Amazon EKS

Guide de l'utilisateur d'Eksctl

Guide de l'utilisateur d'Eksctl: Amazon EKS

Table of Contents

Qu'est-ce que Eksctl ?

eksctl est un utilitaire en ligne de commande qui automatise et simplifie le processus de création, de gestion et d'exploitation des clusters Amazon Elastic Kubernetes Service (Amazon EKS). Écrit en Go, eksctl fournit une syntaxe déclarative via des configurations YAML et des commandes CLI pour gérer les opérations complexes du cluster EKS qui nécessiteraient autrement plusieurs étapes manuelles dans les différents services AWS.

eksctl est particulièrement utile pour les DevOps ingénieurs, les équipes de plateforme et les administrateurs Kubernetes qui doivent déployer et gérer de manière cohérente des clusters EKS à grande échelle. Il est particulièrement utile pour les entreprises qui passent de Kubernetes autogéré à EKS, ou pour celles qui mettent en œuvre des pratiques d'infrastructure en tant que code (IaC), car il peut être intégré aux pipelines et aux flux de travail d'automatisation existants CI/CD . L'outil élimine de nombreuses interactions complexes entre les services AWS nécessaires à la configuration du cluster EKS, telles que la configuration VPC, la création de rôles IAM et la gestion des groupes de sécurité.

Les principales fonctionnalités d'eksctl incluent la possibilité de créer des clusters EKS entièrement fonctionnels à l'aide d'une seule commande, la prise en charge de configurations réseau personnalisées, la gestion automatisée des groupes de nœuds et l'intégration des GitOps flux de travail. L'outil gère les mises à niveau des clusters, adapte les groupes de nœuds et gère la gestion des modules complémentaires selon une approche déclarative. eksctl fournit également des fonctionnalités avancées telles que la configuration du profil Fargate, la personnalisation des groupes de nœuds gérés et l'intégration d'instances ponctuelles, tout en maintenant la compatibilité avec les autres outils et services AWS grâce à l'intégration native du SDK AWS.

Caractéristiques

Les fonctionnalités actuellement mises en œuvre sont les suivantes :

- Créer, obtenir, répertorier et supprimer des clusters
- Création, vidange et suppression de groupes de nœuds
- Redimensionner un groupe de nœuds
- Mettre à jour un cluster
- Utiliser personnalisé AMIs
- Configuration de la mise en réseau VPC

- Configuration de l'accès aux points de terminaison de l'API
- Support pour les groupes de nœuds GPU
- Instances ponctuelles et instances mixtes
- Gestion IAM et politiques complémentaires
- Répertorier les piles Cloudformation du cluster
- Installer coredns
- Écrire un fichier kubeconfig pour un cluster

FAQ Eksctl

Général

Puis-je utiliser **`eksctl`** pour gérer des clusters qui n'ont pas été créés par **`eksctl`** ?

Oui ! À partir de la version, `0.40.0` vous `eksctl` pouvez exécuter sur n'importe quel cluster, qu'il ait été créé par `eksctl` ou non. Pour de plus amples informations, veuillez consulter the section called “Clusters non créés par eksctl”.

Groupes de nœuds

Comment puis-je modifier le type d'instance de mon groupe de nœuds ?

Du point de vue de`eksctl`, les groupes de nœuds sont immuables. Cela signifie qu'une fois créé, la seule chose que `eksctl` vous pouvez faire est de redimensionner le groupe de nœuds vers le haut ou vers le bas.

Pour modifier le type d'instance, créez un nouveau groupe de nœuds avec le type d'instance souhaité, puis videz-le afin que les charges de travail soient déplacées vers le nouveau. Une fois cette étape terminée, vous pouvez supprimer l'ancien groupe de nœuds.

Comment puis-je voir les données utilisateur générées pour un groupe de nœuds ?

Vous aurez d'abord besoin du nom de la pile Cloudformation qui gère le groupe de nœuds :

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

Vous verrez un nom similaire à`eksctl-CLUSTER_NAME-nodegroup-NODEGROUP_NAME`.

Vous pouvez exécuter ce qui suit pour obtenir les données utilisateur. Notez la dernière ligne qui décode à partir de base64 et décompresse les données compressées.

```
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

Comment configurer l'entrée avec **`eksctl`** ?

Nous vous recommandons d'utiliser le contrôleur AWS Load Balancer. La documentation sur le déploiement du contrôleur sur votre cluster, ainsi que sur la migration depuis l'ancien contrôleur d'entrée ALB, est disponible ici.

Pour le Nginx Ingress Controller, la configuration serait la même que celle des autres clusters Kubernetes.

Kubectl

J'utilise un proxy HTTPS et la validation du certificat de cluster échoue. Comment puis-je utiliser le système CAs ?

Définissez la variable `KUBECONFIG_USE_SYSTEM_CA` d'environnement pour `kubeconfig` respecter les autorités de certification du système.

Run à sec

La fonction d'exécution à sec vous permet d'inspecter et de modifier les instances correspondant au sélecteur d'instance avant de procéder à la création d'un groupe de nœuds.

Lorsqu'il `eksctl create cluster` est appelé avec les options et options du sélecteur d'instance`--dry-run`, eksctl produit un ClusterConfig fichier contenant un groupe de nœuds

représentant les options de la CLI et les types d'instances définis sur les instances correspondant aux critères de ressources du sélecteur d'instance.

```
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
```

Le résultat généré ClusterConfig peut ensuite être transmis à `eksctl create cluster` :

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

Lorsqu'un ClusterConfig fichier est transmis`--dry-run`, eksctl produit un ClusterConfig fichier contenant les valeurs définies dans le fichier.

Options uniques dans eksctl

Certaines options ponctuelles ne peuvent pas être représentées dans le `ClusterConfig` fichier, `--install-vpc-controllers` par exemple.

On s'attend à ce que :

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

suivi par :

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

serait équivalent à exécuter la première commande sans`--dry-run`.

eksctl interdit donc de transmettre des options qui ne peuvent pas être représentées dans le fichier de configuration lorsqu'il est `--dry-run` transmis.

> **Important**
>
> Si vous devez transmettre un profil AWS, définissez la variable d'`AWS_PROFILE`environnement au lieu de transmettre l'option `--profile` CLI.

didacticiel

Cette rubrique explique comment installer et configurer eksctl, puis comment l'utiliser pour créer un cluster Amazon EKS.

Étape 1 : installer eksctl

Procédez comme suit pour télécharger et installer la dernière version d'eksctl sur votre appareil Linux ou macOS :

Pour installer eksctl avec Homebrew

1. (Prérequis) Installez Homebrew.
2. Ajoutez le robinet AWS :

```
brew tap aws/tap
```

3. Installer eksctl

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

Avant d'utiliser eksctl, effectuez les étapes de configuration suivantes :

1. Installation des prérequis :
   - Installez la version 2.x ou ultérieure de l'interface de ligne de commande AWS.
   - Installez kubectl en utilisant Homebrew :

```
brew install kubernetes-cli
```

2. Configurez les informations d'identification AWS dans votre environnement :

```
aws configure
```

3. Vérifiez la configuration de la CLI AWS :

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

Étape 2 : Création du fichier de configuration du cluster

Créez un fichier de configuration de cluster en procédant comme suit :

1. Créez un nouveau fichier nommé `cluster.yaml` :

```
touch cluster.yaml
```

2. Ajoutez la configuration de base de cluster suivante :

```
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. Personnalisez la configuration :
   - Mettez à jour le `region` pour qu'il corresponde à la région AWS souhaitée.
   - Modifiez le en `instanceType` fonction de vos exigences en matière de charge de travail.
   - Ajustez le `desiredCapacityminSize`, et `maxSize` selon vos besoins.
4. Validez le fichier de configuration :

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

Étape 3 : créer un cluster

Pour créer votre cluster EKS, procédez comme suit :

1. Créez le cluster à l'aide du fichier de configuration :

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

2. Attendez la création du cluster (cela prend généralement 15 à 20 minutes).
3. Vérifiez la création du cluster :

```
eksctl get cluster
```

4. Configurez kubectl pour utiliser votre nouveau cluster :

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

5. Vérifiez la connectivité du cluster :

```
kubectl get nodes
```

Votre cluster est maintenant prêt à être utilisé.

Facultatif : Supprimer le cluster

N'oubliez pas de supprimer le cluster lorsque vous avez terminé pour éviter des frais inutiles :

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

> **Note**
>
> La création de clusters peut entraîner des frais AWS. Assurez-vous de consulter les tarifs d'Amazon EKS avant de créer un cluster.

Étapes suivantes

- Configurer Kubectl pour se connecter au cluster
- Déployer un exemple d'application

Options d'installation pour Eksctl

`eksctl`est disponible à l'installation à partir des versions officielles, comme décrit ci-dessous. Nous vous recommandons d'installer uniquement `eksctl` à partir des GitHub versions officielles. Vous pouvez choisir d'utiliser un programme d'installation tiers, mais sachez qu'AWS ne gère ni ne prend en charge ces méthodes d'installation. Utilisez-les à votre propre discrétion.

Prérequis

Vous devez avoir configuré les informations d'identification de l'API AWS. Ce qui fonctionne pour l'AWS CLI ou pour tout autre outil (kops, Terraform, etc.) devrait suffire. Vous pouvez utiliser des variables de `~/.aws/credentials`fichier ou d'environnement. Pour plus d'informations, consultez le document de référence de la CLI AWS.

Vous aurez également besoin de la commande AWS IAM Authenticator for Kubernetes `aws-iam-authenticator` (disponible dans la version 1.16.156 `aws eks get-token` ou ultérieure de l'AWS CLI) dans votre. `PATH`

Le compte IAM utilisé pour la création du cluster EKS doit avoir ces niveaux d'accès minimaux.

Service AWS	Niveau d'accès
CloudFormation	Accès complet
EC2	Complet : Balisage limité : liste, lecture, écriture
EC2 Auto Scaling	Limité : liste, écriture
EKS	Accès complet
IAM	Limité : gestion des listes, de la lecture, de l'écriture et des autorisations
Systems Manager	Limité : List (Liste), Read (Lire)

Pour Unix

Pour télécharger la dernière version, exécutez :

```
# 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
```

Pour Windows

Téléchargement direct (dernière version) :

- AMD64/x86_64
- ARMv6
- ARMv7
- ARM64

Assurez-vous de décompresser l'archive dans un dossier de la PATH variable.

Vérifiez éventuellement le checksum :

1. Téléchargez le fichier checksum : le plus récent
2. Utilisez l'invite de commande pour comparer manuellement `CertUtil` le résultat au fichier de somme de contrôle téléchargé.

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

3. Utilisation PowerShell pour automatiser la vérification en utilisant l'`-eq`opérateur pour obtenir un `False` résultat `True` OR :

   ```
   # 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]
```

À l'aide de Git Bash :

```
# 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
```

L'`eksctl`exécutable est placé dans`$HOME/bin`, c'est-à-dire dans `$PATH` Git Bash.

Homebrew

Vous pouvez utiliser Homebrew pour installer des logiciels sur macOS et Linux.

AWS gère un robinet Homebrew incluant eksctl.

Pour plus d'informations sur le Homebrew tap, consultez le projet sur Github et la formule Homebrew pour eksctl.

Pour installer eksctl avec Homebrew

1. (Prérequis) Installer Homebrew
2. Ajoutez le robinet AWS

```
brew tap aws/tap
```

3. Installer eksctl

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

Docker

Pour chaque version et chaque RC, une image de conteneur est envoyée vers le référentiel `public.ecr.aws/eksctl/eksctl` ECR. En savoir plus sur l'utilisation sur ECR Public Gallery - eksctl. Par exemple,

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

Achèvement de la coque

Bash

Pour activer la complétion par bash, exécutez ce qui suit, ou mettez-le dans `~/.bashrc` ou `~/.profile` :

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

Zsh

Pour terminer zsh, veuillez exécuter :

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

et insérez ce qui suit dans `~/.zshrc` :

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

Notez que si vous n'utilisez pas une distribution, oh-my-zsh vous devrez peut-être d'abord activer l'autocomplétion (et la mettre `~/.zshrc` pour la rendre persistante) :

```
autoload -U compinit
compinit
```

Poisson

Les commandes ci-dessous peuvent être utilisées pour la complétion automatique du fish :

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

Powershell

La commande ci-dessous peut être consultée pour la configurer. Notez que le chemin peut être différent en fonction des paramètres de votre système.

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

Mises à jour

> **Important**
>
> Si vous installez eksctl en le téléchargeant directement (sans utiliser de gestionnaire de paquets), vous devez le mettre à jour manuellement.

Clusters

Ce chapitre traite de la création et de la configuration de clusters EKS à l'aide d'eksctl. Il inclut également des modules complémentaires et le mode automatique EKS.

Rubriques :

- the section called “Entrées EKS Access”
  - Simplifiez la gestion RBAC de Kubernetes en remplaçant ConfigMap aws-auth par des entrées d'accès EKS
  - Migrer les mappages d'identité IAM existants depuis ConfigMap aws-auth vers les entrées d'accès
  - Configurer les modes d'authentification du cluster et contrôler les autorisations d'administrateur du créateur du cluster
- the section called “Mises à jour complémentaires par défaut”
  - Protégez les clusters en mettant à jour les modules complémentaires EKS par défaut sur les anciens clusters
- the section called “Modules complémentaires”
  - Automatisez les tâches de routine pour l'installation, la mise à jour et la suppression des modules complémentaires.
  - Les modules complémentaires Amazon EKS incluent les modules complémentaires AWS, les modules complémentaires communautaires open source et les modules complémentaires du marché.
- the section called “Mode automatique EKS”
  - Réduisez les frais opérationnels en laissant AWS gérer votre infrastructure EKS
  - Configurer des pools de nœuds personnalisés au lieu des pools à usage général et des pools de systèmes par défaut
  - Convertissez les clusters EKS existants pour utiliser le mode automatique
- the section called “CloudWatch journalisation”
  - Résolvez les problèmes de cluster en activant les journaux pour des composants spécifiques du plan de contrôle EKS
  - Configuration des périodes de conservation des journaux pour les journaux du cluster EKS

- Modifier les paramètres de journalisation du cluster existants à l'aide des commandes eksctl

- the section called “Améliorations de clusters”
  - Préservez la sécurité et la stabilité en mettant à niveau en toute sécurité les versions du plan de contrôle EKS
  - Déployez les mises à niveau entre les groupes de nœuds en remplaçant les anciens groupes par de nouveaux
  - Mettre à jour les extensions de cluster par défaut
- the section called “Création et gestion de clusters”
  - Commencez rapidement avec des clusters EKS de base à l'aide de groupes de nœuds gérés par défaut
  - Créez des clusters personnalisés à l'aide de fichiers de configuration avec des configurations spécifiques
  - Déployez des clusters dans des VPCs environnements existants avec un réseau privé et des politiques IAM personnalisées
- the section called “Configurer kubelet”
  - Empêchez la privation de ressources des nœuds en configurant les réservations de kubelet et de daemon système
  - Personnalisez les seuils d'expulsion en fonction de la disponibilité de la mémoire et du système de fichiers
  - Activer ou désactiver des portes de fonctionnalités Kubelet spécifiques dans les groupes de nœuds
- the section called “Connecteur EKS”
  - Centralisez la gestion des déploiements hybrides de Kubernetes via la console EKS
  - Configuration des rôles et autorisations IAM pour l'accès au cluster externe
  - Supprimer les clusters externes et nettoyer les ressources AWS associées
- the section called “Cluster entièrement privé EKS”
  - Répondez aux exigences de sécurité avec des clusters EKS entièrement privés sans accès Internet sortant
  - Configuration de l'accès privé aux services AWS via des points de terminaison VPC
  - Création et gestion de groupes de nœuds privés avec des paramètres réseau explicites
- the section called “Support pour charpentiers”
  - Automatisez le provisionnement des nœuds

- Créez des configurations personnalisées pour les approvisionneurs Karpenter
- Configurer Karpenter avec la gestion des interruptions ponctuelles

- the section called “Amazon EMR”
  - Création d'un mappage d'identité IAM entre EMR et le cluster EKS
- the section called “Support EKS Fargate”
  - Définissez des profils Fargate personnalisés pour la planification des pods
  - Gérez les profils Fargate via la création et les mises à jour de configuration
- the section called “Clusters non créés par eksctl”
  - Standardisez la gestion des clusters créés en dehors d'eksctl
  - Utiliser les commandes eksctl sur des clusters non eksctl existants
- the section called “Activer le changement de zone”
  - Améliorez la disponibilité des applications en activant les fonctionnalités de basculement rapide des zones
  - Configurer le changement de zone sur les nouveaux déploiements de clusters EKS
  - Activez les fonctionnalités de changement de zone sur les clusters EKS existants

Création et gestion de clusters

Cette rubrique explique comment créer et supprimer des clusters EKS à l'aide d'Eksctl. Vous pouvez créer des clusters à l'aide d'une commande CLI ou en créant un fichier YAML de configuration de cluster.

Création d'un cluster simple

Créez un cluster simple à l'aide de la commande suivante :

```
eksctl create cluster
```

Cela créera un cluster EKS dans votre région par défaut (telle que spécifiée par la configuration de votre CLI AWS) avec un groupe de nœuds géré contenant deux nœuds m5.large.

eksctl crée désormais un groupe de nœuds géré par défaut lorsqu'aucun fichier de configuration n'est utilisé. Pour créer un groupe de nœuds autogéré, passez `--managed=false` à ou. `eksctl create cluster` `eksctl create nodegroup`

Considérations

- Lorsque vous créez des clusters dans`us-east-1`, vous pouvez rencontrer un`UnsupportedAavailabilityZoneException`. Si cela se produit, copiez les zones suggérées et passez le `--zones` drapeau, par exemple :`eksctl create cluster --region=us-east-1 --zones=us-east-1a,us-east-1b,us-east-1d`. Ce problème peut se produire dans d'autres régions, mais il est moins courant. Dans la plupart des cas, vous n'aurez pas besoin d'utiliser le `--zone` drapeau.

Créer un cluster à l'aide du fichier de configuration

Vous pouvez créer un cluster à l'aide d'un fichier de configuration au lieu d'indicateurs.

Tout d'abord, créez `cluster.yaml` le fichier :

```
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
```

Ensuite, exécutez cette commande :

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

Cela créera un cluster tel que décrit.

Si vous deviez utiliser un VPC existant, vous pouvez utiliser un fichier de configuration comme celui-ci :

```
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
```

Le nom du cluster ou du groupe de nœuds ne doit contenir que des caractères alphanumériques (en distinguant majuscules et minuscules) et des tirets. Il doit commencer par un caractère alphabétique et ne doit pas dépasser 128 caractères, sinon vous recevrez une erreur de validation. Pour plus d'informations, consultez la section Créer une pile depuis la CloudFormation console dans le guide de l'utilisateur d'AWS CLoud Formation.

Mettre à jour kubeconfig pour le nouveau cluster

Une fois le cluster créé, la configuration kubernetes appropriée sera ajoutée à votre fichier kubeconfig. Il s'agit du fichier que vous avez configuré dans la variable d'environnement

`KUBECONFIG` ou `~/.kube/config` par défaut. Le chemin d'accès au fichier kubeconfig peut être remplacé à l'aide de l'indicateur. `--kubeconfig`

Autres indicateurs qui peuvent modifier la façon dont le fichier kubeconfig est écrit :

pour restricteur	type	use	valeur par défaut
--kubeconfig	chaîne	chemin pour écrire kubeconfig (incompatible avec --auto-kubeconfig)	$KUBECONFIG ou ~/.kube/config
--set-kubeconfig-context	bool	si vrai, alors le contexte actuel sera défini dans kubeconfig ; si un contexte est déjà défini, il sera remplacé	true
--auto-kubeconfig	bool	enregistrer le fichier kubeconfig par nom de cluster	true
--write-kubeconfig	bool	activer/désactiver l'écriture de kubeconfig	true

supprimer un cluster

Pour supprimer ce cluster, exécutez :

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

> **Warning**
>
> Utilisez l'`--wait`indicateur lors des opérations de suppression pour vous assurer que les erreurs de suppression sont correctement signalées.

Sans cet `--wait` indicateur, eksctl n'effectuera qu'une opération de suppression sur la CloudFormation pile du cluster et n'attendra pas sa suppression. Dans certains cas, les ressources AWS utilisant le cluster ou son VPC peuvent entraîner l'échec de la suppression du cluster. Si votre suppression échoue ou si vous oubliez l'indicateur d'attente, vous devrez peut-être accéder à l' CloudFormation interface graphique et supprimer les piles eks qui s'y trouvent.

> **Warning**
>
> Les politiques PDB peuvent bloquer la suppression des nœuds lors de la suppression du cluster.

Lors de la suppression d'un cluster contenant des groupes de nœuds, les politiques Pod Disruption Budget (PDB) peuvent empêcher la suppression réussie des nœuds. Par exemple, les clusters `aws-ebs-csi-driver` installés comportent généralement deux pods dotés d'une politique PDB autorisant l'indisponibilité d'un seul pod à la fois, rendant l'autre pod inévitable lors de la suppression. Pour supprimer correctement le cluster dans ces scénarios, utilisez l'`disable-nodegroup-eviction`indicateur pour contourner les vérifications de politique PDB :

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

Consultez le `examples/`répertoire dans le GitHub dépôt eksctl pour d'autres exemples de fichiers de configuration.

Run à sec

La fonctionnalité d'exécution à sec permet de générer un ClusterConfig fichier qui ignore la création de clusters et génère un ClusterConfig fichier qui représente les options de la CLI fournies et contient les valeurs par défaut définies par eksctl.

Vous trouverez plus d'informations sur la page Dry Run.

Mode automatique EKS

eksctl prend en charge le mode automatique d'EKS, une fonctionnalité qui étend la gestion des clusters Kubernetes par AWS au-delà du cluster lui-même, afin de permettre à AWS de configurer et de gérer également l'infrastructure qui permet le bon fonctionnement de vos charges de travail. Cela vous permet de déléguer les décisions clés en matière d'infrastructure et de tirer parti de

l'expertise d'AWS pour les day-to-day opérations. L'infrastructure de clusters gérée par AWS inclut de nombreuses fonctionnalités Kubernetes en tant que composants principaux, par opposition aux modules complémentaires, tels que le dimensionnement automatique du calcul, la mise en réseau des pods et des services, l'équilibrage de charge des applications, le DNS du cluster, le stockage par blocs et le support du GPU.

Création d'un cluster EKS avec le mode automatique activé

`eksctl`a ajouté un nouveau `autoModeConfig` champ pour activer et configurer le mode automatique. La forme du `autoModeConfig` champ est

```
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
```

Si `autoModeConfig.enabled` c'est vrai, eksctl crée un cluster EKS en passant`computeConfig.enabled: true`, et `storageConfig.blockStorage.enabled: true` à l'API EKS`kubernetesNetworkConfig.elasticLoadBalancing.enabled: true`, ce qui permet de gérer les composants du plan de données tels que le calcul, le stockage et le réseau.

Pour créer un cluster EKS avec le mode automatique activé, définissez`autoModeConfig.enabled: true`, comme dans

```
# 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 crée un rôle de nœud à utiliser pour les nœuds lancés par le mode automatique. eksctl crée également les pools de nœuds et. `general-purpose` `system` Pour désactiver la création des pools de nœuds par défaut, par exemple pour configurer vos propres pools de nœuds qui utilisent un ensemble de sous-réseaux différent, définissez`nodePools: []`, comme dans

```
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.
```

Mise à jour d'un cluster EKS pour utiliser le mode automatique

Pour mettre à jour un cluster EKS existant afin d'utiliser le mode automatique, exécutez

```
# 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**
>
> Si le cluster a été créé par eksctl et qu'il utilise des sous-réseaux publics comme sous-réseaux de cluster, le mode automatique lancera les nœuds dans les sous-réseaux publics. Pour utiliser des sous-réseaux privés pour les nœuds de travail lancés par le mode automatique, mettez à jour le cluster pour qu'il utilise des sous-réseaux privés.

Désactivation du mode automatique

Pour désactiver le mode automatique, configurez `autoModeConfig.enabled: false` et exécutez

```
# 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
```

Plus d'informations

- Mode automatique EKS

Entrées EKS Access

Vous pouvez utiliser eksctl pour gérer les entrées d'accès EKS. Utilisez les entrées d'accès pour accorder des autorisations Kubernetes à AWS IAM Identities. Par exemple, vous pouvez accorder à un rôle de développeur l'autorisation de lire les ressources Kubernetes dans un cluster.

Cette rubrique explique comment utiliser eksctl pour gérer les entrées d'accès. Pour des informations générales sur les entrées d'accès, voir Accorder aux utilisateurs IAM l'accès à Kubernetes avec des entrées d'accès EKS.

Vous pouvez joindre des politiques d'accès Kubernetes définies par AWS ou associer une identité IAM à un groupe Kubernetes.

Pour plus d'informations sur les politiques prédéfinies disponibles, voir Associer des politiques d'accès aux entrées d'accès.

Si vous devez définir les politiques Kubernetes du client, associez l'identité IAM à un groupe Kubernetes et accordez des autorisations à ce groupe.

Mode d'authentification du cluster

Vous ne pouvez utiliser les entrées d'accès que si le mode d'authentification du cluster le permet.

Pour plus d'informations, voir Définir le mode d'authentification du cluster

Définir le mode d'authentification avec un fichier YAML

`eksctl`a ajouté un nouveau `accessConfig.authenticationMode` champ sous ClusterConfig, qui peut être défini sur l'une des trois valeurs suivantes :

- `CONFIG_MAP`- par défaut dans l'API EKS - seul `aws-auth` ConfigMap sera utilisé
- `API`- seule l'API d'entrées d'accès sera utilisée
- `API_AND_CONFIG_MAP`- par défaut dans `eksctl` - les deux `aws-auth` ConfigMap et l'API d'entrées d'accès peuvent être utilisées

Définissez le mode d'authentification dans ClusterConfig YAML :

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

Mettre à jour le mode d'authentification à l'aide d'une commande

Si vous souhaitez utiliser des entrées d'accès sur un cluster déjà existant, non créé par eksctl, où l'`CONFIG_MAP`option est utilisée, l'utilisateur devra d'abord définir sur. `authenticationMode` `API_AND_CONFIG_MAP` Pour cela, `eksctl` a introduit une nouvelle commande pour mettre à jour le mode d'authentification du cluster, qui fonctionne à la fois avec les indicateurs CLI, par exemple

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

Accédez aux ressources d'entrée

Les entrées d'accès ont un type, tel que `STANDARD` ou`EC2_LINUX`. Le type dépend de la manière dont vous utilisez l'entrée d'accès.

- Le `standard` type est destiné à accorder des autorisations Kubernetes aux utilisateurs IAM et aux rôles IAM.

- Par exemple, vous pouvez consulter les ressources Kubernetes dans la console AWS en attachant une politique d'accès au rôle ou à l'utilisateur que vous utilisez pour accéder à la console.
- Les `EC2_WINDOWS` types `EC2_LINUX` et permettent d'accorder des autorisations Kubernetes aux instances EC2. Les instances utilisent ces autorisations pour rejoindre le cluster.

Pour plus d'informations sur les types d'entrées d'accès, voir Création d'entrées d'accès

IAM entités

Vous pouvez utiliser des entrées d'accès pour accorder des autorisations Kubernetes aux identités IAM telles que les utilisateurs IAM et les rôles IAM.

Utilisez le `accessConfig.accessEntries` champ pour associer l'ARN d'une ressource IAM à une API EKS Access Entries. Par exemple :

```
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
```

En plus d'associer des politiques EKS, on peut également spécifier les groupes Kubernetes auxquels appartient une entité IAM, octroyant ainsi des autorisations via RBAC.

Groupes de nœuds gérés et Fargate

L'intégration avec les entrées d'accès à ces ressources sera réalisée en coulisse grâce à l'API EKS. Les groupes de nœuds gérés et les pods Fargate nouvellement créés créeront des entrées d'accès à l'API, plutôt que d'utiliser des ressources RBAC préchargées. Les groupes de nœuds et les pods Fargate existants ne seront pas modifiés et continueront de s'appuyer sur les entrées de la carte de configuration aws-auth.

Groupes de nœuds autogérés

Chaque entrée d'accès possède un type. Pour autoriser les groupes de nœuds autogérés, `eksctl` vous créerez une entrée d'accès unique pour chaque groupe de nœuds, l'ARN principal étant défini sur l'ARN du rôle de nœud et le type défini sur AmiFamily `EC2_LINUX` ou `EC2_WINDOWS` en fonction de celui-ci.

Lorsque vous créez vos propres entrées d'accès, vous pouvez également spécifier `EC2_LINUX` (pour un rôle IAM utilisé avec des nœuds autogérés Linux ou Bottlerocket), `EC2_WINDOWS` (pour un rôle IAM utilisé avec des nœuds autogérés Windows), `FARGATE_LINUX` (pour un rôle IAM utilisé avec AWS Fargate (Fargate)) ou en tant que type. `STANDARD` Si vous ne spécifiez aucun type, le type par défaut est défini sur`STANDARD`.

> **Note**
>
> Lors de la suppression d'un groupe de nœuds créé avec un groupe préexistant`instanceRoleARN`, il est de la responsabilité de l'utilisateur de supprimer l'entrée d'accès correspondante lorsqu'aucun autre groupe de nœuds n'y est associé. Cela est dû au fait que eksctl ne cherche pas à savoir si une entrée d'accès est toujours utilisée par des groupes de nœuds autogérés qui n'ont pas été créés par eksctl, car il s'agit d'un processus compliqué.

Créer une entrée d'accès

Cela peut être fait de deux manières différentes, soit lors de la création du cluster, soit en spécifiant les entrées d'accès souhaitées dans le cadre du fichier de configuration, soit en exécutant :

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

OU après la création du cluster, en exécutant :

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

Pour un exemple de fichier de configuration permettant de créer des entrées d'accès, consultez 40-access-entries.yaml dans le dépôt eksctl. GitHub

Obtenir l'entrée d'accès

L'utilisateur peut récupérer toutes les entrées d'accès associées à un certain cluster en exécutant l'une des opérations suivantes :

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

OU

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

Sinon, pour récupérer uniquement l'entrée d'accès correspondant à une certaine entité IAM, il faut utiliser le `--principal-arn` drapeau. Par exemple

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

Supprimer l'entrée d'accès

Pour supprimer une seule entrée d'accès à la fois, utilisez :

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

Pour supprimer plusieurs entrées d'accès, utilisez le `--config-file` drapeau et spécifiez toutes les entrées `principalARN's` correspondantes, sous le `accessEntry` champ de niveau supérieur, par ex.

```
...
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
```

Migrer depuis aws-auth ConfigMap

L'utilisateur peut migrer ses identités IAM existantes de `aws-auth` configmap vers les entrées d'accès en exécutant ce qui suit :

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

Lorsque l'`--target-authentication-mode`indicateur est défini sur`API`, le mode d'authentification passe en `API` `API` mode (ignoré s'il est déjà activé), les mappages d'identité IAM sont migrés vers les entrées d'accès et le `aws-auth` fichier configmap est supprimé du cluster.

Lorsque l'`--target-authentication-mode`indicateur est défini sur`API_AND_CONFIG_MAP`, le mode d'authentification passe en `API_AND_CONFIG_MAP` `API_AND_CONFIG_MAP` mode (ignoré s'il est déjà activé), les mappages d'identité IAM sont migrés vers les entrées d'accès, mais `aws-auth` le configmap est conservé.

> **Note**
>
> Lorsque l'`--target-authentication-mode`indicateur est défini sur`API`, cette commande ne met pas à jour le mode d'authentification `API` en mode si `aws-auth` configmap possède l'une des contraintes ci-dessous.

- Il existe un mappage d'identité au niveau du compte.

- Un ou plusieurs Roles/Users sont mappés vers le ou les groupes Kubernetes commençant par un préfixe `system:` (sauf pour les groupes spécifiques à EKS, c'est-à-dire`system:masters`, etc.). `system:bootstrappers` `system:nodes`
- Un ou plusieurs mappages d'identité IAM concernent un [rôle lié au service] (lien : IAM/latest/ UserGuide/using - service-linked-roles .html).

Désactiver les autorisations d'administrateur du créateur du cluster

`eksctl`a ajouté un nouveau champ `accessConfig.bootstrapClusterCreatorAdminPermissions: boolean` qui, lorsqu'il est défini sur false, désactive l'octroi d'autorisations d'administrateur de cluster à l'identité IAM créant le cluster. par exemple

ajoutez l'option au fichier de configuration :

```
accessConfig:
  bootstrapClusterCreatorAdminPermissions: false
```

et lancez :

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

Clusters non créés par eksctl

Vous pouvez exécuter `eksctl` des commandes sur des clusters qui n'ont pas été créés par`eksctl`.

> **Note**
>
> Eksctl ne peut prendre en charge que des clusters non propriétaires dont les noms sont compatibles avec AWS. CloudFormation Tout nom de cluster qui ne correspond pas à cette valeur échouera au contrôle de validation de CloudFormation l'API.

Commandes prises en charge

Les commandes suivantes peuvent être utilisées contre des clusters créés par tout autre moyen que`eksctl`. Les commandes, les drapeaux et les options du fichier de configuration peuvent être utilisés exactement de la même manière.

Si certaines fonctionnalités nous ont échappé, merci de nous le faire savoir.

- ✓ Créez :
  - ✓ `eksctl create nodegroup` (voir note ci-dessous)
  - ✓ `eksctl create fargateprofile`
  - ✓ `eksctl create iamserviceaccount`
  - ✓ `eksctl create iamidentitymapping`
- ✓ Obtenez :
  - ✓ `eksctl get clusters/cluster`
  - ✓ `eksctl get fargateprofile`
  - ✓ `eksctl get nodegroup`
  - ✓ `eksctl get labels`
- ✓ Supprimer :
  - ✓ `eksctl delete cluster`
  - ✓ `eksctl delete nodegroup`
  - ✓ `eksctl delete fargateprofile`
  - ✓ `eksctl delete iamserviceaccount`
  - ✓ `eksctl delete iamidentitymapping`
- ✓ Mise à niveau :
  - ✓ `eksctl upgrade cluster`
  - ✓ `eksctl upgrade nodegroup`
- ✓ Débranchement/désactivation :
  - ✓ `eksctl set labels`
  - ✓ `eksctl unset labels`
- ✓ Échelle :
  - ✓ `eksctl scale nodegroup`
- ✓ Égouttage :
  - ✓ `eksctl drain nodegroup`
- ✓ Activer :
  - ✓ `eksctl enable profile`
  - ✓ `eksctl enable repo`

- ✓ Utilitaires :
  - ✓ `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`

Création de groupes de nœuds

`eksctl create nodegroup`est la seule commande qui nécessite une saisie spécifique de la part de l'utilisateur.

Étant donné que les utilisateurs peuvent créer leurs clusters avec la configuration réseau de leur choix, ils n'`eksctl`essaieront pas pour le moment de récupérer ou de deviner ces valeurs. Cela pourrait changer à l'avenir à mesure que nous en apprendrons davantage sur la façon dont les utilisateurs utilisent cette commande sur des clusters non créés par eksctl.

Cela signifie que pour créer des groupes de nœuds ou des groupes de nœuds gérés sur un cluster qui n'a pas été créé par`eksctl`, un fichier de configuration contenant les détails du VPC doit être fourni. Au minimum :

```
---
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"

...
```

Pour plus d'informations sur les options de configuration VPC, consultez la section Mise en réseau.

Enregistrement de clusters non EKS avec le connecteur EKS

Vous pouvez utiliser le connecteur EKS pour afficher les clusters en dehors d'AWS dans la console EKS. Ce processus nécessite l'enregistrement du cluster auprès d'EKS et l'exécution de l'agent EKS Connector sur le cluster Kubernetes externe.

`eksctl`simplifie l'enregistrement des clusters non EKS en créant les ressources AWS requises et en générant des manifestes Kubernetes que le connecteur EKS doit appliquer au cluster externe.

Enregistrer un cluster

Pour enregistrer ou connecter un cluster Kubernetes autre qu'EKS, exécutez

```
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
```

Cette commande enregistre le cluster et écrit trois fichiers contenant les manifestes Kubernetes pour EKS Connector qui doivent être appliqués au cluster externe avant l'expiration de l'enregistrement.

> **Note**
>
> `eks-connector-clusterrole.yaml`get et `eks-connector-console-dashboard-full-access-clusterrole.yaml` accordez `list` des autorisations pour les ressources Kubernetes dans tous les espaces de noms à l'identité IAM appelante et doivent être modifiées en conséquence si nécessaire avant de les appliquer au cluster. Pour configurer un accès plus restreint, consultez la section Accorder l'accès à un utilisateur pour qu'il puisse consulter un cluster.

Pour fournir un rôle IAM existant à utiliser pour EKS Connector, transmettez-le `--role-arn` comme suit :

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

Si le cluster existe déjà, eksctl renverra une erreur.

Désenregistrer un cluster

Pour désenregistrer ou déconnecter un cluster enregistré, exécutez

```
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
```

Cette commande annulera l'enregistrement du cluster externe et supprimera les ressources AWS associées, mais vous devez supprimer les ressources Kubernetes du connecteur EKS du cluster.

Plus d'informations

- Connecteur EKS

Personnalisation de la configuration de Kubelet

Les ressources du système peuvent être réservées via la configuration du kubelet. Ceci est recommandé, car en cas de manque de ressources, le kubelet risque de ne pas être en mesure d'expulser les pods et éventuellement de transformer le nœud en question. `NotReady` Pour ce faire, les fichiers de configuration peuvent inclure le `kubeletExtraConfig` champ qui accepte un yaml de forme libre qui sera intégré dans le`kubelet.yaml`.

Certains champs du `kubelet.yaml` sont définis par eksctl et ne sont donc pas réinscriptibles, tels que,`address`,`clusterDomain`, `authentication` ou. `authorization` `serverTLSBootstrap`

L'exemple de fichier de configuration suivant crée un groupe de nœuds qui réserve le `300m` vCPU, `300Mi` de la mémoire et `1Gi` du stockage éphémère au kubelet ; le `300m` vCPU`300Mi`, de la mémoire et du stockage éphémère aux démons du système d'exploitation ; `1Gi` et déclenche l'expulsion des pods lorsqu'il y a moins de mémoire disponible ou moins de 10 % du système de fichiers racine. `200Mi`

```
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
```

Dans cet exemple, étant donné les instances de type `m5a.xlarge` 4 V CPUs et 16 GiB de mémoire, la `Allocatable` quantité de mémoire CPUs serait de 3,4 et 15,4 GiB. Il est important de savoir que les valeurs spécifiées dans le fichier de configuration pour les champs de ce fichier `kubeletExtraconfig` remplaceront complètement les valeurs par défaut spécifiées par eksctl. Cependant, si vous omettez un ou plusieurs `kubeReserved` paramètres, les paramètres manquants seront définis par défaut à des valeurs correctes en fonction du type d'instance aws utilisé.

`kubeReserved`calcul

Bien qu'il soit généralement recommandé de configurer une instance mixte NodeGroup pour utiliser des instances ayant la même configuration de processeur et de RAM, cela n'est pas une exigence stricte. Le `kubeReserved` calcul utilise donc la plus petite instance du `InstanceDistribution.InstanceTypes` champ. Ainsi, NodeGroups avec des types d'instances disparates, vous ne réserverez pas trop de ressources sur la plus petite instance. Toutefois, cela peut entraîner une réservation trop petite pour le type d'instance le plus important.

> **Warning**
>
> Par défaut, `eksctl` il est défini`featureGates.RotateKubeletServerCertificate=true`, mais lorsque des `featureGates` paramètres personnalisés sont fournis, ils ne sont pas définis. Vous devez toujours l'inclure`featureGates.RotateKubeletServerCertificate=true`, sauf si vous devez le désactiver.

CloudWatch journalisation

Cette rubrique explique comment configurer la CloudWatch journalisation Amazon pour les composants du plan de contrôle de votre cluster EKS. CloudWatch la journalisation fournit une

visibilité sur les opérations du plan de contrôle de votre cluster, ce qui est essentiel pour résoudre les problèmes, auditer les activités du cluster et surveiller l'état de santé de vos composants Kubernetes.

Activer la CloudWatch journalisation

CloudWatch la journalisation pour le plan de contrôle EKS n'est pas activée par défaut en raison de l'ingestion de données et des coûts de stockage.

Pour activer la journalisation du plan de contrôle lors de la création d'un cluster, vous devez définir le **`cloudWatch.clusterLogging.enableTypes`**paramètre dans votre `ClusterConfig` (voir des exemples ci-dessous).

Ainsi, si vous avez un fichier de configuration avec les **`cloudWatch.clusterLogging.enableTypes`**paramètres corrects, vous pouvez créer un cluster avec`eksctl create cluster --config-file=<path>`.

Si vous avez déjà créé un cluster, vous pouvez utiliser`eksctl utils update-cluster-logging`.

> **Note**
>
> cette commande s'exécute en mode plan par défaut, vous devrez spécifier un `--approve` drapeau pour appliquer les modifications à votre cluster.

Si vous utilisez un fichier de configuration, exécutez :

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

Vous pouvez également utiliser les drapeaux de la CLI.

Pour activer tous les types de journaux, exécutez :

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

Pour activer `audit` les journaux, exécutez :

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

Pour activer tout sauf les `controllerManager` journaux, exécutez :

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

Si les types `api` et `scheduler` log étaient déjà activés, pour les désactiver `scheduler` et les activer `controllerManager` en même temps, exécutez :

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

Cela laissera `api` et sera `controllerManager` le seul type de journal activé.

Pour désactiver tous les types de journaux, exécutez :

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

`ClusterConfig`Exemples

Dans un cluster EKS, le `enableTypes` champ ci-dessous `clusterLogging` peut contenir une liste de valeurs possibles pour activer les différents types de journaux pour les composants du plan de contrôle.

Les valeurs possibles sont les suivantes :

- `api`: active les journaux du serveur d'API Kubernetes.
- `audit`: active les journaux d'audit Kubernetes.
- `authenticator`: active les journaux de l'authentificateur.
- `controllerManager`: Active les journaux du gestionnaire de contrôleurs Kubernetes.
- `scheduler`: active les journaux du planificateur Kubernetes.

Pour en savoir plus, consultez la documentation EKS.

Désactiver tous les journaux

Pour désactiver tous les types, utilisez `[]` ou supprimez complètement la `cloudWatch` section.

Activer tous les journaux

Vous pouvez activer tous les types avec `"*"` ou`"all"`. Par exemple :

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

Activer un ou plusieurs journaux

Vous pouvez activer un sous-ensemble de types en répertoriant les types que vous souhaitez activer. Par exemple :

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

Période de conservation des journaux

Par défaut, les journaux sont stockés dans CloudWatch Logs, indéfiniment. Vous pouvez spécifier le nombre de jours pendant lesquels les journaux du plan de contrôle doivent être conservés dans CloudWatch Logs. L'exemple suivant conserve les journaux pendant 7 jours :

```
cloudWatch:
  clusterLogging:
    logRetentionInDays: 7
```

Exemple complet

```
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
```

Cluster entièrement privé EKS

eksctl prend en charge la création de clusters entièrement privés qui n'ont aucun accès Internet sortant et ne disposant que de sous-réseaux privés. Les points de terminaison VPC sont utilisés pour permettre un accès privé aux services AWS.

Ce guide explique comment créer un cluster privé sans accès Internet sortant.

Création d'un cluster entièrement privé

Le seul champ obligatoire pour créer un cluster entièrement privé est `privateCluster.enabled` le suivant :

```
privateCluster:
  enabled: true
```

Après la création du cluster, les commandes eksctl nécessitant un accès au serveur d'API Kubernetes devront être exécutées depuis le VPC du cluster, un VPC pair ou en utilisant un autre moyen tel qu'AWS Direct Connect. Les commandes eksctl nécessitant un accès à l'EKS APIs ne fonctionneront pas si elles sont exécutées depuis le VPC du cluster. Pour résoudre ce problème, créez un point de terminaison d'interface permettant à Amazon EKS d'accéder de manière privée à la gestion d'Amazon Elastic Kubernetes Service (Amazon EKS APIs) depuis votre Amazon Virtual Private Cloud (VPC). Dans une future version, eksctl ajoutera un support pour créer ce point de terminaison afin qu'il n'ait pas besoin d'être créé manuellement. Les commandes nécessitant un accès à l'URL du fournisseur OpenID Connect devront être exécutées en dehors du VPC de votre cluster une fois que vous aurez activé AWS pour PrivateLink Amazon EKS.

La création de groupes de nœuds gérés continuera de fonctionner, et la création de groupes de nœuds autogérés fonctionnera car elle nécessite un accès au serveur d'API via le point de terminaison de l'interface EKS si la commande est exécutée depuis le VPC du cluster, un VPC pair ou en utilisant un autre moyen tel qu'AWS Direct Connect.

> **Note**
>
> Les points de terminaison VPC sont facturés à l'heure et en fonction de l'utilisation. Vous trouverez de plus amples informations sur les tarifs sur la page PrivateLink des tarifs AWS

> **Warning**
>
> Les clusters entièrement privés ne sont pas pris en charge dans. `eu-south-1`

Configuration de l'accès privé à des services AWS supplémentaires

Pour permettre aux nœuds de travail d'accéder aux services AWS en privé, eksctl crée des points de terminaison VPC pour les services suivants :

- Points de terminaison d'interface pour l'ECR (à la fois `ecr.apiecr.dkr`) afin d'extraire des images de conteneurs (plugin AWS CNI, etc.)
- Un point de terminaison passerelle permettant à S3 d'extraire les couches d'image réelles
- Un point de terminaison d'interface pour EC2 requis par l'intégration `aws-cloud-provider`
- Un point de terminaison d'interface permettant à STS de prendre en charge les rôles Fargate et IAM pour les comptes de services (IRSA)
- Un point de terminaison d'interface pour CloudWatch logging (`logs`) si la CloudWatch journalisation est activée

Ces points de terminaison VPC sont essentiels pour un cluster privé fonctionnel et, par conséquent, eksctl ne prend pas en charge leur configuration ou leur désactivation. Cependant, un cluster peut avoir besoin d'un accès privé à d'autres services AWS (par exemple, Autoscaling requis par le Cluster Autoscaler). Ces services peuvent être spécifiés dans`privateCluster.additionalEndpointServices`, qui indique à eksctl de créer un point de terminaison VPC pour chacun d'entre eux.

Par exemple, pour autoriser un accès privé à Autoscaling et à la CloudWatch journalisation :

```
privateCluster:
  enabled: true
  additionalEndpointServices:
```

```
  # For Cluster Autoscaler
  - "autoscaling"
  # CloudWatch logging
  - "logs"
```

Les points de terminaison pris en charge dans `additionalEndpointServices` sont`autoscaling`, `cloudformation` et. `logs`

Ignorer les créations de terminaux

Si un VPC a déjà été créé avec les points de terminaison AWS nécessaires configurés et liés aux sous-réseaux décrits dans la documentation EKS, `eksctl` vous pouvez ignorer leur création en fournissant l'option suivante : `skipEndpointCreation`

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

Ce paramètre ne peut pas être utilisé conjointement avec`additionalEndpointServices`. Il ignorera toute création de point de terminaison. En outre, ce paramètre n'est recommandé que si la topologie du <# sous-réseau du point de terminaison est correctement configurée. Si les identifiants de sous-réseau sont corrects, le `vpce` routage est configuré avec des adresses préfixes, tous les points de terminaison EKS nécessaires sont créés et liés au VPC fourni. `eksctl`ne modifiera aucune de ces ressources.

Groupes de nœuds

Seuls les groupes de nœuds privés (gérés et autogérés) sont pris en charge dans un cluster entièrement privé, car le VPC du cluster est créé sans aucun sous-réseau public. Le `privateNetworking` champ (`nodeGroup[]`**`.privateNetworking`** et`managedNodeGroup[]`) doit être défini de manière explicite. Le fait de ne `privateNetworking` pas définir cette option dans un cluster entièrement privé constitue une erreur.

```
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
```

Accès au point de terminaison de cluster

Un cluster entièrement privé ne prend pas en charge les modifications `clusterEndpointAccess` lors de la création du cluster. Il s'agit d'une erreur lorsque vous définissez l'une `clusterEndpoints.publicAccess` ou l'autre`clusterEndpoints.privateAccess`, car un cluster entièrement privé ne peut avoir qu'un accès privé, et autoriser la modification de ces champs peut entraîner la rupture du cluster.

VPC et sous-réseaux fournis par l'utilisateur

eksctl prend en charge la création de clusters entièrement privés à l'aide d'un VPC et de sous-réseaux préexistants. Seuls les sous-réseaux privés peuvent être spécifiés et c'est une erreur de spécifier des sous-réseaux sous. `vpc.subnets.public`

eksctl crée des points de terminaison VPC dans le VPC fourni et modifie les tables de routage pour les sous-réseaux fournis. Chaque sous-réseau doit être associé à une table de routage explicite car eksctl ne modifie pas la table de routage principale.

```
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
```

Gestion d'un cluster entièrement privé

Pour que toutes les commandes fonctionnent après la création du cluster, eksctl aura besoin d'un accès privé au point de terminaison du serveur API EKS et d'un accès Internet sortant (pour). `EKS:DescribeCluster` Les commandes qui ne nécessitent pas d'accès au serveur API seront prises en charge si eksctl dispose d'un accès Internet sortant.

Supprimer de force un cluster entièrement privé

Des erreurs sont susceptibles de se produire lors de la suppression d'un cluster entièrement privé via eksctl car eksctl n'a pas automatiquement accès à toutes les ressources du cluster. `--force`existe pour résoudre ce problème : cela forcera la suppression du cluster et continuera en cas d'erreur.

Limitations

L'une des limites de l'implémentation actuelle est qu'eksctl crée initialement le cluster avec l'accès aux points de terminaison publics et privés activé, et désactive l'accès aux points de terminaison

publics une fois toutes les opérations terminées. Cela est nécessaire car eksctl a besoin d'accéder au serveur d'API Kubernetes pour permettre aux nœuds autogérés de rejoindre le cluster et de prendre en charge et Fargate. GitOps Une fois ces opérations terminées, eksctl fait passer l'accès du point de terminaison du cluster à un accès privé uniquement. Cette mise à jour supplémentaire signifie que la création d'un cluster entièrement privé prendra plus de temps que pour un cluster standard. À l'avenir, eksctl pourrait passer à une fonction Lambda compatible VPC pour effectuer ces opérations d'API.

Accès sortant via des serveurs proxy HTTP

eksctl est capable de communiquer avec AWS APIs via un serveur proxy HTTP (S) configuré, mais vous devez vous assurer que vous avez correctement défini votre liste d'exclusion de proxy.

En règle générale, vous devez vous assurer que les demandes relatives au point de terminaison VPC de votre cluster ne sont pas acheminées via vos proxys en définissant une variable d'`no_proxy`environnement appropriée incluant la valeur. `.eks.amazonaws.com`

Si votre serveur proxy effectue une « interception SSL » et que vous utilisez des rôles IAM pour les comptes de service (IRSA), vous devez vous assurer de contourner explicitement le protocole SSL Man-in-the-Middle pour le domaine. `oidc.<region>.amazonaws.com` Dans le cas contraire, eksctl obtiendra l'empreinte numérique du certificat racine incorrecte pour le fournisseur OIDC, et le plug-in AWS VPC CNI ne démarrera pas car il ne pourra pas obtenir les informations d'identification IAM, ce qui rendra votre cluster inopérant.

Plus d'informations

- Clusters privés EKS

Modules complémentaires

Cette rubrique décrit comment gérer les modules complémentaires Amazon EKS pour vos clusters Amazon EKS à l'aide d'eksctl. Les modules complémentaires EKS sont une fonctionnalité qui vous permet d'activer et de gérer le logiciel opérationnel Kubernetes via l'API EKS, simplifiant ainsi le processus d'installation, de configuration et de mise à jour des modules complémentaires de cluster.

> **Warning**
>
> eksctl installe désormais des addons par défaut (vpc-cni, coredns, kube-proxy) en tant qu'addons EKS au lieu d'addons autogérés. Cela signifie que vous devez utiliser des

commandes à la `eksctl update addon` place des `eksctl utils update-*` commandes pour les clusters créés avec eksctl v0.184.0 et versions ultérieures.

Vous pouvez créer des clusters sans aucun addon réseau par défaut lorsque vous souhaitez utiliser des plugins CNI alternatifs tels que Cilium et Calico.

Les modules complémentaires EKS prennent désormais en charge la réception d'autorisations IAM via EKS Pod Identity Associations, ce qui leur permet de se connecter aux services AWS en dehors du cluster

Création d'addons

Eksctl offre plus de flexibilité pour gérer les extensions de cluster :

Dans votre fichier de configuration, vous pouvez spécifier les extensions que vous souhaitez et (si nécessaire) le rôle ou les politiques à leur associer :

```
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: '*'
```

Vous pouvez spécifier au plus l'une des valeurs `attachPolicy` suivantes : `attachPolicyARNs` et`serviceAccountRoleARN`.

Si aucune de ces options n'est spécifiée, l'addon sera créé avec un rôle auquel sont associées toutes les politiques recommandées.

> **Note**
>
> Pour associer des politiques aux modules complémentaires, votre cluster doit avoir été `OIDC` activé. S'il n'est pas activé, nous ignorons les politiques associées.

Vous pouvez ensuite créer ces addons lors du processus de création du cluster :

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

Ou créez les addons de manière explicite après la création du cluster à l'aide du fichier de configuration ou des indicateurs de la CLI :

```
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**
>
> Utilisez l'`--namespace-config`indicateur pour déployer des modules complémentaires dans un espace de noms personnalisé au lieu de l'espace de noms par défaut.

Lors de la création de l'addon, si une version autogérée de l'addon existe déjà sur le cluster, vous pouvez choisir comment les `configMap` conflits potentiels doivent être résolus en définissant une `resolveConflicts` option via le fichier de configuration, par ex.

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

Pour la création d'un addon, le `resolveConflicts` champ prend en charge trois valeurs distinctes :

- `none`- EKS ne modifie pas la valeur. La création risque d'échouer.
- `overwrite`- EKS remplace toutes les modifications de configuration aux valeurs par défaut d'EKS.
- `preserve`- EKS ne modifie pas la valeur. La création risque d'échouer. (Identique`none`, mais différent de la mise `preserve` à jour des addons).

Répertorier les addons activés

Vous pouvez voir quels modules complémentaires sont activés dans votre cluster en exécutant :

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

or

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

Configuration de la version de l'addon

La définition de la version de l'addon est facultative. Si le `version` champ est laissé vide, la version par défaut de l'addon `eksctl` sera résolue. Vous trouverez plus d'informations sur la version par

défaut pour des modules complémentaires spécifiques dans la documentation AWS relative à EKS. Notez que la version par défaut n'est pas nécessairement la dernière version disponible.

La version de l'addon peut être définie sur. `latest` La version peut également être définie avec la balise de construction EKS spécifiée, telle que `v1.7.5-eksbuild.1` ou`v1.7.5-eksbuild.2`. Il peut également être défini sur la version finale de l'addon, telle que `v1.7.5` ou`1.7.5`, et la balise de `eksbuild` suffixe sera découverte et définie pour vous.

Consultez la section ci-dessous pour savoir comment découvrir les extensions disponibles et leurs versions.

Découvrir les addons

Vous pouvez découvrir quels modules complémentaires peuvent être installés sur votre cluster en exécutant :

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

Cela permettra de découvrir la version de Kubernetes de votre cluster et de filtrer en fonction de celle-ci. Sinon, si vous voulez voir quels addons sont disponibles pour une version particulière de Kubernetes, vous pouvez exécuter :

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

Vous pouvez également découvrir des addons en filtrant sur leur`type`, `owner` and/or `publisher`. Par exemple, pour voir les addons pour un propriétaire et un type particuliers, vous pouvez exécuter :

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

Les `types` `publishers` indicateurs `owners` et sont facultatifs et peuvent être spécifiés ensemble ou individuellement pour filtrer les résultats.

Découverte du schéma de configuration des addons

Après avoir découvert l'addon et la version, vous pouvez consulter les options de personnalisation en récupérant son schéma de configuration JSON.

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

Cela renvoie un schéma JSON des différentes options disponibles pour cet addon.

Utilisation des valeurs de configuration

`ConfigurationValues`peuvent être fournis dans le fichier de configuration lors de la création ou de la mise à jour des addons. Seuls les formats JSON et YAML sont pris en charge.

Pour, par exemple. ,

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

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

> **Note**
>
> N'oubliez pas que lorsque les valeurs de configuration des modules complémentaires sont modifiées, des conflits de configuration peuvent survenir.

```
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`.
```

De plus, la commande get sera désormais également récupérée `ConfigurationValues` pour l'addon. par exemple

```
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
```

Utilisation d'un espace de noms personnalisé

Un espace de noms personnalisé peut être fourni dans le fichier de configuration lors de la création des addons. Un espace de noms ne peut pas être mis à jour une fois qu'un addon est créé.

Utilisation du fichier de configuration

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

Utilisation de l'indicateur CLI

Vous pouvez également spécifier un espace de noms personnalisé à l'aide de l'`--namespace-config`indicateur :

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

La commande get récupérera également la valeur de l'espace de noms pour l'addon

```
- ConfigurationValues: ""
  IAMRole: ""
  Issues: null
  Name: aws-ebs-csi-driver
  NamespaceConfig:
    namespace: custom-namespace
  NewerVersion: ""
  PodIdentityAssociations: null
  Status: ACTIVE
```

```
  Version: v1.47.0-eksbuild.1
```

Mise à jour des addons

Vous pouvez mettre à jour vos modules complémentaires vers des versions plus récentes et modifier les politiques associées en exécutant :

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

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

> **Note**
>
> La configuration de l'espace de noms ne peut pas être mise à jour une fois qu'un addon est créé. Le `--namespace-config` drapeau n'est disponible que lors de la création de l'addon.

Comme pour la création d'un addon, lors de la mise à jour d'un addon, vous avez un contrôle total sur les modifications de configuration que vous avez peut-être appliquées précédemment à ce module complémentaire. `configMap` Plus précisément, vous pouvez les conserver ou les remplacer. Cette fonctionnalité optionnelle est disponible via le même champ du fichier de configuration`resolveConflicts`. Par exemple,

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

Pour la mise à jour de l'addon, le `resolveConflicts` champ accepte trois valeurs distinctes :

- none- EKS ne modifie pas la valeur. La mise à jour risque d'échouer.
- `overwrite`- EKS remplace toutes les modifications de configuration aux valeurs par défaut d'EKS.
- `preserve`- EKS préserve la valeur. Si vous choisissez cette option, nous vous recommandons de tester les modifications de champ et de valeur sur un cluster hors production avant de mettre à jour le module complémentaire sur votre cluster de production.

Supprimer des addons

Vous pouvez supprimer un addon en exécutant :

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

Cela supprimera l'addon et tous les rôles IAM qui lui sont associés.

Lorsque vous supprimez votre cluster, tous les rôles IAM associés aux addons sont également supprimés.

Flexibilité de création de clusters pour les extensions réseau par défaut

Lorsqu'un cluster est créé, EKS installe automatiquement VPC CNI, CoreDNS et kube-proxy en tant qu'addons autogérés. Pour désactiver ce comportement afin d'utiliser d'autres plugins CNI tels que Cilium et Calico, eksctl prend désormais en charge la création d'un cluster sans aucun addon réseau par défaut. Pour créer un tel cluster, définissez`addonsConfig.disableDefaultAddons`, comme dans :

```
addonsConfig:
  disableDefaultAddons: true
```

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

Pour créer un cluster avec uniquement CoreDNS et kube-proxy et non VPC CNI, spécifiez les addons explicitement et définissez-les, comme dans : `addons` `addonsConfig.disableDefaultAddons`

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

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

Dans le cadre de cette modification, eksctl installe désormais des addons par défaut en tant qu'addons EKS au lieu d'addons autogérés lors de la création du cluster s'il n'`addonsConfig.disableDefaultAddons`est pas explicitement défini sur true. Ainsi, `eksctl`

`utils update-*` les commandes ne peuvent plus être utilisées pour mettre à jour les addons pour les clusters créés avec eksctl v0.184.0 et versions ultérieures :

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

Au lieu de cela, `eksctl update addon` devrait être utilisé maintenant.

Pour en savoir plus, consultez Amazon EKS qui introduit la flexibilité de création de clusters pour les extensions réseau.

Activation de l'accès pour Amazon EMR

Afin de permettre à EMR d'effectuer des opérations sur l'API Kubernetes, son SLR doit disposer des autorisations RBAC requis. eksctl fournit une commande qui crée les ressources RBAC requises pour EMR et les met à jour pour lier le rôle au SLR pour EMR. `aws-auth` ConfigMap

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

Support EKS Fargate

AWS Fargate est un moteur de calcul géré pour Amazon ECS capable d'exécuter des conteneurs. Dans Fargate, vous n'avez pas besoin de gérer des serveurs ou des clusters.

Amazon EKS peut désormais lancer des pods sur AWS Fargate. Vous n'avez donc plus à vous soucier de la manière dont vous provisionnez ou gérez l'infrastructure pour les pods et facilite la création et l'exécution d'applications Kubernetes performantes et hautement disponibles sur AWS.

Création d'un cluster avec le support de Fargate

Vous pouvez ajouter un cluster compatible avec Fargate avec :

```
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
```

Cette commande aura créé un cluster et un profil Fargate. Ce profil contient certaines informations dont AWS a besoin pour instancier des pods dans Fargate. Il s'agit des types suivants :

- rôle d'exécution du pod pour définir les autorisations requises pour exécuter le pod et l'emplacement réseau (sous-réseau) pour exécuter le pod. Cela permet d'appliquer les mêmes autorisations réseau et de sécurité à plusieurs pods Fargate et facilite la migration des pods existants d'un cluster vers Fargate.
- Sélecteur pour définir quels pods doivent fonctionner sur Fargate. Il est composé d'un `namespace` et`labels`.

Lorsque le profil n'est pas spécifié mais que la prise en charge de Fargate est activée, `--fargate` un profil Fargate par défaut est créé. Ce profil cible les espaces de noms `default` et les `kube-system` espaces de noms afin que les pods de ces espaces de noms s'exécutent sur Fargate.

Le profil Fargate créé peut être vérifié à l'aide de la commande suivante :

```
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
```

Pour en savoir plus sur les sélecteurs, voir Conception de profils Fargate.

Création d'un cluster compatible avec Fargate à l'aide d'un fichier de configuration

Le fichier de configuration suivant déclare un cluster EKS avec à la fois un groupe de nœuds composé d'une `m5.large` instance EC2 et de deux profils Fargate. Tous les pods définis dans les `kube-system` espaces de noms `default` et s'exécuteront sur Fargate. Tous les pods de l'espace de `dev` noms qui portent également l'étiquette `dev=passed` fonctionneront également sur Fargate. Tous les autres pods seront planifiés sur le nœud `enng-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
```

Conception de profils Fargate

Chaque entrée du sélecteur comporte jusqu'à deux composants, un espace de noms et une liste de paires clé-valeur. Seul le composant namespace est requis pour créer une entrée de

sélection. Toutes les règles (espaces de noms, paires clé-valeur) doivent s'appliquer à un pod pour correspondre à une entrée de sélecteur. Un pod ne doit correspondre qu'à une seule entrée de sélecteur pour être exécuté sur le profil. Tout module répondant à toutes les conditions d'un champ de sélection sera programmé pour être exécuté sur Fargate. Tous les pods ne correspondant à aucun des espaces de noms figurant sur la liste blanche mais sur lesquels l'utilisateur a défini manuellement le fichier scheduler : fargate-scheduler seraient bloqués dans l'état En attente, car ils n'étaient pas autorisés à s'exécuter sur Fargate.

Les profils doivent répondre aux exigences suivantes :

- Un sélecteur est obligatoire par profil
- Chaque sélecteur doit inclure un espace de noms ; les libellés sont facultatifs

Exemple : planification de la charge de travail dans Fargate

Pour planifier des pods sur Fargate dans l'exemple mentionné ci-dessus, on peut, par exemple, créer un espace de noms `dev` appelé et y déployer la charge de travail :

```
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
```

À partir de la sortie de la dernière `kubectl get pods` commande, nous pouvons voir que le `nginx` pod est déployé dans un nœud appelé`fargate-ip-192-168-183-140.ap-northeast-1.compute.internal`.

Gestion des profils Fargate

Pour déployer des charges de travail Kubernetes sur Fargate, EKS a besoin d'un profil Fargate. Lorsque vous créez un cluster, comme dans les exemples ci-dessus, `eksctl` prenez soin de cela en créant un profil par défaut. Étant donné qu'un cluster existe déjà, il est également possible de créer un profil Fargate à l'aide de la commande suivante : `eksctl create fargateprofile`

> **Note**
>
> Cette opération n'est prise en charge que sur les clusters qui s'exécutent sur la version `eks.5` ou supérieure de la plate-forme EKS.

> **Note**
>
> Si le profil existant a été créé avec une version `eksctl` antérieure à 0.11.0, vous devrez l'exécuter `eksctl upgrade cluster` avant de créer le profil Fargate.

```
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"
```

Vous pouvez également spécifier le nom du profil Fargate à créer. Ce nom ne doit pas commencer par le préfixe`eks-`.

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

En utilisant cette commande avec les drapeaux de la CLI, eksctl ne peut créer qu'un seul profil Fargate avec un simple sélecteur. Pour les sélecteurs plus complexes, par exemple avec plus d'espaces de noms, eksctl prend en charge l'utilisation d'un fichier de configuration :

```
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
```

Pour voir les profils Fargate existants dans un cluster :

```
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
```

Et pour les voir en `yaml` format :

```
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
```

Ou au `json` format :

```
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"
        ]
    }
]
```

Les profils Fargate sont immuables par conception. Pour changer quelque chose, créez un nouveau profil Fargate avec les modifications souhaitées et supprimez l'ancien avec la commande comme dans `eksctl delete fargateprofile` l'exemple suivant :

```
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"
```

```
}
```

Notez que la suppression du profil est un processus qui peut prendre jusqu'à quelques minutes. Lorsque l'`--wait`indicateur n'est pas spécifié, il s'attend à ce que le profil soit supprimé de `eksctl` manière optimiste et le renvoie dès que la demande d'API AWS a été envoyée. Pour faire `eksctl` attendre que le profil soit correctement supprimé, utilisez `--wait` comme dans l'exemple ci-dessus.

Suggestions de lecture

- AWS Fargate
- Amazon EKS peut désormais lancer des pods sur AWS Fargate

Améliorations de clusters

Un cluster géré par « eksctl » peut être mis à niveau en 3 étapes simples :

1. mise à niveau de la version du plan de contrôle avec `eksctl upgrade cluster`
2. mettre à niveau les groupes de nœuds
3. mettez à jour les modules complémentaires réseau par défaut (pour plus d'informations, voirthe section called “Mises à jour complémentaires par défaut”) :

Consultez attentivement les ressources relatives à la mise à niveau du cluster :

- Mettre à jour le cluster existant vers la nouvelle version de Kubernetes dans le guide de l'utilisateur Amazon EKS
- Meilleures pratiques pour les mises à niveau de clusters dans le guide des meilleures pratiques EKS

> **Note**
>
> L'ancien `eksctl update cluster` sera obsolète. Utilisez `eksctl upgrade cluster` à la place.

Mettre à jour la version du plan de contrôle

Les mises à niveau des versions du plan de contrôle doivent être effectuées pour une version mineure à la fois.

Pour mettre à niveau le plan de contrôle vers la prochaine version disponible, exécutez :

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

Cette commande n'appliquera aucune modification immédiatement, vous devrez la réexécuter `--approve` pour appliquer les modifications.

La version cible pour la mise à niveau du cluster peut être spécifiée à la fois avec l'indicateur CLI :

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

ou avec le fichier de configuration

```
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**
>
> Les seules valeurs autorisées pour les `metadata.version` arguments `--version` et sont la version actuelle du cluster ou une version supérieure. Les mises à niveau de plusieurs versions de Kubernetes ne sont pas prises en charge.

Mises à jour complémentaires par défaut

Cette rubrique explique comment mettre à jour les modules complémentaires préinstallés par défaut inclus dans les clusters EKS.

> **Warning**
>
> eksctl installe désormais les addons par défaut en tant qu'addons EKS au lieu d'addons autogérés. En savoir plus sur ses implications en termes de flexibilité de création de clusters pour les extensions réseau par défaut.
> Pour la mise à jour des addons, cela `eksctl utils update-<addon>` ne peut pas être utilisé pour les clusters créés avec eksctl v0.184.0 et versions ultérieures. Ce guide n'est valable que pour les clusters créés avant cette modification.

Trois modules complémentaires par défaut sont inclus dans chaque cluster EKS :

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

Mettre à jour le module complémentaire préinstallé

Pour les addons EKS officiels créés manuellement lors de `eksctl create addons` ou lors de la création d'un cluster, la méthode de gestion est la suivante. `eksctl create/get/update/delete addon` Dans de tels cas, veuillez consulter la documentation sur les modules complémentaires EKS.

Le processus de mise à jour de chacune d'entre elles est différent, il y a donc 3 commandes distinctes que vous devrez exécuter. Toutes les commandes suivantes sont acceptées`--config-file`. Par défaut, chacune de ces commandes s'exécute en mode plan. Si vous êtes satisfait des modifications proposées, réexécutez-la avec`--approve`.

Pour effectuer la mise à jour`kube-proxy`, exécutez :

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

Pour effectuer la mise à jour`aws-node`, exécutez :

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

Pour effectuer la mise à jour`coredns`, exécutez :

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

Une fois la mise à niveau effectuée, assurez-vous de lancer `kubectl get pods -n kube-system` et de vérifier si tous les modules complémentaires sont prêts. Vous devriez voir quelque chose comme ceci :

```
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
```

Support du changement de zone dans les clusters EKS

EKS prend désormais en charge le changement de zone et le décalage automatique de zone Amazon Application Recovery Controller (ARC) qui améliorent la résilience des environnements de clusters multi-AZ. Avec AWS Zonal Shift, les clients peuvent déplacer le trafic interne au cluster hors d'une zone de disponibilité réduite, garantissant ainsi le lancement de nouveaux pods et nœuds Kubernetes uniquement dans des zones de disponibilité saines.

Création d'un cluster avec le décalage de zone activé

```
# 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
```

Activation du changement de zone sur un cluster existant

Pour activer ou désactiver le décalage de zone sur un cluster existant, exécutez

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

ou sans fichier de configuration :

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

Plus d'informations

- Changement de zone EKS

Support pour charpentiers

`eksctl`fournit un support pour ajouter Karpenter à un cluster nouvellement créé. Cela créera tous les prérequis nécessaires décrits dans la section Getting Started de Karpenter, y compris l'installation de Karpenter lui-même à l'aide de Helm. Nous prenons actuellement en charge l'installation de versions`0.28.0+`. Consultez la section sur la compatibilité avec Karpenter pour plus de détails.

La configuration de cluster suivante décrit une installation Karpenter typique :

```
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
```

La version est la version de Karpenter telle qu'elle se trouve dans leur référentiel Helm. Les options suivantes peuvent également être définies :

```
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
```

OIDC doit être défini pour installer Karpenter.

Une fois Karpenter correctement installé, ajoutez NodePool(s) et NodeClass(s) pour permettre à Karpenter de commencer à ajouter des nœuds au cluster.

La NodePool `nodeClassRef` section 'doit correspondre au nom d'un`EC2NodeClass`. Par exemple :

```
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
```

Notez que vous devez spécifier l'un `role` ou `instanceProfile` l'autre des nœuds de lancement. Si vous choisissez d'utiliser `instanceProfile` le nom du profil créé, `eksctl` suivez le modèle :`eksctl-KarpenterNodeInstanceProfile-<cluster-name>`.

Balisage automatique des groupes de sécurité

`eksctl`balise automatiquement le groupe de sécurité du nœud partagé du cluster `karpenter.sh/discovery` lorsque Karpenter est activé (`karpenter.version`spécifié) et que le `karpenter.sh/`

`discovery` tag existe dans`metadata.tags`. Cela permet la compatibilité avec AWS Load Balancer Controller.

Remarque : avec Karpenter 0.32.0+, les Provisioners sont devenus obsolètes et remplacés par. NodePool

Schéma de configuration du cluster

> **Note**
>
> L'emplacement du schéma est en cours de migration.

Vous pouvez utiliser un fichier yaml pour créer un cluster. Consultez la référence du schéma.

Par exemple :

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

La référence du schéma de ce fichier est disponible sur GitHub.

Pour plus d'informations sur l'utilisation du fichier, consultezthe section called “Création et gestion de clusters”.

Groupes de nœuds

Ce chapitre contient des informations sur la façon dont vous créez et configurez des groupes de nœuds avec Eksctl. Les groupes de nœuds sont des groupes d'instances EC2 attachés à un cluster EKS.

Rubriques :

- the section called “Instances Spot”
  - Créez et gérez des clusters EKS avec des instances Spot à l'aide de groupes de nœuds gérés
  - Configurez les instances Spot pour les groupes de nœuds non gérés à l'aide du MixedInstancesPolicy
  - Distinguer les instances Spot et On-Demand à l'aide du label `node-lifecycle` Kubernetes
- the section called “Auto Scaling”
  - Activez le dimensionnement automatique des nœuds du cluster Kubernetes en créant un cluster ou un groupe de nœuds doté du rôle IAM qui autorise l'utilisation de l'autoscaler du cluster
  - Configurer les définitions de groupes de nœuds pour inclure les balises et les annotations nécessaires pour que l'autoscaler du cluster puisse redimensionner le groupe de nœuds
  - Créez des groupes de nœuds distincts pour chaque zone de disponibilité si les charges de travail ont des exigences spécifiques à une zone, telles que des règles d'affinité ou de stockage spécifiques à une zone
- the section called “Groupes de nœuds gérés par EKS”
  - Fournir et gérer des instances EC2 (nœuds) pour les clusters Amazon EKS Kubernetes
  - Appliquez facilement des corrections de bogues, des correctifs de sécurité et des nœuds de mise à jour aux dernières versions de Kubernetes
- the section called “Nœuds hybrides EKS”
  - Permettez l'exécution d'applications sur site et de périphérie sur une infrastructure gérée par le client avec les mêmes clusters, fonctionnalités et outils AWS EKS que ceux utilisés dans le cloud AWS
  - Configurer le réseau pour connecter des réseaux sur site à un VPC AWS, à l'aide d'options telles qu'AWS VPN ou Site-to-Site AWS Direct Connect
  - Configurez les informations d'identification pour les nœuds distants afin de s'authentifier auprès du cluster EKS, à l'aide d'AWS Systems Manager (SSM) ou d'AWS IAM Roles Anywhere

- the section called “Config de réparation du nœud”
  - Activation de la réparation des nœuds pour les groupes de nœuds gérés par EKS afin de surveiller et de remplacer ou redémarrer automatiquement les nœuds de travail défectueux
- the section called “Support ARM”
  - Créez un cluster EKS avec des instances Graviton basées sur ARM pour améliorer les performances et la rentabilité
- the section called “Souillures”
  - Appliquer des altérations à des groupes de nœuds spécifiques dans un cluster Kubernetes
  - Contrôlez la planification et l'expulsion des pods en fonction des touches, des valeurs et des effets altérés
- the section called “Support de modèle de lancement”
  - Lancement de groupes de nœuds gérés à l'aide d'un modèle de lancement EC2 fourni
  - Mise à niveau d'un groupe de nœuds gérés pour utiliser une version différente d'un modèle de lancement
  - Comprendre les limites et les facteurs à prendre en compte lors de l'utilisation de modèles personnalisés AMIs et de modèles de lancement avec des groupes de nœuds gérés
- the section called “Travailler avec des groupes de nœuds”
  - Activer l'accès SSH aux instances EC2 du groupe de nœuds
  - Augmenter ou diminuer le nombre de nœuds d'un groupe de nœuds
- the section called “Sous-réseaux personnalisés”
  - Étendre un VPC existant avec un nouveau sous-réseau et ajouter un groupe de nœuds à ce sous-réseau
- the section called “Démarrage des nœuds”
  - Comprendre le nouveau processus d'initialisation des nœuds (nodeadm) introduit en 2023 AmazonLinux
  - Découvrez les NodeConfig paramètres par défaut appliqués par eksctl pour les nœuds autogérés et gérés par EKS
  - Personnalisez le processus d'amorçage des nœuds en overrideBootstrapCommand fournissant un NodeConfig
- the section called “Groupes de nœuds non gérés”
  - Création ou mise à jour de groupes de nœuds non gérés dans un cluster EKS

- Mettre à jour les modules complémentaires Kubernetes par défaut tels que kube-proxy, aws-node et CoreDNS

- the section called “Support du GPU”
  - Eksctl prend en charge la sélection de types d'instances GPU pour les groupes de nœuds, ce qui permet d'utiliser des charges de travail accélérées par GPU sur les clusters EKS.
  - Eksctl installe automatiquement le plug-in de périphérique NVIDIA Kubernetes lorsqu'un type d'instance compatible GPU est sélectionné, facilitant ainsi l'utilisation des ressources GPU dans le cluster.
  - Les utilisateurs peuvent désactiver l'installation automatique du plug-in et installer manuellement une version spécifique du plug-in pour appareil NVIDIA Kubernetes à l'aide des commandes fournies.
- the section called “Sélecteur d'instance”
  - Générez automatiquement une liste des types d'instances EC2 appropriés en fonction de critères de ressources tels que v CPUs GPUs, mémoire et architecture du processeur
  - Créez des clusters et des groupes de nœuds avec les types d'instances correspondant aux critères de sélection d'instance spécifiés
  - Effectuez un essai à sec pour inspecter et modifier les types d'instances correspondant au sélecteur d'instance avant de créer un groupe de nœuds
- the section called “Mappages de volumes supplémentaires”
  - Configurer des mappages de volumes supplémentaires pour un groupe de nœuds gérés dans un cluster EKS
  - Personnalisez les propriétés des volumes telles que la taille, le type, le chiffrement, les IOPS et le débit pour les volumes supplémentaires
  - Joindre des instantanés EBS existants en tant que volumes supplémentaires au groupe de nœuds
- the section called “Nœuds Windows Worker”
  - Ajouter des groupes de nœuds Windows à un cluster Linux Kubernetes existant pour permettre l'exécution de charges de travail Windows
  - Planifiez les charges de travail sur le système d'exploitation approprié (Windows ou Linux) à l'aide de sélecteurs de nœuds basés sur les étiquettes et `kubernetes.io/os` `kubernetes.io/arch`
- the section called “Support personnalisé pour les AMI”

- Utilisez l'`--node-ami`indicateur pour spécifier une AMI personnalisée pour les groupes de nœuds, demandez à AWS la dernière AMI optimisée pour EKS ou utilisez AWS Systems Manager Parameter Store pour trouver l'AMI.
- Définissez l'`--node-ami-family`indicateur pour spécifier la famille de systèmes d'exploitation pour l'AMI du groupe de nœuds, par exemple AmazonLinux 2, Ubuntu2204 ou 2022. WindowsServer CoreContainer
- Pour les groupes de nœuds Windows, spécifiez une AMI personnalisée et fournissez un script d' PowerShell amorçage via le`overrideBootstrapCommand`.

- the section called “DNS personnalisé”
  - Remplacer l'adresse IP du serveur DNS utilisée pour les recherches DNS internes et externes

Travailler avec des groupes de nœuds

Création de groupes de nœuds

Vous pouvez ajouter un ou plusieurs groupes de nœuds en plus du groupe de nœuds initial créé avec le cluster.

Pour créer un groupe de nœuds supplémentaire, utilisez :

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

> **Note**
>
> `--version`l'indicateur n'est pas pris en charge pour les groupes de nœuds gérés. Il hérite toujours de la version du plan de contrôle.

Par défaut, les nouveaux groupes de nœuds non gérés héritent de la version du plan de contrôle (`--version=auto`), mais vous pouvez spécifier une version différente, que vous pouvez également utiliser `--version=latest` pour forcer l'utilisation de la version la plus récente.

De plus, vous pouvez utiliser le même fichier de configuration que celui utilisé pour `eksctl create cluster` :

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

Création d'un groupe de nœuds à partir d'un fichier de configuration

Les groupes de nœuds peuvent également être créés via une définition de cluster ou un fichier de configuration. À partir de l'exemple de fichier de configuration suivant et d'un cluster existant appelé `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
```

Les groupes de nœuds `ng-1-workers` `ng-2-builders` peuvent être créés à l'aide de cette commande :

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

Équilibrage de charge

Si vous avez déjà préparé l'association de groupes or/and cibles d'équilibreurs de charge classiques existants aux groupes de nœuds, vous pouvez les spécifier dans le fichier de configuration. Les groupes or/and cibles des équilibreurs de charge classiques sont automatiquement associés à l'ASG lors de la création de groupes de nœuds. Ceci n'est pris en charge que pour les groupes de nœuds autogérés définis via le champ. `nodeGroups`

```
# 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
```

Sélection de groupes de nœuds dans les fichiers de configuration

Pour effectuer une `delete` opération `create` ou uniquement sur un sous-ensemble des groupes de nœuds spécifiés dans un fichier de configuration, deux indicateurs CLI acceptent une liste de globs, `0` par exemple : 1

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

En utilisant l'exemple de fichier de configuration ci-dessus, on peut créer tous les groupes de nœuds de travail sauf celui des travailleurs avec la commande suivante :

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

Ou on pourrait supprimer le groupe de nœuds du constructeur avec :

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

Dans ce cas, nous devons également fournir la `--approve` commande pour supprimer réellement le groupe de nœuds.

Inclure et exclure des règles

- si non `--include` ou si `--exclude` c'est spécifié, tout est inclus
- si seul `--include` est spécifié, seuls les groupes de nœuds correspondant à ces globs seront inclus
- si seul `--exclude` est spécifié, tous les groupes de nœuds qui ne correspondent pas à ces globs sont inclus
- si les deux sont spécifiées, `--exclude` les règles ont priorité `--include` (c'est-à-dire que les groupes de nœuds qui correspondent aux règles des deux groupes seront exclus)

Répertorier les groupes de nœuds

Pour répertorier les détails d'un groupe de nœuds ou de tous les groupes de nœuds, utilisez :

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

Pour répertorier un ou plusieurs groupes de nœuds au format YAML ou JSON, qui génèrent plus d'informations que la table de journal par défaut, utilisez :

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

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

Immuabilité des groupes de nœuds

De par leur conception, les groupes de nœuds sont immuables. Cela signifie que si vous devez modifier quelque chose (autre que le dimensionnement) comme l'AMI ou le type d'instance d'un groupe de nœuds, vous devez créer un nouveau groupe de nœuds avec les modifications souhaitées, déplacer la charge et supprimer l'ancien. Consultez la section Supprimer et vider des groupes de nœuds.

Dimensionnement des groupes de nœuds

La mise à l'échelle des groupes de nœuds est un processus qui peut prendre jusqu'à quelques minutes. Lorsque l'`--wait`indicateur n'est pas spécifié, il s'attend à ce que le groupe de nœuds soit redimensionné de `eksctl` manière optimiste et renvoie le résultat dès que la demande d'API AWS a été envoyée. Pour faire `eksctl` attendre que les nœuds soient disponibles, ajoutez un `--wait` drapeau comme dans l'exemple ci-dessous.

> **Note**
>
> La mise à l'échelle d'un groupe de nœuds down/in (c'est-à-dire la réduction du nombre de nœuds) peut entraîner des erreurs car nous nous appuyons uniquement sur les modifications apportées à l'ASG. Cela signifie que le ou les nœuds removed/terminated ne sont pas explicitement drainés. Cela pourrait être un domaine à améliorer à l'avenir.

Le dimensionnement d'un groupe de nœuds géré s'effectue en appelant directement l'API EKS qui met à jour la configuration d'un groupe de nœuds gérés.

Dimensionnement d'un seul groupe de nœuds

Un groupe de nœuds peut être redimensionné à l'aide de la commande : `eksctl scale nodegroup`

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

Par exemple, pour étendre le groupe de nœuds `ng-a345f4e1` `cluster-1` à 5 nœuds, exécutez :

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

Un groupe de nœuds peut également être redimensionné en utilisant un fichier de configuration transmis `--config-file` et en spécifiant le nom du groupe de nœuds avec lequel le dimensionnement doit être effectué. `--name` Eksctl recherchera le fichier de configuration et découvrira ce groupe de nœuds ainsi que ses valeurs de configuration de dimensionnement.

Si le nombre de nœuds souhaité se situe NOT dans la plage du nombre minimum actuel et du nombre maximum actuel de nœuds, une erreur spécifique sera affichée. Ces valeurs peuvent également être transmises avec des drapeaux `--nodes-min` et `--nodes-max` respectivement.

Mise à l'échelle de plusieurs groupes de nœuds

Eksctl peut découvrir et redimensionner tous les groupes de nœuds trouvés dans un fichier de configuration transmis avec. `--config-file`

Comme pour le dimensionnement d'un seul groupe de nœuds, le même ensemble de validations s'applique à chaque groupe de nœuds. Par exemple, le nombre de nœuds souhaité doit être compris entre le nombre minimum et maximum de nœuds.

Suppression et vidange de groupes de nœuds

Pour supprimer un groupe de nœuds, exécutez :

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

Les règles d'inclusion et d'exclusion peuvent également être utilisées avec cette commande.

> **Note**
>
> Cela drainera tous les pods de ce groupe de nœuds avant que les instances ne soient supprimées.

Pour ignorer les règles d'expulsion pendant le processus de vidange, exécutez :

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

Tous les nœuds sont bouclés et tous les pods sont expulsés d'un groupe de nœuds lors de la suppression, mais si vous devez vider un groupe de nœuds sans le supprimer, exécutez :

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

Pour débloquer un groupe de nœuds, exécutez :

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

Pour ignorer les règles d'expulsion telles que PodDisruptionBudget les paramètres, exécutez :

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

Pour accélérer le processus de vidange, vous pouvez spécifier `--parallel <value>` le nombre de nœuds à drainer en parallèle.

Autres fonctions

Vous pouvez également activer l'accès SSH, ASG et d'autres fonctionnalités pour un groupe de nœuds, par exemple :

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

Mettre à jour les étiquettes

Il n'existe aucune commande spécifique `eksctl` pour mettre à jour les étiquettes d'un groupe de nœuds, mais cela peut être facilement réalisé en utilisant`kubectl`, par exemple :

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

Accès SSH

Vous pouvez activer l'accès SSH pour les groupes de nœuds en configurant l'un des `publicKeyName` et `publicKeyPath` dans la configuration de `publicKey` votre groupe de nœuds.

Vous pouvez également utiliser AWS Systems Manager (SSM) pour accéder à des nœuds en SSH, en configurant le groupe de nœuds avec : `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
```

Groupes de nœuds non gérés

Dans`eksctl`, définir `--managed=false` ou utiliser le `nodeGroups` champ crée un groupe de nœuds non géré. N'oubliez pas que les groupes de nœuds non gérés n'apparaissent pas dans la console EKS qui, en règle générale, ne connaît que les groupes de nœuds gérés par EKS.

Vous ne devriez mettre à niveau les groupes de nœuds qu'après avoir exécuté. `eksctl upgrade cluster` (Voir Mise à niveau des clusters.)

Si vous avez un cluster simple avec juste un groupe de nœuds initial (c'est-à-dire créé avec`eksctl create cluster`), le processus est très simple :

1. Obtenez le nom de l'ancien groupe de nœuds :

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

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

2. Créez un nouveau groupe de nœuds :

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

3. Supprimez l'ancien groupe de nœuds :

```
 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.
> ```

Mise à jour de plusieurs groupes de nœuds

Si vous avez plusieurs groupes de nœuds, il est de votre responsabilité de suivre la configuration de chacun d'entre eux. Vous pouvez le faire en utilisant des fichiers de configuration, mais si vous ne les avez pas déjà utilisés, vous devrez inspecter votre cluster pour savoir comment chaque groupe de nœuds a été configuré.

D'une manière générale, vous souhaitez :

- vérifiez quels groupes de nœuds vous avez et lesquels peuvent être supprimés ou doivent être remplacés pour la nouvelle version
- notez la configuration de chaque groupe de nœuds, pensez à utiliser un fichier de configuration pour faciliter les mises à niveau la prochaine fois

Mise à jour avec le fichier de configuration

Si vous utilisez un fichier de configuration, vous devez effectuer les opérations suivantes.

Modifiez le fichier de configuration pour ajouter de nouveaux groupes de nœuds et supprimez les anciens groupes de nœuds. Si vous souhaitez simplement mettre à niveau les groupes de nœuds et conserver la même configuration, vous pouvez simplement modifier les noms des groupes de nœuds, par exemple les ajouter `-v2` au nom.

Pour créer tous les nouveaux groupes de nœuds définis dans le fichier de configuration, exécutez :

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

Une fois que vous avez mis en place de nouveaux groupes de nœuds, vous pouvez supprimer les anciens :

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

> **Note**
>
> La première exécution est en mode plan, si vous êtes satisfait des modifications proposées, réexécutez avec`--approve`.

Mettre à jour les extensions par défaut

Vous devrez peut-être mettre à jour les modules complémentaires réseau installés sur votre cluster. Pour de plus amples informations, veuillez consulter the section called “Mises à jour complémentaires par défaut”.

Groupes de nœuds gérés par EKS

Les groupes de nœuds gérés par Amazon EKS sont une fonctionnalité qui automatise le provisionnement et la gestion du cycle de vie des nœuds (instances EC2) pour les clusters Amazon EKS Kubernetes. Les clients peuvent fournir des groupes de nœuds optimisés pour leurs clusters et EKS tiendra leurs nœuds à jour avec les dernières versions de Kubernetes et du système d'exploitation hôte.

Un groupe de nœuds géré par EKS est un groupe de mise à l'échelle automatique et les instances EC2 associées qui sont gérés par AWS pour un cluster Amazon EKS. Chaque groupe de nœuds utilise l'AMI Amazon Linux 2 optimisée pour Amazon EKS. Amazon EKS permet d'appliquer facilement des corrections de bogues et des correctifs de sécurité aux nœuds, ainsi que de les mettre à jour vers les dernières versions de Kubernetes. Chaque groupe de nœuds lance un groupe de mise à l'échelle automatique pour votre cluster, qui peut couvrir plusieurs zones de disponibilité et sous-réseaux AWS VPC pour garantir une haute disponibilité.

NOUVEAU support du modèle de lancement pour les groupes de nœuds gérés

> **Note**
>
> Le terme « groupes de nœuds non gérés » a été utilisé pour désigner les groupes de nœuds pris en charge par eksctl depuis le début (représentés par le champ). `nodeGroups` Le `ClusterConfig` fichier continue d'utiliser le `nodeGroups` champ pour définir les groupes de nœuds non gérés, et les groupes de nœuds gérés sont définis avec le champ. `managedNodeGroups`

Création de groupes de nœuds gérés

```
$ eksctl create nodegroup
```

Nouveaux clusters

Pour créer un nouveau cluster avec un groupe de nœuds géré, exécutez

```
eksctl create cluster
```

Pour créer plusieurs groupes de nœuds gérés et mieux contrôler la configuration, un fichier de configuration peut être utilisé.

> **Note**
>
> Les fonctionnalités des groupes de nœuds gérés ne sont pas totalement équivalentes à celles des groupes de nœuds non gérés.

```
# 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
```

Vous trouverez un autre exemple de fichier de configuration pour créer un groupe de nœuds géré ici.

Il est possible d'avoir un cluster avec des groupes de nœuds gérés et non gérés. Les groupes de nœuds non gérés n'apparaissent pas dans la console AWS EKS mais `eksctl get nodegroup` répertorient les deux types de groupes de nœuds.

```
# 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
```

NOUVEAU Support pour les AMI personnalisées, les groupes de sécurité `instancePrefix`instanceName,`ebsOptimized`,`volumeType`,`volumeName`,`volumeEncrypted`,`volum` et `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'
```

Si vous demandez un type d'instance uniquement disponible dans une zone (et que la configuration eksctl nécessite la spécification de deux), assurez-vous d'ajouter la zone de disponibilité à votre demande de groupe de nœuds :

```
# 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
```

Cela peut être vrai pour des types tels que la famille Hpc6 qui ne sont disponibles que dans une seule zone.

Clusters existants

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

Conseil : si vous utilisez un `ClusterConfig` fichier pour décrire l'ensemble de votre cluster, décrivez votre nouveau groupe de nœuds gérés `managedNodeGroups` sur le terrain et exécutez :

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

Mise à niveau des groupes de nœuds gérés

Vous pouvez mettre à jour un groupe de nœuds vers la dernière version d'AMI optimisée pour EKS pour le type d'AMI que vous utilisez à tout moment.

Si votre groupe de nœuds possède la même version de Kubernetes que le cluster, vous pouvez effectuer la mise à jour vers la dernière version de l'AMI correspondant à cette version de Kubernetes

du type d'AMI que vous utilisez. Si votre groupe de nœuds est la version précédente de Kubernetes par rapport à la version Kubernetes du cluster, vous pouvez mettre à jour le groupe de nœuds vers la dernière version de l'AMI qui correspond à la version Kubernetes du groupe de nœuds, ou le mettre à jour vers la dernière version de l'AMI correspondant à la version Kubernetes du cluster. Vous ne pouvez pas restaurer un groupe de nœuds vers une version antérieure de Kubernetes.

Pour mettre à niveau un groupe de nœuds géré vers la dernière version de l'AMI :

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

Le groupe de nœuds peut être mis à niveau vers la dernière version de l'AMI pour une version spécifiée de Kubernetes en utilisant :

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

Pour effectuer une mise à niveau vers une version spécifique de l'AMI plutôt que vers la dernière version, passez `--release-version` :

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

> **Note**
>
> Si les nœuds gérés sont déployés de manière personnalisée AMIs, le flux de travail suivant doit être suivi afin de déployer une nouvelle version de l'AMI personnalisée.

- le déploiement initial du groupe de nœuds doit être effectué à l'aide d'un modèle de lancement. par exemple

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

- créer une nouvelle version de l'AMI personnalisée (à l'aide de la console AWS EKS).

- créer une nouvelle version du modèle de lancement avec le nouvel ID AMI (à l'aide de la console AWS EKS).
- mettez à niveau les nœuds vers la nouvelle version du modèle de lancement. par exemple

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

Gestion des mises à niveau parallèles pour les nœuds

Plusieurs nœuds gérés peuvent être mis à niveau simultanément. Pour configurer les mises à niveau parallèles, définissez le nom `updateConfig` d'un groupe de nœuds lors de la création du groupe de nœuds. Un exemple `updateConfig` peut être trouvé ici.

Pour éviter toute interruption de vos charges de travail due à la mise à niveau simultanée de plusieurs nœuds, vous pouvez limiter le nombre de nœuds susceptibles de devenir indisponibles lors d'une mise à niveau en le spécifiant dans le `maxUnavailable` champ d'un`updateConfig`. Vous pouvez également utiliser`maxUnavailablePercentage`, qui définit le nombre maximum de nœuds indisponibles en pourcentage du nombre total de nœuds.

Notez qu'il `maxUnavailable` ne peut pas être supérieur à`maxSize`. De plus, `maxUnavailable` et `maxUnavailablePercentage` ne peut pas être utilisé simultanément.

Cette fonctionnalité n'est disponible que pour les nœuds gérés.

Mise à jour des groupes de nœuds gérés

`eksctl`permet de mettre à jour la UpdateConfigsection d'un groupe de nœuds géré. Cette section définit deux champs. `MaxUnavailable`et`MaxUnavailablePercentage`. Vos groupes de nœuds ne sont pas affectés lors de la mise à jour, il ne faut donc pas s'attendre à une interruption de service.

La commande `update nodegroup` doit être utilisée avec un fichier de configuration utilisant le `--config-file` drapeau. Le groupe de nœuds doit contenir une `nodeGroup.updateConfig` section. Vous trouverez de plus amples informations ici.

Problèmes de santé liés à Nodegroup

EKS Managed Nodegroups vérifie automatiquement la configuration de votre groupe de nœuds et de vos nœuds pour détecter tout problème de santé et les signale via l'API et la console EKS. Pour consulter les problèmes de santé d'un groupe de nœuds, procédez comme suit :

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

Gestion des étiquettes

EKS Managed Nodegroups permet de joindre des étiquettes qui sont appliquées aux nœuds Kubernetes du groupe de nœuds. Ceci est spécifié via le `labels` champ dans eksctl lors de la création du cluster ou du groupe de nœuds.

Pour définir de nouvelles étiquettes ou mettre à jour des étiquettes existantes sur un groupe de nœuds :

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

Pour annuler ou supprimer les libellés d'un groupe de nœuds :

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

Pour afficher toutes les étiquettes définies sur un groupe de nœuds :

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

Dimensionnement des groupes de nœuds gérés

`eksctl scale nodegroup`prend également en charge les groupes de nœuds gérés. La syntaxe pour le dimensionnement d'un groupe de nœuds géré ou non géré est la même.

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

Plus d'informations

- Groupes de nœuds gérés par EKS

Démarrage des nœuds

AmazonLinux2023

AL2023 a introduit un nouveau processus d'initialisation des nœuds nodeadm qui utilise un schéma de configuration YAML, supprimant ainsi l'utilisation de scripts. `/etc/eks/bootstrap.sh`

> **Note**
>
> Avec les versions 1.30 et supérieures de Kubernetes, Amazon Linux 2023 est le système d'exploitation par défaut.

Paramètres par défaut pour AL2

Pour les nœuds autogérés et les nœuds gérés par EKS sur la base de la personnalisation AMIs, `eksctl` crée une valeur par défaut, minimale, `NodeConfig` et l'injecte automatiquement dans les données utilisateur du modèle de lancement du groupe de nœuds.

```
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

--//--
```

Pour les nœuds gérés par EKS basés sur le mode natif AMIs, la valeur par défaut `NodeConfig` est ajoutée par EKS MNG sous le capot, directement ajoutée aux données utilisateur de l'EC2. Ainsi, dans ce scénario, il `eksctl` n'est pas nécessaire de l'inclure dans le modèle de lancement.

Configuration du processus d'amorçage

Pour définir les propriétés avancées de`NodeConfig`, ou simplement remplacer les valeurs par défaut, eksctl vous permet de spécifier une valeur personnalisée `NodeConfig` via ou par ex. `nodeGroup.overrideBootstrapCommand` `managedNodeGroup.overrideBootstrapCommand`

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

Cette configuration personnalisée sera ajoutée aux données utilisateur par eksctl et fusionnée avec la configuration par `nodeadm` défaut. Pour en savoir plus sur `nodeadm` la capacité de fusionner plusieurs objets de configuration, cliquez ici.

Support des modèles de lancement pour les groupes de nœuds gérés

eksctl prend en charge le lancement de groupes de nœuds gérés à l'aide d'un modèle de lancement EC2 fourni. Cela permet de multiples options de personnalisation pour les groupes de nœuds, notamment la fourniture de groupes personnalisés AMIs et de sécurité, et la transmission de données utilisateur pour le démarrage des nœuds.

Création de groupes de nœuds gérés à l'aide d'un modèle de lancement fourni

```
# 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
```

Mise à niveau d'un groupe de nœuds géré pour utiliser une version de modèle de lancement différente

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

> **Note**
>
> Si un modèle de lancement utilise une AMI personnalisée, la nouvelle version doit également utiliser une AMI personnalisée, sinon l'opération de mise à niveau échouera

Si un modèle de lancement n'utilise pas d'AMI personnalisée, la version de Kubernetes vers laquelle effectuer la mise à niveau peut également être spécifiée :

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

Remarques sur la prise en charge des AMI personnalisées et des modèles de lancement

- Lorsqu'un modèle de lancement est fourni, les champs suivants ne sont pas pris en charge : `instanceType` `ami` `ssh.allowssh.sourceSecurityGroupIds`,`securityGroups`,`instancePrefix`,`instanceName`,`eb` `overrideBootstrapCommand` et`disableIMDSv1`.
- Lorsque vous utilisez une AMI personnalisée (`ami`), elle `overrideBootstrapCommand` doit également être configurée pour effectuer le démarrage.
- `overrideBootstrapCommand`ne peut être défini que lors de l'utilisation d'une AMI personnalisée.
- Lorsqu'un modèle de lancement est fourni, les balises spécifiées dans la configuration du groupe de nœuds s'appliquent uniquement à la ressource EKS Nodegroup et ne sont pas propagées aux instances EC2.

Sous-réseaux personnalisés

Il est possible d'étendre un VPC existant avec un nouveau sous-réseau et d'ajouter un groupe de nœuds à ce sous-réseau.

Pourquoi

Si le cluster n'a plus de paramètres préconfigurés IPs, il est possible de redimensionner le VPC existant avec un nouveau CIDR pour y ajouter un nouveau sous-réseau. Pour savoir comment procéder, lisez ce guide sur AWS Extended VPCs.

TL ; DR

Accédez à la configuration du VPC et ajoutez-la, cliquez sur Actions->Modifier CIDRs et ajoutez une nouvelle plage. Par exemple :

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

Vous devez maintenant ajouter un nouveau sous-réseau. Selon qu'il s'agit d'un nouveau sous-réseau privé ou public, vous devrez copier les informations de routage depuis un sous-réseau privé ou public respectivement.

Une fois le sous-réseau créé, ajoutez le routage et copiez l'ID de passerelle NAT ou l'Internet Gateway depuis un autre sous-réseau du VPC. S'il s'agit d'un sous-réseau public, veillez à activer l'attribution automatique des adresses IP. Actions->Modifier les paramètres d'attribution IP automatique->Activer l'attribution automatique des adresses publiques. IPv4

N'oubliez pas de copier également les TAGS des sous-réseaux existants en fonction de la configuration des sous-réseaux publics ou privés. Ceci est important, sinon le sous-réseau ne fera pas partie du cluster et les instances du sous-réseau ne pourront pas se joindre.

Lorsque vous avez terminé, copiez l'ID du nouveau sous-réseau. Répéter aussi souvent que nécessaire.

Comment ?

Pour créer un groupe de nœuds dans le ou les sous-réseaux créés, exécutez la commande suivante :

```
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
```

Ou utilisez la configuration telle quelle :

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

Avec une configuration comme celle-ci :

```
# 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
```

Attendez que le groupe de nœuds soit créé et que les nouvelles instances devraient avoir les nouvelles plages d'adresses IP du ou des sous-réseaux.

Supprimer le cluster

Étant donné que le nouvel ajout a modifié le VPC existant en ajoutant une dépendance en dehors de la CloudFormation pile, il n'est plus CloudFormation possible de supprimer le cluster.

Avant de supprimer le cluster, supprimez manuellement tous les sous-réseaux supplémentaires créés, puis appelez `eksctl` :

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

DNS personnalisé

Il existe deux manières de remplacer l'adresse IP du serveur DNS utilisée pour toutes les recherches DNS internes et externes. C'est l'équivalent du `--cluster-dns` drapeau du`kubelet`.

La première, c'est sur le `clusterDNS` terrain. Les fichiers de configuration acceptent un `string` champ appelé `clusterDNS` avec l'adresse IP du serveur DNS à utiliser. Cela sera transmis au

`kubelet` qui, à son tour, le transmettra aux pods via le `/etc/resolv.conf` fichier. Pour plus d'informations, consultez le schéma du fichier de configuration.

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

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

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

Notez que cette configuration n'accepte qu'une seule adresse IP. Pour spécifier plusieurs adresses, utilisez le `kubeletExtraConfig`paramètre :

```
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"]
```

Souillures

Pour appliquer des taches à un groupe de nœuds spécifique, utilisez la section de `taints` configuration comme suit :

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

```
        effect: NoExecute
```

Un exemple complet peut être trouvé ici.

Sélecteur d'instance

eksctl permet de spécifier plusieurs types d'instances pour les groupes de nœuds gérés et autogérés, mais avec plus de 270 types d'instances EC2, les utilisateurs doivent passer du temps à déterminer quels types d'instances conviennent le mieux à leur groupe de nœuds. C'est encore plus difficile lorsque vous utilisez des instances Spot, car vous devez choisir un ensemble d'instances qui fonctionne bien avec le Cluster Autoscaler.

eksctl s'intègre désormais au sélecteur d'instance EC2, qui résout ce problème en générant une liste de types d'instances en fonction de critères de ressources : vCPUs, mémoire, nombre de et architecture du GPUs processeur. Lorsque les critères de sélection d'instance sont passés, eksctl crée un groupe de nœuds avec les types d'instances définis sur les types d'instance correspondant aux critères fournis.

Création de clusters et de groupes de nœuds

Pour créer un cluster avec un seul groupe de nœuds qui utilise des types d'instances correspondant aux critères de ressources du sélecteur d'instance transmis à eksctl, exécutez

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

Cela créera un cluster et un groupe de nœuds géré avec le `instanceTypes` champ défini sur `[c5.large, c5a.large, c5ad.large, c5d.large, t2.medium, t3.medium, t3a.medium]` (l'ensemble de types d'instances renvoyé peut changer).

Pour les groupes de nœuds non gérés, le `instancesDistribution.instanceTypes` champ sera défini comme suit :

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

Les critères du sélecteur d'instance peuvent également être spécifiés dans ClusterConfig :

```
# 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
```

Les options CLI du sélecteur d'instance suivantes sont prises en charge par `eksctl create cluster` et `eksctl create nodegroup` :

`--instance-selector-vcpus`, `--instance-selector-memory`, `--instance-selector-gpus` et `instance-selector-cpu-architecture`

Vous trouverez un exemple de fichier ici.

Run à sec

La fonction d'exécution à sec vous permet d'inspecter et de modifier les instances correspondant au sélecteur d'instance avant de procéder à la création d'un groupe de nœuds.

```
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
```

Le résultat généré ClusterConfig peut ensuite être transmis à `eksctl create cluster` :

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

Le `instanceSelector` champ représentant les options de la CLI sera également ajouté au ClusterConfig fichier à des fins de visibilité et de documentation. Lorsqu'il `--dry-run` est omis, ce champ sera ignoré et le `instanceTypes` champ sera utilisé, sinon toute modification `instanceTypes` sera annulée par eksctl.

Lorsqu'un ClusterConfig fichier est transmis`--dry-run`, eksctl produit un ClusterConfig fichier contenant le même ensemble de groupes de nœuds après avoir développé les critères de ressources du sélecteur d'instance de chaque groupe de nœuds.

```
# 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
# ...
```

Instances Spot

Groupes de nœuds gérés

`eksctl`prend en charge les nœuds de travail ponctuels à l'aide de groupes de nœuds gérés par EKS, une fonctionnalité qui permet aux clients d'EKS dotés d'applications tolérantes aux pannes de configurer et de gérer facilement des instances ponctuelles EC2 pour leurs clusters EKS. EKS Managed Nodegroup configurera et lancera un groupe d'instances Spot EC2 Autoscaling en suivant les meilleures pratiques Spot et en vidant automatiquement les nœuds de travail Spot avant que les instances ne soient interrompues par AWS. L'utilisation de cette fonctionnalité est gratuite et les clients ne paient que pour l'utilisation des ressources AWS, telles que les instances Spot EC2 et les volumes EBS.

Pour créer un cluster avec un groupe de nœuds géré à l'aide d'instances Spot, transmettez l'`--spot`indicateur et une liste facultative de types d'instances :

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

Pour créer un groupe de nœuds géré à l'aide d'instances Spot sur un cluster existant :

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

Pour créer des instances Spot à l'aide de groupes de nœuds gérés via un fichier de configuration :

```
# 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**
>
> Les groupes de nœuds non gérés ne prennent pas en charge les `instanceTypes` champs `spot` et, à la place, le `instancesDistribution` champ est utilisé pour configurer les instances Spot. Voir ci-dessous

Plus d'informations

- Groupes de nœuds EKS Spot
- Types de capacité des groupes de nœuds gérés par EKS

Groupes de nœuds non gérés

`eksctl`prend en charge les instances ponctuelles via le MixedInstancesPolicy for Auto Scaling Groups.

Voici un exemple de groupe de nœuds qui utilise 50 % d'instances ponctuelles et 50 % d'instances à la demande :

```
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
```

Notez que le `nodeGroups.X.instanceType` champ ne doit pas être défini lors de l'utilisation du `instancesDistribution` champ.

Cet exemple utilise des instances de GPU :

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

Cet exemple utilise la stratégie d'allocation de places optimisée en termes de capacité :

```
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"
```

Cet exemple utilise la stratégie d'allocation au capacity-optimized-prioritized comptant :

```
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"
```

Utilisez la stratégie `capacity-optimized-prioritized` d'allocation, puis définissez l'ordre des types d'instances dans la liste des remplacements de modèles de lancement, de la priorité la plus élevée à la plus faible (du premier au dernier de la liste). Amazon EC2 Auto Scaling respecte les priorités relatives aux types d'instances dans la mesure du possible, mais optimise d'abord la capacité. Il s'agit d'une bonne option pour les charges de travail où les risques d'interruption doivent être minimisés, mais où la préférence pour certains types d'instances est également importante. Pour plus d'informations, consultez la section Options d'achat ASG.

Notez que le `spotInstancePools` champ ne doit pas être défini lors de l'utilisation du `spotAllocationStrategy` champ. Si ce n'`spotAllocationStrategy`est pas spécifié, EC2 utilisera par défaut la `lowest-price` stratégie.

Voici un exemple minimal :

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

Pour distinguer les nœuds entre les instances ponctuelles ou à la demande, vous pouvez utiliser l'étiquette kubernetes `node-lifecycle` qui aura la valeur `spot` ou `on-demand` dépendra de son type.

Paramètres dans InstancesDistribution

Consultez le schéma de configuration du cluster pour plus de détails.

Support du GPU

Eksctl prend en charge la sélection des types d'instances GPU pour les groupes de nœuds. Fournissez simplement un type d'instance compatible à la commande de création ou via le fichier de configuration.

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

> **Note**
>
> Il n'est plus nécessaire de s'abonner à l'AMI Marketplace pour bénéficier du support GPU sur EKS.

Les résolveurs d'AMI (`autoetauto-ssm`) verront que vous souhaitez utiliser un type d'instance GPU et sélectionneront l'AMI accélérée optimisée pour EKS appropriée.

Eksctl détectera qu'une AMI avec un type d'instance compatible GPU a été sélectionnée et installera automatiquement le plug-in de périphérique NVIDIA Kubernetes.

> **Note**
>
> Windows et Ubuntu AMIs ne sont pas livrés avec les pilotes GPU installés. Par conséquent, l'exécution de charges de travail accélérées par GPU ne fonctionnera pas immédiatement.

Pour désactiver l'installation automatique du plugin et installer manuellement une version spécifique, utilisez `--install-nvidia-plugin=false` la commande create. Par exemple :

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

et, pour les versions 0.15.0 et supérieures,

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

ou, pour les anciennes versions,

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

L'installation du plug-in d'appareil NVIDIA Kubernetes sera ignorée si le cluster inclut uniquement des groupes de nœuds Bottlerocket, puisque Bottlerocket gère déjà l'exécution du plug-in d'appareil. Si vous utilisez différentes familles d'AMI dans les configurations de votre cluster, vous devrez peut-être utiliser des restrictions et des tolérances pour empêcher le plug-in de l'appareil de s'exécuter sur les nœuds Bottlerocket.

Support ARM

Cette rubrique explique comment créer un cluster avec un groupe de nœuds ARM et comment ajouter un groupe de nœuds ARM à un cluster existant.

EKS prend en charge l'architecture ARM 64 bits avec ses processeurs Graviton.
Pour créer un cluster, sélectionnez l'un des types d'instances basés sur `a1` Graviton (`t4gm6g`,`m7g`,`m6gd`,`c6g`,`c7g`,`c6gd`,`r6g`,`r7g`, `r6gd` `m8gr8g`,`c8g`) et exécutez :

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

ou utilisez un fichier de configuration :

```
---
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 est également pris en charge dans les groupes de nœuds gérés :

```
---
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
```

L'AMI résout `auto` et déduira `auto-ssm` l'AMI correcte en fonction du type d'instance ARM. Seules les familles AmazonLinux 2023, AmazonLinux 2 et Bottlerocket disposent d'EKS optimisé pour AMIs ARM.

> **Note**
>
> ARM est pris en charge pour les clusters dotés de la version 1.15 et supérieure.

Auto Scaling

Activer Auto Scaling

Vous pouvez créer un cluster (ou un groupe de nœuds dans un cluster existant) avec le rôle IAM qui permettra d'utiliser l'autoscaler du cluster :

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

Cet indicateur définit `k8s.io/cluster-autoscaler/enabled` et `k8s.io/cluster-autoscaler/<clusterName>` balise également, de sorte que la découverte des groupes de nœuds devrait fonctionner.

Une fois le cluster lancé, vous devez installer Cluster Autoscaler lui-même.

Vous devez également ajouter les éléments suivants à vos définitions de groupes de nœuds gérés ou non gérés pour ajouter les balises requises pour que le Cluster Autoscaler puisse redimensionner le groupe de nœuds :

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

Mise à l'échelle à partir de 0

Si vous souhaitez pouvoir redimensionner votre groupe de nœuds à partir de 0 et que vous avez défini des and/or taches d'étiquettes sur vos groupes de nœuds, vous devrez les propager sous forme de balises sur vos Auto Scaling Groups (). ASGs

Pour ce faire, vous pouvez définir les balises ASG dans le `tags` champ de définition de vos groupes de nœuds. Par exemple, étant donné un groupe de nœuds avec les étiquettes et les nuances suivantes :

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

Vous devez ajouter les balises ASG suivantes :

```
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"
```

Pour les groupes de nœuds gérés et non gérés, cela peut être fait automatiquement en `propagateASGTags` réglant sur`true`, ce qui ajoutera les étiquettes et les nuances sous forme de balises au groupe Auto Scaling :

```
nodeGroups:
  - name: ng1-public
    ...
    labels:
```

```
      my-cool-label: pizza
    taints:
      feaster: "true:NoSchedule"
    propagateASGTags: true
```

Auto Scaling adapté aux zones

Si vos charges de travail sont spécifiques à une zone, vous devez créer des groupes de nœuds distincts pour chaque zone. Cela est dû au fait que l'on `cluster-autoscaler` suppose que tous les nœuds d'un groupe sont exactement équivalents. Ainsi, par exemple, si un événement de mise à l'échelle est déclenché par un pod qui a besoin d'un PVC spécifique à une zone (par exemple un volume EBS), le nouveau nœud peut être planifié dans la mauvaise AZ et le pod ne démarrera pas.

Vous n'aurez pas besoin d'un groupe de nœuds distinct pour chaque AZ si votre environnement répond aux critères suivants :

- Aucune exigence de stockage spécifique à une zone.
- Aucune PodAffinity requise avec une topologie autre que l'hôte.
- Aucune affinité de nœud requise sur l'étiquette de zone.
- Aucun NodeSelector sur une étiquette de zone.

(Pour en savoir plus, cliquez ici et ici.)

Si vous répondez à toutes les exigences ci-dessus (et éventuellement à d'autres), vous devriez être en sécurité avec un seul groupe de nœuds qui s'étend sur plusieurs. AZs Sinon, vous souhaiterez créer des groupes de nœuds mono-AZ distincts :

AVANT :

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

APRÈS :

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

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

Support personnalisé pour les AMI

Configuration de l'ID AMI du nœud

L'`--node-ami`indicateur permet un certain nombre de cas d'utilisation avancés, tels que l'utilisation d'une AMI personnalisée ou l'interrogation d'AWS en temps réel pour déterminer l'AMI à utiliser. Le drapeau peut être utilisé à la fois pour les images sans GPU et pour les images avec GPU.

L'indicateur peut prendre l'identifiant de l'image AMI d'une image à utiliser explicitement. Il peut également utiliser les mots clés « spéciaux » suivants :

Mot clé	Description
auto	Indique que l'AMI à utiliser pour les nœuds doit être trouvée en interrogeant AWS EC2. Cela concerne le résolveur automatique.
auto-ssm	Indique que l'AMI à utiliser pour les nœuds doit être trouvée en interrogeant le magasin de paramètres AWS SSM.

> **Note**
>
> Pour le moment, les groupes de nœuds gérés par EKS ne prennent en charge que les familles d'AMI suivantes lorsque vous travaillez avec des fonctionnalités personnalisées AMIs :`AmazonLinux2023`,`AmazonLinux2`,`Bottlerocket`,`Ubuntu2004`, et `UbuntuPro2004` `Ubuntu2204` `Ubuntu2404`

Lorsque vous définissez une chaîne `--node-ami` d'identification, cela `eksctl` supposera qu'une AMI personnalisée a été demandée. Pour les nœuds AmazonLinux 2 et Ubuntu, gérés ou autogérés par EKS, cela signifie que cela `overrideBootstrapCommand` est nécessaire. Pour AmazonLinux 2023, puisqu'il cesse d'utiliser le `/etc/eks/bootstrap.sh` script pour le

démarrage des nœuds, au profit d'un processus d'initialisation de nodeadm (pour plus d'informations, reportez-vous à la documentation sur le démarrage des nœuds), n'est pas pris en charge. `overrideBootstrapCommand`

Exemples d'indicateurs CLI :

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

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

Exemple de fichier de configuration :

```
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>
```

Le `--node-ami` drapeau peut également être utilisé avec`eksctl create nodegroup`.

Configuration de la famille AMI du nœud

Ils `--node-ami-family` peuvent utiliser les mots clés suivants :

Mot clé	Description
AmazonLinux2	Indique que l'image AMI EKS basée sur Amazon Linux 2 doit être utilisée (par défaut).

Mot clé	Description
AmazonLinux2023	Indique que l'image AMI EKS basée sur Amazon Linux 2023 doit être utilisée.
Ubuntu 2004	Indique que l'image AMI EKS basée sur Ubuntu 20.04 LTS (Focal) doit être utilisée (prise en charge pour EKS 1.29).
UbuntuPro2004	Indique que l'image de l'AMI EKS basée sur Ubuntu Pro 20.04 LTS (Focal) doit être utilisée (disponible pour EKS >= 1.27 et 1.29).
Ubuntu 2204	Indique que l'image AMI EKS basée sur Ubuntu 22.04 LTS (Jammy) doit être utilisée (disponible pour EKS >= 1.29).
UbuntuPro2204	Indique que l'image AMI EKS basée sur Ubuntu Pro 22.04 LTS (Jammy) doit être utilisée (disponible pour EKS >= 1.29).
Ubuntu 2404	Indique que l'image AMI EKS basée sur Ubuntu 24.04 LTS (Noble) doit être utilisée (disponible pour EKS >= 1.31).
UbuntuPro2404	Indique que l'image AMI EKS basée sur Ubuntu Pro 24.04 LTS (Noble) doit être utilisée (disponible pour EKS >= 1.31).
Flacon Rocket	Indique que l'image AMI EKS basée sur Bottlerocket doit être utilisée.
WindowsServer2019 FullContainer	Indique que l'image AMI EKS basée sur le conteneur complet de Windows Server 2019 doit être utilisée.
WindowsServer2019 CoreContainer	Indique que l'image AMI EKS basée sur Windows Server 2019 Core Container doit être utilisée.

Mot clé	Description
WindowsServer2022 FullContainer	Indique que l'image AMI EKS basée sur le conteneur complet Windows Server 2022 doit être utilisée.
WindowsServer2022 CoreContainer	Indique que l'image AMI EKS basée sur Windows Server 2022 Core Container doit être utilisée.

Exemple d'indicateur CLI :

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

Exemple de fichier de configuration :

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

Le `--node-ami-family` drapeau peut également être utilisé avec`eksctl create nodegroup`. `eksctl`nécessite que la famille d'AMI soit explicitement définie via un fichier de configuration ou via un indicateur `--node-ami-family` CLI, chaque fois que vous travaillez avec une AMI personnalisée.

> **Note**
>
> Pour le moment, les groupes de nœuds gérés par EKS ne prennent en charge que les familles d'AMI suivantes lorsque vous travaillez avec des fonctionnalités personnalisées AMIs :`AmazonLinux2023`,`AmazonLinux2`,`Bottlerocket`,`Ubuntu2004`, et `UbuntuPro2004` `Ubuntu2204` `Ubuntu2404`

Prise en charge des AMI personnalisées pour Windows

Seuls les groupes de nœuds Windows autogérés peuvent spécifier une AMI personnalisée. `amiFamily`doit être défini sur une famille d'AMI Windows valide.

Les PowerShell variables suivantes seront disponibles pour le script bootstrap :

```
$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
```

Exemple de fichier de configuration :

```
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
```

Support d'AMI personnalisé Bottlerocket

Pour les nœuds Bottlerocket, le `overrideBootstrapCommand` n'est pas pris en charge. Au lieu de cela, pour désigner leur propre conteneur bootstrap, il faut utiliser le `bottlerocket` champ dans le cadre du fichier de configuration. P. ex.

```
  nodeGroups:
  - name: bottlerocket-ng
    ami: ami-custom1234
    amiFamily: Bottlerocket
    bottlerocket:
      enableAdminContainer: true
```

```
      settings:
        bootstrap-containers:
          bootstrap:
            source: <MY-CONTAINER-URI>
```

Nœuds Windows Worker

À partir de la version 1.14, Amazon EKS prend en charge les nœuds Windows qui permettent d'exécuter des conteneurs Windows. Outre les nœuds Windows, un nœud Linux du cluster est nécessaire pour exécuter CoreDNS, car Microsoft ne prend pas encore en charge le mode réseau hôte. Ainsi, un cluster Windows EKS sera un mélange de nœuds Windows et d'au moins un nœud Linux. Les nœuds Linux sont essentiels au fonctionnement du cluster. Par conséquent, pour un cluster de production, il est recommandé de disposer d'au moins deux nœuds `t2.large` Linux pour HA.

> **Note**
>
> Il n'est plus nécessaire d'installer le contrôleur de ressources VPC sur les nœuds de travail Linux pour exécuter les charges de travail Windows dans les clusters EKS créés après le 22 octobre 2021. Vous pouvez activer la gestion des adresses IP Windows sur le plan de contrôle EKS via un paramètre ConfigMap (voir le lien : eks/latest/userguide/windows -support.html pour plus de détails). eksctl patchera automatiquement le ConfigMap pour activer la gestion des adresses IP Windows lorsqu'un groupe de nœuds Windows est créé.

Création d'un nouveau cluster compatible avec Windows

La syntaxe du fichier de configuration permet de créer un cluster entièrement fonctionnel compatible avec Windows en une seule commande :

```
# 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
```

Pour créer un nouveau cluster avec un groupe de nœuds non géré sous Windows sans utiliser de fichier de configuration, exécutez les commandes suivantes :

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

Ajout du support Windows à un cluster Linux existant

Pour permettre l'exécution de charges de travail Windows sur un cluster existant avec des nœuds Linux (famille `AmazonLinux2` AMI), vous devez ajouter un groupe de nœuds Windows.

NOUVEAU Support pour les groupes de nœuds gérés par Windows a été ajouté (--managed=true ou omettre le drapeau).

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

Pour garantir que les charges de travail sont planifiées sur le bon système d'exploitation, elles doivent `nodeSelector` cibler le système d'exploitation sur lequel elles doivent s'exécuter :

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

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

Si vous utilisez un cluster plus ancien que `1.19` le `kubernetes.io/os` et, `kubernetes.io/arch` les étiquettes doivent être remplacées par `beta.kubernetes.io/os` et `beta.kubernetes.io/arch` respectivement.

Plus d'informations

- Support EKS pour Windows

Mappages de volumes supplémentaires

En tant qu'option de configuration supplémentaire, lorsqu'il s'agit de mappages de volumes, il est possible de configurer des mappages supplémentaires lors de la création du groupe de nœuds.

Pour ce faire, définissez le champ `additionalVolumes` comme suit :

```
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'
```

Pour plus de détails sur la sélection de VolumeNames, consultez la documentation relative à la dénomination des appareils. Pour en savoir plus sur les volumes EBS, les limites de volume des instances ou les mappages de périphériques par blocs, rendez-vous sur cette page.

Nœuds hybrides EKS

Introduction

AWS EKS introduit les nœuds hybrides, une nouvelle fonctionnalité qui vous permet d'exécuter des applications sur site et de périphérie sur une infrastructure gérée par le client avec les mêmes clusters, fonctionnalités et outils AWS EKS que ceux que vous utilisez dans le cloud AWS. AWS EKS Hybird Nodes apporte une expérience Kubernetes gérée par AWS aux environnements sur site afin que les clients simplifient et normalisent la façon dont vous exécutez les applications dans les environnements sur site, de périphérie et cloud. En savoir plus sur EKS Hybrid Nodes.

Pour faciliter le support de cette fonctionnalité, eksctl introduit un nouveau champ de premier niveau appelé. `remoteNetworkConfig` Toute configuration liée aux nœuds hybrides doit être configurée via ce champ, dans le cadre du fichier de configuration ; il n'existe aucun équivalent d'indicateur CLI. De plus, au lancement, toute configuration de réseau distant ne peut être configurée que lors de la création du cluster et ne peut pas être mise à jour par la suite. Cela signifie que vous ne pourrez pas mettre à jour les clusters existants pour utiliser des nœuds hybrides.

La `remoteNetworkConfig` section du fichier de configuration vous permet de configurer les deux domaines principaux lorsqu'il s'agit de joindre des nœuds distants à vos clusters EKS : le réseau et les informations d'identification.

Réseaux

EKS Hybrid Nodes s'adapte à votre méthode préférée pour connecter votre ou vos réseaux sur site à un VPC AWS. Plusieurs options documentées sont disponibles, notamment AWS Site-to-Site VPN et AWS Direct Connect, et vous pouvez choisir la méthode la mieux adaptée à votre cas d'utilisation. Dans la plupart des méthodes que vous pouvez choisir, votre VPC sera attaché à une passerelle privée virtuelle (VGW) ou à une passerelle de transit (TGW). Si vous comptez sur eksctl pour créer un VPC pour vous, eksctl configurera également, dans le cadre de votre VPC, tous les prérequis liés au réseau afin de faciliter la communication entre votre plan de contrôle EKS et les nœuds distants, c'est-à-dire

- règles SG d'entrée/de sortie
- routes dans les tables de routage des sous-réseaux privés
- l'attachement de la passerelle VPC au TGW ou VGW donné

Exemple de fichier de configuration :

```
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"]
```

Si la méthode de connectivité que vous avez choisie n'implique pas l'utilisation d'un TGW ou d'un VGW, vous ne devez pas vous fier à eksctl pour créer le VPC à votre place, mais en fournir un préexistant. Dans le même ordre d'idées, si vous utilisez un VPC préexistant, eksctl n'y apportera aucune modification, et il est de votre responsabilité de vous assurer que toutes les exigences réseau sont en place.

> **Note**
>
> eksctl ne configure aucune infrastructure réseau en dehors de votre VPC AWS (c'est-à-dire aucune infrastructure depuis VGW/TGW les réseaux distants)

Informations d’identification

Les nœuds hybrides EKS utilisent l'authenticator AWS IAM et les informations d'identification IAM temporaires fournies par AWS SSM ou AWS IAM Roles Anywhere pour s'authentifier auprès du cluster EKS. Semblable aux groupes de nœuds autogérés, eksctl créera pour vous un rôle IAM de nœuds hybrides à assumer par les nœuds distants, sauf indication contraire. De plus, lorsque vous utilisez IAM Roles Anywhere comme fournisseur d'informations d'identification, eksctl configurera un profil et une ancre de confiance en fonction d'un ensemble d'autorités de certification donné (), par exemple `iam.caBundleCert`

```
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
```

L'ARN du rôle de nœuds hybrides créé par eksctl est nécessaire plus tard dans le processus de connexion de vos nœuds distants au cluster, pour la configuration et `NodeConfig` pour `nodeadm` la création d'activations (si vous utilisez SSM). Pour le récupérer, utilisez :

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

De même, si vous utilisez IAM Roles Anywhere, vous pouvez récupérer l'ARN de l'ancre de confiance et du profil anywhere créés par eksctl, en modifiant la commande précédente en la `RemoteNodesRoleARN` remplaçant `RemoteNodesTrustAnchorARN` par ou, respectivement. `RemoteNodesAnywhereProfileARN`

Si vous disposez d'une configuration IAM Roles Anywhere préexistante ou si vous utilisez SSM, vous pouvez fournir un rôle IAM pour les nœuds hybrides via. `remoteNetworkConfig.iam.roleARN` N'oubliez pas que dans ce scénario, eksctl ne créera pas pour vous l'ancre de confiance ni le profil anywhere. Par exemple

```
remoteNetworkConfig:
```

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

Pour associer le rôle à une identité Kubernetes et autoriser les nœuds distants à rejoindre le cluster EKS, eksctl crée une entrée d'accès avec le rôle IAM des nœuds hybrides comme ARN principal et de type. `HYBRID_LINUX`

```
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"
        ]
    }
]
```

Support des modules complémentaires

Interface réseau de conteneurs (CNI) : l'AWS VPC CNI ne peut pas être utilisée avec des nœuds hybrides. Les fonctionnalités de base de Cilium et Calico sont prises en charge pour une utilisation avec des nœuds hybrides. Vous pouvez gérer votre CNI avec l'outillage de votre choix, tel que Helm. Pour plus d'informations, voir Configurer un CNI pour les nœuds hybrides.

> **Note**
>
> Si vous installez un VPC CNI dans votre cluster pour vos groupes de nœuds autogérés ou gérés par EKS, vous devez utiliser `v1.19.0-eksbuild.1` ou une version ultérieure, car cela inclut une mise à jour du daemonset du module complémentaire afin d'empêcher son installation sur les nœuds hybrides.

Autres références

- Nœuds hybrides EKS UserDocs
- Annonce de lancement

Configuration de réparation des nœuds de support pour les groupes de nœuds gérés par EKS

EKS Managed Nodegroups prend en charge la réparation des nœuds, où l'état des nœuds gérés est surveillé et où les nœuds de travail défectueux sont remplacés ou redémarrés en réponse. eksctl propose désormais des options de configuration complètes pour un contrôle précis du comportement de réparation des nœuds.

Configuration de base de la réparation des nœuds

Utilisation des drapeaux de la CLI

Pour créer un cluster avec un groupe de nœuds géré à l'aide de la réparation de base des nœuds, passez l'`--enable-node-repair`indicateur suivant :

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

Pour créer un groupe de nœuds géré avec réparation de nœuds sur un cluster existant :

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

Utilisation des fichiers de configuration

```
# 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
```

Configuration améliorée de réparation des nœuds

Configuration du seuil

Vous pouvez configurer le moment où les actions de réparation des nœuds cesseront en utilisant des seuils basés sur le pourcentage ou le nombre. Remarque : Vous ne pouvez pas utiliser à la fois les seuils de pourcentage et de nombre.

Indicateurs CLI pour les seuils

```
# 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
```

Fichier de configuration pour les seuils

```
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
```

Limites de réparation en parallèle

Contrôlez le nombre maximum de nœuds pouvant être réparés simultanément ou en parallèle. Cela vous offre un contrôle plus précis sur le rythme des remplacements de nœuds. Remarque : Vous ne pouvez pas utiliser à la fois des limites de pourcentage et de nombre.

Indicateurs CLI pour les limites de parallélisme

```
# 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
```

Fichier de configuration pour les limites de parallélisme

```
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
```

Dérogations de réparation personnalisées

Spécifiez des remplacements granulaires pour des actions de réparation spécifiques. Ces remplacements contrôlent l'action de réparation et le délai de réparation avant qu'un nœud ne soit considéré comme éligible à la réparation. Si vous l'utilisez, vous devez spécifier toutes les valeurs pour chaque remplacement.

```
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"
```

Exemples de configuration complets

Pour un exemple complet avec toutes les options de configuration, consultez examples/44-node-repair.yaml.

Exemple 1 : réparation de base avec seuils de pourcentage

```
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
```

Exemple 2 : Réparation prudente pour les charges de travail critiques

```
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"
```

Exemple 3 : charge de travail du GPU avec réparation spécialisée

```
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 Reference

Drapeaux de réparation de nœuds

Indicateur	Description	Exemple
`--enable-node-repair`	Activer la réparation automatique des nœuds	`--enable-node-repair`

Indicateur	Description	Exemple
`--node-repair-max-unhealthy-percentage`	Pourcentage maximal de nœuds défectueux avant réparation	`--node-repair-max-unhealthy-percentage=20`
`--node-repair-max-unhealthy-count`	Nombre maximal de nœuds défectueux avant réparation	`--node-repair-max-unhealthy-count=5`
`--node-repair-max-parallel-percentage`	Pourcentage maximum de nœuds à réparer en parallèle	`--node-repair-max-parallel-percentage=15`
`--node-repair-max-parallel-count`	Nombre maximum de nœuds à réparer en parallèle	`--node-repair-max-parallel-count=2`

Remarque : les remplacements de configuration de réparation de nœuds ne sont pris en charge que par le biais des fichiers de configuration YAML en raison de leur complexité.

Référence de configuration

nodeRepairConfig

Champ	Type	Description	Constaintes	Exemple
`enabled`	boolean	Activer/désactiver la réparation des nœuds	-	`true`
`maxUnhealthyNodeThresholdPercentage`	entier	Pourcentage de nœuds défectueux, au-dessus duquel les actions de réparation automatique des nœuds s'arrêteront	Ne peut pas être utilisé avec `maxUnhealthyNodeThresholdCount`	`20`

Champ	Type	Description	Constaintes	Exemple
`maxUnhealthyNodeThresholdCount`	entier	Seuil de comptage des nœuds défectueux, au-dessus duquel les actions de réparation automatique des nœuds s'arrêteront	Ne peut pas être utilisé avec `maxUnhealthyNodeThresholdPercentage`	5
`maxParallelNodesRepairedPercentage`	entier	Pourcentage maximal de nœuds défectueux pouvant être réparés simultanément ou en parallèle	Ne peut pas être utilisé avec `maxParallelNodesRepairedCount`	15
`maxParallelNodesRepairedCount`	entier	Nombre maximal de nœuds défectueux pouvant être réparés simultanément ou en parallèle	Ne peut pas être utilisé avec `maxParallelNodesRepairedPercentage`	2
`nodeRepairConfigOverrides`	array	Dérogations granulaires pour des actions de réparation spécifiques contrôlant l'action de réparation et le délai	Toutes les valeurs doivent être spécifiées pour chaque remplacement	Voir les exemples ci-dessus

nodeRepairConfigDérogations

Champ	Type	Description	Valeurs valides
`nodeMonitoringCondition`	chaîne	État défectueux signalé par l'agent de surveillance des nœuds auquel cette dérogation s'applique	`"AcceleratedInstanceNotReady"` , `"NetworkNotReady"`
`nodeUnhealthyReason`	chaîne	Raison signalée par l'agent de surveillance des nœuds pour laquelle cette dérogation s'applique	`"NvidiaXID13Error"` ,`"InterfaceNotUp"`
`minRepairWaitTimeMins`	entier	Temps d'attente minimal en minutes avant de tenter de réparer un nœud avec la condition et le motif spécifiés	Tout entier positif
`repairAction`	chaîne	Action de réparation à effectuer pour les nœuds lorsque toutes les conditions spécifiées sont remplies	`"Terminate"` , `"Restart"` , `"NoAction"`

Plus d'informations

- Santé des nœuds du groupe de nœuds géré par EKS

Réseaux

Ce chapitre contient des informations sur la façon dont Eksctl crée des réseaux Virtual Private Cloud (VPC) pour les clusters EKS.

Rubriques :

- the section called “VPC Configuration”
  - Modifier la plage d'adresses CIDR VPC et configurer l'adressage IPv6
  - Utiliser un VPC existant
  - Personnalisez le VPC, les sous-réseaux, les groupes de sécurité et les passerelles NAT pour le nouveau cluster EKS
- the section called “Paramètres du sous-réseau”
  - Utilisez des sous-réseaux privés pour le groupe de nœuds initial afin de l'isoler de l'Internet public
  - Personnalisez la topologie des sous-réseaux en répertoriant plusieurs sous-réseaux par zone de disponibilité et en spécifiant des sous-réseaux dans les configurations de groupes de nœuds
  - Restreindre les groupes de nœuds à des sous-réseaux nommés spécifiques dans la configuration VPC
  - Lorsque vous utilisez des sous-réseaux privés pour des groupes de nœuds, définissez sur `privateNetworking` `true`
  - Fournissez une spécification de sous-réseau complète avec `public` les deux `private` configurations dans la spécification VPC
  - Un seul des `subnets` ou `availabilityZones` peut être fourni dans la configuration du groupe de nœuds
- the section called “Accès au cluster”
  - Gérez l'accès public et privé aux points de terminaison du serveur d'API Kubernetes dans un cluster EKS
  - Limitez l'accès au point de terminaison de l'API publique EKS Kubernetes en spécifiant les plages d'adresses CIDR autorisées
  - Mettre à jour la configuration de l'accès aux terminaux du serveur API et les restrictions CIDR d'accès public pour un cluster existant
- the section called “Mise en réseau du plan de contrôle”

  - Mettre à jour les sous-réseaux utilisés par le plan de contrôle EKS pour un cluster
- the section called “IPv6 Support”
  - Spécifiez la version IP (IPv4 ou IPv6) à utiliser lors de la création d'un VPC avec un cluster EKS

VPC Configuration

VPC dédié pour le cluster

Par défaut`eksctl create cluster`, un VPC dédié sera créé pour le cluster. Cela est fait afin d'éviter toute interférence avec les ressources existantes pour diverses raisons, notamment la sécurité, mais aussi parce qu'il est difficile de détecter tous les paramètres d'un VPC existant.

- Le CIDR VPC par défaut utilisé par est. `eksctl` `192.168.0.0/16`
  - Il est divisé en 8 (`/19`) sous-réseaux (3 privés, 3 publics et 2 réservés).
- Le groupe de nœuds initial est créé dans les sous-réseaux publics.
- L'accès SSH est désactivé sauf indication contraire`--allow-ssh`.
- Le groupe de nœuds autorise par défaut le trafic entrant en provenance du groupe de sécurité du plan de contrôle sur les ports 1025 à 65535.

> **Note**
>
> Dans `us-east-1` eksctl, il ne crée que 2 sous-réseaux publics et 2 sous-réseaux privés par défaut.

Modifier l'adresse CIDR du VPC

Si vous devez configurer le peering avec un autre VPC, ou si vous avez simplement besoin d'une plage IPs plus ou moins grande, vous pouvez `--vpc-cidr` utiliser le drapeau pour le modifier. Reportez-vous à la documentation AWS pour obtenir des guides sur le choix des blocs CIDR dont l'utilisation est autorisée dans un VPC AWS.

Si vous créez un IPv6 cluster, vous pouvez le configurer `VPC.IPv6Cidr` dans le fichier de configuration du cluster. Ce paramètre se trouve uniquement dans le fichier de configuration, pas dans un indicateur CLI.

Si vous possédez un bloc d'adresses IPv6 IP, vous pouvez également apporter votre propre IPv6 pool. Consultez Bring your own IP addresses (BYOIP) to Amazon EC2 pour savoir comment importer votre propre pool. Utilisez ensuite le fichier `VPC.IPv6Cidr` de configuration du cluster pour configurer Eksctl.

Utiliser un VPC existant : partagé avec kops

Vous pouvez utiliser le VPC d'un cluster Kubernetes existant géré par kops. Cette fonctionnalité est fournie pour faciliter le peering des and/or clusters de migration.

Si vous avez déjà créé un cluster avec kops, par exemple en utilisant des commandes similaires à celles-ci :

```
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
```

Vous pouvez créer un cluster EKS dans le même environnement AZs en utilisant les mêmes sous-réseaux VPC (REMARQUE : au moins 2 AZs/subnets sont requis) :

```
eksctl create cluster --name=cluster-2 --region=us-west-2 --vpc-from-kops-
cluster=cluster-1.k8s.local
```

Utiliser un VPC existant : autre configuration personnalisée

`eksctl`fournit une certaine flexibilité, mais pas complète, pour les topologies de VPC et de sous-réseaux personnalisées.

Vous pouvez utiliser un VPC existant en fournissant des sous-réseaux and/or publics privés à l'aide des `--vpc-private-subnets` indicateurs and. `--vpc-public-subnets` C'est à vous de vous assurer que les sous-réseaux que vous utilisez sont correctement catégorisés, car il n'existe aucun moyen simple de vérifier si un sous-réseau est réellement privé ou public, car les configurations varient.

À partir de ces indicateurs, il `eksctl create cluster` déterminera automatiquement l'ID du VPC, mais il ne créera aucune table de routage ni aucune autre ressource, telle que internet/NAT des passerelles. Il créera toutefois des groupes de sécurité dédiés pour le groupe de nœuds initial et le plan de contrôle.

Vous devez vous assurer de fournir au moins 2 sous-réseaux différents AZs et cette condition est vérifiée par EKS. Si vous utilisez un VPC existant, les exigences suivantes ne sont ni appliquées ni vérifiées par EKS ou Eksctl et EKS crée le cluster. Certaines fonctions de base du cluster fonctionnent sans ces exigences. (Par exemple, le balisage n'est pas strictement nécessaire, des tests ont montré qu'il est possible de créer un cluster fonctionnel sans qu'aucune balise ne soit définie sur les sous-réseaux, mais rien ne garantit que cela soit toujours valable et le balisage est recommandé.)

Exigences standard :

- tous les sous-réseaux donnés doivent se trouver dans le même VPC, dans le même bloc de IPs
- un nombre suffisant d'adresses IP sont disponibles, en fonction des besoins
- nombre suffisant de sous-réseaux (minimum 2), en fonction des besoins
- les sous-réseaux sont étiquetés avec au moins les éléments suivants :
  - `kubernetes.io/cluster/<name>`balise définie sur `shared` ou `owned`
  - `kubernetes.io/role/internal-elb`balise définie sur 1 pour les sous-réseaux privés
  - `kubernetes.io/role/elb`balise définie sur 1 pour les sous-réseaux publics
- passerelles Internet and/or NAT correctement configurées
- les tables de routage ont des entrées correctes et le réseau est fonctionnel
- NOUVEAU : la propriété doit être `MapPublicIpOnLaunch` activée sur tous les sous-réseaux publics (c'est-à-dire `Auto-assign public IPv4 address` dans la console AWS). Les groupes de nœuds gérés et Fargate n'attribuent pas d'adresses IPv4 publiques, la propriété doit être définie sur le sous-réseau.

D'autres exigences peuvent être imposées par EKS ou Kubernetes, et c'est à vous de vous en tenir up-to-date à ces exigences. and/or recommendations, and implement those as needed/possible

Les paramètres de groupe de sécurité par défaut appliqués par `eksctl` peuvent être suffisants ou ne pas être suffisants pour partager l'accès avec les ressources d'autres groupes de sécurité. Si vous souhaitez modifier les ingress/egress règles des groupes de sécurité, vous devrez peut-être utiliser un autre outil pour automatiser les modifications, ou le faire via la console EC2.

En cas de doute, n'utilisez pas de VPC personnalisé. L'utilisation `eksctl create cluster` sans aucun `--vpc-*` indicateur configurera toujours le cluster avec un VPC dédié entièrement fonctionnel.

Exemples

Créez un cluster à l'aide d'un VPC personnalisé avec 2 sous-réseaux privés et 2 sous-réseaux publics :

```
eksctl create cluster \
  --vpc-private-subnets=subnet-0ff156e0c4a6d300c,subnet-0426fb4a607393184 \
  --vpc-public-subnets=subnet-0153e560b3129a696,subnet-009fa0199ec203c37
```

ou utilisez le fichier de configuration équivalent suivant :

```
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
```

Créez un cluster à l'aide d'un VPC personnalisé avec 3 sous-réseaux privés et faites en sorte que le groupe de nœuds initial utilise ces sous-réseaux :

```
eksctl create cluster \
  --vpc-private-
subnets=subnet-0ff156e0c4a6d300c,subnet-0549cdab573695c03,subnet-0426fb4a607393184 \
  --node-private-networking
```

ou utilisez le fichier de configuration équivalent suivant :

```
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
```

Créez un cluster à l'aide d'un VPC 4x sous-réseaux publics personnalisé :

```
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
```

D'autres exemples peuvent être trouvés dans le `examples` dossier du dépôt :

- utilisation d'un VPC existant
- à l'aide d'un VPC CIDR personnalisé

Groupe de sécurité de nœuds partagés personnalisé

`eksctl`créera et gérera un groupe de sécurité de nœuds partagés qui permet la communication entre les nœuds non gérés, le plan de contrôle du cluster et les nœuds gérés.

Si vous souhaitez plutôt fournir votre propre groupe de sécurité personnalisé, vous pouvez remplacer le `sharedNodeSecurityGroup` champ dans le fichier de configuration :

```
vpc:
  sharedNodeSecurityGroup: sg-0123456789
```

Par défaut, lors de la création du cluster, des règles `eksctl` seront ajoutées à ce groupe de sécurité pour autoriser les communications vers et depuis le groupe de sécurité de cluster par défaut créé par EKS. Le groupe de sécurité du cluster par défaut est utilisé à la fois par le plan de contrôle EKS et par les groupes de nœuds gérés.

Si vous souhaitez gérer vous-même les règles du groupe de sécurité, vous pouvez `eksctl` empêcher leur création en définissant `manageSharedNodeSecurityGroupRules` ce qui suit `false` dans le fichier de configuration :

```
vpc:
  sharedNodeSecurityGroup: sg-0123456789
  manageSharedNodeSecurityGroupRules: false
```

Passerelle NAT

La passerelle NAT d'un cluster peut être configurée comme `Disable` suit : `Single` (par défaut) ou`HighlyAvailable`. L'`HighlyAvailable`option déploiera une passerelle NAT dans chaque zone de disponibilité de la région, de sorte que si une AZ est en panne, les nœuds de l'autre zone AZs pourront toujours communiquer avec Internet.

Il peut être spécifié via l'indicateur `--vpc-nat-mode` CLI ou dans le fichier de configuration du cluster, comme dans l'exemple ci-dessous :

```
vpc:
  nat:
    gateway: HighlyAvailable # other options: Disable, Single (default)
```

Consultez l'exemple complet ici.

> **Note**
>
> La spécification de la passerelle NAT n'est prise en charge que lors de la création du cluster. Il n'est pas touché lors d'une mise à niveau du cluster.

Paramètres du sous-réseau

Utiliser des sous-réseaux privés pour le groupe de nœuds initial

Si vous préférez isoler le groupe de nœuds initial de l'Internet public, vous pouvez utiliser le `--node-private-networking` drapeau. Lorsqu'il est utilisé conjointement avec le `--ssh-access` drapeau, le port SSH n'est accessible que depuis le VPC.

> **Note**
>
> L'utilisation de l'`--node-private-networking`indicateur fera passer le trafic sortant par la passerelle NAT à l'aide de son adresse IP élastique. D'autre part, si les nœuds se trouvent dans un sous-réseau public, le trafic sortant ne passera pas par la passerelle NAT et, par conséquent, le trafic sortant possède l'adresse IP de chaque nœud individuel.

Topologie de sous-réseau personnalisée

`eksctl`la version `0.32.0` a introduit une personnalisation supplémentaire de la topologie des sous-réseaux avec la possibilité de :

- Répertorier plusieurs sous-réseaux par AZ dans la configuration VPC
- Spécifier les sous-réseaux dans la configuration des groupes de nœuds

Dans les versions antérieures, les sous-réseaux personnalisés devaient être fournis par zone de disponibilité, ce qui signifie qu'un seul sous-réseau par AZ pouvait être répertorié. À partir de `0.32.0` l'identification, les clés peuvent être arbitraires.

```
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**
>
> Si vous utilisez l'AZ comme clé d'identification, la `az` valeur peut être omise.

Si vous utilisez une chaîne arbitraire comme clé d'identification, comme ci-dessus, soit :

- `id`doit être défini (`az`et `cidr` facultatif)
- ou `az` doit être défini (`cidr`facultatif)

Si un utilisateur spécifie un sous-réseau par AZ sans spécifier de CIDR ni d'ID, un sous-réseau de cette AZ sera choisi dans le VPC, arbitrairement s'il existe plusieurs sous-réseaux de ce type.

> **Note**
>
> Une spécification de sous-réseau complète doit être fournie, c'est-à-dire à la fois `public` et les `private` configurations déclarées dans la spécification du VPC.

Les groupes de nœuds peuvent être limités à des sous-réseaux nommés via la configuration. Lorsque vous spécifiez des sous-réseaux lors de la configuration du groupe de nœuds, utilisez la clé d'identification indiquée dans la spécification du VPC et non l'identifiant du sous-réseau. Par exemple :

```
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**
>
> Un seul des `subnets` ou `availabilityZones` peut être fourni dans la configuration du groupe de nœuds.

Lorsque vous placez des groupes de nœuds dans un sous-réseau privé, vous `privateNetworking` devez définir la valeur `true` sur le groupe de nœuds :

```
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
```

Consultez 24-nodegroup-subnets.yaml dans le dépôt eksctl pour un exemple de configuration complet. GitHub

Accès au cluster

Gestion de l'accès aux points de terminaison du serveur d'API Kubernetes

Par défaut, un cluster EKS expose le serveur d'API Kubernetes publiquement, mais pas directement depuis les sous-réseaux VPC (public=true, private=false). Le trafic destiné au serveur d'API depuis le VPC doit d'abord quitter les réseaux VPC (mais pas le réseau d'Amazon), puis y entrer à nouveau pour atteindre le serveur d'API.

L'accès au point de terminaison du serveur d'API Kubernetes pour un cluster peut être configuré pour un accès public et privé lors de la création du cluster à l'aide du fichier de configuration du cluster. Exemple ci-dessous :

```
vpc:
  clusterEndpoints:
    publicAccess:  <true|false>
    privateAccess: <true|false>
```

Il existe quelques mises en garde supplémentaires lors de la configuration de l'accès aux points de terminaison de l'API Kubernetes :

1. EKS n'autorise pas les clusters dont l'accès privé ou public n'est pas activé.

2. EKS permet de créer une configuration qui autorise uniquement l'accès privé, mais eksctl ne le prend pas en charge lors de la création du cluster car il empêche eksctl de joindre les nœuds de travail au cluster.
3. La mise à jour d'un cluster pour avoir un accès privé aux points de terminaison de l'API Kubernetes uniquement signifie que les commandes Kubernetes, par défaut, (par exemple`kubectl`) ainsi que, et éventuellement la commande `eksctl delete clustereksctl utils write-kubeconfig`, `eksctl utils update-kube-proxy` doivent être exécutées dans le VPC du cluster.
   - Cela nécessite de modifier certaines ressources AWS. Pour plus d'informations, consultez la section Point de terminaison du serveur Cluster API.
   - Vous pouvez fournir `vpc.extraCIDRs` ce qui ajoutera des plages CIDR supplémentaires au ControlPlaneSecurityGroup, permettant aux sous-réseaux extérieurs au VPC d'atteindre le point de terminaison de l'API Kubernetes. De même, vous pouvez également `vpc.extraIPv6CIDRs` prévoir d'ajouter des plages IPv6 CIDR.

Voici un exemple de la façon dont on peut configurer l'accès au point de terminaison de l'API Kubernetes à l'aide de la sous-commande : `utils`

```
eksctl utils update-cluster-vpc-config --cluster=<clustername> --private-access=true --
public-access=false
```

Pour mettre à jour le paramètre à l'aide d'un **`ClusterConfig`** fichier, utilisez :

```
eksctl utils update-cluster-vpc-config -f config.yaml --approve
```

Notez que si vous ne passez pas d'indicateur, il conservera la valeur actuelle. Une fois que vous êtes satisfait des modifications proposées, ajoutez l'`approve`indicateur pour apporter la modification au cluster en cours d'exécution.

Restreindre l'accès au point de terminaison de l'API publique EKS Kubernetes

La création par défaut d'un cluster EKS expose le serveur d'API Kubernetes publiquement.

Cette fonctionnalité s'applique uniquement au point de terminaison public. Les options de configuration de l'accès au point de terminaison du serveur API ne changeront pas et vous aurez toujours la possibilité de désactiver le point de terminaison public afin que votre cluster ne soit pas

accessible depuis Internet. (Source : feuille de https://github.com/aws/ route-conteneurs/issues/108 #issuecomment -552766489)

Pour limiter l'accès au point de terminaison d'API public à un ensemble de CIDRs lors de la création d'un cluster, définissez le **`publicAccessCIDRs`** champ :

```
vpc:
  publicAccessCIDRs: ["1.1.1.1/32", "2.2.2.0/24"]
```

Pour mettre à jour les restrictions sur un cluster existant, utilisez :

```
eksctl utils update-cluster-vpc-config --cluster=<cluster> 1.1.1.1/32,2.2.2.0/24
```

Pour mettre à jour les restrictions à l'aide d'un **`ClusterConfig`** fichier, définissez le nouveau CIDRs fichier **`vpc.publicAccessCIDRs`** et exécutez :

```
eksctl utils update-cluster-vpc-config -f config.yaml
```

> **Important**
>
> Si vous définissez `publicAccessCIDRs` et créez des groupes de nœuds, vous `privateAccess` devez définir la valeur « nœuds » `true` ou les nœuds IPs doivent être ajoutés à la liste. `publicAccessCIDRs`

Si les nœuds ne peuvent pas accéder au point de terminaison de l'API du cluster en raison d'un accès restreint, la création du cluster échouera `context deadline exceeded` car les nœuds ne pourront pas accéder au point de terminaison public et ne pourront pas rejoindre le cluster.

Pour mettre à jour à la fois l'accès aux terminaux du serveur API et CIDRs l'accès public d'un cluster en une seule commande, exécutez :

```
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
```

Pour mettre à jour le paramètre à l'aide d'un fichier de configuration :

```
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
```

Mise à jour des sous-réseaux et des groupes de sécurité du plan de contrôle

Cette documentation explique comment modifier la configuration réseau du plan de contrôle de votre cluster EKS après sa création initiale. Cela inclut la mise à jour des sous-réseaux et des groupes de sécurité du plan de contrôle.

Mise à jour des sous-réseaux du plan de contrôle

Lorsqu'un cluster est créé avec eksctl, un ensemble de sous-réseaux publics et privés est créé et transmis à l'API EKS. EKS crée 2 à 4 interfaces réseau élastiques entre comptes (ENIs) dans ces sous-réseaux pour permettre la communication entre le plan de contrôle Kubernetes géré par EKS et votre VPC.

Pour mettre à jour les sous-réseaux utilisés par le plan de contrôle EKS, exécutez :

```
eksctl utils update-cluster-vpc-config --cluster=<cluster> --control-plane-subnet-
ids=subnet-1234,subnet-5678
```

Pour mettre à jour le paramètre à l'aide d'un fichier de configuration :

```
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
```

Sans le `--approve` drapeau, eksctl enregistre uniquement les modifications proposées. Une fois que vous êtes satisfait des modifications proposées, réexécutez la commande avec le `--approve` drapeau.

Mise à jour des groupes de sécurité du plan de contrôle

Pour gérer le trafic entre le plan de contrôle et les nœuds de travail, EKS prend en charge le transfert de groupes de sécurité supplémentaires qui sont appliqués aux interfaces réseau entre comptes fournies par EKS. Pour mettre à jour les groupes de sécurité pour le plan de contrôle EKS, exécutez :

```
eksctl utils update-cluster-vpc-config --cluster=<cluster> --control-plane-security-
group-ids=sg-1234,sg-5678
```

Pour mettre à jour le paramètre à l'aide d'un fichier de configuration :

```
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
```

Pour mettre à jour à la fois les sous-réseaux du plan de contrôle et les groupes de sécurité d'un cluster, exécutez :

```
eksctl utils update-cluster-vpc-config --cluster=<cluster> --control-plane-subnet-
ids=<> --control-plane-security-group-ids=<>
```

Pour mettre à jour les deux champs à l'aide d'un fichier de configuration :

```
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
```

Pour un exemple complet, reportez-vous à cluster-subnets-sgs.yaml.

Sans le `--approve` drapeau, eksctl enregistre uniquement les modifications proposées. Une fois que vous êtes satisfait des modifications proposées, réexécutez la commande avec le `--approve` drapeau.

IPv6 Support

Définition de la famille d'adresses IP

Lors `eksctl` de la création d'un vpc, vous pouvez définir la version IP qui sera utilisée. Les options suivantes peuvent être configurées :

- IPv4
- IPv6

La valeur par défaut est `IPv4`.

Pour le définir, utilisez l'exemple suivant :

```
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**
>
> Ce paramètre se trouve uniquement dans le fichier de configuration, pas dans un indicateur CLI.

Si vous l'utilisez IPv6, vous devez configurer les exigences suivantes :

- OIDC est activé
- les addons gérés sont définis comme indiqué ci-dessus
- la version du cluster doit être => 1.21
- La version de l'addon vpc-cni doit être => 1.10.0
- les groupes de nœuds autogérés ne sont pas pris en charge par les clusters IPv6
- les groupes de nœuds gérés ne sont pas pris en charge avec des clusters non détenus IPv6
- `vpc.nat`et `serviceIPv4CIDR` les champs sont créés par eksctl pour les clusters IPv6 et ne sont pas des options de configuration prises en charge
- AutoAllocateIPv6 n'est pas pris en charge conjointement avec IPv6
- Pour le IPv6 cluster, le rôle IAM de vpc-cni doit avoir les politiques IAM requises pour le mode associé IPv6

La mise en réseau privée peut également être réalisée avec la famille IPv6 IP. Veuillez suivre les instructions décrites dans la section Cluster privé EKS.

IAM

Ce chapitre contient des informations sur l'utilisation d'AWS IAM.

Rubriques :

- the section called “Gestion des utilisateurs et des rôles IAM”
  - Gérez les mappages d'utilisateurs et de rôles IAM pour contrôler l'accès à un cluster EKS
  - Configurer les mappages d'identité IAM via le fichier de configuration du cluster ou les commandes de la CLI
- the section called “Rôles IAM pour les comptes de service”
  - Gérez des autorisations détaillées pour les applications exécutées sur Amazon EKS qui utilisent d'autres services AWS
  - Créez et configurez des paires de rôles IAM et de comptes de service Kubernetes à l'aide d'eksctl
  - Activer le fournisseur IAM OpenID Connect pour un cluster EKS afin d'activer les rôles IAM pour les comptes de service
- the section called “Limite des autorisations IAM”
  - Contrôlez les autorisations maximales accordées aux entités IAM (utilisateurs ou rôles) en définissant une limite d'autorisations
- the section called “Associations d'identité EKS Pod”
  - Configurer les autorisations IAM pour les modules complémentaires EKS à l'aide des associations d'identité de pod recommandées
  - Permettre aux applications Kubernetes de recevoir les autorisations IAM requises pour se connecter aux services AWS en dehors du cluster
  - Simplifiez le processus d'automatisation des rôles IAM et des comptes de service sur plusieurs clusters EKS
- the section called “politiques IAM”
  - Gérez les politiques IAM pour les groupes de nœuds EKS, notamment en prenant en charge diverses politiques complémentaires telles que le générateur d'images, le scaleur automatique, le DNS externe, le gestionnaire de certificats, etc.
  - Associez des rôles d'instance personnalisés ou des politiques intégrées aux groupes de nœuds pour obtenir des autorisations supplémentaires.

  - Associez des politiques spécifiques gérées par AWS par ARN aux groupes de nœuds, afin de vous assurer que les politiques requises telles qu'Amazon EKSWorker NodePolicy et Amazoneks_CNI_Policy sont incluses.
- the section called “Politiques IAM minimales”
  - Gérez les ressources AWS EC2, notamment les équilibreurs de charge, les groupes d'auto-scaling et la surveillance CloudWatch
  - Création et gestion de CloudFormation piles AWS
  - Gérez les clusters Amazon Elastic Kubernetes Service (EKS), les groupes de nœuds et les ressources associées telles que les rôles et les politiques IAM

Politiques IAM minimales

Ce document décrit les politiques IAM minimales nécessaires pour exécuter les principaux cas d'utilisation d'eksctl. Ce sont ceux utilisés pour exécuter les tests d'intégration.

> **Note**
>
> N'oubliez pas de le `<account_id>` remplacer par le vôtre.

> **Note**
>
> Une politique gérée par AWS est créée et administrée par AWS. Vous ne pouvez pas modifier les autorisations définies dans les politiques gérées par AWS.

Amazon EC2 FullAccess (politique gérée par AWS)

Consultez la définition EC2 FullAccess de la politique Amazon.

AWSCloudFormationFullAccess (Politique gérée par AWS)

Afficher la définition AWSCloud FormationFullAccess de la politique.

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"
                    ]
                }
            }
        }
    ]
}
```

Limite des autorisations IAM

Une limite d'autorisations est une fonctionnalité avancée d'AWS IAM dans laquelle les autorisations maximales qu'une politique basée sur l'identité peut accorder à une entité IAM ont été définies ; ces entités étant soit des utilisateurs, soit des rôles. Lorsqu'une limite d'autorisation est définie pour une entité, celle-ci ne peut effectuer que les actions autorisées à la fois par ses politiques basées sur l'identité et par ses limites d'autorisations.

Vous pouvez fournir votre limite d'autorisations afin que toutes les entités basées sur l'identité créées par eksctl soient créées dans cette limite. Cet exemple montre comment une limite d'autorisations peut être fournie aux différentes entités basées sur l'identité créées par eksctl :

```
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**
>
> Il n'est pas possible de fournir à la fois un ARN de rôle et une limite d'autorisations.

Définition de la limite d'autorisation CNI du VPC

Veuillez noter que lorsque vous créez un cluster avec OIDC activé, eksctl en créera automatiquement un `iamserviceaccount` pour le VPC-CNI pour des raisons de sécurité. Si vous souhaitez y ajouter une limite d'autorisation, vous devez la spécifier manuellement `iamserviceaccount` dans votre fichier de configuration :

```
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"
```

politiques IAM

Vous pouvez associer des rôles d'instance à des groupes de nœuds. Les charges de travail exécutées sur le nœud recevront des autorisations IAM de la part du nœud. Pour plus d'informations, consultez la section Rôles IAM pour Amazon EC2.

Cette page répertorie les modèles de politique IAM prédéfinis disponibles dans eksctl. Ces modèles simplifient le processus d'octroi à vos nœuds EKS des autorisations de service AWS appropriées sans avoir à créer manuellement des politiques IAM personnalisées.

Politiques complémentaires IAM prises en charge

Exemple de toutes les politiques relatives aux modules complémentaires prises en charge :

```
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
```

Politique d'Image Builder

La `imageBuilder` politique permet un accès complet à l'ECR (Elastic Container Registry). Cela est utile pour créer, par exemple, un serveur CI qui doit envoyer des images vers ECR.

Politique EBS

La `ebs` politique active le nouveau pilote EBS CSI (Elastic Block Store Container Storage Interface).

Politique du gestionnaire de certificats

La `certManager` politique permet d'ajouter des enregistrements à Route 53 afin de relever le DNS01 défi. Vous trouverez plus d'informations ici.

Ajouter un rôle d'instance personnalisé

Cet exemple crée un groupe de nœuds qui réutilise un rôle d'instance IAM existant provenant d'un autre cluster :

```
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"
```

Joindre des politiques en ligne

```
nodeGroups:
  - name: my-special-nodegroup
    iam:
      attachPolicy:
        Version: "2012-10-17"
        Statement:
        - Effect: Allow
          Action:
          - 's3:GetObject'
          Resource: 'arn:aws:s3:::example-bucket/*'
```

Joindre des politiques par ARN

```
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

Si un groupe de nœuds inclut le, `attachPolicyARNs` il doit également inclure les politiques de nœud par défaut`AmazonEKSWorkerNodePolicy`, `AmazonEKS_CNI_Policy` comme `AmazonEC2ContainerRegistryPullOnly` dans cet exemple.

Gestion des utilisateurs et des rôles IAM

Note

AWS suggère de migrer vers lethe section called “Associations d'identité EKS Pod”. `aws-auth` ConfigMap

Les clusters EKS utilisent des utilisateurs et des rôles IAM pour contrôler l'accès au cluster. Les règles sont implémentées dans une carte de configuration

Modifier ConfigMap à l'aide d'une commande CLI

appeléaws-auth. `eksctl`fournit des commandes pour lire et modifier cette carte de configuration.

Obtenez tous les mappages d'identité :

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

Obtenez tous les mappages d'identité correspondant à un ARN :

```
eksctl get iamidentitymapping --cluster <clusterName> --region=<region> --arn
 arn:aws:iam::123456:role/testing-role
```

Créez un mappage d'identité :

```
 eksctl create iamidentitymapping --cluster  <clusterName> --region=<region> --arn
 arn:aws:iam::123456:role/testing --group system:masters --username admin
```

Supprimer un mappage d'identité :

```
eksctl delete iamidentitymapping --cluster  <clusterName> --region=<region> --arn
 arn:aws:iam::123456:role/testing
```

> **Note**
>
> La commande ci-dessus supprime une seule FIFO de mappage sauf indication contraire`--all`, auquel cas elle supprime toutes les correspondances. Avertira si d'autres mappages correspondant à ce rôle sont trouvés.

Créez un mappage de compte :

```
 eksctl create iamidentitymapping --cluster  <clusterName> --region=<region> --account
 user-account
```

Supprimer un mappage de compte :

```
 eksctl delete iamidentitymapping --cluster  <clusterName> --region=<region> --account
 user-account
```

Modifier ConfigMap à l'aide d'un ClusterConfig fichier

Les mappages d'identité peuvent également être spécifiés dans ClusterConfig :

```
---
```

```
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
```

Rôles IAM pour les comptes de service

> **Tip**
>
> `eksctl`prend en charge la configuration d'autorisations détaillées pour les applications EKS exécutant des applications via EKS Pod Identity Associations

Amazon EKS prend en charge ici [Roles for Service Accounts (IRSA)], qui permet aux opérateurs de clusters de mapper les rôles AWS IAM aux comptes de service Kubernetes.

Cela permet une gestion précise des autorisations pour les applications qui s'exécutent sur EKS et utilisent d'autres services AWS. Il peut s'agir d'applications utilisant S3, tout autre service de données (RDS, MQ, STS, DynamoDB) ou des composants Kubernetes tels que le contrôleur AWS Load Balancer ou ExternalDNS.

Vous pouvez facilement créer des paires de rôles IAM et de comptes de service avec`eksctl`.

> **Note**
>
> Si vous avez utilisé des rôles d'instance et que vous envisagez d'utiliser IRSA à la place, vous ne devez pas mélanger les deux.

Comment ça marche

Il fonctionne via le fournisseur IAM OpenID Connect (OIDC) exposé par EKS, et les rôles IAM doivent être construits en référence au fournisseur IAM OIDC (spécifique à un cluster EKS donné) et en référence au compte de service Kubernetes auquel il sera lié. Une fois qu'un rôle IAM est créé, un compte de service doit inclure l'ARN de ce rôle sous forme d'annotation (`eks.amazonaws.com/role-arn`). Par défaut, le compte de service sera créé ou mis à jour pour inclure l'annotation du rôle. Cela peut être désactivé à l'aide du drapeau`--role-only`.

Dans EKS, il existe un contrôleur d'admission qui injecte les informations d'identification de session AWS dans les pods respectivement des rôles en fonction de l'annotation sur le compte de service utilisé par le pod. Les informations d'identification seront exposées par les variables `AWS_ROLE_ARN` d'`AWS_WEB_IDENTITY_TOKEN_FILE`environnement &. Étant donné qu'une version récente du SDK AWS est utilisée (voir ici pour plus de détails sur la version exacte), l'application utilisera ces informations d'identification.

`eksctl`Le nom de la ressource est iamserviceaccount, qui représente une paire de rôle IAM et de compte de service.

Utilisation depuis la CLI

> **Note**
>
> Les rôles IAM pour les comptes de service nécessitent la version 1.13 ou supérieure de Kubernetes.

Le fournisseur IAM OIDC n'est pas activé par défaut, vous pouvez utiliser la commande suivante pour l'activer ou utiliser le fichier de configuration (voir ci-dessous) :

```
eksctl utils associate-iam-oidc-provider --cluster=<clusterName>
```

Une fois que le fournisseur IAM OIDC est associé au cluster, pour créer un rôle IAM lié à un compte de service, exécutez :

```
eksctl create iamserviceaccount --cluster=<clusterName> --name=<serviceAccountName> --
namespace=<serviceAccountNamespace> --attach-policy-arn=<policyARN>
```

> **Note**
>
> Vous pouvez spécifier `--attach-policy-arn` plusieurs fois l'utilisation de plusieurs politiques.

Plus précisément, vous pouvez créer un compte de service avec un accès en lecture seule à S3 en exécutant :

```
eksctl create iamserviceaccount --cluster=<clusterName> --name=s3-read-only --attach-
policy-arn=arn:aws:iam::aws:policy/AmazonS3ReadOnlyAccess
```

Par défaut, il sera créé dans l'espace de `default` noms, mais vous pouvez spécifier n'importe quel autre espace de noms, par exemple :

```
eksctl create iamserviceaccount --cluster=<clusterName> --name=s3-read-only --
namespace=s3-app --attach-policy-arn=arn:aws:iam::aws:policy/AmazonS3ReadOnlyAccess
```

> **Note**
>
> Si l'espace de noms n'existe pas déjà, il sera créé.

Si vous avez déjà créé un compte de service dans le cluster (sans rôle IAM), vous devrez utiliser le `--override-existing-serviceaccounts` drapeau.

Le balisage personnalisé peut également être appliqué au rôle IAM en spécifiant : `--tags`

```
eksctl create iamserviceaccount --cluster=<clusterName> --name=<serviceAccountName> --
tags "Owner=John Doe,Team=Some Team"
```

CloudFormation générera un nom de rôle incluant une chaîne aléatoire. Si vous préférez un nom de rôle prédéterminé, vous pouvez spécifier `--role-name` :

```
eksctl create iamserviceaccount --cluster=<clusterName> --name=<serviceAccountName> --
role-name "custom-role-name"
```

Lorsque le compte de service est créé et géré par un autre outil, tel que Helm, utilisez-le `--role-only` pour éviter les conflits. L'autre outil est alors chargé de gérer l'annotation ARN du rôle. Notez que cela n'`--override-existing-serviceaccounts`a aucun effet sur les comptes`roleOnly`/`--role-only`service, le rôle sera toujours créé.

```
eksctl create iamserviceaccount --cluster=<clusterName> --name=<serviceAccountName> --
role-only --role-name=<customRoleName>
```

Lorsque vous avez un rôle existant que vous souhaitez utiliser avec un compte de service, vous pouvez fournir le `--attach-role-arn` drapeau au lieu de fournir les politiques. Pour garantir que le rôle ne peut être assumé que par le compte de service spécifié, vous devez définir un document de politique de relation ici].

```
eksctl create iamserviceaccount --cluster=<clusterName> --name=<serviceAccountName> --
attach-role-arn=<customRoleARN>
```

Pour mettre à jour les autorisations des rôles d'un compte de service, vous pouvez exécuter`eksctl update iamserviceaccount`.

> **Note**
>
> `eksctl delete iamserviceaccount`supprime Kubernetes `ServiceAccounts` même s'ils n'ont pas été créés par. `eksctl`

Utilisation avec des fichiers de configuration

Pour gérer à `iamserviceaccounts` l'aide du fichier de configuration, vous devrez définir `iam.withOIDC: true` et répertorier le compte sous lequel vous souhaitez accéder`iam.serviceAccount`.

Toutes les commandes sont prises en charge*`--config-file`*, vous pouvez gérer les comptes iamservicede la même manière que les groupes de nœuds. La `eksctl create iamserviceaccount` commande prend en charge `--include` et `--exclude` signale (voir cette section pour plus de détails sur leur fonctionnement). Et la `eksctl delete iamserviceaccount` commande le prend également `--only-missing` en charge, vous pouvez donc effectuer des suppressions de la même manière que les groupes de nœuds.

> **Note**
>
> Les comptes de service IAM sont limités à un espace de noms, c'est-à-dire que deux comptes de service portant le même nom peuvent exister dans des espaces de noms différents. Ainsi, pour définir de manière unique un compte de service dans le cadre de `--exclude` drapeaux`--include`, vous devez transmettre la chaîne de nom au `namespace/name` format. P. ex.

```
eksctl create iamserviceaccount --config-file=<path> --include backend-apps/s3-reader
```

L'option d'activation `wellKnownPolicies` est incluse pour utiliser IRSA dans des cas d'utilisation bien connus tels que `cluster-autoscaler` et`cert-manager`, en tant que raccourci pour les listes de politiques.

Les politiques connues prises en charge et les autres propriétés de `serviceAccounts` sont documentées dans le schéma de configuration.

Vous utilisez l'exemple de configuration suivant avec `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
```

Si vous créez un cluster sans définir ces champs, vous pouvez utiliser les commandes suivantes pour activer tout ce dont vous avez besoin :

```
eksctl utils associate-iam-oidc-provider --config-file=<path>
eksctl create iamserviceaccount --config-file=<path>
```

Informations supplémentaires

- Présentation de rôles IAM précis pour les comptes de service
- Guide de l'utilisateur EKS - Rôles IAM pour les comptes de service
- Mappage des utilisateurs et des rôles IAM avec les rôles Kubernetes RBAC

Associations d'identité EKS Pod

AWS EKS a introduit un nouveau mécanisme amélioré appelé Pod Identity Association permettant aux administrateurs de clusters de configurer les applications Kubernetes afin de recevoir les autorisations IAM requises pour se connecter aux services AWS en dehors du cluster. Pod Identity Association tire parti de l'IRSA, mais le rend configurable directement via l'API EKS, éliminant ainsi le besoin d'utiliser l'API IAM.

Par conséquent, les rôles IAM n'ont plus besoin de faire référence à un fournisseur OIDC et ne seront donc plus liés à un seul cluster. Cela signifie que les rôles IAM peuvent désormais être utilisés sur plusieurs clusters EKS sans qu'il soit nécessaire de mettre à jour la politique de confiance des rôles à chaque fois qu'un nouveau cluster est créé. Cela élimine à son tour le besoin de duplication des rôles et simplifie complètement le processus d'automatisation de l'IRSA.

Conditions préalables

Dans les coulisses, la mise en œuvre des associations d'identité des pods consiste à exécuter un agent en tant que daemonset sur les nœuds de travail. Pour exécuter l'agent prérequis sur le cluster, EKS fournit un nouveau module complémentaire appelé EKS Pod Identity Agent. Par conséquent, la création d'associations d'identité de pod (en général et avec`eksctl`) nécessite que l'`eks-pod-identity-agent`addon soit préinstallé sur le cluster. Cet addon peut être créé de la même manière que n'importe quel autre addon compatible. `eksctl`

```
eksctl create addon --cluster my-cluster --name eks-pod-identity-agent
```

En outre, si vous utilisez un rôle IAM préexistant lors de la création d'une association d'identité de pod, vous devez configurer le rôle pour qu'il fasse confiance au nouveau principal de service EKS ()`pods.eks.amazonaws.com`. Vous trouverez ci-dessous un exemple de politique de confiance IAM :

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

Si, au contraire, vous ne fournissez pas l'ARN d'un rôle existant à la commande create, `eksctl` vous en créerez un en arrière-plan et configurerez la politique de confiance ci-dessus.

Création d'associations d'identité de pods

Pour manipuler les associations d'identité des pods, `eksctl` a ajouté un nouveau champ sous`iam.podIdentityAssociations`, par ex.

```
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
```

Pour un exemple complet, reportez-vous à pod-identity-associations.yaml.

> **Note**
>
> Hormis celui `permissionPolicy` qui est utilisé comme document de politique intégré, tous les autres champs ont un équivalent d'indicateur CLI.

La création d'associations d'identité de pods peut être réalisée de la manière suivante. Lors de la création du cluster, en spécifiant les associations d'identité de pod souhaitées dans le fichier de configuration et en exécutant :

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

Après la création du cluster, en utilisant soit un fichier de configuration, par exemple

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

OU en utilisant des drapeaux CLI, par exemple

```
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**
>
> Un seul rôle IAM peut être associé à un compte de service à la fois. Par conséquent, toute tentative de création d'une deuxième association d'identité de pod pour le même compte de service entraînera une erreur.

Récupération des associations d'identité des pods

Pour récupérer toutes les associations d'identité des pods pour un cluster donné, exécutez l'une des commandes suivantes :

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

OU

```
eksctl get podidentityassociation --cluster my-cluster
```

De plus, pour récupérer uniquement les associations d'identité des pods dans un espace de noms donné, utilisez le `--namespace` drapeau, par exemple

```
eksctl get podidentityassociation --cluster my-cluster --namespace default
```

Enfin, pour récupérer une seule association, correspondant à un certain compte de service K8s, incluez également la `--service-account-name` commande ci-dessus, c'est-à-dire

```
eksctl get podidentityassociation --cluster my-cluster --namespace default --service-
account-name s3-reader
```

Mettre à jour les associations d'identité des pods

Pour mettre à jour le rôle IAM d'une ou de plusieurs associations d'identité de pod, transmettez la nouvelle `roleARN(s)` au fichier de configuration, par exemple

```
iam:
  podIdentityAssociations:
    - namespace: default
      serviceAccountName: s3-reader
      roleARN: new-role-arn-1
    - namespace: dev
      serviceAccountName: app-cache-access
      roleARN: new-role-arn-2
```

et lancez :

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

OU (pour mettre à jour une seule association) transmettez les nouveaux indicateurs `--role-arn` via la CLI :

```
eksctl update podidentityassociation --cluster my-cluster --namespace default --
service-account-name s3-reader --role-arn new-role-arn
```

Supprimer les associations d'identité des pods

Pour supprimer une ou plusieurs associations d'identité de pod, transmettez-les `namespace(s)` `serviceAccountName(s)` au fichier de configuration, par exemple

```
iam:
  podIdentityAssociations:
    - namespace: default
      serviceAccountName: s3-reader
    - namespace: dev
      serviceAccountName: app-cache-access
```

et lancez :

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

OU (pour supprimer une seule association) transmettez les indicateurs `--namespace` et `--service-account-name` via la CLI :

```
eksctl delete podidentityassociation --cluster my-cluster --namespace default --
service-account-name s3-reader
```

Support des modules complémentaires EKS pour les associations d'identité des pods

Les modules complémentaires EKS permettent également de recevoir des autorisations IAM via EKS Pod Identity Associations. Le fichier de configuration expose trois champs qui permettent de les configurer :`addon.podIdentityAssociations`, `addonsConfig.autoApplyPodIdentityAssociations` et`addon.useDefaultPodIdentityAssociations`. Vous pouvez soit configurer explicitement les associations d'identité d'espace souhaitées, en utilisant`addon.podIdentityAssociations`, soit

résoudre (et appliquer) `eksctl` automatiquement la configuration d'identité d'espace recommandée, en utilisant l'un `addonsConfig.autoApplyPodIdentityAssociations` ou l'autre des deux`addon.useDefaultPodIdentityAssociations`.

> **Note**
>
> Les modules complémentaires EKS ne prendront pas tous en charge les associations d'identité des pods au lancement. Dans ce cas, les autorisations IAM requises continueront d'être fournies à l'aide des paramètres IRSA.

Création d'addons avec des autorisations IAM

Lorsque vous créez un addon nécessitant des autorisations IAM, `eksctl` vous devez d'abord vérifier si les associations d'identité du pod ou les paramètres IRSA sont explicitement configurés dans le fichier de configuration, et si c'est le cas, utiliser l'un d'entre eux pour configurer les autorisations pour l'addon. Par exemple

```
addons:
- name: vpc-cni
  podIdentityAssociations:
  - serviceAccountName: aws-node
    permissionPolicyARNs: ["arn:aws:iam::aws:policy/AmazonEKS_CNI_Policy"]
```

et courez

```
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**
>
> La définition simultanée de l'identité du pod et de l'IRSA n'est pas autorisée et entraînera une erreur de validation.

Pour les modules complémentaires EKS qui prennent en charge les identités `eksctl` des pods, offre la possibilité de configurer automatiquement les autorisations IAM recommandées, lors de la création

de l'addon. Cela peut être réalisé `addonsConfig.autoApplyPodIdentityAssociations: true` en configurant simplement le fichier de configuration. par exemple

```
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
```

et courez

```
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
```

De manière équivalente, la même chose peut être faite via les drapeaux de la CLI, par exemple

```
eksctl create addon --cluster my-cluster --name vpc-cni --auto-apply-pod-identity-
associations
```

Pour migrer un addon existant afin d'utiliser l'identité du pod avec les politiques IAM recommandées, utilisez

```
addons:
- name: vpc-cni
  useDefaultPodIdentityAssociations: true
```

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

Mise à jour des addons avec des autorisations IAM

Lors de la mise à jour d'un addon, la spécification `addon.PodIdentityAssociations` représentera la source unique de vérité pour l'état que l'addon devra avoir, une fois l'opération de mise à jour terminée. Dans les coulisses, différents types d'opérations sont effectués afin d'atteindre l'état souhaité, c'est-à-dire

- créer des identifiants de pod présents dans le fichier de configuration, mais absents du cluster
- supprimer les identifiants de pod existants qui ont été supprimés du fichier de configuration, ainsi que toutes les ressources IAM associées
- mettre à jour les identités de pod existantes qui sont également présentes dans le fichier de configuration et pour lesquelles l'ensemble des autorisations IAM a changé

> **Note**
>
> Le cycle de vie des associations d'identité des pods détenues par EKS Add-ons est directement géré par l'API EKS Addons.

Vous ne pouvez pas utiliser `eksctl update podidentityassociation` (pour mettre à jour les autorisations IAM) ou `eksctl delete podidentityassociations` (pour supprimer l'association) pour les associations utilisées avec un module complémentaire Amazon EKS. Au lieu de cela, `eksctl update addon` ou `eksctl delete addon` doit être utilisé.

Voyons un exemple pour ce qui précède, en commençant par analyser la configuration initiale de l'identité du pod pour l'addon :

```
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"
```

```
    }
]
```

Utilisez maintenant la configuration ci-dessous :

```
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
```

et courez

```
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
```

vérifiez maintenant que la configuration de l'identité du pod a été correctement mise à jour

```
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"
    }
]
```

Pour supprimer toutes les associations d'identité de pod d'un addon, celui-ci `addon.PodIdentityAssociations` doit être explicitement défini sur`[]`, par exemple

```
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: []
```

et courez

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

Supprimer des addons avec des autorisations IAM

La suppression d'un addon supprimera également toutes les identités de pod associées à l'addon. La suppression du cluster produira le même effet pour tous les addons. Tous les rôles IAM pour les identités des pods, créés par`eksctl`, seront également supprimés.

Migration de comptes iamservice et d'extensions existants vers des associations d'identité de pod

Il existe une commande `eksctl` utils pour migrer les rôles IAM existants pour les comptes de service vers des associations d'identité de pod, c'est-à-dire

```
eksctl utils migrate-to-pod-identity --cluster my-cluster --approve
```

Dans les coulisses, la commande appliquera les étapes suivantes :

- installer l'`eks-pod-identity-agent`addon s'il n'est pas déjà actif sur le cluster
- identifier tous les rôles IAM associés à iamserviceaccounts
- identifier tous les rôles IAM associés aux modules complémentaires EKS qui prennent en charge les associations d'identité des pods
- mettre à jour la politique de confiance IAM de tous les rôles identifiés, avec une entité de confiance supplémentaire, pointant vers le nouveau principal du service EKS (et, éventuellement, supprimer la relation de confiance existante avec le fournisseur OIDC)
- créer des associations d'identité de pod pour les rôles filtrés associés à iamserviceaccounts
- mettre à jour les extensions EKS avec les identités des pods (l'API EKS créera les identités des pods en arrière-plan)

L'exécution de la commande sans l'`--approve`indicateur produira uniquement un plan composé d'un ensemble de tâches reflétant les étapes ci-dessus, par ex.

```
[#]  (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
```

La relation de confiance existante avec le fournisseur OIDC est toujours supprimée des rôles IAM associés aux modules complémentaires EKS. En outre, pour supprimer la relation de confiance existante du fournisseur OIDC dans les rôles IAM associés à iamserviceaccounts, exécutez la commande avec un indicateur, par exemple `--remove-oidc-provider-trust-relationship`

```
eksctl utils migrate-to-pod-identity --cluster my-cluster --approve --remove-oidc-
provider-trust-relationship
```

Support d'identité multi-comptes Pod

eksctl prend en charge l'accès entre comptes EKS Pod Identity. Cette fonctionnalité permet aux pods exécutés dans votre cluster EKS d'accéder aux ressources AWS depuis un autre compte AWS.

Usage

Pour créer une association d'identité de pod avec un accès entre comptes, configurez d'abord les rôles et politiques IAM autorisant l'accès d'un compte AWS source (avec le cluster) à un compte AWS cible (avec les ressources auxquelles le cluster peut accéder). Pour un exemple, consultez « Amazon EKS Pod Identity rationalise l'accès entre comptes ».

Une fois qu'un rôle IAM est configuré dans chaque compte, utilisez eksctl pour créer les associations d'identité du pod :

```
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
```

Autres références

Support officiel d'AWS Userdocs pour les modules complémentaires EKS pour les identités des pods

Article de blog officiel d'AWS sur les associations d'identité des pods

Documents utilisateur AWS officiels pour Pod Identity Associations

Options de déploiement

Ce chapitre décrit l'utilisation d'eksctl pour gérer les clusters EKS déployés dans d'autres environnements.

Pour obtenir les informations les plus précises sur les options de déploiement d'EKS, consultez la section Déployer des clusters Amazon EKS dans des environnements cloud et sur site dans le guide de l'utilisateur EKS.

Rubriques :

- the section called “EKS N'importe où”
  - Utilisez eksctl avec les clusters Amazon EKS Anywhere.
  - Amazon EKS Anywhere est un logiciel de gestion de conteneurs développé par AWS qui facilite l'exécution et la gestion de Kubernetes sur site et en périphérie.
- the section called “Support pour AWS Outposts”
  - Utilisez eksctl avec des clusters EKS sur AWS Outposts.
  - AWS Outposts est une famille de solutions entièrement gérées fournissant une infrastructure et des services AWS à pratiquement tous les sites locaux ou périphériques pour une expérience hybride vraiment cohérente.
  - La prise en charge d'AWS Outposts dans eksctl vous permet de créer des clusters locaux avec l'ensemble du cluster Kubernetes, y compris le plan de contrôle EKS et les nœuds de travail, exécutés localement sur AWS Outposts.
- the section called “Nœuds hybrides EKS”
  - Exécutez des applications sur site et de périphérie sur une infrastructure gérée par le client avec les mêmes clusters, fonctionnalités et outils AWS EKS que ceux que vous utilisez dans le cloud AWS.

EKS N'importe où

`eksctl`fournit un accès à la fonctionnalité AWS appelée `EKS Anywhere` avec la sous-commande`eksctl anywhere`. Cela nécessite que le `eksctl-anywhere` binaire soit activé`PATH`. Veuillez suivre les instructions décrites ici Installer eksctl-anywhere pour l'installer.

Une fois cela fait, exécutez les commandes n'importe où en exécutant :

```
eksctl anywhere version
v0.5.0
```

Pour plus d'informations sur EKS Anywhere, rendez-vous sur le site Web d'EKS Anywhere.

Support pour AWS Outposts

> **Warning**
>
> Les groupes de nœuds gérés par EKS ne sont pas pris en charge sur les Outposts.

Étendre les clusters existants à AWS Outposts

Vous pouvez étendre un cluster EKS existant exécuté dans une région AWS à AWS Outposts en configurant de nouveaux groupes de nœuds afin de créer `nodeGroup.outpostARN` des groupes de nœuds sur les Outposts, comme dans :

```
# 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
```

Dans cette configuration, le plan de contrôle EKS s'exécute dans une région AWS tandis que les groupes de nœuds `outpostARN` définis s'exécutent sur l'avant-poste spécifié. Lorsqu'un groupe de

nœuds est créé sur Outposts pour la première fois, eksctl étend le VPC en créant des sous-réseaux sur l'Outpost spécifié. Ces sous-réseaux sont utilisés pour créer des groupes de nœuds définis. `outpostARN`

Les clients possédant un VPC préexistant sont tenus de créer les sous-réseaux sur les Outposts et de les transmettre`nodeGroup.subnets`, comme dans :

```
# 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"
```

Création d'un cluster local sur AWS Outposts

> **Note**
>
> Les clusters locaux prennent uniquement en charge les racks Outpost.

> **Note**
>
> Seul Amazon Linux 2 est pris en charge pour les groupes de nœuds lorsque le plan de contrôle se trouve sur Outposts. Seuls les types de volumes EBS gp2 sont pris en charge pour les groupes de nœuds sur les Outposts.

La prise en charge d'AWS Outposts dans eksctl vous permet de créer des clusters locaux avec l'ensemble du cluster Kubernetes, y compris le plan de contrôle EKS et les nœuds de travail, exécutés localement sur AWS Outposts. Les clients peuvent soit créer un cluster local avec le plan de contrôle EKS et les nœuds de travail exécutés localement sur AWS Outposts, soit étendre un cluster EKS existant exécuté dans une région AWS à AWS Outposts en créant des nœuds de travail sur les Outposts.

Pour créer le plan de contrôle EKS et les groupes de nœuds sur AWS Outposts, `outpost.controlPlaneOutpostARN` définissez l'Outpost ARN, comme dans :

```
# 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
```

Cela indique à eksctl de créer le plan de contrôle EKS et les sous-réseaux sur l'avant-poste spécifié. Comme un rack Outposts existe dans une seule zone de disponibilité, eksctl ne crée qu'un seul sous-réseau public et privé. eksctl n'associe pas le VPC créé à une passerelle locale et, par conséquent, eksctl ne sera pas connecté au serveur API et ne pourra pas créer de groupes de nœuds. Par

conséquent, s'il `ClusterConfig` contient des groupes de nœuds lors de la création du cluster, la commande doit être exécutée avec`--without-nodegroup`, comme dans :

```
eksctl create cluster -f outpost.yaml --without-nodegroup
```

Il est de la responsabilité du client d'associer le VPC créé par eksctl à la passerelle locale après la création du cluster afin de permettre la connectivité au serveur API. Après cette étape, les groupes de nœuds peuvent être créés à l'aide de. `eksctl create nodegroup`

Vous pouvez éventuellement spécifier le type d'instance pour les nœuds du plan de contrôle dans `outpost.controlPlaneInstanceType` ou pour les groupes de nœuds dans`nodeGroup.instanceType`, mais le type d'instance doit exister sur Outpost, sinon eksctl renverra une erreur. Par défaut, eksctl essaie de choisir le plus petit type d'instance disponible sur Outpost pour les nœuds et les groupes de nœuds du plan de contrôle.

Lorsque le plan de contrôle se trouve sur les Outposts, des groupes de nœuds sont créés sur ces Outposts. Vous pouvez éventuellement spécifier l'ARN d'avant-poste pour le groupe de nœuds dans, `nodeGroup.outpostARN` mais il doit correspondre à l'ARN d'avant-poste du plan de contrôle.

```
# 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
```

VPC existant

Les clients disposant d'un VPC existant peuvent créer des clusters locaux sur AWS Outposts en spécifiant la configuration `vpc.subnets` du sous-réseau dans, comme dans :

```
# 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
```

Les sous-réseaux doivent exister sur l'avant-poste spécifié dans, `outpost.controlPlaneOutpostARN` sinon eksctl renverra une erreur. Vous pouvez également spécifier des groupes de nœuds lors de la création du cluster si vous avez accès à la passerelle locale pour le sous-réseau ou si vous êtes connecté aux ressources VPC.

Fonctionnalités non prises en charge sur les clusters locaux

- Modules complémentaires
- Rôles IAM pour les comptes de service
- IPv6
- Fournisseurs d'identité
- Fargate
- Chiffrement KMS
- Zones locales
- Charpentier
- Sélecteur d'instance
- Les zones de disponibilité ne peuvent pas être spécifiées car par défaut, il s'agit de la zone de disponibilité de l'avant-poste.
- `vpc.publicAccessCIDRs` et `vpc.autoAllocateIPv6` ne sont pas prises en charge.
- L'accès des terminaux publics au serveur API n'est pas pris en charge car un cluster local ne peut être créé qu'avec un accès réservé aux terminaux privés.

Plus d'informations

- Amazon EKS sur AWS Outposts
- Clusters locaux pour Amazon EKS sur AWS Outposts
- Création de clusters locaux
- Lancement de nœuds Amazon Linux autogérés sur un Outpost

Sécurité

`eksctl`propose certaines options qui peuvent améliorer la sécurité de votre cluster EKS.

`withOIDC`

Activez `withOIDC`cette option pour créer automatiquement un IRSA pour le plugin Amazon CNI et limiter les autorisations accordées aux nœuds de votre cluster, au lieu d'accorder les autorisations nécessaires uniquement au compte de service CNI.

Le contexte est décrit dans cette documentation AWS.

`disablePodIMDS`

Pour les groupes de nœuds gérés et non gérés, l'`disablePodIMDS`option est disponible pour empêcher tous les pods réseau non hôtes exécutés dans ce groupe de nœuds de faire des demandes IMDS.

> **Note**
>
> Cela ne peut pas être utilisé conjointement avec `withAddonPolicies`.

Chiffrement de l'enveloppe KMS pour les clusters EKS

> **Note**
>
> Amazon Elastic Kubernetes Service (Amazon EKS) fournit un chiffrement par défaut pour toutes les données API Kubernetes dans les clusters EKS exécutant la version 1.28 ou supérieure de Kubernetes. Pour plus d'informations, consultez la section Chiffrement d'enveloppe par défaut pour toutes les données de l'API Kubernetes dans le guide de l'utilisateur EKS.

EKS prend en charge l'utilisation des clés AWS KMS pour chiffrer l'enveloppe des secrets Kubernetes stockés dans EKS. Le chiffrement des enveloppes ajoute une couche de chiffrement

supplémentaire gérée par le client pour les secrets d'application ou les données utilisateur stockées dans un cluster Kubernetes.

Auparavant, Amazon EKS prenait en charge l'activation du chiffrement des enveloppes à l'aide de clés KMS uniquement lors de la création du cluster. Vous pouvez désormais activer le chiffrement d'enveloppe pour les clusters Amazon EKS à tout moment.

Pour en savoir plus sur l'utilisation du support du fournisseur de chiffrement EKS, consultez un defense-in-depth article sur le blog consacré aux conteneurs AWS.

Création d'un cluster avec le chiffrement KMS activé

```
# 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
```

Activation du chiffrement KMS sur un cluster existant

Pour activer le chiffrement KMS sur un cluster sur lequel il n'est pas encore activé, exécutez

```
eksctl utils enable-secrets-encryption -f kms-cluster.yaml
```

ou sans fichier de configuration :

```
eksctl utils enable-secrets-encryption --cluster=kms-cluster --key-arn=arn:aws:kms:us-
west-2:<account>:key/<key> --region=<region>
```

En plus d'activer le chiffrement KMS sur le cluster EKS, eksctl rechiffre également tous les secrets Kubernetes existants à l'aide de la nouvelle clé KMS en les mettant à jour avec l'annotation. `eksctl.io/kms-encryption-timestamp` Ce comportement peut être désactivé en passant`--encrypt-existing-secrets=false`, comme dans :

```
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>
```

Si le chiffrement KMS est déjà activé sur un cluster, eksctl procédera au rechiffrement de tous les secrets existants.

> **Note**
>
> Une fois le chiffrement KMS activé, il ne peut pas être désactivé ou mis à jour pour utiliser une autre clé KMS.

Résolution des problèmes

Cette rubrique contient des instructions sur la façon de résoudre les erreurs courantes avec Eksctl.

échec de la création de la pile

Vous pouvez utiliser l'`--cfn-disable-rollback`indicateur pour empêcher Cloudformation de supprimer les piles défaillantes afin de faciliter le débogage.

l'ID de sous-réseau « subnet-11111111 » n'est pas identique à « subnet-22222222 »

Étant donné un fichier de configuration spécifiant les sous-réseaux pour un VPC comme celui-ci :

```
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: []
```

Une erreur `subnet ID "subnet-11111111" is not the same as "subnet-22222222"` signifie que les sous-réseaux spécifiés ne sont pas placés dans la bonne zone de disponibilité. Vérifiez dans la console AWS quel est le bon ID de sous-réseau pour chaque zone de disponibilité.

Dans cet exemple, la configuration correcte pour le VPC serait la suivante :

```
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}
```

Problèmes de suppression

Si votre suppression ne fonctionne pas ou si vous oubliez d'ajouter `--wait` la suppression, vous devrez peut-être utiliser les autres outils d'Amazon pour supprimer les piles Cloudformation. Cela peut être accompli via l'interface graphique ou avec l'interface de ligne de commande AWS.

Les journaux kubectl et l'exécution de kubectl échouent avec une erreur d'autorisation

Si vos nœuds sont déployés dans un sous-réseau privé `kubectl logs` et/ou `kubectl run` échouent avec une erreur telle que la suivante :

```
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)
```

Ensuite, vous devrez peut-être définir enableDnsHostnames. Vous trouverez plus de détails dans ce numéro.

Annonces

Cette rubrique couvre les annonces passées concernant les nouvelles fonctionnalités d'Eksctl.

Groupes de nœuds gérés par défaut

Depuis eksctl v0.58.0, eksctl crée des groupes de nœuds gérés par défaut lorsqu'aucun fichier n'est spécifié pour et. `ClusterConfig` `eksctl create cluster` `eksctl create nodegroup` Pour créer un groupe de nœuds autogéré, passez. `--managed=false` Cela peut interrompre les scripts n'utilisant pas de fichier de configuration si une fonctionnalité non prise en charge dans les groupes de nœuds gérés, par exemple les groupes de nœuds Windows, est utilisée. Pour résoudre ce problème`--managed=false`, transmettez ou spécifiez votre configuration de groupe de nœuds dans un `ClusterConfig` fichier à l'aide du `nodeGroups` champ qui crée un groupe de nœuds autogéré.

Nodegroup Bootstrap Override pour la personnalisation AMIs

Ce changement a été annoncé dans le numéro Breaking : overrideBootstrapCommand soon.... Maintenant, c'est devenu réalité dans ce PR. Veuillez lire attentivement le numéro ci-joint pour savoir pourquoi nous avons décidé de ne plus prendre en charge les scripts personnalisés AMIs sans scripts d'amorçage ou avec des scripts de démarrage partiels.

Nous fournissons toujours une aide ! J'espère que migrer ne sera pas si pénible. `eksctl`fournit toujours un script qui, une fois obtenu, exportera quelques propriétés et paramètres d'environnement utiles. Ce script se trouve ici.

Les propriétés d'environnement suivantes seront à votre disposition :

```
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
```

Le minimum à utiliser lors du remplacement pour `eksctl` ne pas échouer, ce sont les étiquettes ! `eksctl`repose sur un ensemble spécifique d'étiquettes à placer sur le nœud, afin qu'il puisse les trouver. Lors de la définition de la dérogation, veuillez fournir cette commande de dérogation minimale :

```
    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}"
```

Pour les groupes de nœuds qui n'ont pas d'accès Internet sortant, vous devez fournir `--apiserver-endpoint` et `--b64-cluster-ca` au script bootstrap comme suit :

```
    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}
```

Notez le paramètre `--node-labels`. Si cela n'est pas défini, le nœud rejoindra le cluster, mais il `eksctl` arrivera finalement à expiration lors de la dernière étape, lorsqu'il attend que les nœuds le soient`Ready`. Il effectue une recherche dans Kubernetes pour les nœuds portant le label. `alpha.eksctl.io/nodegroup-name=<cluster-name>` Cela n'est vrai que pour les groupes de nœuds non gérés. Pour géré, une étiquette différente est utilisée.

S'il est possible de passer à des groupes de nœuds gérés pour éviter cette surcharge, le moment est venu de le faire. Facilite beaucoup toutes les remplacements.

Les traductions sont fournies par des outils de traduction automatique. En cas de conflit entre le contenu d'une traduction et celui de la version originale en anglais, la version anglaise prévaudra.