux2023	Amazon Linux 2023 に基づく EKS AMI イメージを使用する必要があることを示します。
Ubuntu2004	Ubuntu 20.04 LTS (Focal) に基づく EKS AMI イメージを使用する必要があることを示します (EKS ⇐ 1.29 でサポートされています)。
UbuntuPro2004	Ubuntu Pro 20.04 LTS (Focal) に基づく EKS AMI イメージを使用する必要があることを示します (EKS >= 1.27、⇐ 1.29 で使用可能)。

キーワード	説明
Ubuntu2204	Ubuntu 22.04 LTS (Jammy) に基づく EKS AMI イメージを使用する必要があることを示します (EKS >= 1.29 で使用可能)。
UbuntuPro2204	Ubuntu Pro 22.04 LTS (Jammy) に基づく EKS AMI イメージを使用する必要があることを示します (EKS >= 1.29 で使用可能)。
Ubuntu2404	Ubuntu 24.04 LTS (Noble) に基づく EKS AMI イメージを使用する必要があることを示します (EKS >= 1.31 で使用可能)。
UbuntuPro2404	Ubuntu Pro 24.04 LTS (Noble) に基づく EKS AMI イメージを使用する必要があることを示します (EKS >= 1.31 で使用可能)。
Bottlerocket	Bottlerocket に基づく EKS AMI イメージを使用する必要があることを示します。
WindowsServer2019FullContainer	Windows Server 2019 Full Container に基づく EKS AMI イメージを使用する必要があることを示します。
WindowsServer2019CoreContainer	Windows Server 2019 Core Container に基づく EKS AMI イメージを使用する必要があることを示します。
WindowsServer2022FullContainer	Windows Server 2022 Full Container に基づく EKS AMI イメージを使用する必要があることを示します。
WindowsServer2022CoreContainer	Windows Server 2022 Core Container に基づく EKS AMI イメージを使用する必要があることを示します。

CLI フラグの例:

```
eksctl create cluster --node-ami-family=AmazonLinux2
```

Config ファイルの例:

```
nodeGroups:
  - name: ng1
    instanceType: m5.large
    amiFamily: AmazonLinux2
managedNodeGroups:
  - name: m-ng-2
    instanceType: m5.large
    amiFamily: Ubuntu2204
```

`--node-ami-family` フラグは でも使用できますeksctl create nodegroup。 eksctlでは、カスタム AMI を使用するたびに、設定ファイルまたは `--node-ami-family` CLI フラグを使用して AMI ファミリーを明示的に設定する必要があります。

> **Note**
>
> 現時点では、EKS マネージド型ノードグループは、カスタム AMI を使用する場合、次の AMIs ファミリーのみをサポートします: AmazonLinux2023、AmazonLinux2、Bottlerocket、Ubuntu2004、UbuntuPro2004、Ubuntu よび Ubuntu2404

Windows カスタム AMI のサポート

カスタム AMI を指定できるのはセルフマネージド型の Windows ノードグループのみです。 は有効な Windows AMI ファミリーに設定amiFamilyする必要があります。

ブートストラップスクリプトでは、次の PowerShell 変数を使用できます。

```
$EKSBootstrapScriptFile
$EKSClusterName
$APIServerEndpoint
$Base64ClusterCA
$ServiceCIDR
$KubeletExtraArgs
$KubeletExtraArgsMap: A hashtable containing arguments for the kubelet, e.g., @{ 'node-
labels' = ''; 'register-with-taints' = ''; 'max-pods' = '10'}
```

```
$DNSClusterIP
$ContainerRuntime
```

Config ファイルの例:

```
nodeGroups:
  - name: custom-windows
    amiFamily: WindowsServer2022FullContainer
    ami: ami-01579b74557facaf7
    overrideBootstrapCommand: |
      & $EKSBootstrapScriptFile -EKSClusterName "$EKSClusterName" -APIServerEndpoint
 "$APIServerEndpoint" -Base64ClusterCA "$Base64ClusterCA" -ContainerRuntime
 "containerd" -KubeletExtraArgs "$KubeletExtraArgs" 3>&1 4>&1 5>&1 6>&1
```

Bottlerocket カスタム AMI のサポート

Bottlerocket ノードでは、 `overrideBootstrapCommand`はサポートされていません。
代わりに、独自のブートストラップコンテナを指定するには、設定ファイルの一部として
`bottlerocket`フィールドを使用する必要があります。例:

```
  nodeGroups:
  - name: bottlerocket-ng
    ami: ami-custom1234
    amiFamily: Bottlerocket
    bottlerocket:
      enableAdminContainer: true
      settings:
        bootstrap-containers:
          bootstrap:
            source: <MY-CONTAINER-URI>
```

Windows ワーカーノード

バージョン 1.14 以降、Amazon EKS は Windows コンテナの実行を許可する Windows ノードをサポートしています。Windows ノードに加えて、Microsoft はまだホストネットワーキングモードをサポートしていないため、 CoreDNS を実行するにはクラスター内の Linux ノードが必要です。したがって、Windows EKS クラスターは Windows ノードと少なくとも 1 つの Linux ノードが混在します。Linux ノードはクラスターの機能にとって重要です。したがって、本番稼働用クラスターでは、HA 用に少なくとも 2 つの `t2.large` Linux ノードを持つことをお勧めします。

> **Note**
>
> 2021 年 10 月 22 日以降に作成された EKS クラスターで Windows ワークロードを実行するために、VPC リソースコントローラーを Linux ワーカーノードにインストールする必要がなくなりました。EKS コントロールプレーンで Windows IP アドレス管理を有効にするには、ConfigMap 設定を使用します (詳細については、「link:eks/latest/userguide/windows-support.html」を参照してください)。eksctl は ConfigMap に自動的にパッチを適用して、Windows ノードグループの作成時に Windows IP アドレス管理を有効にします。

Windows サポートを使用した新しいクラスターの作成

設定ファイルの構文を使用すると、Windows をサポートする完全に機能するクラスターを 1 つのコマンドで作成できます。

```
# cluster.yaml
# An example of ClusterConfig containing Windows and Linux node groups to support
 Windows workloads
---
apiVersion: eksctl.io/v1alpha5
kind: ClusterConfig

metadata:
  name: windows-cluster
  region: us-west-2

nodeGroups:
  - name: windows-ng
    amiFamily: WindowsServer2019FullContainer
    minSize: 2
    maxSize: 3

managedNodeGroups:
  - name: linux-ng
    instanceType: t2.large
    minSize: 2
    maxSize: 3

  - name: windows-managed-ng
    amiFamily: WindowsServer2019FullContainer
    minSize: 2
```

```
    maxSize: 3
```

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

設定ファイルを使用せずに Windows アンマネージド型ノードグループで新しいクラスターを作成するには、次のコマンドを発行します。

```
eksctl create cluster --managed=false --name=windows-cluster --node-ami-
family=WindowsServer2019CoreContainer
```

既存の Linux クラスターへの Windows サポートの追加

Linux ノード (AmazonLinux2 AMI ファミリー) を持つ既存のクラスターで Windows ワークロードの実行を有効にするには、Windows ノードグループを追加する必要があります。

新しい Windows マネージド型ノードグループのサポートが追加されました (--managed=true またはフラグを省略) 。

```
eksctl create nodegroup --managed=false --cluster=existing-cluster --node-ami-
family=WindowsServer2019CoreContainer
eksctl create nodegroup --cluster=existing-cluster --node-ami-
family=WindowsServer2019CoreContainer
```

ワークロードを適切な OS でスケジュールするには、実行する必要がある OS をnodeSelectorターゲットとする が必要です。

```
# Targeting Windows
  nodeSelector:
    kubernetes.io/os: windows
    kubernetes.io/arch: amd64
```

```
# Targeting Linux
  nodeSelector:
    kubernetes.io/os: linux
    kubernetes.io/arch: amd64
```

より古いクラスターを使用している場合は、 kubernetes.io/os および 1.19 kubernetes.io/archラベルbeta.kubernetes.io/archをそれぞれ beta.kubernetes.io/osおよび に置き換える必要があります。

詳細情報

- EKS Windows のサポート

追加のボリュームマッピング

追加の設定オプションとして、ボリュームマッピングを処理する場合、ノードグループの作成時に追加のマッピングを設定できます。

これを行うには、additionalVolumes次のように フィールドを設定します。

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

metadata:
  name: dev-cluster
  region: eu-north-1

managedNodeGroups:
  - name: ng-1-workers
    labels: { role: workers }
    instanceType: m5.xlarge
    desiredCapacity: 10
    volumeSize: 80
    additionalVolumes:
      - volumeName: '/tmp/mount-1' # required
        volumeSize: 80
        volumeType: 'gp3'
        volumeEncrypted: true
        volumeKmsKeyID: 'id'
        volumeIOPS: 3000
        volumeThroughput: 125
      - volumeName: '/tmp/mount-2'  # required
        volumeSize: 80
        volumeType: 'gp2'
        snapshotID: 'snapshot-id'
```

volumeNames の選択の詳細については、デバイスの命名に関するドキュメントを参照してください。EBS ボリューム、インスタンスボリュームの制限、ブロックデバイスマッピングの詳細については、このページを参照してください。

EKS ハイブリッドノード

序章

AWS EKS では、ハイブリッドノードが導入されています。ハイブリッドノードは、AWS クラウドで使用するのと同じ AWS EKS クラスター、機能、ツールを使用して、カスタマーマネージド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るようにする新機能です。AWS EKS Hybird Nodes は、AWS が管理する Kubernetes エクスペリエンスをオンプレミス環境に導入し、お客様がオンプレミス、エッジ、クラウド環境間でアプリケーションを実行する方法を簡素化および標準化できるようにします。詳細については、「EKS Hybrid Nodes」を参照してください。

この機能のサポートを容易にするために、eksctl は という新しい最上位フィールドを導入しましたremoteNetworkConfig。Hybrid Nodes 関連の設定は、設定ファイルの一部としてこのフィールドを介して設定する必要があります。対応する CLI フラグはありません。さらに、起動時に、リモートネットワーク設定はクラスターの作成時にのみセットアップでき、後で更新することはできません。つまり、ハイブリッドノードを使用するように既存のクラスターを更新することはできません。

設定ファイルの remoteNetworkConfigセクションでは、リモートノードを EKS クラスターに結合するときに、ネットワークと認証情報の 2 つのコア領域を設定できます。

ネットワーク

EKS Hybrid Nodes は、オンプレミスネットワーク (複数可) を AWS VPC に接続するための任意の方法に柔軟に対応します。AWS Site-to-Site VPN や AWS Direct Connect など、いくつかの文書化されたオプションがあり、ユースケースに最適な方法を選択できます。選択したほとんどの方法では、VPC 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) のいずれかにアタッチされます。eksctl を使用して VPC を作成する場合、eksctl は VPC の範囲内で、EKS コントロールプレーンとリモートノード間の通信を容易にするために、ネットワーク関連の前提条件も設定します。

- Ingress/Egress SG ルール
- プライベートサブネットのルートテーブルの ルート
- 指定された TGW または VGW への VPC ゲートウェイアタッチメント

設定ファイルの例:

```
remoteNetworkConfig:
  vpcGatewayID: tgw-xxxx # either VGW or TGW to be attached to your VPC
  remoteNodeNetworks:
    # eksctl will create, behind the scenes, SG rules, routes, and a VPC gateway
 attachment,
    # to facilitate communication between remote network(s) and EKS control plane, via
 the attached gateway
    - cidrs: ["10.80.146.0/24"]
  remotePodNetworks:
    - cidrs: ["10.86.30.0/23"]
```

選択した接続方法に TGW または VGW を使用しない場合は、eksctl を使用して VPC を作成し、代わりに既存の VPC を提供することはできません。関連する注意点として、既存の VPC を使用している場合、eksctl はそれを修正せず、すべてのネットワーク要件が満たされていることを確認するのはお客様の責任です。

> **Note**
>
> eksctl は、AWS VPC の外部にネットワークインフラストラクチャ (つまり、VPG/TGW からリモートネットワークへのインフラストラクチャ) をセットアップしません。

認証情報

EKS Hybrid Nodes は、AWS SSM または AWS IAM Roles Anywhere によってプロビジョニングされた AWS IAM Authenticator と一時的な IAM 認証情報を使用して、EKS クラスターで認証します。セルフマネージド型ノードグループと同様に、特に指定されていない場合、eksctl はリモートノードが引き受けるハイブリッドノード IAM ロールを作成します。さらに、IAM Roles Anywhere を認証情報プロバイダーとして使用すると、eksctl は特定の認証機関バンドル (`iam.caBundleCert`) に基づいてプロファイルとトラストアンカーを設定します。例:

```
remoteNetworkConfig:
  iam:
    # the provider for temporary IAM credentials. Default is SSM.
    provider: IRA
    # the certificate authority bundle that serves as the root of trust,
    # used to validate the X.509 certificates provided by your nodes.
    # can only be set when provider is IAMRolesAnywhere.
    caBundleCert: xxxx
```

eksctl によって作成されたハイブリッドノードロールの ARN は、リモートノードをクラスターに結合し、 NodeConfigをセットアップしnodeadm、アクティベーションを作成するプロセスの後半で必要です (SSM を使用している場合) 。取得するには、以下を使用します。

```
aws cloudformation describe-stacks \
  --stack-name eksctl-<CLUSTER_NAME>-cluster \
  --query 'Stacks[].Outputs[?OutputKey==`RemoteNodesRoleARN`].[OutputValue]' \
  --output text
```

同様に、IAM Roles Anywhere を使用する場合、eksctl によって作成されたトラストアンカーと anywhere プロファイルの ARN を取得し、RemoteNodesAnywhereProfileARNそれぞれ を RemoteNodesTrustAnchorARNまたは RemoteNodesRoleARNに置き換えて前のコマンドを修正できます。

既存の IAM Roles Anywhere 設定がある場合、または SSM を使用している場合は、 を介してハイブリッドノードの IAM ロールを指定できますremoteNetworkConfig.iam.roleARN。このシナリオでは、eksctl はトラストアンカーと anywhere プロファイルを作成しないことに注意してください。例:

```
remoteNetworkConfig:
  iam:
    roleARN: arn:aws:iam::000011112222:role/HybridNodesRole
```

ロールを Kubernetes ID にマッピングし、リモートノードが EKS クラスターに参加することを許可するために、eksctl はハイブリッドノードの IAM ロールをプリンシパル ARN として、タイプ HYBRID_LINUXのアクセスエントリを作成します。つまり、

```
eksctl get accessentry --cluster my-cluster --principal-arn
 arn:aws:iam::000011112222:role/eksctl-my-cluster-clust-HybridNodesSSMRole-XiIAg0d29PkO
 --output json
[
    {
        "principalARN": "arn:aws:iam::000011112222:role/eksctl-my-cluster-clust-
HybridNodesSSMRole-XiIAg0d29PkO",
        "kubernetesGroups": [
            "system:nodes"
        ]
    }
]
```

アドオンのサポート

Container Networking Interface (CNI): AWS VPC CNI はハイブリッドノードでは使用できません。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、Helm などの任意のツールで管理できます。詳細については、「ハイブリッドノードの CNI を設定する」を参照してください。

> **Note**
>
> セルフマネージド型または EKS マネージド型ノードグループのクラスターに VPC CNI をインストールする場合は、 `v1.19.0-eksbuild.1`以降を使用する必要があります。これには、ハイブリッドノードへのインストールを除外するためにアドオンのデーモンセットへの udpate が含まれるためです。

その他のリファレンス

- EKS Hybrid Nodes UserDocs
- 起動のお知らせ

EKS マネージド型ノードグループのノード修復設定のサポート

EKS Managed Nodegroups は、マネージドノードの状態がモニタリングされ、異常なワーカーノードがそれに応じて置き換えまたは再起動されるノード修復をサポートしています。eksctl は、ノード修復動作をきめ細かく制御するための包括的な設定オプションを提供するようになりました。

基本的なノード修復設定

CLI フラグの使用

基本的なノード修復を使用してマネージド型ノードグループを持つクラスターを作成するには、 `--enable-node-repair`フラグを渡します。

```
eksctl create cluster --enable-node-repair
```

既存のクラスターでノード修復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:

```
eksctl create nodegroup --cluster=<clusterName> --enable-node-repair
```

設定ファイルの使用

```
# basic-node-repair.yaml
apiVersion: eksctl.io/v1alpha5
kind: ClusterConfig

metadata:
  name: basic-node-repair-cluster
  region: us-west-2

managedNodeGroups:
- name: ng-1
  nodeRepairConfig:
    enabled: true
```

```
eksctl create cluster -f basic-node-repair.yaml
```

拡張ノード修復設定

しきい値の設定

ノード修復アクションが、パーセンテージまたはカウントベースのしきい値を使用していつ停止するかを設定できます。注: パーセンテージしきい値とカウントしきい値を同時に使用することはできません。

しきい値の CLI フラグ

```
# Percentage-based threshold - repair stops when 20% of nodes are unhealthy
eksctl create cluster --enable-node-repair \
  --node-repair-max-unhealthy-percentage=20

# Count-based threshold - repair stops when 5 nodes are unhealthy
eksctl create cluster --enable-node-repair \
  --node-repair-max-unhealthy-count=5
```

しきい値の設定ファイル

```
managedNodeGroups:
```

```
- name: threshold-ng
  nodeRepairConfig:
    enabled: true
    # Stop repair actions when 20% of nodes are unhealthy
    maxUnhealthyNodeThresholdPercentage: 20
    # Alternative: stop repair actions when 3 nodes are unhealthy
    # maxUnhealthyNodeThresholdCount: 3
    # Note: Cannot use both percentage and count thresholds simultaneously
```

並列修復の制限

同時にまたは並行して修復できるノードの最大数を制御します。これにより、ノード置換のペースをよりきめ細かく制御できます。注: パーセンテージ制限とカウント制限を同時に使用することはできません。

並列制限の CLI フラグ

```
# Percentage-based parallel limits - repair at most 15% of unhealthy nodes in parallel
eksctl create cluster --enable-node-repair \
  --node-repair-max-parallel-percentage=15

# Count-based parallel limits - repair at most 2 unhealthy nodes in parallel
eksctl create cluster --enable-node-repair \
  --node-repair-max-parallel-count=2
```

並列制限の設定ファイル

```
managedNodeGroups:
- name: parallel-ng
  nodeRepairConfig:
    enabled: true
    # Repair at most 15% of unhealthy nodes in parallel
    maxParallelNodesRepairedPercentage: 15
    # Alternative: repair at most 2 unhealthy nodes in parallel
    # maxParallelNodesRepairedCount: 2
    # Note: Cannot use both percentage and count limits simultaneously
```

カスタム修復オーバーライド

特定の修復アクションの詳細なオーバーライドを指定します。これらのオーバーライドは、ノードが修復対象と見なされるまでの修復アクションと修復遅延時間を制御します。これを使用する場合は、各オーバーライドのすべての値を指定する必要があります。

```
managedNodeGroups:
- name: custom-repair-ng
  instanceType: g4dn.xlarge  # GPU instances
  nodeRepairConfig:
    enabled: true
    maxUnhealthyNodeThresholdPercentage: 25
    maxParallelNodesRepairedCount: 1
    nodeRepairConfigOverrides:
      # Handle GPU-related failures with immediate termination
      - nodeMonitoringCondition: "AcceleratedInstanceNotReady"
        nodeUnhealthyReason: "NvidiaXID13Error"
        minRepairWaitTimeMins: 10
        repairAction: "Terminate"
      # Handle network issues with restart after waiting
      - nodeMonitoringCondition: "NetworkNotReady"
        nodeUnhealthyReason: "InterfaceNotUp"
        minRepairWaitTimeMins: 20
        repairAction: "Restart"
```

完全な設定例

すべての設定オプションを含む包括的な例については、「examples/44-node-repair.yaml」を参照してください。

例 1: しきい値の割合による基本的な修復

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

metadata:
  name: basic-repair-cluster
  region: us-west-2

managedNodeGroups:
- name: basic-ng
  instanceType: m5.large
  desiredCapacity: 3
  nodeRepairConfig:
    enabled: true
    maxUnhealthyNodeThresholdPercentage: 20
    maxParallelNodesRepairedPercentage: 15
```

例 2: 重要なワークロードの保守的な修復

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

metadata:
  name: critical-workload-cluster
  region: us-west-2

managedNodeGroups:
- name: critical-ng
  instanceType: c5.2xlarge
  desiredCapacity: 6
  nodeRepairConfig:
    enabled: true
    # Very conservative settings
    maxUnhealthyNodeThresholdPercentage: 10
    maxParallelNodesRepairedCount: 1
    nodeRepairConfigOverrides:
      # Wait longer before taking action on critical workloads
      - nodeMonitoringCondition: "NetworkNotReady"
        nodeUnhealthyReason: "InterfaceNotUp"
        minRepairWaitTimeMins: 45
        repairAction: "Restart"
```

例 3: 特殊な修復による GPU ワークロード

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

metadata:
  name: gpu-workload-cluster
  region: us-west-2

managedNodeGroups:
- name: gpu-ng
  instanceType: g4dn.xlarge
  desiredCapacity: 4
  nodeRepairConfig:
    enabled: true
    maxUnhealthyNodeThresholdPercentage: 25
    maxParallelNodesRepairedCount: 1
    nodeRepairConfigOverrides:
```

```
      # GPU failures require immediate termination
      - nodeMonitoringCondition: "AcceleratedInstanceNotReady"
        nodeUnhealthyReason: "NvidiaXID13Error"
        minRepairWaitTimeMins: 5
        repairAction: "Terminate"
```

CLI リファレンス

ノード修復フラグ

フラグ	説明	例
`--enable-node-repair`	自動ノード修復を有効にする	`--enable-node-repair`
`--node-repair-max-unhealthy-percentage`	修復前の異常なノードの最大パーセンテージ	`--node-repair-max-unhealthy-percentage=20`
`--node-repair-max-unhealthy-count`	修復前の異常なノードの最大数	`--node-repair-max-unhealthy-count=5`
`--node-repair-max-parallel-percentage`	並行して修復するノードの最大パーセンテージ	`--node-repair-max-parallel-percentage=15`
`--node-repair-max-parallel-count`	並行して修復するノードの最大数	`--node-repair-max-parallel-count=2`

注: ノード修復設定の上書きは、その複雑さにより YAML 設定ファイルでのみサポートされます。

設定リファレンス

nodeRepairConfig

フィールド	タイプ	説明	制約	例
`enabled`	boolean	ノード修復の有効化/無効化	-	`true`
`maxUnhealthyNodeThresholdPercentage`	integer	ノードの自動修復アクションが停止する異常ノードのしきい値の割合	では使用できません `maxUnhealthyNodeThresholdCount`	`20`
`maxUnhealthyNodeThresholdCount`	integer	ノードの自動修復アクションが停止する異常なノードのカウントしきい値	では使用できません `maxUnhealthyNodeThresholdPercentage`	`5`
`maxParallelNodesRepairedPercentage`	integer	同時または並行して修復できる異常なノードの最大割合	では使用できません `maxParallelNodesRepairedCount`	`15`
`maxParallelNodesRepairedCount`	integer	同時または並行して修復できる異常なノードの最大数	では使用できません `maxParallelNodesRepairedPercentage`	`2`

フィールド	タイプ	説明	制約	例
nodeRepairConfigOverrides	array	修復アクションと遅延時間を制御する特定の修復アクションの詳細なオーバーライド	オーバーライドごとにすべての値を指定する必要があります	上記の例を参照してください。

nodeRepairConfigOverrides

フィールド	タイプ	説明	有効な値
nodeMonitoringCondition	string	このオーバーライドが適用されるノードモニタリングエージェントによって報告される異常な状態	"AcceleratedInstanceNotReady" , "NetworkNotReady"
nodeUnhealthyReason	string	このオーバーライドが適用されるノードモニタリングエージェントによって報告された理由	"NvidiaXID13Error" ,"InterfaceNotUp"
minRepairWaitTimeMins	integer	指定された条件と理由でノードを修復しようとするまでの最小待機時間	任意の正の整数
repairAction	string	指定された条件がすべて満たされたときにノードに対して実行する修復アクション	"Terminate" , "Restart" , "NoAction"

詳細情報

- EKS マネージド型ノードグループノードのヘルス

ネットワーク

この章では、Eksctl が EKS クラスター用の Virtual Private Cloud (VPC) ネットワークを作成する方法について説明します。

トピック:

- the section called “VPC 設定”
  - VPC CIDR 範囲を変更し、IPv6 アドレス指定を設定する
  - 既存の VPC を使用する
  - 新しい EKS クラスターの VPC、サブネット、セキュリティグループ、NAT ゲートウェイをカスタマイズする
- the section called “サブネット設定”
  - 初期ノードグループにプライベートサブネットを使用してパブリックインターネットから分離する
  - アベイラビリティーゾーンごとに複数のサブネットを一覧表示し、ノードグループ設定でサブネットを指定してサブネットトポロジをカスタマイズする
  - VPC 設定内の特定の名前付きサブネットにノードグループを制限する
  - ノードグループにプライベートサブネットを使用する場合は、 `privateNetworking`を に設定します。 `true`
  - VPC 仕様の `public`と の両方`private`の設定で完全なサブネット仕様を提供する
  - ノードグループ設定で指定`availabilityZones`できる `subnets`または は 1 つだけです。
- the section called “クラスターアクセス”
  - EKS クラスター内の Kubernetes API サーバーエンドポイントへのパブリックアクセスとプライベートアクセスを管理する
  - 許可された CIDR 範囲を指定して EKS Kubernetes パブリック API エンドポイントへのアクセスを制限する
  - 既存のクラスターの API サーバーエンドポイントアクセス設定とパブリックアクセス CIDR 制限を更新する
- the section called “コントロールプレーンネットワーク”
  - クラスターの EKS コントロールプレーンで使用されるサブネットを更新する
- the section called “IPv6 のサポート”

- EKS クラスターで VPC を作成するときに使用する IP バージョン (IPv4 または IPv6) を指定します。

VPC 設定

クラスター専用 VPC

デフォルトでは`eksctl create cluster`、クラスター専用の VPC が作成されます。これは、セキュリティなど、さまざまな理由で既存のリソースとの干渉を避けるために行われますが、既存の VPC 内のすべての設定を検出するのは難しいためでもあります。

- で使用されるデフォルトの VPC CIDR `eksctl`は です`192.168.0.0/16`。
  - 8 (/19) サブネット (3 つのプライベート、3 つのパブリック、2 つのリザーブド) に分割されます。
- 初期ノードグループはパブリックサブネットに作成されます。
- `--allow-ssh` が指定されていない限り、SSH アクセスは無効になります。
- デフォルトでは、ノードグループはポート 1025 ~ 65535 のコントロールプレーン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します。

> **Note**
>
> eksctl `us-east-1` では、デフォルトで 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のみが作成されます。

VPC CIDR を変更する

別の VPC とのピア接続を設定する必要がある場合、または単により大きい範囲またはより小さい範囲の IPs が必要な場合は、 `--vpc-cidr`フラグを使用して変更できます。AWS VPC での使用が許可されている CIDR ブロックの選択に関するガイドについては、AWS ドキュメントを参照してください。

IPv6 クラスターを作成する場合は、cluser 設定ファイル`VPC.IPv6Cidr`で を設定できます。この設定は、CLI フラグではなく、設定ファイルでのみ行われます。

IPv6 IP アドレスブロックを所有している場合は、独自の IPv6 プールを持ち込むこともできます。独自のプールのインポート方法については、「独自の IP アドレス (BYOIP) を Amazon EC2 に持ち込む」を参照してください。次に、cluser 設定ファイルの `VPC.IPv6Cidr` を使用して Eksctl を設定します。

既存の VPC を使用する: kops と共有

kops によって管理されている既存の Kubernetes クラスターの VPC を使用できます。この機能は、移行やクラスターピアリングを容易にするために提供されています。

kops を使用して以前にクラスターを作成したことがある場合は、次のようなコマンドを使用します。

```
export KOPS_STATE_STORE=s3://kops
kops create cluster cluster-1.k8s.local --zones=us-west-2c,us-west-2b,us-west-2a --
networking=weave --yes
```

同じ VPC サブネットを使用して、同じ AZs に EKS クラスターを作成できます (注: 少なくとも 2 つの AZs/サブネットが必要です) 。

```
eksctl create cluster --name=cluster-2 --region=us-west-2 --vpc-from-kops-
cluster=cluster-1.k8s.local
```

既存の VPC を使用する: その他のカスタム設定

`eksctl` は、カスタム VPC およびサブネットトポロジーの柔軟性を提供しますが、完全ではありません。

`--vpc-private-subnets` および `--vpc-public-subnets`フラグを使用してプライベートサブネットやパブリックサブネットを指定することで、既存の VPC を使用できます。設定が異なるため、サブネットが実際にプライベートかパブリックかを検証する簡単な方法がないため、使用するサブネットが正しく分類されていることを確認するのはお客様次第です。

これらのフラグがある場合、 `eksctl create cluster` は自動的に VPC ID を決定しますが、ルーティングテーブルやインターネット/NAT ゲートウェイなどの他のリソースは作成されません。ただし、最初のノードグループとコントロールプレーン専用のセキュリティグループが作成されます。

異なる AZs、この条件は EKS によってチェックされます。既存の VPC を使用する場合、以下の要件は EKS または Eksctl によって強制またはチェックされず、EKS によってクラスターが作成され

ます。クラスターの一部の基本関数は、これらの要件なしで動作します。(例えば、タグ付けは厳密に必要ではなく、サブネットにタグを設定せずに機能クラスターを作成できることがテストで示されていますが、これが常に保持される保証はなく、タグ付けが推奨されます)。

標準要件:

- 指定されたすべてのサブネットは、IP の同じブロック内の同じ VPC IPs 内にある必要があります。
- 必要に応じて、十分な数の IP アドレスが使用可能
- 必要に応じて十分な数のサブネット (最小 2)
- サブネットには、少なくとも次のタグが付けられます。
  - kubernetes.io/cluster/<name> タグを sharedまたは に設定 owned
  - kubernetes.io/role/internal-elb プライベートサブネット1の タグを に設定
  - kubernetes.io/role/elb パブリックサブネット1の タグを に設定
- 正しく設定されたインターネットゲートウェイや NAT ゲートウェイ
- ルーティングテーブルに正しいエントリがあり、ネットワークが機能している
- 新規: すべてのパブリックサブネットで プロパティMapPublicIpOnLaunchが有効になっている必要があります (AWS コンソールAuto-assign public IPv4 address内)。マネージド型ノードグループと Fargate はパブリック IPv4 アドレスを割り当てません。 プロパティはサブネットに設定する必要があります。

EKS または Kubernetes によって課されるその他の要件があり、要件やレコメンデーションup-to-date状態に保ち、必要に応じて実装するか、可能な場合は実装するかはお客様次第です。

によって適用されるデフォルトのセキュリティグループ設定はeksctl、他のセキュリティグループのリソースとアクセスを共有するのに十分ではない場合があります。セキュリティグループの進入/退出ルールを変更する場合は、別のツールを使用して変更を自動化するか、EC2 コンソールを使用して変更を行う必要があります。

疑わしい場合は、カスタム VPC を使用しないでください。--vpc-* フラグeksctl create clusterなしで を使用すると、常に完全に機能する専用 VPC でクラスターが設定されます。

例

2 つのプライベートサブネットと 2 つのパブリックサブネットを持つカスタム VPC を使用してクラスターを作成します。

```
eksctl create cluster \
  --vpc-private-subnets=subnet-0ff156e0c4a6d300c,subnet-0426fb4a607393184 \
  --vpc-public-subnets=subnet-0153e560b3129a696,subnet-009fa0199ec203c37
```

または、以下の同等の設定ファイルを使用します。

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

metadata:
  name: my-test
  region: us-west-2

vpc:
  id: "vpc-11111"
  subnets:
    private:
      us-west-2a:
          id: "subnet-0ff156e0c4a6d300c"
      us-west-2c:
          id: "subnet-0426fb4a607393184"
    public:
      us-west-2a:
          id: "subnet-0153e560b3129a696"
      us-west-2c:
          id: "subnet-009fa0199ec203c37"

nodeGroups:
  - name: ng-1
```

3 つのプライベートサブネットを持つカスタム VPC を使用してクラスターを作成し、初期ノードグループにそれらのサブネットを使用させます。

```
eksctl create cluster \
  --vpc-private-
subnets=subnet-0ff156e0c4a6d300c,subnet-0549cdab573695c03,subnet-0426fb4a607393184 \
  --node-private-networking
```

または、以下の同等の設定ファイルを使用します。

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

```
metadata:
  name: my-test
  region: us-west-2

vpc:
  id: "vpc-11111"
  subnets:
    private:
      us-west-2d:
          id: "subnet-0ff156e0c4a6d300c"
      us-west-2c:
          id: "subnet-0549cdab573695c03"
      us-west-2a:
          id: "subnet-0426fb4a607393184"

nodeGroups:
  - name: ng-1
    privateNetworking: true
```

カスタム VPC 4x パブリックサブネットを使用してクラスターを作成します。

```
eksctl create cluster \
  --vpc-public-
subnets=subnet-0153e560b3129a696,subnet-0cc9c5aebe75083fd,subnet-009fa0199ec203c37,subnet-018fa
```

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

metadata:
  name: my-test
  region: us-west-2

vpc:
  id: "vpc-11111"
  subnets:
    public:
      us-west-2d:
          id: "subnet-0153e560b3129a696"
      us-west-2c:
          id: "subnet-0cc9c5aebe75083fd"
      us-west-2a:
          id: "subnet-009fa0199ec203c37"
```

```
      us-west-2b:
          id: "subnet-018fa0176ba320e45"

nodeGroups:
  - name: ng-1
```

その他の例は、リポジトリの examplesフォルダにあります。

- 既存の VPC の使用
- カスタム VPC CIDR の使用

カスタム共有ノードセキュリティグループ

eksctl は、アンマネージドノードとクラスターコントロールプレーンおよびマネージドノード間の通信を許可する共有ノードセキュリティグループを作成および管理します。

代わりに独自のカスタムセキュリティグループを指定する場合は、設定ファイルの sharedNodeSecurityGroupフィールドを上書きできます。

```
vpc:
  sharedNodeSecurityGroup: sg-0123456789
```

デフォルトでは、 eksctlはクラスターを作成するときに、このセキュリティグループにルールを追加して、EKS が作成するデフォルトのクラスターセキュリティグループとの間の通信を許可します。デフォルトのクラスターセキュリティグループは、EKS コントロールプレーンとマネージド型ノードグループの両方で使用されます。

セキュリティグループルールを自分で管理する場合は、設定ファイルfalseで manageSharedNodeSecurityGroupRulesを に設定することで、 がルールを作成eksctlできない場合があります。

```
vpc:
  sharedNodeSecurityGroup: sg-0123456789
  manageSharedNodeSecurityGroupRules: false
```

NAT Gateway

クラスターの NAT ゲートウェイはDisable、、 Single (デフォルト) 、または に設定できますHighlyAvailable。HighlyAvailable オプションは、リージョンの各アベイラビリティー

ゾーンに NAT ゲートウェイをデプロイするため、AZ がダウンしても、他の AZs のノードは引き続きインターネットと通信できます。

これは、CLI `--vpc-nat-mode` フラグまたは以下の例のようにクラスター設定ファイルで指定できます。

```
vpc:
  nat:
    gateway: HighlyAvailable # other options: Disable, Single (default)
```

完全な例については、こちらを参照してください。

> **Note**
>
> NAT ゲートウェイの指定は、クラスターの作成時にのみサポートされます。クラスターのアップグレード中は操作されません。

サブネット設定

初期ノードグループにプライベートサブネットを使用する

初期ノードグループをパブリックインターネットから分離する場合は、 `--node-private-networking`フラグを使用できます。`--ssh-access` フラグと組み合わせて使用すると、SSH ポートには VPC 内からのみアクセスできます。

> **Note**
>
> `--node-private-networking` フラグを使用すると、送信トラフィックは Elastic IP を使用して NAT ゲートウェイを通過します。一方、ノードがパブリックサブネットにある場合、送信トラフィックは NAT ゲートウェイを通過しないため、送信トラフィックには個々のノードの IP があります。

カスタムサブネットトポロジ

`eksctl` バージョン `0.32.0` では、次の機能を備えたサブネットトポロジのカスタマイズがさらに導入されました。

- VPC 設定の AZ ごとに複数のサブネットを一覧表示する
- ノードグループ設定でサブネットを指定する

以前のバージョンでは、カスタムサブネットはアベイラビリティーゾーンによって提供される必要がありました。つまり、AZ ごとに 1 つのサブネットしかリストできませんでした。識別キー`0.32.0`は任意です。

```
vpc:
  id: "vpc-11111"
  subnets:
    public:
      public-one:                           # arbitrary key
          id: "subnet-0153e560b3129a696"
      public-two:
          id: "subnet-0cc9c5aebe75083fd"
      us-west-2b:                           # or list by AZ
          id: "subnet-018fa0176ba320e45"
    private:
      private-one:
          id: "subnet-0153e560b3129a696"
      private-two:
          id: "subnet-0cc9c5aebe75083fd"
```

> **Important**
>
> AZ を識別キーとして使用する場合は、`az`値を省略できます。

上記のような任意の文字列を識別キーとして使用する場合は、次のいずれかを実行します。

- `id` を設定する必要があります (`az` および`cidr`オプション)
- または を設定`az`する必要があります (`cidr`オプション)

ユーザーが CIDR と ID を指定せずに AZ でサブネットを指定した場合、その AZ 内のサブネットは VPC から任意に選択されます。

> **Note**
>
> 完全なサブネット仕様、つまり VPC 仕様で宣言された publicと の両方privateの設定を指定する必要があります。

ノードグループは、 設定を介して名前付きサブネットに制限できます。ノードグループ設定でサブネットを指定する場合は、サブネット ID ではなく VPC 仕様で指定されている識別キーを使用します。例えば、次のようになります。

```
vpc:
  id: "vpc-11111"
  subnets:
    public:
      public-one:
          id: "subnet-0153e560b3129a696"
    ... # subnet spec continued

nodeGroups:
  - name: ng-1
    instanceType: m5.xlarge
    desiredCapacity: 2
    subnets:
      - public-one
```

> **Note**
>
> ノードグループ設定で指定availabilityZonesできる subnetsまたは は 1 つだけです。

プライベートサブネット内にノードグループを配置する場合、ノードグループtrueで を に設定privateNetworkingする必要があります。

```
vpc:
  id: "vpc-11111"
  subnets:
    public:
      private-one:
          id: "subnet-0153e560b3129a696"
    ... # subnet spec continued
```

```
nodeGroups:
  - name: ng-1
    instanceType: m5.xlarge
    desiredCapacity: 2
    privateNetworking: true
    subnets:
      - private-one
```

完全な設定例については、eksctl GitHub リポジトリの「24-nodegroup-subnets.yaml」を参照してください。

クラスターアクセス

Kubernetes API サーバーエンドポイントへのアクセスの管理

デフォルトでは、EKS クラスターは Kubernetes API サーバーをパブリックに公開しますが、VPC サブネット内 (public=true、private=false) から直接公開することはありません。VPC 内から API サーバー宛てのトラフィックは、まず VPC ネットワーク (Amazon のネットワークではなく) を離れてから再入力して API サーバーに到達する必要があります。

クラスターの Kubernetes API サーバーエンドポイントアクセスは、クラスター設定ファイルを使用してクラスターを作成するときに、パブリックアクセスとプライベートアクセスに設定できます。以下の例:

```
vpc:
  clusterEndpoints:
    publicAccess:  <true|false>
    privateAccess: <true|false>
```

Kubernetes API エンドポイントアクセスを設定する際には、いくつかの追加の注意点があります。

1. EKS では、プライベートアクセスまたはパブリックアクセスが有効になっていないクラスターは許可されません。
2. EKS では、プライベートアクセスのみを有効にする設定を作成できますが、eksctl はクラスターの作成中にその設定をサポートしません。これは、eksctl がワーカーノードをクラスターに結合できないためです。
3. プライベートのみの Kubernetes API エンドポイントアクセスを持つようにクラスターを更新すると、デフォルトでは、Kubernetes コマンド (例: `kubectl`) だけでなく`eksctl delete`

cluster、、eksctl utils write-kubeconfig、および場合によっては コマンドをクラスター VPC 内で実行eksctl utils update-kube-proxyする必要があります。

- これには、さまざまな AWS リソースへのいくつかの変更が必要です。詳細については、「クラスター API サーバーエンドポイント」を参照してください。
- ControlPlaneSecurityGroup に追加の CIDR 範囲を追加する vpc.extraCIDRs を指定して、VPC 外のサブネットが kubernetes API エンドポイントに到達できるようにします。同様に、 を指定vpc.extraIPv6CIDRsして IPv6 CIDR 範囲を追加することもできます。

以下は、 utilsサブコマンドを使用して Kubernetes API エンドポイントアクセスを設定する方法の例です。

```
eksctl utils update-cluster-vpc-config --cluster=<clustername> --private-access=true --
public-access=false
```

ClusterConfig ファイルを使用して設定を更新するには、以下を使用します。

```
eksctl utils update-cluster-vpc-config -f config.yaml --approve
```

フラグを渡さないと、現在の値が保持されることに注意してください。提案された変更に満足したら、 approveフラグを追加して実行中のクラスターに変更を加えます。

EKS Kubernetes Public API エンドポイントへのアクセスの制限

EKS クラスターのデフォルト作成では、Kubernetes API サーバーが公開されます。

この機能はパブリックエンドポイントにのみ適用されます。API サーバーエンドポイントのアクセス設定オプションは変更されず、クラスターがインターネットからアクセスできないようにパブリックエンドポイントを無効にするオプションもあります。(出典: https://github.com/aws/containers-roadmap/issues/108#issuecomment-552766489)

クラスターの作成時にパブリック API エンドポイントへのアクセスを CIDRs のセットに制限するには、 **publicAccessCIDRs**フィールドを設定します。

```
vpc:
  publicAccessCIDRs: ["1.1.1.1/32", "2.2.2.0/24"]
```

既存のクラスターの制限を更新するには、以下を使用します。

```
eksctl utils update-cluster-vpc-config --cluster=<cluster> 1.1.1.1/32,2.2.2.0/24
```

ClusterConfig ファイルを使用して制限を更新するには、 で新しい CIDRsを設定し**vpc.publicAccessCIDRs**、以下を実行します。

```
eksctl utils update-cluster-vpc-config -f config.yaml
```

> **Important**
>
> ノードグループを設定publicAccessCIDRsおよび作成する場合はprivateAccess、 に設定するtrueか、ノードの IPs をpublicAccessCIDRsリストに追加する必要があります。

アクセスが制限されているためにノードがクラスター API エンドポイントにアクセスできない場合、ノードがパブリックエンドポイントにアクセスできず、クラスターに参加できないcontext deadline exceededため、クラスターの作成は で失敗します。

クラスターの API サーバーエンドポイントアクセスとパブリックアクセス CIDRs の両方を 1 つのコマンドで更新するには、以下を実行します。

```
eksctl utils update-cluster-vpc-config --cluster=<cluster> --public-access=true --
private-access=true --public-access-cidrs=1.1.1.1/32,2.2.2.0/24
```

設定ファイルを使用して設定を更新するには:

```
vpc:
  clusterEndpoints:
    publicAccess:  <true|false>
    privateAccess: <true|false>
  publicAccessCIDRs: ["1.1.1.1/32"]
```

```
eksctl utils update-cluster-vpc-config --cluster=<cluster> -f config.yaml
```

コントロールプレーンサブネットとセキュリティグループの更新

このドキュメントでは、最初の作成後に EKS クラスターのコントロールプレーンのネットワーク設定を変更する方法について説明します。これには、コントロールプレーンサブネットとセキュリティグループの更新が含まれます。

コントロールプレーンサブネットの更新

eksctl を使用してクラスターを作成すると、一連のパブリックサブネットとプライベートサブネットが作成され、EKS API に渡されます。EKS は、これらのサブネットに 2～4 個のクロスアカウント Elastic Network Interface (ENIs) を作成し、EKS マネージド Kubernetes コントロールプレーンと VPC 間の通信を可能にします。

EKS コントロールプレーンで使用されるサブネットを更新するには、以下を実行します。

```
eksctl utils update-cluster-vpc-config --cluster=<cluster> --control-plane-subnet-
ids=subnet-1234,subnet-5678
```

設定ファイルを使用して設定を更新するには:

```
apiVersion: eksctl.io/v1alpha5
kind: ClusterConfig
metadata:
  name: cluster
  region: us-west-2

vpc:
  controlPlaneSubnetIDs: [subnet-1234, subnet-5678]
```

```
eksctl utils update-cluster-vpc-config -f config.yaml
```

--approve フラグがない場合、eksctl は提案された変更のみをログに記録します。提案された変更に満足したら、 コマンドを --approveフラグで再実行します。

コントロールプレーンセキュリティグループの更新

コントロールプレーンとワーカーノード間のトラフィックを管理するために、EKS は、EKS によってプロビジョニングされたクロスアカウントネットワークインターフェイスに適用される追加のセキュリティグループを渡すことをサポートしています。EKS コントロールプレーンのセキュリティグループを更新するには、以下を実行します。

```
eksctl utils update-cluster-vpc-config --cluster=<cluster> --control-plane-security-
group-ids=sg-1234,sg-5678
```

設定ファイルを使用して設定を更新するには:

```
apiVersion: eksctl.io/v1alpha5
kind: ClusterConfig
metadata:
  name: cluster
  region: us-west-2

vpc:
  controlPlaneSecurityGroupIDs: [sg-1234, sg-5678]
```

```
eksctl utils update-cluster-vpc-config -f config.yaml
```

クラスターのコントロールプレーンサブネットとセキュリティグループの両方を更新するには、以下を実行します。

```
eksctl utils update-cluster-vpc-config --cluster=<cluster> --control-plane-subnet-
ids=<> --control-plane-security-group-ids=<>
```

設定ファイルを使用して両方のフィールドを更新するには:

```
apiVersion: eksctl.io/v1alpha5
kind: ClusterConfig
metadata:
  name: cluster
  region: us-west-2

vpc:
  controlPlaneSubnetIDs: [subnet-1234, subnet-5678]
  controlPlaneSecurityGroupIDs: [sg-1234, sg-5678]
```

```
eksctl utils update-cluster-vpc-config -f config.yaml
```

完全な例については、「cluster-subnets-sgs.yaml」を参照してください。

--approve フラグがない場合、eksctl は提案された変更のみをログに記録します。提案された変更に満足したら、 コマンドを --approveフラグで再実行します。

IPv6 のサポート

IP ファミリーを定義する

が vpc eksctlを作成するときに、使用する IP バージョンを定義できます。以下のオプションを設定できます。

- IPv4
- IPv6

デフォルト値は IPv4 です。

これを定義するには、次の例を使用します。

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

metadata:
  name: my-test
  region: us-west-2
  version: "1.21"

kubernetesNetworkConfig:
  ipFamily: IPv6 # or IPv4

addons:
  - name: vpc-cni
  - name: coredns
  - name: kube-proxy

iam:
  withOIDC: true
```

> **Note**
>
> この設定は CLI フラグではなく、設定ファイルでのみ行われます。

IPv6 を使用する場合は、次の要件を設定する必要があります。

- OIDC が有効になっている
- マネージドアドオンは上記のように定義されています
- クラスターバージョンは => 1.21 である必要があります
- vpc-cni アドオンのバージョンは => 1.10.0 である必要があります
- セルフマネージド型ノードグループは IPv6 クラスターではサポートされていません
- マネージド型ノードグループは、非所有 IPv6 クラスターではサポートされていません
- `vpc.nat` および `serviceIPv4CIDR`フィールドは ipv6 クラスター用に eksctl によって作成され、サポートされていない設定オプションです。
- AutoAllocateIPv6 は IPv6 と一緒にはサポートされていません
- IPv6 クラスターの場合、vpc-cni の IAM ロールにはIPv6 モードに必要な IAM ポリシーが関連付けられている必要があります

プライベートネットワーキングは IPv6 IP ファミリーでも実行できます。EKS プライベートクラスターで説明されている指示に従ってください。

IAM

この章には、AWS IAM の使用に関する情報が含まれています。

トピック:

- the section called “IAM ユーザーとロールを管理する”
  - IAM ユーザーとロールのマッピングを管理し、EKS クラスターへのアクセスを制御する
  - クラスター設定ファイルまたは CLI コマンドを使用して IAM ID マッピングを設定する
- the section called “サービスアカウントの IAM ロール”
  - 他の AWS のサービスを使用する Amazon EKS で実行されているアプリケーションのきめ細かなアクセス許可を管理する
  - eksctl を使用して IAM ロールと Kubernetes サービスアカウントのペアを作成および設定する
  - EKS クラスターの IAM OpenID Connect プロバイダーを有効にして、サービスアカウントの IAM ロールを有効にする
- the section called “IAM アクセス許可の境界”
  - アクセス許可の境界を設定して、IAM エンティティ (ユーザーまたはロール) に付与されるアクセス許可の最大数を制御する
- the section called “EKS Pod Identity の関連付け”
  - 推奨ポッド ID 関連付けを使用して EKS アドオンの IAM アクセス許可を設定する
  - Kubernetes アプリケーションがクラスター外の AWS サービスに接続するために必要な IAM アクセス許可を受信できるようにする
  - 複数の EKS クラスターで IAM ロールとサービスアカウントを自動化するプロセスを簡素化する
- the section called “IAM ポリシー”
  - Image Builder、自動スケーラー、外部 DNS、証明書マネージャーなどのさまざまなアドオンポリシーのサポートなど、EKS ノードグループの IAM ポリシーを管理します。
  - 追加のアクセス許可のために、カスタムインスタンスロールまたはインラインポリシーをノードグループにアタッチします。
  - ARN ごとに特定の AWS 管理ポリシーをノードグループにアタッチし、AmazonEKSWorkerNodePolicy や AmazonEKS_CNI_Policy などの必要なポリシーが含まれていることを確認します。
- the section called “最小 IAM ポリシー”

- ロードバランサー、自動スケーリンググループ、CloudWatch モニタリングなどの AWS EC2 リソースを管理する
- AWS CloudFormation スタックの作成と管理
- Amazon Elastic Kubernetes Service (EKS) クラスター、ノードグループ、および IAM ロールやポリシーなどの関連リソースを管理する

最小 IAM ポリシー

このドキュメントでは、eksctl の主なユースケースを実行するために必要な最小限の IAM ポリシーについて説明します。これらは、統合テストの実行に使用されるものです。

> **Note**
>
> を独自の <account_id>に置き換えることを忘れないでください。

> **Note**
>
> AWS 管理ポリシーは、AWS によって作成および管理されます。AWS 管理ポリシーで定義されているアクセス許可は変更できません。

AmazonEC2FullAccess (AWS 管理ポリシー)

AmazonEC2FullAccess ポリシー定義を表示します。

AWSCloudFormationFullAccess (AWS 管理ポリシー)

AWSCloudFormationFullAccess ポリシー定義を表示します。

EksAllAccess

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "eks:*",
            "Resource": "*"
```

```
        },
        {
            "Action": [
                "ssm:GetParameter",
                "ssm:GetParameters"
            ],
            "Resource": [
                "arn:aws:ssm:*:123456789012:parameter/aws/*",
                "arn:aws:ssm:*::parameter/aws/*"
            ],
            "Effect": "Allow"
        },
        {
             "Action": [
               "kms:CreateGrant",
               "kms:DescribeKey"
             ],
             "Resource": "*",
             "Effect": "Allow"
        },
        {
             "Action": [
               "logs:PutRetentionPolicy"
             ],
             "Resource": "*",
             "Effect": "Allow"
        }
    ]
}
```

IamLimitedAccess

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "iam:CreateInstanceProfile",
                "iam:DeleteInstanceProfile",
                "iam:GetInstanceProfile",
                "iam:RemoveRoleFromInstanceProfile",
                "iam:GetRole",
```

```
                "iam:CreateRole",
                "iam:DeleteRole",
                "iam:AttachRolePolicy",
                "iam:PutRolePolicy",
                "iam:UpdateAssumeRolePolicy",
                "iam:AddRoleToInstanceProfile",
                "iam:ListInstanceProfilesForRole",
                "iam:PassRole",
                "iam:DetachRolePolicy",
                "iam:DeleteRolePolicy",
                "iam:GetRolePolicy",
                "iam:GetOpenIDConnectProvider",
                "iam:CreateOpenIDConnectProvider",
                "iam:DeleteOpenIDConnectProvider",
                "iam:TagOpenIDConnectProvider",
                "iam:ListAttachedRolePolicies",
                "iam:TagRole",
                "iam:UntagRole",
                "iam:GetPolicy",
                "iam:CreatePolicy",
                "iam:DeletePolicy",
                "iam:ListPolicyVersions"
            ],
            "Resource": [
                "arn:aws:iam::123456789012:instance-profile/eksctl-*",
                "arn:aws:iam::123456789012:role/eksctl-*",
                "arn:aws:iam::123456789012:policy/eksctl-*",
                "arn:aws:iam::123456789012:oidc-provider/*",
                "arn:aws:iam::123456789012:role/aws-service-role/eks-
nodegroup.amazonaws.com/AWSServiceRoleForAmazonEKSNodegroup",
                "arn:aws:iam::123456789012:role/eksctl-managed-*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "iam:GetRole",
                "iam:GetUser"
            ],
            "Resource": [
                "arn:aws:iam::123456789012:role/*",
                "arn:aws:iam::123456789012:user/*"
            ]
        },
```

```
        {
            "Effect": "Allow",
            "Action": [
                "iam:CreateServiceLinkedRole"
            ],
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "iam:AWSServiceName": [
                        "eks.amazonaws.com",
                        "eks-nodegroup.amazonaws.com",
                        "eks-fargate.amazonaws.com"
                    ]
                }
            }
        }
    ]
}
```

IAM 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のポリシーが IAM エンティティに付与できるアクセス許可の上限が設定されている高度な AWS IAM 機能です。これらのエンティティはユーザーまたはロールです。エンティティのアクセス許可の境界が設定されている場合、そのエンティティはアイデンティティベースのポリシーとアクセス許可の境界の両方で許可されているアクションのみを実行できます。

アクセス許可の境界を指定して、eksctl によって作成されたすべてのアイデンティティベースのエンティティがその境界内に作成されるようにできます。この例では、eksctl によって作成されたさまざまなアイデンティティベースのエンティティにアクセス許可の境界を提供する方法を示します。

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

metadata:
  name: cluster-17
  region: us-west-2

iam:
  withOIDC: true
  serviceRolePermissionsBoundary: "arn:aws:iam::11111:policy/entity/boundary"
```

```
  fargatePodExecutionRolePermissionsBoundary: "arn:aws:iam::11111:policy/entity/
boundary"
  serviceAccounts:
    - metadata:
        name: s3-reader
      attachPolicyARNs:
      - "arn:aws:iam::aws:policy/AmazonS3ReadOnlyAccess"
      permissionsBoundary: "arn:aws:iam::11111:policy/entity/boundary"

nodeGroups:
  - name: "ng-1"
    desiredCapacity: 1
    iam:
      instanceRolePermissionsBoundary: "arn:aws:iam::11111:policy/entity/boundary"
```

> **Warning**
>
> ロール ARN とアクセス許可の境界の両方を指定することはできません。

VPC CNI アクセス許可境界の設定

OIDC が有効な eksctl でクラスターを作成すると、セキュリティ上の理由から VPC-CNI `iamserviceaccount`用の が自動的に作成されます。アクセス許可の境界を追加する場合は、設定ファイル`iamserviceaccount`で を手動で指定する必要があります。

```
iam:
  serviceAccounts:
    - metadata:
        name: aws-node
        namespace: kube-system
      attachPolicyARNs:
      - "arn:aws:iam::<arn>:policy/AmazonEKS_CNI_Policy"
      permissionsBoundary: "arn:aws:iam::11111:policy/entity/boundary"
```

IAM ポリシー

インスタンスロールをノードグループにアタッチできます。ノードで実行されているワークロードは、ノードから IAM アクセス許可を受け取ります。mroe の詳細については、Amazon EC2 の IAM ロール」を参照してください。

このページには、eksctl で使用できる事前定義された IAM ポリシーテンプレートが一覧表示されます。これらのテンプレートは、カスタム IAM ポリシーを手動で作成することなく、EKS ノードに適切な AWS サービスアクセス許可を付与するプロセスを簡素化します。

サポートされている IAM アドオンポリシー

サポートされているすべてのアドオンポリシーの例:

```
nodeGroups:
  - name: ng-1
    instanceType: m5.xlarge
    desiredCapacity: 1
    iam:
      withAddonPolicies:
        imageBuilder: true
        autoScaler: true
        externalDNS: true
        certManager: true
        appMesh: true
        appMeshPreview: true
        ebs: true
        fsx: true
        efs: true
        awsLoadBalancerController: true
        xRay: true
        cloudWatch: true
```

Image Builder ポリシー

このimageBuilderポリシーでは、ECR (Elastic Container Registry) へのフルアクセスを許可します。これは、イメージを ECR にプッシュする必要がある CI サーバーを構築する場合などに便利です。

EBS ポリシー

このebsポリシーは、新しい EBS CSI (Elastic Block Store Container Storage Interface) ドライバーを有効にします。

Cert Manager ポリシー

このcertManagerポリシーにより、DNS01 チャレンジを解決するために Route 53 にレコードを追加できます。詳細については、こちらを参照してください。

カスタムインスタンスロールの追加

この例では、別のクラスターから既存の IAM インスタンスロールを再利用するノードグループを作成します。

```
apiVersion: eksctl.io/v1alpha4
kind: ClusterConfig
metadata:
  name: test-cluster-c-1
  region: eu-north-1

nodeGroups:
  - name: ng2-private
    instanceType: m5.large
    desiredCapacity: 1
    iam:
      instanceProfileARN: "arn:aws:iam::123:instance-profile/eksctl-test-cluster-a-3-
nodegroup-ng2-private-NodeInstanceProfile-Y4YKHLNINMXC"
      instanceRoleARN: "arn:aws:iam::123:role/eksctl-test-cluster-a-3-nodegroup-
NodeInstanceRole-DNGMQTQHQHBJ"
```

インラインポリシーのアタッチ

```
nodeGroups:
  - name: my-special-nodegroup
    iam:
      attachPolicy:
        Version: "2012-10-17"
        Statement:
        - Effect: Allow
          Action:
          - 's3:GetObject'
          Resource: 'arn:aws:s3:::example-bucket/*'
```

ARN によるポリシーのアタッチ

```
nodeGroups:
  - name: my-special-nodegroup
    iam:
      attachPolicyARNs:
        - arn:aws:iam::aws:policy/AmazonEKSWorkerNodePolicy
```

```
        - arn:aws:iam::aws:policy/AmazonEKS_CNI_Policy
        - arn:aws:iam::aws:policy/AmazonEC2ContainerRegistryPullOnly
        - arn:aws:iam::aws:policy/ElasticLoadBalancingFullAccess
        - arn:aws:iam::1111111111:policy/kube2iam
      withAddonPolicies:
        autoScaler: true
        imageBuilder: true
```

> **Warning**
>
> ノードグループに が含まれているattachPolicyARNs場合、AmazonEC2ContainerRegistryPullOnlyこの例では AmazonEKSWorkerNodePolicyAmazonEKS_CNI_Policyや などのデフォルトのノードポリシーも含める必要があります。

IAM ユーザーとロールを管理する

> **Note**
>
> AWS は、ConfigMap aws-auth the section called “EKS Pod Identity の関連付け”から への移行を提案します。

EKS クラスターは、IAM ユーザーとロールを使用してクラスターへのアクセスを制御します。ルールは設定マップに実装されます

CLI コマンドを使用して ConfigMap を編集する

という名前aws-auth。 eksctlは、この設定マップを読み取って編集するためのコマンドを提供します。

すべての ID マッピングを取得します。

```
eksctl get iamidentitymapping --cluster <clusterName> --region=<region>
```

arn に一致するすべての ID マッピングを取得します。

```
eksctl get iamidentitymapping --cluster <clusterName> --region=<region> --arn
 arn:aws:iam::123456:role/testing-role
```

ID マッピングを作成します。

```
 eksctl create iamidentitymapping --cluster  <clusterName> --region=<region> --arn
 arn:aws:iam::123456:role/testing --group system:masters --username admin
```

ID マッピングを削除します。

```
eksctl delete iamidentitymapping --cluster  <clusterName> --region=<region> --arn
 arn:aws:iam::123456:role/testing
```

> **Note**
>
> 上記のコマンドは、 が指定されていない限り、単一のマッピング FIFO を削除します。指定されていない場合--all、一致するものはすべて削除されます。この口ールに一致するマッピングがさらに見つかった場合に警告します。

アカウントマッピングを作成します。

```
 eksctl create iamidentitymapping --cluster  <clusterName> --region=<region> --account
 user-account
```

アカウントマッピングを削除します。

```
 eksctl delete iamidentitymapping --cluster  <clusterName> --region=<region> --account
 user-account
```

ClusterConfig ファイルを使用して ConfigMap を編集する

ID マッピングは ClusterConfig で指定することもできます。

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

metadata:
```

```
  name: cluster-with-iamidentitymappings
  region: us-east-1

iamIdentityMappings:
  - arn: arn:aws:iam::000000000000:role/myAdminRole
    groups:
      - system:masters
    username: admin
    noDuplicateARNs: true # prevents shadowing of ARNs

  - arn: arn:aws:iam::000000000000:user/myUser
    username: myUser
    noDuplicateARNs: true # prevents shadowing of ARNs

  - serviceName: emr-containers
    namespace: emr # serviceName requires namespace

  - account: "000000000000" # account must be configured with no other options

nodeGroups:
  - name: ng-1
    instanceType: m5.large
    desiredCapacity: 1
```

```
eksctl create iamidentitymapping -f cluster-with-iamidentitymappings.yaml
```

サービスアカウントの IAM ロール

> **Tip**
>
> eksctl は、EKS Pod Identity Associations を介してアプリケーションを実行する EKS へのきめ細かなアクセス許可の設定をサポートします。

Amazon EKS は、ここでサービスアカウントのロール (IRSA)] をサポートしています。これにより、クラスターオペレーターは AWS IAM ロールを Kubernetes サービスアカウントにマッピングできます。

これにより、EKS で実行され、他の AWS のサービスを使用するアプリケーションのアクセス許可をきめ細かく管理できます。これらは、S3、その他のデータサービス

(RDS、MQ、STS、DynamoDB)、または AWS Load Balancer コントローラーや ExternalDNS などの Kubernetes コンポーネントを使用するアプリケーションです。

を使用して IAM ロールとサービスアカウントのペアを簡単に作成できますeksctl。

> **Note**
>
> インスタンスロールを使用していて、代わりに IRSA の使用を検討している場合は、この 2 つを混在させないでください。

仕組み

EKS が公開する IAM OpenID Connect プロバイダー (OIDC) を介して動作し、IAM ロールは IAM OIDC プロバイダー (特定の EKS クラスターに固有) を参照し、バインドされる Kubernetes サービスアカウントを参照して構築する必要があります。IAM ロールが作成されると、サービスアカウントはそのロールの ARN を注釈 () として含める必要がありますeks.amazonaws.com/role-arn。デフォルトでは、サービスアカウントはロール注釈を含めるように作成または更新されます。これは、 フラグ を使用して無効にできます--role-only。

EKS 内には、ポッドが使用するサービスアカウントの注釈に基づいて、AWS セッション認証情報をそれぞれロールのポッドに挿入するアドミッションコントローラーがあります。認証情報は AWS_ROLE_ARN & AWS_WEB_IDENTITY_TOKEN_FILE環境変数によって公開されます。最新バージョンの AWS SDK が使用されている場合 (正確なバージョンの詳細については、こちらを参照) 、アプリケーションはこれらの認証情報を使用します。

リソースeksctlの名前は iamserviceaccount で、IAM ロールとサービスアカウントのペアを表します。

CLI の使用

> **Note**
>
> サービスアカウントの IAM ロールには、Kubernetes バージョン 1.13 以降が必要です。

IAM OIDC プロバイダーはデフォルトでは有効になっていません。次のコマンドを使用して有効にすることも、設定ファイル (以下を参照) を使用することもできます。

```
eksctl utils associate-iam-oidc-provider --cluster=<clusterName>
```

クラスターに IAM OIDC プロバイダーを関連付けたら、サービスアカウントにバインドされた IAM ロールを作成するには、以下を実行します。

```
eksctl create iamserviceaccount --cluster=<clusterName> --name=<serviceAccountName> --
namespace=<serviceAccountNamespace> --attach-policy-arn=<policyARN>
```

> **Note**
>
> 複数のポリシーを使用するには、`--attach-policy-arn`複数回指定できます。

具体的には、以下を実行して、S3 への読み取り専用アクセス権を持つサービスアカウントを作成できます。

```
eksctl create iamserviceaccount --cluster=<clusterName> --name=s3-read-only --attach-
policy-arn=arn:aws:iam::aws:policy/AmazonS3ReadOnlyAccess
```

デフォルトでは、`default`名前空間に作成されますが、次のような他の名前空間を指定できます。

```
eksctl create iamserviceaccount --cluster=<clusterName> --name=s3-read-only --
namespace=s3-app --attach-policy-arn=arn:aws:iam::aws:policy/AmazonS3ReadOnlyAccess
```

> **Note**
>
> 名前空間がまだ存在しない場合は、作成されます。

サービスアカウントが既にクラスターに作成されている場合 (IAM ロールなし)、 `--override-existing-serviceaccounts`フラグを使用する必要があります。

カスタムタグ付けは、 を指定して IAM ロールに適用することもできます`--tags`。

```
eksctl create iamserviceaccount --cluster=<clusterName> --name=<serviceAccountName> --
tags "Owner=John Doe,Team=Some Team"
```

CloudFormation は、ランダムな文字列を含むロール名を生成します。事前定義されたロール名が必要な場合は、 を指定できます--role-name。

```
eksctl create iamserviceaccount --cluster=<clusterName> --name=<serviceAccountName> --
role-name "custom-role-name"
```

サービスアカウントが helm などの他のツールによって作成および管理されている場合は、 を使用して競合を防止--role-onlyします。もう 1 つのツールは、ロール ARN 注釈の維持を担当します。--override-existing-serviceaccounts は roleOnly/--role-only サービスアカウントには影響せず、ロールは常に作成されます。

```
eksctl create iamserviceaccount --cluster=<clusterName> --name=<serviceAccountName> --
role-only --role-name=<customRoleName>
```

サービスアカウントで使用する既存のロールがある場合は、ポリシーを指定する代わりに --attach-role-arnフラグを指定できます。ロールが指定されたサービスアカウントでのみ引き受けられるようにするには、ここで関係ポリシードキュメントを設定する必要があります〕。

```
eksctl create iamserviceaccount --cluster=<clusterName> --name=<serviceAccountName> --
attach-role-arn=<customRoleARN>
```

サービスアカウントロールのアクセス許可を更新するには、 を実行しますeksctl update iamserviceaccount。

> **Note**
>
> eksctl delete iamserviceaccount は、 によって作成されていない場合ServiceAccountsでも Kubernetes を削除しますeksctl。

設定ファイルでの使用

設定ファイルiamserviceaccountsを使用して を管理するには、 で必要なアカウントを設定iam.withOIDC: trueして一覧表示しますiam.serviceAccount。

すべてのコマンドは をサポートしており--config-file、ノードグループと同じ方法で iamservice アカウントを管理できます。eksctl create iamserviceaccount コマンドは --includeと --excludeフラグをサポートします (これらの仕組みの詳細については、このセクショ

ンを参照してください)。また、 eksctl delete iamserviceaccount コマンドは --only-missingもサポートしているため、ノードグループと同じ方法で削除を実行できます。

> **Note**
>
> IAM サービスアカウントは名前空間内でスコープされます。つまり、同じ名前の 2 つのサービスアカウントが別の名前空間に存在する可能性があります。したがって、 --include--excludeフラグの一部としてサービスアカウントを一意に定義するには、 namespace/name形式で名前文字列を渡す必要があります。例:

```
eksctl create iamserviceaccount --config-file=<path> --include backend-apps/s3-reader
```

を有効にするオプションwellKnownPoliciesは、ポリシーのリストの短縮としてcert-manager、 cluster-autoscalerや などのよく知られたユースケースで IRSA を使用する場合に付属しています。

サポートされている既知のポリシーやその他のプロパティserviceAccountsは、設定スキーマに記載されています。

では、次の設定例を使用しますeksctl create cluster。

```
# An example of ClusterConfig with IAMServiceAccounts:
---
apiVersion: eksctl.io/v1alpha5
kind: ClusterConfig

metadata:
  name: cluster-13
  region: us-west-2

iam:
  withOIDC: true
  serviceAccounts:
  - metadata:
      name: s3-reader
      # if no namespace is set, "default" will be used;
      # the namespace will be created if it doesn't exist already
      namespace: backend-apps
      labels: {aws-usage: "application"}
```

```
    attachPolicyARNs:
    - "arn:aws:iam::aws:policy/AmazonS3ReadOnlyAccess"
    tags:
      Owner: "John Doe"
      Team: "Some Team"
  - metadata:
      name: cache-access
      namespace: backend-apps
      labels: {aws-usage: "application"}
    attachPolicyARNs:
    - "arn:aws:iam::aws:policy/AmazonDynamoDBReadOnlyAccess"
    - "arn:aws:iam::aws:policy/AmazonElastiCacheFullAccess"
  - metadata:
      name: cluster-autoscaler
      namespace: kube-system
      labels: {aws-usage: "cluster-ops"}
    wellKnownPolicies:
      autoScaler: true
    roleName: eksctl-cluster-autoscaler-role
    roleOnly: true
  - metadata:
      name: some-app
      namespace: default
    attachRoleARN: arn:aws:iam::123:role/already-created-role-for-app
nodeGroups:
  - name: "ng-1"
    tags:
      # EC2 tags required for cluster-autoscaler auto-discovery
      k8s.io/cluster-autoscaler/enabled: "true"
      k8s.io/cluster-autoscaler/cluster-13: "owned"
    desiredCapacity: 1
```

これらのフィールドを設定せずにクラスターを作成する場合は、次のコマンドを使用して必要なものをすべて有効にできます。

```
eksctl utils associate-iam-oidc-provider --config-file=<path>
eksctl create iamserviceaccount --config-file=<path>
```

詳細情報

- サービスアカウントのきめ細かな IAM ロールの紹介
- EKS ユーザーガイド - サービスアカウントの IAM ロール

- IAM ユーザーとロールを Kubernetes RBAC ロールにマッピングする

EKS Pod Identity の関連付け

AWS EKS は、クラスター管理者がクラスター外の AWS サービスに接続するために必要な IAM アクセス許可を受け取るように Kubernetes アプリケーションを設定するために、Pod Identity Association と呼ばれる新しい拡張メカニズムを導入しました。Pod Identity Association は IRSA を活用しますが、EKS API から直接設定できるため、IAM API を完全に使用する必要がなくなります。

その結果、IAM ロールは OIDC プロバイダーを参照する必要がなくなり、1 つのクラスターに関連付けられなくなります。つまり、IAM ロールを複数の EKS クラスターで使用できるようになりました。新しいクラスターが作成されるたびにロールの信頼ポリシーを更新する必要はありません。これにより、ロールの重複が不要になり、IRSA を完全に自動化するプロセスが簡素化されます。

前提条件

バックグラウンドでは、ポッドアイデンティティの関連付けの実装は、ワーカーノードでデーモンセットとしてエージェントを実行しています。クラスターで前提条件エージェントを実行するために、EKS は EKS Pod Identity Agent と呼ばれる新しいアドオンを提供します。したがって、ポッド ID の関連付け (一般的に と `eksctl`) を作成するには、`eks-pod-identity-agent`クラスターにアドオンがプリインストールされている必要があります。このアドオンは、他のサポートされているアドオンと同じ方法で `eksctl` を使用して作成できます。

```
eksctl create addon --cluster my-cluster --name eks-pod-identity-agent
```

さらに、ポッド ID 関連付けの作成時に既存の IAM ロールを使用する場合は、新しく導入された EKS サービスプリンシパル () を信頼するようにロールを設定する必要がありますpods.eks.amazonaws.com。IAM 信頼ポリシーの例を以下に示します。

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

```
                "sts:AssumeRole",
                "sts:TagSession"
            ]
        }
    ]
}
```

代わりに、既存のロールの ARN を create コマンドに提供しない場合、 eksctlはバックグラウンドでロールを作成し、上記の信頼ポリシーを設定します。

Pod Identity の関連付けの作成

ポッド ID の関連付けを操作するために、 eksctlは iam.podIdentityAssociationsに新しいフィールドを追加しました。例:

```
iam:
  podIdentityAssociations:
  - namespace: <string> #required
    serviceAccountName: <string> #required
    createServiceAccount: true #optional, default is false
    roleARN: <string> #required if none of permissionPolicyARNs, permissionPolicy and
 wellKnownPolicies is specified. Also, cannot be used together with any of the three
 other referenced fields.
    roleName: <string> #optional, generated automatically if not provided, ignored if
 roleARN is provided
    permissionPolicy: {} #optional
    permissionPolicyARNs: [] #optional
    wellKnownPolicies: {} #optional
    permissionsBoundaryARN: <string> #optional
    tags: {} #optional
```

完全な例については、pod-identity-associations.yaml を参照してください。

> **Note**
>
> permissionPolicy がインラインポリシードキュメントとして使用されるのとは別に、他のすべてのフィールドには CLI フラグが付きます。

ポッド ID の関連付けは、次の方法で作成できます。クラスターの作成時に、設定ファイルの一部として目的のポッド ID の関連付けを指定し、以下を実行します。

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

クラスターの作成後、 などの設定ファイルを使用します。

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

OR CLI フラグの使用例:

```
eksctl create podidentityassociation \
    --cluster my-cluster \
    --namespace default \
    --service-account-name s3-reader \
    --permission-policy-arns="arn:aws:iam::111122223333:policy/permission-policy-1,
 arn:aws:iam::111122223333:policy/permission-policy-2" \
    --well-known-policies="autoScaler,externalDNS" \
    --permissions-boundary-arn arn:aws:iam::111122223333:policy/permissions-boundary
```

> **Note**
>
> 一度にサービスアカウントに関連付けることができる IAM ロールは 1 つだけです。したがって、同じサービスアカウントに 2 番目のポッド ID の関連付けを作成しようとすると、エラーが発生します。

Pod Identity の関連付けの取得

特定のクラスターのすべてのポッド ID の関連付けを取得するには、次のいずれかのコマンドを実行します。

```
eksctl get podidentityassociation -f config.yaml
```

OR

```
eksctl get podidentityassociation --cluster my-cluster
```

さらに、特定の名前空間内のポッド ID の関連付けのみを取得するには、 `--namespace`フラグを使用します。例:

```
eksctl get podidentityassociation --cluster my-cluster --namespace default
```

最後に、特定の K8s サービスアカウントに対応する単一の関連付けを取得するには、上記のコマンド--service-account-nameに も含めます。つまり、

```
eksctl get podidentityassociation --cluster my-cluster --namespace default --service-
account-name s3-reader
```

Pod Identity の関連付けの更新

1 つ以上のポッド ID 関連付けの IAM ロールを更新するには、新しい roleARN(s)を 設定ファイルに渡します。例:

```
iam:
  podIdentityAssociations:
    - namespace: default
      serviceAccountName: s3-reader
      roleARN: new-role-arn-1
    - namespace: dev
      serviceAccountName: app-cache-access
      roleARN: new-role-arn-2
```

と を実行します。

```
eksctl update podidentityassociation -f config.yaml
```

OR (単一の関連付けを更新するため) は CLI フラグ--role-arnを介して新しい を渡します。

```
eksctl update podidentityassociation --cluster my-cluster --namespace default --
service-account-name s3-reader --role-arn new-role-arn
```

Pod Identity の関連付けの削除

1 つ以上のポッド ID の関連付けを削除するには、 namespace(s)と serviceAccountName(s)を設定ファイルに渡します。例:

```
iam:
  podIdentityAssociations:
    - namespace: default
```

```
      serviceAccountName: s3-reader
    - namespace: dev
      serviceAccountName: app-cache-access
```

と を実行します。

```
eksctl delete podidentityassociation -f config.yaml
```

OR (単一の関連付けを削除) は、 `--namespace`と を CLI フラグ`--service-account-name`経由で渡します。

```
eksctl delete podidentityassociation --cluster my-cluster --namespace default --
service-account-name s3-reader
```

EKS アドオンによるポッド ID の関連付けのサポート

EKS アドオンは、EKS Pod Identity Associations を介した IAM アクセス許可の受信もサポートしています。設定ファイルには、これらを設定できる 3 つのフィールド`addon.podIdentityAssociations`、`addonsConfig.autoApplyPodIdentityAssociations`、が公開されています`addon.useDefaultPodIdentityAssociations`。を使用して目的のポッド ID の関連付けを明示的に設定するか`addon.podIdentityAssociations`、`addonsConfig.autoApplyPodIdentityAssociations`または を使用して推奨ポッド ID 設定`eksctl`を自動的に解決 (および適用) できます`addon.useDefaultPodIdentityAssociations`。

> **Note**
>
> すべての EKS アドオンが起動時にポッド ID の関連付けをサポートするわけではありません。この場合、必要な IAM アクセス許可は IRSA 設定を使用して引き続き提供されます。

IAM アクセス許可を使用したアドオンの作成

IAM アクセス許可を必要とするアドオンを作成する場合、 `eksctl` はまず、ポッド ID の関連付けまたは IRSA 設定が設定ファイルの一部として明示的に設定されているかどうかを確認し、設定されている場合は、そのいずれかを使用してアドオンのアクセス許可を設定します。例:

```
addons:
- name: vpc-cni
```

```
  podIdentityAssociations:
  - serviceAccountName: aws-node
    permissionPolicyARNs: ["arn:aws:iam::aws:policy/AmazonEKS_CNI_Policy"]
```

と の実行

```
eksctl create addon -f config.yaml
2024-05-13 15:38:58 [#] pod identity associations are set for "vpc-cni" addon; will use
 these to configure required IAM permissions
```

> **Note**
>
> ポッド ID と IRSA の両方を同時に設定することは許可されず、検証エラーが発生します。

ポッド ID をサポートする EKS アドオンの場合、 eksctlはアドオンの作成時に推奨される IAM アクセス許可を自動的に設定するオプションを提供します。これは、設定ファイルaddonsConfig.autoApplyPodIdentityAssociations: trueで を設定するだけで実現できます。例:

```
addonsConfig:
  autoApplyPodIdentityAssociations: true
# bear in mind that if either pod identity or IRSA configuration is explicitly set in
 the config file,
# or if the addon does not support pod identities,
# addonsConfig.autoApplyPodIdentityAssociations won't have any effect.
addons:
- name: vpc-cni
```

と の実行

```
eksctl create addon -f config.yaml
2024-05-13 15:38:58 [#] "addonsConfig.autoApplyPodIdentityAssociations" is set to true;
 will lookup recommended pod identity configuration for "vpc-cni" addon
```

同様に、 などの CLI フラグを使用しても同じことができます。

```
eksctl create addon --cluster my-cluster --name vpc-cni --auto-apply-pod-identity-
associations
```

推奨される IAM ポリシーでポッド ID を使用するように既存のアドオンを移行するには、 を使用します。

```
addons:
- name: vpc-cni
  useDefaultPodIdentityAssociations: true
```

```
eksctl update addon -f config.yaml
```

IAM アクセス許可を使用したアドオンの更新

アドオンを更新する場合、 を指定するaddon.PodIdentityAssociationsと、更新オペレーションが完了した後にアドオンが持つべき状態の 1 つの信頼できるソースを表します。バックグラウンドでは、目的の状態を達成するためにさまざまなタイプのオペレーションが実行されます。つまり、

- 設定ファイルには存在するが、クラスターには存在しないポッドアイデンティティを作成する
- 設定ファイルから削除された既存のポッド ID と関連する IAM リソースを削除する
- 設定ファイルにも存在し、IAM アクセス許可のセットが変更された既存のポッド ID を更新する

> **Note**
>
> EKS アドオンが所有するポッド ID 関連付けのライフサイクルは、EKS アドオン API によって直接処理されます。

Amazon EKS アドオンで使用される関連付けに `eksctl update podidentityassociation` (IAM アクセス許可を更新するために) または `eksctl delete podidentityassociations` (関連付けを削除するために) を使用することはできません。代わりに、 `eksctl update addon`または `eksctl delete addon` が使用されます。

アドオンの初期ポッド ID 設定を分析することから、上記の例を見てみましょう。

```
eksctl get podidentityassociation --cluster my-cluster --namespace opentelemetry-
operator-system --output json
[
    {
        ...
        "ServiceAccountName": "adot-col-prom-metrics",
```

```
        "RoleARN": "arn:aws:iam::111122223333:role/eksctl-my-cluster-addon-adot-
podident-Role1-JwrGA4mn1Ny8",
        # OwnerARN is populated when the pod identity lifecycle is handled by the EKS
 Addons API
        "OwnerARN": "arn:aws:eks:us-west-2:111122223333:addon/my-cluster/adot/
b2c7bb45-4090-bf34-ec78-a2298b8643f6"
    },
    {
        ...
        "ServiceAccountName": "adot-col-otlp-ingest",
        "RoleARN": "arn:aws:iam::111122223333:role/eksctl-my-cluster-addon-adot-
podident-Role1-Xc7qVg5fgCqr",
        "OwnerARN": "arn:aws:eks:us-west-2:111122223333:addon/my-cluster/adot/
b2c7bb45-4090-bf34-ec78-a2298b8643f6"
    }
]
```

次に、次の設定を使用します。

```
addons:
- name: adot
  podIdentityAssociations:

  # For the first association, the permissions policy of the role will be updated
  - serviceAccountName: adot-col-prom-metrics
    permissionPolicyARNs:
    #- arn:aws:iam::aws:policy/AmazonPrometheusRemoteWriteAccess
    - arn:aws:iam::aws:policy/CloudWatchAgentServerPolicy

  # The second association will be deleted, as it's been removed from the config file
  #- serviceAccountName: adot-col-otlp-ingest
  #  permissionPolicyARNs:
  #  - arn:aws:iam::aws:policy/AWSXrayWriteOnlyAccess

  # The third association will be created, as it's been added to the config file
  - serviceAccountName: adot-col-container-logs
    permissionPolicyARNs:
    - arn:aws:iam::aws:policy/CloudWatchAgentServerPolicy
```

と の実行

```
eksctl update addon -f config.yaml
...
```

```
# updating the permission policy for the first association
2024-05-14 13:27:43 [#]  updating IAM resources stack "eksctl-my-cluster-addon-
adot-podidentityrole-adot-col-prom-metrics" for pod identity association "a-
reaxk2uz1iknwazwj"
2024-05-14 13:27:44 [#]  waiting for CloudFormation changeset "eksctl-opentelemetry-
operator-system-adot-col-prom-metrics-update-1715682463" for stack "eksctl-my-cluster-
addon-adot-podidentityrole-adot-col-prom-metrics"
2024-05-14 13:28:47 [#]  waiting for CloudFormation stack "eksctl-my-cluster-addon-
adot-podidentityrole-adot-col-prom-metrics"
2024-05-14 13:28:47 [#]  updated IAM resources stack "eksctl-my-cluster-addon-adot-
podidentityrole-adot-col-prom-metrics" for "a-reaxk2uz1iknwazwj"
# creating the IAM role for the second association
2024-05-14 13:28:48 [#]  deploying stack "eksctl-my-cluster-addon-adot-podidentityrole-
adot-col-container-logs"
2024-05-14 13:28:48 [#]  waiting for CloudFormation stack "eksctl-my-cluster-addon-
adot-podidentityrole-adot-col-container-logs"
2024-05-14 13:29:19 [#]  waiting for CloudFormation stack "eksctl-my-cluster-addon-
adot-podidentityrole-adot-col-container-logs"
# updating the addon, which handles the pod identity config changes behind the scenes
2024-05-14 13:29:19 [#]  updating addon
# deleting the IAM role for the third association
2024-05-14 13:29:19 [#]  deleting IAM resources for pod identity service account adot-
col-otlp-ingest
2024-05-14 13:29:20 [#]  will delete stack "eksctl-my-cluster-addon-adot-
podidentityrole-adot-col-otlp-ingest"
2024-05-14 13:29:20 [#]  waiting for stack "eksctl-my-cluster-addon-adot-
podidentityrole-adot-col-otlp-ingest" to get deleted
2024-05-14 13:29:51 [#]  waiting for CloudFormation stack "eksctl-my-cluster-addon-
adot-podidentityrole-adot-col-otlp-ingest"
2024-05-14 13:29:51 [#]  deleted IAM resources for addon adot
```

ポッド ID 設定が正しく更新されたことを確認するようになりました

```
eksctl get podidentityassociation --cluster my-cluster --output json
[
    {
        ...
        "ServiceAccountName": "adot-col-prom-metrics",
        "RoleARN": "arn:aws:iam::111122223333:role/eksctl-my-cluster-addon-adot-
podident-Role1-nQAlp0KktS2A",
        "OwnerARN": "arn:aws:eks:us-west-2:111122223333:addon/my-cluster/
adot/1ec7bb63-8c4e-ca0a-f947-310c4b55052e"
    },
```

```
    {
        ...
        "ServiceAccountName": "adot-col-otlp-ingest",
        "RoleARN": "arn:aws:iam::111122223333:role/eksctl-my-cluster-addon-adot-
podident-Role1-1k1XhAdziGzX",
        "OwnerARN": "arn:aws:eks:us-west-2:111122223333:addon/my-cluster/
adot/1ec7bb63-8c4e-ca0a-f947-310c4b55052e"
    }
]
```

アドオンからすべてのポッド ID の関連付けを削除するには、 []を明示的に に設定addon.PodIdentityAssociationsする必要があります。例:

```
addons:
- name: vpc-cni
  # omitting the `podIdentityAssociations` field from the config file,
  # instead of explicitly setting it to [], will result in a validation error
  podIdentityAssociations: []
```

と の実行

```
eksctl update addon -f config.yaml
```

IAM アクセス許可を持つアドオンの削除

アドオンを削除すると、アドオンに関連付けられているすべてのポッド ID も削除されます。クラスターを削除すると、すべてのアドオンで同じ効果が得られます。によって作成されたポッド ID の IAM ロールはeksctl、すべて削除されます。

既存の iamservice アカウントとアドオンをポッド ID の関連付けに移行する

サービスアカウントの既存の IAM ロールをポッド ID の関連付けに移行するための eksctlutils コマンドがあります。つまり、

```
eksctl utils migrate-to-pod-identity --cluster my-cluster --approve
```

バックグラウンドでは、 コマンドは次のステップを適用します。

- クラスターでまだアクティブでない場合は、eks-pod-identity-agentアドオンをインストールする
- iamserviceaccounts に関連付けられているすべての IAM ロールを識別する
- ポッド ID の関連付けをサポートする EKS アドオンに関連付けられているすべての IAM ロールを識別する
- 識別されたすべてのロールの IAM 信頼ポリシーを、新しい EKS サービスプリンシパルを指す追加の信頼されたエンティティで更新します (オプションで、出現する OIDC プロバイダーの信頼関係を削除します) 。
- iamserviceaccounts に関連付けられたフィルタリングされたロールのポッド ID 関連付けを作成する
- ポッド ID を使用して EKS アドオンを更新する (EKS API はバックグラウンドでポッド ID を作成します)

--approve フラグなしでコマンドを実行すると、上記のステップを反映した一連のタスクで構成される計画のみが出力されます。例:

```
[#]  (plan) would migrate 2 iamserviceaccount(s) and 2 addon(s) to pod identity
 association(s) by executing the following tasks
[#]  (plan)

3 sequential tasks: { install eks-pod-identity-agent addon,
    ## tasks for migrating the addons
    2 parallel sub-tasks: {
        2 sequential sub-tasks: {
            update trust policy for owned role "eksctl-my-cluster--Role1-DDuMLoeZ8weD",
            migrate addon aws-ebs-csi-driver to pod identity,
        },
        2 sequential sub-tasks: {
            update trust policy for owned role "eksctl-my-cluster--Role1-xYiPFOVp1aeI",
            migrate addon vpc-cni to pod identity,
        },
    },
    ## tasks for migrating the iamserviceaccounts
    2 parallel sub-tasks: {
        2 sequential sub-tasks: {
            update trust policy for owned role "eksctl-my-cluster--Role1-QLXqHcq9O1AR",
            create pod identity association for service account "default/sa1",
        },
        2 sequential sub-tasks: {
```

```
            update trust policy for unowned role "Unowned-Role1",
            create pod identity association for service account "default/sa2",
        },
    }
}
[#]  all tasks were skipped
[!]  no changes were applied, run again with '--approve' to apply the changes
```

既存の OIDC プロバイダーの信頼関係は、EKS アドオンに関連付けられた IAM ロールから常に削除されています。さらに、iamserviceaccounts に関連付けられた IAM ロールから既存の OIDC プロバイダーの信頼関係を削除するには、 `--remove-oidc-provider-trust-relationship`フラグを使用して コマンドを実行します。例:

```
eksctl utils migrate-to-pod-identity --cluster my-cluster --approve --remove-oidc-
provider-trust-relationship
```

クロスアカウント Pod Identity サポート

eksctl は EKS Pod Identity クロスアカウントアクセスをサポートしています。この機能を使用すると、EKS クラスターで実行されているポッドは、別の AWS アカウントの AWS リソースにアクセスできます。

使用方法

クロスアカウントアクセスでポッド ID の関連付けを作成するには、まずソース AWS アカウント (クラスターを使用) からターゲット AWS アカウント (クラスターがアクセスできるリソースを使用) へのアクセスを許可する IAM ロールとポリシーを設定します。この例については、「Amazon EKS Pod Identity がクロスアカウントアクセスを合理化する」を参照してください。

各アカウントで IAM ロールを設定したら、eksctl を使用してポッド ID の関連付けを作成します。

```
apiVersion: eksctl.io/v1alpha5
kind: ClusterConfig
metadata:
  # The cluster name and service account name should match the target
  # account policy's trust relationship.
  name: my-cluster
  region: us-west-2
  version: "1.32"

addons:
```

```
  - name: vpc-cni
  - name: coredns
  - name: kube-proxy
  - name: eks-pod-identity-agent

iam:
  podIdentityAssociations:
  - namespace: default
    serviceAccountName: demo-app-sa
    createServiceAccount: true
    # The source role in the same account as the cluster
    roleARN: arn:aws:iam::1111111111:role/account-a-role
    # The target role in a different account
    targetRoleARN: arn:aws:iam::2222222222:role/account-b-role
    # Optional: Disable session tags
    disableSessionTags: false

managedNodeGroups:
  - name: my-cluster
    instanceType: m6a.large
    privateNetworking: true
    minSize: 2
    desiredCapacity: 2
    maxSize: 3
```

その他のリファレンス

ポッド ID の EKS アドオンの公式 AWS Userdocs サポート

ポッドアイデンティティの関連付けに関する AWS 公式ブログ投稿

Pod Identity Associations の公式 AWS ユーザードキュメント

デプロイオプション

この章では、eksctl を使用して代替環境にデプロイされた EKS クラスターを管理する方法について説明します。

EKS デプロイオプションに関する最も正確な情報については、「EKS ユーザーガイド」の「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」を参照してください。

トピック:

- the section called “EKS Anywhere”
  - Amazon EKS Anywhere クラスターで eksctl を使用します。
  - Amazon EKS Anywhere は、AWS によって構築されたコンテナ管理ソフトウェアであり、オンプレミスとエッジで Kubernetes を簡単に実行および管理できます。
- the section called “AWS Outposts のサポート”
  - AWS Outposts の EKS クラスターで eksctl を使用します。
  - AWS Outposts は、AWS インフラストラクチャとサービスを事実上あらゆるオンプレミスまたはエッジロケーションに提供し、真に一貫したハイブリッドエクスペリエンスを実現するフルマネージドソリューションのファミリーです。
  - eksctl での AWS Outposts のサポートにより、EKS コントロールプレーンやワーカーノードなど、Kubernetes クラスター全体でローカルクラスターを作成し、AWS Outposts でローカルに実行できます。
- the section called “EKS ハイブリッドノード”
  - AWS クラウドで使用するのと同じ AWS EKS クラスター、機能、ツールを使用して、カスタマーマネージド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します。

EKS Anywhere

`eksctl` は、 サブコマンド `EKS Anywhere`を使用して という AWS の機能へのアクセスを提供します`eksctl anywhere`。これには、 に存在する`eksctl-anywhere`バイナリが必要です`PATH`。「eksctl-anywhere をインストールする」で説明されている手順に従ってインストールしてください。

完了したら、以下を実行して任意のコマンドを実行します。

```
eksctl anywhere version
v0.5.0
```

EKS Anywhere の詳細については、EKS Anywhere ウェブサイトを参照してください。

AWS Outposts のサポート

> **Warning**
>
> EKS Managed Nodegroups は Outposts ではサポートされていません。

既存のクラスターを AWS Outposts に拡張する

AWS リージョンで実行されている既存の EKS クラスターを AWS Outposts に拡張するには、次のように、新しいノードグループnodeGroup.outpostARNが Outposts にノードグループを作成できるように を設定します。

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

metadata:
  name: existing-cluster
  region: us-west-2

nodeGroups:
  # Nodegroup will be created in an AWS region.
  - name: ng

  # Nodegroup will be created on the specified Outpost.
  - name: outpost-ng
    privateNetworking: true
    outpostARN: "arn:aws:outposts:us-west-2:1234:outpost/op-1234"
```

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

この設定では、EKS コントロールプレーンは AWS リージョンで実行され、 outpostARNが設定されたノードグループは指定された Outpost で実行されます。Outposts でノードグループが初めて作

成されるとき、eksctl は指定された Outpost にサブネットを作成して VPC を拡張します。これらのサブネットは、 がoutpostARN設定したノードグループを作成するために使用されます。

既存の VPC を持つお客様は、Outposts でサブネットを作成しnodeGroup.subnets、次のように に渡す必要があります。

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

metadata:
  name: extended-cluster-vpc
  region: us-west-2

vpc:
  id: vpc-1234
    subnets:
      private:
        outpost-subnet-1:
          id: subnet-1234

nodeGroups:
  # Nodegroup will be created in an AWS region.
  - name: ng

  # Nodegroup will be created on the specified Outpost.
  - name: outpost-ng
    privateNetworking: true
    # Subnet IDs for subnets created on Outpost.
    subnets: [subnet-5678]
    outpostARN: "arn:aws:outposts:us-west-2:1234:outpost/op-1234"
```

AWS Outposts でのローカルクラスターの作成

> **Note**
>
> ローカルクラスターは Outpost ラックのみをサポートします。

> **Note**
>
> コントロールプレーンが Outposts にある場合、Amazon Linux 2 のみがノードグループでサポートされています。Outposts のノードグループでは、EBS gp2 ボリュームタイプのみがサポートされています。

eksctl での AWS Outposts のサポートにより、EKS コントロールプレーンやワーカーノードなど、Kubernetes クラスター全体でローカルクラスターを作成し、AWS Outposts でローカルに実行できます。お客様は、AWS Outposts でローカルに実行されている EKS コントロールプレーンとワーカーノードの両方を使用してローカルクラスターを作成するか、Outposts でワーカーノードを作成することで、AWS リージョンで実行されている既存の EKS クラスターを AWS Outposts に拡張できます。

AWS Outposts で EKS コントロールプレーンとノードグループを作成するには、次のように `outpost.controlPlaneOutpostARN`を Outpost ARN に設定します。

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

metadata:
  name: outpost
  region: us-west-2

outpost:
  # Required.
  controlPlaneOutpostARN: "arn:aws:outposts:us-west-2:1234:outpost/op-1234"
  # Optional, defaults to the smallest available instance type on the Outpost.
  controlPlaneInstanceType: m5d.large
```

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

これにより、指定した Outpost に EKS コントロールプレーンとサブネットを作成するように eksctl に指示します。Outposts ラックは単一のアベイラビリティーゾーンに存在するため、eksctl はパブリックサブネットとプライベートサブネットを 1 つだけ作成します。eksctl は、作成された VPC をローカルゲートウェイと関連付けないため、eksctl は API サーバーに接続できず、ノードグループ

を作成できません。したがって、クラスターの作成中に にノードグループClusterConfigが含まれている場合、 コマンドは--without-nodegroup次のように で実行する必要があります。

```
eksctl create cluster -f outpost.yaml --without-nodegroup
```

クラスターの作成後に eksctl が作成した VPC をローカルゲートウェイに関連付けて、API サーバーへの接続を有効にするのはお客様の責任です。このステップの後、 を使用してノードグループを作成できますeksctl create nodegroup。

必要に応じて、 のコントロールプレーンノードoutpost.controlPlaneInstanceTypeまたは のノードグループのインスタンスタイプを指定できますがnodeGroup.instanceType、インスタンスタイプが Outpost に存在する必要があります。存在しない場合、eksctl はエラーを返します。デフォルトでは、eksctl はコントロールプレーンノードとノードグループに対して Outpost で使用可能な最小のインスタンスタイプを選択しようとします。

コントロールプレーンが Outposts にある場合、その Outpost にノードグループが作成されます。必要に応じて、 のノードグループの Outpost ARN を指定できますnodeGroup.outpostARNが、コントロールプレーンの Outpost ARN と一致する必要があります。

```
# outpost-fully-private.yaml
---
apiVersion: eksctl.io/v1alpha5
kind: ClusterConfig

metadata:
  name: outpost-fully-private
  region: us-west-2

privateCluster:
  enabled: true

outpost:
  # Required.
  controlPlaneOutpostARN: "arn:aws:outposts:us-west-2:1234:outpost/op-1234"
  # Optional, defaults to the smallest available instance type on the Outpost.
  controlPlaneInstanceType: m5d.large
```

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

```
metadata:
  name: outpost
  region: us-west-2

outpost:
  # Required.
  controlPlaneOutpostARN: "arn:aws:outposts:us-west-2:1234:outpost/op-1234"
  # Optional, defaults to the smallest available instance type on the Outpost.
  controlPlaneInstanceType: m5d.large

  controlPlanePlacement:
    groupName: placement-group-name
```

既存の VPC

既存の VPC を持つお客様は、次のように でサブネット設定を指定することでvpc.subnets、AWS Outposts でローカルクラスターを作成できます。

```
# outpost-existing-vpc.yaml
---
apiVersion: eksctl.io/v1alpha5
kind: ClusterConfig

metadata:
  name: outpost
  region: us-west-2

vpc:
  id: vpc-1234
  subnets:
    private:
      outpost-subnet-1:
        id: subnet-1234

nodeGroups:
  - name: outpost-ng
    privateNetworking: true

outpost:
    # Required.
    controlPlaneOutpostARN: "arn:aws:outposts:us-west-2:1234:outpost/op-1234"
    # Optional, defaults to the smallest available instance type on the Outpost.
```

```
    controlPlaneInstanceType: m5d.large
```

```
eksctl create cluster -f outpost-existing-vpc.yaml
```

サブネットは、 で指定された Outpost に存在する必要があります。存在しないoutpost.controlPlaneOutpostARN場合、eksctl はエラーを返します。サブネットのローカルゲートウェイにアクセスできる場合、または VPC リソースに接続している場合は、クラスターの作成時にノードグループを指定することもできます。

ローカルクラスターでサポートされていない機能

- アドオン
- サービスアカウントの IAM ロール
- IPv6
- ID プロバイダー
- Fargate
- KMS 暗号化
- ローカルゾーン
- Karpenter
- インスタンスセレクタ
- アベイラビリティーゾーンは、デフォルトで Outpost アベイラビリティーゾーンであるため指定できません。
- vpc.publicAccessCIDRs と vpc.autoAllocateIPv6 はサポートされていません。
- API サーバーへのパブリックエンドポイントアクセスはサポートされていません。ローカルクラスターはプライベートのみのエンドポイントアクセスでのみ作成できるためです。

詳細情報

- AWS Outposts での Amazon EKS
- Amazon EKS on AWS Outposts のローカルクラスター
- ローカルクラスターの作成
- Outpost でのセルフマネージド Amazon Linux ノードの起動

セキュリティ

eksctl には、EKS クラスターのセキュリティを向上させるためのオプションがいくつか用意されています。

withOIDC

を有効にwithOIDCして、Amazon CNI プラグインの IRSA を自動的に作成し、クラスター内のノードに付与されるアクセス許可を制限します。代わりに、必要なアクセス許可を CNI サービスアカウントにのみ付与します。

背景については、この AWS ドキュメントで説明しています。

disablePodIMDS

マネージド型ノードグループとアンマネージド型ノードグループの場合、disablePodIMDSオプションを使用すると、このノードグループで実行されているすべての非ホストネットワーキングポッドが IMDS リクエストを実行できなくなります。

> **Note**
>
> これは と一緒に使用することはできませんwithAddonPolicies。

EKS クラスターの KMS エンベロープ暗号化

> **Note**
>
> 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。詳細については、「EKS ユーザーガイド」の「すべての Kubernetes API データのデフォルトのエンベロープ暗号化」を参照してください。

EKS では、AWS KMS キーを使用して、EKS に保存されている Kubernetes シークレットをエンベロープ暗号化できます。エンベロープ暗号化は、Kubernetes クラスター内に保存されているアプリ

ケーションシークレットまたはユーザーデータに、カスタマーマネージド型の暗号化レイヤーを追加します。

以前は、Amazon EKS はクラスターの作成時にのみ KMS キーを使用したエンベロープ暗号化の有効化をサポートしていました。これで、Amazon EKS クラスターのエンベロープ暗号化をいつでも有効にできます。

EKS 暗号化プロバイダーのサポートを使用したdefense-in-depthの詳細については、AWS コンテナブログの記事を参照してください。

KMS 暗号化を有効にしたクラスターの作成

```
# kms-cluster.yaml
# A cluster with KMS encryption enabled
---
apiVersion: eksctl.io/v1alpha5
kind: ClusterConfig

metadata:
  name: kms-cluster
  region: us-west-2

managedNodeGroups:
- name: ng
# more config

secretsEncryption:
  # KMS key used for envelope encryption of Kubernetes secrets
  keyARN: arn:aws:kms:us-west-2:<account>:key/<key>
```

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

既存のクラスターで KMS 暗号化を有効にする

まだ有効にしていないクラスターで KMS 暗号化を有効にするには、 を実行します。

```
eksctl utils enable-secrets-encryption -f kms-cluster.yaml
```

設定ファイルがない場合:

```
eksctl utils enable-secrets-encryption --cluster=kms-cluster --key-arn=arn:aws:kms:us-
west-2:<account>:key/<key> --region=<region>
```

EKS クラスターで KMS 暗号化を有効にすることに加えて、eksctl は注釈 で更新することで、新しい KMS キーを使用して既存のすべての Kubernetes シークレットを再暗号化しますeksctl.io/kms-encryption-timestamp。この動作は--encrypt-existing-secrets=false、次のように を渡すことで無効にできます。

```
eksctl utils enable-secrets-encryption --cluster=kms-cluster --key-arn=arn:aws:kms:us-
west-2:<account>:key/<key> --encrypt-existing-secrets=false --region=<region>
```

クラスターで既に KMS 暗号化が有効になっている場合、eksctl は既存のすべてのシークレットの再暗号化に進みます。

> **Note**
>
> KMS 暗号化を有効にすると、別の KMS キーを使用するように無効化または更新することはできません。

トラブルシューティング

このトピックでは、Eksctl で一般的なエラーを解決する方法について説明します。

スタック作成の失敗

`--cfn-disable-rollback` フラグを使用して、Cloudformation が失敗したスタックのロールバックを停止し、デバッグを容易にすることができます。

サブネット ID "subnet-11111111" が "subnet-22222222" と同じではありません

次のような VPC のサブネットを指定する設定ファイルを指定します。

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

metadata:
  name: test
  region: us-east-1

vpc:
  subnets:
    public:
      us-east-1a: {id: subnet-11111111}
      us-east-1b: {id: subnet-22222222}
    private:
      us-east-1a: {id: subnet-33333333}
      us-east-1b: {id: subnet-44444444}

nodeGroups: []
```

エラーは、指定されたサブネットが適切なアベイラビリティーゾーンに配置されていない`subnet ID "subnet-11111111" is not the same as "subnet-22222222"`ことを意味します。各アベイラビリティーゾーンの適切なサブネット ID である AWS コンソールで確認します。

この例では、VPC の正しい設定は次のとおりです。

```
vpc:
```

```
  subnets:
    public:
      us-east-1a: {id: subnet-22222222}
      us-east-1b: {id: subnet-11111111}
    private:
      us-east-1a: {id: subnet-33333333}
      us-east-1b: {id: subnet-44444444}
```

削除の問題

削除が機能しない場合、または削除--waitの追加を忘れた場合は、Amazon の他のツールを使用してクラウドフォーメーションスタックを削除する必要がある場合があります。これは、gui または aws cli を使用して実行できます。

kubectl ログと kubectl 実行が認可エラーで失敗する

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、 `kubectl logs`または が次のようなエラーで`kubectl run`失敗します。

```
Error attaching, falling back to logs: unable to upgrade connection: Authorization
 error (user=kube-apiserver-kubelet-client, verb=create, resource=nodes,
 subresource=proxy)
```

```
Error from server (InternalError): Internal error occurred: Authorization error
 (user=kube-apiserver-kubelet-client, verb=get, resource=nodes, subresource=proxy)
```

その後、enableDnsHostnames を設定する必要があります。詳細については、この問題を参照してください。

お知らせ

このトピックでは、新しい Eksctl 機能の過去の発表について説明します。

マネージド型ノードグループのデフォルト

eksctl v0.58.0 以降、ClusterConfigファイルが eksctl create clusterおよび に指定されていない場合、eksctl はデフォルトでマネージド型ノードグループを作成しますeksctl create nodegroup。セルフマネージド型ノードグループを作成するには、 を渡します--managed=false。これにより、Windows ノードグループなどのマネージド型ノードグループでサポートされていない機能が使用されている場合、設定ファイルを使用しないスクリプトが破損する可能性があります。これを修正するには、 を渡すか--managed=false、セルフマネージド型ノードグループを作成する nodeGroupsフィールドを使用してClusterConfigファイルでノードグループ設定を指定します。

カスタム AMIs のノードグループブートストラップオーバーライド

この変更は、間もなく Breaking: overrideBootstrapCommand という問題で発表されました。これで、この PR が渡されました。ブートストラップスクリプトを使用しない、または部分的なブートストラップスクリプトを使用しないカスタム AMIs のサポートから遠ざけることにした理由については、添付の問題をよくお読みください。

引き続きヘルパーを提供します。移行はそれほど難しいことではありません。 はeksctl引き続きスクリプトを提供します。このスクリプトをソースにすると、いくつかの便利な環境プロパティと設定がエクスポートされます。このスクリプトはここにあります。

以下の環境プロパティを自由に使用できます。

```
API_SERVER_URL
B64_CLUSTER_CA
INSTANCE_ID
INSTANCE_LIFECYCLE
CLUSTER_DNS
NODE_TAINTS
MAX_PODS
NODE_LABELS
CLUSTER_NAME
CONTAINER_RUNTIME # default is docker
```

```
KUBELET_EXTRA_ARGS # for details, look at the script
```

が失敗eksctlしないように上書きするときに使用する必要がある最小値は、ラベルです。 はノードにある特定のラベルセットeksctlに依存しているため、それらを見つけることができます。オーバーライドを定義するときは、次の最低限のオーバーライドコマンドを指定してください。

```
    overrideBootstrapCommand: |
      #!/bin/bash

      source /var/lib/cloud/scripts/eksctl/bootstrap.helper.sh

      # Note "--node-labels=${NODE_LABELS}" needs the above helper sourced to work,
 otherwise will have to be defined manually.
      /etc/eks/bootstrap.sh ${CLUSTER_NAME} --container-runtime containerd --kubelet-
extra-args "--node-labels=${NODE_LABELS}"
```

アウトバウンドインターネットアクセスがないノードグループの場合、次のようにブートストラップスクリプト--b64-cluster-caに --apiserver-endpointと を指定する必要があります。

```
    overrideBootstrapCommand: |
      #!/bin/bash

      source /var/lib/cloud/scripts/eksctl/bootstrap.helper.sh

      # Note "--node-labels=${NODE_LABELS}" needs the above helper sourced to work,
 otherwise will have to be defined manually.
      /etc/eks/bootstrap.sh ${CLUSTER_NAME} --container-runtime containerd --kubelet-
extra-args "--node-labels=${NODE_LABELS}" \
        --apiserver-endpoint ${API_SERVER_URL} --b64-cluster-ca ${B64_CLUSTER_CA}
```

「--node-labels」設定に注意してください。これが定義されていない場合、ノードはクラスターに参加eksctlしますが、ノードが になるのを待っている最後のステップで最終的にタイムアウトしますReady。ラベル を持つノードの Kubernetes ルックアップを実行していますalpha.eksctl.io/nodegroup-name=<cluster-name>。これは、アンマネージド型ノードグループにのみ適用されます。マネージド型 では、別のラベルを使用しています。

マネージド型ノードグループに切り替えて、このオーバーヘッドを回避できる場合、その時期が近づいています。すべての上書きがはるかに簡単になります。

翻訳は機械翻訳により提供されています。提供された翻訳内容と英語版の間で齟齬、不一致または矛盾がある場合、英語版が優先します。