Amazon EKS

Guia do usuário do Eksctl

Guia do usuário do Eksctl: Amazon EKS

Table of Contents

O que é Eksctl?

eksctl é uma ferramenta utilitária de linha de comando que automatiza e simplifica o processo de criação, gerenciamento e operação de clusters do Amazon Elastic Kubernetes Service (Amazon EKS). Escrito em Go, o eksctl fornece uma sintaxe declarativa por meio de configurações YAML e comandos CLI para lidar com operações complexas de cluster EKS que, de outra forma, exigiriam várias etapas manuais em diferentes serviços da AWS.

O eksctl é particularmente valioso para DevOps engenheiros, equipes de plataforma e administradores do Kubernetes que precisam implantar e gerenciar consistentemente clusters EKS em grande escala. É especialmente útil para organizações que estão fazendo a transição do Kubernetes autogerenciado para o EKS ou para aquelas que implementam práticas de infraestrutura como código (IaC), pois pode ser integrado aos pipelines e fluxos de trabalho de automação existentes. CI/CD A ferramenta abstrai muitas das interações complexas entre os serviços da AWS necessárias para a configuração do cluster EKS, como configuração de VPC, criação de funções do IAM e gerenciamento de grupos de segurança.

Os principais recursos do eksctl incluem a capacidade de criar clusters EKS totalmente funcionais com um único comando, suporte para configurações de rede personalizadas, gerenciamento automatizado de grupos de nós e GitOps integração de fluxo de trabalho. A ferramenta gerencia atualizações de clusters, escala grupos de nós e gerencia o gerenciamento de complementos por meio de uma abordagem declarativa. O eksctl também fornece recursos avançados, como configuração de perfil do Fargate, personalização gerenciada de grupos de nós e integração de instâncias spot, mantendo a compatibilidade com outras ferramentas e serviços da AWS por meio da integração nativa do SDK da AWS.

Recursos

Os recursos atualmente implementados são:

- Crie, obtenha, liste e exclua clusters
- Crie, drene e exclua grupos de nós
- Dimensionar um grupo de nós
- Atualizar um cluster
- Use personalizado AMIs

- Configurar redes VPC
- Configurar o acesso aos endpoints da API
- Support para grupos de nós de GPU
- Instâncias spot e instâncias mistas
- Políticas complementares e de gerenciamento do IAM
- Listar pilhas do cluster Cloudformation
- Instale coredns
- Grave o arquivo kubeconfig para um cluster

Perguntas frequentes sobre o Eksctl

Geral

Posso usar **`eksctl`** para gerenciar clusters que não foram criados pelo**`eksctl`**?

Sim! A partir da versão, `0.40.0` você `eksctl` pode executar em qualquer cluster, tenha ele sido criado `eksctl` ou não. Para obter mais informações, consulte the section called “Clusters não criados pelo eksctl”.

Grupos de nós

Como posso alterar o tipo de instância do meu nodegroup?

Do ponto de vista de`eksctl`, os grupos de nós são imutáveis. Isso significa que, uma vez criado, a única coisa que `eksctl` você pode fazer é aumentar ou diminuir o grupo de nós.

Para alterar o tipo de instância, crie um novo grupo de nós com o tipo de instância desejado e, em seguida, drene-o para que as cargas de trabalho sejam transferidas para o novo. Depois que essa etapa for concluída, você poderá excluir o antigo grupo de nós.

Como posso ver os dados de usuário gerados para um grupo de nós?

Primeiro, você precisará do nome da pilha do Cloudformation que gerencia o grupo de nós:

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

Você verá um nome semelhante `eksctl-CLUSTER_NAME-nodegroup-NODEGROUP_NAME` a.

Você pode executar o seguinte para obter os dados do usuário. Observe a linha final que decodifica a partir da base64 e descompacta os dados compactados em gzip.

```
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

Como faço para configurar o ingresso com**`eksctl`**?

Recomendamos usar o AWS Load Balancer Controller. A documentação sobre como implantar o controlador em seu cluster, bem como sobre como migrar do antigo ALB Ingress Controller, pode ser encontrada aqui.

Para o Nginx Ingress Controller, a configuração seria a mesma de qualquer outro cluster Kubernetes.

Kubectl

Estou usando um proxy HTTPS e a validação do certificado de cluster falha. Como posso usar o sistema CAs?

Defina a variável de ambiente `KUBECONFIG_USE_SYSTEM_CA` para `kubeconfig` respeitar as autoridades de certificação do sistema.

Corrida a seco

O recurso dry-run permite que você inspecione e altere as instâncias correspondentes ao seletor de instâncias antes de continuar com a criação de um grupo de nós.

Quando `eksctl create cluster` é chamado com as opções do seletor de instância e`--dry-run`, eksctl produzirá um ClusterConfig arquivo contendo um grupo de nós representando as opções

da CLI e os tipos de instância definidos para as instâncias que correspondem aos critérios de recurso do seletor de instâncias.

```
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
```

O gerado ClusterConfig pode então ser passado para`eksctl create cluster`:

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

Quando um ClusterConfig arquivo é passado com`--dry-run`, eksctl exibirá um ClusterConfig arquivo contendo os valores definidos no arquivo.

Opções únicas em eksctl

Há certas opções únicas que não podem ser representadas no `ClusterConfig` arquivo, por exemplo,`--install-vpc-controllers`.

Espera-se que:

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

seguido por:

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

seria equivalente a executar o primeiro comando sem`--dry-run`.

Portanto, eksctl não permite a passagem de opções que não podem ser representadas no arquivo de configuração quando são passadas. `--dry-run`

> **Important**
>
> Se você precisar passar um perfil da AWS, defina a variável de `AWS_PROFILE` ambiente, em vez de passar a opção `--profile` CLI.

Tutorial

Este tópico mostra como instalar e configurar o eksctl e, em seguida, usá-lo para criar um cluster Amazon EKS.

Etapa 1: instalar o eksctl

Conclua as etapas a seguir para baixar e instalar a versão mais recente do eksctl em seu dispositivo Linux ou macOS:

Para instalar o eksctl com o Homebrew

1. (Pré-requisito) Instale o Homebrew.
2. Adicione o toque da AWS:

```
brew tap aws/tap
```

3. Instalar o eksctl

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

Antes de usar o eksctl, conclua estas etapas de configuração:

1. Pré-requisitos de instalação:
   - Instale a AWS CLI versão 2.x ou posterior.
   - Instale o kubectl usando o Homebrew:

```
brew install kubernetes-cli
```

2. Configure as credenciais da AWS em seu ambiente:

```
aws configure
```

3. Verifique a configuração do AWS CLI:

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

Etapa 2: criar o arquivo de configuração do cluster

Crie um arquivo de configuração de cluster usando estas etapas:

1. Crie um novo arquivo chamado`cluster.yaml`:

```
touch cluster.yaml
```

2. Adicione a seguinte configuração básica de cluster:

```
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. Personalize a configuração:
   - Atualize o `region` para corresponder à sua região da AWS desejada.
   - Modifique o `instanceType` com base em seus requisitos de carga de trabalho.
   - Ajuste o`desiredCapacity`,`minSize`, e de `maxSize` acordo com suas necessidades.
4. Valide o arquivo de configuração:

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

Etapa 3: criar um cluster

Siga estas etapas para criar seu cluster EKS:

1. Crie o cluster usando o arquivo de configuração:

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

2. Aguarde a criação do cluster (isso normalmente leva de 15 a 20 minutos).
3. Verifique a criação do cluster:

```
eksctl get cluster
```

4. Configure o kubectl para usar seu novo cluster:

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

5. Verifique a conectividade do cluster:

```
kubectl get nodes
```

Seu cluster agora está pronto para uso.

Opcional: Excluir cluster

Lembre-se de excluir o cluster quando terminar para evitar cobranças desnecessárias:

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

> **Note**
>
> A criação de clusters pode incorrer em cobranças da AWS. Certifique-se de revisar os preços do Amazon EKS antes de criar um cluster.

Próximas etapas

- Configurar o Kubectl para se conectar ao cluster
- Implantar uma aplicação de amostra

Opções de instalação para Eksctl

`eksctl`está disponível para instalação a partir das versões oficiais, conforme descrito abaixo. Recomendamos que você instale somente `eksctl` a partir das GitHub versões oficiais. Você pode optar por usar um instalador terceirizado, mas esteja ciente de que a AWS não mantém nem oferece suporte a esses métodos de instalação. Use-os a seu próprio critério.

Pré-requisito

Você precisará ter as credenciais da API da AWS configuradas. O que funciona para o AWS CLI ou qualquer outra ferramenta (kops, Terraform etc.) deve ser suficiente. Você pode usar variáveis de `~/.aws/credentials`arquivo ou ambiente. Para obter mais informações, consulte a referência da AWS CLI.

Você também precisará do comando AWS IAM Authenticator for Kubernetes (um ou `aws eks get-token` (disponível na versão 1.16.156 `aws-iam-authenticator` ou superior do AWS CLI) em seu. `PATH`

A conta do IAM usada para a criação do cluster EKS deve ter esses níveis mínimos de acesso.

Serviço da AWS	Nível de acesso
CloudFormation	Acesso total
EC2	Completo: Marcação limitada: listar, ler, escrever
EC2 Auto Scaling	Limitado: listar, escrever
EKS	Acesso total
IAM	Limitado: lista, leitura, gravação, gerenciam ento de permissões
Systems Manager	Limitado: Listar, Leir

Para Unix

Para baixar a versão mais recente, execute:

```
# 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
```

No Windows

Download direto (versão mais recente):

- AMD64/x86_64
- ARMv6
- ARMv7
- ARM64

Certifique-se de descompactar o arquivo em uma pasta na PATH variável.

Opcionalmente, verifique a soma de verificação:

1. Baixe o arquivo de soma de verificação: mais recente
2. Use o prompt de comando para comparar manualmente `CertUtil` a saída com o arquivo de soma de verificação baixado.

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

3. Usando PowerShell para automatizar a verificação usando o `-eq` operador para obter um `False` resultado `True` ou:

```
# 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]
```

Usando o 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
```

O `eksctl` executável é colocado em`$HOME/bin`, que `$PATH` vem do Git Bash.

Homebrew

Você pode usar o Homebrew para instalar software no macOS e no Linux.

A AWS mantém uma torneira Homebrew, incluindo eksctl.

Para obter mais informações sobre o Homebrew tap, consulte o projeto no Github e a fórmula do Homebrew para eksctl.

Para instalar o eksctl com o Homebrew

1. (Pré-requisito) Instale o Homebrew
2. Adicione a torneira da AWS

```
brew tap aws/tap
```

3. Instalar o eksctl

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

Docker

Para cada versão e RC, uma imagem de contêiner é enviada para o repositório ECR. `public.ecr.aws/eksctl/eksctl` Saiba mais sobre o uso da Galeria Pública ECR - eksctl. Por exemplo,

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

Conclusão do Shell

Bash

Para habilitar a conclusão do bash, execute o seguinte ou coloque-o em `~/.bashrc` ou`~/.profile`:

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

Cinzas

Para completar o zsh, execute:

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

e insira o seguinte`~/.zshrc`:

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

Observe que, se você não estiver executando uma distribuição como oh-my-zsh, talvez seja necessário primeiro ativar o preenchimento automático (e inseri-lo `~/.zshrc` para torná-la persistente):

```
autoload -U compinit
compinit
```

Peixe

Os comandos abaixo podem ser usados para o preenchimento automático de fish:

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

PowerShell

O comando abaixo pode ser consultado para configurá-lo. Observe que o caminho pode ser diferente dependendo das configurações do sistema.

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

Atualizações

> **Important**
>
> Se você instalar o eksctl baixando-o diretamente (sem usar um gerenciador de pacotes), precisará atualizá-lo manualmente.

Clusters

Este capítulo aborda a criação e configuração de clusters EKS usando eksctl. Também inclui complementos e o EKS Auto Mode.

Tópicos:

- the section called “Entradas de acesso ao EKS”
  - Simplifique o gerenciamento do Kubernetes RBAC substituindo aws-auth por entradas de acesso EKS ConfigMap
  - Migre os mapeamentos de identidade do IAM existentes do ConfigMap aws-auth para acessar as entradas
  - Configure os modos de autenticação do cluster e controle as permissões de administrador do criador do cluster
- the section called “Atualizações de complementos padrão”
  - Mantenha os clusters seguros atualizando os complementos padrão do EKS em clusters mais antigos
- the section called “Complementos”
  - Automatize as tarefas rotineiras de instalação, atualização e remoção de complementos.
  - Os complementos do Amazon EKS incluem complementos da AWS, complementos da comunidade de código aberto e complementos do marketplace.
- the section called “Modo Automático do EKS”
  - Reduza a sobrecarga operacional permitindo que a AWS gerencie sua infraestrutura EKS
  - Configure grupos de nós personalizados em vez dos pools padrão de uso geral e do sistema
  - Converta clusters EKS existentes para usar o Modo Automático
- the section called “CloudWatch registro”
  - Solucione problemas de cluster habilitando registros para componentes específicos do plano de controle EKS
  - Configurar períodos de retenção de registros para registros de cluster EKS
  - Modifique as configurações de registro de cluster existentes usando os comandos eksctl
- the section called “Atualizações de cluster”

  - Mantenha a segurança e a estabilidade atualizando com segurança as versões do plano de controle EKS
  - Implemente atualizações em grupos de nós substituindo grupos antigos por novos
  - Atualizar complementos de cluster padrão
- the section called “Criar e gerenciar clusters”
  - Comece rapidamente com clusters EKS básicos usando grupos de nós gerenciados padrão
  - Crie clusters personalizados usando arquivos de configuração com configurações específicas
  - Implante clusters em redes privadas existentes VPCs e políticas personalizadas de IAM
- the section called “Configurar o kubelet”
  - Evite a falta de recursos do nó configurando as reservas do kubelet e do daemon do sistema
  - Personalize os limites de despejo para garantir a disponibilidade da memória e do sistema de arquivos
  - Ative ou desative portas de recursos específicas do kubelet em grupos de nós
- the section called “Conector EKS”
  - Centralize o gerenciamento de implantações híbridas do Kubernetes por meio do EKS Console
  - Configurar funções e permissões do IAM para acesso externo ao cluster
  - Remova clusters externos e limpe os recursos associados da AWS
- the section called “Cluster totalmente privado EKS”
  - Atenda aos requisitos de segurança com clusters EKS totalmente privados sem acesso de saída à Internet
  - Configure o acesso privado aos serviços da AWS por meio de VPC endpoints
  - Crie e gerencie grupos de nós privados com configurações de rede explícitas
- the section called “Suporte do Karpenter”
  - Automatize o provisionamento de nós
  - Crie configurações personalizadas do provisionador Karpenter
  - Configure o Karpenter com tratamento de interrupções de instâncias pontuais
- the section called “Amazon EMR”
  - Crie um mapeamento de identidade do IAM entre o EMR e o cluster EKS
- the section called “Suporte do EKS Fargate”
  - Defina perfis Fargate personalizados para agendamento de pods
  - Gerencie perfis do Fargate por meio de atualizações de criação e configuração

- the section called “Clusters não criados pelo eksctl”
  - Padronize o gerenciamento de clusters criados fora do eksctl
  - Use comandos eksctl em clusters não eksctl existentes
- the section called “Habilitar a mudança de zona”
  - Melhore a disponibilidade dos aplicativos habilitando recursos rápidos de failover de zona
  - Configure a mudança zonal em novas implantações de cluster EKS
  - Habilite recursos de mudança zonal em clusters EKS existentes

Criar e gerenciar clusters

Este tópico aborda como criar e excluir clusters EKS usando o Eksctl. Você pode criar clusters com um comando da CLI ou criando um arquivo YAML de configuração de cluster.

Criação de um cluster simples

Crie um cluster simples com o seguinte comando:

```
eksctl create cluster
```

Isso criará um cluster EKS em sua região padrão (conforme especificado pela configuração do AWS CLI) com um grupo de nós gerenciado contendo dois nós m5.large.

O eksctl agora cria um grupo de nós gerenciado por padrão quando um arquivo de configuração não é usado. Para criar um grupo de nós autogerenciado, passe `--managed=false` para ou. `eksctl create cluster` `eksctl create nodegroup`

Considerações

- Ao criar clusters em`us-east-1`, você pode encontrar um`UnsupportedAavailabilityZoneException`. Se isso acontecer, copie as zonas sugeridas e passe a `--zones` bandeira, por exemplo:`eksctl create cluster --region=us-east-1 --zones=us-east-1a,us-east-1b,us-east-1d`. Esse problema pode ocorrer em outras regiões, mas é menos comum. Na maioria dos casos, você não precisará usar a `--zone` bandeira.

Crie um cluster usando o arquivo de configuração

Você pode criar um cluster usando um arquivo de configuração em vez de sinalizadores.

Primeiro, crie `cluster.yaml` o arquivo:

```
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
```

Em seguida, execute este comando:

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

Isso criará um cluster conforme descrito.

Se precisar usar uma VPC existente, você pode usar um arquivo de configuração como este:

```
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
```

O nome do cluster ou do grupo de nós deve conter somente caracteres alfanuméricos (com distinção entre maiúsculas e minúsculas) e hífens. Ele deve começar com um caractere alfabético e não pode exceder 128 caracteres, ou você receberá um erro de validação. Para obter mais informações, consulte Criar uma pilha a partir do CloudFormation console no guia do usuário do AWS CLoud Formation.

Atualize o kubeconfig para o novo cluster

Depois que o cluster for criado, a configuração apropriada do kubernetes será adicionada ao seu arquivo kubeconfig. Esse é o arquivo que você configurou na variável de ambiente `KUBECONFIG` ou `~/.kube/config` por padrão. O caminho para o arquivo kubeconfig pode ser substituído usando a sinalização. `--kubeconfig`

Outros sinalizadores que podem alterar a forma como o arquivo kubeconfig é gravado:

sinalizador	type	use	valor padrão
--kubeconfig	string	caminho para escrever kubeconfig (incompatível com --auto-kubeconfig)	$KUBECONFIG ou ~/.kube/config
--set-kubeconfig-context	bool	se verdadeiro, o contexto atual será	true

sinalizador	type	use	valor padrão
		definido no kubeconfig; se um contexto já estiver definido, ele será sobrescrito	
--configuração automática do kube	bool	salve o arquivo kubeconfig pelo nome do cluster	true
--write-kubeconfig	bool	alternar a gravação do kubeconfig	true

Excluir cluster

Para excluir esse cluster, execute:

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

> **Warning**
>
> Use o `--wait` sinalizador com operações de exclusão para garantir que os erros de exclusão sejam reportados corretamente.

Sem o `--wait` sinalizador, o eksctl emitirá apenas uma operação de exclusão na CloudFormation pilha do cluster e não aguardará sua exclusão. Em alguns casos, os recursos da AWS que usam o cluster ou sua VPC podem fazer com que a exclusão do cluster falhe. Se sua exclusão falhar ou você esquecer o sinalizador de espera, talvez seja necessário acessar a CloudFormation GUI e excluir as pilhas eks de lá.

> **Warning**
>
> As políticas de PDB podem bloquear a remoção de nós durante a exclusão do cluster.

Ao excluir um cluster com grupos de nós, as políticas do Pod Disruption Budget (PDB) podem impedir que os nós sejam removidos com sucesso. Por exemplo, os clusters `aws-ebs-csi-driver` instalados normalmente têm dois pods com uma política de PDB que permite que apenas um pod fique indisponível por vez, tornando o outro pod impossível de ser removido durante a exclusão. Para excluir com êxito o cluster nesses cenários, use o `disable-nodegroup-eviction` sinalizador para ignorar as verificações de política do PDB:

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

Consulte o `examples/`diretório no GitHub repositório eksctl para ver mais exemplos de arquivos de configuração.

Corrida a seco

O recurso dry-run permite gerar um ClusterConfig arquivo que ignora a criação do cluster e gera um ClusterConfig arquivo que representa as opções CLI fornecidas e contém os valores padrão definidos por eksctl.

Mais informações podem ser encontradas na página Dry Run.

Modo Automático do EKS

O eksctl é compatível com o EKS Auto Mode, um recurso que estende o gerenciamento de clusters Kubernetes pela AWS além do próprio cluster, para permitir que a AWS também configure e gerencie a infraestrutura que permite a operação tranquila de suas cargas de trabalho. Isso permite que você delegue as principais decisões de infraestrutura e aproveite a experiência da AWS para day-to-day operações. A infraestrutura de cluster gerenciada pela AWS inclui muitos recursos do Kubernetes como componentes principais, em vez de complementos, como escalonamento automático de computação, redes de pods e serviços, balanceamento de carga de aplicativos, DNS de cluster, armazenamento em blocos e suporte a GPU.

Criação de um cluster EKS com o modo automático ativado

`eksctl`adicionou um novo `autoModeConfig` campo para ativar e configurar o Modo Automático. A forma do `autoModeConfig` campo é

```
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
```

Se `autoModeConfig.enabled` for verdade, o eksctl cria um cluster EKS passando `computeConfig.enabled: truekubernetesNetworkConfig.elasticLoadBalancing.enabled: true`, e `storageConfig.blockStorage.enabled: true` para a API EKS, permitindo o gerenciamento de componentes do plano de dados, como computação, armazenamento e rede.

Para criar um cluster EKS com o Modo Automático ativado, defina`autoModeConfig.enabled: true`, como em

```
# 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 cria uma função de nó para ser usada em nós iniciados pelo Modo Automático. O eksctl também cria os pools de nós e. `general-purpose system` Para desativar a criação dos pools de nós padrão, por exemplo, para configurar seus próprios pools de nós que usam um conjunto diferente de sub-redes, defina`nodePools: []`, como em

```
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.
```

Atualizando um cluster EKS para usar o modo automático

Para atualizar um cluster EKS existente para usar o Modo Automático, execute

```
# 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**
>
> Se o cluster foi criado pelo eksctl e usa sub-redes públicas como sub-redes de cluster, o Modo Automático iniciará nós em sub-redes públicas. Para usar sub-redes privadas para nós de trabalho iniciados pelo Modo Automático, atualize o cluster para usar sub-redes privadas.

Desativando o modo automático

Para desativar o modo automático, defina `autoModeConfig.enabled: false` e execute

```
# 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
```

Mais informações

- Modo Automático do EKS

Entradas de acesso ao EKS

Você pode usar o eksctl para gerenciar as entradas de acesso ao EKS. Use entradas de acesso para conceder permissões do Kubernetes às identidades do AWS IAM. Por exemplo, você pode conceder permissão ao papel de desenvolvedor para ler recursos do Kubernetes em um cluster.

Este tópico aborda como usar o eksctl para gerenciar entradas de acesso. Para obter informações gerais sobre entradas de acesso, consulte Conceder aos usuários do IAM acesso ao Kubernetes com entradas de acesso do EKS.

Você pode anexar políticas de acesso ao Kubernetes definidas pela AWS ou associar uma identidade do IAM a um grupo do Kubernetes.

Para obter mais informações sobre as políticas predefinidas disponíveis, consulte Associar políticas de acesso às entradas de acesso.

Se você precisar definir as políticas do Kubernetes do cliente, associe a identidade do IAM a um grupo do Kubernetes e conceda permissões a esse grupo.

Modo de autenticação de cluster

Você só pode usar entradas de acesso se o modo de autenticação do cluster permitir.

Para obter mais informações, consulte Definir o modo de autenticação de cluster

Definir o modo de autenticação com um arquivo YAML

`eksctl`adicionou um novo `accessConfig.authenticationMode` campo abaixo ClusterConfig, que pode ser definido como um dos três valores a seguir:

- `CONFIG_MAP`- padrão na API EKS - somente `aws-auth` ConfigMap será usado

- `API`- somente a API de entradas de acesso será usada
- `API_AND_CONFIG_MAP`- padrão em `eksctl` - ambas `aws-auth` ConfigMap e a API de entradas de acesso podem ser usadas

Defina o modo de autenticação no ClusterConfig YAML:

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

Atualize o modo de autenticação com um comando

Se você quiser usar entradas de acesso em um cluster já existente, não criado pelo eksctl, onde a `CONFIG_MAP` opção é usada, o usuário precisará primeiro definir como. `authenticationMode` `API_AND_CONFIG_MAP` Para isso, `eksctl` introduziu um novo comando para atualizar o modo de autenticação do cluster, que funciona tanto com sinalizadores CLI, por exemplo

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

Acesse os recursos de entrada

As entradas de acesso têm um tipo, como `STANDARD` ou`EC2_LINUX`. O tipo depende de como você está usando a entrada de acesso.

- O `standard` tipo é para conceder permissões do Kubernetes a usuários e funções do IAM.
  - Por exemplo, você pode visualizar os recursos do Kubernetes no console da AWS anexando uma política de acesso à função ou ao usuário que você usa para acessar o console.
- Os `EC2_WINDOWS` tipos `EC2_LINUX` e são para conceder permissões do Kubernetes às instâncias do EC2. As instâncias usam essas permissões para ingressar no cluster.

Para obter mais informações sobre os tipos de entradas de acesso, consulte Criar entradas de acesso

Entidades do IAM

Você pode usar entradas de acesso para conceder permissões do Kubernetes às identidades do IAM, como usuários do IAM e funções do IAM.

Use o `accessConfig.accessEntries` campo para associar o ARN de um recurso do IAM a uma API EKS de entradas de acesso. Por exemplo:

```
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
```

Além de associar as políticas do EKS, também é possível especificar os grupos do Kubernetes aos quais uma entidade do IAM pertence, concedendo permissões via RBAC.

Grupos de nós gerenciados e Fargate

A integração com entradas de acesso para esses recursos será alcançada nos bastidores, pela API EKS. Grupos de nós gerenciados e pods Fargate recém-criados criarão entradas de acesso à API, em vez de usar recursos RBAC pré-carregados. Os grupos de nós e os pods do Fargate existentes não serão alterados e continuarão a depender das entradas no mapa de configuração aws-auth.

Grupos de nós autogerenciados

Cada entrada de acesso tem um tipo. Para autorizar grupos de nós autogerenciados, `eksctl` criará uma entrada de acesso exclusiva para cada grupo de nós com o ARN principal definido como o ARN da função do nó e o tipo definido como um ou dependendo do grupo de nós AmiFamily. `EC2_LINUX` `EC2_WINDOWS`

Ao criar suas próprias entradas de acesso, você também pode especificar `EC2_LINUX` (para uma função do IAM usada com nós autogerenciados do Linux ou Bottlerocket), `EC2_WINDOWS` (para funções do IAM usadas com nós autogerenciados do Windows), `FARGATE_LINUX` (para funções do IAM usadas com o AWS Fargate (Fargate)) ou como um tipo. `STANDARD` Se você não especificar um tipo, o tipo padrão será definido como`STANDARD`.

> **Note**
>
> Ao excluir um grupo de nós criado com um preexistente`instanceRoleARN`, é responsabilidade do usuário excluir a entrada de acesso correspondente quando não houver mais grupos de nós associados a ela. Isso ocorre porque o eksctl não tenta descobrir se uma entrada de acesso ainda está em uso por grupos de nós autogerenciados criados por não-eksctl, pois é um processo complicado.

Criar entrada de acesso

Isso pode ser feito de duas maneiras diferentes, seja durante a criação do cluster, especificando as entradas de acesso desejadas como parte do arquivo de configuração e executando:

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

OU após a criação do cluster, executando:

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

Para ver um exemplo de arquivo de configuração para criar entradas de acesso, consulte 40-access-entries.yaml no repositório eksctl. GitHub

Obtenha a entrada de acesso

O usuário pode recuperar todas as entradas de acesso associadas a um determinado cluster executando uma das seguintes opções:

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

OU

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

Como alternativa, para recuperar apenas a entrada de acesso correspondente a uma determinada entidade do IAM, deve-se usar o `--principal-arn` sinalizador. por exemplo

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

Excluir entrada de acesso

Para excluir uma única entrada de acesso por vez, use:

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

Para excluir várias entradas de acesso, use a `--config-file` bandeira e especifique todas as `principalARN's` correspondentes às entradas de acesso, no `accessEntry` campo de nível superior, por exemplo

```
...
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
```

Migre do aws-auth ConfigMap

O usuário pode migrar suas identidades IAM existentes do `aws-auth` configmap para acessar as entradas executando o seguinte:

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

Quando o `--target-authentication-mode` sinalizador é definido como`API`, o modo de autenticação é alterado para o `API` modo (ignorado se já estiver no `API` modo), os mapeamentos de identidade do IAM serão migrados para as entradas de acesso e o `aws-auth` configmap será excluído do cluster.

Quando o `--target-authentication-mode` sinalizador é definido como`API_AND_CONFIG_MAP`, o modo de autenticação é alterado para o `API_AND_CONFIG_MAP` modo (ignorado se já estiver no `API_AND_CONFIG_MAP` modo), os mapeamentos de identidade do IAM serão migrados para as entradas de acesso, mas o configmap é preservado. `aws-auth`

> **Note**
>
> Quando o `--target-authentication-mode` sinalizador estiver definido como`API`, esse comando não atualizará o modo de autenticação para o `API` modo se o `aws-auth` configmap tiver uma das restrições abaixo.

- Há um mapeamento de identidade no nível da conta.
- Um ou mais Roles/Users são mapeados para os grupos kubernetes que começam com o prefixo `system:` (exceto para grupos específicos do EKS, ou seja`system:masters`,, etc). `system:bootstrappers` `system:nodes`
- Um ou mais mapeamentos de identidade do IAM são para uma [Função vinculada ao serviço] (link: IAM/latest/UserGuide/using - service-linked-roles .html).

Desative as permissões de administrador do criador do cluster

`eksctl`adicionou um novo campo `accessConfig.bootstrapClusterCreatorAdminPermissions: boolean` que, quando

definido como false, desativa a concessão de permissões de administrador de cluster à identidade do IAM que cria o cluster. ou seja

adicione a opção ao arquivo de configuração:

```
accessConfig:
  bootstrapClusterCreatorAdminPermissions: false
```

e execute:

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

Clusters não criados pelo eksctl

Você pode executar `eksctl` comandos em clusters que não foram criados pelo`eksctl`.

> **Note**
>
> O Eksctl só pode suportar clusters sem proprietário com nomes compatíveis com a AWS. CloudFormation Qualquer nome de cluster que não corresponda a isso falhará na verificação de validação CloudFormation da API.

Comandos compatíveis

Os comandos a seguir podem ser usados em clusters criados por qualquer outro meio que não seja`eksctl`. Os comandos, sinalizadores e opções do arquivo de configuração podem ser usados exatamente da mesma maneira.

Se perdemos alguma funcionalidade, entre em contato conosco.

- ✓ Crie:
  - ✓ `eksctl create nodegroup` (veja a nota abaixo)
  - ✓ `eksctl create fargateprofile`
  - ✓ `eksctl create iamserviceaccount`
  - ✓ `eksctl create iamidentitymapping`
- ✓ Obtenha:
  - ✓ `eksctl get clusters/cluster`

- ✓ `eksctl get fargateprofile`
- ✓ `eksctl get nodegroup`
- ✓ `eksctl get labels`

✓ Excluir:

- ✓ `eksctl delete cluster`
- ✓ `eksctl delete nodegroup`
- ✓ `eksctl delete fargateprofile`
- ✓ `eksctl delete iamserviceaccount`
- ✓ `eksctl delete iamidentitymapping`

✓ Atualização:

- ✓ `eksctl upgrade cluster`
- ✓ `eksctl upgrade nodegroup`

✓ Definir/Desativar:

- ✓ `eksctl set labels`
- ✓ `eksctl unset labels`

✓ Escala:

- ✓ `eksctl scale nodegroup`

✓ Drene:

- ✓ `eksctl drain nodegroup`

✓ Habilitar:

- ✓ `eksctl enable profile`
- ✓ `eksctl enable repo`

✓ Utilitários:

- ✓ `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`

Criação de grupos de nós

`eksctl create nodegroup`é o único comando que requer uma entrada específica do usuário.

Como os usuários podem criar seus clusters com qualquer configuração de rede que desejarem, por enquanto, não `eksctl` tentarão recuperar ou adivinhar esses valores. Isso pode mudar no futuro, à medida que aprendermos mais sobre como as pessoas estão usando esse comando em clusters não criados pelo eksctl.

Isso significa que, para criar grupos de nós ou grupos de nós gerenciados em um cluster que não foi criado por`eksctl`, um arquivo de configuração contendo detalhes da VPC deve ser fornecido. No mínimo:

```
---
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"
```

```
...
```

Para obter mais informações sobre as opções de configuração da VPC, consulte Rede.

Registrando clusters não EKS com o EKS Connector

Você pode usar o EKS Connector para visualizar clusters fora da AWS no console EKS. Esse processo requer o registro do cluster com o EKS e a execução do agente EKS Connector no cluster externo do Kubernetes.

`eksctl`simplifica o registro de clusters que não são do EKS criando os recursos necessários da AWS e gerando manifestos do Kubernetes para que o EKS Connector seja aplicado ao cluster externo.

Registrar cluster

Para registrar ou conectar um cluster Kubernetes que não seja do EKS, execute

```
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
```

Esse comando registrará o cluster e gravará três arquivos que contêm os manifestos do Kubernetes para o EKS Connector que devem ser aplicados ao cluster externo antes que o registro expire.

> **Note**
>
> `eks-connector-clusterrole.yaml`get `eks-connector-console-dashboard-full-access-clusterrole.yaml` concedem `list` permissões para recursos do Kubernetes em todos os namespaces para a identidade do IAM de chamada e devem ser editados adequadamente, se necessário, antes de aplicá-los ao cluster. Para configurar um acesso mais restrito, consulte Conceder acesso a um usuário para visualizar um cluster.

Para fornecer uma função do IAM existente para usar no EKS Connector, passe-a via `--role-arn` as em:

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

Se o cluster já existir, eksctl retornará um erro.

Cancelar o registro do cluster

Para cancelar o registro ou desconectar um cluster registrado, execute

```
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
```

Esse comando cancelará o registro do cluster externo e removerá os recursos associados da AWS, mas você precisará remover os recursos Kubernetes do conector EKS do cluster.

Mais informações

- Conector EKS

Personalizando a configuração do kubelet

Os recursos do sistema podem ser reservados por meio da configuração do kubelet. Isso é recomendado porque, no caso de falta de recursos, o kubelet pode não conseguir expulsar os pods

e, eventualmente, transformar o nó em. `NotReady` Para fazer isso, os arquivos de configuração podem incluir o `kubeletExtraConfig` campo que aceita um yaml de formato livre que será incorporado ao. `kubelet.yaml`

Alguns campos no `kubelet.yaml` são definidos por eksctl e, portanto, não podem ser substituídos, como,`address`,`clusterDomain`, `authentication` ou. `authorization` `serverTLSBootstrap`

O exemplo de arquivo de configuração a seguir cria um grupo `300Mi` de nós que reserva `300m` vCPU, memória e armazenamento efêmero para o kubelet; `300m` vCPU`300Mi`, `1Gi` de memória e armazenamento efêmero para daemons do sistema operacional; `1Gi` e aciona a remoção de pods quando há menos de memória disponível ou menos de 10% do sistema de arquivos raiz. `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
```

Neste exemplo, dadas instâncias do tipo `m5a.xlarge` que têm 4 v CPUs e 16 GiB de memória, a `Allocatable` quantidade de CPUs seria 3,4 e 15,4 GiB de memória. É importante saber que

os valores especificados no arquivo de configuração para os campos em `kubeletExtraconfig` substituirão completamente os valores padrão especificados por eksctl. No entanto, omitir um ou mais `kubeReserved` parâmetros fará com que os parâmetros ausentes tenham como padrão valores sensatos com base no tipo de instância aws que está sendo usado.

`kubeReserved`cálculo

Embora geralmente seja recomendável configurar uma instância mista NodeGroup para usar instâncias com a mesma configuração de CPU e RAM, isso não é um requisito estrito. Portanto, o `kubeReserved` cálculo usa a menor instância no `InstanceDistribution.InstanceTypes` campo. Dessa forma, NodeGroups com tipos de instância diferentes, não reservaremos muitos recursos na instância menor. No entanto, isso pode levar a uma reserva muito pequena para o maior tipo de instância.

> **Warning**
>
> Por padrão, é `eksctl` definido`featureGates.RotateKubeletServerCertificate=true`, mas quando o personalizado `featureGates` é fornecido, ele não será definido. Você deve sempre incluir`featureGates.RotateKubeletServerCertificate=true`, a menos que precise desativá-lo.

CloudWatch registro

Este tópico explica como configurar o Amazon CloudWatch Logging para os componentes do plano de controle do seu cluster EKS. CloudWatch o registro fornece visibilidade das operações do plano de controle do cluster, o que é essencial para solucionar problemas, auditar as atividades do cluster e monitorar a integridade dos componentes do Kubernetes.

Ativando CloudWatch o registro

CloudWatch o registro para o plano de controle EKS não está habilitado por padrão devido aos custos de ingestão e armazenamento de dados.

Para habilitar o registro do plano de controle quando o cluster for criado, você precisará definir a **`cloudWatch.clusterLogging.enableTypes`**configuração em seu `ClusterConfig` (veja exemplos abaixo).

Portanto, se você tiver um arquivo de configuração com a **`cloudWatch.clusterLogging.enableTypes`**configuração correta, poderá criar um cluster com`eksctl create cluster --config-file=<path>`.

Se você já criou um cluster, você pode usar`eksctl utils update-cluster-logging`.

> **Note**
>
> esse comando é executado no modo plano por padrão. Você precisará especificar o `--approve` sinalizador para aplicar as alterações ao seu cluster.

Se você estiver usando um arquivo de configuração, execute:

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

Como alternativa, você pode usar sinalizadores da CLI.

Para habilitar todos os tipos de registros, execute:

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

Para ativar `audit` os registros, execute:

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

Para habilitar tudo, exceto `controllerManager` os registros, execute:

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

Se os tipos de `scheduler` log `api` e já estiverem ativados, para desativar `scheduler` e ativar ao `controllerManager` mesmo tempo, execute:

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

Isso deixará `api` e `controllerManager` como os únicos tipos de log habilitados.

Para desativar todos os tipos de registros, execute:

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

Exemplos do `ClusterConfig`

Em um cluster EKS, o `enableTypes` campo abaixo `clusterLogging` pode obter uma lista de valores possíveis para habilitar os diferentes tipos de registros para os componentes do plano de controle.

Os valores possíveis são os seguintes:

- `api`: ativa os registros do servidor da API Kubernetes.
- `audit`: ativa os registros de auditoria do Kubernetes.
- `authenticator`: ativa os registros do autenticador.
- `controllerManager`: ativa os registros do gerenciador do controlador Kubernetes.
- `scheduler`: ativa os registros do agendador do Kubernetes.

Para saber mais, consulte a documentação do EKS.

Desativar todos os registros

Para desativar todos os tipos, use `[]` ou remova a `cloudWatch` seção completamente.

Ativar todos os registros

Você pode habilitar todos os tipos com `"*"` ou`"all"`. Por exemplo:

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

Ativar um ou mais registros

Você pode ativar um subconjunto de tipos listando os tipos que você deseja ativar. Por exemplo:

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

Período de retenção de registros

Por padrão, os registros são armazenados em CloudWatch Registros indefinidamente. Você pode especificar o número de dias durante os quais os registros do plano de controle devem ser retidos em CloudWatch Registros. O exemplo a seguir retém os registros por 7 dias:

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

Exemplo completo

```
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 totalmente privado EKS

O eksctl suporta a criação de clusters totalmente privados que não têm acesso de saída à Internet e têm apenas sub-redes privadas. Os VPC endpoints são usados para permitir o acesso privado aos serviços da AWS.

Este guia descreve como criar um cluster privado sem acesso de saída à Internet.

Criação de um cluster totalmente privado

O único campo obrigatório para criar um cluster totalmente privado é: `privateCluster.enabled`

```
privateCluster:
  enabled: true
```

Após a criação do cluster, os comandos eksctl que precisam acessar o servidor da API Kubernetes precisarão ser executados de dentro da VPC do cluster, de uma VPC emparelhada ou usando algum outro meio, como o AWS Direct Connect. Os comandos eksctl que precisam de acesso ao EKS não APIs funcionarão se estiverem sendo executados de dentro da VPC do cluster. Para corrigir isso, crie um endpoint de interface para o Amazon EKS acessar de forma privada o gerenciamento do Amazon Elastic Kubernetes Service (Amazon APIs EKS) a partir da sua Amazon Virtual Private Cloud (VPC). Em uma versão futura, o eksctl adicionará suporte para criar esse endpoint para que ele não precise ser criado manualmente. Os comandos que precisam acessar a URL do provedor do OpenID Connect precisarão ser executados de fora da VPC do seu cluster depois que você habilitar o AWS para PrivateLink o Amazon EKS.

A criação de grupos de nós gerenciados continuará funcionando, e a criação de grupos de nós autogerenciados funcionará, pois é necessário acessar o servidor de API por meio do endpoint da interface EKS se o comando for executado de dentro da VPC do cluster, de uma VPC emparelhada ou usando algum outro meio, como o AWS Direct Connect.

> **Note**
>
> Os VPC endpoints são cobrados por hora e com base no uso. Mais detalhes sobre preços podem ser encontrados em PrivateLink Preços da AWS

> **Warning**
>
> Não há suporte para clusters totalmente privados no. `eu-south-1`

Configurando o acesso privado a serviços adicionais da AWS

Para permitir que os nós de trabalho acessem os serviços da AWS de forma privada, o eksctl cria endpoints VPC para os seguintes serviços:

- Endpoints de interface para ECR (ambos `ecr.api` `eecr.dkr`) para extrair imagens de contêineres (plug-in AWS CNI etc.)
- Um endpoint de gateway para o S3 extrair as camadas reais da imagem

- Um endpoint de interface para EC2 exigido pela integração `aws-cloud-provider`
- Um endpoint de interface para STS para dar suporte às funções de Fargate e IAM para contas de serviços (IRSA)
- Um endpoint de interface para CloudWatch logging (`logs`) se o CloudWatch registro estiver ativado

Esses endpoints VPC são essenciais para um cluster privado funcional e, como tal, o eksctl não oferece suporte para configurá-los ou desativá-los. No entanto, um cluster pode precisar de acesso privado a outros serviços da AWS (por exemplo, escalonamento automático exigido pelo autoescalador de cluster). Esses serviços podem ser especificados em`privateCluster.additionalEndpointServices`, o que instrui o eksctl a criar um VPC endpoint para cada um deles.

Por exemplo, para permitir acesso privado ao escalonamento automático e CloudWatch ao registro:

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

Os endpoints suportados em `additionalEndpointServices` são`autoscaling`, `cloudformation` e. `logs`

Ignorando criações de endpoints

Se uma VPC já tiver sido criada com os endpoints da AWS necessários configurados e vinculados às sub-redes descritas na documentação do EKS, `eksctl` pode pular a criação deles fornecendo a seguinte opção: `skipEndpointCreation`

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

Essa configuração não pode ser usada junto com `additionalEndpointServices` o. Isso ignorará toda a criação de endpoints. Além disso, essa configuração só é recomendada se a topologia da <# sub-rede do endpoint estiver configurada corretamente. Se os IDs de sub-rede estiverem corretos, o

`vpce` roteamento será configurado com endereços de prefixo, todos os endpoints EKS necessários serão criados e vinculados à VPC fornecida. `eksctl`não alterará nenhum desses recursos.

Grupos de nós

Somente grupos de nós privados (gerenciados e autogerenciados) são suportados em um cluster totalmente privado porque a VPC do cluster é criada sem nenhuma sub-rede pública. O `privateNetworking` campo (`nodeGroup[].privateNetworking` e `managedNodeGroup[]`) deve ser definido explicitamente. É um erro deixar `privateNetworking` sem definição em um cluster totalmente privado.

```
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
```

Acesso ao endpoint do cluster

Um cluster totalmente privado não oferece suporte à modificação `clusterEndpointAccess` durante a criação do cluster. É um erro definir um `clusterEndpoints.publicAccess` ou`clusterEndpoints.privateAccess`, pois um cluster totalmente privado só pode ter acesso privado, e permitir a modificação desses campos pode interromper o cluster.

VPC e sub-redes fornecidas pelo usuário

O eksctl suporta a criação de clusters totalmente privados usando uma VPC e sub-redes preexistentes. Somente sub-redes privadas podem ser especificadas e é um erro especificar sub-redes em. `vpc.subnets.public`

eksctl cria endpoints de VPC na VPC fornecida e modifica as tabelas de rotas para as sub-redes fornecidas. Cada sub-rede deve ter uma tabela de rotas explícita associada a ela porque o eksctl não modifica a tabela de rotas principal.

```
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
```

Gerenciando um cluster totalmente privado

Para que todos os comandos funcionem após a criação do cluster, o eksctl precisará de acesso privado ao endpoint do servidor da API EKS e acesso de saída à Internet (for). `EKS:DescribeCluster` Os comandos que não precisam de acesso ao servidor da API serão suportados se o eksctl tiver acesso de saída à Internet.

Exclusão forçada de um cluster totalmente privado

É provável que ocorram erros ao excluir um cluster totalmente privado por meio do eksctl, pois o eksctl não tem acesso automático a todos os recursos do cluster. `--force`existe para resolver isso: ele forçará a exclusão do cluster e continuará quando ocorrerem erros.

Limitações

Uma limitação da implementação atual é que o eksctl inicialmente cria o cluster com o acesso ao endpoint público e privado habilitado e desativa o acesso ao endpoint público após a conclusão de todas as operações. Isso é necessário porque o eksctl precisa acessar o servidor da API Kubernetes para permitir que nós autogerenciados se juntem ao cluster e ofereçam suporte ao Fargate. GitOps Depois que essas operações forem concluídas, o eksctl alterna o acesso ao endpoint do cluster para somente privado. Essa atualização adicional significa que a criação de um cluster totalmente privado levará mais tempo do que a de um cluster padrão. No futuro, o eksctl poderá mudar para uma função Lambda habilitada para VPC para realizar essas operações de API.

Acesso de saída via servidores proxy HTTP

O eksctl é capaz de se comunicar com a AWS APIs por meio de um servidor proxy HTTP (S) configurado, no entanto, você precisará garantir que definiu sua lista de exclusão de proxy corretamente.

Geralmente, você precisará garantir que as solicitações do VPC endpoint do seu cluster não sejam roteadas por meio de seus proxies definindo uma variável de ambiente apropriadano`_proxy`, incluindo o valor. `.eks.amazonaws.com`

Se seu servidor proxy realizar a “interceptação de SSL” e você estiver usando funções do IAM para contas de serviço (IRSA), você precisará garantir que você ignore explicitamente o SSL para o domínio. Man-in-the-Middle `oidc.<region>.amazonaws.com` Se não fizer isso, o eksctl obterá a impressão digital incorreta do certificado raiz do provedor OIDC, e o plug-in CNI do AWS VPC

falhará na inicialização devido à impossibilidade de obter as credenciais do IAM, tornando seu cluster inoperante.

Mais informações

- Clusters privados EKS

Complementos

Este tópico descreve como gerenciar complementos do Amazon EKS para seus clusters do Amazon EKS usando eksctl. O EKS Add-Ons é um recurso que permite habilitar e gerenciar o software operacional Kubernetes por meio da API EKS, simplificando o processo de instalação, configuração e atualização de complementos de cluster.

> **Warning**
>
> O eksctl agora instala complementos padrão (vpc-cni, coredns, kube-proxy) como complementos do EKS em vez de complementos autogerenciados. Isso significa que você deve usar comandos `eksctl update addon` em vez de `eksctl utils update-*` comandos para clusters criados com eksctl v0.184.0 e superior.

Você pode criar clusters sem nenhum complemento de rede padrão quando quiser usar plug-ins CNI alternativos, como Cilium e Calico.

Os complementos do EKS agora oferecem suporte ao recebimento de permissões do IAM por meio do EKS Pod Identity Associations, permitindo que eles se conectem aos serviços da AWS fora do cluster

Criação de complementos

O Eksctl fornece mais flexibilidade para gerenciar complementos de cluster:

Em seu arquivo de configuração, você pode especificar os complementos que deseja e (se necessário) a função ou as políticas a serem anexadas a eles:

```
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: '*'
```

Você pode especificar no máximo um dos`attachPolicy`, `attachPolicyARNs` `serviceAccountRoleARN` e.

Se nenhuma delas for especificada, o complemento será criado com uma função que tenha todas as políticas recomendadas anexadas.

> **Note**
>
> Para anexar políticas aos complementos, seu cluster deve estar `OIDC` habilitado. Se não estiver ativado, ignoramos todas as políticas anexadas.

Em seguida, você pode criar esses complementos durante o processo de criação do cluster:

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

Ou crie os complementos explicitamente após a criação do cluster usando o arquivo de configuração ou os sinalizadores da 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**
>
> Use a `--namespace-config` sinalização para implantar complementos em um namespace personalizado em vez do namespace padrão.

Durante a criação do complemento, se uma versão autogerenciada do complemento já existir no cluster, você pode escolher como os possíveis `configMap` conflitos devem ser resolvidos definindo a `resolveConflicts` opção por meio do arquivo de configuração, por exemplo

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

Para criação de complementos, o `resolveConflicts` campo suporta três valores distintos:

- `none`- O EKS não altera o valor. A criação pode falhar.
- `overwrite`- O EKS substitui todas as alterações de configuração para os valores padrão do EKS.
- `preserve`- O EKS não altera o valor. A criação pode falhar. (Da mesma forma`none`, mas diferente da atualização `preserve` de complementos).

Listando complementos habilitados

Você pode ver quais complementos estão habilitados no seu cluster executando:

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

or

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

Configurando a versão do complemento

Definir a versão do complemento é opcional. Se o `version` campo for deixado vazio, a versão padrão do complemento `eksctl` será resolvida. Mais informações sobre qual versão é a versão padrão para complementos específicos podem ser encontradas na documentação da AWS sobre o EKS. Observe que a versão padrão pode não ser necessariamente a versão mais recente disponível.

A versão do complemento pode ser definida como. `latest` Como alternativa, a versão pode ser definida com a tag de construção EKS especificada, como `v1.7.5-eksbuild.1` ou`v1.7.5-eksbuild.2`. Também pode ser configurado para a versão de lançamento do complemento, como `v1.7.5` ou`1.7.5`, e a tag de `eksbuild` sufixo será descoberta e definida para você.

Veja a seção abaixo sobre como descobrir os complementos disponíveis e suas versões.

Descobrindo complementos

Você pode descobrir quais complementos estão disponíveis para instalação em seu cluster executando:

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

Isso descobrirá a versão do kubernetes do seu cluster e a filtrará. Como alternativa, se você quiser ver quais complementos estão disponíveis para uma versão específica do kubernetes, execute:

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

Você também pode descobrir complementos filtrando por seus,. `type` `owner` and/or `publisher` Por exemplo, para ver complementos de um determinado proprietário e tipo, você pode executar:

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

Os `types` `publishers` sinalizadores `owners` e são opcionais e podem ser especificados juntos ou individualmente para filtrar os resultados.

Descobrindo o esquema de configuração para complementos

Depois de descobrir o complemento e a versão, você pode ver as opções de personalização buscando seu esquema de configuração JSON.

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

Isso retorna um esquema JSON das várias opções disponíveis para esse complemento.

Trabalhando com valores de configuração

`ConfigurationValues`podem ser fornecidos no arquivo de configuração durante a criação ou atualização dos complementos. Somente os formatos JSON e YAML são compatíveis.

Por exemplo. ,

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

```
addons:
- name: coredns
  version: latest
```

```
  configurationValues: "{\"replicaCount\":3}"
  resolveConflicts: overwrite
```

> **Note**
>
> Lembre-se de que, quando os valores de configuração do complemento forem modificados, surgirão conflitos de configuração.

```
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`.
```

Além disso, o comando get agora também será recuperado `ConfigurationValues` para o complemento. por exemplo

```
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
```

Usando namespace personalizado

Um namespace personalizado pode ser fornecido no arquivo de configuração durante a criação dos complementos. Um namespace não pode ser atualizado depois que um complemento é criado.

Usando o arquivo de configuração

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

```
      namespace: custom-namespace
```

Usando o sinalizador CLI

Como alternativa, você pode especificar um namespace personalizado usando a `--namespace-config` sinalização:

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

O comando get também recuperará o valor do namespace para o complemento

```
- ConfigurationValues: ""
  IAMRole: ""
  Issues: null
  Name: aws-ebs-csi-driver
  NamespaceConfig:
    namespace: custom-namespace
  NewerVersion: ""
  PodIdentityAssociations: null
  Status: ACTIVE
  Version: v1.47.0-eksbuild.1
```

Atualizando complementos

Você pode atualizar seus complementos para versões mais recentes e alterar quais políticas estão anexadas executando:

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

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

> **Note**
>
> A configuração do namespace não pode ser atualizada após a criação de um complemento. A `--namespace-config` bandeira só está disponível durante a criação do complemento.

Da mesma forma que na criação de um complemento, ao atualizar um complemento, você tem controle total sobre as alterações de configuração que você pode ter aplicado anteriormente nesse complemento. `configMap` Especificamente, você pode preservá-los ou sobrescrevê-los. Essa funcionalidade opcional está disponível por meio do mesmo campo do arquivo de `resolveConflicts` configuração. por exemplo,

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

Para atualização do complemento, o `resolveConflicts` campo aceita três valores distintos:

- `none`- O EKS não altera o valor. A atualização pode falhar.
- `overwrite`- O EKS substitui todas as alterações de configuração para os valores padrão do EKS.
- `preserve`- O EKS preserva o valor. Se você escolher essa opção, recomendamos testar todas as alterações de campo e valor em um cluster que não seja de produção antes de atualizar o complemento em seu cluster de produção.

Excluindo complementos

Você pode excluir um complemento executando:

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

Isso excluirá o complemento e todas as funções do IAM associadas a ele.

Quando você exclui seu cluster, todas as funções do IAM associadas aos complementos também são excluídas.

Flexibilidade de criação de clusters para complementos de rede padrão

Quando um cluster é criado, o EKS instala automaticamente o VPC CNI, o CoreDNS e o kube-proxy como complementos autogerenciados. Para desativar esse comportamento para usar outros plug-ins CNI, como Cilium e Calico, o eksctl agora suporta a criação de um cluster sem nenhum complemento de rede padrão. Para criar esse cluster, defina`addonsConfig.disableDefaultAddons`, como em:

```
addonsConfig:
  disableDefaultAddons: true
```

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

Para criar um cluster somente com CoreDNS e kube-proxy e não com VPC CNI, especifique os complementos explicitamente em e defina, como em: `addons` `addonsConfig.disableDefaultAddons`

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

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

Como parte dessa mudança, o eksctl agora instala complementos padrão como complementos do EKS em vez de complementos autogerenciados durante a criação do cluster, se `addonsConfig.disableDefaultAddons` não estiver explicitamente definido como verdadeiro. Dessa forma, `eksctl utils update-*` os comandos não podem mais ser usados para atualizar complementos para clusters criados com eksctl v0.184.0 e superior:

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

Em vez disso, `eksctl update addon` deve ser usado agora.

Para saber mais, consulte O Amazon EKS introduz a flexibilidade de criação de clusters para complementos de rede.

Habilitando o acesso para o Amazon EMR

Para permitir que o EMR execute operações na API do Kubernetes, seu SLR precisa receber as permissões RBAC necessárias. O eksctl fornece um comando que cria os recursos RBAC

necessários para o EMR e os atualiza para vincular a função ao SLR para EMR. `aws-auth` ConfigMap

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

Suporte do EKS Fargate

O AWS Fargate é um mecanismo de computação gerenciado para o Amazon ECS que pode executar contêineres. No Fargate, você não precisa gerenciar servidores ou clusters.

O Amazon EKS agora pode lançar pods no AWS Fargate. Isso elimina a necessidade de se preocupar com a forma como você provisiona ou gerencia a infraestrutura para pods e facilita a criação e a execução de aplicativos Kubernetes de alto desempenho e alta disponibilidade na AWS.

Criação de um cluster com o suporte do Fargate

Você pode adicionar um cluster com suporte ao Fargate com:

```
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
```

Esse comando terá criado um cluster e um perfil Fargate. Esse perfil contém determinadas informações necessárias para a AWS instanciar pods no Fargate. Eles são:

- função de execução do pod para definir as permissões necessárias para executar o pod e o local de rede (sub-rede) para executar o pod. Isso permite que as mesmas permissões de rede e segurança sejam aplicadas a vários pods do Fargate e facilita a migração dos pods existentes em um cluster para o Fargate.
- Seletor para definir quais pods devem ser executados no Fargate. Isso é composto por um `namespace` `labels` e.

Quando o perfil não é especificado, mas o suporte para o Fargate está ativado, `--fargate` um perfil Fargate padrão é criado. Esse perfil tem como alvo os namespaces `default` e os `kube-system` namespaces para que os pods nesses namespaces sejam executados no Fargate.

O perfil do Fargate que foi criado pode ser verificado com o seguinte comando:

```
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
```

Para saber mais sobre seletores, consulte Criação de perfis do Fargate.

Criação de um cluster com suporte ao Fargate usando um arquivo de configuração

O arquivo de configuração a seguir declara um cluster EKS com um grupo de nós composto por uma instância do EC2 `m5.large` e dois perfis Fargate. Todos os pods definidos nos `kube-system` namespaces `default` e serão executados no Fargate. Todos os pods no `dev` namespace que também têm o rótulo também `dev=passed` serão executados no Fargate. Quaisquer outros pods serão programados no node inng-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
```

Projetando perfis Fargate

Cada entrada do seletor tem até dois componentes, namespace e uma lista de pares de valores-chave. Somente o componente de namespace é necessário para criar uma entrada seletora. Todas as regras (namespaces, pares de valores-chave) devem ser aplicadas a um pod para corresponder a uma entrada do seletor. Um pod só precisa corresponder a uma entrada do seletor para ser executado no perfil. Qualquer pod que corresponda a todas as condições em um campo seletor seria programado para ser executado no Fargate. Todos os pods que não correspondessem aos namespaces da lista branca, mas em que o usuário definisse manualmente o arquivo scheduler: fargate-scheduler, ficariam presos em um estado pendente, pois não estavam autorizados a serem executados no Fargate.

Os perfis devem atender aos seguintes requisitos:

- Um seletor é obrigatório por perfil
- Cada seletor deve incluir um namespace; os rótulos são opcionais

Exemplo: agendamento da carga de trabalho no Fargate

Para agendar pods no Fargate para o exemplo mencionado acima, pode-se, por exemplo, criar um namespace `dev` chamado e implantar a carga de trabalho lá:

```
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
```

Na saída do último `kubectl get pods` comando, podemos ver que o `nginx` pod está implantado em um nó chamado`fargate-ip-192-168-183-140.ap-northeast-1.compute.internal`.

Gerenciando perfis do Fargate

Para implantar cargas de trabalho do Kubernetes no Fargate, o EKS precisa de um perfil do Fargate. Ao criar um cluster como nos exemplos acima, `eksctl` cuide disso criando um perfil padrão. Considerando um cluster já existente, também é possível criar um perfil Fargate com o `eksctl create fargateprofile` comando:

> **Note**
>
> Essa operação só é suportada em clusters executados na versão da plataforma EKS `eks.5` ou superior.

> **Note**
>
> Se o existente foi criado com uma versão `eksctl` anterior à 0.11.0, você precisará executar `eksctl upgrade cluster` antes de criar o perfil 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"
```

Você também pode especificar o nome do perfil do Fargate a ser criado. Esse nome não deve começar com o prefixo`eks-`.

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

Usando esse comando com sinalizadores CLI, o eksctl só pode criar um único perfil do Fargate com um seletor simples. Para seletores mais complexos, por exemplo, com mais namespaces, o eksctl suporta o uso de um arquivo de configuração:

```
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
```

Para ver os perfis Fargate existentes em um 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
```

E para vê-los em `yaml` formato:

```
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 no `json` formato:

```
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"
        ]
    }
]
```

Os perfis Fargate são imutáveis por design. Para alterar alguma coisa, crie um novo perfil do Fargate com as alterações desejadas e exclua o antigo com o `eksctl delete fargateprofile` comando, como no exemplo a seguir:

```
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"
}
```

Observe que a exclusão do perfil é um processo que pode levar alguns minutos. Quando a `--wait` bandeira não é especificada, espera com `eksctl` otimismo que o perfil seja excluído e retorne assim que a solicitação da API da AWS for enviada. Para `eksctl` esperar até que o perfil seja excluído com sucesso, use `--wait` como no exemplo acima.

Outras fontes de leitura

- AWS Fargate
- O Amazon EKS agora pode lançar pods no AWS Fargate

Atualizações de cluster

Um cluster gerenciado por `eksctl` pode ser atualizado em 3 etapas fáceis:

1. atualize a versão do plano de controle com `eksctl upgrade cluster`
2. atualizar grupos de nós
3. atualize os complementos de rede padrão (para obter mais informações, consultethe section called “Atualizações de complementos padrão”):

Analise cuidadosamente os recursos relacionados ao upgrade do cluster:

- Atualize o cluster existente para a nova versão do Kubernetes no Guia do usuário do Amazon EKS
- Práticas recomendadas para atualizações de clusters no Guia de melhores práticas do EKS

> **Note**
>
> O antigo `eksctl update cluster` será descontinuado. Use `eksctl upgrade cluster` em vez disso.

Atualizando a versão do plano de controle

As atualizações da versão do plano de controle devem ser feitas para uma versão secundária por vez.

Para atualizar o plano de controle para a próxima versão disponível, execute:

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

Esse comando não aplicará nenhuma alteração imediatamente. Você precisará executá-lo novamente `--approve` para aplicar as alterações.

A versão de destino para o upgrade do cluster pode ser especificada com o sinalizador CLI:

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

ou com o arquivo de configuração

```
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**
>
> Os únicos valores permitidos para os `metadata.version` argumentos `--version` e são a versão atual do cluster ou uma versão superior. Não há suporte para atualizações de mais de uma versão do Kubernetes.

Atualizações de complementos padrão

Este tópico explica como atualizar os complementos pré-instalados padrão que estão incluídos nos clusters EKS.

> **Warning**
>
> O eksctl agora instala complementos padrão como complementos do EKS em vez de complementos autogerenciados. Leia mais sobre suas implicações na flexibilidade de criação de clusters para complementos de rede padrão.
> Para atualizar complementos, `eksctl utils update-<addon>` não pode ser usado para clusters criados com eksctl v0.184.0 e superior. Este guia é válido somente para clusters criados antes dessa alteração.

Há três complementos padrão que são incluídos em cada cluster EKS:

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

- `coredns`

Atualize o complemento pré-instalado

Para complementos oficiais do EKS que são criados manualmente por meio `eksctl create addons` ou após a criação do cluster, a maneira de gerenciá-los é por meio `eksctl create/get/update/delete addon` de. Nesses casos, consulte a documentação sobre os complementos do EKS.

O processo de atualização de cada um deles é diferente, portanto, há 3 comandos distintos que você precisará executar. Todos os comandos a seguir são aceitos`--config-file`. Por padrão, cada um desses comandos é executado no modo plano. Se você estiver satisfeito com as alterações propostas, execute novamente com`--approve`.

Para atualizar`kube-proxy`, execute:

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

Para atualizar`aws-node`, execute:

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

Para atualizar`coredns`, execute:

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

Depois de atualizar, certifique-se de executar `kubectl get pods -n kube-system` e verificar se todos os pods de complemento estão prontos. Você verá algo assim:

```
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 para Zonal Shift em clusters EKS

O EKS agora oferece suporte à mudança zonal e à mudança automática zonal do Amazon Application Recovery Controller (ARC) que aprimoram a resiliência dos ambientes de cluster Multi-AZ. Com o AWS Zonal Shift, os clientes podem afastar o tráfego no cluster de uma zona de disponibilidade prejudicada, garantindo que novos pods e nós do Kubernetes sejam lançados somente em zonas de disponibilidade saudáveis.

Criação de um cluster com mudança zonal habilitada

```
# 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
```

Habilitando a mudança zonal em um cluster existente

Para ativar ou desativar a mudança de zona em um cluster existente, execute

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

ou sem um arquivo de configuração:

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

Mais informações

- Mudança zonal EKS

Suporte do Karpenter

`eksctl`fornece suporte para adicionar o Karpenter a um cluster recém-criado. Ele criará todos os pré-requisitos necessários descritos na seção Introdução do Karpenter, incluindo a instalação do próprio Karpenter usando o Helm. Atualmente, oferecemos suporte à instalação de versões`0.28.0+`. Consulte a seção de compatibilidade do Karpenter para obter mais detalhes.

A configuração de cluster a seguir descreve uma instalação típica do Karpenter:

```
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
```

A versão é a versão do Karpenter, pois pode ser encontrada em seu repositório Helm. As seguintes opções também estão disponíveis para serem definidas:

```
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
```

O OIDC deve ser definido para instalar o Karpenter.

Depois que o Karpenter for instalado com sucesso, adicione NodePool(s) e NodeClass(es) para permitir que o Karpenter comece a adicionar nós ao cluster.

A `nodeClassRef` seção NodePool “s” deve corresponder ao nome de um`EC2NodeClass`. Por exemplo:

```
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
```

Observe que você deve especificar um dos nós de `instanceProfile` inicialização `role` ou para eles. Se você optar por usar `instanceProfile` o nome do perfil criado, `eksctl` siga o padrão:`eksctl-KarpenterNodeInstanceProfile-<cluster-name>`.

Marcação automática de grupos de segurança

`eksctl`marca automaticamente o grupo de segurança de nós compartilhados do cluster com `karpenter.sh/discovery` quando o Karpenter está habilitado (`karpenter.version`especificado) e a `karpenter.sh/discovery` tag existe em. `metadata.tags` Isso permite a compatibilidade com o AWS Load Balancer Controller.

Observe que com o karpenter 0.32.0+, os provisionadores foram descontinuados e substituídos por. NodePool

Esquema de configuração do cluster

> **Note**
>
> A localização do esquema está sendo migrada no momento.

Você pode usar um arquivo yaml para criar um cluster. Veja a referência do esquema.

Por exemplo:

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

A referência do esquema para esse arquivo está disponível em. GitHub

Para obter mais informações sobre como usar o arquivo, consultethe section called “Criar e gerenciar clusters”.

Grupos de nós

Este capítulo inclui informações sobre como você cria e configura grupos de nós com o Eksctl. Os grupos de nós são grupos de instâncias do EC2 anexados a um cluster EKS.

Tópicos:

- the section called “Instâncias spot”
  - Crie e gerencie clusters EKS com instâncias Spot usando grupos de nós gerenciados
  - Configure instâncias spot para grupos de nós não gerenciados usando o MixedInstancesPolicy
  - Distinga as instâncias spot e sob demanda usando o rótulo `node-lifecycle` Kubernetes
- the section called “ajuste de escala automático”
  - Habilite o escalonamento automático dos nós de cluster do Kubernetes criando um cluster ou grupo de nós com a função IAM que permita o uso do autoescalador de cluster
  - Configure as definições do grupo de nós para incluir as tags e anotações necessárias para que o autoescalador do cluster escale o grupo de nós.
  - Crie grupos de nós separados para cada zona de disponibilidade se as cargas de trabalho tiverem requisitos específicos da zona, como armazenamento específico da zona ou regras de afinidade
- the section called “Grupos de nós gerenciados pelo EKS”
  - Provisione e gerencie instâncias EC2 (nós) para clusters Amazon EKS Kubernetes
  - Aplique facilmente correções de bugs, patches de segurança e atualize os nós para as versões mais recentes do Kubernetes
- the section called “Nodos híbridos EKS”
  - Permita a execução de aplicativos locais e de ponta em uma infraestrutura gerenciada pelo cliente com os mesmos clusters, recursos e ferramentas do AWS EKS usados na nuvem da AWS
  - Configure a rede para conectar redes locais a uma AWS VPC, usando opções como AWS VPN ou Site-to-Site AWS Direct Connect
  - Configure credenciais para que os nós remotos se autentiquem com o cluster EKS, usando o AWS Systems Manager (SSM) ou o AWS IAM Roles Anywhere
- the section called “Config de reparo do Node”

  - Habilitando o Node Repair para grupos de nós gerenciados do EKS para monitorar e substituir ou reinicializar automaticamente nós de trabalho não íntegros
- the section called “Suporte ARM”
  - Crie um cluster EKS com instâncias Graviton baseadas em ARM para melhorar o desempenho e a economia
- the section called “Manchas”
  - Aplique manchas a grupos de nós específicos em um cluster Kubernetes
  - Controle o agendamento e o despejo de cápsulas com base em chaves, valores e efeitos de contaminação
- the section called “Suporte a modelo de execução”
  - Lançamento de grupos de nós gerenciados usando um modelo de lançamento do EC2 fornecido
  - Atualizando um grupo de nós gerenciados para usar uma versão diferente de um modelo do Launch
  - Compreender as limitações e considerações ao usar modelos personalizados AMIs e do Launch com grupos de nós gerenciados
- the section called “Trabalhe com grupos de nós”
  - Habilite o acesso SSH às instâncias do EC2 no grupo de nós
  - Aumentar ou diminuir o número de nós em um grupo de nós
- the section called “Sub-redes personalizadas”
  - Estenda uma VPC existente com uma nova sub-rede e adicione um grupo de nós a essa sub-rede
- the section called “Inicialização do Node”
  - Entenda o novo processo de inicialização de nós (nodeadm) introduzido em 2023 AmazonLinux
  - Saiba mais sobre as NodeConfig configurações padrão aplicadas pelo eksctl para nós autogerenciados e gerenciados pelo EKS
  - Personalize o processo de inicialização do nó fornecendo um overrideBootstrapCommand NodeConfig
- the section called “Grupos de nós não gerenciados”
  - Crie ou atualize grupos de nós não gerenciados em um cluster EKS
  - Atualize complementos padrão do Kubernetes, como kube-proxy, aws-node e CoreDNS
- the section called “Support para GPU”

  - O Eksctl suporta a seleção de tipos de instância de GPU para grupos de nós, permitindo o uso de cargas de trabalho aceleradas por GPU em clusters EKS.
  - O Eksctl instala automaticamente o plug-in do dispositivo NVIDIA Kubernetes quando um tipo de instância habilitado para GPU é selecionado, facilitando o uso dos recursos da GPU no cluster.
  - Os usuários podem desativar a instalação automática do plug-in e instalar manualmente uma versão específica do plug-in do dispositivo NVIDIA Kubernetes usando os comandos fornecidos.
- the section called “Seletor de instâncias”
  - Gere automaticamente uma lista de tipos de instância EC2 adequados com base em critérios de recursos como v CPUs GPUs, memória e arquitetura de CPU
  - Crie clusters e grupos de nós com os tipos de instância correspondentes aos critérios especificados do seletor de instâncias
  - Realize uma execução seca para inspecionar e modificar os tipos de instância correspondentes ao seletor de instâncias antes de criar um grupo de nós
- the section called “Mapeamentos de volume adicionais”
  - Configurar mapeamentos de volume adicionais para um grupo de nós gerenciados em um cluster EKS
  - Personalize as propriedades do volume, como tamanho, tipo, criptografia, IOPS e taxa de transferência para os volumes adicionais
  - Anexe snapshots existentes do EBS como volumes adicionais ao grupo de nós
- the section called “Nodos de trabalho do Windows”
  - Adicione grupos de nós do Windows a um cluster Linux Kubernetes existente para permitir a execução de cargas de trabalho do Windows
  - Agende cargas de trabalho no sistema operacional apropriado (Windows ou Linux) usando seletores de nós com base nos `kubernetes.io/os` rótulos e `kubernetes.io/arch`
- the section called “Suporte personalizado para AMI”
  - Use a `--node-ami` bandeira para especificar uma AMI personalizada para grupos de nós, consultar a AWS para obter a AMI otimizada para EKS mais recente ou usar o AWS Systems Manager Parameter Store para encontrar a AMI.
  - Defina o `--node-ami-family` sinalizador para especificar a família do sistema operacional para o grupo de nós AMI, como AmazonLinux 2, Ubuntu2204 ou 2022. WindowsServer CoreContainer
  - Para grupos de nós do Windows, especifique uma AMI personalizada e forneça um script de PowerShell bootstrap por meio do`overrideBootstrapCommand`.

- the section called “DNS personalizado”
  - Substituir o endereço IP do servidor DNS usado para pesquisas de DNS internas e externas

Trabalhe com grupos de nós

Criação de grupos de nós

Você pode adicionar um ou mais grupos de nós além do grupo de nós inicial criado junto com o cluster.

Para criar um grupo de nós adicional, use:

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

> **Note**
>
> `--version`O sinalizador não é compatível com grupos de nós gerenciados. Ele sempre herda a versão do plano de controle.

Por padrão, novos grupos de nós não gerenciados herdam a versão do plano de controle (`--version=auto`), mas você pode especificar uma versão diferente, que também pode ser usada `--version=latest` para forçar o uso da versão mais recente.

Além disso, você pode usar o mesmo arquivo de configuração usado para`eksctl create cluster`:

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

Criando um grupo de nós a partir de um arquivo de configuração

Os grupos de nós também podem ser criados por meio de uma definição de cluster ou arquivo de configuração. Dado o seguinte exemplo de arquivo de configuração e um cluster existente chamado`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
```

Os grupos de nós `ng-1-workers` e `ng-2-builders` podem ser criados com este comando:

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

Balanceamento de carga

Se você já se preparou para anexar os or/and grupos-alvo dos balanceadores de carga clássicos existentes aos grupos de nós, você pode especificá-los no arquivo de configuração. Os or/and grupos-alvo dos balanceadores de carga clássicos são automaticamente associados ao ASG ao criar grupos de nós. Isso só é suportado para grupos de nós autogerenciados definidos por meio do campo. `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
```

Seleção de grupos de nós em arquivos de configuração

Para realizar uma `delete` operação `create` or em apenas um subconjunto dos grupos de nós especificados em um arquivo de configuração, há dois sinalizadores CLI que aceitam uma lista de globs e, por exemplo: `0` 1

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

Usando o arquivo de configuração de exemplo acima, é possível criar todos os grupos de nós de trabalhadores, exceto o de trabalhadores, com o seguinte comando:

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

Ou pode-se excluir o grupo de nós do construtor com:

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

Nesse caso, também precisamos fornecer o `--approve` comando para realmente excluir o grupo de nós.

Incluir e excluir regras

- se não `--include` ou `--exclude` for especificado, tudo está incluído
- se only `--include` for especificado, somente grupos de nós que correspondam a esses globos serão incluídos
- se apenas `--exclude` for especificado, todos os grupos de nós que não correspondem a esses globos serão incluídos
- se ambos forem especificados, `--exclude` as regras terão precedência `--include` (ou seja, grupos de nós que correspondam às regras em ambos os grupos serão excluídos)

Listando grupos de nós

Para listar os detalhes sobre um grupo de nós ou todos os grupos de nós, use:

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

Para listar um ou mais grupos de nós no formato YAML ou JSON, que gera mais informações do que a tabela de log padrão, use:

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

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

Imutabilidade do grupo de nós

Por design, os grupos de nós são imutáveis. Isso significa que, se você precisar alterar algo (além do escalonamento), como a AMI ou o tipo de instância de um grupo de nós, precisará criar um novo grupo de nós com as alterações desejadas, mover a carga e excluir o antigo. Consulte a seção Excluindo e drenando grupos de nós.

Escalando grupos de nós

O escalonamento de grupos de nós é um processo que pode levar alguns minutos. Quando a `--wait` bandeira não é especificada, espera com `eksctl` otimismo que o grupo de nós seja escalado e retorne assim que a solicitação da API da AWS for enviada. Para `eksctl` esperar até que os nós estejam disponíveis, adicione um `--wait` sinalizador como no exemplo abaixo.

> **Note**
>
> Escalar um grupo de nós down/in (ou seja, reduzir o número de nós) pode resultar em erros, pois dependemos apenas de alterações no ASG. Isso significa que os nós que removed/terminated estão sendo não estão explicitamente drenados. Essa pode ser uma área que pode ser melhorada no futuro.

O escalonamento de um grupo de nós gerenciados é obtido chamando diretamente a API EKS, que atualiza a configuração de um grupo de nós gerenciados.

Dimensionando um único grupo de nós

Um grupo de nós pode ser escalado usando o comando: `eksctl scale nodegroup`

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

Por exemplo, para escalar o grupo de nós `ng-a345f4e1` em `cluster-1` até 5 nós, execute:

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

Um grupo de nós também pode ser escalado usando um arquivo de configuração passado `--config-file` e especificando o nome do grupo de nós com o qual deve ser escalado. `--name` O Eksctl pesquisará o arquivo de configuração e descobrirá esse grupo de nós, bem como seus valores de configuração de escala.

Se o número desejado de nós estiver `NOT` dentro da faixa do número mínimo atual e do número máximo atual, um erro específico será mostrado. Esses valores também podem ser passados com sinalizadores `--nodes-min` e `--nodes-max` respectivamente.

Dimensionando vários grupos de nós

O Eksctl pode descobrir e escalar todos os grupos de nós encontrados em um arquivo de configuração que é passado com. `--config-file`

Da mesma forma que escalar um único grupo de nós, o mesmo conjunto de validações se aplica a cada grupo de nós. Por exemplo, o número desejado de nós deve estar dentro da faixa do número mínimo e máximo de nós.

Excluindo e drenando grupos de nós

Para excluir um grupo de nós, execute:

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

As regras de inclusão e exclusão também podem ser usadas com esse comando.

> **Note**
>
> Isso drenará todos os pods desse grupo de nós antes que as instâncias sejam excluídas.

Para ignorar as regras de despejo durante o processo de drenagem, execute:

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

Todos os nós são isolados e todos os pods são removidos de um grupo de nós na exclusão, mas se você precisar drenar um grupo de nós sem excluí-lo, execute:

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

Para desconectar o fio em um grupo de nós, execute:

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

Para ignorar as regras de despejo, como PodDisruptionBudget configurações, execute:

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

Para acelerar o processo de drenagem, você pode especificar `--parallel <value>` o número de nós a serem drenados em paralelo.

Outros recursos

Você também pode ativar o acesso SSH, ASG e outros recursos para um grupo de nós, por exemplo:

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

Atualizar rótulos

Não há comandos específicos `eksctl` para atualizar os rótulos de um grupo de nós, mas isso pode ser facilmente obtido usando`kubectl`, por exemplo:

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

Acesso SSH

Você pode habilitar o acesso SSH para grupos de nós configurando um dos `publicKeyName` e `publicKeyPath` em sua configuração de `publicKey` grupo de nós. Como alternativa, você pode usar o AWS Systems Manager (SSM) para fazer SSH nos nós, configurando o grupo de nós com: `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
```

Grupos de nós não gerenciados

Dentro`eksctl`, definir `--managed=false` ou usar o `nodeGroups` campo cria um grupo de nós não gerenciado. Lembre-se de que grupos de nós não gerenciados não aparecem no console do EKS, que, via de regra, só conhece grupos de nós gerenciados pelo EKS.

Você deve atualizar os nodegroups somente após a execução. `eksctl upgrade cluster` (Consulte Atualização de clusters.)

Se você tiver um cluster simples com apenas um grupo de nós inicial (ou seja, criado com`eksctl create cluster`), o processo é muito simples:

1. Obtenha o nome do antigo nodegroup:

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

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

2. Crie um novo grupo de nós:

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

3. Exclua o grupo de nós antigo:

```
 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.
> ```

Atualizando vários grupos de nós

Se você tiver vários grupos de nós, é sua responsabilidade monitorar como cada um foi configurado. Você pode fazer isso usando arquivos de configuração, mas se ainda não os tiver usado, precisará inspecionar seu cluster para descobrir como cada grupo de nós foi configurado.

Em termos gerais, você está procurando:

- revise quais grupos de nós você tem e quais podem ser excluídos ou devem ser substituídos pela nova versão
- anote a configuração de cada grupo de nós, considere usar o arquivo de configuração para facilitar as atualizações na próxima vez

Atualizando com arquivo de configuração

Se você estiver usando o arquivo de configuração, você precisará fazer o seguinte.

Edite o arquivo de configuração para adicionar novos grupos de nós e remover grupos de nós antigos. Se você quiser apenas atualizar os grupos de nós e manter a mesma configuração, basta alterar os nomes dos grupos de nós, por exemplo, acrescentar ao nome. `-v2`

Para criar todos os novos grupos de nós definidos no arquivo de configuração, execute:

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

Depois de instalar novos grupos de nós, você pode excluir os antigos:

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

> **Note**
>
> A primeira execução está no modo de planejamento; se você estiver satisfeito com as alterações propostas, execute novamente com `--approve`.

Atualizando complementos padrão

Talvez seja necessário atualizar os complementos de rede instalados em seu cluster. Para obter mais informações, consulte the section called “Atualizações de complementos padrão”.

Grupos de nós gerenciados pelo EKS

Os grupos de nós gerenciados do Amazon EKS são um recurso que automatiza o provisionamento e o gerenciamento do ciclo de vida dos nós (instâncias EC2) para clusters do Amazon EKS Kubernetes. Os clientes podem provisionar grupos otimizados de nós para seus clusters e o EKS manterá seus nós atualizados com as versões mais recentes do Kubernetes e do sistema operacional host.

Um grupo de nós gerenciados pelo EKS é um grupo de escalonamento automático e instâncias EC2 associadas que são gerenciadas pela AWS para um cluster Amazon EKS. Cada grupo de nós usa a AMI Amazon Linux 2 otimizada para Amazon EKS. O Amazon EKS facilita a aplicação de correções de bugs e patches de segurança aos nós, bem como a atualização deles para as versões mais recentes do Kubernetes. Cada grupo de nós lança um grupo de escalonamento automático para seu cluster, que pode abranger várias zonas de disponibilidade e sub-redes do AWS VPC para alta disponibilidade.

NOVO suporte ao modelo de lançamento para grupos de nós gerenciados

> **Note**
>
> O termo “grupos de nós não gerenciados” tem sido usado para se referir aos grupos de nós que o eksctl suporta desde o início (representados por meio do campo). `nodeGroups`

O `ClusterConfig` arquivo continua usando o `nodeGroups` campo para definir grupos de nós não gerenciados, e os grupos de nós gerenciados são definidos com o campo. `managedNodeGroups`

Criação de grupos de nós gerenciados

```
$ eksctl create nodegroup
```

Novos clusters

Para criar um novo cluster com um grupo de nós gerenciado, execute

```
eksctl create cluster
```

Para criar vários grupos de nós gerenciados e ter mais controle sobre a configuração, um arquivo de configuração pode ser usado.

Note

Grupos de nós gerenciados não têm paridade completa de recursos com grupos de nós não gerenciados.

```
# 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
```

Outro exemplo de arquivo de configuração para criar um grupo de nós gerenciado pode ser encontrado aqui.

É possível ter um cluster com grupos de nós gerenciados e não gerenciados. Grupos de nós não gerenciados não aparecem no console do AWS EKS, mas `eksctl get nodegroup` listarão os dois tipos de grupos de nós.

```
# 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
```

NOVO Suporte para AMI personalizada, grupos de segurançainstancePrefix,instanceName,ebsOptimized,volumeType,volumeName,volumeEncryp e 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'
```

Se você estiver solicitando um tipo de instância disponível somente em uma zona (e a configuração do eksctl exigir a especificação de duas), certifique-se de adicionar a zona de disponibilidade à sua solicitação de grupo de nós:

```
# 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
```

Isso pode ser verdade para tipos de exemplo, como a família Hpc6, que só estão disponíveis em uma zona.

Clusters existentes

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

Dica: se você estiver usando um `ClusterConfig` arquivo para descrever todo o cluster, descreva seu novo grupo de nós gerenciados no `managedNodeGroups` campo e execute:

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

Atualizando grupos de nós gerenciados

Você pode atualizar um grupo de nós para a versão mais recente da AMI otimizada para EKS para o tipo de AMI que você está usando a qualquer momento.

Se seu grupo de nós for da mesma versão do Kubernetes do cluster, você poderá atualizar para a versão mais recente da AMI para aquela versão do Kubernetes do tipo de AMI que você está usando. Se seu nodegroup for a versão anterior do Kubernetes da versão do Kubernetes do cluster, você poderá atualizar o nodegroup para a versão mais recente da AMI que corresponda à versão do Kubernetes do nodegroup ou atualizar para a versão mais recente da AMI que corresponda à versão do Kubernetes do cluster. Você não pode reverter um grupo de nós para uma versão anterior do Kubernetes.

Para atualizar um grupo de nós gerenciado para a versão mais recente da AMI:

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

O nodegroup pode ser atualizado para a versão mais recente da AMI para uma versão específica do Kubernetes usando:

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

Para fazer o upgrade para uma versão específica da AMI em vez da versão mais recente, passe`--release-version`:

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

> **Note**
>
> Se os nós gerenciados forem implantados usando a personalização AMIs, o fluxo de trabalho a seguir deverá ser seguido para implantar uma nova versão da AMI personalizada.

- a implantação inicial do grupo de nós deve ser feita usando um modelo de lançamento. por exemplo

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

- crie uma nova versão da AMI personalizada (usando o console AWS EKS).
- crie uma nova versão do modelo de lançamento com o novo ID da AMI (usando o console AWS EKS).
- atualize os nós para a nova versão do modelo de lançamento. por exemplo

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

Manipulando atualizações paralelas para nós

Vários nós gerenciados podem ser atualizados simultaneamente. Para configurar atualizações paralelas, defina o `updateConfig` de um grupo de nós ao criar o grupo de nós. Um exemplo `updateConfig` pode ser encontrado aqui.

Para evitar qualquer tempo de inatividade em suas cargas de trabalho devido à atualização de vários nós ao mesmo tempo, você pode limitar o número de nós que podem ficar indisponíveis durante uma atualização especificando isso no campo de um. `maxUnavailable` `updateConfig` Como alternativa, use`maxUnavailablePercentage`, que define o número máximo de nós indisponíveis como uma porcentagem do número total de nós.

Observe que `maxUnavailable` não pode ser maior que`maxSize`. Além disso, `maxUnavailable` e `maxUnavailablePercentage` não pode ser usado simultaneamente.

Esse recurso está disponível somente para nós gerenciados.

Atualizando grupos de nós gerenciados

`eksctl`permite atualizar a UpdateConfigseção de um grupo de nós gerenciado. Esta seção define dois campos. `MaxUnavailable`MaxUnavailablePercentagee. Seus grupos de nós não são afetados durante a atualização, portanto, o tempo de inatividade não deve ser esperado.

O comando `update nodegroup` deve ser usado com um arquivo de configuração usando o `--config-file` sinalizador. O grupo de nós deve conter uma `nodeGroup.updateConfig` seção. Mais informações podem ser encontradas aqui.

Problemas de saúde do Nodegroup

O EKS Managed Nodegroups verifica automaticamente a configuração do seu grupo de nós e dos nós em busca de problemas de saúde e os relata por meio da API e do console do EKS. Para ver os problemas de saúde de um grupo de nós:

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

Gerenciando rótulos

Os grupos de nós gerenciados do EKS oferecem suporte à anexação de rótulos que são aplicados aos nós do Kubernetes no grupo de nós. Isso é especificado por meio do `labels` campo em eksctl durante a criação do cluster ou do nodegroup.

Para definir novos rótulos ou atualizar rótulos existentes em um grupo de nós:

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

Para desmarcar ou remover rótulos de um grupo de nós:

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

Para ver todos os rótulos definidos em um grupo de nós:

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

Dimensionando grupos de nós gerenciados

`eksctl scale nodegroup`também oferece suporte a grupos de nós gerenciados. A sintaxe para escalar um grupo de nós gerenciado ou não gerenciado é a mesma.

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

Mais informações

- Grupos de nós gerenciados pelo EKS

Inicialização do Node

AmazonLinux2023

AL2023 introduziu um novo processo de inicialização de nós nodeadm que usa um esquema de configuração YAML, eliminando o uso de script. `/etc/eks/bootstrap.sh`

> **Note**
>
> Com as versões 1.30 e superiores do Kubernetes, o Amazon Linux 2023 é o sistema operacional padrão.

Configurações padrão para AL2

Para nós autogerenciados e nós gerenciados pelo EKS com base na personalização AMIs, `eksctl` cria um padrão, mínimo `NodeConfig` e o injeta automaticamente nos dados do usuário do modelo de lançamento do nodegroups. ou seja

```
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

--//--
```

Para nós gerenciados pelo EKS baseados em nativos AMIs, o padrão `NodeConfig` é ser adicionado pelo EKS MNG nos bastidores, anexado diretamente aos dados do usuário do EC2. Portanto, nesse cenário, `eksctl` não é necessário incluí-lo no modelo de lançamento.

Configurando o processo de inicialização

Para definir propriedades avançadas ou simplesmente substituir os valores padrãoNodeConfig, o eksctl permite que você especifique uma via personalizada `NodeConfig` ou, por exemplo `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
```

Essa configuração personalizada será anexada aos dados do usuário por eksctl e mesclada com a configuração padrão. `nodeadm` Leia mais sobre a capacidade `nodeadm` de mesclar vários objetos de configuração aqui.

Suporte ao Launch Template para grupos de nós gerenciados

O eksctl suporta o lançamento de grupos de nós gerenciados usando um modelo de inicialização do EC2 fornecido. Isso permite várias opções de personalização para grupos de nós, incluindo o fornecimento de grupos personalizados AMIs e de segurança e a transmissão de dados do usuário para inicialização de nós.

Criação de grupos de nós gerenciados usando um modelo de lançamento fornecido

```
# 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
```

Atualizando um grupo de nós gerenciado para usar uma versão diferente do modelo de lançamento

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

> **Note**
>
> Se um modelo de lançamento estiver usando uma AMI personalizada, a nova versão também deverá usar uma AMI personalizada, caso contrário, a operação de upgrade falhará

Se um modelo de lançamento não estiver usando uma AMI personalizada, a versão do Kubernetes para a qual fazer o upgrade também poderá ser especificada:

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

Notas sobre AMI personalizada e suporte a modelos de lançamento

- Quando um modelo de lançamento é fornecido, os seguintes campos não são suportados: `instanceType` `amissh.allow`,`ssh.sourceSecurityGroupIds`,`securityGroups`,`instancePrefix`,`instanceNam` `overrideBootstrapCommand` `disableIMDSv1` e.
- Ao usar uma AMI personalizada (`ami`), ela também `overrideBootstrapCommand` deve ser configurada para realizar a inicialização.
- `overrideBootstrapCommand`só pode ser definido ao usar uma AMI personalizada.
- Quando um modelo de execução é fornecido, as tags especificadas na configuração do nodegroup se aplicam somente ao recurso EKS Nodegroup e não são propagadas para instâncias do EC2.

Sub-redes personalizadas

É possível estender uma VPC existente com uma nova sub-rede e adicionar um Nodegroup a essa sub-rede.

Por que

Se o cluster ficar sem a configuração pré-configurada IPs, é possível redimensionar a VPC existente com um novo CIDR para adicionar uma nova sub-rede a ela. Para ver como fazer isso, leia este guia sobre o AWS Extending VPCs.

TL; DR

Vá para a configuração da VPC e adicione, clique em Ações->Editar CIDRs e adicione um novo intervalo. Por exemplo:

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

Agora você precisa adicionar uma nova sub-rede. Dependendo se é uma nova sub-rede privada ou pública, você terá que copiar as informações de roteamento de uma sub-rede privada ou pública, respectivamente.

Depois que a sub-rede for criada, adicione roteamento e copie o ID do gateway NAT ou o Internet Gateway de outra sub-rede na VPC. Certifique-se de que, se for uma sub-rede pública, habilite a atribuição automática de IP. Ações->Modificar configurações de IP de atribuição automática->Ativar atribuição automática de endereço público. IPv4

Não se esqueça de copiar também as TAGS das sub-redes existentes, dependendo da configuração da sub-rede pública ou privada. Isso é importante, caso contrário, a sub-rede não fará parte do cluster e as instâncias na sub-rede não poderão se unir.

Ao terminar, copie o ID da nova sub-rede. Repita sempre que necessário.

Como

Para criar um grupo de nós na (s) sub-rede (s) criada (s), execute o seguinte comando:

```
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 use a configuração da seguinte forma:

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

Com uma configuração como esta:

```
# 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
```

Aguarde a criação do grupo de nós e as novas instâncias devem ter os novos intervalos de IP da (s) sub-rede (s).

Excluindo o cluster

Como a nova adição modificou a VPC existente adicionando uma dependência fora da CloudFormation pilha, não é mais CloudFormation possível remover o cluster.

Antes de excluir o cluster, remova manualmente todas as sub-redes extras criadas e continue chamando: `eksctl`

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

DNS personalizado

Há duas maneiras de sobrescrever o endereço IP do servidor DNS usado para todas as pesquisas de DNS internas e externas. Isso é o equivalente à `--cluster-dns` bandeira do`kubelet`.

A primeira é pelo `clusterDNS` campo. Os arquivos de configuração aceitam um `string` campo chamado `clusterDNS` com o endereço IP do servidor DNS a ser usado. Isso será passado para o `kubelet` que, por sua vez, o passará para os pods por meio do `/etc/resolv.conf` arquivo. Para obter mais informações, consulte o esquema do arquivo de configuração.

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

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

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

Observe que essa configuração aceita somente um endereço IP. Para especificar mais de um endereço, use o `kubeletExtraConfig`parâmetro:

```
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"]
```

Manchas

Para aplicar manchas a um grupo de nós específico, use a seção de `taints` configuração como esta:

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

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

Um exemplo completo pode ser encontrado aqui.

Seletor de instâncias

O eksctl suporta a especificação de vários tipos de instância para grupos de nós gerenciados e autogerenciados, mas com mais de 270 tipos de instância do EC2, os usuários precisam dedicar algum tempo para descobrir quais tipos de instância seriam adequados para seu grupo de nós. É ainda mais difícil usar instâncias spot porque você precisa escolher um conjunto de instâncias que funcione bem em conjunto com o autoescalador de cluster.

O eksctl agora se integra ao seletor de instâncias do EC2, que resolve esse problema gerando uma lista de tipos de instância com base nos critérios de recursos: vCPUs, memory, # of e arquitetura da CPU. GPUs Quando os critérios do seletor de instância são aprovados, eksctl cria um grupo de nós com os tipos de instância definidos para os tipos de instância que correspondem aos critérios fornecidos.

Crie grupos de clusters e nós

Para criar um cluster com um único grupo de nós que usa tipos de instância correspondentes aos critérios de recursos do seletor de instâncias passados para eksctl, execute

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

Isso criará um cluster e um grupo de nós gerenciado com o `instanceTypes` campo definido como `[c5.large, c5a.large, c5ad.large, c5d.large, t2.medium, t3.medium, t3a.medium]` (o conjunto de tipos de instância retornados pode mudar).

Para grupos de nós não gerenciados, o `instancesDistribution.instanceTypes` campo será definido:

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

Os critérios do seletor de instâncias também podem ser especificados em: 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
```

As seguintes opções de CLI do seletor de instância são suportadas por `eksctl create cluster` e: `eksctl create nodegroup`

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

Um arquivo de exemplo pode ser encontrado aqui.

Corrida a seco

O recurso dry-run permite que você inspecione e altere as instâncias correspondentes ao seletor de instâncias antes de continuar com a criação de um grupo de nós.

```
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
```

O gerado ClusterConfig pode então ser passado para`eksctl create cluster`:

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

O `instanceSelector` campo que representa as opções da CLI também será adicionado ao ClusterConfig arquivo para fins de visibilidade e documentação. Quando `--dry-run` for omitido, esse campo será ignorado e o `instanceTypes` campo será usado, caso contrário, qualquer alteração será substituída pelo `instanceTypes` eksctl.

Quando um ClusterConfig arquivo é passado com`--dry-run`, eksctl exibirá um ClusterConfig arquivo contendo o mesmo conjunto de grupos de nós após expandir os critérios de recurso do seletor de instâncias de cada grupo de nós.

```
# 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
# ...
```

Instâncias spot

Grupos de nós gerenciados

`eksctl`suporta nós de trabalhadores Spot usando grupos de nós gerenciados do EKS, um recurso que permite aos clientes do EKS com aplicativos tolerantes a falhas provisionar e gerenciar facilmente instâncias spot do EC2 para seus clusters EKS. O EKS Managed Nodegroup configurará e iniciará um grupo de instâncias spot de escalonamento automático do EC2 seguindo as melhores práticas do Spot e drenando automaticamente os nós de trabalho spot antes que as instâncias sejam interrompidas pela AWS. Não há cobrança incremental para usar esse recurso e os clientes pagam somente pelo uso dos recursos da AWS, como instâncias spot do EC2 e volumes do EBS.

Para criar um cluster com um grupo de nós gerenciado usando instâncias spot, passe a `--spot` sinalização e uma lista opcional de tipos de instância:

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

Para criar um grupo de nós gerenciado usando instâncias spot em um cluster existente:

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

Para criar instâncias spot usando grupos de nós gerenciados por meio de um arquivo de configuração:

```
# 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**
>
> Grupos de nós não gerenciados não oferecem suporte aos `instanceTypes` campos `spot` e, em vez disso, o `instancesDistribution` campo é usado para configurar instâncias spot. Veja abaixo

Mais informações

- Grupos de nós EKS Spot
- Tipos de capacidade do EKS Managed Nodegroup

Grupos de nós não gerenciados

`eksctl`tem suporte para instâncias spot por meio do MixedInstancesPolicy for Auto Scaling Groups.

Aqui está um exemplo de um grupo de nós que usa 50% de instâncias spot e 50% de instâncias sob demanda:

```
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
```

Observe que o `nodeGroups.X.instanceType` campo não deve ser definido ao usar o `instancesDistribution` campo.

Este exemplo usa instâncias de GPU:

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

Este exemplo usa a estratégia de alocação spot otimizada para capacidade:

```
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"
```

Este exemplo usa a estratégia de alocação capacity-optimized-prioritized spot:

```
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"
```

Use a estratégia de `capacity-optimized-prioritized` alocação e, em seguida, defina a ordem dos tipos de instância na lista de substituições do modelo de execução da prioridade mais alta para a mais baixa (da primeira à última na lista). O Amazon EC2 Auto Scaling honra as prioridades de tipo de instância com base no melhor esforço, mas otimiza primeiro a capacidade. Essa é uma boa opção para cargas de trabalho em que a possibilidade de interrupção deve ser minimizada, mas também é importante a preferência por determinados tipos de instância. Para obter mais informações, consulte Opções de compra do ASG.

Observe que o `spotInstancePools` campo não deve ser definido ao usar o `spotAllocationStrategy` campo. Se o não `spotAllocationStrategy` for especificado, o EC2 usará a `lowest-price` estratégia como padrão.

Aqui está um exemplo mínimo:

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

Para distinguir os nós entre instâncias spot e sob demanda, você pode usar o rótulo kubernetes, `node-lifecycle` que terá o valor `spot` ou `on-demand` dependerá do tipo.

Parâmetros em InstancesDistribution

Consulte o esquema de configuração do cluster para obter detalhes.

Support para GPU

O Eksctl suporta a seleção de tipos de instância de GPU para grupos de nós. Basta fornecer um tipo de instância compatível ao comando create ou por meio do arquivo de configuração.

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

> **Note**
>
> Não é mais necessário assinar a AMI do marketplace para obter suporte à GPU no EKS.

Os resolvedores da AMI (`autoeauto-ssm`) verão que você deseja usar um tipo de instância de GPU e selecionarão a AMI acelerada otimizada para EKS correta.

O Eksctl detectará que uma AMI com um tipo de instância habilitado para GPU foi selecionada e instalará automaticamente o plug-in do dispositivo NVIDIA Kubernetes.

> **Note**
>
> O Windows e o Ubuntu AMIs não vêm com drivers de GPU instalados, portanto, executar cargas de trabalho aceleradas por GPU não funcionará imediatamente.

Para desativar a instalação automática do plug-in e instalar manualmente uma versão específica, use `--install-nvidia-plugin=false` com o comando create. Por exemplo:

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

e, para as versões 0.15.0 e superiores,

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

ou, para versões mais antigas,

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

A instalação do plug-in do dispositivo NVIDIA Kubernetes será ignorada se o cluster incluir apenas grupos de nós do Bottlerocket, pois o Bottlerocket já gerencia a execução do plug-in do dispositivo. Se você usa famílias de AMI diferentes nas configurações do seu cluster, talvez seja necessário usar manchas e tolerâncias para impedir que o plug-in do dispositivo seja executado nos nós do Bottlerocket.

Suporte ARM

Este tópico aborda como criar um cluster com um grupo de nós ARM e como adicionar um grupo de nós ARM a um cluster existente.

O EKS suporta a arquitetura ARM de 64 bits com seus processadores Graviton.
Para criar um cluster, selecione um dos tipos de instância baseados em Graviton (`a1t4g`,`m6g`,`m7g`,`m6gd`,`c6g`,`c7g`,,`c6gd`,`r6g`,`r7g`, `r6gd` `m8gr8g`,`c8g`) e execute:

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

ou use um arquivo de configuração:

```
---
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
```

O ARM também é suportado em grupos de nós gerenciados:

```
---
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
```

Os resolvedores da AMI `auto` e`auto-ssm`, inferirão a AMI correta com base no tipo de instância do ARM. Somente as famílias AmazonLinux 2023, AmazonLinux 2 e Bottlerocket têm o EKS otimizado para AMIs ARM.

> **Note**
>
> O ARM é compatível com clusters com a versão 1.15 e superior.

ajuste de escala automático

Ativar o Auto Scaling

Você pode criar um cluster (ou grupo de nós em um cluster existente) com a função do IAM que permitirá o uso do autoescalador de cluster:

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

Esse sinalizador também define `k8s.io/cluster-autoscaler/enabled` e marca`k8s.io/cluster-autoscaler/<clusterName>`, portanto, a descoberta de grupos de nós deve funcionar.

Depois que o cluster estiver em execução, você precisará instalar o próprio Cluster Autoscaler.

Você também deve adicionar o seguinte às suas definições de grupo de nós gerenciado ou não gerenciado para adicionar as tags necessárias para que o autoescalador de cluster escale o grupo de nós:

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

Aumentando a partir de 0

Se você quiser escalar seu grupo de nós a partir de 0 e tiver manchas de rótulos definidas em seus grupos de and/or nós, precisará propagá-las como tags em seus Grupos de Auto Scaling (). ASGs

Uma maneira de fazer isso é definir as tags ASG no `tags` campo de suas definições de grupo de nós. Por exemplo, dado um grupo de nós com os seguintes rótulos e manchas:

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

Você precisaria adicionar as seguintes tags ASG:

```
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"
```

Para grupos de nós gerenciados e não gerenciados, isso pode ser feito automaticamente configurando `propagateASGTags` para`true`, que adicionará os rótulos e manchas como tags ao grupo Auto Scaling:

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

```
    propagateASGTags: true
```

Auto Scaling com reconhecimento de zona

Se suas cargas de trabalho forem específicas de uma zona, você precisará criar grupos de nós separados para cada zona. Isso ocorre porque `cluster-autoscaler` pressupõe que todos os nós em um grupo sejam exatamente equivalentes. Então, por exemplo, se um evento de aumento de escala for acionado por um pod que precisa de um PVC específico da zona (por exemplo, um volume do EBS), o novo nó pode ser programado na AZ errada e o pod não será iniciado.

Você não precisará de um grupo de nós separado para cada AZ se seu ambiente atender aos seguintes critérios:

- Não há requisitos de armazenamento específicos para cada zona.
- Não é necessário PodAffinity com topologia diferente do host.
- Não é necessário NodeAffinity no rótulo da zona.
- Sem NodeSelector em um rótulo de zona.

(Leia mais aqui e aqui.)

Se você atender a todos os requisitos acima (e possivelmente a outros), deverá estar seguro com um único grupo de nós que abrange vários. AZs Caso contrário, você desejará criar grupos de nós Single-AZ separados:

ANTES DE:

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

DEPOIS DE:

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

```
    availabilityZones: ["eu-west-2b"]
```

Suporte personalizado para AMI

Configurando o ID da AMI do nó

A `--node-ami` bandeira permite vários casos de uso avançados, como usar uma AMI personalizada ou consultar a AWS em tempo real para determinar qual AMI usar. O sinalizador pode ser usado para imagens sem GPU e GPU.

O sinalizador pode usar o ID da imagem da AMI para uso explícito de uma imagem. Ele também pode usar as seguintes palavras-chave “especiais”:

Palavra-chave	Description
auto	Indica que a AMI a ser usada para os nós deve ser encontrada consultando o AWS EC2. Isso está relacionado ao resolvedor automático.
ssm automático	Indica que a AMI a ser usada para os nós deve ser encontrada consultando o AWS SSM Parameter Store.

> **Note**
>
> No momento, os grupos de nós gerenciados pelo EKS oferecem suporte apenas às seguintes famílias de AMI ao trabalhar com versões personalizadas AMIs:`AmazonLinux2023`,,`AmazonLinux2`,`Bottlerocket`,`Ubuntu2004`, e `UbuntuPro2004` `Ubuntu2204` `Ubuntu2404`

Ao `--node-ami` definir uma string de ID, `eksctl` assumirá que uma AMI personalizada foi solicitada. Para nós AmazonLinux 2 e Ubuntu, gerenciados e autogerenciados pelo EKS, isso significa que isso `overrideBootstrapCommand` é necessário. Para AmazonLinux 2023, uma vez que ele para de usar o `/etc/eks/bootstrap.sh` script para inicialização de nós, não há suporte para um processo de inicialização do nodeadm (para obter mais informações, consulte a documentação de inicialização do nó). `overrideBootstrapCommand`

Exemplos de sinalizadores CLI:

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

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

Exemplo de arquivo de configuração:

```
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>
```

A `--node-ami` bandeira também pode ser usada com`eksctl create nodegroup`.

Configurando a família AMI do nó

Eles `--node-ami-family` podem usar as seguintes palavras-chave:

Palavra-chave	Description
AmazonLinux2	Indica que a imagem EKS AMI baseada no Amazon Linux 2 deve ser usada (padrão).
AmazonLinux2023	Indica que a imagem EKS AMI baseada no Amazon Linux 2023 deve ser usada.

Palavra-chave	Description
Ubuntu 2004	Indica que a imagem do EKS AMI baseada no Ubuntu 20.04 LTS (Focal) deve ser usada (compatível com EKS . 1.29).
UbuntuPro2004	Indica que a imagem do EKS AMI baseada no Ubuntu Pro 20.04 LTS (Focal) deve ser usada (disponível para EKS >= 1.27, 1.29).
Ubuntu 2204	Indica que a imagem do EKS AMI baseada no Ubuntu 22.04 LTS (Jammy) deve ser usada (disponível para EKS >= 1.29).
UbuntuPro2204	Indica que a imagem EKS AMI baseada no Ubuntu Pro 22.04 LTS (Jammy) deve ser usada (disponível para EKS >= 1.29).
Ubuntu 2404	Indica que a imagem EKS AMI baseada no Ubuntu 24.04 LTS (Noble) deve ser usada (disponível para EKS >= 1.31).
UbuntuPro2404	Indica que a imagem EKS AMI baseada no Ubuntu Pro 24.04 LTS (Noble) deve ser usada (disponível para EKS >= 1.31).
Foguete de garrafas	Indica que a imagem EKS AMI baseada em Bottlerocket deve ser usada.
WindowsServer2019 FullContainer	Indica que a imagem da AMI do EKS baseada no contêiner completo do Windows Server 2019 deve ser usada.
WindowsServer2019 CoreContainer	Indica que a imagem da AMI do EKS baseada no contêiner principal do Windows Server 2019 deve ser usada.

Palavra-chave	Description
WindowsServer2022 FullContainer	Indica que a imagem EKS AMI baseada no Windows Server 2022 Full Container deve ser usada.
WindowsServer2022 CoreContainer	Indica que a imagem EKS AMI baseada no Windows Server 2022 Core Container deve ser usada.

Exemplo de bandeira CLI:

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

Exemplo de arquivo de configuração:

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

A `--node-ami-family` bandeira também pode ser usada com`eksctl create nodegroup`. `eksctl`exige que a família AMI seja definida explicitamente por meio do arquivo de configuração ou via sinalização da `--node-ami-family` CLI, sempre que estiver trabalhando com uma AMI personalizada.

> **Note**
>
> No momento, os grupos de nós gerenciados pelo EKS oferecem suporte apenas às seguintes famílias de AMI ao trabalhar com versões personalizadas AMIs:`AmazonLinux2023`,,`AmazonLinux2`,`Bottlerocket`,`Ubuntu2004`, e `UbuntuPro2004` `Ubuntu2204` `Ubuntu2404`

Suporte à AMI personalizada do Windows

Somente grupos de nós autogerenciados do Windows podem especificar uma AMI personalizada. `amiFamily`deve ser definido como uma família de AMI do Windows válida.

As seguintes PowerShell variáveis estarão disponíveis para o script de 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
```

Exemplo de arquivo de configuração:

```
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
```

Suporte personalizado para AMI da Bottlerocket

Para nós Bottlerocket, o não `overrideBootstrapCommand` é suportado. Em vez disso, para designar seu próprio contêiner de bootstrap, deve-se usar o `bottlerocket` campo como parte do arquivo de configuração. Por exemplo:

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

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

Nodos de trabalho do Windows

A partir da versão 1.14, o Amazon EKS oferece suporte aos nós do Windows que permitem a execução de contêineres do Windows. Além de ter nós do Windows, é necessário um nó Linux no cluster para executar o CoreDNS, pois a Microsoft ainda não oferece suporte ao modo de rede host. Assim, um cluster do Windows EKS será uma mistura de nós do Windows e pelo menos um nó do Linux. Os nós Linux são essenciais para o funcionamento do cluster e, portanto, para um cluster de nível de produção, é recomendável ter pelo menos dois nós `t2.large` Linux para HA.

> **Note**
>
> Você não precisa mais instalar o controlador de recursos VPC nos nós de trabalho do Linux para executar cargas de trabalho do Windows em clusters EKS criados após 22 de outubro de 2021. Você pode ativar o gerenciamento de endereços IP do Windows no plano de controle do EKS por meio de uma configuração do ConfigMap (consulte o link: eks/latest/userguide/windows -support.html para obter detalhes). O eksctl corrigirá automaticamente o ConfigMap para habilitar o gerenciamento de endereços IP do Windows quando um grupo de nós do Windows for criado.

Criação de um novo cluster com suporte para Windows

A sintaxe do arquivo de configuração permite criar um cluster totalmente funcional com suporte ao Windows em um único comando:

```
# 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
```

Para criar um novo cluster com o grupo de nós não gerenciado do Windows sem usar um arquivo de configuração, emita os seguintes comandos:

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

Adicionando suporte ao Windows a um cluster Linux existente

Para permitir a execução de cargas de trabalho do Windows em um cluster existente com nós Linux (família `AmazonLinux2` AMI), você precisa adicionar um grupo de nós do Windows.

O NOVO Support for Windows managed nodegroup foi adicionado (--managed=true ou omita a sinalização).

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

Para garantir que as cargas de trabalho sejam programadas no sistema operacional correto, elas devem ter um sistema operacional `nodeSelector` direcionado ao qual devem ser executadas:

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

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

Se você estiver usando um cluster mais antigo que `1.19` o `kubernetes.io/os` e, os `kubernetes.io/arch` rótulos precisarão ser substituídos por `beta.kubernetes.io/os` e`beta.kubernetes.io/arch`, respectivamente.

Mais informações

- Suporte para EKS Windows

Mapeamentos de volume adicionais

Como opção de configuração adicional, ao lidar com mapeamentos de volume, é possível configurar mapeamentos extras quando o grupo de nós é criado.

Para fazer isso, defina o campo da `additionalVolumes` seguinte forma:

```
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'
```

Para obter mais detalhes sobre a seleção de nomes de volume, consulte a documentação de nomenclatura de dispositivos. Para saber mais sobre volumes do EBS, limites de volume de instâncias ou mapeamentos de dispositivos de blocos, visite esta página.

Nodos híbridos EKS

Introdução

O AWS EKS apresenta os nós híbridos, um novo recurso que permite que você execute aplicativos locais e de ponta em uma infraestrutura gerenciada pelo cliente com os mesmos clusters, recursos e ferramentas do AWS EKS que você usa na nuvem da AWS. O AWS EKS Hybird Nodes traz uma experiência de Kubernetes gerenciada pela AWS para ambientes locais para que os clientes simplifiquem e padronizem a forma como você executa aplicativos em ambientes locais, periféricos e na nuvem. Leia mais em EKS Hybrid Nodes.

Para facilitar o suporte a esse recurso, o eksctl introduz um novo campo de nível superior chamado. `remoteNetworkConfig` Qualquer configuração relacionada ao Hybrid Nodes deve ser configurada por meio desse campo, como parte do arquivo de configuração; não há sinalizadores CLI equivalentes. Além disso, na inicialização, qualquer configuração de rede remota só pode ser configurada durante a criação do cluster e não pode ser atualizada posteriormente. Isso significa que você não poderá atualizar os clusters existentes para usar nós híbridos.

A `remoteNetworkConfig` seção do arquivo de configuração permite que você configure as duas áreas principais quando se trata de unir nós remotos aos seus clusters EKS: rede e credenciais.

Redes

O EKS Hybrid Nodes é flexível de acordo com seu método preferido de conectar sua (s) rede (s) local (s) a uma AWS VPC. Há várias opções documentadas disponíveis, incluindo o AWS Site-to-Site VPN e o AWS Direct Connect, e você pode escolher o método que melhor se adequa ao seu caso de uso. Na maioria dos métodos que você pode escolher, sua VPC será conectada a um gateway privado virtual (VGW) ou a um gateway de trânsito (TGW). Se você confiar no eksctl para criar uma VPC para você, o eksctl também configurará, dentro do escopo de sua VPC, quaisquer pré-requisitos relacionados à rede para facilitar a comunicação entre seu plano de controle EKS e os nós remotos, ou seja,

- regras SG de entrada/saída
- rotas nas tabelas de rotas das sub-redes privadas
- o anexo do gateway VPC ao determinado TGW ou VGW

Exemplo de arquivo de configuração:

```
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"]
```

Se seu método de conectividade preferido não envolver o uso de um TGW ou VGW, você não deve confiar no eksctl para criar a VPC para você e, em vez disso, fornecer uma pré-existente. Em uma observação relacionada, se você estiver usando uma VPC preexistente, a eksctl não fará nenhuma alteração nela, e garantir que todos os requisitos de rede estejam em vigor é de sua responsabilidade.

> **Note**
>
> eksctl não configura nenhuma infraestrutura de rede fora da sua AWS VPC (ou seja, qualquer infraestrutura de VGW/TGW até as redes remotas)

Credenciais

Os EKS Hybrid Nodes usam o AWS IAM Authenticator e credenciais temporárias do IAM provisionadas pelo AWS SSM ou pelo AWS IAM Roles Anywhere para se autenticar com o cluster EKS. Semelhante aos grupos de nós autogerenciados, se não for fornecido de outra forma, o eksctl criará para você uma função do IAM de nós híbridos a ser assumida pelos nós remotos. Além disso, ao usar o IAM Roles Anywhere como seu provedor de credenciais, o eksctl configurará um perfil e uma âncora confiável com base em um determinado pacote de autoridade de certificação (), por exemplo `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
```

O ARN da função de nós híbridos criada por eksctl é necessário posteriormente no processo de unir seus nós remotos ao cluster`nodeadm`, `NodeConfig` para configurar e criar ativações (se estiver usando SSM). Para buscá-lo, use:

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

Da mesma forma, se estiver usando o IAM Roles Anywhere, você pode buscar o ARN da âncora de confiança e do perfil anywhere criado por eksctl, alterando o comando anterior substituindo-o por ou, respectivamente. `RemoteNodesRoleARN` `RemoteNodesTrustAnchorARN` `RemoteNodesAnywhereProfileARN`

Se você tiver uma configuração preexistente do IAM Roles Anywhere ou estiver usando SSM, poderá fornecer uma função do IAM para nós híbridos por meio de. `remoteNetworkConfig.iam.roleARN` Lembre-se de que, nesse cenário, o eksctl não criará a âncora de confiança e o perfil em qualquer lugar para você. Por exemplo

```
remoteNetworkConfig:
```

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

Para mapear a função para uma identidade do Kubernetes e autorizar os nós remotos a se juntarem ao cluster EKS, eksctl cria uma entrada de acesso com a função IAM de nós híbridos como ARN principal e do tipo. ou seja `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"
        ]
    }
]
```

Suporte a complementos

Interface de rede de contêineres (CNI): a CNI da AWS VPC não pode ser usada com nós híbridos. Os principais recursos do Cilium e do Calico são compatíveis com o uso com nós híbridos. Você pode gerenciar a CNI com ferramentas de sua escolha, como o Helm. Para obter mais informações, consulte Configurar uma CNI para nós híbridos.

> **Note**
>
> Se você instalar o VPC CNI em seu cluster para seus grupos de nós autogerenciados ou gerenciados pelo EKS, precisará usar `v1.19.0-eksbuild.1` ou posterior, pois isso inclui uma atualização do daemonset do complemento para impedir que ele seja instalado em nós híbridos.

Referências adicionais

- Nodos híbridos EKS UserDocs
- Anúncio de lançamento

Configuração do Support Node Repair para grupos de nós gerenciados do EKS

O EKS Managed Nodegroups suporta o Node Repair, em que a integridade dos nós gerenciados é monitorada e os nós de trabalho não íntegros são substituídos ou reinicializados em resposta. O eksctl agora fornece opções de configuração abrangentes para um controle refinado sobre o comportamento de reparo dos nós.

Configuração básica de reparo de nós

Usando sinalizadores CLI

Para criar um cluster com um grupo de nós gerenciado usando o reparo básico de nós, passe a `--enable-node-repair` sinalização:

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

Para criar um grupo de nós gerenciado com reparo de nós em um cluster existente:

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

Usando arquivos de configuração

```
# 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
```

Configuração aprimorada de reparo de nós

Configuração de limite

Você pode configurar quando as ações de reparo do nó serão interrompidas usando limites baseados em porcentagem ou contagem. Nota: Você não pode usar os limites percentuais e de contagem ao mesmo tempo.

Sinalizadores CLI para limites

```
# 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
```

Arquivo de configuração para limites

```
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 reparo paralelo

Controle o número máximo de nós que podem ser reparados simultaneamente ou paralelamente. Isso proporciona um controle mais detalhado da velocidade das substituições de nós. Nota: Você não pode usar os limites de porcentagem e contagem ao mesmo tempo.

Sinalizadores CLI para limites paralelos

```
# 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
```

Arquivo de configuração para limites paralelos

```
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
```

Substituições de reparo personalizadas

Especifique substituições granulares para ações de reparo específicas. Essas substituições controlam a ação de reparo e o tempo de espera antes que um nó seja considerado apto para o processo de reparo. Se você usar isso, deverá especificar todos os valores para cada substituição.

```
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"
```

Exemplos completos de configuração

Para ver um exemplo abrangente com todas as opções de configuração, consulte examples/44-node-repair.yaml.

Exemplo 1: reparo básico com limites percentuais

```
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
```

Exemplo 2: reparo conservador para cargas de trabalho críticas

```
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"
```

Exemplo 3: carga de trabalho da GPU com reparo especializado

```
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"
```

Referência de CLI

Sinalizadores de reparo de nós

Sinalizador	Description	Exemplo
`--enable-node-repair`	Ativar o reparo automático de nós	`--enable-node-repair`

Sinalizador	Description	Exemplo
`--node-repair-max-unhealthy-percentage`	Porcentagem máxima de nós não saudáveis antes do reparo	`--node-repair-max-unhealthy-percentage=20`
`--node-repair-max-unhealthy-count`	Contagem máxima de nós não saudáveis antes do reparo	`--node-repair-max-unhealthy-count=5`
`--node-repair-max-parallel-percentage`	Porcentagem máxima de nós a serem reparados em paralelo	`--node-repair-max-parallel-percentage=15`
`--node-repair-max-parallel-count`	Contagem máxima de nós a serem reparados em paralelo	`--node-repair-max-parallel-count=2`

Observação: as substituições de configuração de reparo de nós só são suportadas por meio de arquivos de configuração YAML devido à sua complexidade.

Referência da configuração

nodeRepairConfig

Campo	Tipo	Description	Restrições	Exemplo
`enabled`	booleano	Ativar/desativar o reparo do nó	-	`true`
`maxUnhealthyNodeThresholdPercentage`	integer	Limite percentual de nós não íntegros, acima do qual as ações de reparo automático do nó serão interrompidas	Não pode ser usado com `maxUnhealthyNodeThresholdCount`	`20`
`maxUnhealthyNodeTh`	integer	Limite de contagem de nós não íntegros, acima do qual as	Não pode ser usado com	`5`

Campo	Tipo	Description	Restrições	Exemplo
`resholdCount`		ações de reparo automático do nó serão interrompidas	`maxUnhealthyNodeThresholdPercentage`	
`maxParallelNodesRepairedPercentage`	integer	Porcentagem máxima de nós não íntegros que podem ser reparados simultaneamente ou paralelamente	Não pode ser usado com `maxParallelNodesRepairedCount`	15
`maxParallelNodesRepairedCount`	integer	Contagem máxima de nós não íntegros que podem ser reparados simultaneamente ou paralelamente	Não pode ser usado com `maxParallelNodesRepairedPercentage`	2
`nodeRepairConfigOverrides`	array	Substituições granulares para ações de reparo específicas, controlando a ação de reparo e o tempo de atraso	Todos os valores devem ser especificados para cada substituição	Veja os exemplos acima

nodeRepairConfigSubstituições

Campo	Tipo	Description	Valores válidos
`nodeMonitoringCondition`	string	Condição insalubre relatada pelo agente de monitoramento de nós à qual essa substituição se aplica	`"AcceleratedInstanceNotReady" ,`

Campo	Tipo	Description	Valores válidos
			`"NetworkNotReady"`
`nodeUnhealthyReason`	string	Motivo relatado pelo agente de monitoramento de nós ao qual essa substituição se aplica	`"NvidiaXID13Error"` , `"InterfaceNotUp"`
`minRepairWaitTimeMins`	integer	Tempo mínimo de espera em minutos antes de tentar reparar um nó com a condição e o motivo especificados	Qualquer número inteiro positivo
`repairAction`	string	Ação de reparo a ser tomada para os nós quando todas as condições especificadas forem atendidas	`"Terminate"` , `"Restart"` , `"NoAction"`

Mais informações

- Grupo de nós gerenciado EKS Node Health

Redes

Este capítulo inclui informações sobre como a Eksctl cria redes Virtual Private Cloud (VPC) para clusters EKS.

Tópicos:

- Especifique a versão IP (IPv4 ou IPv6) a ser usada ao criar uma VPC com cluster EKS

Configuração da VPC

VPC dedicada para cluster

Por padrão, `eksctl create cluster` criará uma VPC dedicada para o cluster. Isso é feito para evitar interferência nos recursos existentes por vários motivos, incluindo segurança, mas também porque é difícil detectar todas as configurações em uma VPC existente.

- O CIDR VPC padrão usado por é. `eksctl` `192.168.0.0/16`
  - É dividido em 8 (`/19`) sub-redes (3 privadas, 3 públicas e 2 reservadas).
- O grupo de nós inicial é criado em sub-redes públicas.
- O acesso SSH está desativado, a menos que `--allow-ssh` seja especificado.
- Por padrão, o grupo de nós permite tráfego de entrada do grupo de segurança do plano de controle nas portas 1025 a 65535.

> **Note**
>
> No `us-east-1` eksctl, cria apenas 2 sub-redes públicas e 2 privadas por padrão.

Alterar o CIDR da VPC

Se você precisar configurar o peering com outra VPC, ou simplesmente precisar de um intervalo IPs maior ou menor, você pode `--vpc-cidr` usar o sinalizador para alterá-lo. Consulte os documentos da AWS para obter guias sobre como escolher blocos CIDR que são permitidos para uso em uma VPC da AWS.

Se você estiver criando um IPv6 cluster, poderá configurá-lo `VPC.IPv6Cidr` no arquivo de configuração do cluster. Essa configuração está somente no arquivo de configuração, não em um sinalizador CLI.

Se você possui um bloco de endereços IPv6 IP, também pode trazer seu próprio IPv6 pool. Consulte Traga seus próprios endereços IP (BYOIP) para o Amazon EC2 sobre como importar seu próprio

pool. Em seguida, use o `VPC.IPv6Cidr` no arquivo de configuração do cluster para configurar o Eksctl.

Use uma VPC existente: compartilhada com kops

Você pode usar a VPC de um cluster Kubernetes existente gerenciado pelo kops. Esse recurso é fornecido para facilitar o emparelhamento de and/or clusters de migração.

Se você já criou um cluster com kops, por exemplo, usando comandos semelhantes a este:

```
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
```

Você pode criar um cluster EKS no mesmo AZs usando as mesmas sub-redes VPC (NOTA: são necessárias pelo menos 2 AZs/subnets):

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

Use a VPC existente: outra configuração personalizada

`eksctl`fornece alguma flexibilidade, mas não completa, para topologias personalizadas de VPC e sub-rede.

Você pode usar uma VPC existente fornecendo sub-redes and/or públicas privadas usando os sinalizadores e. `--vpc-private-subnets` `--vpc-public-subnets` Cabe a você garantir que as sub-redes que você usa sejam categorizadas corretamente, pois não há uma maneira simples de verificar se uma sub-rede é realmente privada ou pública, pois as configurações variam.

Dadas essas sinalizações, `eksctl create cluster` determinará o ID da VPC automaticamente, mas não criará nenhuma tabela de roteamento ou outros recursos, como gateways. internet/NAT No entanto, ele criará grupos de segurança dedicados para o grupo de nós inicial e o plano de controle.

Você deve garantir o fornecimento de pelo menos 2 sub-redes diferentes AZs e essa condição é verificada pelo EKS. Se você usa uma VPC existente, os requisitos a seguir não são aplicados ou verificados pelo EKS ou pelo Eksctl, e o EKS cria o cluster. Algumas funções básicas do cluster funcionam sem esses requisitos. (Por exemplo, a marcação não é estritamente necessária; testes mostraram que é possível criar um cluster funcional sem nenhuma tag definida nas sub-redes, no entanto, não há garantia de que isso sempre será válido e a marcação é recomendada.)

Requisitos padrão:

- todas as sub-redes fornecidas devem estar na mesma VPC, dentro do mesmo bloco de IPs
- um número suficiente de endereços IP está disponível, com base nas necessidades
- número suficiente de sub-redes (mínimo 2), com base nas necessidades
- as sub-redes são marcadas com pelo menos o seguinte:
  - `kubernetes.io/cluster/<name>`tag definida como `shared` ou `owned`
  - `kubernetes.io/role/internal-elb`tag definida como 1 para sub-redes privadas
  - `kubernetes.io/role/elb`tag definida como 1 para sub-redes públicas
- gateways and/or NAT de internet configurados corretamente
- as tabelas de roteamento têm entradas corretas e a rede está funcional
- NOVO: todas as sub-redes públicas devem ter a propriedade `MapPublicIpOnLaunch` ativada (ou seja, `Auto-assign public IPv4 address` no console da AWS). Os grupos de nós gerenciados e o Fargate não atribuem IPv4 endereços públicos, a propriedade deve ser definida na sub-rede.

Pode haver outros requisitos impostos pelo EKS ou pelo Kubernetes, e cabe inteiramente a você cumprir quaisquer requisitos up-to-date. and/or recommendations, and implement those as needed/possible

As configurações padrão do grupo de segurança aplicadas por `eksctl` podem ou não ser suficientes para compartilhar o acesso com recursos em outros grupos de segurança. Se você quiser modificar as ingress/egress regras dos grupos de segurança, talvez seja necessário usar outra ferramenta para automatizar as alterações ou fazer isso por meio do console do EC2.

Em caso de dúvida, não use uma VPC personalizada. Usar `eksctl create cluster` sem `--vpc-*` sinalizadores sempre configurará o cluster com uma VPC dedicada totalmente funcional.

Exemplos

Crie um cluster usando uma VPC personalizada com 2x sub-redes privadas e 2x públicas:

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

ou use o seguinte arquivo de configuração equivalente:

```
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
```

Crie um cluster usando uma VPC personalizada com três sub-redes privadas e faça com que o nodegroup inicial use essas sub-redes:

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

ou use o seguinte arquivo de configuração equivalente:

```
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
```

Crie um cluster usando uma sub-rede pública VPC 4x personalizada:

```
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
```

Mais exemplos podem ser encontrados na `examples` pasta do repositório:

- usando uma VPC existente
- usando um CIDR de VPC personalizado

Grupo de segurança de nós compartilhados personalizados

`eksctl`criará e gerenciará um grupo de segurança de nós compartilhados que permite a comunicação entre nós não gerenciados e o plano de controle do cluster e os nós gerenciados.

Se, em vez disso, desejar fornecer seu próprio grupo de segurança personalizado, você pode substituir o `sharedNodeSecurityGroup` campo no arquivo de configuração:

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

Por padrão, ao criar o cluster, `eksctl` adicionará regras a esse grupo de segurança para permitir a comunicação de e para o grupo de segurança de cluster padrão criado pelo EKS. O grupo de segurança de cluster padrão é usado tanto pelo plano de controle do EKS quanto pelos grupos de nós gerenciados.

Se você quiser gerenciar as regras do grupo de segurança sozinho, você pode `eksctl` evitar a criação das regras configurando `manageSharedNodeSecurityGroupRules` como `false` no arquivo de configuração:

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

NAT Gateway

O NAT Gateway de um cluster pode ser configurado para ser`Disable`, `Single` (padrão) ou`HighlyAvailable`. A `HighlyAvailable` opção implantará um gateway NAT em cada zona de disponibilidade da região, de forma que, se uma AZ estiver inativa, os nós da outra AZs ainda possam se comunicar com a Internet.

Ele pode ser especificado por meio do sinalizador `--vpc-nat-mode` CLI ou no arquivo de configuração do cluster, como no exemplo abaixo:

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

Veja o exemplo completo aqui.

> **Note**
>
> A especificação do NAT Gateway só é suportada durante a criação do cluster. Ele não é tocado durante uma atualização do cluster.

Configurações de sub-rede

Use sub-redes privadas para o grupo de nós inicial

Se você preferir isolar o grupo de nós inicial da Internet pública, você pode usar o sinalizador. `--node-private-networking` Quando usada em conjunto com a `--ssh-access` bandeira, a porta SSH só pode ser acessada de dentro da VPC.

> **Note**
>
> O uso da `--node-private-networking` bandeira resultará no tráfego de saída para passar pelo gateway NAT usando seu IP elástico. Por outro lado, se os nós estiverem em uma sub-rede pública, o tráfego de saída não passará pelo gateway NAT e, portanto, o tráfego de saída terá o IP de cada nó individual.

Topologia de sub-rede personalizada

`eksctl`A versão `0.32.0` introduziu mais personalização da topologia de sub-rede com a capacidade de:

- Listar várias sub-redes por AZ na configuração de VPC
- Especifique sub-redes na configuração do grupo de nós

Nas versões anteriores, as sub-redes personalizadas precisavam ser fornecidas por zona de disponibilidade, o que significava que apenas uma sub-rede por AZ podia ser listada. `0.32.0`A partir das chaves de identificação, pode ser arbitrário.

```
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

Se estiver usando o AZ como chave de identificação, o `az` valor pode ser omitido.

Se estiver usando uma string arbitrária como chave de identificação, como acima, faça o seguinte:

- `id`deve ser definido (`az`e `cidr` opcional)
- ou `az` deve ser definido (`cidr`opcional)

Se um usuário especificar uma sub-rede por AZ sem especificar CIDR e ID, uma sub-rede nessa AZ será escolhida da VPC, arbitrariamente se existirem várias dessas sub-redes.

Note

Uma especificação de sub-rede completa deve ser fornecida, ou seja, ambas `public` e as `private` configurações declaradas na especificação VPC.

Os grupos de nós podem ser restritos a sub-redes nomeadas por meio da configuração. Ao especificar sub-redes na configuração do grupo de nós, use a chave de identificação conforme fornecida na especificação da VPC, não o ID da sub-rede. Por exemplo:

```
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**
>
> Somente um dos `subnets` ou `availabilityZones` pode ser fornecido na configuração do nodegroup.

Ao colocar grupos de nós dentro de uma sub-rede privada, `privateNetworking` deve ser definido como `true` no grupo de nós:

```
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
```

Consulte 24-nodegroup-subnets.yaml no repositório eksctl para ver um exemplo completo de configuração. GitHub

Acesso ao cluster

Gerenciando o acesso aos endpoints do servidor da API Kubernetes

Por padrão, um cluster EKS expõe o servidor da API Kubernetes publicamente, mas não diretamente das sub-redes da VPC (public=true, private=false). O tráfego destinado ao servidor da API de dentro da VPC deve primeiro sair das redes VPC (mas não da rede da Amazon) e depois entrar novamente para acessar o servidor da API.

O acesso ao endpoint do servidor da API Kubernetes para um cluster pode ser configurado para acesso público e privado ao criar o cluster usando o arquivo de configuração do cluster. Exemplo abaixo:

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

Há algumas ressalvas adicionais ao configurar o acesso ao endpoint da API Kubernetes:

1. O EKS não permite clusters sem o acesso público ou privado habilitado.
2. O EKS permite criar uma configuração que permite que somente o acesso privado seja habilitado, mas o eksctl não a suporta durante a criação do cluster, pois impede que o eksctl seja capaz de unir os nós de trabalho ao cluster.
3. Atualizar um cluster para ter acesso privado somente ao endpoint da API Kubernetes significa que os comandos do Kubernetes, por padrão, (por exemplo`kubectl`), e possivelmente o comando `eksctl delete clustereksctl utils write-kubeconfig`, `eksctl utils update-kube-proxy` devem ser executados na VPC do cluster.
   - Isso requer algumas mudanças em vários recursos da AWS. Para obter mais informações, consulte Endpoint do servidor da API de cluster.
   - Você pode fornecer `vpc.extraCIDRs` o que acrescentará intervalos CIDR adicionais ao ControlPlaneSecurityGroup, permitindo que sub-redes fora da VPC cheguem ao endpoint da

API kubernetes. Da mesma forma, você também pode `vpc.extraIPv6CIDRs` acrescentar intervalos de IPv6 CIDR.

Veja a seguir um exemplo de como configurar o acesso ao endpoint da API Kubernetes usando o subcomando: `utils`

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

Para atualizar a configuração usando um **`ClusterConfig`** arquivo, use:

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

Observe que, se você não passar um sinalizador, ele manterá o valor atual. Quando estiver satisfeito com as alterações propostas, adicione a `approve` sinalização para fazer a alteração no cluster em execução.

Restringindo o acesso ao endpoint da API pública EKS Kubernetes

A criação padrão de um cluster EKS expõe publicamente o servidor da API Kubernetes.

Esse recurso se aplica somente ao endpoint público. As opções de configuração de acesso ao endpoint do servidor da API não serão alteradas, e você ainda terá a opção de desativar o endpoint público para que seu cluster não seja acessível pela Internet. (Fonte: https://github.com/aws/containers-roadmap/issues/108 #issuecomment -552766489)

Para restringir o acesso ao endpoint público da API a um conjunto de CIDRs ao criar um cluster, defina o **`publicAccessCIDRs`** campo:

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

Para atualizar as restrições em um cluster existente, use:

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

Para atualizar as restrições usando um **`ClusterConfig`** arquivo, defina o novo CIDRs em **`vpc.publicAccessCIDRs`** e execute:

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

> **Important**
>
> Se a configuração `publicAccessCIDRs` e a criação de grupos de nós `privateAccess` devem ser definidas como `true` ou os "nós" IPs devem ser adicionados à lista. `publicAccessCIDRs`

Se os nós não conseguirem acessar o endpoint da API do cluster devido ao acesso restrito, a criação do cluster falhará `context deadline exceeded` porque os nós não conseguirão acessar o endpoint público e não conseguirem ingressar no cluster.

Para atualizar o acesso ao endpoint do servidor de API e o acesso público CIDRs para um cluster em um único comando, execute:

```
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
```

Para atualizar a configuração usando um arquivo de configuração:

```
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
```

Atualizando sub-redes e grupos de segurança do plano de controle

Esta documentação explica como modificar a configuração de rede do plano de controle do seu cluster EKS após a criação inicial. Isso inclui atualizar as sub-redes do plano de controle e os grupos de segurança.

Atualizando sub-redes do plano de controle

Quando um cluster é criado com eksctl, um conjunto de sub-redes públicas e privadas é criado e passado para a API EKS. O EKS cria de 2 a 4 interfaces de rede elástica entre contas (ENIs) nessas sub-redes para permitir a comunicação entre o plano de controle do Kubernetes gerenciado pelo EKS e sua VPC.

Para atualizar as sub-redes usadas pelo plano de controle EKS, execute:

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

Para atualizar a configuração usando um arquivo de configuração:

```
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
```

Sem o `--approve` sinalizador, o eksctl registra apenas as alterações propostas. Quando estiver satisfeito com as alterações propostas, execute novamente o comando com o `--approve` sinalizador.

Atualizando grupos de segurança do plano de controle

Para gerenciar o tráfego entre o plano de controle e os nós de trabalho, o EKS suporta a transmissão de grupos de segurança adicionais que são aplicados às interfaces de rede entre contas provisionadas pelo EKS. Para atualizar os grupos de segurança do plano de controle EKS, execute:

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

Para atualizar a configuração usando um arquivo de configuração:

```
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
```

Para atualizar as sub-redes do plano de controle e os grupos de segurança de um cluster, execute:

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

Para atualizar os dois campos usando um arquivo de configuração:

```
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
```

Para ver um exemplo completo, consulte cluster-subnets-sgs.yaml.

Sem o `--approve` sinalizador, o eksctl registra apenas as alterações propostas. Quando estiver satisfeito com as alterações propostas, execute novamente o comando com o `--approve` sinalizador.

IPv6 Support

Defina a família IP

Ao `eksctl` criar uma vpc, você pode definir a versão IP que será usada. As seguintes opções estão disponíveis para serem configuradas:

- IPv4
- IPv6

O valor padrão é `IPv4`.

Para defini-lo, use o exemplo a seguir:

```
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**
>
> Essa configuração está somente no arquivo de configuração, não em um sinalizador CLI.

Se você usa IPv6, você deve configurar os seguintes requisitos:

- O OIDC está habilitado
- complementos gerenciados são definidos conforme mostrado acima
- a versão do cluster deve ser => 1.21
- A versão do complemento vpc-cni deve ser => 1.10.0
- grupos de nós autogerenciados não são compatíveis com clusters IPv6
- grupos de nós gerenciados não são compatíveis com clusters sem propriedade IPv6
- `vpc.nate` `serviceIPv4CIDR` os campos são criados pelo eksctl para clusters ipv6 e não são opções de configuração suportadas
- AutoAllocateIPv6 não é suportado junto com IPv6
- Para o IPv6 cluster, a função do IAM para vpc-cni deve ter políticas do IAM necessárias para o modo associado IPv6

A rede privada também pode ser feita com a família IPv6 IP. Siga as instruções descritas em EKS Private Cluster.

IAM

Este capítulo inclui informações sobre como trabalhar com o AWS IAM.

Tópicos:

- the section called “Gerencie usuários e funções do IAM”
  - Gerencie mapeamentos de usuários e funções do IAM para controlar o acesso a um cluster EKS
  - Configure mapeamentos de identidade do IAM por meio do arquivo de configuração do cluster ou dos comandos da CLI
- the section called “Funções do IAM para contas de serviço”
  - Gerencie permissões refinadas para aplicativos executados no Amazon EKS que usam outros serviços da AWS
  - Crie e configure funções do IAM e pares de contas de serviço do Kubernetes usando eksctl
  - Ative o IAM OpenID Connect Provider para um cluster EKS para habilitar funções do IAM para contas de serviço
- the section called “Limite de permissões do IAM”
  - Controle o máximo de permissões concedidas às entidades do IAM (usuários ou funções) definindo um limite de permissões
- the section called “Associações de identidade do EKS Pod”
  - Configure as permissões do IAM para complementos do EKS usando associações de identidade de pod recomendadas
  - Permita que os aplicativos Kubernetes recebam as permissões do IAM necessárias para se conectar aos serviços da AWS fora do cluster
  - Simplifique o processo de automatização de funções e contas de serviço do IAM em vários clusters EKS
- the section called “Políticas do IAM”
  - Gerencie políticas do IAM para grupos de nós do EKS, incluindo suporte para várias políticas complementares, como criador de imagens, escalador automático, DNS externo, gerenciador de certificados e muito mais.
  - Anexe funções de instância personalizadas ou políticas embutidas a grupos de nós para obter permissões adicionais.

- Anexe políticas específicas gerenciadas pela AWS pelo ARN aos grupos de nós, garantindo que as políticas necessárias, como Amazon e EKSWorker NodePolicy Amazoneks_CNI_Policy, sejam incluídas.

- the section called “Políticas mínimas de IAM”
  - Gerencie recursos do AWS EC2, incluindo balanceadores de carga, grupos de auto-scaling e monitoramento CloudWatch
  - Crie e gerencie CloudFormation pilhas da AWS
  - Gerencie clusters, grupos de nós e recursos relacionados do Amazon Elastic Kubernetes Service (EKS), como funções e políticas do IAM

Políticas mínimas de IAM

Este documento descreve as políticas mínimas do IAM necessárias para executar os principais casos de uso do eksctl. Esses são os usados para executar os testes de integração.

> **Note**
>
> Lembre-se de `<account_id>` substituir pelo seu.

> **Note**
>
> Uma política gerenciada da AWS é criada e administrada pela AWS. Você não pode alterar as permissões definidas nas políticas gerenciadas pela AWS.

Amazon EC2 FullAccess (política gerenciada da AWS)

Veja a definição da EC2 FullAccess política da Amazon.

AWSCloudFormationFullAccess (Política gerenciada da AWS)

Veja a definição AWSCloud FormationFullAccess da política.

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 de permissões do IAM

Um limite de permissões é um recurso avançado do AWS IAM no qual foram definidas as permissões máximas que uma política baseada em identidade pode conceder a uma entidade do IAM; onde essas entidades são usuários ou funções. Quando um limite de permissões é definido para uma entidade, essa entidade só pode realizar as ações permitidas por suas políticas baseadas em identidade e seus limites de permissões.

Você pode fornecer seu limite de permissões para que todas as entidades baseadas em identidade criadas pelo eksctl sejam criadas dentro desse limite. Este exemplo demonstra como um limite de permissões pode ser fornecido às várias entidades baseadas em identidade criadas pelo 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

Não é possível fornecer um ARN de função e um limite de permissões.

Definindo o limite de permissão da VPC CNI

Observe que quando você cria um cluster com o OIDC habilitado, o eksctl cria automaticamente um para o VPC-CNI `iamserviceaccount` por motivos de segurança. Se você quiser adicionar um limite de permissão a ele, você deve especificá-lo manualmente `iamserviceaccount` em seu arquivo de configuração:

```
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"
```

Políticas do IAM

Você pode anexar funções de instância a grupos de nós. As cargas de trabalho em execução no nó receberão permissões do IAM do nó. Para obter mais informações, consulte Funções do IAM para o Amazon EC2.

Esta página lista os modelos predefinidos de políticas do IAM disponíveis em eksctl. Esses modelos simplificam o processo de conceder aos nós do EKS as permissões apropriadas do serviço da AWS sem precisar criar manualmente políticas personalizadas do IAM.

Políticas complementares do IAM compatíveis

Exemplo de todas as políticas complementares suportadas:

```
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
```

Política do Image Builder

A `imageBuilder` política permite acesso total ao ECR (Elastic Container Registry). Isso é útil para criar, por exemplo, um servidor de CI que precisa enviar imagens para o ECR.

Política do EBS

A `ebs` política ativa o novo driver EBS CSI (Elastic Block Store Container Storage Interface).

Política do Cert Manager

A `certManager` política permite adicionar registros ao Route 53 para resolver o DNS01 desafio. Mais informações podem ser encontradas aqui.

Adicionar uma função de instância personalizada

Este exemplo cria um grupo de nós que reutiliza uma função de instância do IAM existente de outro 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"
```

Anexando políticas em linha

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

Anexando políticas pelo 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

Se um grupo de nós incluir `oattachPolicyARNs`, ele também deverá incluir as políticas de nós padrão`AmazonEKSWorkerNodePolicy`, como `AmazonEKS_CNI_Policy` e `AmazonEC2ContainerRegistryPullOnly` neste exemplo.

Gerencie usuários e funções do IAM

Note

A AWS sugere migrar para the section called “Associações de identidade do EKS Pod” a partir do. `aws-auth` ConfigMap

Os clusters EKS usam usuários e funções do IAM para controlar o acesso ao cluster. As regras são implementadas em um mapa de configuração

Editar ConfigMap com um comando CLI

chamado`aws-auth`. `eksctl`fornece comandos para ler e editar esse mapa de configuração.

Obtenha todos os mapeamentos de identidade:

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

Obtenha todos os mapeamentos de identidade correspondentes a um arn:

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

Crie um mapeamento de identidade:

```
 eksctl create iamidentitymapping --cluster  <clusterName> --region=<region> --arn
 arn:aws:iam::123456:role/testing --group system:masters --username admin
```

Exclua um mapeamento de identidade:

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

> **Note**
>
> O comando acima exclui um único FIFO de mapeamento, a menos que `--all` seja fornecido. Nesse caso, ele remove todas as correspondências. Avisará se forem encontrados mais mapeamentos que correspondam a essa função.

Crie um mapeamento de conta:

```
 eksctl create iamidentitymapping --cluster  <clusterName> --region=<region> --account
 user-account
```

Exclua um mapeamento de conta:

```
 eksctl delete iamidentitymapping --cluster  <clusterName> --region=<region> --account
 user-account
```

Editar ConfigMap usando um ClusterConfig arquivo

Os mapeamentos de identidade também podem ser especificados em: 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
```

Funções do IAM para contas de serviço

> **Tip**
>
> `eksctl`suporta a configuração de permissões refinadas para aplicativos em execução do EKS por meio do EKS Pod Identity Associations

O Amazon EKS oferece suporte aqui para Roles for Service Accounts (IRSA)], que permitem que operadores de cluster mapeiem funções do AWS IAM para contas de serviço do Kubernetes.

Isso fornece gerenciamento de permissões refinado para aplicativos que são executados no EKS e usam outros serviços da AWS. Podem ser aplicativos que usam o S3, qualquer outro serviço de dados (RDS, MQ, STS, DynamoDB) ou componentes do Kubernetes, como o controlador AWS Load Balancer ou o ExternalDNS.

Você pode criar facilmente pares de funções e contas de serviço do IAM com `eksctl`.

> **Note**
>
> Se você usou funções de instância e está pensando em usar o IRSA em vez disso, não deve misturar as duas.

Como funciona

Ele funciona por meio do IAM OpenID Connect Provider (OIDC) que o EKS expõe, e as funções do IAM devem ser construídas com referência ao provedor IAM OIDC (específico para um determinado cluster EKS) e uma referência à conta de serviço do Kubernetes à qual ele será vinculado. Depois que uma função do IAM é criada, uma conta de serviço deve incluir o ARN dessa função como uma anotação (). `eks.amazonaws.com/role-arn` Por padrão, a conta de serviço será criada ou atualizada para incluir a anotação da função. Isso pode ser desativado usando o sinalizador. `--role-only`

Dentro do EKS, há um controlador de admissão que injeta credenciais de sessão da AWS em pods, respectivamente, das funções com base na anotação na conta de serviço usada pelo pod. As credenciais serão expostas por `AWS_ROLE_ARN` & variáveis de `AWS_WEB_IDENTITY_TOKEN_FILE` ambiente. Como uma versão recente do AWS SDK é usada (veja aqui os detalhes da versão exata), o aplicativo usará essas credenciais.

No nome `eksctl` do recurso está iamserviceaccount, que representa um par de funções e contas de serviço do IAM.

Uso da CLI

> **Note**
>
> As funções do IAM para contas de serviço exigem a versão 1.13 ou superior do Kubernetes.

O provedor IAM OIDC não está habilitado por padrão. Você pode usar o comando a seguir para ativá-lo ou usar o arquivo de configuração (veja abaixo):

```
eksctl utils associate-iam-oidc-provider --cluster=<clusterName>
```

Depois de ter o provedor IAM OIDC associado ao cluster, para criar uma função do IAM vinculada a uma conta de serviço, execute:

```
eksctl create iamserviceaccount --cluster=<clusterName> --name=<serviceAccountName> --
namespace=<serviceAccountNamespace> --attach-policy-arn=<policyARN>
```

> **Note**
>
> Você pode especificar `--attach-policy-arn` várias vezes para usar mais de uma política.

Mais especificamente, você pode criar uma conta de serviço com acesso somente de leitura ao S3 executando:

```
eksctl create iamserviceaccount --cluster=<clusterName> --name=s3-read-only --attach-
policy-arn=arn:aws:iam::aws:policy/AmazonS3ReadOnlyAccess
```

Por padrão, ele será criado no `default` namespace, mas você pode especificar qualquer outro namespace, por exemplo:

```
eksctl create iamserviceaccount --cluster=<clusterName> --name=s3-read-only --
namespace=s3-app --attach-policy-arn=arn:aws:iam::aws:policy/AmazonS3ReadOnlyAccess
```

> **Note**
>
> Se o namespace ainda não existir, ele será criado.

Se você já tiver uma conta de serviço criada no cluster (sem uma função do IAM), precisará usar a `--override-existing-serviceaccounts` sinalização.

A marcação personalizada também pode ser aplicada à função do IAM `--tags` especificando:

```
eksctl create iamserviceaccount --cluster=<clusterName> --name=<serviceAccountName> --
tags "Owner=John Doe,Team=Some Team"
```

CloudFormation gerará um nome de função que inclui uma string aleatória. Se você preferir um nome de função predeterminado, você pode especificar`--role-name`:

```
eksctl create iamserviceaccount --cluster=<clusterName> --name=<serviceAccountName> --
role-name "custom-role-name"
```

Quando a conta de serviço é criada e gerenciada por alguma outra ferramenta, como o helm, use `--role-only` para evitar conflitos. A outra ferramenta é então responsável por manter a anotação do ARN da função. Observe que não `--override-existing-serviceaccounts` tem efeito nas contas `roleOnly` `--role-only` /service, a função sempre será criada.

```
eksctl create iamserviceaccount --cluster=<clusterName> --name=<serviceAccountName> --
role-only --role-name=<customRoleName>
```

Quando você tem uma função existente que deseja usar com uma conta de serviço, pode fornecer o `--attach-role-arn` sinalizador em vez de fornecer as políticas. Para garantir que a função só possa ser assumida pela conta de serviço especificada, você deve definir um documento de política de relacionamento aqui].

```
eksctl create iamserviceaccount --cluster=<clusterName> --name=<serviceAccountName> --
attach-role-arn=<customRoleARN>
```

Para atualizar as funções de uma conta de serviço, as permissões que você pode executar`eksctl update iamserviceaccount`.

> **Note**
>
> `eksctl delete iamserviceaccount`exclui o Kubernetes `ServiceAccounts` mesmo que eles não tenham sido criados por. `eksctl`

Uso com arquivos de configuração

Para gerenciar `iamserviceaccounts` usando o arquivo de configuração, você deverá definir `iam.withOIDC: true` e listar a conta que deseja usar. `iam.serviceAccount`

Todos os comandos suportam`--config-file`, você pode gerenciar iamserviceaccounts da mesma forma que nodegroups. O `eksctl create iamserviceaccount` comando suporta `--include` e `--exclude` sinaliza (consulte esta seção para obter mais detalhes sobre como eles funcionam). E o `eksctl delete iamserviceaccount` comando `--only-missing` também suporta, então você pode realizar exclusões da mesma forma que os grupos de nós.

> **Note**
>
> As contas de serviço do IAM têm como escopo um namespace, ou seja, duas contas de serviço com o mesmo nome podem existir em namespaces diferentes. Portanto, para definir de forma exclusiva uma conta de serviço como parte dos `--include` `--exclude` sinalizadores, você precisará passar a string do nome no formato. `namespace/name` Por exemplo:

```
eksctl create iamserviceaccount --config-file=<path> --include backend-apps/s3-reader
```

A opção de habilitar `wellKnownPolicies` está incluída para usar o IRSA com casos de uso conhecidos, como `cluster-autoscaler` `ecert-manager`, como uma abreviatura para listas de políticas.

As políticas conhecidas suportadas e outras propriedades do `serviceAccounts` estão documentadas no esquema de configuração.

Você usa o seguinte exemplo de configuração com`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
```

Se você criar um cluster sem esses campos definidos, poderá usar os seguintes comandos para habilitar tudo o que você precisa:

```
eksctl utils associate-iam-oidc-provider --config-file=<path>
eksctl create iamserviceaccount --config-file=<path>
```

Mais informações

- Apresentando funções refinadas do IAM para contas de serviço
- Guia do usuário do EKS - Funções do IAM para contas de serviço
- Mapeamento de usuários e funções do IAM para funções RBAC do Kubernetes

Associações de identidade do EKS Pod

O AWS EKS introduziu um novo mecanismo aprimorado chamado Pod Identity Association para administradores de cluster configurarem aplicativos Kubernetes para receber as permissões do IAM necessárias para se conectar aos serviços da AWS fora do cluster. A Pod Identity Association aproveita o IRSA, mas o torna configurável diretamente por meio da API EKS, eliminando totalmente a necessidade de usar a API IAM.

Como resultado, as funções do IAM não precisam mais fazer referência a um provedor OIDC e, portanto, não estarão mais vinculadas a um único cluster. Isso significa que as funções do IAM agora podem ser usadas em vários clusters do EKS sem a necessidade de atualizar a política de confiança da função sempre que um novo cluster é criado. Isso, por sua vez, elimina a necessidade de duplicação de funções e simplifica completamente o processo de automatização do IRSA.

Pré-requisitos

Nos bastidores, a implementação de associações de identidade de pod está executando um agente como daemonset nos nós de trabalho. Para executar o agente pré-requisito no cluster, o EKS fornece um novo complemento chamado EKS Pod Identity Agent. Portanto, criar associações de identidade de pod (em geral e com`eksctl`) requer o `eks-pod-identity-agent` complemento pré-instalado no cluster. Esse complemento pode ser criado usando `eksctl` da mesma forma que qualquer outro complemento compatível.

```
eksctl create addon --cluster my-cluster --name eks-pod-identity-agent
```

Além disso, se estiver usando uma função preexistente do IAM ao criar uma associação de identidade de pod, você deve configurar a função para confiar no recém-introduzido EKS service principal (`pods.eks.amazonaws.com`). Um exemplo de política de confiança do IAM pode ser encontrado abaixo:

```
{
```

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

Se, em vez disso, você não fornecer o ARN de uma função existente ao comando create, `eksctl` criará uma nos bastidores e configurará a política de confiança acima.

Criação de associações de identidade de pod

Para manipular associações de identidade de pod, `eksctl` adicionou um novo campo abaixo`iam.podIdentityAssociations`, por exemplo

```
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
```

Para ver um exemplo completo, consulte pod-identity-associations.yaml.

> **Note**
>
> Além de ser `permissionPolicy` usado como um documento de política embutido, todos os outros campos têm uma contrapartida de bandeira CLI.

A criação de associações de identidade de pod pode ser feita das seguintes maneiras. Durante a criação do cluster, especificando as associações de identidade do pod desejadas como parte do arquivo de configuração e executando:

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

Pós-criação do cluster, usando um arquivo de configuração, por exemplo

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

OU usando sinalizadores CLI, por exemplo

```
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**
>
> Somente uma única função do IAM pode ser associada a uma conta de serviço por vez. Portanto, tentar criar uma segunda associação de identidade de pod para a mesma conta de serviço resultará em um erro.

Buscando associações de identidade do Pod

Para recuperar todas as associações de identidade de pod para um determinado cluster, execute um dos seguintes comandos:

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

OU

```
eksctl get podidentityassociation --cluster my-cluster
```

Além disso, para recuperar somente as associações de identidade do pod em um determinado namespace, use o `--namespace` sinalizador, por exemplo

```
eksctl get podidentityassociation --cluster my-cluster --namespace default
```

Por fim, para recuperar uma única associação, correspondente a uma determinada conta de serviço K8s, inclua também o comando `--service-account-name` para o comando acima, ou seja,

```
eksctl get podidentityassociation --cluster my-cluster --namespace default --service-
account-name s3-reader
```

Atualizando associações de identidade do pod

Para atualizar a função do IAM de uma ou mais associações de identidade de pod, passe a nova `roleARN(s)` para o arquivo de configuração, por exemplo

```
iam:
  podIdentityAssociations:
    - namespace: default
      serviceAccountName: s3-reader
      roleARN: new-role-arn-1
    - namespace: dev
      serviceAccountName: app-cache-access
      roleARN: new-role-arn-2
```

e execute:

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

OU (para atualizar uma única associação) passe a nova `--role-arn` por meio de sinalizadores CLI:

```
eksctl update podidentityassociation --cluster my-cluster --namespace default --
service-account-name s3-reader --role-arn new-role-arn
```

Excluindo associações de identidade do pod

Para excluir uma ou mais associações de identidade de pod, passe `namespace(s)` e `serviceAccountName(s)` para o arquivo de configuração, por exemplo

```
iam:
  podIdentityAssociations:
    - namespace: default
      serviceAccountName: s3-reader
    - namespace: dev
      serviceAccountName: app-cache-access
```

e execute:

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

OU (para excluir uma única associação) passe os sinalizadores `--namespace` e `--service-account-name` por meio da CLI:

```
eksctl delete podidentityassociation --cluster my-cluster --namespace default --
service-account-name s3-reader
```

Suporte de complementos do EKS para associações de identidade de pods

Os complementos do EKS também oferecem suporte ao recebimento de permissões do IAM por meio do EKS Pod Identity Associations. O arquivo de configuração expõe três campos que permitem configurá-los: e. `addon.podIdentityAssociations` `addonsConfig.autoApplyPodIdentityAssociations` `addon.useDefaultPodIdentityAssociations` Você pode configurar explicitamente as associações de identidade de pod desejadas`addon.podIdentityAssociations`, usando ou fazer com que resolva (e aplique) `eksctl` automaticamente a configuração de identidade de pod recomendada, usando `addonsConfig.autoApplyPodIdentityAssociations` ou`addon.useDefaultPodIdentityAssociations`.

> **Note**
>
> Nem todos os complementos do EKS suportarão associações de identidade de pod no lançamento. Nesse caso, as permissões necessárias do IAM continuarão sendo fornecidas usando as configurações do IRSA.

Criação de complementos com permissões do IAM

Ao criar um complemento que exija permissões do IAM, primeiro `eksctl` verificará se as associações de identidade do pod ou as configurações do IRSA estão sendo configuradas explicitamente como parte do arquivo de configuração e, em caso afirmativo, use uma delas para configurar as permissões do complemento. por exemplo

```
addons:
- name: vpc-cni
  podIdentityAssociations:
  - serviceAccountName: aws-node
    permissionPolicyARNs: ["arn:aws:iam::aws:policy/AmazonEKS_CNI_Policy"]
```

e corra

```
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**
>
> Não é permitido definir as identidades do pod e do IRSA ao mesmo tempo e resultará em um erro de validação.

Para complementos do EKS que suportam identidades de pod, `eksctl` oferece a opção de configurar automaticamente todas as permissões recomendadas do IAM, na criação do complemento. Isso pode ser feito simplesmente configurando o arquivo `addonsConfig.autoApplyPodIdentityAssociations: true` de configuração. por exemplo

```
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
```

e corra

```
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
```

Da mesma forma, o mesmo pode ser feito por meio de sinalizadores CLI, por exemplo

```
eksctl create addon --cluster my-cluster --name vpc-cni --auto-apply-pod-identity-
associations
```

Para migrar um complemento existente para usar a identidade do pod com as políticas recomendadas do IAM, use

```
addons:
- name: vpc-cni
  useDefaultPodIdentityAssociations: true
```

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

Atualização de complementos com permissões do IAM

Ao atualizar um complemento, a especificação `addon.PodIdentityAssociations` representará a única fonte confiável para o estado que o complemento deverá ter, após a conclusão da operação de atualização. Nos bastidores, diferentes tipos de operações são realizados para atingir o estado desejado, ou seja,

- criar identidades de pod que estão presentes no arquivo de configuração, mas ausentes no cluster
- exclua as identidades de pod existentes que foram removidas do arquivo de configuração, junto com quaisquer recursos do IAM associados
- atualize as identidades de pod existentes que também estão presentes no arquivo de configuração e para as quais o conjunto de permissões do IAM foi alterado

> **Note**
>
> O ciclo de vida das associações de identidade de pods pertencentes ao EKS Add-ons é gerenciado diretamente pela API EKS Addons.

Você não pode usar `eksctl update podidentityassociation` (para atualizar as permissões do IAM) ou `eksctl delete podidentityassociations` (para remover a associação) para associações usadas com um complemento do Amazon EKS. Em vez disso, `eksctl update addon` ou `eksctl delete addon` deve ser usado.

Vamos ver um exemplo acima, começando analisando a configuração inicial da identidade do pod para o complemento:

```
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"
    }
]
```

Agora use a configuração abaixo:

```
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
```

e corra

```
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
```

agora verifique se a configuração de identidade do pod foi atualizada corretamente

```
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"
    }
]
```

Para remover todas as associações de identidade de pod de um complemento, ele `addon.PodIdentityAssociations` deve ser definido explicitamente como, por exemplo `[]`

```
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: []
```

e corra

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

Excluindo complementos com permissões do IAM

A exclusão de um complemento também removerá todas as identidades de pods associadas ao complemento. A exclusão do cluster terá o mesmo efeito para todos os complementos. Todas as funções do IAM para identidades de pods, criadas por`eksctl`, também serão excluídas.

Migração de contas e complementos iamserviceaccounts existentes para associações de identidade de pod

Há um comando `eksctl` utils para migrar funções do IAM existentes para contas de serviço para associações de identidade de pod, ou seja,

```
eksctl utils migrate-to-pod-identity --cluster my-cluster --approve
```

Nos bastidores, o comando aplicará as seguintes etapas:

- instale o `eks-pod-identity-agent` complemento se ainda não estiver ativo no cluster
- identificar todas as funções do IAM associadas a iamserviceaccounts
- identifique todas as funções do IAM associadas aos complementos do EKS que oferecem suporte a associações de identidade de pod
- atualize a política de confiança do IAM de todas as funções identificadas, com uma entidade confiável adicional, apontando para o novo diretor do EKS Service (e, opcionalmente, remova a relação de confiança existente com o provedor do OIDC)
- criar associações de identidade de pod para funções filtradas associadas a iamserviceaccounts
- atualize os complementos do EKS com identidades do pod (a API do EKS criará as identidades do pod nos bastidores)

Executar o comando sem o `--approve` sinalizador produzirá apenas um plano que consiste em um conjunto de tarefas que refletem as etapas acima, por exemplo

```
[#]  (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
```

A relação de confiança existente com o provedor do OIDC está sempre sendo removida das funções do IAM associadas aos complementos do EKS. Além disso, para remover a relação de confiança existente do provedor OIDC das funções do IAM associadas a iamserviceaccounts, execute o comando com flag, por exemplo `--remove-oidc-provider-trust-relationship`

```
eksctl utils migrate-to-pod-identity --cluster my-cluster --approve --remove-oidc-
provider-trust-relationship
```

Cross Account Pod Identity Support

O eksctl suporta o acesso entre contas do EKS Pod Identity. Esse recurso permite que pods executados em seu cluster EKS acessem recursos da AWS em uma conta diferente da AWS.

Usage

Para criar uma associação de identidade de pod com acesso entre contas, primeiro configure as funções e políticas do IAM que permitam o acesso de uma conta de origem da AWS (com o cluster) a uma conta da AWS de destino (com os recursos que o cluster pode acessar). Para ver um exemplo disso, consulte "O Amazon EKS Pod Identity simplifica o acesso entre contas".

Depois que uma função do IAM estiver configurada em cada conta, use eksctl para criar as associações de identidade do 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
```

Referências adicionais

Suporte oficial de complementos do AWS Userdocs for EKS para identidades de pods

Publicação oficial do blog da AWS sobre associações de identidade de pods

Documentos de usuário oficiais da AWS para associações de identidade de pods

Opções de implantação

Este capítulo aborda o uso do eksctl para gerenciar clusters EKS implantados em ambientes alternativos.

Para obter as informações mais precisas sobre as opções de implantação do EKS, consulte Implantar clusters do Amazon EKS em ambientes locais e na nuvem no Guia do usuário do EKS.

Tópicos:

- the section called “EKS em qualquer lugar”
  - Use eksctl com clusters do Amazon EKS Anywhere.
  - O Amazon EKS Anywhere é um software de gerenciamento de contêineres criado pela AWS que facilita a execução e o gerenciamento do Kubernetes no local e na borda.
- the section called “Suporte ao AWS Outposts Support”
  - Use eksctl com clusters EKS no AWS Outposts.
  - O AWS Outposts é uma família de soluções totalmente gerenciadas que fornece infraestrutura e serviços da AWS para praticamente qualquer local local ou ponto de presença para uma experiência híbrida verdadeiramente consistente.
  - O suporte do AWS Outposts no eksctl permite criar clusters locais com todo o cluster Kubernetes, incluindo o plano de controle do EKS e os nós de trabalho, executados localmente no AWS Outposts.
- the section called “Nodos híbridos EKS”
  - Execute aplicativos locais e de ponta em uma infraestrutura gerenciada pelo cliente com os mesmos clusters, recursos e ferramentas do AWS EKS que você usa na nuvem da AWS.

EKS em qualquer lugar

`eksctl`fornece acesso ao recurso da AWS chamado `EKS Anywhere` com o subcomando. `eksctl anywhere` Isso requer o `eksctl-anywhere` binário presente`PATH`. Siga as instruções descritas aqui Instale o eksctl-anywhere para instalá-lo.

Uma vez feito isso, execute os comandos em qualquer lugar executando:

```
eksctl anywhere version
```

```
v0.5.0
```

Para obter mais informações sobre o EKS Anywhere, visite o site do EKS Anywhere.

Suporte ao AWS Outposts Support

> **Warning**
>
> Os grupos de nós gerenciados do EKS não são suportados no Outposts.

Estendendo clusters existentes para o AWS Outposts

Você pode estender um cluster EKS existente em execução em uma região da AWS para o AWS Outposts configurando novos grupos de nós `nodeGroup.outpostARN` para criar grupos de nós em Outposts, como em:

```
# 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
```

Nessa configuração, o plano de controle do EKS é executado em uma região da AWS, enquanto os grupos de nós com `outpostARN` set são executados no Outpost especificado. Quando um grupo de

nós está sendo criado em Outposts pela primeira vez, eksctl estende a VPC criando sub-redes no Outpost especificado. Essas sub-redes são usadas para criar grupos de nós definidos. `outpostARN`

Os clientes com uma VPC preexistente precisam criar as sub-redes no Outposts e transmiti-las, como em: `nodeGroup.subnets`

```
# 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"
```

Criação de um cluster local no AWS Outposts

> **Note**
>
> Os clusters locais somente são compatíveis com racks do Outpost.

> **Note**
>
> Somente o Amazon Linux 2 é compatível com grupos de nós quando o plano de controle está em Outposts. Somente os tipos de volume EBS gp2 são suportados para grupos de nós em Outposts.

O suporte do AWS Outposts no eksctl permite criar clusters locais com todo o cluster Kubernetes, incluindo o plano de controle do EKS e os nós de trabalho, executados localmente no AWS Outposts. Os clientes podem criar um cluster local com o plano de controle do EKS e os nós de trabalho em execução local no AWS Outposts, ou podem estender um cluster EKS existente em execução em uma região da AWS para o AWS Outposts criando nós de trabalho no Outposts.

Para criar o plano de controle e os grupos de nós do EKS no AWS Outposts, `outpost.controlPlaneOutpostARN` defina como o Outpost ARN, como em:

```
# 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
```

Isso instrui o eksctl a criar o plano de controle e as sub-redes do EKS no Outpost especificado. Como existe um rack do Outposts em uma única zona de disponibilidade, o eksctl cria somente uma sub-rede pública e privada. O eksctl não associa a VPC criada a um gateway local e, portanto, o eksctl não terá conectividade com o servidor da API e não poderá criar grupos de nós. Portanto, se o `ClusterConfig` contiver algum grupo de nós durante a criação do cluster, o comando deverá ser executado com`--without-nodegroup`, como em:

```
eksctl create cluster -f outpost.yaml --without-nodegroup
```

É responsabilidade do cliente associar a VPC criada pelo eksctl ao gateway local após a criação do cluster para permitir a conectividade com o servidor da API. Após essa etapa, grupos de nós podem ser criados usando o. `eksctl create nodegroup`

Opcionalmente, você pode especificar o tipo de instância para os nós do plano de controle em `outpost.controlPlaneInstanceType` ou para os grupos de nós em`nodeGroup.instanceType`, mas o tipo de instância deve existir no Outpost ou eksctl retornará um erro. Por padrão, o eksctl tenta escolher o menor tipo de instância disponível no Outpost para os nós e grupos de nós do plano de controle.

Quando o plano de controle está em Postos Avançados, grupos de nós são criados nesse Posto Avançado. Opcionalmente, você pode especificar o ARN do Posto Avançado para o grupo de nós, `nodeGroup.outpostARN` mas ele deve corresponder ao ARN do Posto Avançado do plano de controle.

```
# 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 existente

Clientes com uma VPC existente podem criar clusters locais no AWS Outposts especificando a configuração da sub-rede em, como em: `vpc.subnets`

```
# 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
```

As sub-redes devem existir no Outpost especificado em `outpost.controlPlaneOutpostARN` ou o eksctl retornará um erro. Você também pode especificar grupos de nós durante a criação do cluster se tiver acesso ao gateway local da sub-rede ou se tiver conectividade com recursos de VPC.

Recursos não suportados em clusters locais

- Complementos
- Funções do IAM para contas de serviço
- IPv6
- Provedores de identidade
- Fargate
- Criptografia KMS
- Zonas locais
- Karpenter
- Seletor de instâncias
- As zonas de disponibilidade não podem ser especificadas, pois o padrão é a zona de disponibilidade do Outpost.
- `vpc.publicAccessCIDRs` e `vpc.autoAllocateIPv6` não são aceitos.
- O acesso público ao endpoint ao servidor da API não é suportado, pois um cluster local só pode ser criado com acesso exclusivo ao endpoint privado.

Mais informações

- Amazon EKS no AWS Outposts
- Clusters locais para Amazon EKS no AWS Outposts
- Criação de clusters locais
- Lançamento de nós autogerenciados do Amazon Linux em um Outpost

Segurança

`eksctl`fornece algumas opções que podem melhorar a segurança do seu cluster EKS.

`withOIDC`

Ative `withOIDC`a criação automática de um IRSA para o plug-in CNI da Amazon e limite as permissões concedidas aos nós em seu cluster, em vez de conceder as permissões necessárias somente à conta de serviço da CNI.

O histórico é descrito nesta documentação da AWS.

`disablePodIMDS`

Para grupos de nós gerenciados e não gerenciados, a `disablePodIMDS`opção está disponível e impede que todos os pods de rede não hospedeiros executados nesse grupo de nós façam solicitações de IMDS.

> **Note**
>
> Isso não pode ser usado junto com `withAddonPolicies`.

Criptografia de envelope KMS para clusters EKS

> **Note**
>
> O Amazon Elastic Kubernetes Service (Amazon EKS) fornece criptografia envelopada padrão para todos os dados de API do Kubernetes nos clusters de EKS em execução no Kubernetes versão 1.28 ou mais recente. Para obter mais informações, consulte Criptografia de envelope padrão para todos os dados da API Kubernetes no Guia do usuário do EKS.

O EKS suporta o uso de chaves do AWS KMS para fornecer criptografia de envelope de segredos do Kubernetes armazenados no EKS. A criptografia de envelope adiciona uma camada adicional de criptografia gerenciada pelo cliente para segredos de aplicativos ou dados do usuário armazenados em um cluster Kubernetes.

Anteriormente, o Amazon EKS dava suporte à ativação da criptografia de envelopes usando chaves KMS somente durante a criação do cluster. Agora, você pode habilitar a criptografia de envelope para clusters Amazon EKS a qualquer momento.

Leia mais sobre como usar o suporte do provedor de criptografia EKS para defense-in-depth publicar no blog de contêineres da AWS.

Criação de um cluster com criptografia KMS ativada

```
# 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
```

Habilitando a criptografia KMS em um cluster existente

Para habilitar a criptografia KMS em um cluster que ainda não a tenha habilitado, execute

```
eksctl utils enable-secrets-encryption -f kms-cluster.yaml
```

ou sem um arquivo de configuração:

```
eksctl utils enable-secrets-encryption --cluster=kms-cluster --key-arn=arn:aws:kms:us-
west-2:<account>:key/<key> --region=<region>
```

Além de habilitar a criptografia KMS no cluster EKS, o eksctl também criptografa novamente todos os segredos existentes do Kubernetes usando a nova chave KMS, atualizando-os com a anotação. `eksctl.io/kms-encryption-timestamp` Esse comportamento pode ser desativado passando`--encrypt-existing-secrets=false`, como em:

```
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>
```

Se um cluster já tiver a criptografia KMS ativada, o eksctl continuará a criptografar novamente todos os segredos existentes.

> **Note**
>
> Depois que a criptografia KMS estiver ativada, ela não poderá ser desativada ou atualizada para usar uma chave KMS diferente.

Solução de problemas

Este tópico inclui instruções sobre como resolver erros comuns com o Eksctl.

Falha na criação da pilha

Você pode usar a `--cfn-disable-rollback` sinalização para impedir que o Cloudformation reverta pilhas que falharam para facilitar a depuração.

ID de sub-rede “subnet-11111111" não é o mesmo que “subnet-22222222"

Dado um arquivo de configuração especificando sub-redes para uma VPC, como o seguinte:

```
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: []
```

Um erro `subnet ID "subnet-11111111" is not the same as "subnet-22222222"` significa que as sub-redes especificadas não estão colocadas na zona de disponibilidade correta. Verifique no console da AWS qual é o ID de sub-rede correto para cada zona de disponibilidade.

Neste exemplo, a configuração correta para a VPC seria:

```
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}
```

Problemas de exclusão

Se sua exclusão não funcionar ou se você esquecer de `--wait` adicioná-la, talvez seja necessário usar outras ferramentas da Amazon para excluir as pilhas do cloudformation. Isso pode ser feito por meio da interface gráfica ou com o aws cli.

kubectl logs e kubectl run falham com erro de autorização

Se seus nós estiverem implantados em uma sub-rede privada `kubectl logs` e/ou `kubectl run` falharem com um erro como o seguinte:

```
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)
```

Então, talvez seja necessário definir enableDnsHostnames. Mais detalhes podem ser encontrados nesta edição.

Anúncios

Este tópico aborda anúncios anteriores de novos recursos do Eksctl.

Grupos de nós gerenciados padrão

A partir do eksctl v0.58.0, o eksctl cria grupos de nós gerenciados por padrão quando um arquivo não é especificado para e. `ClusterConfig` `eksctl create cluster` `eksctl create nodegroup` Para criar um grupo de nós autogerenciado, passe. `--managed=false` Isso pode interromper scripts que não usam um arquivo de configuração se um recurso não suportado em grupos de nós gerenciados, por exemplo, grupos de nós do Windows, estiver sendo usado. Para corrigir isso`--managed=false`, passe ou especifique sua configuração de grupo de nós em um `ClusterConfig` arquivo usando o `nodeGroups` campo que cria um grupo de nós autogerenciado.

Substituição de Bootstrap do Nodegroup para personalização AMIs

Essa mudança foi anunciada na edição Breaking: overrideBootstrapCommand soon... Agora, aconteceu neste PR. Leia atentamente a edição em anexo sobre por que decidimos deixar de oferecer suporte personalizado AMIs sem scripts de bootstrap ou com scripts de bootstrap parciais.

Ainda fornecemos um ajudante! Espero que migrar não seja tão doloroso. `eksctl`ainda fornece um script que, quando fornecido, exportará algumas propriedades e configurações úteis do ambiente. Esse script está localizado aqui.

As seguintes propriedades do ambiente estarão à sua disposição:

```
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
```

O mínimo que precisa ser usado ao substituir para `eksctl` não falhar são os rótulos! `eksctl`depende de um conjunto específico de rótulos no nó, para que ele possa encontrá-los. Ao definir a substituição, forneça este comando mínimo de substituição:

```
    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}"
```

Para grupos de nós que não têm acesso externo à Internet, você precisará fornecer `--apiserver-endpoint` e `--b64-cluster-ca` adicionar o script de bootstrap da seguinte forma:

```
    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}
```

Observe a configuração `--node-labels`. Se isso não for definido, o nó se juntará ao cluster, mas `eksctl` acabará atingindo o tempo limite na última etapa, quando estiver aguardando a chegada dos nós`Ready`. É fazer uma pesquisa no Kubernetes por nós que tenham o rótulo. `alpha.eksctl.io/nodegroup-name=<cluster-name>` Isso só é válido para grupos de nós não gerenciados. Para gerenciado, está usando um rótulo diferente.

Se, de alguma forma, for possível mudar para grupos de nós gerenciados para evitar essa sobrecarga, chegou a hora de fazer isso. Facilita muito a substituição.

As traduções são geradas por tradução automática. Em caso de conflito entre o conteúdo da tradução e da versão original em inglês, a versão em inglês prevalecerá.