Amazon EKS

Eksctl 用户指南

Eksctl 用户指南: Amazon EKS

Table of Contents

什么是 Eksctl？

eksctl 是一个命令行实用工具，可自动执行和简化创建、管理和操作亚马逊 Elastic Kubernetes Service (Amazon EKS) 集群的过程。eksctl 用 Go 编写，通过 YAML 配置和 CLI 命令提供了一种声明式语法，用于处理复杂的 EKS 集群操作，否则这些操作需要在不同的 AWS 服务上执行多个手动步骤。

eksctl 对于需要持续大规模部署和管理 EKS 集群的 DevOps 工程师、平台团队和 Kubernetes 管理员来说特别有价值。它对于从自我管理 Kubernetes 过渡到 EKS 的组织或实施基础设施即代码 (IaC) 实践的组织特别有用，因为它可以集成到现有的管道和自动化工作流程中。 CI/CD 该工具抽象出了 EKS 集群设置所需的 AWS 服务之间的许多复杂交互，例如 VPC 配置、IAM 角色创建和安全组管理。

eksctl 的主要功能包括能够使用单个命令创建功能齐全的 EKS 集群、支持自定义网络配置、自动化节点组管理和 GitOps 工作流程集成。该工具通过声明式方法管理集群升级、扩展节点组并处理附加组件管理。eksctl 还提供高级功能，例如 Fargate 配置文件配置、托管节点组自定义和 Spot 实例集成，同时通过原生 AWS SDK 集成保持与其他 AWS 工具和服务的兼容性。

功能

目前实现的功能有：

- 创建、获取、列出和删除集群
- 创建、清空和删除节点组
- 缩放节点组
- 更新集群
- 使用自定义 AMIs
- 配置 VPC 网络
- 配置对 API 端点的访问权限
- 对 GPU 节点组的支持
- 竞价型实例和混合型实例
- IAM 管理和附加政策
- 列出集群 Cloudformati
- 安装 coreDNS
- 为集群写入 kubeconfig 文件

Eksctl 常见问题

General

我能否使用`eksctl`来管理不是由创建的集群`eksctl`？

Yes! 从版本开始，`0.40.0`您可以`eksctl`针对任何集群运行，无论该集群`eksctl`是否由创建。有关更多信息，请参阅 the section called “非 eksctl 创建的集群”。

节点组

如何更改我的节点组的实例类型？

从的角度来看`eksctl`，节点组是不可变的。这意味着，一旦创建，唯一`eksctl`能做的就是向上或向下扩展节点组。

要更改实例类型，请使用所需的实例类型创建一个新的节点组，然后将其耗尽，以便工作负载转移到新的节点组。该步骤完成后，您可以删除旧的节点组。

如何查看节点组生成的用户数据？

首先，你需要管理节点组的 Cloudformation 堆栈的名称：

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

你会看到一个类似于的名字`eksctl-CLUSTER_NAME-nodegroup-NODEGROUP_NAME`。

你可以执行以下命令来获取用户数据。请注意最后一行，它从 base64 解码并解压压缩后的数据。

```
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
```

入口

如何设置入口？`eksctl`

我们建议使用 AWS Load Balancer 控制器。有关如何将控制器部署到集群以及如何从旧的 ALB Ingress Controller 迁移的文档可在此处找到。

对于 Nginx 入口控制器，设置将与其他 Kubernetes 集群上的任何控制器相同。

Kubectl

我使用的是 HTTPS 代理，但集群证书验证失败，我该如何使用系统 CAs？

设置环境变量`KUBECONFIG_USE_SYSTEM_CA`以`kubeconfig`尊重系统证书颁发机构。

试跑

试运行功能允许您在继续创建节点组之前检查和更改与实例选择器匹配的实例。

当`eksctl create cluster`使用实例选择器选项和调用时`--dry-run`，eksctl 将输出一个 ClusterConfig 文件，其中包含一个节点组，该节点组表示 CLI 选项和根据实例选择器资源标准匹配的实例设置的实例类型。

```
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
```

然后 ClusterConfig 可以将生成的数据传递给`eksctl create cluster`：

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

使用传递 ClusterConfig 文件时`--dry-run`，eksctl 将输出一个包含 ClusterConfig 文件中设置的值的文件。

eksctl 中的一次性选项

有些一次性选项无法在`ClusterConfig`文件中表示，`--install-vpc-controllers`例如。

预计：

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

其次是：

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

等同于在没有`--dry-run`。的情况下运行第一个命令。

因此，eksctl 不允许在传递时`--dry-run`传递配置文件中无法表示的选项。

> **Important**
>
> 如果您需要传递 AWS 配置文件，请设置`AWS_PROFILE`环境变量，而不是传递 `--profile` CLI 选项。

教程

本主题将指导您安装和配置 eksctl，然后使用它创建 Amazon EKS 集群。

第 1 步：安装 eksctl

完成以下步骤，在你的 Linux 或 macOS 设备上下载并安装最新版本的 eksctl：

使用 Homebrew 安装 eksctl

1. （先决条件）安装自制软件。
2. 添加 AWS 水龙头：

```
brew tap aws/tap
```

3. 安装 eksctl

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

在使用 eksctl 之前，请完成以下配置步骤：

1. 安装先决条件：
   - 安装 AWS CLI 版本 2.x 或更高版本。
   - 使用 Hom ebrew 安装 kubectl：

```
brew install kubernetes-cli
```

2. 在您的环境中@@ 配置 AWS 证书：

```
aws configure
```

3. 验证 AWS CLI 配置：

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

步骤 2：创建集群配置文件

使用以下步骤创建集群配置文件：

1. 创建一个名为cluster.yaml：的新文件

```
touch cluster.yaml
```

2. 添加以下基本群集配置：

```
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. 自定义配置：
   - 更新region以匹配您所需的 AWS 区域。
   - instanceType根据您的工作负载要求进行修改。
   - maxSize根据需要调整desiredCapacityminSize、和。
4. 验证配置文件：

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

步骤 3：创建集群

按照以下步骤创建 EKS 集群：

1. 使用配置文件创建集群：

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

2. 等待集群创建（这通常需要 15-20 分钟）。
3. 验证集群创建：

```
eksctl get cluster
```

4. 将 kubectl 配置为使用你的新集群：

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

5. 验证集群连接：

```
kubectl get nodes
```

您的集群现已准备就绪，可以使用。

可选：删除集群

请记住在完成后删除集群，以免产生不必要的费用：

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

> **Note**
>
> 创建集群可能会产生 AWS 费用。在创建集群之前，请务必查看 Amazon EKS 的定价。

后续步骤

- 将 Kubectl 配置为连接到集群
- 部署示例应用程序

Eksctl 的安装选项

eksctl可从官方版本中安装，如下所述。我们建议您仅eksctl从官方 GitHub 版本进行安装。您可以选择使用第三方安装程序，但请注意，AWS 不维护也不支持这些安装方法。您可以自行决定使用它们。

先决条件

您需要配置 AWS API 证书。适用于 AWS CLI 或任何其他工具（kops、Terraform 等）的工具应该足够了。您可以使用~/.aws/credentials文件或环境变量。有关更多信息，请参阅 AWS CLI 参考。

你还需要在你的命令中使用 AWS IAM Authenticator for Kubernetes 命令（aws-iam-authenticator或aws eks get-token（在 AWS CLI 的 1.16.156 或更高版本中可用）。PATH

用于创建 EKS 集群的 IAM 账户应具有这些最低访问级别。

AWS 服务	访问级别
CloudFormation	完全访问权限
EC2	完整：Tagging L imited：列出、读取、写入
EC2 Auto Scaling	限制：列出、写入
EKS	完全访问权限
IAM	限制：列表、读取、写入、权限管理
Systems Manager	Limited: List, Read

适用于Unix

要下载最新版本，请运行：

```
# 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
```

对于 Windows：

直接下载（最新版本）：

- AMD64/x86_64
- ARMv6
- ARMv7
- ARM64

确保将存档解压缩到PATH变量中的某个文件夹。

或者，验证校验和：

1. 下载校验和文件：最新
2. 使用命令提示符手动将输出与下载CertUtil的校验和文件进行比较。

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

3. 使用 PowerShell 运-eq算符自动进行验证，以获得True或False结果：

```
# 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]
```

使用 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
```

`eksctl`可执行文件放在 `$HOME/bin` Git Bash 中`$PATH`。

Homebrew

你可以使用 Homebrew 在 macOS 和 Linux 上安装软件。

AWS 维护着一个包含 eksctl 的 Homebrew 水龙头。

有关 Homebrew tap 的更多信息，请参阅 Github 上的项目和 eksctl 的 H omebrew 公式。

使用 Homebrew 安装 eksctl

1. （先决条件）安装自制软件
2. 添加 AWS 水龙头

```
brew tap aws/tap
```

3. 安装 eksctl

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

Docker

对于每个版本和 RC，都会将容器镜像推送到 ECR 存储库`public.ecr.aws/eksctl/eksctl`。在 ECR Public Gallery-ek sctl 上了解有关用法的更多信息。例如，

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

外壳完成

Bash

要启用 bash 完成功能，请运行以下命令，或者将其放入`~/.bashrc`或`~/.profile`：

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

Zsh

要完成 zsh，请运行：

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

然后输入以下内容`~/.zshrc`：

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

请注意，如果你没有像 oh-my-zsh那样运行发行版，则可能需要先启用自动补全功能（然后将其设置`~/.zshrc`为持久化）：

```
autoload -U compinit
compinit
```

鱼

以下命令可用于 fish auto 自动完成：

```
mkdir -p ~/.config/fish/completions
```

```
eksctl completion fish > ~/.config/fish/completions/eksctl.fish
```

Powershell

可以参考以下命令进行设置。请注意，路径可能会有所不同，具体取决于您的系统设置。

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

更新

> **Important**
>
> 如果您通过直接下载 eksctl（而不是使用软件包管理器）来安装，则需要手动对其进行更新。

集群

本章介绍如何使用 eksctl 创建和配置 EKS 集群。它还包括附加组件和 EKS 自动模式。

主题：

- the section called “EKS 访问条目”
  - 将 aws-auth 替换为 EKS 访问条目来简化 Kubernetes RBAC 管理 ConfigMap
  - 将现有 IAM 身份映射从 aws-auth ConfigMap 迁移到访问条目
  - 配置集群身份验证模式并控制集群创建者管理员权限
- the section called “默认插件更新”
  - 通过更新旧集群上的默认 EKS 插件来保护集群安全
- the section called “插件”
  - 自动执行安装、更新和删除插件的例行任务。
  - Amazon EKS 插件包括 AWS 插件、开源社区插件和市场插件。
- the section called “EKS 自动模式”
  - 让 AWS 管理您的 EKS 基础设施，从而减少运营开销
  - 配置自定义节点池，而不是默认的通用节点池和系统池
  - 将现有 EKS 集群转换为使用自动模式
- the section called “CloudWatch 日志记录”
  - 通过为特定 EKS 控制平面组件启用日志，解决集群问题
  - 为 EKS 集群日志配置日志保留期
  - 使用 eksctl 命令修改现有的集群日志记录设置
- the section called “集群升级”
  - 通过安全升级 EKS 控制平面版本来维护安全性和稳定性
  - 通过用新组替换旧组，跨节点组推出升级
  - 更新默认集群插件
- the section called “创建和管理集群”
  - 使用默认托管节点组快速开始使用基本 EKS 集群
  - 使用具有特定配置的配置文件创建自定义集群
  - VPCs 使用私有网络和自定义 IAM 策略部署现有集群

- the section called “配置 kubelet”
  - 通过配置 kubelet 和系统守护程序预留来防止节点资源短缺
  - 自定义内存和文件系统可用性的驱逐阈值
  - 跨节点组启用或禁用特定的 kubelet 功能门
- the section called “EKS 连接器”
  - 通过 EKS 控制台集中管理混合 Kubernetes 部署
  - 为外部集群访问配置 IAM 角色和权限
  - 移除外部集群并清理关联的 AWS 资源
- the section called “EKS 全私有集群”
  - 使用没有出站互联网访问权限的完全私有 EKS 集群满足安全要求
  - 配置通过 VPC 终端节点对 AWS 服务的私有访问
  - 使用明确的网络设置创建和管理私有节点组
- the section called “Karpenter Support”
  - 自动配置节点
  - 创建自定义 Karpenter 配置程序配置
  - 使用 Spot 实例中断处理功能设置 Karpenter
- the section called “Amazon EMR”
  - 在 EMR 和 EKS 集群之间创建 IAM 身份映射
- the section called “EKS Fargate Support”
  - 为吊舱调度定义自定义 Fargate 配置文件
  - 通过创建和配置更新来管理 Fargate 配置文件
- the section called “非 eksctl 创建的集群”
  - 对在 eksctl 之外创建的集群进行标准化管理
  - 在现有的非 eksctl 集群上使用 eksctl 命令
- the section called “启用可用区转移”
  - 通过启用快速区域故障切换功能提高应用程序可用性
  - 在新的 EKS 集群部署上配置区域切换
  - 在现有 EKS 集群上启用区域转移功能

创建和管理集群

本主题介绍如何使用 Eksctl 创建和删除 EKS 集群。您可以使用 CLI 命令创建集群，也可以通过创建集群配置 YAML 文件来创建集群。

创建简单集群

使用以下命令创建一个简单的集群：

```
eksctl create cluster
```

这将在您的默认区域（由您的 AWS CLI 配置指定）中创建一个 EKS 集群，其中一个托管节点组包含两个 m5.large 节点。

现在，当不使用配置文件时，eksctl 会默认创建托管节点组。要创建自我管理的节点组，请传递`--managed=false`到或。`eksctl create cluster` `eksctl create nodegroup`

注意事项

- 在中创建集群时`us-east-1`，您可能会遇到`UnsupportedAavailabilityZoneException`。如果发生这种情况，请复制建议的区域并传递`--zones`标志，例如：`eksctl create cluster --region=us-east-1 --zones=us-east-1a,us-east-1b,us-east-1d`。此问题可能发生在其他地区，但不太常见。在大多数情况下，您不需要使用标`--zone`志。

使用配置文件创建集群

您可以使用配置文件而不是标志来创建集群。

首先，创建`cluster.yaml`文件：

```
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
```

接下来，运行以下命令：

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

这将按所述创建集群。

如果您需要使用现有 VPC，则可以使用如下配置文件：

```
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
```

集群名称或节点组名称必须仅包含字母数字字符（区分大小写）和连字符。它必须以字母字符开头且不能超过 128 个字符，否则您将收到验证错误。有关更多信息，请参阅 AWS Formati CLoud on 用户指南中的通过 CloudFormation 控制台创建堆栈。

为新集群更新 kubeconfig

创建集群后，相应的 kubernetes 配置将添加到您的 kubeconfig 文件中。这是您在环境变量中配置的文件，KUBECONFIG或者是~/.kube/config默认配置的文件。使用该标志可以覆盖 kubeconfig 文件的路径。--kubeconfig

其他可以改变 kubeconfig 文件写入方式的标志：

标记	类型	使用	默认值
--kubeconfig	字符串	写入 kubeconfig 的路径（与--auto-kubeconfig 不兼容）	$KUBECONFIG 或 ~/.kube/config
--set-kubeconfig-context	布尔	如果为 true 则将在 kubeconfig 中设置当前上下文；如果已经设置了上下文，则它将被覆盖	true
--auto-kubeconfig	布尔	按集群名称保存 kubeconfig 文件	true
--write-kubeconfig	布尔	切换 kubeconfig 的写入	true

删除集群

要删除此集群，请运行：

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

> ⚠ Warning
>
> 在删除操作中使用该`--wait`标志，以确保正确报告删除错误。

如果没有该`--wait`标志，eksctl 只会向集群的 CloudFormation 堆栈发出删除操作，而不会等待其删除。在某些情况下，使用集群或其 VPC 的 AWS 资源可能会导致集群删除失败。如果删除失败或者忘记了等待标志，则可能需要转到 CloudFormation GUI 并从那里删除 eks 堆栈。

> ⚠ Warning
>
> PDB 策略可能会阻止在集群删除期间移除节点。

删除带有节点组的集群时，Pod 中断预算 (PDB) 策略可能会阻止成功删除节点。例如，`aws-ebs-csi-driver`安装了的集群通常有两个 pod，它们的 PDB 策略允许一次只能有一个 pod 不可用，这使得另一个 Pod 在删除期间无法被驱逐。要在这些情况下成功删除集群，请使用该`disable-nodegroup-eviction`标志绕过 PDB 策略检查：

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

有关更多示例配置文件，请参阅 eksc GitHub tl 存储库中的`examples/`目录。

试跑

dry-run 功能允许生成一个跳过集群创建的 ClusterConfig 文件，并输出一个表示所提供的 CLI 选项并包含 eksctl 设置的默认值的 ClusterConfig 文件。

更多信息可以在 D ry Run 页面上找到。

EKS 自动模式

eksctl 支持 EKS 自动模式，该功能将 AWS 对 Kubernetes 集群的管理扩展到集群本身之外，从而允许 AWS 设置和管理基础设施，使您的工作负载能够顺利运行。这使您可以委托关键基础设施决策并利用 AWS 的专业知识进行 day-to-day运营。AWS 管理的集群基础设施包括许多 Kubernetes 功能作为核心

组件，而不是附加组件，例如计算自动扩展、容器和服务联网、应用程序负载平衡、集群 DNS、区块存储和 GPU 支持。

创建启用自动模式的 EKS 集群

eksctl添加了一个用于启用和配置自动模式的新autoModeConfig字段。autoModeConfig场地的形状是

```
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
```

如果autoModeConfig.enabled为 true，则 eksctl 通过将computeConfig.enabled: truekubernetesNetworkConfig.elasticLoadBalancing.enabled: true、和传递给 EKS API storageConfig.blockStorage.enabled: true 来创建 EKS 集群，从而实现计算、存储和网络等数据平面组件的管理。

要创建启用了自动模式的 EKS 集群，请进行设置autoModeConfig.enabled: true，如所示

```
# 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 创建一个节点角色用于自动模式启动的节点。eksctl 还会创建和节点池。general-purpose system要禁用默认节点池的创建，例如，要配置自己的节点池以使用不同的子网集，请进行设置nodePools: []，如下所示

```
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.
```

更新 EKS 集群以使用自动模式

要将现有 EKS 集群更新为使用自动模式，请运行

```
# 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**
>
> 如果集群由 eksctl 创建，并且它使用公有子网作为集群子网，则自动模式将启动公共子网中的节点。要将私有子网用于由自动模式启动的工作节点，请更新集群以使用私有子网。

禁用自动模式

要禁用自动模式，请设置`autoModeConfig.enabled: false`并运行

```
# 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
```

进一步信息

- EKS 自动模式

EKS 访问条目

您可以使用 eksctl 来管理 EKS 访问条目。使用访问条目向 Kubernetes 授予 AWS IAM 身份的权限。例如，你可以向开发者角色授予读取集群中 Kubernetes 资源的权限。

本主题介绍如何使用 eksctl 管理访问条目。有关访问条目的一般信息，请参阅使用 EKS 访问条目向 IAM 用户授予访问 Kubernetes 的访问权限。

您可以附加 AWS 定义的 Kubernetes 访问策略，也可以将 IAM 身份与 Kubernetes 群组相关联。

有关可用预定义策略的更多信息，请参阅将访问策略与访问条目关联。

如果您需要定义客户 Kubernetes 策略，请将 IAM 身份与 Kubernetes 群组关联，然后向该群组授予权限。

集群身份验证模式

只有在集群的身份验证模式允许的情况下，您才能使用访问条目。

有关更多信息，请参阅设置集群身份验证模式

使用 YAML 文件设置身份验证模式

`eksctl`在下方添加了一个新`accessConfig.authenticationMode`字段 ClusterConfig，可以将其设置为以下三个值之一：

- `CONFIG_MAP`-在 EKS API 中为默认值-只`aws-auth` ConfigMap 会使用

- API-将仅使用访问入口 API
- API_AND_CONFIG_MAP-默认为 eksctl-可以aws-auth ConfigMap 同时使用访问条目 API

在 ClusterConfig YAML 中设置身份验证模式：

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

使用命令更新身份验证模式

如果要在已存在的、非 eksctl 创建的集群上使用访问条目（其中使用了CONFIG_MAP选项），则用户需要先将其设置为。authenticationMode API_AND_CONFIG_MAP为此，引入eksctl了一个用于更新集群身份验证模式的新命令，该命令均适用于 CLI 标志，例如

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

访问参赛资源

访问条目的类型为STANDARD或EC2_LINUX。类型取决于您使用访问权限条目的方式。

- 该standard类型用于向 IAM 用户和 IAM 角色授予 Kubernetes 权限。
  - 例如，您可以通过将访问策略附加到用于访问控制台的角色或用户来查看 AWS 控制台中的 Kubernetes 资源。
- EC2_LINUX和EC2_WINDOWS类型用于授予 Kubernetes 对 EC2 实例的权限。实例使用这些权限加入集群。

有关访问条目类型的更多信息，请参阅创建访问条目

IAM 实体

您可以使用访问权限条目向 IAM 用户和 IAM 角色等 IAM 身份授予 Kubernetes 权限。

使用该accessConfig.accessEntries字段将 IAM 资源的 ARN 与访问条目 EKS API 关联。例如：

```
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
```

除了关联 EKS 策略外，还可以指定 IAM 实体所属的 Kubernetes 组，从而通过 RBAC 授予权限。

托管节点组和 Fargate

将通过 EKS API 在幕后实现与这些资源的访问条目的集成。新创建的托管节点组和 Fargate 容器将创建 API 访问条目，而不是使用预加载的 RBAC 资源。现有的节点组和 Fargate 容器将不会更改，并且会继续依赖于 aws-auth 配置映射中的条目。

自我管理的节点组

每个访问条目都有一种类型。为了授权自我管理的节点组，`eksctl`将为每个节点组创建一个唯一的访问条目，并将主体 ARN 设置为节点角色 ARN，类型设置为任一或取决于节点组 amiFamily。`EC2_LINUX` `EC2_WINDOWS`

在创建自己的访问条目时，您还可以指定`EC2_LINUX`（用于与 Linux 或 Bottlerocket 自管理节点一起使用的 IAM 角色）、`EC2_WINDOWS`（用于与 Windows 自管理节点一起使用的 IAM 角

色）、FARGATE_LINUX（用于与 AWS Fargate（Fargate）一起使用的 IAM 角色）或指定为类型。STANDARD如果未指定类型，则默认类型将设置为STANDARD。

> **Note**
>
> 删除使用预先存在的节点组创建的节点组时instanceRoleARN，当不再有与之关联的节点组时，用户有责任删除相应的访问条目。这是因为 eksctl 不会尝试找出非 eksctl 创建的自管理节点组是否仍在使用访问条目，因为这是一个复杂的过程。

创建访问条目

这可以通过两种不同的方式完成，一种是在创建集群期间，在配置文件中指定所需的访问条目并运行：

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

或者在创建集群后，运行以下命令：

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

有关创建访问条目的配置文件示例，请参阅 eksctl 存储库中的 40-access-entries.yaml。 GitHub

获取访问权限

用户可以通过运行以下命令之一来检索与某个集群关联的所有访问条目：

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

或

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

或者，要仅检索与某个 IAM 实体相对应的访问条目，则应使用--principal-arn标志。例如

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

删除访问权限条目

要一次删除单个访问条目，请使用：

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

要删除多个访问条目，请使用`--config-file`标志并在顶级`accessEntry`字段下指定所有与访问条目`principalARN's`对应的条目，例如

```
...
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
```

从 aws-auth 迁移 ConfigMap

用户可以通过运行以下命令将其现有的 IAM 身份从 aws-auth configmap 迁移到访问条目：

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

当`--target-authentication-mode`标志设置为时`API`，身份验证模式将切换到`API`模式（如果已处于`API`模式则跳过），IAM 身份映射将迁移到访问条目，并从集群中删除 aws-auth configmap。

当`--target-authentication-mode`标志设置为时`API_AND_CONFIG_MAP`，身份验证模式将切换到模式（如果已`API_AND_CONFIG_MAP`处于`API_AND_CONFIG_MAP`模式则跳过），IAM 身份映射将迁移到访问条目，但会保留 aws-auth configmap。

> **Note**
>
> 当 f `--target-authentication-mode` lag 设置为时`API`，如果 aws-auth configmap 具有以下限制之一，则此命令不会将身份验证`API`模式更新为模式。

- 有一个账户级别的身份映射。

- 一个或多个 Roles/Users 被映射到以前缀开头的 kubernetes 组`system:`（ EKS 特定的群组除外`system:masterssystem:bootstrappers`，`system:nodes`例如等）。
- 一个或多个 IAM 身份映射用于 [服务关联角色]（链接：IAM/latest/UserGuide/using-service-linked-roles .html）。

禁用集群创建者管理员权限

`eksctl`添加了一个新字段，当设置为 false 时，`accessConfig.bootstrapClusterCreatorAdminPermissions: boolean`该字段将禁止向创建集群的 IAM 身份授予集群管理员权限。即

在配置文件中添加选项：

```
accessConfig:
  bootstrapClusterCreatorAdminPermissions: false
```

然后运行：

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

非 eksctl 创建的集群

您可以对不是由创建的集群运行`eksctl`命令`eksctl`。

> **Note**
>
> Eksctl 只能支持名称与 AWS 兼容的无主集群。 CloudFormation任何与此不匹配的集群名称都将无法通过 CloudFormation API 验证检查。

支持的 命令

以下命令可用于通过除之外的任何方式创建的集群`eksctl`。命令、标志和配置文件选项的使用方式完全相同。

如果我们错过了某些功能，请告诉我们。

✓ 创建：

- ✓eksctl create nodegroup（参见下面的注释）
- ✓ eksctl create fargateprofile
- ✓ eksctl create iamserviceaccount
- ✓ eksctl create iamidentitymapping

✓ 获取：

- ✓ eksctl get clusters/cluster
- ✓ eksctl get fargateprofile
- ✓ eksctl get nodegroup
- ✓ eksctl get labels

✓ 删除：

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

✓ 升级：

- ✓ eksctl upgrade cluster
- ✓ eksctl upgrade nodegroup

✓ 设置/取消设置：

- ✓ eksctl set labels
- ✓ eksctl unset labels

✓ 比例：

- ✓ eksctl scale nodegroup

✓ 排水：

- ✓ eksctl drain nodegroup

✓ 启用：

- ✓ eksctl enable profile
- ✓ eksctl enable repo

✓ 实用程序：

- ✓ 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

创建节点组

eksctl create nodegroup是唯一需要用户提供特定输入的命令。

由于用户可以使用自己喜欢的任何网络配置创建集群，因此暂时eksctl不会尝试检索或猜测这些值。随着我们更多地了解人们如何在非 eksctl 创建的集群上使用此命令，这种情况将来可能会改变。

这意味着，要在不是由创建的集群上创建节点组或托管节点组eksctl，必须提供包含 VPC 详细信息的配置文件。至少：

```
---
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"

...
```

有关 VPC 配置选项的更多信息，请参阅网络。

使用 EKS 连接器注册非 EKS 集群

您可以使用 EKS 连接器在 EKS 控制台中查看 AWS 以外的集群。此过程需要向 EKS 注册集群，并在外部 Kubernetes 集群上运行 EKS Connector 代理。

`eksctl`通过创建所需的 AWS 资源并生成 Kubernetes 清单以供 EKS Connector 应用于外部集群，从而简化了非 EKS 集群的注册。

注册集群

要注册或连接非 EKS Kubernetes 集群，请运行

```
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
```

此命令将注册集群并写入三个文件，其中包含 EKS Connector 的 Kubernetes 清单，这些清单必须在注册到期之前应用于外部集群。

> **Note**
>
> `eks-connector-clusterrole.yaml`get并`eks-connector-console-dashboard-full-access-clusterrole.yaml`向调用的 IAM 身份授予所有命名空间中的 Kubernetes 资源的`list`权限，在将其应用到集群之前，必须根据需要进行相应的编辑。要配置更多受限访问权限，请参阅向用户授予查看集群的访问权限。

要提供现有 IAM 角色以用于 EKS Connector，请将其传递，`--role-arn`如下所示：

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

如果集群已经存在，eksctl 将返回错误。

取消注册集群

要取消注册或断开已注册集群的连接，请运行

```
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
```

此命令将取消注册外部集群并移除其关联的 AWS 资源，但您需要从集群中删除 EKS 连接器 Kubernetes 资源。

进一步信息

- EKS 连接器

自定义 kubelet 配置

可以通过配置 kubelet 来预留系统资源。建议这样做，因为在资源匮乏的情况下，kubelet 可能无法驱逐 pod 并最终使节点变成。`NotReady`为此，配置文件可以包含接受将嵌入到的自由格式 yaml 的`kubeletExtraConfig`字段。`kubelet.yaml`

中的kubelet.yaml某些字段由 eksctl 设置，因此不可重写，例
如、、addressclusterDomain、authentication或。authorization
serverTLSBootstrap

以下示例配置文件创建了一个节点组，该节点组为 kubelet 预留 300m vCPU、300Mi内存和1Gi临时存储空间；为操作系统守护程序保留 300m vCPU 300Mi、内存1Gi和临时存储；当可用内存少于或根文件系统的少于 10% 时，启动驱逐 Pod。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
```

在此示例中，给定具有 4 v CPUs 和 16GiB 内存的类型m5a.xlarge实例，内
存Allocatable量 CPUs 将为 3.4 和 15.4 GiB。重要的是要知道，配置文件中为中的字段指定的值kubeletExtraconfig将完全覆盖 eksctl 指定的默认值。但是，根据所使用的 aws 实例类型，省略一个或多个kubeReserved参数会导致缺少的参数默认为合理的值。

kubeReserved计算

虽然通常建议将混合实例配置 NodeGroup 为使用具有相同 CPU 和 RAM 配置的实例，但这并不是一项严格的要求。因此，kubeReserved计算使用InstanceDistribution.InstanceTypes字段中最小的实例。这样 NodeGroups ，对于不同的实例类型，就不会在最小的实例上预留太多资源。但是，对于最大的实例类型，这可能会导致预留量过小。

> **Warning**
>
> 默认情况下featureGates.RotateKubeletServerCertificate=true，eksctl设置为未设置，但是当提供自定义featureGates设置时，它将被取消设置。除非必须将其禁用featureGates.RotateKubeletServerCertificate=true，否则应始终将其包括在内。

CloudWatch 日志记录

本主题介绍如何为您的 EKS 集群的控制平面组件配置 Amazon CloudWatch 日志。 CloudWatch 日志记录提供了对集群控制平面操作的可见性，这对于故障排除、审计集群活动和监控 Kubernetes 组件的运行状况至关重要。

启用 CloudWatch 日志记录

CloudWatch 由于数据摄取和存储成本，EKS 控制平面的@@ 日志记录在默认情况下未启用。

要在创建集群时启用控制平面日志记录，您需要在中定义**cloudWatch.clusterLogging.enableTypes**设置ClusterConfig（有关示例，请参见下文）。

因此，如果您的配置文件**cloudWatch.clusterLogging.enableTypes**设置正确，则可以使用创建集群eksctl create cluster --config-file=<path>。

如果您已经创建了集群，则可以使用eksctl utils update-cluster-logging。

> **Note**
>
> 默认情况下，此命令在计划模式下运行，您需要指定--approve标志才能将更改应用于您的集群。

如果你使用的是配置文件，请运行：

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

或者，您可以使用 CLI 标志。

要启用所有类型的日志，请运行：

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

要启用audit日志，请运行：

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

要启用除controllerManager日志之外的所有内容，请运行：

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

如果api和scheduler日志类型已经启用，要同时禁用scheduler和启controllerManager用，请运行：

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

这将使api和controllerManager成为唯一启用的日志类型。

要禁用所有类型的日志，请运行：

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

ClusterConfig 示例

在 EKS 集群中，下面的enableTypes字段clusterLogging可以列出可能的值，以便为控制平面组件启用不同类型的日志。

有以下可能值：

- api：启用 Kubernetes API 服务器日志。
- audit：启用 Kubernetes 审计日志。

- `authenticator`：启用身份验证器日志。
- `controllerManager`：启用 Kubernetes 控制器管理器日志。
- `scheduler`：启用 Kubernetes 调度器日志。

要了解更多信息，请参阅 EKS 文档。

禁用所有日志

要禁用所有类型，请使用`[]`或完全删除该`cloudWatch`部分。

启用所有日志

您可以使用`"*"`或启用所有类型`"all"`。例如：

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

启用一个或多个日志

您可以通过列出要启用的类型来启用类型子集。例如：

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

日志保留期

默认情况下，日志无限期地存储在 CloudWatch 日志中。您可以在 Logs 中指定控制平面日志应保留的 CloudWatch 天数。以下示例将日志保留 7 天：

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

完整示例

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

```
kind: ClusterConfig

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

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

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

EKS 全私有集群

eksctl 支持创建没有出站 Internet 访问权限且只有私有子网的完全私有集群。VPC 终端节点用于实现对 AWS 服务的私有访问。

本指南介绍如何创建没有出站 Internet 访问权限的私有集群。

创建完全私有集群

创建完全私有集群的唯一必填字段是：`privateCluster.enabled`

```
privateCluster:
  enabled: true
```

创建集群后，需要访问 Kubernetes API 服务器的 eksctl 命令必须从集群的 VPC、对等 VPC 或使用其他方式（例如 AWS Direct Connect）运行。需要访问 EKS 的 eksctl 命令如果在集群的 VPC 内运行，APIs 则无法运行。要解决这个问题，请为亚马逊 EKS 创建一个接口终端节点，以便从您的亚马逊虚拟私有云 (VPC) 私有访问亚马逊 Elastic Kubernetes Service (Amazon EKS APIs) 管理。在 future 版本中，eksctl 将增加对创建此端点的支持，因此无需手动创建。启用 AWS for A PrivateLink mazon EKS 后，需要访问 OpenID Connect 提供商 URL 的命令将需要从集群的 VPC 外部运行。

创建托管节点组将继续起作用，并且创建自我管理的节点组将起作用，因为如果命令是在集群的 VPC、对等的 VPC 内运行的，或者使用其他方式（例如 AWS Direct Connect），则需要通过 EKS 接口终端节点访问 API 服务器。

> **Note**
>
> VPC 终端节点按小时和使用量收费。有关定价的更多详细信息，请参阅 A WS PrivateLink 定价

> **Warning**
>
> 中eu-south-1不支持完全私有群集。

配置对其他 AWS 服务的私有访问权限

为了使工作节点能够私下访问 AWS 服务，eksctl 为以下服务创建 VPC 终端节点：

- 用于提取容器镜像的 ECR（两者ecr.api兼有ecr.dkr）的接口终端节点（AWS CNI 插件等）
- S3 用于提取实际图像层的网关端点
- aws-cloud-provider集成所需的 EC2 接口终端节点
- STS 的接口终端节点，用于支持服务账户的 Fargate 和 IAM 角色 (IRSA)
- 用于 CloudWatch 记录的接口端点 (logs)（如果启用了 CloudWatch 日志记录）

这些 VPC 终端节点对于正常运行的私有集群来说是必不可少的，因此，eksctl 不支持配置或禁用它们。但是，集群可能需要私有访问其他 AWS 服务（例如，集群自动扩缩器所需的自动扩缩功能）。可以在中指定这些服务privateCluster.additionalEndpointServices，它指示 eksctl 为每个服务创建一个 VPC 终端节点。

例如，要允许私人访问自动缩放和 CloudWatch 日志记录，请执行以下操作：

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

中支持的端点additionalEndpointServices是autoscaling、cloudformation和logs。

跳过端点创建

如果已经创建了 VPC，并设置了必要的 AWS 终端节点并链接到 EKS 文档中描述的子网，则eksctl可以通过提供skipEndpointCreation如下选项来跳过创建它们：

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

此设置不能与一起使用additionalEndpointServices。它将跳过所有端点的创建。此外，仅当端点<#子网拓扑设置正确时，才建议使用此设置。如果子网 ID 正确，则使用前缀地址设置vpce路由，创建所有必要的 EKS 端点并将其链接到提供的 VPC。 eksctl不会更改这些资源中的任何一个。

节点组

在完全私有集群中仅支持私有节点组（包括托管和自管理），因为集群的 VPC 是在没有任何公有子网的情况下创建的。必须明确设置privateNetworking字段（nodeGroup**[].privateNetworking**和managedNodeGroup[）。在完全私有集群中privateNetworking未设置是错误的。

```
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
```

集群终端节点访问

完全私有集群不支持在创建集群clusterEndpointAccess期间进行修改。设置clusterEndpoints.publicAccess或都是错误的clusterEndpoints.privateAccess，因为完全私有集群只能拥有私有访问权限，并且允许修改这些字段可能会破坏集群。

用户提供的 VPC 和子网

eksctl 支持使用预先存在的 VPC 和子网创建完全私有集群。只能指定私有子网，在下方指定子网是错误的。`vpc.subnets.public`

eksctl 在提供的 VPC 中创建 VPC 终端节点，并修改所提供子网的路由表。每个子网都应有一个与之关联的显式路由表，因为 eksctl 不会修改主路由表。

```
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
```

管理全私有集群

要使所有命令在集群创建后起作用，eksctl 需要私有访问 EKS API 服务器端点和出站互联网访问权限（对于`EKS:DescribeCluster`）。如果 eksctl 具有出站互联网访问权限，则将支持不需要访问 API 服务器的命令。

强制删除全私有集群

通过 eksctl 删除完全私有集群时可能会出现错误，因为 eksctl 无法自动访问集群的所有资源。 `--force`存在是为了解决这个问题：它将强制删除集群，并在出现错误时继续。

限制

当前实现的一个局限性是，eksctl 最初在创建集群时启用了公有和私有终端节点访问权限，并在所有操作完成后禁用公共端点访问权限。这是必需的，因为 eksctl 需要访问 Kubernetes API 服务器才能允许自我管理的节点加入集群并支持和 Fargate。 GitOps 这些操作完成后，eksctl 会将集群终端节点的访问权限切换为仅限私有。此额外更新确实意味着创建完全私有集群所需的时间将比标准集群更长。将来，eksctl 可能会切换到支持 vPC 的 Lambda 函数来执行这些 API 操作。

通过 HTTP 代理服务器进行出站访问

eksctl 能够 APIs 通过配置的 HTTP (S) 代理服务器与 AWS 通信，但是您需要确保正确设置代理排除列表。

通常，您需要通过设置适当的`no_proxy`环境变量（包括该值）来确保集群的 VPC 终端节点请求不会通过代理进行路由。`.eks.amazonaws.com`

如果您的代理服务器执行 “SSL 拦截”，并且您使用的是服务账户的 IAM 角色 (IRSA)，则需要确保明确绕过该域 Man-in-the-Middle的 SSL。`oidc.<region>.amazonaws.com`不这样做将导致 eksctl 为 OIDC 提供商获取错误的根证书指纹，并且 AWS VPC CNI 插件将由于无法获取 IAM 证书而无法启动，从而导致您的集群无法运行。

进一步信息

- EKS 私有集群

插件

本主题介绍如何使用 eksctl 管理亚马逊 EKS 集群的 Amazon EKS 插件。EKS Add-Ons 是一项允许您通过 EKS API 启用和管理 Kubernetes 操作软件的功能，从而简化了安装、配置和更新集群插件的过程。

> **Warning**
>
> eksctl 现在将默认插件（vpc-cni、coredns、kube-proxy）安装为 EKS 插件，而不是自行管理的插件。这意味着对于使用 eksctl v0.184.0 及更高版本创建的集群，您应该使用`eksctl update addon`而不是`eksctl utils update-*`命令。

当你想使用 Cilium 和 Calico 等其他 CNI 插件时，可以在没有任何默认网络插件的情况下创建集群。

EKS 插件现在支持通过 EKS Pod 身份关联接收 IAM 权限，允许他们连接集群外的 AWS 服务

创建插件

Eksctl 为管理集群插件提供了更大的灵活性：

在配置文件中，您可以指定所需的插件以及（如果需要）要附加到这些插件的一个或多个策略：

```
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: '*'
```

您最多可以指定一个attachPolicy、attachPolicyARNs和serviceAccountRoleARN。

如果未指定这些策略，则将使用附加所有推荐策略的角色创建插件。

> **Note**
>
> 要将策略附加到插件，您的集群必须已OIDC启用。如果未启用，我们将忽略任何附加的政策。

然后，你可以在集群创建过程中创建以下插件：

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

或者在集群创建后使用配置文件或 CLI 标志显式创建插件：

```
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**
>
> 使用该--namespace-config标志将插件部署到自定义命名空间而不是默认命名空间。

在创建插件期间，如果集群上已经存在插件的自我管理版本，则可以通过配置文件设置resolveConflicts选项来选择如何解决潜在的configMap冲突，例如

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

对于插件创建，该resolveConflicts字段支持三个不同的值：

- none-EKS 不会更改该值。创建可能会失败。
- overwrite-EKS 会将所有配置更改改回 EKS 默认值。
- preserve-EKS 不会更改该值。创建可能会失败。（与更新插件类似none，但preserve有所不同）。

列出已启用的插件

你可以通过运行以下命令来查看集群中启用了哪些插件：

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

或者

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

设置插件的版本

设置插件的版本是可选的。如果该version字段留空，eksctl将解析插件的默认版本。有关哪个版本是特定插件的默认版本的更多信息，可以在有关 EKS 的 AWS 文档中找到。请注意，默认版本不一定是可用的最新版本。

插件版本可以设置为。latest或者，也可以使用指定的 EKS 构建标签来设置版本，例如v1.7.5-eksbuild.1或v1.7.5-eksbuild.2。也可以将其设置为插件的发行版本，例如v1.7.5或1.7.5，eksbuild后缀标签将被发现并为您设置。

有关如何发现可用插件及其版本的信息，请参阅以下部分。

发现插件

你可以通过运行以下命令来发现哪些插件可以安装在你的集群上：

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

这将发现你的集群的 kubernetes 版本并对其进行过滤。或者，如果你想查看特定的 kubernetes 版本有哪些插件可用，你可以运行：

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

您也可以通过筛选插件来发现插件。type owner and/or publisher例如，要查看特定所有者和类型的插件，你可以运行：

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

types、owners和 publishers flags 是可选的，可以一起指定，也可以单独指定以筛选结果。

发现插件的配置架构

发现插件和版本后，您可以通过获取其 JSON 配置架构来查看自定义选项。

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

这将返回此插件可用的各种选项的 JSON 架构。

使用配置值

ConfigurationValues可以在创建或更新插件期间在配置文件中提供。仅支持 JSON 和 YAML 格式。

例如 ，

```
addons:
```

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

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

> **Note**
>
> 请记住，修改插件配置值时，会出现配置冲突。

```
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`.
```

此外，get 命令现在还将检索ConfigurationValues插件。例如

```
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
```

使用自定义命名空间

在创建插件期间，可以在配置文件中提供自定义命名空间。一旦创建了插件，就无法更新命名空间。

使用配置文件

```
addons:
```

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

使用 CLI 标志

或者，您可以使用以下`--namespace-config`标志指定自定义命名空间：

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

get 命令还将检索插件的命名空间值

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

更新插件

您可以通过运行以下命令将插件更新到新版本并更改附加的政策：

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

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

> **Note**
>
> 一旦创建了插件，就无法更新命名空间配置。该`--namespace-config`标志仅在插件创建期间可用。

与创建插件类似，在更新插件时，您可以完全控制之前可能在该插件上应用的配置更改。configMap具体而言，您可以保留或覆盖它们。此可选功能可通过相同的配置文件字段获得resolveConflicts。例如，

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

对于插件更新，该resolveConflicts字段接受三个不同的值：

- none-EKS 不会更改该值。更新可能会失败。
- overwrite-EKS 会将所有配置更改改回 EKS 默认值。
- preserve-EKS 会保留该值。如果您选择此选项，我们建议您在更新生产集群上的插件之前，先在非生产集群上测试任何字段和值的更改。

删除插件

你可以通过运行以下命令来删除插件：

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

这将删除插件以及与之关联的任何 IAM 角色。

当您删除集群时，与插件关联的所有 IAM 角色也会被删除。

为默认网络插件灵活创建集群

创建集群后，EKS 会自动安装 VPC CNI、CoreDNS 和 kube-proxy 作为自管插件。为了禁用此行为以使用其他 CNI 插件，例如 Cilium 和 Calico，eksctl 现在支持创建没有任何默认网络插件的集群。要创建这样的集群，请进行设置addonsConfig.disableDefaultAddons，如下所示：

```
addonsConfig:
  disableDefaultAddons: true
```

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

要创建只有 CoreDNS 和 kube-proxy 而不是 VPC CNI 的集群，请在中明确指定并设置插件，如下所示：addonsaddonsConfig.disableDefaultAddons

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

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

作为此更改的一部分，如果未addonsConfig.disableDefaultAddons明确设置为 true，eksctl 现在会在集群创建期间将默认插件安装为 EKS 插件，而不是自行管理的插件。因此，不能再使用eksctl utils update-*命令更新使用 eksctl v0.184.0 及更高版本创建的集群的插件：

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

相反，现在eksctl update addon应该使用。

要了解更多信息，请参阅 Amazon EKS 为联网插件引入了集群创建灵活性。

启用 Amazon EMR 的访问权限

为了允许 EMR 在 Kubernetes API 上执行操作，需要向其 SLR 授予所需的 RBAC 权限。eksctl 提供了一个命令，用于为 EMR 创建所需的 RBAC 资源，并更新以将角色与 EMR 的 SLR 绑定。aws-auth ConfigMap

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

EKS Fargate Support

AWS Fargate 是一款适用于亚马逊 ECS 的托管计算引擎，可以运行容器。在 Fargate 中，您无需管理服务器或集群。

亚马逊 EKS 现在可以在 AWS Fargate 上启动 pod。这样就不必担心如何为 Pod 预置或管理基础设施，并且可以更轻松地在 AWS 上构建和运行高性能、高可用的 Kubernetes 应用程序。

创建支持 Fargate 的集群

您可以通过以下方式添加支持 Fargate 的集群：

```
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
```

此命令将创建集群和 Fargate 配置文件。此配置文件包含 AWS 在 Fargate 中实例化 pod 所需的某些信息。这些是：

- pod 执行角色，用于定义运行 Pod 所需的权限和运行 Pod 的网络位置（子网）。这允许将相同的网络和安全权限应用于多个 Fargate 容器，并且可以更轻松地将集群上的现有 pod 迁移到 Fargate。
- 用于定义应在 Fargate 上运行哪些 pod 的选择器。它由`namespace`和组成`labels`。

如果未指定配置文件但启用了对 Fargate 的支持，则会创建默认 `--fargate` Fargate 配置文件。此配置文件针对`default`和命名空间，因此这些命`kube-system`名空间中的 pod 将在 Fargate 上运行。

可以使用以下命令检查创建的 Fargate 配置文件：

```
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
```

要了解有关选择器的更多信息，请参阅[设计 Fargate 配置文件]。

使用配置文件创建支持 Fargate 的集群

以下配置文件声明一个 EKS 集群，其节点组由一个 EC2 `m5.large` 实例和两个 Fargate 配置文件组成。在`default`和`kube-system`命名空间中定义的所有 pod 都将在 Fargate 上运行。`dev`命名空间中所有同样带有该标签的 pod 也`dev=passed`将在 Fargate 上运行。任何其他 Pod 都将在中的节点上调度`ng-1`。

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

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

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

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

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

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

设计 Fargate 配置文件

每个选择器条目最多包含两个组件：命名空间和键值对列表。创建选择器条目只需要命名空间组件。所有规则（命名空间、键值对）都必须应用于 pod 才能匹配选择器条目。一个 pod 只需要匹配一个选择器条目即可在配置文件上运行。任何符合选择器字段中所有条件的 pod 都将被安排在 Fargate 上运行。任何与列入白名单的命名空间不匹配但用户手动设置调度器：fargate-scheduler 归档的 pod 都将处于待处理状态，因为它们无权在 Fargate 上运行。

配置文件必须满足以下要求：

- 每个配置文件必须有一个选择器
- 每个选择器都必须包含一个命名空间；标签是可选的

示例：在 Fargate 中调度工作负载

要在上面提到的示例中在 Fargate 上调度 pod，例如，可以创建一个名为的命名空间dev并将工作负载部署到那里：

```
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
```

从最后一条`kubectl get pods`命令的输出中，我们可以看到 nginx pod 部署在一个名为的节点中`fargate-ip-192-168-183-140.ap-northeast-1.compute.internal`。

管理 Fargate 个人资料

要在 Fargate 上部署 Kubernetes 工作负载，EKS 需要一个 Fargate 配置文件。在像上面的示例中那样创建集群时`eksctl`，请通过创建默认配置文件来解决这个问题。鉴于已经存在集群，也可以使用以下命令创建 Fargate 配置文件：`eksctl create fargateprofile`

> **Note**
>
> 仅在 EKS 平台版本`eks.5`或更高版本上运行的集群支持此操作。

> **Note**
>
> 如果现有版本是使用 0.11.0 `eksctl` 之前的版本创建的，则需要`eksctl upgrade cluster`在创建 Fargate 配置文件之前运行。

```
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"
```

您也可以指定要创建的 Fargate 配置文件的名称。此名称不得以前缀开头`eks-`。

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

使用带有 CLI 标志的此命令 eksctl 只能使用简单选择器创建单个 Fargate 配置文件。对于更复杂的选择器，例如具有更多命名空间的选择器，eksctl 支持使用配置文件：

```
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
```

要查看集群中的现有 Fargate 配置文件，请执行以下操作：

```
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
```

要以yaml格式查看它们，请执行以下操作：

```
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
```

或者采用以下json格式：

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

Fargate 配置文件在设计上是不可变的。要更改某些内容，请创建一个包含所需更改的新 Fargate 配置文件，然后使用eksctl delete fargateprofile命令删除旧的 Fargate 配置文件，如下例所示：

```
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"
}
```

请注意，删除个人资料的过程最多可能需要几分钟。如果未指定--wait标志，eksctl乐观地预计配置文件将被删除，并在发送 AWS API 请求后立即返回。要eksctl等到配置文件成功删除，请--wait像上面的示例一样使用。

延伸阅读

- AWS Fargate
- 亚马逊 EKS 现在可以在 AWS Fargate 上启动 pod

集群升级

通过 3 个简单步骤即可升级 “eksctl” 托管的集群：

1. 使用升级控制平面版本 `eksctl upgrade cluster`
2. 升级节点组
3. 更新默认网络插件（有关更多信息，请参阅the section called “默认插件更新”）：

仔细查看集群升级相关资源：

- 在 Amazon E@@ KS 用户指南中将现有集群更新到新的 Kubernetes 版本
- 《EKS 最佳实践指南》中的集群升级最佳实践

> **Note**
>
> 旧版本`eksctl update cluster`将被弃用。请改用 `eksctl upgrade cluster`。

更新控制平面版本

一次只能对一个次要版本进行控制平面版本升级。

要将控制平面升级到下一个可用版本，请运行：

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

此命令不会立即应用任何更改，您需要重新运行它`--approve`才能应用更改。

可以使用 CLI 标志指定集群升级的目标版本：

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

或者使用配置文件

```
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**
>
> `--version`和`metadata.version`参数的唯一允许值是集群的当前版本或更高的版本。不支持多个 Kubernetes 版本的升级。

默认插件更新

本主题介绍如何更新 EKS 集群中包含的默认预安装插件。

> **Warning**
>
> eksctl 现在将默认插件安装为 EKS 插件，而不是自行管理的插件。在默认网络插件的集群创建灵活性中阅读有关其含义的更多信息。
> 对于更新插件，`eksctl utils update-<addon>`不能用于使用 eksctl v0.184.0 及更高版本创建的集群。本指南仅对在此更改之前创建的集群有效。

每个 EKS 集群中都包含 3 个默认插件：

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

更新预安装的附加组件

对于通过集群创建`eksctl create addons`或创建集群时手动创建的官方 EKS 插件，管理它们的方法是通过`eksctl create/get/update/delete addon`。在这种情况下，请参阅有关 EKS 插件的文档。

更新每个命令的过程各不相同，因此需要运行 3 个不同的命令。以下所有命令都接受`--config-file`。默认情况下，这些命令中的每一个都以计划模式运行，如果您对建议的更改感到满意，请重新运行。`--approve`

要更新`kube-proxy`，请运行：

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

要更新`aws-node`，请运行：

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

要更新`coredns`，请运行：

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

升级后，一定要运行`kubectl get pods -n kube-system`并检查所有插件 pod 是否都处于就绪状态，你应该会看到这样的内容：

```
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 支持 EKS 集群中的区域移动

EKS 现在支持 Amazon 应用程序恢复控制器 (ARC) 区域转移和区域自动切换，从而增强了多可用区集群环境的弹性。借助 AWS Zonal Shift，客户可以将集群内的流量从受损的可用区域转移出去，从而确保新的 Kubernetes 容器和节点仅在运行良好的可用区中启动。

创建启用了区域偏移的集群

```
# 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
```

在现有集群上启用区域偏移

要在现有集群上启用或禁用区域切换，请运行

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

或者没有配置文件：

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

进一步信息

- EKS 区域移动

Karpenter Support

`eksctl`支持将 Karpenter 添加到新创建的集群。它将创建 Karpenter 的 “入门” 部分中概述的所有必要先决条件，包括使用 Helm 安装 Karpenter 本身。我们目前支持安装版本`0.28.0+`。有关更多详细信息，请参阅 Karpenter 兼容性部分。

以下集群配置概述了典型的 Karpenter 安装：

```
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
```

该版本是 Karpenter 的版本，可以在他们的 Helm Repository 中找到。也可以设置以下选项：

```
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
```

必须定义 OIDC 才能安装 Karpenter。

成功安装 Karpenter 后，添加 NodePool(s) 和 NodeClass(es) 以允许 Karpenter 开始向集群添加节点。

NodePool的`nodeClassRef`部分必须与 a 的名称匹配`EC2NodeClass`。例如：

```
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
```

请注意，必须`instanceProfile`为启动节点指定`role`或之一。如果您选择使用`instanceProfile`由创建的配置文件的名称，请`eksctl`遵循以下模式:`eksctl-KarpenterNodeInstanceProfile-<cluster-name>`.

自动安全组标记

`eksctl`当同时启用（`karpenter.version`指定）Karpenter 且标签存在于中时，会自动为集群的共享节点安全组`karpenter.sh/discovery`添加标签`karpenter.sh/discovery`。`metadata.tags`这样可以兼容 AWS Load Balancer 控制器。

请注意，在 karpenter 0.32.0+ 中，Provisioners 已被弃用并替换为。NodePool

集群 Config 架构

> **Note**
>
> 架构的位置当前正在迁移中。

您可以使用 yaml 文件创建集群。查看架构参考。

例如：

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

此文件的架构参考可在上找到 GitHub。

有关使用该文件的更多信息，请参阅the section called “创建和管理集群”。

节点组

本章包含有关如何使用 Eksctl 创建和配置节点组的信息。节点组是连接到 EKS 集群的 EC2 实例组。

主题：

- the section called “Spot 实例”
  - 使用托管节点组创建和管理带有 Spot 实例的 EKS 集群
  - 使用为非托管节点组配置 Spot 实例 MixedInstancesPolicy
  - 使用 `node-lifecycle` Kubernetes 标签区分竞价型实例和按需实例
- the section called “Auto Scaling”
  - 通过创建允许使用集群自动扩缩程序的 IAM 角色的集群或节点组，启用 Kubernetes 集群节点的自动扩展
  - 配置节点组定义以包括集群自动扩缩器扩展节点组所需的标签和注释
  - 如果工作负载具有区域特定的要求（例如区域特定的存储或关联性规则），则为每个可用区创建单独的节点组
- the section called “EKS 托管的节点组”
  - 为 Amazon EKS Kubernetes 集群配置和管理 EC2 实例（节点）
  - 轻松将错误修复、安全补丁和更新节点应用到最新的 Kubernetes 版本
- the section called “EKS 混合节点”
  - 使用与 AWS 云中使用的 AWS EKS 集群、功能和工具相同的 AWS EKS 集群、功能和工具，支持在客户托管的基础设施上运行本地和边缘应用程序
  - 使用 AWS VP Site-to-Site N 或 AWS Direct Connect 等选项配置联网以将本地网络连接到 AWS VPC
  - 使用 AWS Systems Manager (SSM) 或 AWS IAM Roles Anywhere 为远程节点设置凭证，以便通过 EKS 集群进行身份验证
- the section called “节点修复 Config”
  - 启用 EKS 托管节点组的节点修复以自动监控、替换或重启不健康的工作节点
- the section called “ARM Support”
  - 使用基于 ARM 的 Graviton 实例创建 EKS 集群，以提高性能和成本效益
- the section called “污点”

  - 将污点应用于 Kubernetes 集群中的特定节点组
  - 根据污点键、值和效果控制 Pod 的调度和驱逐
- the section called “启动模板支持”
  - 使用提供的 EC2 启动模板启动托管节点组
  - 升级托管节点组以使用其他版本的启动模板
  - 了解在托管节点组中使用自定义模板 AMIs 和启动模板时的限制和注意事项
- the section called “使用节点组”
  - 启用对节点组中的 EC2 实例的 SSH 访问权限
  - 向上或向下扩展节点组中的节点数量
- the section called “自定义子网”
  - 使用新子网扩展现有 VPC 并向该子网添加节点组
- the section called “节点引导”
  - 了解 2023 年推出的新节点初始化流程 (nodeadm) AmazonLinux
  - 了解 eksctl 为自我管理和 EKS 管理的节点应用的默认 NodeConfig 设置
  - 通过提供自定义，自定义节点引导流程 overrideBootstrapCommand NodeConfig
- the section called “非托管节点组”
  - 在 EKS 集群中创建或更新非托管节点组
  - 更新默认 Kubernetes 插件，比如 kube-proxy、aws-node 和 CoreDNS
- the section called “GPU Suppor”
  - Eksctl 支持为节点组选择 GPU 实例类型，从而允许在 EKS 集群上使用 GPU 加速的工作负载。
  - 选择支持 GPU 的实例类型后，Eksctl 会自动安装 NVIDIA Kubernetes 设备插件，从而便于在集群中使用 GPU 资源。
  - 用户可以禁用自动插件安装，并使用提供的命令手动安装特定版本的 NVIDIA Kubernetes 设备插件。
- the section called “实例选择器”
  - 根据资源标准（例如 v CPUs、内存和 CPU 架构）自动生成合适的 EC2 实例类型列表 GPUs
  - 使用符合指定实例选择器标准的实例类型创建集群和节点组
  - 在创建节点组之前，执行试运行以检查和修改与实例选择器匹配的实例类型
- the section called “其他卷映射”
  - 为 EKS 集群中的托管节点组配置其他卷映射
  - 自定义其他卷的大小、类型、加密、IOPS 和吞吐量等卷属性

  - 将现有 EBS 快照作为额外卷附加到节点组
- the section called “Windows 工作节点”
  - 将 Windows 节点组添加到现有的 Linux Kubernetes 集群以支持运行 Windows 工作负载
  - 使用基于和标签的节点选择器在相应的操作系统（Windows 或 Linux）上安排工作负载 `kubernetes.io/os` `kubernetes.io/arch`
- the section called “自定义 AMI 支持”
  - 使用该`--node-ami`标志为节点组指定自定义 AMI，向 AWS 查询最新 EKS 优化的 AMI，或者使用 AWS Systems Manager Parameter Systems Store 查找 AMI。
  - 设置该`--node-ami-family`标志以指定节点组 AMI 的操作系统系列，例如 AmazonLinux 2、Ubuntu2204 或 2022。 WindowsServer CoreContainer
  - 对于 Windows 节点组，请指定自定义 AMI 并通过提供 PowerShell 引导脚本。`overrideBootstrapCommand`
- the section called “自定义 DNS”
  - 覆盖用于内部和外部 DNS 查找的 DNS 服务器 IP 地址

使用节点组

创建节点组

除了与集群一起创建的初始节点组外，您还可以添加一个或多个节点组。

要创建其他节点组，请使用：

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

> **Note**
>
> `--version`托管节点组不支持标志。它总是从控制平面继承版本。

默认情况下，新的非托管节点组从控制平面继承版本 (`--version=auto`)，但您可以指定其他版本，也可以使用`--version=latest`强制使用最新版本。

此外，您可以将与以下内容相同的配置文件用于`eksctl create cluster`以下用途：

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

从配置文件创建节点组

也可以通过集群定义或配置文件创建节点组。给出以下示例配置文件和一个名为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
```

ng-2-builders可以使用以下命令创建节点组ng-1-workers和：

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

负载均衡

如果您已经准备好将现有的经典负载均衡器 or/and 目标组附加到节点组，则可以在配置文件中指定这些目标组。创建节点组时，经典负载均衡器 or/and 目标组会自动与 ASG 关联。只有通过该字段定义的自我管理节点组才支持此功能。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
```

配置文件中的节点组选择

要仅对配置文件中指定的节点组的子集执行create或delete操作，有两个接受全局列表的 CLI 标志0，1例如：

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

使用上面的示例配置文件，可以使用以下命令创建除工作节点组之外的所有工作节点组：

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

或者可以通过以下方式删除构建器节点组：

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

在这种情况下，我们还需要提供--approve命令来实际删除节点组。

包含和排除规则

- 如果未指定--include或--exclude，则包含所有内容
- 如果仅指定，--include则仅包含与这些全局匹配的节点组
- 如果只指定--exclude了，则包括所有与这些全局不匹配的节点组
- 如果同时指定了两者，则--exclude规则优先于--include（即两个组中与规则匹配的节点组将被排除在外）

列出节点组

要列出有关一个节点组或所有节点组的详细信息，请使用：

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

要以 YAML 或 JSON 格式列出一个或多个节点组，这些节点组输出比默认日志表更多的信息，请使用：

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

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

节点组不可变性

从设计上讲，节点组是不可变的。这意味着，如果您需要更改诸如 AMI 或节点组的实例类型之类的内容（扩展除外），则需要创建一个具有所需更改的新节点组，移动负载并删除旧的节点组。请参阅 “删除和清空节点组” 部分。

缩放节点组

节点组扩展过程最多可能需要几分钟。如果未指定该--wait标志，则eksctl乐观地期望节点组能够扩展，并在发送 AWS API 请求后立即返回。要eksctl等到节点可用，请添加一个--wait标志，如下例所示。

> **Note**
>
> 缩放节点组 down/in （即减少节点数量）可能会导致错误，因为我们完全依赖于对 ASG 的更改。这意味着被清空的节点并 removed/terminated 未被显式耗尽。这可能是 future 需要改进的领域。

扩展托管节点组是通过直接调用更新托管节点组配置的 EKS API 来实现的。

缩放单个节点组

可以使用以下命令缩放节点组：eksctl scale nodegroup

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

例如，要将节点组ng-a345f4e1扩展cluster-1到 5 个节点，请运行：

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

也可以使用传递给的配置文件来缩放节点组，--config-file并指定应使用其进行缩放的节点组的名称。--nameEksctl 将搜索配置文件并发现该节点组及其缩放配置值。

如果所需的节点数在当前最小和当前最大节点数的范围NOT内，则会显示一个特定的错误。这些值也可以--nodes-max分别与标志--nodes-min和一起传递。

扩展多个节点组

Eksctl 可以发现和缩放随传递的配置文件中找到的所有节点组。--config-file

与扩展单个节点组类似，同一组验证适用于每个节点组。例如，所需的节点数必须在最小和最大节点数的范围内。

删除和清空节点组

要删除节点组，请运行：

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

包含和排除规则也可以与该命令一起使用。

> **Note**
>
> 这将在删除实例之前耗尽该节点组中的所有 Pod。

要在排空过程中跳过驱逐规则，请运行：

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

所有节点都被封锁，所有的 pod 在删除时都会被逐出节点组，但是如果你需要在不删除节点组的情况下将其清空，请运行：

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

要解除对节点组的封锁，请运行：

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

要忽略诸如 PodDisruptionBudget 设置之类的驱逐规则，请运行：

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

为了加快排水过程，您可以指定要`--parallel <value>`并行排出的节点数。

其他功能

您还可以为节点组启用 SSH、ASG 访问和其他功能，例如：

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

更新标签

里面没有特定的命令eksctl来更新节点组的标签，但是可以很容易地使用以下方法来实现kubectl，例如：

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

SSH 访问权限

您可以通过在节点组配置publicKeyPath中配置publicKeyName和来为节点组启用 SSH 访问。publicKey或者，您可以使用 AWS Systems Manager (SSM) 通过 SSH 连接到节点，方法是为节点组配置以下内容：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
```

非托管节点组

在中eksctl，设置--managed=false或使用该nodeGroups字段会创建非托管节点组。请记住，非托管节点组不会出现在 EKS 控制台中，EKS 控制台通常只知道 EKS 管理的节点组。

只有在运行之后才应该升级节点组。eksctl upgrade cluster（请参阅升级集群。）

如果你有一个只有初始节点组的简单集群（即用创建的eksctl create cluster），则过程非常简单：

1. 获取旧节点组的名称：

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

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

2. 创建一个新的节点组：

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

3. 删除旧的节点组：

```
 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.
```

更新多个节点组

如果您有多个节点组，则您有责任跟踪每个节点组是如何配置的。你可以使用配置文件来做到这一点，但是如果你还没有使用过，则需要检查你的集群以了解每个节点组是如何配置的。

一般而言，您希望：

- 查看您拥有哪些节点组以及哪些节点组可以删除或必须替换为新版本
- 记下每个节点组的配置，下次可以考虑使用配置文件来简化升级

使用配置文件进行更新

如果您使用的是配置文件，则需要执行以下操作。

编辑配置文件以添加新的节点组，并删除旧的节点组。如果您只想升级节点组并保持相同的配置，则只需更改节点组名称即可，例如在名称后面追加-v2。

要创建配置文件中定义的所有新节点组，请运行：

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

有了新的节点组后，就可以删除旧的节点组：

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

> **Note**
>
> 第一次运行处于计划模式，如果您对建议的更改感到满意，请重新运行。--approve

更新默认插件

您可能需要更新集群上安装的网络插件。有关更多信息，请参阅 the section called “默认插件更新”。

EKS 托管的节点组

Amazon EKS 托管节点组是一项功能，可自动为 Amazon EKS Kubernetes 集群配置节点（EC2 实例）进行配置和生命周期管理。客户可以为其集群配置优化的节点组，EKS 将使用最新的 Kubernetes 和主机操作系统版本使其节点保持最新状态。

EKS 托管节点组是一个自动扩缩组和相关的 EC2 实例，由 AWS 为 Amazon EKS 集群管理。每个节点组都使用亚马逊 EKS 优化的亚马逊 Linux 2 AMI。借助 Amazon EKS，您可以轻松地将错误修复和安全补丁应用于节点，也可以将其更新到最新的 Kubernetes 版本。每个节点组都会为您的集群启动一个自动扩缩组，该组可以跨越多个 AWS VPC 可用区和子网，以实现高可用性。

新增托管节点组启动模板支持

> **Note**
>
> “非托管节点组” 一词用于指代 eksctl 从一开始就支持的节点组（通过字段表示）。`nodeGroups`该`ClusterConfig`文件继续使用该`nodeGroups`字段来定义非托管节点组，托管节点组是用该字段定义的。`managedNodeGroups`

创建托管节点组

```
$ eksctl create nodegroup
```

新集群

要使用托管节点组创建新集群，请运行

```
eksctl create cluster
```

要创建多个托管节点组并更好地控制配置，可以使用配置文件。

> **Note**
>
> 托管节点组与非托管节点组的功能不完全相同。

```
# 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
```

可以在此处找到用于创建托管节点组的配置文件的另一个示例。

集群可以同时包含托管节点组和非托管节点组。非托管节点组不会显示在 AWS EKS 控制台中，但`eksctl get nodegroup`会列出这两种类型的节点组。

```
# 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
```

全新 Support 支持自定义 AMI `instancePrefix`、安全组`instanceNameebsOptimized`、`volumeType`、`volumeName`、、`volumeEncrypted`、`volumeKmsKe`和 `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'
```

如果您请求的实例类型仅在一个区域中可用（eksctl 配置要求指定两个区域），请务必将可用区添加到您的节点组请求中：

```
# 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
```

对于像 Hpc6 系列这样仅在一个区域中可用的实例类型，情况可能就是这样。

现有集群

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

提示：如果您使用`ClusterConfig`文件来描述整个集群，请在`managedNodeGroups`字段中描述您的新托管节点组并运行：

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

升级托管节点组

您可以随时将节点组更新到适用于您正在使用的 AMI 类型的最新 EKS 优化的 AMI 发行版本。

如果您的节点组与集群的 Kubernetes 版本相同，则可以将您正在使用的 AMI 类型的 Kubernetes 版本更新到最新 AMI 发行版本。如果你的节点组是集群 Kubernetes 版本中的先前 Kubernetes 版本，你可以将节点组更新到与节点组的 Kubernetes 版本相匹配的最新 AMI 发行版本，或者更新到与集群 Kubernetes 版本匹配的最新 AMI 发行版本。你无法将节点组回滚到较早的 Kubernetes 版本。

要将托管节点组升级到最新的 AMI 发行版本，请执行以下操作：

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

可以使用以下命令将节点组升级到指定 Kubernetes 版本的最新 AMI 版本：

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

要升级到特定的 AMI 发行版本而不是最新版本，请传递`--release-version`：

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

> **Note**
>
> 如果托管节点是使用自定义部署的 AMIs，则必须遵循以下工作流程才能部署新版本的自定义 AMI。

- 节点组的初始部署必须使用启动模板完成。例如

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

- 创建自定义 AMI 的新版本（使用 AWS EKS 控制台）。
- 使用新的 AMI ID 创建新的启动模板版本（使用 AWS EKS 控制台）。
- 将节点升级到启动模板的新版本。例如

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

处理节点的并行升级

可以同时升级多个托管节点。要配置并行升级，请在创建节点组时定义节点组的。updateConfigupdateConfig可以在这里找到一个例子。

为避免由于一次升级多个节点而导致工作负载停机，您可以通过在maxUnavailable字段中指定此项来限制升级期间可能不可用的节点数量updateConfig。或者maxUnavailablePercentage，使用，它将不可用节点的最大数量定义为节点总数的百分比。

请注意，该值maxUnavailable不能高于maxSize。此外，maxUnavailable和maxUnavailablePercentage不能同时使用。

此功能仅适用于托管节点。

更新托管节点组

eksctl允许更新托管节点组的UpdateConfig部分。本节定义了两个字段。MaxUnavailable和MaxUnavailablePercentage。更新期间，您的节点组不会受到影响，因此预计不会出现停机时间。

该命令update nodegroup应与使用该--config-file标志的配置文件一起使用。节点组应包含一个部分。nodeGroup.updateConfig更多信息可以在这里找到。

Nodegroup Health 问题

EKS Managed Nodegroups 会自动检查您的节点组和节点的配置是否存在运行状况问题，并通过 EKS API 和控制台报告这些问题。要查看节点组的运行状况问题，请执行以下操作：

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

管理标签

EKS 托管节点组支持附加应用于节点组中的 Kubernetes 节点的标签。这是在创建集群或节点组期间通过 eksctl 中的labels字段指定的。

要在节点组上设置新标签或更新现有标签，请执行以下操作：

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

要取消设置或移除节点组中的标签，请执行以下操作：

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

要查看在节点组上设置的所有标签，请执行以下操作：

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

扩展托管节点组

eksctl scale nodegroup还支持托管节点组。扩展托管或非托管节点组的语法相同。

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

进一步信息

- EKS 托管节点组

节点引导

AmazonLinux2023

AL2023 引入了一个新的节点初始化进程 nodeadm，它使用 YAML 配置架构，放弃了脚本的使用。/etc/eks/bootstrap.sh

> **Note**
>
> 在 Kubernetes 1.30 及更高版本中，亚马逊 Linux 2023 是默认操作系统。

的默认设置 AL2

对于自管节点和基于自定义的 EKS 管理节点 AMIs，eksctl创建一个默认的最小值，NodeConfig然后自动将其注入到节点组的启动模板用户数据中。即

```
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

--//--
```

对于基于原生的 EKS 托管节点 AMIs，默认NodeConfig值由 EKS MNG 在后台添加，直接附加到 EC2 的用户数据中。因此，在这种情况下，eksctl无需将其包含在启动模板中。

配置引导过程

要设置高级属性或干脆覆盖默认值，eksctl 允许您NodeConfig通过nodeGroup.overrideBootstrapCommand或例如，指定自定义 NodeConfig managedNodeGroup.overrideBootstrapCommand

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

此自定义配置将由 eksctl 添加到用户数据之前，并与默认配置合并。nodeadm在此处阅读有关合并多个配置对象nodeadm的功能的更多信息。

启动托管节点组的模板支持

eksctl 支持使用提供的 EC2 启动模板启动托管节点组。这为节点组启用了多个自定义选项，包括提供自定义组 AMIs 和安全组，以及为节点引导传递用户数据。

使用提供的启动模板创建托管节点组

```
# 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
```

升级托管节点组以使用不同的启动模板版本

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

> **Note**
>
> 如果启动模板使用自定义 AMI，则新版本也应使用自定义 AMI，否则升级操作将失败

如果启动模板未使用自定义 AMI，也可以指定要升级到的 Kubernetes 版本：

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

关于自定义 AMI 和启动模板支持的注意事项

- 提供启动模板时，不支持以下字段：`instanceType`、、、、`ami`、`ssh.allow`、`ssh.sourceSecurityGroupIds`、`securityGroups`
- 使用自定义 AMI (`ami`) 时，还`overrideBootstrapCommand`必须设置为执行引导。
- `overrideBootstrapCommand`只能在使用自定义 AMI 时进行设置。
- 提供启动模板时，节点组配置中指定的标签仅适用于 EKS 节点组资源，不会传播到 EC2 实例。

自定义子网

可以使用新子网扩展现有 VPC，然后向该子网添加节点组。

为什么

如果集群用完了预先配置的集群 IPs，则可以使用新的 CIDR 调整现有 VPC 的大小，以便向其添加新的子网。要了解如何做到这一点，请阅读 AWS Extending 上的这份指南 VPCs。

TL; DR

转到 VPC 的配置并添加，单击 “操作”-> “编辑”， CIDRs 然后添加一个新范围。例如：

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

现在你需要添加一个新的子网。根据它是新的私有子网还是公有子网，您必须分别从私有子网或公有子网复制路由信息。

创建子网后，添加路由，然后从 VPC 中的另一个子网复制 NAT 网关 ID 或 Internet Gateway。请注意，如果它是公有子网，请启用自动 IP 分配。操作->修改自动分配 IP 设置->启用自动分配公共地址。IPv4

别忘了根据公有或私有子网的配置复制现有子网的标签。这一点很重要，否则子网将不是集群的一部分，子网中的实例将无法加入。

完成后，复制新子网的 ID。根据需要经常重复。

操作方法

要在创建的子网中创建节点组，请运行以下命令：

```
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
```

或者，使用这样的配置：

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

使用这样的配置：

```
# 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
```

等待节点组创建，新实例应具有子网的新 IP 范围。

删除集群

由于新增内容通过在 CloudFormation 堆栈之外添加依赖项来修改现有 VPC，因此 CloudFormation无法再移除集群。

在删除集群之前，请手动移除所有已创建的额外子网，然后调用`eksctl`以下命令继续：

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

自定义 DNS

有两种方法可以覆盖用于所有内部和外部 DNS 查找的 DNS 服务器 IP 地址。这等同于的`--cluster-dns`标志`kubelet`。

首先，是穿过`clusterDNS`田野。Config 文件接受一个名为、`clusterDNS`带有 DNS 服务器的 IP 地址的`string`字段以供使用。这将传递给 pod`kubelet`，然后再通过`/etc/resolv.conf`文件将其传递给 pod。有关更多信息，请参阅配置文件的架构。

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

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

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

请注意，此配置仅接受一个 IP 地址。要指定多个地址，请使用以下`kubeletExtraConfig`参数：

```
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"]
```

污点

要将污点应用于特定的节点组，请使用如下所示的`taints`配置部分：

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

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

完整的例子可以在这里找到。

实例选择器

eksctl 支持为托管节点组和自我管理节点组指定多种实例类型，但是由于有 270 多种 EC2 实例类型，用户必须花时间弄清楚哪些实例类型最适合他们的节点组。使用竞价型实例就更难了，因为你需要选择一组能很好地与集群自动扩缩器配合使用的实例。

eksctl 现在与 EC2 实例选择器集成，该选择器通过根据资源标准（v CPUs、内存、# GPUs 和 CPU 架构）生成实例类型列表来解决这个问题。通过实例选择器标准后，eksctl 会创建一个节点组，其实例类型设置为符合所提供标准的实例类型。

创建集群和节点组

要创建具有单个节点组的集群，该节点组使用的实例类型与传递给 eksctl 的实例选择器资源标准相匹配，请运行

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

这将创建一个集群和一个托管节点组，`instanceTypes`字段设置为`[c5.large, c5a.large, c5ad.large, c5d.large, t2.medium, t3.medium, t3a.medium]`（返回的实例类型集可能会发生变化）。

对于非托管节点组，将设置以下`instancesDistribution.instanceTypes`字段：

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

实例选择器标准也可以在以下位置指定 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
```

和支持以下实例选择器 CLI 选项`eksctl create nodegroup`：`eksctl create cluster`

`--instance-selector-vcpus`、`--instance-selector-memory`、`--instance-selector-gpus` 和 `instance-selector-cpu-architecture`

可以在此处找到示例文件。

试跑

试运行功能允许您在继续创建节点组之前检查和更改与实例选择器匹配的实例。

```
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
```

然后 ClusterConfig 可以将生成的数据传递给`eksctl create cluster`：

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

代表 CLI 选项的`instanceSelector`字段也将添加到 ClusterConfig 文件中，以供查看和记录之用。省略`--dry-run`时，该字段将被忽略并使用该`instanceTypes`字段，否则对的任何更改都将被 ek `instanceTypes` sctl 覆盖。

通过传递 ClusterConfig 文件时`--dry-run`，eksctl 将在扩展每个节点组的实例选择器资源标准后，输出一个包含相同节点组集的 ClusterConfig 文件。

```
# 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
# ...
```

Spot 实例

托管节点组

eksctl支持使用 EKS 托管节点组的竞价型工作节点，该功能允许拥有容错应用程序的 EKS 客户轻松地为其 EKS 集群配置和管理 EC2 竞价型实例。EKS Managed Nodegroup 将按照竞价最佳实践配置和启动由竞价型实例组成的 EC2 Autoscaling 组，并在实例被 AWS 中断之前自动耗尽竞价型工作节点。使用此功能不收取增量费用，客户只需为使用 AWS 资源（例如 EC2 竞价型实例和 EBS 卷）付费。

要使用竞价型实例创建带有托管节点组的集群，请传递该--spot标志和可选的实例类型列表：

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

要在现有集群上使用竞价型实例创建托管节点组，请执行以下操作：

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

要通过配置文件使用托管节点组创建 Spot 实例，请执行以下操作：

```
# 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**
>
> 非托管节点组不支持spot和instanceTypes字段，而是使用该instancesDistribution字段来配置竞价型实例。见下文

进一步信息

- EKS Spot 节点组
- EKS 托管节点组容量类型

非托管节点组

eksctl已通过 Auto Scaling Grou MixedInstancesPolicy ps 支持竞价型实例。

以下是使用 50% 竞价型实例和 50% 按需实例的节点组示例：

```
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
```

请注意，使用该nodeGroups.X.instanceType字段时不应设置该instancesDistribution字段。

此示例使用 GPU 实例：

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

此示例使用容量优化的竞价分配策略：

```
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"
```

此示例使用现 capacity-optimized-prioritized货分配策略：

```
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"
```

使用capacity-optimized-prioritized分配策略，然后在启动模板覆盖列表中按从最高到最低优先级（列表中的第一个到最后）设置实例类型的顺序。Amazon EC2 Auto Scaling 在尽力而为的基础上尊重实例类型优先级，但会首先针对容量进行优化。对于必须最大限度地减少中断可能性的工作负载来说，这是一个不错的选择，但对某些实例类型的偏好也很重要。有关更多信息，请参阅 ASG 购买选项。

请注意，使用该spotInstancePools字段时不应设置该spotAllocationStrategy字段。如果spotAllocationStrategy未指定，则 EC2 将默认使用该lowest-price策略。

以下是一个最小的例子：

```
nodeGroups:
  - name: ng-1
    instancesDistribution:
```

```
      instanceTypes: ["t3.small", "t3.medium"] # At least one instance type should be
specified
```

要区分竞价型实例和按需型实例之间的节点，您可以使用 kubernetes 标签`node-lifecycle`，该标签将具有值`spot`或`on-demand`取决于其类型。

实例分布中的参数

有关详细信息，请参阅集群配置架构。

GPU Suppor

Eksctl 支持为节点组选择 GPU 实例类型。只需向 create 命令或通过配置文件提供兼容的实例类型即可。

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

> **Note**
>
> 无需再订阅市场 AMI 即可获得 EKS 上的 GPU 支持。

AMI 解析器（`auto`和`auto-ssm`）将看到您要使用 GPU 实例类型，他们将选择正确的 EKS 优化加速 AMI。

Eksctl 将检测到已选择了支持 GPU 的实例类型的 AMI，并将自动安装 NVIDIA Kubernetes 设备插件。

> **Note**
>
> Windows和Ubuntu AMIs 没有安装GPU驱动程序，因此运行GPU加速的工作负载将无法开箱即用。

要禁用自动插件安装并手动安装特定版本，请`--install-nvidia-plugin=false`使用 create 命令。例如：

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

而且，对于 0.15.0 及更高版本，

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

或者，对于较旧的版本，

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

如果集群仅包含 Bott lerocket 节点组，则将跳过 NVIDIA Kubernetes 设备插件的安装，因为 Bottlerocket 已经在处理设备插件的执行。如果您在集群配置中使用不同的 AMI 系列，则可能需要使用污点和容忍度来阻止设备插件在 Bottlerocket 节点上运行。

ARM Support

本主题介绍如何使用 ARM 节点组创建集群，以及如何向现有集群添加 ARM 节点组。

EKS 的 G raviton 处理器支持 64 位 ARM 架构。要创建集群，请选择一种基于 Graviton 的实例类型（a1、、、、、t4g、m6g、、m7g、m6gd、、c6g、c7g、c6gd、、r6g、r7g、、r6gdm8gr8g、c8g）并运行：

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

或者使用配置文件：

```
---
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
```

托管节点组也支持 ARM：

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

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

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

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

AMI 解析器`auto`和`auto-ssm`，将根据 ARM 实例类型推断出正确的 AMI。只有 AmazonLinux 2023、AmazonLinux 2 和 Bottlerocket 系列对 ARM 进行了优化 AMIs EKS。

> **Note**
>
> 1.15 及更高版本的集群支持 ARM。

Auto Scaling

启用 Auto Scaling

您可以使用 IAM 角色创建集群（或现有集群中的节点组），该角色将允许使用集群自动扩缩器：

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

此标志还会设置`k8s.io/cluster-autoscaler/enabled`和`k8s.io/cluster-autoscaler/<clusterName>`标记，因此节点组发现应该可以正常工作。

集群运行后，您需要自行安装集群自动扩缩器。

您还应在托管或非托管节点组定义中添加以下内容，以添加集群自动扩缩器扩展节点组所需的标签：

```
nodeGroups:
```

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

从 0 向上扩展

如果您希望能够从 0 向上扩展节点组，并且在节点组上定义了标签 and/or 污点，则需要将这些标签作为标签传播到 Auto Scaling Groups () 上。ASGs

一种方法是在节点组定义的`tags`字段中设置 ASG 标签。例如，给定一个带有以下标签和污点的节点组：

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

您需要添加以下 ASG 标签：

```
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"
```

对于托管节点组和非托管节点组，都可以通过将设置为来自动完成此操作`true`，这会将标签和污点作为标签添加`propagateASGTags`到 Auto Scaling 组中：

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

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

区域感知型 Auto Scaling

如果您的工作负载是特定于区域的，则需要为每个区域创建单独的节点组。这是因为cluster-autoscaler假设组中的所有节点完全相同。因此，例如，如果扩容事件是由需要区域特定的 PVC（例如 EBS 卷）的 pod 触发的，则新节点可能会被调度到错误的可用区，并且 Pod 将无法启动。

如果您的环境符合以下标准，则无需为每个 AZ 设置单独的节点组：

- 没有区域特定的存储要求。
- 除主机以外的拓扑结构无需使用 PodaFfinity。
- 区域标签上没有必需的 NodeAffinity。
- 区域标签上没有 nodeSelector。

（在这里和这里阅读更多。）

如果您满足上述所有要求（可能还满足其他要求），那么使用跨越多个节点组的单个节点组应该是安全的。 AZs否则，您需要创建单独的单可用区节点组：

之前：

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

之后：

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

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

自定义 AMI 支持

设置节点 AMI ID

该--node-ami标志支持许多高级用例，例如使用自定义 AMI 或实时查询 AWS 以确定要使用哪个 AMI。该标志可用于非 GPU 和 GPU 映像。

该标志可以获取要明确使用的图像的 AMI 图像 ID。它也可以使用以下 “特殊” 关键字：

Keyword	说明
自动	表示应通过查询 AWS EC2 来找到用于节点的 AMI。这与 auto 解析器有关。
auto-ssm	表示应通过查询 AWS SSM Parameter Store 来找到用于节点的 AMI。

> **Note**
>
> 目前，EKS 托管节点组在使用自定义节点组时仅支持以下 AMI 系列 AMIs：AmazonLinux2023、、AmazonLinux2、BottlerocketUbuntu2004、和 UbuntuPro2004 Ubuntu2204 Ubuntu2404

设置--node-ami为 ID 字符串时，eksctl将假设已请求自定义 AMI。对于 AmazonLinux 2 和 Ubuntu 节点（包括 EKS 托管和自我管理），这意味着overrideBootstrapCommand这是必需的。 AmazonLinux2023 年，由于它停止使用/etc/eks/bootstrap.sh脚本进行节点引导，因此不支持使用 nodeadm 初始化过程（欲了解更多信息，请参阅节点引导文档）。overrideBootstrapCommand

CLI 标志示例：

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

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

Config 文件示例：

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

该`--node-ami`标志也可以与一起使用`eksctl create nodegroup`。

设置节点 AMI 系列

`--node-ami-family`可以采用以下关键字：

Keyword	说明
AmazonLinux2	表示应使用基于亚马逊 Linux 2 的 EKS AMI 镜像（默认）。
AmazonLinux2023	表示应使用基于亚马逊 Linux 2023 的 EKS AMI 镜像。
Ubuntu2004	表示应使用基于 Ubuntu 20.04 LTS（Focal）的 EKS AMI 镜像（EKS 1.29 支持）。
UbuntuPro2004	表示应使用基于 Ubuntu Pro 20.04 LTS（Focal）的 EKS AMI 图像（适用于 >= 1.27、1.29 的 EKS）。

Keyword	说明
Ubuntu2204	表示应使用基于 Ubuntu 22.04 LTS (Jammy) 的 EKS AMI 镜像（适用于 >= 1.29 的 EKS）。
UbuntuPro2204	表示应使用基于 Ubuntu Pro 22.04 LTS（Jammy）的 EKS AMI 镜像（适用于 >= 1.29 的 EKS）。
Ubuntu2404	表示应使用基于 Ubuntu 24.04 LTS（Noble）的 EKS AMI 镜像（适用于 >= 1.31 的 EKS）。
UbuntuPro2404	表示应使用基于 Ubuntu Pro 24.04 LTS（Noble）的 EKS AMI 镜像（适用于 >= 1.31 的 EKS）。
Bottlerocket	表示应使用基于 Bottlerocket 的 EKS AMI 镜像。
WindowsServer2019 FullContainer	表示应使用基于 Windows Server 2019 完整容器的 EKS AMI 镜像。
WindowsServer2019 CoreContainer	表示应使用基于 Windows Server 2019 核心容器的 EKS AMI 镜像。
WindowsServer2022 FullContainer	表示应使用基于 Windows Server 2022 完整容器的 EKS AMI 镜像。
WindowsServer2022 CoreContainer	表示应使用基于 Windows Server 2022 核心容器的 EKS AMI 镜像。

CLI 标志示例：

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

Config 文件示例：

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

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

该--node-ami-family标志也可以与一起使用eksctl create nodegroup。 eksctl每当使用自定义 AMI 时，都需要通过配置文件或 --node-ami-family CLI 标志明确设置 AMI 系列。

> **Note**
>
> 目前，EKS 托管节点组在使用自定义节点组时仅支持以下 AMI 系列 AMIs：AmazonLinux2023、、AmazonLinux2、BottlerocketUbuntu2004、和 UbuntuPro2004 Ubuntu2204 Ubuntu2404

Windows 自定义 AMI 支持

只有自我管理的 Windows 节点组才能指定自定义 AMI。 amiFamily应设置为有效的 Windows AMI 系列。

以下 PowerShell 变量将可供引导脚本使用：

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

Config 文件示例：

```
nodeGroups:
  - name: custom-windows
    amiFamily: WindowsServer2022FullContainer
    ami: ami-01579b74557facaf7
```

```
    overrideBootstrapCommand: |
      & $EKSBootstrapScriptFile -EKSClusterName "$EKSClusterName" -APIServerEndpoint
"$APIServerEndpoint" -Base64ClusterCA "$Base64ClusterCA" -ContainerRuntime
"containerd" -KubeletExtraArgs "$KubeletExtraArgs" 3>&1 4>&1 5>&1 6>&1
```

Bottlerocket 自定义 AMI 支持

对于 Bottlerocket 节点，`overrideBootstrapCommand`不支持。相反，要指定自己的引导容器，应使用该`bottlerocket`字段作为配置文件的一部分。例如，

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

Windows 工作节点

从 1.14 版本开始，亚马逊 EKS 支持允许运行 Windows 容器的 Windows 节点。除了拥有 Windows 节点外，还需要集群中的 Linux 节点才能运行 CoreDNS，因为微软还不支持主机网络模式。因此，一个 Windows EKS 集群将是 Windows 节点和至少一个 Linux 节点的混合体。Linux 节点对群集的运行至关重要，因此，对于生产级群集，建议至少有两个 `t2.large` Linux 节点用于 HA。

> **Note**
>
> 您不再需要在 Linux 工作节点上安装 VPC 资源控制器即可在 2021 年 10 月 22 日之后创建的 EKS 集群中运行 Windows 工作负载。你可以通过 ConfigMap 设置在 EKS 控制平面上启用 Windows IP 地址管理（详情请参阅链接：eks/latest/userguide/windows-support.html）。创建 Windows 节点组时，eksctl 将自动修补 ConfigMap 以启用 Windows IP 地址管理。

创建支持 Windows 的新集群

配置文件语法允许通过单个命令创建支持 Windows 的功能齐全的集群：

```
# 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
```

要在不使用配置文件的情况下使用 Windows 非托管节点组创建新集群，请发出以下命令：

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

为现有的 Linux 群集添加 Windows 支持

要允许在具有 Linux 节点（AmazonLinux2AMI 系列）的现有集群上运行 Windows 工作负载，您需要添加一个 Windows 节点组。

新增了对 Windows 托管节点组的支持（--managed=true 或省略标志）。

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

为确保将工作负载安排在正确的操作系统上，它们必须nodeSelector针对必须运行的操作系统：

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

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

如果您使用的是早于的集群1.19，则需要将kubernetes.io/os和kubernetes.io/arch标签beta.kubernetes.io/arch分别替换为beta.kubernetes.io/os和。

进一步信息

- EK Windows Support

其他卷映射

作为额外的配置选项，在处理卷映射时，可以在创建节点组时配置额外的映射。

为此，请按additionalVolumes如下方式设置该字段：

```
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'
```

有关选择 VolumeNames 的更多详细信息，请参阅设备命名文档。要了解有关 EBS 卷、实例容量限制或块储存设备映射的更多信息，请访问此页面。

EKS 混合节点

简介

AWS EKS 推出了混合节点，这是一项新功能，使您能够使用与 AWS 云中相同的 AWS EKS 集群、功能和工具在客户托管的基础设施上运行本地和边缘应用程序。AWS EKS Hybird Nodes 为本地环境带来了 AWS 托管的 Kubernetes 体验，让客户可以简化和标准化您在本地、边缘和云环境中运行应用程序的方式。在 EKS 混合节点上阅读更多内容。

为了便于支持此功能，eksctl 引入了一个名为的新顶级字段。`remoteNetworkConfig`任何与混合节点相关的配置都应通过此字段进行设置，作为配置文件的一部分；没有对应的 CLI 标志。此外，启动时，任何远程网络配置都只能在集群创建期间进行设置，之后无法更新。这意味着，您将无法更新现有集群以使用混合节点。

配置文件的`remoteNetworkConfig`部分允许您在将远程节点加入您的 EKS 集群时设置两个核心区域：网络和证书。

Networking

EKS 混合节点可以灵活选择将本地网络连接到 AWS VPC 的首选方法。有多种记录在案的选项可供选择，包括 AWS Site-to-Site VPN 和 AWS Direct Connect，您可以选择最适合您的使用案例的方法。在

您可能选择的大多数方法中，您的 VPC 将连接到虚拟私有网关 (VGW) 或传输网关 (TGW)。如果您依靠 eksctl 为您创建 VPC，eksctl 还将在您的 VPC 范围内配置任何与网络相关的先决条件，以促进您的 EKS 控制平面和远程节点之间的通信，即

- 入口/出口 SG 规则
- 私有子网路由表中的路由
- 与给定 TGW 或 VGW 的 VPC 网关连接

配置文件示例：

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

如果您选择的连接方法不涉及使用 TGW 或 VGW，则不得依赖 eksctl 为您创建 VPC，而是要提供预先存在的 VPC。与此相关的是，如果您使用的是先前存在的 VPC，eksctl 不会对其进行任何修改，确保所有联网要求都已到位则由您负责。

> **Note**
>
> eksctl 不会在您的 AWS VPC 之外设置任何网络基础设施（即从远程网络 VGW/TGW 到远程网络的任何基础设施）

凭据

EKS 混合节点使用 AWS IAM Authenticator 和由 AWS SSM 或 AW S IAM R oles Anywhere 配置的临时 IAM 凭证对 EKS 集群进行身份验证。与自我管理的节点组类似，如果未另行提供，eksctl 将为您创建一个混合节点 IAM 角色，由远程节点担任。此外，在使用 IAM Roles Anywhere 作为证书提供商时，eksctl 将根据给定的证书颁发机构捆绑包 () 设置配置文件和信任锚点 ()，例如 `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
```

eksctl 创建的混合节点角色的 ARN 需要稍后在将远程节点加入集群、设置NodeConfig和创建激活（如果使用 SSM）的过程中。nodeadm要获取它，请使用：

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

同样，如果使用 IAM Roles Anywhere，则可以获取 eksctl 创建的信任锚点和 anywhere 配置文件的 ARN，通过分别替换为或来修改之前RemoteNodesRoleARN的RemoteNodesTrustAnchorARN命令。RemoteNodesAnywhereProfileARN

如果您已有预先存在的 IAM Roles Anywhere 配置，或者您正在使用 SSM，则可以通过为混合节点提供 IAM 角色。remoteNetworkConfig.iam.roleARN请记住，在这种情况下，eksctl 不会为你创建信任锚和任何地方的个人资料。例如

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

为了将角色映射到 Kubernetes 身份并授权远程节点加入 EKS 集群，eksctl 创建一个访问条目，将混合节点 IAM 角色作为委托人 ARN，类型为 ARN，即 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"
```

```
        ]
    }
]
```

附加组件支持

容器网络接口 (CNI)：AWS VPC CNI 不能用于混合节点。支持 Cilium 和 Calico 的核心功能以用于混合节点。您可以使用自己选择的工具（例如 Helm）来管理 CNI。有关更多信息，请参阅为混合节点配置 CNI。

> **Note**
>
> 如果您在集群中为自行管理或 EKS 管理的节点组安装 VPC CNI，则必须使用`v1.19.0-eksbuild.1`或更高版本，因为这包括更新插件的守护程序集以将其排除在混合节点上的安装之外。

更多参考文献

- EKS 混合节点 UserDocs
- 发布公告

EKS 托管节点组的 Support 节点修复配置

EKS Managed Nodegroups 支持节点修复，即监控托管节点的运行状况，并根据需要替换或重新启动不健康的工作节点。eksctl 现在提供了全面的配置选项，用于精细控制节点修复行为。

基本节点修复配置

使用 CLI 标志

要使用基本节点修复功能创建带有托管节点组的集群，请传递以下`--enable-node-repair`标志：

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

要在现有集群上创建带有节点修复功能的托管节点组，请执行以下操作：

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

使用配置文件

```
# 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
```

增强的节点修复配置

阈值配置

您可以使用基于百分比或计数的阈值来配置何时停止节点修复操作。注意：不能同时使用百分比和计数阈值。

阈值的 CLI 标志

```
# 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
```

阈值的配置文件

```
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
```

并行修复限制

控制可以同时或并行修复的最大节点数。这让您可以更精细地控制节点更换的速度。注意：您不能同时使用百分比和计数限制。

并行限制的 CLI 标志

```
# 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
```

并行限制的配置文件

```
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
```

自定义修复优先选项

为特定的修复操作指定精细覆盖。这些覆盖可控制节点被视为符合修复条件之前的修复操作和修复延迟时间。如果使用此选项，则必须为每个覆盖指定所有值。

```
managedNodeGroups:
- name: custom-repair-ng
  instanceType: g4dn.xlarge  # GPU instances
  nodeRepairConfig:
```

```
    enabled: true
    maxUnhealthyNodeThresholdPercentage: 25
    maxParallelNodesRepairedCount: 1
    nodeRepairConfigOverrides:
      # Handle GPU-related failures with immediate termination
      - nodeMonitoringCondition: "AcceleratedInstanceNotReady"
        nodeUnhealthyReason: "NvidiaXID13Error"
        minRepairWaitTimeMins: 10
        repairAction: "Terminate"
      # Handle network issues with restart after waiting
      - nodeMonitoringCondition: "NetworkNotReady"
        nodeUnhealthyReason: "InterfaceNotUp"
        minRepairWaitTimeMins: 20
        repairAction: "Restart"
```

完整的配置示例

有关包含所有配置选项的完整示例，请参阅 examples/44-node-re pair.yam l。

示例 1：带有百分比阈值的基本修复

```
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
```

示例 2：对关键工作负载进行保守修复

```
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"
```

示例 3：具有专门修复功能的 GPU 工作负载

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

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

managedNodeGroups:
- name: gpu-ng
  instanceType: g4dn.xlarge
  desiredCapacity: 4
  nodeRepairConfig:
    enabled: true
    maxUnhealthyNodeThresholdPercentage: 25
    maxParallelNodesRepairedCount: 1
    nodeRepairConfigOverrides:
      # GPU failures require immediate termination
      - nodeMonitoringCondition: "AcceleratedInstanceNotReady"
        nodeUnhealthyReason: "NvidiaXID13Error"
        minRepairWaitTimeMins: 5
        repairAction: "Terminate"
```

CLI 参考

节点修复标志

标记	说明	示例
`--enable-node-repair`	启用自动节点修复	`--enable-node-repair`
`--node-repair-max-unhealthy-percentage`	修复前不健康节点的最大百分比	`--node-repair-max-unhealthy-percentage=20`
`--node-repair-max-unhealthy-count`	修复前不健康节点的最大数量	`--node-repair-max-unhealthy-count=5`
`--node-repair-max-parallel-percentage`	要并行修复的节点的最大百分比	`--node-repair-max-parallel-percentage=15`
`--node-repair-max-parallel-count`	要并行修复的最大节点数	`--node-repair-max-parallel-count=2`

注意：由于其复杂性，仅通过 YAML 配置文件支持节点修复配置覆盖。

配置引用

nodeRepairConfig

字段	Type	说明	约束	示例
`enabled`	布尔值	启用/禁用节点修复	-	`true`
`maxUnhealthyNodeTh`	整数	不健康节点的百分比阈值，超过该阈值的节点 auto 修复操作将停止	不能与 `maxUnhealthyNodeTh`	`20`

字段	Type	说明	约束	示例
resholdPercentage			resholdCount	
maxUnhealthyNodeThresholdCount	整数	计算不健康节点的阈值，超过该阈值的节点自动修复操作将停止	不能与 maxUnhealthyNodeThresholdPercentage	5
maxParallelNodesRepairedPercentage	整数	可以同时或并行修复的不健康节点的最大百分比	不能与 maxParallelNodesRepairedCount	15
maxParallelNodesRepairedCount	整数	可以同时或并行修复的不健康节点的最大数量	不能与 maxParallelNodesRepairedPercentage	2
nodeRepairConfigOverrides	array	控制修复操作和延迟时间的特定修复操作的精细覆盖	必须为每次覆盖指定所有值	参见上面的例子

nodeRepairConfig覆盖

字段	Type	说明	有效值
nodeMonitoringCondition	字符串	节点监视代理报告了此覆盖所适用的不健康状况	"AcceleratedInstanceNotReady"，"NetworkNotReady"

字段	Type	说明	有效值
nodeUnhealthyReason	字符串	节点监视代理报告此覆盖适用的原因	"NvidiaXID13Error" ,"InterfaceNotUp"
minRepairWaitTimeMins	整数	尝试修复具有指定条件和原因的节点之前等待的最短时间（以分钟为单位）	任何正整数
repairAction	字符串	修复满足所有指定条件时要对节点采取的操作	"Terminate" , "Restart" , "NoAction"

进一步信息

- EKS 托管节点组节点健康

Networking

本章包括有关 Eksctl 如何为 EKS 集群创建虚拟私有云 (VPC) 网络的信息。

主题：

VPC 配置

集群专用 VPC

默认情况下，`eksctl create cluster`将为集群创建专用 VPC。这样做是为了避免由于各种原因干扰现有资源，包括安全，但也因为检测现有 VPC 中的所有设置具有挑战性。

- 使用的默认 VPC CIDR `eksctl` 是`192.168.0.0/16`。
  - 它分为 8 (/19) 个子网（3 个私有子网、3 个公共子网和 2 个预留子网）。
- 初始节点组是在公有子网中创建的。
- 除非指定，否则`--allow-ssh`将禁用 SSH 访问。
- 默认情况下，节点组允许来自控制平面安全组的入站流量通过端口 1025-65535。

> **Note**
>
> 在 `us-east-1` eksctl 中，默认情况下仅创建 2 个公有子网和 2 个私有子网。

更改 VPC 网段

如果您需要设置与其他 VPC 的对等互连，或者只是需要更大或更小的范围 IPs，则可以使用 `--vpc-cidr` flag 对其进行更改。有关选择允许在 AWS VPC 中使用的 CIDR 块的指南，请参阅 AWS 文档。

如果您正在创建 IPv6 集群，则可以在集群配置文件`VPC.IPv6Cidr`中进行配置。此设置仅在配置文件中，不在 CLI 标志中。

如果您拥 IPv6 有 IP 地址块，也可以自带地址 IPv6 池。有关如何导入自己的池的信息，请参阅将自己的 IP 地址 (BYOIP) 带到 Amazon EC2。然后使用集群配置文件`VPC.IPv6Cidr`中的来配置 Eksctl。

使用现有 VPC：与 kops 共享

您可以使用由 kops 管理的现有 Kubernetes 集群的 VPC。提供此功能是为了便于迁移 and/or 集群对等。

如果您之前使用 kops 创建过集群，例如使用类似于以下的命令：

```
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
```

您可以使用相同的 VPC 子网在同 AZs 一个集群中创建 EKS 集群（注意：至少需要 2 AZs/subnets 个）：

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

使用现有 VPC：其他自定义配置

eksctl为自定义 VPC 和子网拓扑提供了一些（但不完整）的灵活性。

您可以使用--vpc-private-subnets和--vpc-public-subnets标志提供私有 and/or 公有子网，从而使用现有 VPC。您有责任确保对所使用的子网进行正确分类，因为配置各不相同，因此没有简单的方法可以验证子网实际上是私有子网还是公有子网。

有了这些标志，eksctl create cluster将自动确定 VPC ID，但不会创建任何路由表或其他资源，例如 internet/NAT 网关。但是，它将为初始节点组和控制平面创建专用的安全组。

您必须确保在不同的子网中提供至少 2 个子网， AZs并且 EKS 会检查这种情况。如果您使用现有 VPC，则 EKS 或 Eksctl 不会强制执行或检查以下要求，EKS 会创建集群。集群的某些基本功能无需这些要求即可运行。（例如，标记并不是严格必需的，测试表明，可以在子网上不设置任何标签的情况下创建功能集群，但是不能保证它会一直有效，因此建议使用标记。）

标准要求：

- 所有给定的子网都必须位于同一 VPC 中，位于同一个区块内 IPs
- 根据需要提供足够数量的 IP 地址
- 足够数量的子网（最少 2 个），视需要而定
- 子网至少标有以下内容：
  - kubernetes.io/cluster/<name>标签设置为shared或 owned
  - kubernetes.io/role/internal-elb私有子网1的标签设置为
  - kubernetes.io/role/elb公共子网1的标签设置为
- 正确配置的互联网 and/or NAT 网关
- 路由表中有正确的条目并且网络运行正常
- 新：所有公有子网都应MapPublicIpOnLaunch启用该属性（即Auto-assign public IPv4 address在 AWS 控制台中）。托管节点组和 Fargate 不分配公有 IPv4 地址，必须在子网上设置该属性。

EKS 或 Kubernetes 可能还会施加其他要求，完全取决于你是否遵守任何要求 up-to-date。and/or recommendations, and implement those as needed/possible

应用的默认安全组设置eksctl可能足以与其他安全组中的资源共享访问权限，也可能不够。如果您想修改安全组的 ingress/egress 规则，则可能需要使用其他工具来自动进行更改，或者通过 EC2 控制台进行更改。

如有疑问，请不要使用自定义 VPC。`eksctl create cluster`不使用任何`--vpc-*`标志将始终为集群配置功能齐全的专用 VPC。

示例

使用具有 2 个私有子网和 2 个公有子网的自定义 VPC 创建集群：

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

或者使用以下等效的配置文件：

```
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
```

使用具有 3 个私有子网的自定义 VPC 创建集群，并让初始节点组使用这些子网：

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

```
  --node-private-networking
```

或者使用以下等效的配置文件：

```
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
```

使用自定义 VPC 4x 公有子网创建集群：

```
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
```

更多示例可以在 repo 的examples文件夹中找到：

- 使用现有 VPC
- 使用自定义 VPC CIDR

自定义共享节点安全组

eksctl将创建和管理共享节点安全组，该组允许非托管节点与集群控制平面和托管节点之间进行通信。

如果您希望改为提供自己的自定义安全组，则可以覆盖配置文件中的sharedNodeSecurityGroup字段：

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

默认情况下，在创建集群时，eksctl会向该安全组添加规则，以允许与 EKS 创建的默认集群安全组进行通信。EKS 控制平面和托管节点组均使用默认集群安全组。

如果您想自己管理安全组规则，则可以通过在配置文件false中设置manageSharedNodeSecurityGroupRules为来eksctl阻止创建规则：

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

NAT 网关

可以将集群的 NAT 网关配置为`Disable`、`Single`（默认）或`HighlyAvailable`。该`HighlyAvailable`选项将在该区域的每个可用区部署一个 NAT 网关，这样，如果一个可用区关闭，另一个可用区中的节点仍 AZs 能与 Internet 通信。

它可以通过 `--vpc-nat-mode` CLI 标志或在集群配置文件中指定，如下例所示：

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

请在此处查看完整的示例。

> **Note**
>
> 仅在创建集群期间支持指定 NAT 网关。在集群升级期间不会触及它。

子网设置

使用私有子网作为初始节点组

如果您希望将初始节点组与公共互联网隔离开来，则可以使用标志。`--node-private-networking`与`--ssh-access`标志一起使用时，只能从 VPC 内部访问 SSH 端口。

> **Note**
>
> 使用该`--node-private-networking`标志将导致传出流量使用其弹性 IP 通过 NAT 网关。另一方面，如果节点位于公有子网中，则传出流量将不会通过 NAT 网关，因此传出流量具有每个节点的 IP。

自定义子网拓扑

`eksctl`版本`0.32.0`引入了进一步的子网拓扑定制，可以：

- 在 VPC 配置中列出每个可用区的多个子网
- 在节点组配置中指定子网

在早期版本中，自定义子网必须由可用区提供，这意味着每个可用区只能列出一个子网。从0.32.0识别密钥可以是任意的。

```
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

如果使用 AZ 作为标识密钥，则可以省略该az值。

如果像上面那样使用任意字符串作为标识键，则可以：

- id必须设置（az且cidr可选）
- 或az必须设置（cidr可选）

如果用户在未指定 CIDR 和 ID 的情况下按可用区指定子网，则如果存在多个此类子网，则将从 VPC 中选择该可用区中的子网。

Note

必须提供完整的子网规范，即同时提供两者public以及在 VPC 规范中声明的private配置。

通过配置，可以将节点组限制为命名子网。在节点组配置上指定子网时，请使用 VPC 规范中给出的标识密钥而不是子网 ID。例如：

```
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**
>
> 节点组配置中availabilityZones只能提供subnets或中的一个。

在私有子网中放置节点组时，privateNetworking必须将节点组设置为true：

```
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
```

有关完整的配置示例，请参阅 eksctl 存储库中的 24-nodegroup-subnets.yaml。 GitHub

集群访问权限

管理对 Kubernetes API 服务器端点的访问权限

默认情况下，EKS 集群公开公开 Kubernetes API 服务器，但不会直接从 VPC 子网内部公开（public=true，private=false）。从 VPC 内部发往 API 服务器的流量必须先退出 VPC 网络（但不能退出 Amazon 的网络），然后重新进入才能到达 API 服务器。

使用集群配置文件创建集群时，可以将集群的 Kubernetes API 服务器端点访问权限配置为公有和私有访问。以下示例：

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

在配置 Kubernetes API 端点访问权限时，还有一些额外的注意事项：

1. EKS 不允许未启用私有或公有访问权限的集群。
2. EKS 确实允许创建仅允许启用私有访问的配置，但是 eksctl 在创建集群期间不支持该配置，因为它会阻止 eksctl 将工作节点加入集群。
3. 将集群更新为仅允许私有 Kubernetes API 终端节点访问权限意味着，默认情况下，Kubernetes 命令（例如`kubectl`）以及`eksctl delete cluster`和`eksctl utils write-kubeconfig`、以及可能的命令`eksctl utils update-kube-proxy`必须在集群 VPC 内运行。
   - 这需要对各种 AWS 资源进行一些更改。有关更多信息，请参阅集群 API 服务器终端节点。
   - 您可以提供将额外`vpc.extraCIDRs`的 CIDR 范围附加到的 CIDR 范围 ControlPlaneSecurityGroup，从而允许 VPC 之外的子网到达 kubernetes API 终端节点。同样，您也可以`vpc.extraIPv6CIDRs`提供附加的 IPv6 CIDR 范围。

以下是如何使用子命令配置 Kubernetes API 端点访问权限的示例：`utils`

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

要使用**`ClusterConfig`**文件更新设置，请使用：

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

请注意，如果您不传递标志，它将保留当前值。对建议的更改感到满意后，请添加approve标志以对正在运行的集群进行更改。

限制对 EKS Kubernetes 公共 API 端点的访问

默认创建 EKS 集群会公开公开 Kubernetes API 服务器。

此功能仅适用于公共端点。API 服务器终端节点访问配置选项不会更改，您仍然可以选择禁用公共终端节点，这样您的集群就无法通过 Internet 进行访问。（来源：conta https://github.com/aws/ iners-roadmap/issues/108 #issuecomment -552766489）

要在创建集群 CIDRs 时将公有 API 终端节点的访问权限限制为一组，请设置以下**publicAccessCIDRs**字段：

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

要更新对现有集群的限制，请使用：

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

要使用**ClusterConfig**文件更新限制，请设置新的 CIDRs 文件**vpc.publicAccessCIDRs**并运行：

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

> **Important**
>
> 如果设置publicAccessCIDRs和创建节点组，则privateAccess应将其设置为，true或者将节点添加到 IPs 列表中。publicAccessCIDRs

如果节点由于访问受限而无法访问集群 API 终端节点，则集群创建将失败，context deadline exceeded原因是节点无法访问公有终端节点且无法加入集群。

要在单个命令中更新集群的 API 服务器终端节点访问权限和公共访问权限 CIDRs ，请运行：

```
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
```

要使用配置文件更新设置，请执行以下操作：

```
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
```

更新控制平面子网和安全组

本文档介绍如何在初始创建后修改 EKS 集群控制平面的网络配置。这包括更新控制平面子网和安全组。

更新控制平面子网

使用 eksctl 创建集群时，会创建一组公有和私有子网并将其传递给 EKS API。EKS 在这些子网中创建 2 到 4 个跨账户弹性网络接口 (ENIs)，以实现 EKS 托管的 Kubernetes 控制平面与您的 VPC 之间的通信。

要更新 EKS 控制平面使用的子网，请运行：

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

要使用配置文件更新设置，请执行以下操作：

```
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
```

如果没有标--approve志，eksctl 只记录建议的更改。对建议的更改感到满意后，请重新运行带有该--approve标志的命令。

更新控制平面安全组

为了管理控制平面和工作节点之间的流量，EKS 支持传递应用于 EKS 配置的跨账户网络接口的其他安全组。要更新 EKS 控制平面的安全组，请运行：

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

要使用配置文件更新设置，请执行以下操作：

```
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
```

要更新集群的控制平面子网和安全组，请运行：

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

要使用配置文件更新这两个字段，请执行以下操作：

```
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
```

有关完整示例，请参阅 cluster-subnets-sgs.yaml。

如果没有标--approve志，eksctl 只记录建议的更改。对建议的更改感到满意后，请重新运行带有该--approve标志的命令。

IPv6 Support

定义 IP 家族

eksctl创建 vpc 时，您可以定义将要使用的 IP 版本。以下选项可供配置：

- IPv4
- IPv6

默认值为 IPv4。

要对其进行定义，请使用以下示例：

```
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**
>
> 此设置仅在配置文件中，不在 CLI 标志中。

如果使用 IPv6，则必须配置以下要求：

- OIDC 已启用
- 托管插件的定义如上所示
- 集群版本必须为 => 1.21
- vpc-cni 插件版本必须为 => 1.10.0
- 集群不支持自我管理的节点组 IPv6
- 无主集群不支持托管节点组 IPv6
- `vpc.nat`和`serviceIPv4CIDR`字段由 eksctl 为 ipv6 集群创建，不支持配置选项
- AutoAllocateIPv6 不支持与 IPv6
- 对于 IPv6 集群，vpc-cni 的 IAM 角色必须具有关联模式所需的 IAM 策略 IPv6

也可以使用 IPv6 IP 系列实现私有网络。请按照 EKS 私有集群下概述的说明进行操作。

IAM

本章包含有关使用 AWS IAM 的信息。

主题：

- the section called “管理 IAM 用户和角色”
  - 管理 IAM 用户和角色映射以控制对 EKS 集群的访问权限
  - 通过集群配置文件或 CLI 命令配置 IAM 身份映射
- the section called “服务账户的 IAM 角色”
  - 管理在 Amazon EKS 上运行且使用其他 AWS 服务的应用程序的精细权限
  - 使用 eksctl 创建和配置 IAM 角色和 Kubernetes 服务账户对
  - 为 EKS 集群启用 IAM OpenID Connect 提供商以为服务账户启用 IAM 角色
- the section called “IAM 权限边界”
  - 通过设置权限边界来控制授予给 IAM 实体（用户或角色）的最大权限
- the section called “EKS Pod 身份关联”
  - 使用推荐的 pod 身份关联为 EKS 插件配置 IAM 权限
  - 使 Kubernetes 应用程序能够获得连接集群外的 AWS 服务所需的 IAM 权限
  - 简化在多个 EKS 集群中自动化 IAM 角色和服务账户的流程
- the section called “IAM 策略”
  - 管理 EKS 节点组的 IAM 策略，包括支持各种附加策略，例如映像生成器、auto scaler、外部 DNS、证书管理器等。
  - 将自定义实例角色或内联策略附加到节点组以获得更多权限。
  - 将 ARN 特定 AWS 托管策略附加到节点组，确保包括亚马逊和 A EKSWorker NodePolicy mazoneks_cni_Policy 等必需的策略。
- the section called “最低限度 IAM 策略”
  - 管理 AWS EC2 资源，包括负载均衡器、自动扩展组和监控 CloudWatch
  - 创建和管理 AWS CloudFormation 堆栈
  - 管理 Amazon Elastic Kubernetes Service (EKS) 集群、节点组和 IAM 角色和策略等相关资源

最低限度 IAM 策略

本文档介绍了运行 eksctl 的主要用例所需的最低 IAM 策略。这些是用来运行集成测试的。

> **Note**
>
> 记得<account_id>用自己的替换。

> **Note**
>
> AWS 托管策略由 AWS 创建和管理。您无法更改 AWS 托管策略中定义的权限。

亚马逊 EC2FullAccess （ AWS 托管政策 ）

查看亚马逊EC2FullAccess 政策定义。

AWSCloudFormationFullAccess （ AWS 托管策略 ）

查看 AWSCloudFormationFullAccess 策略定义。

EksAllAccess

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "eks:*",
            "Resource": "*"
        },
        {
            "Action": [
                "ssm:GetParameter",
                "ssm:GetParameters"
            ],
            "Resource": [
                "arn:aws:ssm:*:123456789012:parameter/aws/*",
                "arn:aws:ssm:*::parameter/aws/*"
            ],
```

```
            "Effect": "Allow"
        },
        {
             "Action": [
               "kms:CreateGrant",
               "kms:DescribeKey"
             ],
             "Resource": "*",
             "Effect": "Allow"
        },
        {
             "Action": [
               "logs:PutRetentionPolicy"
             ],
             "Resource": "*",
             "Effect": "Allow"
        }
    ]
}
```

IamLimitedAccess

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "iam:CreateInstanceProfile",
                "iam:DeleteInstanceProfile",
                "iam:GetInstanceProfile",
                "iam:RemoveRoleFromInstanceProfile",
                "iam:GetRole",
                "iam:CreateRole",
                "iam:DeleteRole",
                "iam:AttachRolePolicy",
                "iam:PutRolePolicy",
                "iam:UpdateAssumeRolePolicy",
                "iam:AddRoleToInstanceProfile",
                "iam:ListInstanceProfilesForRole",
                "iam:PassRole",
                "iam:DetachRolePolicy",
                "iam:DeleteRolePolicy",
```

```
                "iam:GetRolePolicy",
                "iam:GetOpenIDConnectProvider",
                "iam:CreateOpenIDConnectProvider",
                "iam:DeleteOpenIDConnectProvider",
                "iam:TagOpenIDConnectProvider",
                "iam:ListAttachedRolePolicies",
                "iam:TagRole",
                "iam:UntagRole",
                "iam:GetPolicy",
                "iam:CreatePolicy",
                "iam:DeletePolicy",
                "iam:ListPolicyVersions"
            ],
            "Resource": [
                "arn:aws:iam::123456789012:instance-profile/eksctl-*",
                "arn:aws:iam::123456789012:role/eksctl-*",
                "arn:aws:iam::123456789012:policy/eksctl-*",
                "arn:aws:iam::123456789012:oidc-provider/*",
                "arn:aws:iam::123456789012:role/aws-service-role/eks-
nodegroup.amazonaws.com/AWSServiceRoleForAmazonEKSNodegroup",
                "arn:aws:iam::123456789012:role/eksctl-managed-*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "iam:GetRole",
                "iam:GetUser"
            ],
            "Resource": [
                "arn:aws:iam::123456789012:role/*",
                "arn:aws:iam::123456789012:user/*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "iam:CreateServiceLinkedRole"
            ],
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "iam:AWSServiceName": [
                        "eks.amazonaws.com",
```

```
                        "eks-nodegroup.amazonaws.com",
                        "eks-fargate.amazonaws.com"
                    ]
                }
            }
        }
    ]
}
```

IAM 权限边界

权限边界是一项高级 AWS IAM 功能，其中设置了基于身份的策略可以向 IAM 实体授予的最大权限；其中这些实体要么是用户，要么是角色。为实体设置权限边界后，该实体只能执行其基于身份的策略和权限边界所允许的操作。

您可以提供权限边界，以便在该边界内创建由 eksctl 创建的所有基于身份的实体。此示例演示了如何向 eksctl 创建的各种基于身份的实体提供权限边界：

```
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
>
> 不可能同时提供角色 ARN 和权限边界。

设置 VPC CNI 权限边界

请注意，当您创建启用了 OIDC 的集群时，出于安全考虑，eksctl 将自动为 VPC-CNI 创建一个iamserviceaccount。如果您想为其添加权限边界，则必须在配置文件iamserviceaccount中手动指定：

```
iam:
  serviceAccounts:
    - metadata:
        name: aws-node
        namespace: kube-system
      attachPolicyARNs:
      - "arn:aws:iam::<arn>:policy/AmazonEKS_CNI_Policy"
      permissionsBoundary: "arn:aws:iam::11111:policy/entity/boundary"
```

IAM 策略

您可以将实例角色附加到节点组。在节点上运行的工作负载将获得该节点的 IAM 权限。有关更多信息，请参阅适用于 A mazon EC2 的 IAM 角色。

本页列出了 eksctl 中可用的预定义 IAM 策略模板。这些模板简化了向您的 EKS 节点授予相应的 AWS 服务权限的过程，而无需手动创建自定义 IAM 策略。

支持的 IAM 插件策略

所有支持的附加策略示例：

```
nodeGroups:
  - name: ng-1
    instanceType: m5.xlarge
    desiredCapacity: 1
    iam:
      withAddonPolicies:
        imageBuilder: true
```

```
        autoScaler: true
        externalDNS: true
        certManager: true
        appMesh: true
        appMeshPreview: true
        ebs: true
        fsx: true
        efs: true
        awsLoadBalancerController: true
        xRay: true
        cloudWatch: true
```

Image Builder 政策

该`imageBuilder`策略允许完全访问 ECR（弹性容器注册表）。例如，这对于构建需要将图像推送到 ECR 的 CI 服务器很有用。

EBS 政策

该`ebs`策略启用了新的 EBS CSI（弹性块存储容器存储接口）驱动程序。

证书管理器政策

该`certManager`策略允许向 Route 53 添加记录以解决难题。 DNS01 更多信息可以在这里找到。

添加自定义实例角色

此示例创建了一个重复使用另一个集群中现有 IAM 实例角色的节点组：

```
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"
```

附加内联策略

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

通过 ARN 附加策略

```
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**
>
> 如果节点组包含，则attachPolicyARNs它还必须包含默
> 认节点策略AmazonEKSWorkerNodePolicy，例如本示
> 例AmazonEC2ContainerRegistryPullOnly中的AmazonEKS_CNI_Policy和。

管理 IAM 用户和角色

> **Note**
>
> AWS 建议the section called “EKS Pod 身份关联”从. 迁移到。aws-auth ConfigMap

EKS 集群使用 IAM 用户和角色来控制对集群的访问权限。这些规则是在配置映射中实现的

使用 CLI 命令 ConfigMap 进行编辑

被称为aws-auth。 eksctl提供了读取和编辑此配置映射的命令。

获取所有身份映射：

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

获取与 arn 匹配的所有身份映射：

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

创建身份映射：

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

删除身份映射：

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

> **Note**
>
> 上面的命令会删除单个映射 FIFO，除非给出--all这种情况，否则它会删除所有匹配项。如果找到更多与此角色匹配的映射，将发出警告。

创建账户映射：

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

删除账户映射：

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

ConfigMap 使用 ClusterConfig 文件进行编辑

身份映射也可以在以下位置指定： 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
```

服务账户的 IAM 角色

> **Tip**
>
> eksctl支持通过 EKS Pod 身份关联为运行应用程序的 EKS 配置精细权限

Amazon EKS 在这里支持服务账户角色 (IRSA)]，允许集群操作员将 AWS IAM 角色映射到 Kubernetes 服务账户。

这为在 EKS 上运行并使用其他 AWS 服务的应用程序提供了精细的权限管理。这些应用程序可能是使用 S3、任何其他数据服务（RDS、MQ、STS、DynamoDB）或 Kubernetes 组件（例如 AWS Load Balancer 控制器或 ExternalDNS）的应用程序。

您可以轻松地使用创建 IAM 角色和服务账户配对eksctl。

> **Note**
>
> 如果您使用了实例角色，并且正在考虑改用 IRSA，则不应将两者混为一谈。

工作原理

它通过 EKS 公开的 IAM OpenID Connect 提供商 (OIDC) 运行，并且必须参照 IAM OIDC 提供商（特定于给定的 EKS 集群）构建 IAM 角色，并引用它将绑定到的 Kubernetes 服务账户。创建 IAM 角色后，服务账号应包含该角色的 ARN 作为注释 () eks.amazonaws.com/role-arn。默认情况下，将创建或更新服务帐号以包含角色注释，可以使用标志将其禁用--role-only。

在 EKS 内部，有一个准入控制器，它根据 Pod 使用的服务账户上的注释将 AWS 会话证书分别注入角色的 pod 中。凭证将由 AWS_ROLE_ARN & AWS_WEB_IDENTITY_TOKEN_FILE 环境变量公开。如果使用的是最新版本的 AWS SDK（具体版本详见此处），则应用程序将使用这些证书。

资源的名称是 iamserviceaccount，它代表 IAM 角色和服务账户对。eksctl

来自 CLI 的用法

> **Note**
>
> 服务账户的 IAM 角色需要 Kubernetes 版本 1.13 或更高版本。

默认情况下，IAM OIDC 提供商未启用，您可以使用以下命令将其启用，也可以使用配置文件（见下文）：

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

将 IAM OIDC 提供商与集群关联后，要创建绑定到服务账户的 IAM 角色，请运行：

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

> **Note**
>
> 您可以`--attach-policy-arn`多次指定使用多个策略。

更具体地说，您可以通过运行以下命令创建对 S3 具有只读访问权限的服务帐户：

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

默认情况下，它将在`default`命名空间中创建，但你可以指定任何其他命名空间，例如：

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

> **Note**
>
> 如果命名空间尚不存在，则会创建它。

如果您已在集群中创建了服务账户（没有 IAM 角色），则需要使用`--override-existing-serviceaccounts`标志。

也可以通过指定--tags以下内容将自定义标记应用于 IAM 角色：

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

CloudFormation 将生成一个包含随机字符串的角色名称。如果您更喜欢预先确定的角色名称，则可以指定--role-name：

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

当服务帐号由其他工具（例如 helm）创建和管理时，请使用--role-only来防止冲突。然后，另一个工具负责维护角色 ARN 注释。请注意，--override-existing-serviceaccounts该角色对roleOnly/--role-only服务帐号没有影响，因此将始终创建该角色。

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

如果您想将现有角色与服务帐号一起使用，则可以提供标--attach-role-arn记，而不是提供策略。为确保该角色只能由指定的服务帐号担任，您应在此处设置关系策略文档]。

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

要更新服务账号的角色权限，你可以运行eksctl update iamserviceaccount。

> **Note**
>
> eksctl delete iamserviceaccount删除 Kubernetes，ServiceAccounts即使它们不是由创建的。eksctl

与配置文件一起使用

要iamserviceaccounts使用配置文件进行管理，您需要设置iam.withOIDC: true并列出您想要的账户iam.serviceAccount。

所有命令都支持*--config-file*，你可以像管理节点组一样管理 iamserviceAc counts。eksctl create iamserviceaccount命令支持--include和--exclude标记（有关其工作原理的更多详

细信息，请参阅本节）。而且该eksctl delete iamserviceaccount命令也支持，因此您可以--only-missing像执行节点组一样执行删除操作。

> **Note**
>
> IAM 服务账户的作用域位于一个命名空间内，也就是说，两个同名的服务账户可能存在于不同的命名空间中。因此，要将服务帐号唯一定义为--exclude标志的一部分--include，您需要按namespace/name格式传递名称字符串。例如，

```
eksctl create iamserviceaccount --config-file=<path> --include backend-apps/s3-reader
```

如果将 IRSA 与众所周知的用例（如cluster-autoscaler和）一起使用cert-manager，则包含启用wellKnownPolicies选项，作为策略列表的简写。

支持的众所周知的策略和其他serviceAccounts属性记录在配置架构中。

您可以将以下配置示例与以下内容一起使用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
```

如果您创建的集群没有设置这些字段，则可以使用以下命令启用所需的一切：

```
eksctl utils associate-iam-oidc-provider --config-file=<path>
eksctl create iamserviceaccount --config-file=<path>
```

进一步信息

- 为服务账户引入精细的 IAM 角色
- EKS 用户指南-服务账户的 IAM 角色
- 将 IAM 用户和角色映射到 Kubernetes RBAC 角色

EKS Pod 身份关联

AWS EKS 引入了一种名为 Pod Identity Association 的新增强机制，供集群管理员配置 Kubernetes 应用程序，使其获得在集群外连接 AWS 服务所需的 IAM 权限。Pod Identity Association 利用 IRSA，但是，它可以直接通过 EKS API 对其进行配置，从而完全无需使用 IAM API。

因此，IAM 角色不再需要引用 OIDC 提供商，因此也不会再绑定到单个集群。这意味着，IAM 角色现在可以在多个 EKS 集群中使用，而无需在每次创建新集群时更新角色信任策略。这反过来又消除了角色重复的需求，并简化了完全自动化 IRSA 的流程。

先决条件

在幕后，pod 身份关联的实现是在工作节点上以守护程序集的形式运行代理。为了在集群上运行先决条件代理，EKS 提供了一个名为 EKS Pod Identity Agent 的新插件。因此，创建 pod 身份关联（一般而言，和 wit `eksctl` h）需要在集`eks-pod-identity-agent`群上预先安装插件。可以使用与任何其他受支持的插件相同`eksctl`的方式创建此插件。

```
eksctl create addon --cluster my-cluster --name eks-pod-identity-agent
```

此外，如果在创建 pod 身份关联时使用预先存在的 IAM 角色，则必须将该角色配置为信任新引入的 EKS 服务委托人 (`pods.eks.amazonaws.com`)。可以在下面找到 IAM 信任策略的示例：

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

相反，如果您没有向 create 命令提供现有角色的 ARN，则`eksctl`将在幕后创建一个角色并配置上述信任策略。

创建 Pod 身份关联

为了操纵 pod 身份关联，eksctl在下面添加了一个新字段iam.podIdentityAssociations，例如

```
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
```

有关完整示例，请参阅 pod-identity-associations.yaml。

> **Note**
>
> 除了permissionPolicy用作内联策略文档外，所有其他字段都有对应的 CLI 标志。

可以通过以下方式创建 pod 身份关联。在创建集群期间，通过在配置文件中指定所需的 pod 身份关联并运行：

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

创建集群后，使用配置文件，例如

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

或者使用 CLI 标志，例如

```
eksctl create podidentityassociation \
    --cluster my-cluster \
```

```
    --namespace default \
    --service-account-name s3-reader \
    --permission-policy-arns="arn:aws:iam::111122223333:policy/permission-policy-1,
 arn:aws:iam::111122223333:policy/permission-policy-2" \
    --well-known-policies="autoScaler,externalDNS" \
    --permissions-boundary-arn arn:aws:iam::111122223333:policy/permissions-boundary
```

> **Note**
>
> 一次只能将一个 IAM 角色与一个服务账号关联。因此，尝试为同一个服务帐号创建第二个 pod 身份关联将导致错误。

获取 Pod 身份关联

要检索特定集群的所有 pod 身份关联，请运行以下命令之一：

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

或

```
eksctl get podidentityassociation --cluster my-cluster
```

此外，要仅检索给定命名空间内的 pod 身份关联，请使用`--namespace`标志，例如

```
eksctl get podidentityassociation --cluster my-cluster --namespace default
```

最后，要检索与某个 K8s 服务帐号对应的单个关联，还`--service-account-name`需要在上面的命令中加上，即

```
eksctl get podidentityassociation --cluster my-cluster --namespace default --service-
account-name s3-reader
```

更新 Pod 身份关联

要更新一个或多个 pod 身份关联的 IAM 角色，请将新的`roleARN(s)`角色传递给配置文件，例如

```
iam:
  podIdentityAssociations:
    - namespace: default
```

```
      serviceAccountName: s3-reader
      roleARN: new-role-arn-1
    - namespace: dev
      serviceAccountName: app-cache-access
      roleARN: new-role-arn-2
```

然后运行：

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

或者（更新单个关联）`--role-arn`通过 CLI 标志传递新的：

```
eksctl update podidentityassociation --cluster my-cluster --namespace default --
service-account-name s3-reader --role-arn new-role-arn
```

删除 Pod 身份关联

要删除一个或多个 pod 身份关联，`serviceAccountName(s)`请将`namespace(s)`和传递给配置文件，例如

```
iam:
  podIdentityAssociations:
    - namespace: default
      serviceAccountName: s3-reader
    - namespace: dev
      serviceAccountName: app-cache-access
```

然后运行：

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

或者（删除单个关联）`--service-account-name`通过 CLI 传递`--namespace`和标志：

```
eksctl delete podidentityassociation --cluster my-cluster --namespace default --
service-account-name s3-reader
```

EKS 插件支持 pod 身份关联

EKS 插件还支持通过 EKS Pod 身份关联接收 IAM 权限。配置文件公开了三个允许配置这些字段的字段：`addon.podIdentityAssociations`、`addonsConfig.autoApplyPodIdentityAssociations`和

可以使用显式配置所需的 pod 身份关联addon.podIdentityAssociations，也可以使用或eksctl自动解析（并应用）推荐的 pod 身份配置addon.useDefaultPodIdentityAssociations。addonsConfig.autoApplyPodIdentityAssoc

> **Note**
>
> 并非所有 EKS 插件在启动时都支持 pod 身份关联。在这种情况下，应继续使用 I RSA 设置提供所需的 I AM 权限。

使用 IAM 权限创建插件

在创建需要 IAM 权限的插件时，eksctl将首先检查 pod 身份关联或 IRSA 设置是否被明确配置为配置文件的一部分，如果是，则使用其中一个来配置插件的权限。例如

```
addons:
- name: vpc-cni
  podIdentityAssociations:
  - serviceAccountName: aws-node
    permissionPolicyARNs: ["arn:aws:iam::aws:policy/AmazonEKS_CNI_Policy"]
```

然后跑

```
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**
>
> 不允许同时设置 pod 身份和 IRSA，这将导致验证错误。

对于支持 Pod 身份的 EKS 插件，eksctl提供了在创建插件时自动配置任何推荐的 IAM 权限的选项。这可以通过简单地在配置文件addonsConfig.autoApplyPodIdentityAssociations: true中进行设置来实现。例如

```
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
```

然后跑

```
eksctl create addon -f config.yaml
2024-05-13 15:38:58 [#] "addonsConfig.autoApplyPodIdentityAssociations" is set to true;
 will lookup recommended pod identity configuration for "vpc-cni" addon
```

同样，也可以通过 CLI 标志来完成同样的事情，例如

```
eksctl create addon --cluster my-cluster --name vpc-cni --auto-apply-pod-identity-
associations
```

要将现有插件迁移到使用带有推荐的 IAM 策略的 pod 身份，请使用

```
addons:
- name: vpc-cni
  useDefaultPodIdentityAssociations: true
```

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

使用 IAM 权限更新插件

更新插件时，指定addon.PodIdentityAssociations将代表更新操作完成后插件应具有的状态的单一真实来源。在幕后，为了达到所需的状态，会执行不同类型的操作，即

- 创建配置文件中存在但集群上缺少的 pod 标识
- 删除从配置文件中移除的现有 pod 标识以及任何关联的 IAM 资源
- 更新配置文件中也存在且其 IAM 权限集已更改的现有 pod 身份

> **Note**
>
> EKS Addons 拥有的 pod 身份关联的生命周期由 EKS Addons API 直接处理。

对于与 Amazon EKS 插件一起使用的关联，您不能使用`eksctl delete podidentityassociations`（更新 IAM 权限）或（移除关联）。`eksctl update podidentityassociation`取而代之的是，`eksctl update addon`或`eksctl delete addon`应使用。

让我们来看上面的例子，首先分析插件的初始 pod 身份配置：

```
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"
    }
]
```

现在使用以下配置：

```
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
```

然后跑

```
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
```

现在检查 pod 标识配置是否已正确更新

```
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"
    }
]
```

要从插件中移除所有 pod 身份关联，addon.PodIdentityAssociations必须明确设置为[]，例如

```
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: []
```

然后跑

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

删除具有 IAM 权限的插件

删除插件还会移除与该插件关联的所有 Pod 身份。对于所有插件，删除集群的效果都是一样的。由eksctl创建的任何 Pod 身份的 IAM 角色也将被删除。

将现有的 iamservice 账户和插件迁移到 pod 身份关联

有一个 eksctl utils 命令用于将服务账户的现有 IAM 角色迁移到 pod 身份关联，即

```
eksctl utils migrate-to-pod-identity --cluster my-cluster --approve
```

在幕后，该命令将应用以下步骤：

- 如果该eks-pod-identity-agent插件尚未在集群上处于活动状态，请安装该插件
- 识别与 iamservice 账户关联的所有 IAM 角色
- 识别与支持 pod 身份关联的 EKS 插件关联的所有 IAM 角色
- 更新所有已识别角色的 IAM 信任策略，添加一个指向新的 EKS 服务主体的可信实体（并可选择删除现有的 OIDC 提供商信任关系）
- 为与 iamserviceAccounts 关联的筛选角色创建 pod 身份关联
- 使用 pod 身份更新 EKS 插件（EKS API 将在幕后创建 pod 身份）

运行不带--approve标志的命令只会输出一个计划，该计划由一组反映上述步骤的任务组成，例如

```
[#]  (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
```

现有的 OIDC 提供商信任关系始终会从与 EKS 插件关联的 IAM 角色中删除。此外，要从与 iamserviceAccounts 关联的 IAM 角色中删除现有的 OIDC 提供商信任关系，请运行带标志的命令，例如 `--remove-oidc-provider-trust-relationship`

```
eksctl utils migrate-to-pod-identity --cluster my-cluster --approve --remove-oidc-
provider-trust-relationship
```

跨账户 Pod 身份支持

eksctl 支持 EK S Pod Identity 跨账户访问。此功能允许在您的 EKS 集群中运行的 Pod 访问其他 AWS 账户中的 AWS 资源。

用法

要创建具有跨账户访问权限的 pod 身份关联，请先设置 IAM 角色和策略，允许从源 AWS 账户（使用集群）访问目标 AWS 账户（使用集群可以访问的资源）。有关示例，请参阅 “Amazon EKS Pod Identity 简化了跨账户访问”。

在每个账户中配置了 IAM 角色后，使用 eksctl 创建 pod 身份关联：

```
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
```

更多参考文献

适用于 EKS 的官方 AWS 用户文档插件支持 pod 身份

关于 Pod 身份关联的 AWS 官方博客文章

Pod 身份关联的官方 AWS 用户文档

部署选项

本章介绍如何使用 eksctl 管理部署到备用环境的 EKS 集群。

有关 EKS 部署选项的最准确信息，请参阅 EK S 用户指南中的跨云和本地环境部署 Amazon EKS 集群。

主题：

- the section called “EKS 任何地方”
  - 在 Amazon EKS Anywhere 集群中使用 eksctl。
  - Amazon EKS Anywhere 是由 AWS 构建的容器管理软件，它可以更轻松地在本地和边缘运行和管理 Kubernetes。
- the section called “AWS Outposts Support”
  - 在 AWS Outposts 上将 eksctl 与 EKS 集群一起使用。
  - AWS Outposts 是一系列完全托管的解决方案，可向几乎任何本地或边缘站点提供 AWS 基础设施和服务，以实现真正一致的混合体验。
  - eksctl 中的 AWS Outposts 支持允许你创建本地集群，整个 Kubernetes 集群，包括 EKS 控制平面和工作节点，在 AWS Outposts 上本地运行。
- the section called “EKS 混合节点”
  - 使用与 AWS 云中相同的 AWS EKS 集群、功能和工具，在客户托管的基础设施上运行本地和边缘应用程序。

EKS 任何地方

`eksctl`提供对使用 sub 命令`eksctl anywhere`调`EKS Anywhere`用的 AWS 功能的访问权限。这需要存在`eksctl-anywhere`二进制文件`PATH`。请按照此处概述的说明安装 eksctl-Anywhere 进行安装。

完成后，通过运行以下命令在任何地方执行命令：

```
eksctl anywhere version
v0.5.0
```

有关 EKS Anywhere 的更多信息，请访问 EKS Anywhere 网站。

AWS Outposts Support

> **Warning**
>
> Outposts 不支持 EKS 托管节点组。

将现有集群扩展到 AWS Outposts

您可以将在 AWS 区域中运行的现有 EKS 集群扩展到 AWS Outposts，方法是设置`nodeGroup.outpostARN`新的节点组以在 Outposts 上创建节点组，如下所示：

```
# 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
```

在此设置中，EKS 控制平面在 AWS 区域中运行，而设置了`outpostARN`设置的节点组在指定的 Outpost 上运行。首次在 Outposts 上创建节点组时，eksctl 会通过在指定的 Outpost 上创建子网来扩展 VPC。这些子网用于创建已设置的节点组。`outpostARN`

已有 VPC 的客户需要在 Outposts 上创建子网并将其传入，如下所`nodeGroup.subnets`示：

```
# 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"
```

在 AWS Outposts 上创建本地集群

> **Note**
>
> 本地集群仅支持 Outpost 机架。

> **Note**
>
> 当控制平面位于 Outposts 上时，节点组仅支持 Amazon Linux 2。Outposts 上的节点组仅支持 EBS gp2 卷类型。

eksctl 中的 AWS Outpost s 支持允许你创建本地集群，整个 Kubernetes 集群，包括 EKS 控制平面和工作节点，在 AWS Outposts 上本地运行。客户可以在 AWS Outposts 上创建同时运行 EKS 控制平面

和工作节点的本地集群，也可以通过在 Outposts 上创建工作节点将在 AWS 区域中运行的现有 EKS 集群扩展到 AWS Outposts。

要在 AWS Outposts 上创建 EKS 控制平面和节点组，请`outpost.controlPlaneOutpostARN`设置为 Outpost ARN，如下所示：

```
# 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
```

这指示 eksctl 在指定的 Outpost 上创建 EKS 控制平面和子网。由于 Outposts 机架存在于单个可用区中，因此 eksctl 只会创建一个公有子网和私有子网。eksctl 不会将创建的 VPC 与本地网关关联，因此，eksctl 将缺乏与 API 服务器的连接并且无法创建节点组。因此，如果在创建集群期间`ClusterConfig`包含任何节点组，则必须使用以下命令运行`--without-nodegroup`，如下所示：

```
eksctl create cluster -f outpost.yaml --without-nodegroup
```

创建集群后，客户有责任将 eksctl 创建的 VPC 与本地网关关联，以实现与 API 服务器的连接。完成此步骤后，可以使用创建节点组。`eksctl create nodegroup`

您可以选择为中的控制平面节点`outpost.controlPlaneInstanceType`或中的节点组指定实例类型，但实例类型必须存在于 Outpost 上`nodeGroup.instanceType`，否则 eksctl 将返回错误。默认情况下，eksctl 会尝试在 Outpost 上为控制平面节点和节点组选择最小的可用实例类型。

当控制平面在 Outposts 上时，将在该 Outpost 上创建节点组。您可以选择为nodeGroup.outpostARN中的节点组指定 Outpost ARN，但它必须与控制平面的 Outpost ARN 相匹配。

```
# 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
```

现有 VPC

拥有现有 VPC 的客户可以通过在中指定子网配置在 AWS Outposts 上创建本地集群`vpc.subnets`，如下所示：

```
# 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
```

子网必须存在于中指定的前哨基地，`outpost.controlPlaneOutpostARN`否则 eksctl 将返回错误。如果您有权访问子网的本地网关或连接到 VPC 资源，则还可以在创建集群时指定节点组。

本地集群不支持这些功能

- 插件
- 服务账户的 IAM 角色
- IPv6

- 身份提供商
- Fargate
- KMS 加密
- 本地扩展区
- Karpenter
- 实例选择器
- 无法指定可用区，因为它默认为 Outpost 可用区。
- 不支持 `vpc.publicAccessCIDRs` 和 `vpc.autoAllocateIPv6`。
- 不支持对 API 服务器的公共终端节点访问，因为只能使用私有终端节点访问权限创建本地集群。

进一步信息

- AWS Outposts 上的 Amazon EKS
- AWS Outposts 上的 Amazon EKS 本地集群
- 创建本地集群
- 在前哨基地启动自我管理的 Amazon Linux 节点

安全性

eksctl提供了一些可以提高 EKS 集群安全性的选项。

withOIDC

启用withOIDC此选项可自动为 amazon CNI 插件创建 IRSA 并限制向集群中节点授予的权限，而不是仅向 CNI 服务账户授予必要的权限。

本 AWS 文档中描述了背景。

disablePodIMDS

对于托管和非托管节点组，disablePodIMDS选项可用可防止在此节点组中运行的所有非主机网络 Pod 发出 IMDS 请求。

> **Note**
>
> 这不能与一起使用withAddonPolicies。

EKS 集群的 KMS 信封加密

> **Note**
>
> 对于运行 Kubernetes 1.28 或更高版本的 EKS 集群，Amazon Elastic Kubernetes Service（Amazon EKS）为所有 Kubernetes API 数据提供了默认信封加密。有关更多信息，请参阅 EKS 用户指南中的所有 Kubernetes API 数据的默认信封加密。

EKS 支持使用 AWS KMS 密钥为存储在 EKS 中的 Kubernetes 密钥提供信封加密。信封加密为存储在 Kubernetes 集群中的应用程序密钥或用户数据添加了一个由客户管理的额外加密层。

以前，Amazon EKS 仅支持在创建集群期间使用 KMS 密钥启用信封加密。现在，您可以随时为 Amazon EKS 集群启用信封加密。

在 AWS 容器博客上阅读有关使用 EKS 加密提供商支持的更多信息。 defense-in-depth

创建启用 KMS 加密的集群

```
# 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
```

在现有集群上启用 KMS 加密

要在尚未启用 KMS 加密的集群上启用 KMS 加密，请运行

```
eksctl utils enable-secrets-encryption -f kms-cluster.yaml
```

或者没有配置文件：

```
eksctl utils enable-secrets-encryption --cluster=kms-cluster --key-arn=arn:aws:kms:us-
west-2:<account>:key/<key> --region=<region>
```

除了在 EKS 集群上启用 KMS 加密外，eksctl 还使用新的 KMS 密钥对所有现有的 Kubernetes 密钥进行重新加密，方法是使用注释对其进行更新。`eksctl.io/kms-encryption-timestamp`可以通过传递来禁用此行为`--encrypt-existing-secrets=false`，例如：

```
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>
```

如果集群已经启用了 KMS 加密，eksctl 将继续重新加密所有现有密钥。

> **Note**
>
> 启用 KMS 加密后，无法将其禁用或更新为使用其他 KMS 密钥。

问题排查

本主题包括有关如何使用 Eksctl 解决常见错误的说明。

堆栈创建失败

您可以使用该`--cfn-disable-rollback`标志来阻止 Cloudformation 回滚失败的堆栈，从而简化调试。

子网 ID “subnet-11111111” 与 “subnet-22222222” 不一样

给定一个配置文件，指定 VPC 的子网，如下所示：

```
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: []
```

错误`subnet ID "subnet-11111111" is not the same as "subnet-22222222"`表示指定的子网未放置在正确的可用区中。在 AWS 控制台中查看每个可用区的正确子网 ID。

在此示例中，VPC 的正确配置为：

```
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}
```

删除问题

如果您的删除不起作用，或者您忘记`--wait`在删除时添加内容，则可能需要使用亚马逊的其他工具来删除 cloudformation 堆栈。这可以通过 gui 或 aws cli 来完成。

kubectl 日志和 kubectl 运行失败并显示授权错误

如果您的节点部署在私有子网中，`kubectl logs`或者出现以下错误而`kubectl run`失败：

```
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)
```

然后你可能需要设置enableDnsHostnames。更多细节可以在本期中找到。

公告

本主题涵盖了过去发布的 Eksctl 新功能。

托管节点组默认

从 eksctl v0.58.0 开始，当没有为和指定文件时，eksctl 会默认创建托管节点组。`ClusterConfig` `eksctl create cluster` `eksctl create nodegroup`要创建自我管理的节点组，请传递。`--managed=false`如果正在使用托管节点组中不支持的功能（例如 Windows 节点组），则可能会破坏不使用配置文件的脚本。要修复此问题，请使用创建自管理节点组的`nodeGroups`字段在`ClusterConfig`文件中传递`--managed=false`或指定您的节点组配置。

自定义节点组引导程序覆盖 AMIs

这一变化是在一期《Breaking：很 overrideBootstrapCommand 快...》中宣布的。现在，它已经在这个 PR 中实现了。请仔细阅读随附的问题，了解为什么我们决定不再支持 AMIs 没有引导脚本或部分引导脚本的自定义。

我们仍然提供帮手！希望迁移不会那么痛苦。 `eksctl`仍然提供了一个脚本，该脚本在获得源代码后将导出几个有用的环境属性和设置。此脚本位于此处。

以下环境属性可供您使用：

```
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
```

重写 so `eksctl` 不会失败时需要使用的最低限度是标签！ `eksctl`依赖于节点上的一组特定的标签，因此它可以找到它们。定义覆盖时，请提供以下最低限度的覆盖命令：

```
    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}"
```

对于没有出站 Internet 访问权限的节点组，您需要按如下方式向引导脚本提供`--apiserver-endpoint`和`--b64-cluster-ca`：

```
    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}
```

注意 “--nod e-labels” 设置。如果未定义此值，则该节点将加入集群，但在等待节点加入集群时，最终`eksctl`将在最后一步超时`Ready`。它正在进行 Kubernetes 查找带有标签的节点。`alpha.eksctl.io/nodegroup-name=<cluster-name>`这仅适用于非托管节点组。对于托管，它使用的是不同的标签。

如果可以切换到托管节点组以避免这种开销，那么现在是时候这样做了。使所有重写变得容易得多。

本文属于机器翻译版本。若本译文内容与英语原文存在差异，则一律以英文原文为准。