Amazon EKS

Eksctl 사용 설명서

Eksctl 사용 설명서: Amazon EKS

Table of Contents

Eksctl이란 무엇입니까?

eksctl은 Amazon Elastic Kubernetes Service(Amazon EKS) 클러스터 생성, 관리 및 운영 프로세스를 자동화하고 간소화하는 명령줄 유틸리티 도구입니다. Go로 작성된 eksctl은 YAML 구성 및 CLI 명령을 통해 선언적 구문을 제공하여 다른 AWS 서비스에서 여러 수동 단계가 필요한 복잡한 EKS 클러스터 작업을 처리합니다.

eksctl은 EKS 클러스터를 대규모로 지속적으로 배포하고 관리해야 하는 DevOps 엔지니어, 플랫폼 팀 및 Kubernetes 관리자에게 특히 유용합니다. 이는 기존 CI/CD 파이프라인 및 자동화 워크플로에 통합할 수 있으므로 자체 관리형 Kubernetes에서 EKS로 전환하거나 코드형 인프라(IaC)를 구현하는 조직에 특히 유용합니다. 이 도구는 VPC 구성, IAM 역할 생성, 보안 그룹 관리 등 EKS 클러스터 설정에 필요한 AWS 서비스 간의 많은 복잡한 상호 작용을 추상화합니다.

eksctl의 주요 기능에는 단일 명령으로 완전한 기능을 갖춘 EKS 클러스터를 생성하는 기능, 사용자 지정 네트워킹 구성 지원, 자동화된 노드 그룹 관리 및 GitOps 워크플로 통합이 포함됩니다. 이 도구는 선언적 접근 방식을 통해 클러스터 업그레이드를 관리하고, 노드 그룹을 확장하고, 추가 기능 관리를 처리합니다. 또한 eksctl은 네이티브 AWS SDK 통합을 통해 다른 AWS 도구 및 서비스와의 호환성을 유지하면서 Fargate 프로파일 구성, 관리형 노드 그룹 사용자 지정 및 스팟 인스턴스 통합과 같은 고급 기능을 제공합니다.

특성

현재 구현된 기능은 다음과 같습니다.

- 클러스터 생성, 가져오기, 나열 및 삭제
- 노드 그룹 생성, 드레이닝 및 삭제
- 노드 그룹 조정
- 클러스터 업데이트
- 사용자 지정 AMIs 사용
- VPC 네트워킹 구성
- API 엔드포인트에 대한 액세스 구성
- GPU 노드 그룹 지원
- 스팟 인스턴스 및 혼합 인스턴스
- IAM 관리 및 추가 기능 정책
- 클러스터 Cloudformation 스택 나열

- 코어 설치
- 클러스터에 대한 kubeconfig 파일 쓰기

Eksctl FAQ

일반

eksctl를 사용하여에서 생성하지 않은 클러스터를 관리할 수 있습니까**eksctl**?

예! 버전에서에 의해 생성되었는지 여부에 관계없이 모든 클러스터에 `eksctl` 대해를 실행할 `0.40.0` 수 `eksctl` 있습니다. 자세한 내용은 the section called “eksctl에서 생성하지 않은 클러스터” 단원을 참조하십시오.

노드 그룹

노드 그룹의 인스턴스 유형을 변경하려면 어떻게 해야 합니까?

의 관점에서 `eksctl`노드 그룹은 변경할 수 없습니다. 즉,가 생성되면 노드 그룹을 확장하거나 축소하는 것만 `eksctl` 가능합니다.

인스턴스 유형을 변경하려면 원하는 인스턴스 유형으로 새 노드 그룹을 생성한 다음 워크로드가 새 노드 그룹으로 이동하도록 드레이닝합니다. 이 단계가 완료되면 이전 노드 그룹을 삭제할 수 있습니다.

노드 그룹에 대해 생성된 사용자 데이터를 보려면 어떻게 해야 합니까?

먼저 노드 그룹을 관리하는 Cloudformation 스택의 이름이 필요합니다.

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

와 비슷한 이름이 표시됩니다`eksctl-CLUSTER_NAME-nodegroup-NODEGROUP_NAME`.

다음을 실행하여 사용자 데이터를 가져올 수 있습니다. base64에서 디코딩하고 압축 해제된 데이터를 압축 해제하는 마지막 줄을 기록해 둡니다.

```
NG_STACK=eksctl-scrumptious-monster-1595247364-nodegroup-ng-29b8862f # your stack here
LAUNCH_TEMPLATE_ID=$(aws cloudformation describe-stack-resources --stack-name $NG_STACK
 \
| jq -r '.StackResources | map(select(.LogicalResourceId == "NodeGroupLaunchTemplate")
 \
| .PhysicalResourceId)[0]')
aws ec2 describe-launch-template-versions --launch-template-id $LAUNCH_TEMPLATE_ID \
```

```
| jq -r '.LaunchTemplateVersions[0].LaunchTemplateData.UserData' \
| base64 -d | gunzip
```

Ingress

에서 수신을 설정하려면 어떻게 해야 합니까**eksctl**?

AWS Load Balancer Controller를 사용하는 것이 좋습니다. 클러스터에 컨트롤러를 배포하는 방법과 이전 ALB 수신 컨트롤러에서 마이그레이션하는 방법에 대한 설명서는 여기에서 확인할 수 있습니다.

Nginx 수신 컨트롤러의 경우 설정은 다른 Kubernetes 클러스터의 설정과 동일합니다.

kubectl

HTTPS 프록시를 사용하는데 클러스터 인증서 검증이 실패합니다. 시스템 CAs를 사용하려면 어떻게 해야 하나요?

시스템 인증 기관`KUBECONFIG_USE_SYSTEM_CA`을 `kubeconfig` 준수하도록 환경 변수를 설정합니다.

드라이 런

모의 실행 기능을 사용하면 노드 그룹 생성을 진행하기 전에 인스턴스 선택기와 일치하는 인스턴스를 검사하고 변경할 수 있습니다.

`eksctl create cluster`가 인스턴스 선택기 옵션 및와 함께 호출되면 `--dry-run`eksctl은 CLI 옵션과 인스턴스 선택기 리소스 기준과 일치하는 인스턴스로 설정된 인스턴스 유형을 나타내는 노드 그룹이 포함된 ClusterConfig 파일을 출력합니다.

```
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
```

그러면 생성된 ClusterConfig를에 전달할 수 있습니다`eksctl create cluster`.

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

ClusterConfig 파일이와 함께 전달되면 `--dry-run`eksctl은 파일에 설정된 값이 포함된 ClusterConfig 파일을 출력합니다.

eksctl의 일회성 옵션

`ClusterConfig` 파일에 표시할 수 없는 특정 일회성 옵션이 있습니다. 예: `--install-vpc-controllers`.

다음을 수행해야 합니다.

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

그 다음에 다음을 수행합니다.

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

는 없이 첫 번째 명령을 실행하는 것과 같습니다`--dry-run`.

따라서 eksctl은가 전달될 때 구성 파일에 표시할 수 없는 `--dry-run` 전달 옵션을 허용하지 않습니다.

> **Important**
>
> AWS 프로파일을 전달해야 하는 경우 `--profile` CLI 옵션을 전달하는 대신 `AWS_PROFILE` 환경 변수를 설정합니다.

자습서

이 주제에서는 eksctl을 설치 및 구성한 다음 이를 사용하여 Amazon EKS 클러스터를 생성하는 방법을 안내합니다.

1단계: eksctl 설치

Linux 또는 macOS 디바이스에 최신 버전의 eksctl을 다운로드하고 설치하려면 다음 단계를 완료하세요.

Homebrew를 사용하여 eksctl을 설치하려면

1. (사전 조건) Homebrew를 설치합니다.
2. AWS 탭을 추가합니다.

```
brew tap aws/tap
```

3. eksctl 설치

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

eksctl을 사용하기 전에 다음 구성 단계를 완료합니다.

1. 사전 조건 설치:
   - AWS CLI 버전 2.x 이상을 설치합니다.
   - Homebrew를 사용하여 kubectl을 설치합니다.

```
brew install kubernetes-cli
```

2. 환경에서 AWS 자격 증명을 구성합니다.

```
aws configure
```

3. AWS CLI 구성을 확인합니다.

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

2단계: 클러스터 구성 파일 생성

다음 단계를 사용하여 클러스터 구성 파일을 생성합니다.

1. 라는 새 파일을 생성합니다`cluster.yaml`.

```
touch cluster.yaml
```

2. 다음과 같은 기본 클러스터 구성을 추가합니다.

```
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. 구성을 사용자 지정합니다.
   - 원하는 AWS 리전`region`에 맞게를 업데이트합니다.
   - 워크로드 요구 사항에 `instanceType` 따라를 수정합니다.
   - 필요에 `maxSize` 따라 `minSize`, 및 `desiredCapacity`를 조정합니다.
4. 구성 파일을 검증합니다.

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

3단계: 클러스터 생성

다음 단계에 따라 EKS 클러스터를 생성합니다.

1. 구성 파일을 사용하여 클러스터를 생성합니다.

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

2. 클러스터가 생성될 때까지 기다립니다(일반적으로 15~20분 소요).
3. 클러스터 생성 확인:

```
eksctl get cluster
```

4. 새 클러스터를 사용하도록 kubectl을 구성합니다.

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

5. 클러스터 연결 확인:

```
kubectl get nodes
```

이제 클러스터를 사용할 준비가 되었습니다.

선택 사항: 클러스터 삭제

불필요한 요금을 방지하려면 완료 시 클러스터를 삭제해야 합니다.

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

> **Note**
>
> 클러스터를 생성하면 AWS 요금이 발생할 수 있습니다. 클러스터를 생성하기 전에 Amazon EKS 요금을 검토해야 합니다.

다음 단계

- 클러스터에 연결하도록 Kubectl 구성
- 샘플 앱 배포

Eksctl 설치 옵션

`eksctl`는 아래 설명과 같이 공식 릴리스에서 설치할 수 있습니다. 공식 GitHub 릴리스`eksctl`에서만를 설치하는 것이 좋습니다. 타사 설치 프로그램을 사용하기로 선택할 수 있지만 AWS는 이러한 설치 방법을 유지 관리하거나 지원하지 않습니다. 자체 재량에 따라 사용합니다.

사전 조건

AWS API 자격 증명을 구성해야 합니다. AWS CLI 또는 기타 도구(kops, Terraform 등)에 사용할 수 있는 것으로 충분합니다. `~/.aws/credentials` 파일 또는 환경 변수를 사용할 수 있습니다. 자세한 내용은 AWS CLI 참조를 참조하세요.

또한에 Kubernetes용 AWS IAM Authenticator 명령(`aws-iam-authenticator`또는 `aws eks get-token` (AWS CLI 버전 1.16.156 이상에서 사용 가능)이 필요합니다PATH.

EKS 클러스터 생성에 사용되는 IAM 계정에는 이러한 최소 액세스 수준이 있어야 합니다.

AWS 서비스	액세스 레벨
CloudFormation	전체 액세스
EC2	전체: Tagging Limited: List, Read, Write
EC2 Auto Scaling	제한: 목록, 쓰기
EKS	전체 액세스
IAM	제한: 목록, 읽기, 쓰기, 권한 관리
Systems Manager	제한: 목록, 읽기

Unix의 경우

최신 릴리스를 다운로드하려면 다음을 실행합니다.

```
# 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
```

Windows의 경우

직접 다운로드(최신 릴리스):

- AMD64/x86_64
- ARMv6
- ARMv7
- ARM64

PATH 변수의 폴더에 아카이브의 압축을 풀어야 합니다.

필요에 따라 체크섬을 확인합니다.

1. 체크섬 파일 다운로드: 최신
2. 명령 프롬프트를 사용하여 CertUtil의 출력을 다운로드한 체크섬 파일과 수동으로 비교합니다.

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

3. PowerShell을 사용하여 -eq 연산자를 사용하여 확인을 자동화하여 True 또는 False 결과를 가져옵니다.

```
# Replace amd64 with armv6, armv7 or arm64
 (Get-FileHash -Algorithm SHA256 .\eksctl_Windows_amd64.zip).Hash -eq ((Get-Content .
\eksctl_checksums.txt) -match 'eksctl_Windows_amd64.zip' -split ' ')[0]
```

Git Bash 사용:

```
# 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` 실행 파일은 Git Bash`$PATH`의에 `$HOME/bin`있는에 배치됩니다.

Homebrew

Homebrew를 사용하여 MacOS 및 Linux에 소프트웨어를 설치할 수 있습니다.

AWS는 eksctl을 포함하여 Homebrew 탭을 유지합니다.

Homebrew 탭에 대한 자세한 내용은 Github의 프로젝트와 eksctl용 Homebrew 공식을 참조하세요.

Homebrew를 사용하여 eksctl을 설치하려면

1. (사전 조건) Homebrew 설치
2. AWS 탭 추가

```
brew tap aws/tap
```

3. eksctl 설치

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

Docker

릴리스 및 RC마다 컨테이너 이미지가 ECR 리포지토리 로 푸시됩니다`public.ecr.aws/eksctl/eksctl`. ECR 퍼블릭 갤러리 - eksctl에서 사용에 대해 자세히 알아봅니다. 예:

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

쉘 완료

Bash

bash 완성을 활성화하려면 다음을 실행하거나 `~/.bashrc` 또는에 넣습니다`~/.profile`.

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

Zsh

zsh를 완료하려면 다음을 실행합니다.

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

및는에 `~/.zshrc`다음을 입력합니다.

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

oh-my-zsh와 같은 배포를 실행하지 않는 경우 먼저 자동 완성을 활성화해야 할 수 있습니다(그리고 이를 영구화`~/.zshrc`하려면를 입력해야 할 수 있습니다).

```
autoload -U compinit
compinit
```

어류

아래 명령은 물고기 자동 완성에 사용할 수 있습니다.

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

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

Powershell

아래 명령을 참조하여 설정할 수 있습니다. 경로는 시스템 설정에 따라 다를 수 있습니다.

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

업데이트

> **Important**
>
> eksctl을 직접 다운로드하여 설치하는 경우(패키지 관리자를 사용하지 않음) 수동으로 업데이트해야 합니다.

클러스터

이 장에서는 eksctl을 사용하여 EKS 클러스터를 생성하고 구성하는 방법을 다룹니다. 여기에는 추가 기능 및 EKS Auto Mode도 포함됩니다.

주제:

- the section called “EKS 액세스 항목”
  - aws-auth ConfigMap을 EKS 액세스 항목으로 대체하여 Kubernetes RBAC 관리 간소화
  - 기존 IAM 자격 증명 매핑을 aws-auth ConfigMap에서 액세스 항목으로 마이그레이션
  - 클러스터 인증 모드 구성 및 클러스터 생성자 관리자 권한 제어
- the section called “기본 추가 기능 업데이트”
  - 이전 클러스터에서 기본 EKS 추가 기능을 업데이트하여 클러스터 보안 유지
- the section called “추가 기능”
  - 추가 기능 설치, 업데이트 및 제거를 위한 일상적인 작업을 자동화합니다.
  - Amazon EKS 추가 기능에는 AWS 추가 기능, 오픈 소스 커뮤니티 추가 기능 및 마켓플레이스 추가 기능이 포함됩니다.
- the section called “EKS Auto Mode”
  - AWS가 EKS 인프라를 관리하도록 하여 운영 오버헤드 감소
  - 기본 범용 및 시스템 풀 대신 사용자 지정 노드 풀 구성
  - Auto Mode를 사용하도록 기존 EKS 클러스터 변환
- the section called “CloudWatch 로깅”
  - 특정 EKS 컨트롤 플레인 구성 요소에 대한 로그를 활성화하여 클러스터 문제 해결
  - EKS 클러스터 로그의 로그 보존 기간 구성
  - eksctl 명령을 사용하여 기존 클러스터 로깅 설정 수정
- the section called “클러스터 업그레이드”
  - EKS 컨트롤 플레인 버전을 안전하게 업그레이드하여 보안 및 안정성 유지
  - 이전 그룹을 새 그룹으로 대체하여 노드 그룹 간에 업그레이드 롤아웃
  - 기본 클러스터 추가 기능 업데이트
- the section called “클러스터 생성 및 관리”
  - 기본 관리형 노드 그룹을 사용하여 기본 EKS 클러스터로 빠르게 시작

  - 특정 구성의 구성 파일을 사용하여 사용자 지정 클러스터 생성
  - 프라이빗 네트워킹 및 사용자 지정 IAM 정책을 사용하여 기존 VPCs에 클러스터 배포
- the section called “kubelet 구성”
  - kubelet 및 시스템 데몬 예약을 구성하여 노드 리소스 결핍 방지
  - 메모리 및 파일 시스템 가용성을 위한 제거 임계값 사용자 지정
  - 노드 그룹 전체에서 특정 kubelet 기능 게이트 활성화 또는 비활성화
- the section called “EKS 커넥터”
  - EKS 콘솔을 통한 하이브리드 Kubernetes 배포 관리 중앙 집중화
  - 외부 클러스터 액세스를 위한 IAM 역할 및 권한 구성
  - 외부 클러스터 제거 및 연결된 AWS 리소스 정리
- the section called “EKS 완전 프라이빗 클러스터”
  - 아웃바운드 인터넷에 액세스할 수 없는 완전 프라이빗 EKS 클러스터로 보안 요구 사항 충족
  - VPC 엔드포인트를 통해 AWS 서비스에 대한 프라이빗 액세스 구성
  - 명시적 네트워킹 설정을 사용하여 프라이빗 노드 그룹 생성 및 관리
- the section called “Karpenter 지원”
  - 노드 프로비저닝 자동화
  - 사용자 지정 Karpenter 프로비저너 구성 생성
  - 스팟 인스턴스 중단 처리로 Karpenter 설정
- the section called “Amazon EMR”
  - EMR 클러스터와 EKS 클러스터 간에 IAM 자격 증명 매핑 생성
- the section called “EKS Fargate 지원”
  - 포드 예약을 위한 사용자 지정 Fargate 프로필 정의
  - 생성 및 구성 업데이트를 통해 Fargate 프로필 관리
- the section called “eksctl에서 생성하지 않은 클러스터”
  - eksctl 외부에서 생성된 클러스터 관리 표준화
  - 기존 비eksctl 클러스터에서 eksctl 명령 사용
- the section called “영역 전환 활성화”
  - 신속 영역 장애 조치 기능을 활성화하여 애플리케이션 가용성 향상
  - 새 EKS 클러스터 배포에서 영역 전환 구성

  - 기존 EKS 클러스터에서 영역 전환 기능 활성화

클러스터 생성 및 관리

이 주제에서는 Eksctl을 사용하여 EKS 클러스터를 생성하고 삭제하는 방법을 다룹니다. CLI 명령을 사용하거나 클러스터 구성 YAML 파일을 생성하여 클러스터를 생성할 수 있습니다.

간단한 클러스터 생성

다음 명령을 사용하여 간단한 클러스터를 생성합니다.

```
eksctl create cluster
```

그러면 기본 리전(AWS CLI 구성에 의해 지정됨)에 m5.large 노드 2개가 포함된 관리형 노드 그룹 1개가 있는 EKS 클러스터가 생성됩니다.

이제 eksctl은 구성 파일을 사용하지 않을 때 기본적으로 관리형 노드 그룹을 생성합니다. 자체 관리형 노드 그룹을 생성하려면 `eksctl create cluster` 또는 `--managed=false`에 전달합니다`eksctl create nodegroup`.

고려 사항

- 에서 클러스터를 생성할 때가 발생할 `us-east-1`수 있습니다`UnsupportedAavailabilityZoneException`. 이 경우 제안된 영역을 복사하고 `--zones` 플래그를 전달합니다. 예: `eksctl create cluster --region=us-east-1 --zones=us-east-1a,us-east-1b,us-east-1d`. 이 문제는 다른 리전에서 발생할 수 있지만 덜 일반적입니다. 대부분의 경우 `--zone` 플래그를 사용할 필요가 없습니다.

구성 파일을 사용하여 클러스터 생성

플래그 대신 구성 파일을 사용하여 클러스터를 생성할 수 있습니다.

먼저 `cluster.yaml` 파일을 생성합니다.

```
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
```

그런 다음 다음 명령을 실행합니다.

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

이렇게 하면 설명된 대로 클러스터가 생성됩니다.

기존 VPC를 사용해야 하는 경우 다음과 같은 구성 파일을 사용할 수 있습니다.

```
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
```

클러스터 이름 또는 노드 그룹 이름에는 영숫자(대소문자 구분)와 하이픈만 포함되어야 합니다. 영문자로 시작해야 하며 128자를 초과할 수 없습니다. 그렇지 않으면 검증 오류가 발생합니다. 자세한 내용은 AWS CLoudFormation 사용 설명서의 CloudFormation 콘솔에서 스택 생성을 참조하세요. CLoudFormation

새 클러스터에 대한 kubeconfig 업데이트

클러스터가 생성되면 적절한 kubernetes 구성이 kubeconfig 파일에 추가됩니다. 환경 변수 `KUBECONFIG` 또는 `~/.kube/config` 기본적으로 구성한 파일입니다. `--kubeconfig` 플래그를 사용하여 kubeconfig 파일의 경로를 재정의할 수 있습니다.

kubeconfig 파일 작성 방법을 변경할 수 있는 기타 플래그:

플래그	type	사용	기본값
--kubeconfig	문자열	kubeconfig 쓰기 경로 (--auto-kubeconfig와 호환되지 않음)	$KUBECONFIG 또는 ~/.kube/config
--set-kubeconfig-context	bool	true인 경우 현재 컨텍스트가 kubeconfig로 설정됩니다. 컨텍스트가 이미 설정된 경우 컨텍스트를 덮어씁니다.	true
--auto-kubeconfig	bool	클러스터 이름으로 kubeconfig 파일 저장	true
--write-kubeconfig	bool	kubeconfig 쓰기 토글	true

클러스터 삭제

이 클러스터를 삭제하려면 다음을 실행합니다.

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

> **Warning**
>
> 삭제 작업과 함께 `--wait` 플래그를 사용하여 삭제 오류가 제대로 보고되도록 합니다.

`--wait` 플래그가 없으면 eksctl은 클러스터의 CloudFormation 스택에만 삭제 작업을 실행하고 삭제를 기다리지 않습니다. 경우에 따라 클러스터 또는 VPC를 사용하는 AWS 리소스로 인해 클러스터 삭제가 실패할 수 있습니다. 삭제에 실패하거나 대기 플래그를 잊어버린 경우 CloudFormation GUI로 이동하여 여기에서 eks 스택을 삭제해야 할 수 있습니다.

> **Warning**
>
> PDB 정책은 클러스터 삭제 중에 노드 제거를 차단할 수 있습니다.

노드 그룹이 있는 클러스터를 삭제할 때 포드 중단 예산(PDB) 정책은 노드가 성공적으로 제거되지 않도록 할 수 있습니다. 예를 들어가 `aws-ebs-csi-driver` 설치된 클러스터에는 일반적으로 한 번에 하나의 포드만 사용할 수 있도록 허용하는 PDB 정책이 있는 포드가 두 개 있으므로 삭제 중에 다른 포드를 제거할 수 없습니다. 이러한 시나리오에서 클러스터를 성공적으로 삭제하려면 `disable-nodegroup-eviction` 플래그를 사용하여 PDB 정책 검사를 우회합니다.

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

자세한 샘플 구성 파일은 eksctl GitHub 리포지토리의 `examples/` 디렉터리를 참조하세요.

드라이 런

모의 실행 기능을 사용하면 클러스터 생성을 ClusterConfig 건너뛰고 제공된 CLI 옵션을 나타내며 eksctl에서 설정한 기본값을 포함하는 ClusterConfig 파일을 출력하는 ClusterConfig 파일을 생성할 수 있습니다.

자세한 내용은 드라이 런 페이지에서 확인할 수 있습니다.

EKS Auto Mode

eksctl은 Kubernetes 클러스터의 AWS 관리를 클러스터 자체 이상으로 확장하여 AWS가 워크로드를 원활하게 운영할 수 있는 인프라를 설정하고 관리할 수 있도록 하는 기능인 EKS Auto Mode를 지원합니다. 이를 통해 주요 인프라 결정을 위임하고 일상적인 운영에 AWSday-to-day. AWS에서 관리하는 클러스터 인프라에는 컴퓨팅 오토 스케일링, 포드 및 서비스 네트워킹, 애플리케이션 로드 밸런싱, 클러스터 DNS, 블록 스토리지 및 GPU 지원과 같은 추가 기능과 달리 많은 Kubernetes 기능이 핵심 구성 요소로 포함되어 있습니다.

자동 모드가 활성화된 상태에서 EKS 클러스터 생성

`eksctl`에서 자동 모드를 활성화하고 구성하기 위해 새 `autoModeConfig` 필드를 추가했습니다. `autoModeConfig` 필드의 모양은 입니다.

```
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
```

true인 경우 eksctl`autoModeConfig.enabled`은 , `kubernetesNetworkConfig.elasticLoadBalancing.enabled: true`및 `computeConfig.enabled: true`를 EKS API에 전달`storageConfig.blockStorage.enabled: true`하여 EKS 클러스터를 생성하여 컴퓨팅, 스토리지 및 네트워킹과 같은 데이터 영역 구성 요소를 관리할 수 있습니다.

자동 모드가 활성화된 EKS 클러스터를 생성하려면 에서와 `autoModeConfig.enabled: true`같이 를 설정합니다.

```
# 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
```

eksctl은 Auto Mode에서 시작한 노드에 사용할 노드 역할을 생성합니다. eksctl은 `general-purpose` 및 `system` 노드 풀도 생성합니다. 다른 서브넷 세트를 사용하는 자체 노드 풀을 구성하는 등 기본 노드 풀 생성을 비활성화하려면 에서와 `nodePools: []`같이를 설정합니다.

```
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.
```

자동 모드를 사용하도록 EKS 클러스터 업데이트

Auto Mode를 사용하도록 기존 EKS 클러스터를 업데이트하려면

```
# 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**
>
> 클러스터가 eksctl에 의해 생성되고 퍼블릭 서브넷을 클러스터 서브넷으로 사용하는 경우 Auto Mode는 퍼블릭 서브넷에서 노드를 시작합니다. Auto Mode에서 시작한 작업자 노드에 프라이빗 서브넷을 사용하려면 프라이빗 서브넷을 사용하도록 클러스터를 업데이트합니다.

자동 모드 비활성화

자동 모드를 비활성화하려면 `autoModeConfig.enabled: false`를 설정하고 실행합니다.

```
# 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
```

추가 정보

- EKS Auto Mode

EKS 액세스 항목

eksctl을 사용하여 EKS 액세스 항목을 관리할 수 있습니다. 액세스 항목을 사용하여 AWS IAM 자격 증명에 Kubernetes 권한을 부여합니다. 예를 들어 개발자 역할에 클러스터의 Kubernetes 리소스를 읽을 수 있는 권한을 부여할 수 있습니다.

이 주제에서는 eksctl을 사용하여 액세스 항목을 관리하는 방법을 다룹니다. 액세스 항목에 대한 일반적인 내용은 IAM 사용자에게 EKS 액세스 항목을 사용하여 Kubernetes에 대한 액세스 권한 부여를 참조하세요.

AWS에서 정의한 Kubernetes 액세스 정책을 연결하거나 IAM 자격 증명을 Kubernetes 그룹과 연결할 수 있습니다.

사용 가능한 사전 정의된 정책에 대한 자세한 내용은 액세스 정책과 액세스 항목 연결을 참조하세요.

고객 Kubernetes 정책을 정의해야 하는 경우 IAM 자격 증명을 Kubernetes 그룹과 연결하고 해당 그룹에 권한을 부여합니다.

클러스터 인증 모드

클러스터의 인증 모드에서 허용하는 경우에만 액세스 항목을 사용할 수 있습니다.

자세한 내용은 클러스터 인증 모드 설정을 참조하세요.

YAML 파일로 인증 모드 설정

`eksctl`는 ClusterConfig 아래에 다음 세 가지 값 중 하나로 설정할 수 있는 새 `accessConfig.authenticationMode` 필드를 추가했습니다.

- `CONFIG_MAP` - EKS API의 기본값 - `aws-auth` ConfigMap만 사용됩니다.
- `API` - 액세스 항목 API만 사용됩니다.
- `API_AND_CONFIG_MAP` -의 기본값 `eksctl` - `aws-auth` ConfigMap 및 액세스 항목 API를 모두 사용할 수 있습니다.

ClusterConfig YAML에서 인증 모드 설정:

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

명령을 사용하여 인증 모드 업데이트

`CONFIG_MAP` 옵션이 사용되는 이미 존재하는 비eksctl 생성 클러스터에서 액세스 항목을 사용하려면 먼저 사용자가를 `authenticationMode`로 설정해야 합니다`API_AND_CONFIG_MAP`. 이를 위해 `eksctl`는 CLI 플래그와 함께 작동하는 클러스터 인증 모드를 업데이트하기 위한 새 명령을 도입했습니다. 예:

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

액세스 항목 리소스

액세스 항목에는 `STANDARD` 또는와 같은 유형이 있습니다`EC2_LINUX`. 유형은 액세스 항목을 사용하는 방법에 따라 달라집니다.

- `standard` 유형은 IAM 사용자 및 IAM 역할에 Kubernetes 권한을 부여하는 것입니다.
  - 예를 들어, 콘솔에 액세스하는 데 사용하는 역할 또는 사용자에 액세스 정책을 연결하여 AWS 콘솔에서 Kubernetes 리소스를 볼 수 있습니다.
- `EC2_LINUX` 및 `EC2_WINDOWS` 유형은 EC2 인스턴스에 Kubernetes 권한을 부여하기 위한 것입니다. 인스턴스는 이러한 권한을 사용하여 클러스터에 조인합니다.

액세스 항목 유형에 대한 자세한 내용은 액세스 항목 생성을 참조하세요.

IAM 엔터티

액세스 항목을 사용하여 IAM 사용자 및 IAM 역할과 같은 IAM 자격 증명에 Kubernetes 권한을 부여할 수 있습니다.

`accessConfig.accessEntries` 필드를 사용하여 IAM 리소스의 ARN을 액세스 항목 EKS API와 연결합니다. 예제:

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

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

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

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

EKS 정책을 연결하는 것 외에도 IAM 엔터티가 속한 Kubernetes 그룹을 지정하여 RBAC를 통해 권한을 부여할 수도 있습니다.

관리형 노드 그룹 및 Fargate

이러한 리소스에 대한 액세스 항목과의 통합은 EKS API에 의해 백그라운드에서 이루어집니다. 새로 생성된 관리형 노드 그룹 및 Fargate 포드는 사전 로드된 RBAC 리소스를 사용하는 대신 API 액세스 항목을 생성합니다. 기존 노드 그룹과 Fargate 포드는 변경되지 않으며 aws-auth 구성 맵의 항목에 계속 의존합니다.

자체 관리형 노드 그룹

각 액세스 항목에는 유형이 있습니다. 자체 관리형 노드 그룹에 권한을 부여하기 위해 `eksctl`는 보안 주체 ARN이 노드 역할 ARN으로 설정되고 유형이 노드 그룹 amiFamily에 `EC2_WINDOWS` 따라 `EC2_LINUX` 또는 로 설정된 각 노드 그룹에 대해 고유한 액세스 항목을 생성합니다.

자체 액세스 항목을 생성할 때 `EC2_LINUX` (Linux 또는 Bottlerocket 자체 관리형 노드에 사용되는 IAM 역할의 경우), `EC2_WINDOWS` (Windows 자체 관리형 노드에 사용되는 IAM 역할의 경우), `FARGATE_LINUX` (AWS Fargate(Fargate)에 사용되는 IAM 역할의 경우) 또는 `STANDARD` 유형을 지정할 수도 있습니다. 유형을 지정하지 않으면 기본 유형이 로 설정됩니다`STANDARD`.

> **Note**
>
> 기존 로 생성된 노드 그룹을 삭제할 때 연결된 노드 그룹이 더 이상 없는 경우 해당 액세스 항목을 삭제하는 `instanceRoleARN`것은 사용자의 책임입니다. 이는 eksctl이 액세스 항목이 아직 eksctl이 생성하지 않은 자체 관리형 노드 그룹에서 사용 중인지 여부를 확인하려고 시도하지 않기 때문입니다. 복잡한 프로세스이기 때문입니다.

액세스 항목 생성

이는 클러스터 생성 중에 구성 파일의 일부로 원하는 액세스 항목을 지정하고 실행하는 두 가지 방법으로 수행할 수 있습니다.

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

다음을 실행하여 클러스터 생성 후 OR:

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

액세스 항목을 생성하기 위한 구성 파일의 예는 eksctl GitHub 리포지토리의 40-access-entries.yaml을 참조하세요.

액세스 항목 가져오기

사용자는 다음 중 하나를 실행하여 특정 클러스터와 연결된 모든 액세스 항목을 검색할 수 있습니다.

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

또는

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

또는 특정 IAM 엔터티에 해당하는 액세스 항목만 검색하려면 `--principal-arn` 플래그를 사용해야 합니다. 예:

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

액세스 항목 삭제

한 번에 단일 액세스 항목을 삭제하려면 다음을 사용합니다.

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

여러 액세스 항목을 삭제하려면 `--config-file` 플래그를 사용하고 최상위 `accessEntry` 필드에서 액세스 항목에 `principalARN's` 해당하는를 모두 지정합니다. 예:

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

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

aws-auth ConfigMap에서 마이그레이션

사용자는 다음을 실행하여 기존 IAM 자격 증명을 aws-auth configmap에서 액세스 항목으로 마이그레이션할 수 있습니다.

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

--target-authentication-mode 플래그를 로 설정하면 API인증 모드가 API 모드로 전환되고(이미 API 모드에 있는 경우 건너뛰기), IAM 자격 증명 매핑이 액세스 항목으로 마이그레이션되며, aws-auth configmap이 클러스터에서 삭제됩니다.

--target-authentication-mode 플래그를 로 설정하면 API_AND_CONFIG_MAP인증 모드가 API_AND_CONFIG_MAP 모드로 전환되고(이미 API_AND_CONFIG_MAP 모드에 있는 경우 건너뛰기), IAM 자격 증명 매핑이 액세스 항목으로 마이그레이션되지만 구성 aws-auth 맵은 보존됩니다.

> **Note**
>
> --target-authentication-mode 플래그가 로 설정된 경우 APIaws-authconfigmap에 아래 제약 조건 중 하나가 있는 경우이 명령은 인증 API 모드를 모드로 업데이트하지 않습니다.

- 계정 수준 자격 증명 매핑이 있습니다.
- 하나 이상의 역할/사용자가 접두사로 시작하는 kubernetes 그룹(들)에 매핑됩니다system:(예: system:masters, system:bootstrappers system:nodes 등의 EKS 특정 그룹 제외).
- 하나 이상의 IAM 자격 증명 매핑(Service Linked Role](link:IAM/latest/UserGuide/using-service-linked-roles.html)입니다.

클러스터 생성자 관리자 권한 비활성화

eksctl는 false로 설정하면 클러스터를 생성하는 IAM 자격 증명에 클러스터 관리자 권한 부여를 비활성화accessConfig.bootstrapClusterCreatorAdminPermissions: boolean하는 새 필드를 추가했습니다.

구성 파일에 옵션을 추가합니다.

```
accessConfig:
  bootstrapClusterCreatorAdminPermissions: false
```

및 실행:

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

eksctl에서 생성하지 않은 클러스터

에서 생성하지 않은 클러스터에 대해 `eksctl` 명령을 실행할 수 있습니다`eksctl`.

> **Note**
>
> Eksctl은 AWS CloudFormation과 호환되는 이름을 가진 소유하지 않은 클러스터만 지원할 수 있습니다. 이 클러스터 이름과 일치하지 않는 클러스터 이름은 CloudFormation API 검증 검사에 실패합니다.

지원되는 명령

다음 명령은 이외의 방법으로 생성된 클러스터에 사용할 수 있습니다`eksctl`. 명령, 플래그 및 구성 파일 옵션은 정확히 동일한 방식으로 사용할 수 있습니다.

일부 기능이 누락된 경우 알려주세요.

- ✓ 생성:
  - ✓ `eksctl create nodegroup` (아래 참고 사항 참조)
  - ✓ `eksctl create fargateprofile`
  - ✓ `eksctl create iamserviceaccount`
  - ✓ `eksctl create iamidentitymapping`
- ✓ 다음을 가져옵니다.
  - ✓ `eksctl get clusters/cluster`
  - ✓ `eksctl get fargateprofile`

- ✓ eksctl get nodegroup
- ✓ eksctl get labels

✓ 삭제:

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

✓ 업그레이드:

- ✓ eksctl upgrade cluster
- ✓ eksctl upgrade nodegroup

✓ 설정/설정 해제:

- ✓ eksctl set labels
- ✓ eksctl unset labels

✓ 규모 조정:

- ✓ eksctl scale nodegroup

✓ 드레이닝:

- ✓ eksctl drain nodegroup

✓ 활성화:

- ✓ eksctl enable profile
- ✓ eksctl enable repo

✓ 유틸리티:

- ✓ 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`

노드 그룹 생성

`eksctl create nodegroup`는 사용자의 특정 입력이 필요한 유일한 명령입니다.

사용자는 원하는 네트워킹 구성으로 클러스터를 생성할 수 있으므로 시간상 `eksctl`는 이러한 값을 검색하거나 추측하려고 시도하지 않습니다. 이는 향후 사람들이 eksctl에서 생성하지 않은 클러스터에서 이 명령을 사용하는 방법에 대해 자세히 알아볼 때 변경될 수 있습니다.

즉`eksctl`,에서 생성하지 않은 클러스터에서 노드 그룹 또는 관리형 노드 그룹을 생성하려면 VPC 세부 정보가 포함된 구성 파일을 제공해야 합니다. 최소한:

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

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

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

...
```

VPC 구성 옵션에 대한 자세한 내용은 네트워킹을 참조하세요.

EKS Connector를 사용하여 비EKS 클러스터 등록

EKS 커넥터를 사용하여 EKS 콘솔에서 AWS 외부의 클러스터를 볼 수 있습니다. 이 프로세스를 수행하려면 EKS에 클러스터를 등록하고 외부 Kubernetes 클러스터에서 EKS 커넥터 에이전트를 실행해야 합니다.

`eksctl`는 필요한 AWS 리소스를 생성하고 EKS 커넥터가 외부 클러스터에 적용할 Kubernetes 매니페스트를 생성하여 비EKS 클러스터 등록을 간소화합니다.

클러스터 등록

비EKS Kubernetes 클러스터를 등록하거나 연결하려면를 실행합니다.

```
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
```

이 명령은 클러스터를 등록하고 등록이 만료되기 전에 외부 클러스터에 적용해야 하는 EKS Connector용 Kubernetes 매니페스트가 포함된 3개의 파일을 작성합니다.

> **Note**
>
> `eks-connector-clusterrole.yaml` 및는 모든 네임스페이스의 Kubernetes 리소스에 대해 호출 IAM 자격 증명에 `get` 및 `list` 권한을 `eks-connector-console-dashboard-full-access-clusterrole.yaml` 부여하며, 필요한 경우 클러스터에 적용하기 전에 적절히 편집해야 합니다. 더 제한된 액세스를 구성하려면 사용자에게 클러스터를 볼 수 있는 액세스 권한 부여를 참조하세요.

EKS 커넥터에 사용할 기존 IAM 역할을 제공하려면 다음과 `--role-arn` 같이를 통해 전달합니다.

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

클러스터가 이미 있는 경우 eksctl은 오류를 반환합니다.

클러스터 등록 취소

등록된 클러스터의 등록을 취소하거나 연결을 해제하려면를 실행합니다.

```
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
```

이 명령은 외부 클러스터의 등록을 취소하고 연결된 AWS 리소스를 제거하지만 클러스터에서 EKS 커넥터 Kubernetes 리소스를 제거해야 합니다.

추가 정보

- EKS 커넥터

kubelet 구성 사용자 지정

시스템 리소스는 kubelet의 구성을 통해 예약할 수 있습니다. 이는 리소스 결핍의 경우 kubelet이 포드를 제거하지 못하고 결국 노드를 로 만들 수 있기 때문에 권장됩니다`NotReady`. 이를 위해 구성 파일에는에 포함할 자유 형식 yaml을 허용하는 `kubeletExtraConfig` 필드가 포함될 수 있습니다`kubelet.yaml`.

의 일부 필드는 eksctl에 의해 `kubelet.yaml` 설정되므로 , `address`, `clusterDomain`, `authentication` `authorization`또는와 같이 덮어쓸 수 없습니다`serverTLSBootstrap`.

다음 예제 구성 파일은 kubelet`300Mi`에 대한 `300m` vCPU, 메모리 및 `1Gi` 임시 스토리지, OS 시스템 데몬에 대한 `300m` vCPU, `300Mi` 메모리 및 `1Gi`임시 스토리지를 예약하고 사용 가능한 `200Mi` 메모리가 적거나 루트 파일 시스템의 10% 미만일 때 포드를 제거하는 노드 그룹을 생성합니다.

```
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
```

이 예제에서는 vCPUs 4개와 메모리 16GiBm5a.xlarge가 있는 유형의 인스턴스가 주어지면 CPUs Allocatable 양은 메모리 3.4GiB와 15.4GiB가 됩니다. 의 필드에 대한 구성 파일에 지정된 값이 eksctl에서 지정한 기본값을 kubeletExtraconfig 완전히 덮어쓴다는 점에 유의해야 합니다. 그러나 하나 이상의 kubeReserved 파라미터를 생략하면 사용 중인 aws 인스턴스 유형에 따라 누락된 파라미터가 sane 값으로 기본 설정됩니다.

kubeReserved 계산

일반적으로 동일한 CPU 및 RAM 구성의 인스턴스를 사용하도록 혼합 인스턴스 NodeGroup을 구성하는 것이 좋지만 엄격한 요구 사항은 아닙니다. 따라서 kubeReserved 계산은 InstanceDistribution.InstanceTypes 필드에서 가장 작은 인스턴스를 사용합니다. 이렇게 하면 서로 다른 인스턴스 유형을 가진 NodeGroups가 가장 작은 인스턴스에 너무 많은 리소스를 예약하지 않습니다. 그러나 이로 인해 가장 큰 인스턴스 유형에 비해 예약이 너무 작을 수 있습니다.

> **Warning**
>
> 기본적으로를 `eksctl` 설정`featureGates.RotateKubeletServerCertificate=true`하지만 사용자 지정`featureGates`이 제공되면 설정이 해제됩니다. 비활성화해야 하는 `featureGates.RotateKubeletServerCertificate=true`경우가 아니면 항상를 포함해야 합니다.

CloudWatch 로깅

이 주제에서는 EKS 클러스터의 컨트롤 플레인 구성 요소에 대해 Amazon CloudWatch 로깅을 구성하는 방법을 설명합니다. CloudWatch 로깅은 문제 해결, 클러스터 활동 감사, Kubernetes 구성 요소 상태 모니터링에 필수적인 클러스터의 컨트롤 플레인 작업에 대한 가시성을 제공합니다.

CloudWatch 로깅 활성화

데이터 수집 및 스토리지 비용으로 인해 EKS 컨트롤 플레인에 대한 CloudWatch 로깅은 기본적으로 활성화되지 않습니다.

클러스터가 생성될 때 컨트롤 플레인 로깅을 활성화하려면에서 **`cloudWatch.clusterLogging.enableTypes`** 설정을 정의해야 합니다`ClusterConfig`(아래 예제 참조).

따라서 올바른 **`cloudWatch.clusterLogging.enableTypes`** 설정의 구성 파일이 있는 경우를 사용하여 클러스터를 생성할 수 있습니다`eksctl create cluster --config-file=<path>`.

클러스터를 이미 생성한 경우를 사용할 수 있습니다`eksctl utils update-cluster-logging`.

> **Note**
>
> 이 명령은 기본적으로 계획 모드에서 실행되며 변경 사항을 클러스터에 적용하려면 `--approve` 플래그를 지정해야 합니다.

구성 파일을 사용하는 경우 다음을 실행합니다.

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

또는 CLI 플래그를 사용할 수 있습니다.

모든 유형의 로그를 활성화하려면 다음을 실행합니다.

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

audit 로그를 활성화하려면 다음을 실행합니다.

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

controllerManager 로그를 제외한 모든 로그를 활성화하려면 다음을 실행합니다.

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

api 및 scheduler 로그 유형이 이미 활성화된 경우 controllerManager를 비활성화scheduler하고 동시에 활성화하려면 다음을 실행합니다.

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

이렇게 하면 api 및가 유일한 로그 유형controllerManager으로 활성화됩니다.

모든 유형의 로그를 비활성화하려면 다음을 실행합니다.

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

ClusterConfig 예제

EKS 클러스터에서 아래의 enableTypes 필드는 가능한 값 목록을 가져와 컨트롤 플레인 구성 요소에 대해 다양한 유형의 로그를 활성화할 clusterLogging 수 있습니다.

사용 가능한 값은 다음과 같습니다.

- api: Kubernetes API 서버 로그를 활성화합니다.
- audit: Kubernetes 감사 로그를 활성화합니다.
- authenticator: 인증자 로그를 활성화합니다.
- controllerManager: Kubernetes 컨트롤러 관리자 로그를 활성화합니다.

- `scheduler`: Kubernetes 스케줄러 로그를 활성화합니다.

자세한 내용은 EKS 설명서를 참조하세요.

모든 로그 비활성화

모든 유형을 비활성화하려면 `cloudWatch` 섹션을 사용하거나 완전히 `[]` 제거합니다.

모든 로그 활성화

`"*"` 또는를 사용하여 모든 유형을 활성화할 수 있습니다`"all"`. 예제:

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

하나 이상의 로그 활성화

활성화하려는 유형을 나열하여 유형의 하위 집합을 활성화할 수 있습니다. 예제:

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

로그 보존 기간

기본적으로 로그는 CloudWatch Logs에 무기한 저장됩니다. CloudWatch Logs에서 컨트롤 플레인 로그를 보존해야 하는 일수를 지정할 수 있습니다. 다음 예제에서는 로그를 7일 동안 유지합니다.

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

전체 예제

```
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 완전 프라이빗 클러스터

eksctl은 아웃바운드 인터넷에 액세스할 수 없고 프라이빗 서브넷만 있는 완전 프라이빗 클러스터 생성을 지원합니다. VPC 엔드포인트는 AWS 서비스에 대한 프라이빗 액세스를 활성화하는 데 사용됩니다.

이 가이드에서는 아웃바운드 인터넷 액세스 없이 프라이빗 클러스터를 생성하는 방법을 설명합니다.

완전 프라이빗 클러스터 생성

완전 프라이빗 클러스터를 생성하는 데 필요한 유일한 필드는 입니다. `privateCluster.enabled`

```
privateCluster:
  enabled: true
```

클러스터 생성 후 Kubernetes API 서버에 액세스해야 하는 eksctl 명령은 클러스터의 VPC, 피어링된 VPC 또는 AWS Direct Connect와 같은 다른 수단을 사용하여 실행해야 합니다. EKS APIs에 액세스해야 하는 eksctl 명령은 클러스터의 VPC 내에서 실행되는 경우 작동하지 않습니다. 이 문제를 해결하려면 Amazon EKS가 Amazon Virtual Private Cloud(VPC)에서 Amazon Elastic Kubernetes Service(Amazon EKS) 관리 API에 비공개로 액세스할 수 있도록 인터페이스 엔드포인트를 생성합니다. APIs Amazon Virtual Private Cloud 향후 릴리스에서 eksctl은이 엔드포인트를 생성하는 지원을 추가하므로 수동으로 생성할 필요가 없습니다. Amazon EKS용 AWS PrivateLink를 활성화한 후에는 OpenID Connect 공급자 URL에 액세스해야 하는 명령을 클러스터의 VPC 외부에서 실행해야 합니다.

클러스터의 VPC, 피어링된 VPC 내에서 또는 AWS Direct Connect와 같은 다른 수단을 사용하여 명령을 실행하는 경우 관리형 노드 그룹을 생성하면 계속 작동하며 자체 관리형 노드 그룹을 생성하면 EKS 인터페이스 엔드포인트를 통해 API 서버에 액세스해야 하므로 작동합니다.

> **Note**
>
> VPC 엔드포인트는 사용량에 따라 시간 단위로 요금이 청구됩니다. 요금에 대한 자세한 내용은 AWS PrivateLink 요금에서 확인할 수 있습니다.

> **Warning**
>
> 완전 프라이빗 클러스터는에서 지원되지 않습니다`eu-south-1`.

추가 AWS 서비스에 대한 프라이빗 액세스 구성

작업자 노드가 AWS 서비스에 비공개로 액세스할 수 있도록 eksctl은 다음 서비스에 대한 VPC 엔드포인트를 생성합니다.

- 컨테이너 이미지를 가져오기 위한 ECR(`ecr.api` 및 모두`ecr.dkr`)의 인터페이스 엔드포인트 (AWS CNI 플러그인 등)
- S3가 실제 이미지 계층을 가져오는 게이트웨이 엔드포인트
- `aws-cloud-provider` 통합에 필요한 EC2용 인터페이스 엔드포인트
- 서비스 계정에 대한 Fargate 및 IAM 역할(IRSA)을 지원하는 STS용 인터페이스 엔드포인트
- CloudWatch 로깅이 활성화된 경우 CloudWatch 로깅을 위한 인터페이스 엔드포인트(`logs`)

이러한 VPC 엔드포인트는 기능 프라이빗 클러스터에 필수이므로 eksctl은 구성 또는 비활성화를 지원하지 않습니다. 그러나 클러스터는 다른 AWS 서비스에 대한 프라이빗 액세스가 필요할 수 있습니다(예: Cluster Autoscaler에 필요한 Autoscaling). 이러한 서비스는에서 지정할 수 있으며`privateCluster.additionalEndpointServices`, eksctl에 각 서비스에 대한 VPC 엔드포인트를 생성하도록 지시합니다.

예를 들어 Autoscaling 및 CloudWatch 로깅에 대한 프라이빗 액세스를 허용하려면

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

```
  - "logs"
```

에서 지원되는 엔드포인트는 autoscaling, cloudformation 및 additionalEndpointServices입니다logs.

엔드포인트 생성 건너뛰기

필요한 AWS 엔드포인트가 설정되고 EKS 설명서에 설명된 서브넷에 연결된 상태로 VPC가 이미 생성된 경우는 다음과 skipEndpointCreation 같은 옵션을 제공하여 VPC 생성을 건너뛸 eksctl 수 있습니다.

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

이 설정은와 함께 사용할 수 없습니다additionalEndpointServices. 모든 엔드포인트 생성을 건너뜁니다. 또한이 설정은 엔드포인트 <# 서브넷 토폴로지가 올바르게 설정된 경우에만 권장됩니다. 서브넷 ID가 올바르고 vpce 라우팅이 접두사 주소로 설정되면 필요한 모든 EKS 엔드포인트가 생성되어 제공된 VPC에 연결됩니다. eksctl는 이러한 리소스를 변경하지 않습니다.

노드 그룹

클러스터의 VPC는 퍼블릭 서브넷 없이 생성되므로 프라이빗 노드 그룹(관리형 및 자체 관리형 모두)만 완전 프라이빗 클러스터에서 지원됩니다. privateNetworking 필드 (nodeGroup**[].privateNetworking** 및 managedNodeGroup[])는 명시적으로 설정해야 합니다. 완전 프라이빗 클러스터에 설정privateNetworking되지 않은 상태로 두는 것은 오류입니다.

```
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
```

클러스터 엔드포인트 액세스

완전 프라이빗 클러스터는 클러스터 생성 `clusterEndpointAccess` 중 수정을 지원하지 않습니다. 완전 프라이빗 클러스터는 프라이빗 액세스만 할 수 있고 이러한 필드를 수정하도록 허용하면 클러스터가 중단될 수 `clusterEndpoints.privateAccess`있으므로 `clusterEndpoints.publicAccess` 또는를 설정하는 것은 오류입니다.

사용자 제공 VPC 및 서브넷

eksctl은 기존 VPC 및 서브넷을 사용하여 완전 프라이빗 클러스터 생성을 지원합니다. 프라이빗 서브넷만 지정할 수 있으며 아래에 서브넷을 지정하는 것은 오류입니다`vpc.subnets.public`.

eksctl은 제공된 VPC에 VPC 엔드포인트를 생성하고 제공된 서브넷의 라우팅 테이블을 수정합니다. eksctl은 기본 라우팅 테이블을 수정하지 않으므로 각 서브넷에는 명시적 라우팅 테이블이 연결되어 있어야 합니다.

```
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
```

완전 프라이빗 클러스터 관리

클러스터 생성 후 모든 명령이 작동하려면 eksctl에서 EKS API 서버 엔드포인트에 대한 프라이빗 액세스와 아웃바운드 인터넷 액세스(용)가 필요합니다`EKS:DescribeCluster`. eksctl에 아웃바운드 인터넷 액세스 권한이 있는 경우 API 서버에 액세스할 필요가 없는 명령이 지원됩니다.

완전 프라이빗 클러스터 강제 삭제

eksctl은 클러스터의 모든 리소스에 자동으로 액세스할 수 없으므로 eksctl을 통해 전체 프라이빗 클러스터를 삭제할 때 오류가 발생할 수 있습니다.는 이를 해결하기 위해 `--force` 존재합니다. 클러스터를 강제로 삭제하고 오류가 발생하면 계속됩니다.

제한 사항

현재 구현의 제한 사항은 eksctl이 처음에 퍼블릭 및 프라이빗 엔드포인트 액세스가 모두 활성화된 클러스터를 생성하고 모든 작업이 완료된 후 퍼블릭 엔드포인트 액세스를 비활성화한다는 것입니다. 이는 eksctl이 자체 관리형 노드가 클러스터에 조인하고 GitOps 및 Fargate를 지원하도록 Kubernetes API 서버에 대한 액세스 권한이 필요하기 때문에 필요합니다. 이러한 작업이 완료되면 eksctl은 클러스터 엔드포인트 액세스를 프라이빗 전용으로 전환합니다. 이 추가 업데이트는 완전 프라이빗 클러스터를 생성하는 데 표준 클러스터보다 시간이 오래 걸린다는 의미입니다. 향후 eksctl은 이러한 API 작업을 수행하기 위해 VPC 지원 Lambda 함수로 전환할 수 있습니다.

HTTP 프록시 서버를 통한 아웃바운드 액세스

eksctl은 구성된 HTTP(S) 프록시 서버를 통해 AWS APIs와 통신할 수 있지만 프록시 제외 목록을 올바르게 설정해야 합니다.

일반적으로 값을 포함한 적절한 no_proxy 환경 변수를 설정하여 클러스터의 VPC 엔드포인트에 대한 요청이 프록시를 통해 라우팅되지 않도록 해야 합니다.eks.amazonaws.com.

프록시 서버가 "SSL 가로채기"를 수행하고 서비스 계정에 대한 IAM 역할(IRSA)을 사용하는 경우 도메인에 대해 SSL Man-in-the-Middle 명시적으로 우회해야 합니다oidc.<region>.amazonaws.com. 이렇게 하지 않으면 eksctl이 OIDC 공급자에 대한 잘못된 루트 인증서 지문을 가져오고, IAM 자격 증명을 얻을 수 없어 클러스터가 작동하지 않아 AWS VPC CNI 플러그인이 시작되지 않습니다.

추가 정보

- EKS 프라이빗 클러스터

추가 기능

이 주제에서는 eksctl을 사용하여 Amazon EKS 클러스터의 Amazon EKS 추가 기능을 관리하는 방법을 설명합니다. EKS 추가 기능은 EKS API를 통해 Kubernetes 운영 소프트웨어를 활성화하고 관리할 수 있는 기능으로, 클러스터 추가 기능의 설치, 구성 및 업데이트 프로세스를 간소화합니다.

> **Warning**
>
> 이제 eksctl은 기본 추가 기능(vpc-cni, coredns, kube-proxy)을 자체 관리형 추가 기능 대신 EKS 추가 기능으로 설치합니다. 즉, eksctl v0.184.0 이상으로 생성된 클러스터에는 `eksctl utils update-*` 명령 `eksctl update addon` 대신를 사용해야 합니다.

Cilium 및 Calico와 같은 대체 CNI 플러그인을 사용하려는 경우 기본 네트워킹 추가 기능 없이 클러스터를 생성할 수 있습니다.

이제 EKS 추가 기능은 EKS Pod Identity Associations를 통한 IAM 권한 수신을 지원하여 클러스터 외부의 AWS 서비스와 연결할 수 있습니다.

추가 기능 생성

Eksctl은 클러스터 추가 기능을 보다 유연하게 관리할 수 있습니다.

구성 파일에서 원하는 추가 기능과 (필요한 경우) 연결할 역할 또는 정책을 지정할 수 있습니다.

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

```
kind: ClusterConfig
metadata:
  name: example-cluster
  region: us-west-2

iam:
  withOIDC: true

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

`attachPolicy`, `attachPolicyARNs` 및 중 최대 하나를 지정할 수 있습니다`serviceAccountRoleARN`.

이 중 아무 것도 지정하지 않으면 모든 권장 정책이 연결된 역할로 추가 기능이 생성됩니다.

> **Note**
>
> 정책을 추가 기능에 연결하려면 클러스터가 OIDC 활성화되어 있어야 합니다. 활성화되지 않은 경우 연결된 정책은 무시됩니다.

그런 다음 클러스터 생성 프로세스 중에 이러한 추가 기능을 생성할 수 있습니다.

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

또는 구성 파일 또는 CLI 플래그를 사용하여 클러스터 생성 후 명시적으로 추가 기능을 생성합니다.

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

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

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

> **Tip**
>
> `--namespace-config` 플래그를 사용하여 기본 네임스페이스 대신 사용자 지정 네임스페이스에 추가 기능을 배포합니다.

추가 기능을 생성하는 동안 클러스터에 자체 관리형 버전의 추가 기능이 이미 있는 경우 구성 파일을 통해 `resolveConflicts` 옵션을 설정하여 잠재적 `configMap` 충돌을 해결하는 방법을 선택할 수 있습니다. 예:

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

추가 기능 생성의 경우 `resolveConflicts` 필드는 세 가지 고유한 값을 지원합니다.

- none - EKS는 값을 변경하지 않습니다. 생성이 실패할 수 있습니다.
- overwrite - EKS는 구성 변경 사항을 EKS 기본값으로 다시 덮어씁니다.
- preserve - EKS는 값을 변경하지 않습니다. 생성이 실패할 수 있습니다. (과 비슷none하지만 preserve 추가 기능 업데이트 시와 다름).

활성화된 추가 기능 나열

다음을 실행하여 클러스터에서 활성화된 추가 기능을 확인할 수 있습니다.

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

또는

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

추가 기능의 버전 설정

추가 기능의 버전 설정은 선택 사항입니다. version 필드를 비워 두면 추가 기능의 기본 버전이 확인eksctl됩니다. 특정 추가 기능의 기본 버전에 대한 자세한 내용은 EKS에 대한 AWS 설명서에서 확인할 수 있습니다. 기본 버전이 반드시 사용 가능한 최신 버전이 아닐 수도 있습니다.

추가 기능 버전을 로 설정할 수 있습니다latest. 또는 v1.7.5-eksbuild.1 또는와 같이 지정된 EKS 빌드 태그로 버전을 설정할 수 있습니다v1.7.5-eksbuild.2. v1.7.5 또는와 같은 추가 기능의 릴리스 버전으로 설정할 수도 있으며 1.7.5접eksbuild미사 태그가 검색되고 자동으로 설정됩니다.

사용 가능한 추가 기능과 해당 버전을 검색하는 방법은 아래 섹션을 참조하세요.

추가 기능 검색

다음을 실행하여 클러스터에 설치할 수 있는 추가 기능을 확인할 수 있습니다.

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

그러면 클러스터의 kubernetes 버전이 검색되고 해당 버전을 기준으로 필터링됩니다. 또는 특정 kubernetes 버전에 사용할 수 있는 추가 기능을 확인하려면 다음을 실행할 수 있습니다.

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

type, owner 및/또는를 필터링하여 추가 기능을 검색할 수도 있습니다publisher. 예를 들어 특정 소유자 및 유형에 대한 추가 기능을 보려면 다음을 실행할 수 있습니다.

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

types, owners 및 publishers 플래그는 선택 사항이며 결과를 필터링하기 위해 함께 또는 개별적으로 지정할 수 있습니다.

추가 기능에 대한 구성 스키마 검색

추가 기능 및 버전을 검색한 후 JSON 구성 스키마를 가져와 사용자 지정 옵션을 볼 수 있습니다.

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

그러면이 추가 기능에 사용할 수 있는 다양한 옵션의 JSON 스키마가 반환됩니다.

구성 값 작업

ConfigurationValues는 추가 기능을 생성하거나 업데이트하는 동안 구성 파일에 제공될 수 있습니다. JSON 및 YAML 형식만 지원됩니다.

예:

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

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

> **Note**
>
> 추가 기능 구성 값을 수정하는 경우 구성 충돌이 발생합니다.

```
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`.
```

또한 get 명령은 이제 추가 기능에 ConfigurationValues 대한 도 검색합니다. 예:

```
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
```

사용자 지정 네임스페이스 사용

추가 기능을 생성하는 동안 구성 파일에 사용자 지정 네임스페이스를 제공할 수 있습니다. 추가 기능이 생성된 후에는 네임스페이스를 업데이트할 수 없습니다.

구성 파일 사용

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

CLI 플래그 사용

또는 --namespace-config 플래그를 사용하여 사용자 지정 네임스페이스를 지정할 수 있습니다.

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

get 명령은 추가 기능의 네임스페이스 값도 검색합니다.

```
- ConfigurationValues: ""
```

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

추가 기능 업데이트

다음을 실행하여 추가 기능을 최신 버전으로 업데이트하고 연결된 정책을 변경할 수 있습니다.

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

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

> **Note**
>
> 추가 기능이 생성되면 네임스페이스 구성을 업데이트할 수 없습니다. `--namespace-config` 플래그는 추가 기능을 생성하는 동안에만 사용할 수 있습니다.

추가 기능 생성과 마찬가지로 추가 기능을 업데이트할 때 해당 추가 기능의에 이전에 적용했을 수 있는 구성 변경 사항을 완전히 제어할 수 있습니다`configMap`. 특히 보존하거나 덮어쓸 수 있습니다. 이 선택적 기능은 동일한 구성 파일 필드를 통해 사용할 수 있습니다`resolveConflicts`. 예:

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

추가 기능 업데이트의 경우 `resolveConflicts` 필드는 세 가지 고유한 값을 허용합니다.

- none - EKS는 값을 변경하지 않습니다. 업데이트가 실패할 수 있습니다.
- `overwrite` - EKS는 구성 변경 사항을 EKS 기본값으로 다시 덮어씁니다.

- `preserve` - EKS는 값을 보존합니다. 이 옵션을 선택하는 경우 프로덕션 클러스터의 추가 기능을 업데이트하기 전에 비프로덕션 클러스터의 필드 및 값 변경 사항을 테스트하는 것이 좋습니다.

추가 기능 삭제

다음을 실행하여 추가 기능을 삭제할 수 있습니다.

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

그러면 추가 기능과 연결된 모든 IAM 역할이 삭제됩니다.

클러스터를 삭제하면 추가 기능에 연결된 모든 IAM 역할도 삭제됩니다.

기본 네트워킹 추가 기능을 위한 클러스터 생성 유연성

클러스터가 생성되면 EKS는 VPC CNI, CoreDNS 및 kube-proxy를 자체 관리형 추가 기능으로 자동으로 설치합니다. Cilium 및 Calico와 같은 다른 CNI 플러그인을 사용하기 위해이 동작을 비활성화하기 위해 eksctl은 이제 기본 네트워킹 추가 기능 없이 클러스터 생성을 지원합니다. 이러한 클러스터를 생성하려면 다음과 `addonsConfig.disableDefaultAddons`같이를 설정합니다.

```
addonsConfig:
  disableDefaultAddons: true
```

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

VPC CNI가 아닌 CoreDNS 및 kube-proxy만 있는 클러스터를 생성하려면에서 추가 기능을 명시적으로 지정`addons`하고 다음과 `addonsConfig.disableDefaultAddons`같이를 설정합니다.

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

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

이 변경의 일환으로 eksctl은 이제 `addonsConfig.disableDefaultAddons`가 명시적으로 true로 설정되지 않은 경우 클러스터 생성 중에 자체 관리형 추가 기능 대신 기본 추가 기능을 EKS 추가 기능

으로 설치합니다. 따라서 eksctl v0.184.0 이상으로 생성된 클러스터의 추가 기능을 업데이트하는 데 명령을 더 이상 사용할 수 `eksctl utils update-*` 없습니다.

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

대신를 지금 사용해야 `eksctl update addon` 합니다.

자세한 내용은 Amazon EKS에서 네트워킹 추가 기능을 위한 클러스터 생성 유연성 도입을 참조하세요.

Amazon EMR에 대한 액세스 활성화

EMR이 Kubernetes API에서 작업을 수행하도록 허용하려면 해당 SLR에 필요한 RBAC 권한을 부여해야 합니다. eksctl은 EMR에 필요한 RBAC 리소스를 생성하고 역할을 EMR에 대한 SLR과 바인딩하도록 `aws-auth` ConfigMap을 업데이트하는 명령을 제공합니다.

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

EKS Fargate 지원

AWS Fargate는 컨테이너를 실행할 수 있는 Amazon ECS용 관리형 컴퓨팅 엔진입니다. Fargate에서는 서버 또는 클러스터를 관리할 필요가 없습니다.

Amazon EKS는 이제 AWS Fargate에서 포드를 시작할 수 있습니다. 따라서 포드용 인프라를 프로비저닝하거나 관리하는 방법에 대해 걱정할 필요가 없으며 AWS에서 성능이 뛰어나고 가용성이 높은 Kubernetes 애플리케이션을 더 쉽게 빌드하고 실행할 수 있습니다.

Fargate를 지원하는 클러스터 생성

다음을 사용하여 Fargate를 지원하는 클러스터를 추가할 수 있습니다.

```
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
```

이 명령은 클러스터와 Fargate 프로파일을 생성했습니다. 이 프로필에는 AWS가 Fargate에서 포드를 인스턴스화하는 데 필요한 특정 정보가 포함되어 있습니다. 예를 들면 다음과 같습니다.

- 포드 실행 역할은 포드를 실행하는 데 필요한 권한과 포드를 실행하는 네트워킹 위치(서브넷)를 정의합니다. 이를 통해 동일한 네트워킹 및 보안 권한을 여러 Fargate 포드에 적용할 수 있으며 클러스터의 기존 포드를 Fargate로 더 쉽게 마이그레이션할 수 있습니다.
- Fargate에서 실행해야 하는 포드를 정의하는 선택기입니다. 이는 namespace 및 로 구성됩니다labels.

프로파일이 지정되지 않았지만 Fargate에 대한 지원이 --fargate 기본 Fargate 프로파일과 함께 활성화된 경우. 이 프로필은 default 및 kube-system 네임스페이스를 대상으로 하므로 해당 네임스페이스의 포드는 Fargate에서 실행됩니다.

생성된 Fargate 프로필은 다음 명령을 사용하여 확인할 수 있습니다.

```
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
```

선택기에 대한 자세한 내용은 Fargate 프로파일 설계를 참조하세요.

구성 파일을 사용하여 Fargate를 지원하는 클러스터 생성

다음 구성 파일은 EC2 m5.large 인스턴스 1개와 Fargate 프로파일 2개로 구성된 노드 그룹을 모두 포함하는 EKS 클러스터를 선언합니다. default 및 kube-system 네임스페이스에 정의된 모든 포드는 Fargate에서 실행됩니다. 레이블도 있는 dev 네임스페이스의 모든 포드도 Fargate에서 실행dev=passed됩니다. 다른 모든 포드는의 노드에 예약됩니다ng-1.

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

metadata:
```

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

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

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

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

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

Fargate 프로파일 설계

각 선택기 항목에는 최대 2개의 구성 요소, 즉 네임스페이스와 키-값 페어 목록이 있습니다. 선택기 항목을 생성하려면 네임스페이스 구성 요소만 필요합니다. 선택기 항목과 일치시키려면 모든 규칙(네임스페이스, 키 값 페어)을 포드에 적용해야 합니다. 포드는 프로파일에서 실행할 선택기 항목 하나와만 일치하면 됩니다. 선택기 필드의 모든 조건과 일치하는 모든 포드는 Fargate에서 실행되도록 예약됩니다. 화이트리스트에 등록된 네임스페이스 중 하나와 일치하지 않지만 사용자가 스케줄러를 수동으로

설정한 모든 포드: fargate-scheduler 필드는 Fargate에서 실행할 권한이 없으므로 보류 중 상태로 멈춥니다.

프로필은 다음 요구 사항을 충족해야 합니다.

- 프로필당 하나의 선택기가 필수입니다.
- 각 선택기에는 네임스페이스가 포함되어야 합니다. 레이블은 선택 사항입니다.

예: Fargate에서 워크로드 예약

위에서 언급한 예제에 대해 Fargate에서 포드를 예약하려면 예를 들어 라는 네임스페이스를 생성하고 여기에 워크로드를 dev 배포할 수 있습니다.

```
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
```

마지막 `kubectl get pods` 명령의 출력에서 포드가 nginx 라는 노드에 배포된 것을 확인할 수 있습니다`fargate-ip-192-168-183-140.ap-northeast-1.compute.internal`.

Fargate 프로필 관리

Fargate에 Kubernetes 워크로드를 배포하려면 EKS에 Fargate 프로필이 필요합니다. 위 예제와 같이 클러스터를 생성할 때 `eksctl`는 기본 프로필을 생성하여 이를 처리합니다. 이미 존재하는 클러스터를 고려할 때 `eksctl create fargateprofile` 명령을 사용하여 Fargate 프로파일을 생성할 수도 있습니다.

> **Note**
>
> 이 작업은 EKS 플랫폼 버전 `eks.5` 이상에서 실행되는 클러스터에서만 지원됩니다.

> **Note**
>
> 기존가 0.11.0 `eksctl` 이전 버전으로 생성된 경우 Fargate 프로파일을 생성하기 `eksctl upgrade cluster` 전에를 실행해야 합니다.

```
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"
```

생성할 Fargate 프로필의 이름을 지정할 수도 있습니다. 이 이름은 접두사 로 시작해서는 안 됩니다`eks-`.

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

CLI 플래그와 함께이 명령을 사용하면 eksctl은 간단한 선택기를 사용하여 단일 Fargate 프로파일만 생성할 수 있습니다. 네임스페이스가 더 많은 경우와 같이 더 복잡한 선택기의 경우 eksctl은 구성 파일 사용을 지원합니다.

```
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
```

클러스터에서 기존 Fargate 프로파일을 보려면:

```
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
```

그리고 `yaml` 형식으로 보려면:

```
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
```

또는 json 형식:

```
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 프로파일은 설계상 변경할 수 없습니다. 무언가를 변경하려면 원하는 변경 사항이 있는 새 Fargate 프로필을 생성하고 다음 예제와 같이 `eksctl delete fargateprofile` 명령을 사용하여 이전 프로필을 삭제합니다.

```
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"
}
```

프로필 삭제는 최대 몇 분 정도 걸릴 수 있는 프로세스입니다. `--wait` 플래그를 지정하지 않으면 `eksctl`는 프로필이 삭제될 것으로 예상하고 AWS API 요청이 전송되는 즉시를 반환합니다. 프로필이 성공적으로 삭제될 때까지 `eksctl` 기다리려면 위 예제와 `--wait` 같이를 사용합니다.

참조 자료

- AWS Fargate
- 이제 Amazon EKS가 AWS Fargate에서 포드를 시작할 수 있습니다.

클러스터 업그레이드

`eksctl` 관리형 클러스터는 3가지 간단한 단계로 업그레이드할 수 있습니다.

1. 를 사용하여 컨트롤 플레인 버전 업그레이드 `eksctl upgrade cluster`
2. 노드 그룹 업그레이드
3. 기본 네트워킹 추가 기능을 업데이트합니다(자세한 내용은 참조the section called “기본 추가 기능 업데이트”).

클러스터 업그레이드 관련 리소스를 주의 깊게 검토합니다.

- Amazon EKS 사용 설명서의 기존 클러스터를 새 Kubernetes 버전으로 업데이트
- EKS 모범 사례 가이드의 클러스터 업그레이드 모범 사례

> **Note**
>
> 이전 `eksctl update cluster`는 더 이상 사용되지 않습니다. 대신 `eksctl upgrade cluster`을 사용하세요.

컨트롤 플레인 버전 업데이트

컨트롤 플레인 버전 업그레이드는 한 번에 하나의 마이너 버전에 대해 수행해야 합니다.

컨트롤 플레인을 사용 가능한 다음 버전으로 업그레이드하려면 다음을 실행합니다.

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

이 명령은 변경 사항을 즉시 적용하지 않으므로 변경 사항을 적용`--approve`하려면를 사용하여 다시 실행해야 합니다.

클러스터 업그레이드의 대상 버전은 CLI 플래그를 사용하여 지정할 수 있습니다.

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

또는 config 파일 사용

```
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` 및 `metadata.version` 인수에 허용되는 유일한 값은 클러스터의 현재 버전 또는 한 버전 이상입니다. 둘 이상의 Kubernetes 버전 업그레이드는 지원되지 않습니다.

기본 추가 기능 업데이트

이 주제에서는 EKS 클러스터에 포함된 사전 설치된 기본 추가 기능을 업데이트하는 방법을 설명합니다.

> **Warning**
>
> 이제 eksctl은 기본 추가 기능을 자체 관리형 추가 기능 대신 EKS 추가 기능으로 설치합니다. 기본 네트워킹 추가 기능의 클러스터 생성 유연성에 미치는 영향에 대해 자세히 알아보세요. 추가 기능 업데이트에는 eksctl v0.184.0 이상으로 생성된 클러스터에를 사용할 수 `eksctl utils update-<addon>` 없습니다. 이 가이드는이 변경 전에 생성된 클러스터에만 유효합니다.

각 EKS 클러스터에는 다음과 같은 3가지 기본 추가 기능이 포함됩니다.

- kube-proxy
- aws-node
- coredns

사전 설치된 추가 기능 업데이트

클러스터를 통해 eksctl create addons 또는 클러스터 생성 시 수동으로 생성되는 공식 EKS 추가 기능의 경우 이를 관리하는 방법은를 통해 이루어집니다eksctl create/get/update/delete addon. 이 경우 EKS 추가 기능에 대한 문서를 참조하세요.

각 명령을 업데이트하는 프로세스는 다르므로 실행해야 하는 3가지 명령이 있습니다. 다음 모든 명령은를 허용합니다--config-file. 기본적으로 이러한 각 명령은 계획 모드에서 실행되며 제안된 변경 사항에 만족하는 경우 로 다시 실행합니다--approve.

를 업데이트하려면 다음을 kube-proxy실행합니다.

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

를 업데이트하려면 다음을 aws-node실행합니다.

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

를 업데이트하려면 다음을 coredns실행합니다.

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

업그레이드한 후에는를 실행kubectl get pods -n kube-system하고 모든 추가 기능 포드가 준비 상태인지 확인해야 합니다. 다음과 같이 표시됩니다.

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

EKS 클러스터에서 영역 전환 지원

이제 EKS는 다중 AZ 클러스터 환경의 복원력을 향상시키는 Amazon Application Recovery Controller(ARC) 영역 전환 및 영역 자동 전환을 지원합니다. AWS 영역 전환을 사용하면 클러스터 내 트래픽을 손상된 가용 영역에서 다른 곳으로 전환하여 새로운 Kubernetes 포드 및 노드가 정상 가용 영역에서만 시작되도록 할 수 있습니다.

영역 전환이 활성화된 클러스터 생성

```
# 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
```

기존 클러스터에서 영역 전환 활성화

기존 클러스터에서 영역 전환을 활성화하거나 비활성화하려면를 실행합니다.

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

또는 구성 파일 없음:

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

추가 정보

- EKS 영역 전환

Karpenter 지원

`eksctl`는 새로 생성된 클러스터에 Karpenter를 추가할 수 있도록 지원합니다. Helm을 사용하여 Karpenter 자체를 설치하는 등 Karpenter의 시작하기 섹션에 설명된 모든 필수 사전 조건을 생성합니다. 현재 버전 설치를 지원합니다`0.28.0+`. 자세한 내용은 Karpenter 호환성 섹션을 참조하세요.

다음 클러스터 구성은 일반적인 Karpenter 설치를 간략하게 설명합니다.

```
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
```

버전은 Helm 리포지토리에서 찾을 수 있는 Karpenter의 버전입니다. 다음 옵션도 설정할 수 있습니다.

```
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
```

Karpenter를 설치하려면 OIDC를 정의해야 합니다.

Karpenter가 성공적으로 설치되면 NodePool(s) 및 NodeClass(es)를 추가하여 Karpenter가 클러스터에 노드 추가를 시작할 수 있도록 합니다.

NodePool의 nodeClassRef 섹션은의 이름과 일치해야 합니다EC2NodeClass. 예제:

```
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
```

`instanceProfile` 휴지통 노드에는 `role` 또는 중 하나를 지정해야 합니다. `instanceProfile`에서 생성한 프로필의 이름을 사용하도록 선택한 경우 패턴은 다음과 `eksctl` 같습니다`eksctl-KarpenterNodeInstanceProfile-<cluster-name>`.

자동 보안 그룹 태그 지정

`eksctl` Karpenter가 활성화(`karpenter.version` 지정)되어 있고 태그가에 있는 `karpenter.sh/discovery` 경우가 클러스터의 공유 노드 보안 그룹에 자동으로 `karpenter.sh/discovery` 태그를 지정합니다`metadata.tags`. 이렇게 하면 AWS Load Balancer Controller 호환성이 활성화됩니다.

참고로 karpenter 0.32.0 이상에서는 Provisioners가 더 이상 사용되지 않으며 NodePool로 대체되었습니다.

클러스터 구성 스키마

> **Note**
>
> 스키마의 위치가 현재 마이그레이션 중입니다.

yaml 파일을 사용하여 클러스터를 생성할 수 있습니다. 스키마 참조를 봅니다.

예제:

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

이 파일의 스키마 참조는 GitHub에서 사용할 수 있습니다.

파일 사용에 대한 자세한 내용은 섹션을 참조하세요the section called “클러스터 생성 및 관리”.

노드 그룹

이 장에는 Eksctl을 사용하여 노드 그룹을 생성하고 구성하는 방법에 대한 정보가 포함되어 있습니다. 노드 그룹은 EKS 클러스터에 연결된 EC2 인스턴스의 그룹입니다.

주제:

- the section called “스팟 인스턴스”
  - 관리형 노드 그룹을 사용하여 스팟 인스턴스로 EKS 클러스터 생성 및 관리
  - MixedInstancesPolicy를 사용하여 비관리형 노드 그룹에 대한 스팟 인스턴스 구성
  - `node-lifecycle` Kubernetes 레이블을 사용하여 스팟 및 온디맨드 인스턴스 구분
- the section called “Auto Scaling”
  - 클러스터 오토스케일러 사용을 허용하는 IAM 역할이 있는 클러스터 또는 노드 그룹을 생성하여 Kubernetes 클러스터 노드의 자동 조정 활성화
  - 클러스터 오토스케일러가 노드 그룹을 조정하는 데 필요한 태그와 주석을 포함하도록 노드 그룹 정의를 구성합니다.
  - 워크로드에 영역별 스토리지 또는 선호도 규칙과 같은 영역별 요구 사항이 있는 경우 각 가용 영역에 대해 별도의 노드 그룹을 생성합니다.
- the section called “EKS 관리형 노드 그룹”
  - Amazon EKS Kubernetes 클러스터용 EC2 인스턴스(노드) 프로비저닝 및 관리
  - 버그 수정, 보안 패치 및 업데이트 노드를 최신 Kubernetes 버전으로 쉽게 적용
- the section called “EKS 하이브리드 노드”
  - AWS 클라우드에서 사용되는 것과 동일한 AWS EKS 클러스터, 기능 및 도구를 사용하여 고객 관리형 인프라에서 온프레미스 및 엣지 애플리케이션 실행 활성화
  - AWS Site-to-Site VPN 또는 AWS Direct Connect와 같은 옵션을 사용하여 온프레미스 네트워크를 AWS VPC에 연결하도록 네트워킹 구성
  - AWS Systems Manager(SSM) 또는 AWS IAM Roles Anywhere를 사용하여 EKS 클러스터로 인증할 원격 노드의 자격 증명을 설정합니다.
- the section called “노드 복구 구성”
  - 비정상 작업자 노드를 자동으로 모니터링 및 교체하거나 재부팅하도록 EKS 관리형 노드 그룹에 대한 노드 복구 활성화
- the section called “ARM 지원”

  - 성능 및 비용 효율성을 개선하기 위해 ARM 기반 Graviton 인스턴스로 EKS 클러스터 생성
- the section called “테인트”
  - Kubernetes 클러스터의 특정 노드 그룹에 테인트 적용
  - 테인트 키, 값 및 효과를 기반으로 포드의 예약 및 제거 제어
- the section called “시작 템플릿 지원”
  - 제공된 EC2 시작 템플릿을 사용하여 관리형 노드 그룹 시작
  - 다른 버전의 시작 템플릿을 사용하도록 관리형 노드 그룹 업그레이드
  - 관리형 노드 그룹에서 사용자 지정 AMIs 및 시작 템플릿을 사용할 때의 제한 사항 및 고려 사항 이해
- the section called “노드 그룹 작업”
  - 노드 그룹의 EC2 인스턴스에 대한 SSH 액세스 활성화
  - 노드 그룹의 노드 수를 늘리거나 줄입니다.
- the section called “사용자 지정 서브넷”
  - 새 서브넷으로 기존 VPC를 확장하고 해당 서브넷에 노드 그룹 추가
- the section called “노드 부트스트래핑”
  - AmazonLinux2023에 도입된 새로운 노드 초기화 프로세스(nodeadm) 이해
  - 자체 관리형 및 EKS 관리형 노드에 대해 eksctl에서 적용하는 기본 NodeConfig 설정에 대해 알아봅니다.
  - 사용자 지정 NodeConfig와 함께 overrideBootstrapCommand를 제공하여 노드 부트스트래핑 프로세스 사용자 지정
- the section called “비관리형 노드 그룹”
  - EKS 클러스터에서 비관리형 노드 그룹 생성 또는 업데이트
  - kube-proxy, aws-node 및 CoreDNS와 같은 기본 Kubernetes 추가 기능 업데이트
- the section called “GPU 지원”
  - Eksctl은 노드 그룹에 GPU 인스턴스 유형을 선택하여 EKS 클러스터에서 GPU 가속 워크로드를 사용할 수 있도록 지원합니다.
  - Eksctl은 GPU 지원 인스턴스 유형을 선택하면 NVIDIA Kubernetes 디바이스 플러그인을 자동으로 설치하므로 클러스터에서 GPU 리소스를 쉽게 사용할 수 있습니다.
  - 사용자는 제공된 명령을 사용하여 자동 플러그인 설치를 비활성화하고 특정 버전의 NVIDIA Kubernetes 디바이스 플러그인을 수동으로 설치할 수 있습니다.
- the section called “인스턴스 선택기”

  - vCPUs, 메모리, GPUs 및 CPU 아키텍처와 같은 리소스 기준에 따라 적합한 EC2 인스턴스 유형 목록을 자동으로 생성
  - 지정된 인스턴스 선택기 기준과 일치하는 인스턴스 유형을 사용하여 클러스터 및 노드 그룹 생성
  - 노드 그룹을 생성하기 전에 모의 실행을 수행하여 인스턴스 선택기와 일치하는 인스턴스 유형을 검사하고 수정합니다.
- the section called “추가 볼륨 매핑”
  - EKS 클러스터에서 관리형 노드 그룹에 대한 추가 볼륨 매핑 구성
  - 추가 볼륨의 크기, 유형, 암호화, IOPS 및 처리량과 같은 볼륨 속성 사용자 지정
  - 기존 EBS 스냅샷을 노드 그룹에 추가 볼륨으로 연결
- the section called “Windows 작업자 노드”
  - 기존 Linux Kubernetes 클러스터에 Windows 노드 그룹을 추가하여 Windows 워크로드 실행 활성화
  - `kubernetes.io/os` 및 `kubernetes.io/arch` 레이블을 기반으로 노드 선택기를 사용하여 적절한 운영 체제(Windows 또는 Linux)에서 워크로드 예약
- the section called “사용자 지정 AMI 지원”
  - `--node-ami` 플래그를 사용하여 노드 그룹에 대한 사용자 지정 AMI를 지정하거나, AWS에서 최신 EKS 최적화 AMI를 쿼리하거나, AWS Systems Manager Parameter Store를 사용하여 AMI를 찾습니다.
  - `--node-ami-family` 플래그를 설정하여 AmazonLinux2, Ubuntu2204 또는 WindowsServer2022CoreContainer와 같은 노드 그룹 AMI의 운영 체제 패밀리를 지정합니다.
  - Windows 노드 그룹의 경우 사용자 지정 AMI를 지정하고를 통해 PowerShell 부트스트랩 스크립트를 제공합니다`overrideBootstrapCommand`.
- the section called “사용자 지정 DNS”
  - 내부 및 외부 DNS 조회에 사용되는 DNS 서버 IP 주소 덮어쓰기

노드 그룹 작업

노드 그룹 생성

클러스터와 함께 생성된 초기 노드 그룹 외에 하나 이상의 노드 그룹을 추가할 수 있습니다.

추가 노드 그룹을 생성하려면 다음을 사용합니다.

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

> **Note**
>
> `--version` 관리형 노드 그룹에는 플래그가 지원되지 않습니다. 항상 컨트롤 플레인에서 버전을 상속합니다.

기본적으로 새 비관리형 노드 그룹은 컨트롤 플레인(`--version=auto`)에서 버전을 상속하지만 다른 버전을 지정할 수 있습니다. 또한 `--version=latest`를 사용하여 최신 버전을 강제로 사용할 수도 있습니다.

또한에 사용된 것과 동일한 구성 파일을 사용할 수 있습니다`eksctl create cluster`.

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

구성 파일에서 노드 그룹 생성

노드 그룹은 클러스터 정의 또는 구성 파일을 통해 생성할 수도 있습니다. 다음 예제 구성 파일과 라는 기존 클러스터를 고려할 때`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
```

노드 그룹 ng-1-workers 및는 다음 명령을 사용하여 생성할 ng-2-builders 수 있습니다.

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

로드 밸런싱

기존 클래식 로드 밸런서 또는 대상 그룹을 노드 그룹에 연결하기 위해 이미 준비한 경우 구성 파일에서 이를 지정할 수 있습니다. 클래식 로드 밸런서 또는 대상 그룹은 노드 그룹을 생성할 때 ASG와 자동으로 연결됩니다. 이는 nodeGroups 필드를 통해 정의된 자체 관리형 노드 그룹에 대해서만 지원됩니다.

```
# 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
```

구성 파일의 노드 그룹 선택

구성 파일에 지정된 노드 그룹의 하위 집합에서만 `create` 또는 `delete` 작업을 수행하려면 globs 및 목록을 허용하는 CLI 플래그가 두 개 있습니다. `0` `1`예:

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

위의 예제 구성 파일을 사용하면 다음 명령을 사용하여 작업자를 제외한 모든 작업자 노드 그룹을 생성할 수 있습니다.

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

또는 다음을 사용하여 빌더 노드 그룹을 삭제할 수 있습니다.

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

이 경우 노드 그룹을 실제로 삭제하는 `--approve` 명령도 제공해야 합니다.

규칙 포함 및 제외

- `--include` 또는 `--exclude`가 지정되지 않은 경우 모든 항목이 포함됩니다.
- 만 `--include` 지정된 경우 해당 globs와 일치하는 노드 그룹만 포함됩니다.
- 만 `--exclude` 지정된 경우 해당 globs와 일치하지 않는 모든 노드 그룹이 포함됩니다.
- 둘 다 지정하면 `--exclude` 규칙이 우선합니다`--include`(즉, 두 그룹의 규칙과 일치하는 노드 그룹은 제외됨).

노드 그룹 나열

노드 그룹 또는 모든 노드 그룹에 대한 세부 정보를 나열하려면 다음을 사용합니다.

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

기본 로그 테이블보다 더 많은 정보를 출력하는 YAML 또는 JSON 형식으로 하나 이상의 노드 그룹을 나열하려면 다음을 사용합니다.

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

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

노드 그룹 불변성

설계상 노드 그룹은 변경할 수 없습니다. 즉, AMI 또는 노드 그룹의 인스턴스 유형과 같은 항목(스케일링 제외)을 변경해야 하는 경우 원하는 변경 사항이 있는 새 노드 그룹을 생성하고, 로드를 이동하고, 이전 노드 그룹을 삭제해야 합니다. 노드 그룹 삭제 및 드레이닝 섹션을 참조하세요.

노드 그룹 조정

노드 그룹 조정은 최대 몇 분 정도 걸릴 수 있는 프로세스입니다. `--wait` 플래그를 지정하지 않으면 `eksctl`는 노드 그룹의 규모가 조정될 것으로 예상하고 AWS API 요청이 전송되는 즉시를 반환합니다. 노드를 사용할 수 있을 때까지 `eksctl` 기다리려면 아래 예제와 같은 `--wait` 플래그를 추가합니다.

> **Note**
>
> ASG 변경에만 의존하므로 노드 그룹을 축소/축소(즉, 노드 수 감소)하면 오류가 발생할 수 있습니다. 즉, 제거/종료되는 노드(들)가 명시적으로 드레이닝되지 않습니다. 이는 향후 개선해야 할 영역일 수 있습니다.

관리형 노드 그룹 구성을 업데이트하는 EKS API를 직접 호출하여 관리형 노드 그룹을 확장할 수 있습니다.

단일 노드 그룹 조정

다음 `eksctl scale nodegroup` 명령을 사용하여 노드 그룹을 확장할 수 있습니다.

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

예를 들어 노드 그룹을 ng-a345f4e1에서 5개의 노드cluster-1로 확장하려면 다음을 실행합니다.

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

에 전달된 구성 파일을 사용하고 로 확장해야 하는 노드 그룹의 이름을 --config-file 지정하여 노드 그룹을 확장할 수도 있습니다--name. Eksctl은 구성 파일을 검색하고 해당 노드 그룹과 조정 구성 값을 검색합니다.

원하는 노드 수가 현재 최소 및 현재 최대 노드 수 범위 NOT 내에 있는 경우 특정 오류 하나가 표시됩니다. 이러한 값은 각각 플래그 --nodes-min 및와 함께 전달할 수도 --nodes-max 있습니다.

여러 노드 그룹 조정

Eksctl은와 함께 전달되는 구성 파일에 있는 모든 노드 그룹을 검색하고 확장할 수 있습니다--config-file.

단일 노드 그룹을 조정하는 것과 마찬가지로 각 노드 그룹에도 동일한 검증 세트가 적용됩니다. 예를 들어 원하는 노드 수는 최소 및 최대 노드 수 범위 내에 있어야 합니다.

노드 그룹 삭제 및 드레이닝

노드 그룹을 삭제하려면 다음을 실행합니다.

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

포함 및 제외 규칙은이 명령과 함께 사용할 수도 있습니다.

> **Note**
>
> 그러면 인스턴스가 삭제되기 전에 해당 노드 그룹에서 모든 포드가 드레이닝됩니다.

드레이닝 프로세스 중에 제거 규칙을 건너뛰려면 다음을 실행합니다.

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

모든 노드가 연결되고 삭제 시 노드 그룹에서 모든 포드가 제거되지만 노드 그룹을 삭제하지 않고 드레이닝해야 하는 경우 다음을 실행합니다.

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

노드 그룹의 코딩을 해제하려면 다음을 실행합니다.

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

PodDisruptionBudget 설정과 같은 제거 규칙을 무시하려면 다음을 실행합니다.

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

드레이닝 프로세스의 속도를 높이려면 병렬`--parallel <value>`로 드레이닝할 노드 수를 지정할 수 있습니다.

기타 기능

노드 그룹에 대해 SSH, ASG 액세스 및 기타 기능을 활성화할 수도 있습니다. 예:

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

레이블 업데이트

에는 노드 그룹의 레이블을 업데이트`eksctl`하는 특정 명령이 없지만 `kubectl`를 사용하여 쉽게 수행할 수 있습니다. 예:

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

SSH 액세스

노드 그룹 구성`publicKeyPath`에서 `publicKey` `publicKeyName` 및 중 하나를 구성하여 노드 그룹에 대한 SSH 액세스를 활성화할 수 있습니다. 또는 `enableSsm`로 노드 그룹을 구성하여 AWS Systems Manager(SSM)를 사용하여 노드에 SSH를 적용할 수 있습니다.

```
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
```

비관리형 노드 그룹

에서 nodeGroups 필드를 eksctl설정--managed=false하거나 사용하면 비관리형 노드 그룹이 생성됩니다. 비관리형 노드 그룹은 일반적으로 EKS 관리형 노드 그룹에 대해서만 알고 있는 EKS 콘솔에 표시되지 않습니다.

를 실행한 후에만 노드 그룹을 업그레이드해야 합니다eksctl upgrade cluster. (클러스터 업그레이드 참조)

초기 노드 그룹(예: 로 생성eksctl create cluster)만 있는 간단한 클러스터가 있는 경우 프로세스는 매우 간단합니다.

1. 이전 노드 그룹의 이름을 가져옵니다.

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

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

2. 새 노드 그룹 생성:

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

3. 이전 노드 그룹을 삭제합니다.

```
 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.
> ```

여러 노드 그룹 업데이트

노드 그룹이 여러 개 있는 경우 각 노드 그룹이 구성된 방식을 추적하는 것은 사용자의 책임입니다. 구성 파일을 사용하여이 작업을 수행할 수 있지만 아직 사용하지 않은 경우 클러스터를 검사하여 각 노드 그룹이 어떻게 구성되었는지 확인해야 합니다.

일반적으로 다음을 찾고 있습니다.

- 보유한 노드 그룹과 삭제할 수 있거나 새 버전으로 교체해야 하는 노드 그룹 검토
- 각 노드 그룹의 구성을 기록해 둡니다. 다음에 쉽게 업그레이드하려면 구성 파일을 사용하는 것이 좋습니다.

구성 파일로 업데이트

구성 파일을 사용하는 경우 다음을 수행해야 합니다.

구성 파일을 편집하여 새 노드 그룹을 추가하고 이전 노드 그룹을 제거합니다. 노드 그룹을 업그레이드하고 동일한 구성을 유지하려는 경우 이름-v2에를 추가하는 등 노드 그룹 이름을 변경할 수 있습니다.

구성 파일에 정의된 새 노드 그룹을 모두 생성하려면 다음을 실행합니다.

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

새 노드 그룹이 있으면 이전 노드 그룹을 삭제할 수 있습니다.

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

> **Note**
>
> 첫 번째 실행은 계획 모드입니다. 제안된 변경 사항에 만족하는 경우 로 다시 실행합니다`--approve`.

기본 추가 기능 업데이트

클러스터에 설치된 네트워킹 추가 기능을 업데이트해야 할 수 있습니다. 자세한 내용은 the section called “기본 추가 기능 업데이트” 단원을 참조하십시오.

EKS 관리형 노드 그룹

Amazon EKS 관리형 노드 그룹은 Amazon EKS Kubernetes 클러스터에 대한 노드(EC2 인스턴스)의 프로비저닝 및 수명 주기 관리를 자동화하는 기능입니다. 고객은 클러스터에 최적화된 노드 그룹을 프로비저닝할 수 있으며 EKS는 최신 Kubernetes 및 호스트 OS 버전으로 노드를 최신 상태로 유지합니다.

EKS 관리형 노드 그룹은 Amazon EKS 클러스터에 대해 AWS에서 관리하는 오토 스케일링 그룹 및 관련 EC2 인스턴스입니다. 각 노드 그룹은 Amazon EKS 최적화 Amazon Linux 2 AMI를 사용합니다. Amazon EKS를 사용하면 노드에 버그 수정 및 보안 패치를 쉽게 적용하고 최신 Kubernetes 버전으로 업데이트할 수 있습니다. 각 노드 그룹은 고가용성을 위해 여러 AWS VPC 가용 영역 및 서브넷에 걸쳐 있을 수 있는 클러스터의 Auto Scaling 그룹을 시작합니다.

관리형 노드 그룹에 대한 새로운 시작 템플릿 지원 ???

> **Note**
>
> "비관리형 노드 그룹"이라는 용어는 eksctl이 처음부터 지원한 노드 그룹을 참조하는 데 사용되었습니다(`nodeGroups` 필드를 통해 표시됨). `ClusterConfig` 파일은 비관리형 노드 그룹을 정의하기 위해 `nodeGroups` 필드를 계속 사용하며 관리형 노드 그룹은 `managedNodeGroups` 필드로 정의됩니다.

관리형 노드 그룹 생성

```
$ eksctl create nodegroup
```

새 클러스터

관리형 노드 그룹을 사용하여 새 클러스터를 생성하려면를 실행합니다.

```
eksctl create cluster
```

여러 관리형 노드 그룹을 생성하고 구성을 더 잘 제어하려면 구성 파일을 사용할 수 있습니다.

> **Note**
>
> 관리형 노드 그룹에는 비관리형 노드 그룹과의 완전한 기능 패리티가 없습니다.

```
# 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
```

관리형 노드 그룹을 생성하기 위한 구성 파일의 또 다른 예는 여기에서 찾을 수 있습니다.

관리형 노드 그룹과 비관리형 노드 그룹이 모두 있는 클러스터를 가질 수 있습니다. 비관리형 노드 그룹은 AWS EKS 콘솔에 표시되지 않지만 두 가지 유형의 노드 그룹을 모두 `eksctl get nodegroup` 나열합니다.

```
# 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
```

신규 사용자 지정 AMI, 보안 그룹, instancePrefix, , instanceName, ebsOptimized, volumeType, volumeName, volumeKmsKeyID, maxPodsPerNode, volumeIOPSpreBootstrapCommands, 및 volumeEncrypted지원 overrideBootstrapCommand 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'
```

한 영역에서만 사용할 수 있는 인스턴스 유형을 요청하는 경우(eksctl 구성에 2개 사양 필요) 노드 그룹 요청에 가용 영역을 추가해야 합니다.

```
# 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
```

이는 한 영역에서만 사용할 수 있는 Hpc6 패밀리와 같은 인스턴스 유형에 해당될 수 있습니다.

기존 클러스터

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

팁: `ClusterConfig` 파일을 사용하여 전체 클러스터를 설명하는 경우 `managedNodeGroups` 필드에 새 관리형 노드 그룹을 설명하고 다음을 실행합니다.

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

관리형 노드 그룹 업그레이드

노드 그룹을 사용 중인 AMI 유형에 대한 최신 EKS 최적화 AMI 릴리스 버전으로 언제든지 업데이트할 수 있습니다.

노드 그룹이 클러스터와 동일한 Kubernetes 버전인 경우 사용 중인 AMI 유형의 해당 Kubernetes 버전에 대한 최신 AMI 릴리스 버전으로 업데이트할 수 있습니다. 노드 그룹이 클러스터의 Kubernetes 버전에서 이전 Kubernetes 버전인 경우 노드 그룹을 노드 그룹의 Kubernetes 버전과 일치하는 최신 AMI 릴리스 버전으로 업데이트하거나 클러스터 Kubernetes 버전과 일치하는 최신 AMI 릴리스 버전으로 업데이트할 수 있습니다. 노드 그룹을 이전 Kubernetes 버전으로 롤백할 수 없습니다.

관리형 노드 그룹을 최신 AMI 릴리스 버전으로 업그레이드하려면:

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

노드 그룹은 다음을 사용하여 지정된 Kubernetes 버전의 최신 AMI 릴리스로 업그레이드할 수 있습니다.

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

최신 버전 대신 특정 AMI 릴리스 버전으로 업그레이드하려면를 전달합니다`--release-version`.

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

> **Note**
>
> 관리형 노드가 사용자 지정 AMIs를 사용하여 배포되는 경우 사용자 지정 AMI의 새 버전을 배포하려면 다음 워크플로를 따라야 합니다.

- 노드 그룹의 초기 배포는 시작 템플릿을 사용하여 수행해야 합니다. 예:

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

- 사용자 지정 AMI의 새 버전을 생성합니다(AWS EKS 콘솔 사용).
- 새 AMI ID를 사용하여 새 시작 템플릿 버전을 생성합니다(AWS EKS 콘솔 사용).
- 노드를 시작 템플릿의 새 버전으로 업그레이드합니다. 예:

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

노드에 대한 병렬 업그레이드 처리

여러 관리형 노드를 동시에 업그레이드할 수 있습니다. 병렬 업그레이드를 구성하려면 `updateConfig` 노드 그룹을 생성할 때 노드 그룹의를 정의합니다. 예제는 여기에서 찾을 `updateConfig` 수 있습니다.

한 번에 여러 노드를 업그레이드하여 워크로드가 가동 중지되지 않도록 하려면의 `maxUnavailable` 필드에 이를 지정하여 업그레이드 중에 사용할 수 없게 될 수 있는 노드 수를 제한할 수 있습니다`updateConfig`. 또는 사용할 수 없는 최대 노드 수를 총 노드 수의 백분율로 `maxUnavailablePercentage`정의하는를 사용합니다.

는 보다 클 수 `maxUnavailable` 없습니다`maxSize`. 또한 `maxUnavailable` 및는 동시에 사용할 수 `maxUnavailablePercentage` 없습니다.

이 기능은 관리형 노드에서만 사용할 수 있습니다.

관리형 노드 그룹 업데이트

eksctl를 사용하면 관리형 노드 그룹의 UpdateConfig 섹션을 업데이트할 수 있습니다. 이 섹션에서는 MaxUnavailable 및의 두 필드를 정의합니다MaxUnavailablePercentage. 노드 그룹은 업데이트 중에 영향을 받지 않으므로 가동 중지 시간이 예상되지 않습니다.

명령은 --config-file 플래그를 사용하는 구성 파일과 함께 사용해야 update nodegroup 합니다. 노드 그룹에는 nodeGroup.updateConfig 섹션이 포함되어야 합니다. 자세한 내용은 여기에서 확인할 수 있습니다.

노드 그룹 상태 문제

EKS 관리형 노드 그룹은 노드 그룹 및 노드의 구성에서 상태 문제를 자동으로 확인하고 EKS API 및 콘솔을 통해 보고합니다. 노드 그룹의 상태 문제를 보려면:

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

레이블 관리

EKS 관리형 노드 그룹은 노드 그룹의 Kubernetes 노드에 적용되는 레이블 연결을 지원합니다. 클러스터 또는 노드 그룹 생성 중에 eksctl의 labels 필드를 통해 지정됩니다.

노드 그룹에서 새 레이블을 설정하거나 기존 레이블을 업데이트하려면:

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

노드 그룹에서 레이블을 설정 해제하거나 제거하려면:

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

노드 그룹에 설정된 모든 레이블을 보려면:

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

관리형 노드 그룹 조정

eksctl scale nodegroup는 관리형 노드 그룹도 지원합니다. 관리형 또는 비관리형 노드 그룹을 조정하기 위한 구문은 동일합니다.

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

추가 정보

- EKS 관리형 노드 그룹

노드 부트스트래핑

AmazonLinux2023

AL2023은 YAML 구성 스키마를 사용하는 새로운 노드 초기화 프로세스 nodeadm을 도입하여 `/etc/eks/bootstrap.sh` 스크립트 사용을 중단했습니다.

> **Note**
>
> Kubernetes 버전 1.30 이상에서는 Amazon Linux 2023이 기본 OS입니다.

AL2의 기본 설정

사용자 지정 AMIs를 기반으로 하는 자체 관리형 노드 및 EKS 관리형 노드의 경우는 기본, 최소를 `eksctl` 생성하고 노드 그룹의 시작 템플릿 사용자 데이터에 `NodeConfig` 자동으로 주입합니다. 예:

```
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

--//--
```

네이티브 AMIs를 기반으로 하는 EKS 관리형 노드의 경우 EKS MNGNodeConfig가 후드 아래에 기본값을 추가하고 EC2의 사용자 데이터에 직접 추가합니다. 따라서이 시나리오에서는 eksctl가 시작 템플릿에 포함할 필요가 없습니다.

부트스트래핑 프로세스 구성

의 고급 속성을 설정NodeConfig하거나 기본값을 재정의하기 위해 eksctl을 사용하면 nodeGroup.overrideBootstrapCommand 또는 managedNodeGroup.overrideBootstrapCommand를 NodeConfig 통해 사용자 지정을 지정할 수 있습니다.

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

이 사용자 지정 구성은 eksctl에서 사용자 데이터에 추가되고에서 기본 구성nodeadm과 병합됩니다. 여기에서 여러 구성 객체를 병합하는 nodeadm의 기능에 대해 자세히 알아보세요.

관리형 노드 그룹에 대한 시작 템플릿 지원

eksctl은 제공된 EC2 시작 템플릿을 사용하여 관리형 노드 그룹 시작을 지원합니다. 이를 통해 사용자 지정 AMIs 및 보안 그룹 제공, 노드 부트스트래핑을 위한 사용자 데이터 전달 등 노드 그룹에 대한 여러 사용자 지정 옵션을 사용할 수 있습니다.

제공된 시작 템플릿을 사용하여 관리형 노드 그룹 생성

```
# 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
```

다른 시작 템플릿 버전을 사용하도록 관리형 노드 그룹 업그레이드

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

> **Note**
>
> 시작 템플릿이 사용자 지정 AMI를 사용하는 경우 새 버전도 사용자 지정 AMI를 사용해야 합니다. 그렇지 않으면 업그레이드 작업이 실패합니다.

시작 템플릿이 사용자 지정 AMI를 사용하지 않는 경우 업그레이드할 Kubernetes 버전도 지정할 수 있습니다.

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

사용자 지정 AMI 및 시작 템플릿 지원에 대한 참고 사항

- 시작 템플릿이 제공되면 , `instanceType`, , , , , `instancePrefix`, `ami` `ssh.allow` `ssh.sourceSecurityGroupIds` `securityGroups``instanceName`, `ebsOptimized`, `volumeEncrypted`, `volumeKmsKeyID`, `volumeIOPS`, `maxPodsPerNode`, 및 필드는 지원되지 않습니다`preBootstrapCommands``overrideBootstrapCommand``disableIMDSv1`.
- 사용자 지정 AMI(`ami`)를 사용하는 경우 부트스트래핑을 수행하도록 도 설정해야 `overrideBootstrapCommand` 합니다.
- `overrideBootstrapCommand`는 사용자 지정 AMI를 사용할 때만 설정할 수 있습니다.
- 시작 템플릿이 제공되면 노드 그룹 구성에 지정된 태그는 EKS 노드 그룹 리소스에만 적용되며 EC2 인스턴스에 전파되지 않습니다.

사용자 지정 서브넷

새 서브넷으로 기존 VPC를 확장하고 해당 서브넷에 노드 그룹을 추가할 수 있습니다.

이유

클러스터에 사전 구성된 IPs 부족한 경우 새 CIDR로 기존 VPC의 크기를 조정하여 새 서브넷을 추가할 수 있습니다. 이를 수행하는 방법을 알아보려면 AWS Extending VPCs.

TL;DR

VPC의 구성으로 이동하여 작업->CIDRs 편집을 클릭하고 새 범위를 추가합니다. 예제:

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

이제 새 서브넷을 추가해야 합니다. 새 프라이빗 서브넷인지 퍼블릭 서브넷인지에 따라 각각 프라이빗 서브넷 또는 퍼블릭 서브넷에서 라우팅 정보를 복사해야 합니다.

서브넷이 생성되면 라우팅을 추가하고 VPC의 다른 서브넷에서 NAT 게이트웨이 ID 또는 인터넷 게이트웨이를 복사합니다. 퍼블릭 서브넷인 경우 자동 IP 할당을 활성화합니다. 작업->IP 설정 자동 할당 수정->퍼블릭 IPv4 주소 자동 할당 활성화.

퍼블릭 또는 프라이빗 서브넷 구성에 따라 기존 서브넷의 TAGS도 복사해야 합니다. 이는 중요합니다. 그렇지 않으면 서브넷이 클러스터의 일부가 되지 않고 서브넷의 인스턴스가 조인할 수 없게 됩니다.

완료되면 새 서브넷의 ID를 복사합니다. 필요한 만큼 자주 반복합니다.

수정할 수 있다면 방법이 무엇입니까?

생성된 서브넷(들)에서 노드 그룹을 생성하려면 다음 명령을 실행합니다.

```
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
```

또는 다음과 같이 구성을 사용합니다.

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

다음과 같은 구성을 사용합니다.

```
# 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
```

노드 그룹이 생성될 때까지 기다리면 새 인스턴스에 서브넷(들)의 새 IP 범위가 있어야 합니다.

클러스터 삭제

새 추가는 CloudFormation 스택 외부에 종속성을 추가하여 기존 VPC를 수정했으므로 CloudFormation은 더 이상 클러스터를 제거할 수 없습니다.

클러스터를 삭제하기 전에 생성된 추가 서브넷을 모두 수동으로 제거한 다음를 호출하여 진행합니다`eksctl`.

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

사용자 지정 DNS

모든 내부 및 외부 DNS 조회에 사용되는 DNS 서버 IP 주소를 덮어쓰는 두 가지 방법이 있습니다. 이는에 대한 `--cluster-dns` 플래그와 동일합니다`kubelet`.

첫 번째는 `clusterDNS` 필드를 통하는 것입니다. Config 파일은 사용할 DNS 서버의 IP 주소와 `clusterDNS` 함께 라는 `string` 필드를 수락합니다. 그러면 로 전달`kubelet`되어 `/etc/resolv.conf` 파일을 통해 포드로 전달됩니다. 자세한 내용은 구성 파일의 스키마를 참조하세요.

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

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

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

이 구성은 하나의 IP 주소만 허용합니다. 주소를 두 개 이상 지정하려면 `kubeletExtraConfig` 파라미터를 사용합니다.

```
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"]
```

테인트

특정 노드 그룹에 테인트를 적용하려면 다음과 같이 `taints` 구성 섹션을 사용합니다.

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

전체 예제는 여기에서 확인할 수 있습니다.

인스턴스 선택기

eksctl은 관리형 및 자체 관리형 노드 그룹에 여러 인스턴스 유형을 지정할 수 있도록 지원하지만 EC2 인스턴스 유형이 270개 이상인 경우 사용자는 노드 그룹에 적합한 인스턴스 유형을 찾는 데 시간을 할애해야 합니다. Cluster Autoscaler와 함께 작동하는 인스턴스 세트를 선택해야 하므로 스팟 인스턴스를 사용할 때는 더 어렵습니다.

이제 eksctl이 EC2 인스턴스 선택기와 통합되어 vCPUs, 메모리, GPUs. 인스턴스 선택기 기준이 통과되면 eksctl은 제공된 기준과 일치하는 인스턴스 유형으로 설정된 인스턴스 유형을 사용하여 노드 그룹을 생성합니다.

클러스터 및 노드 그룹 생성

eksctl에 전달된 인스턴스 선택기 리소스 기준과 일치하는 인스턴스 유형을 사용하는 단일 노드 그룹으로 클러스터를 생성하려면

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

그러면 instanceTypes 필드가 로 설정된 클러스터와 관리형 노드 그룹이 생성됩니다[c5.large, c5a.large, c5ad.large, c5d.large, t2.medium, t3.medium, t3a.medium](반환된 인스턴스 유형 세트가 변경될 수 있음).

비관리형 노드 그룹의 경우 instancesDistribution.instanceTypes 필드가 설정됩니다.

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

인스턴스 선택기 기준은 ClusterConfig에서도 지정할 수 있습니다.

```
# 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
```

다음 인스턴스 선택기 CLI 옵션은 eksctl create cluster 및에서 지원됩니다eksctl create nodegroup.

--instance-selector-vcpus, --instance-selector-memory, --instance-selector-gpus 및 instance-selector-cpu-architecture

예제 파일은 여기에서 찾을 수 있습니다.

드라이 런

모의 실행 기능을 사용하면 노드 그룹 생성을 진행하기 전에 인스턴스 선택기와 일치하는 인스턴스를 검사하고 변경할 수 있습니다.

```
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
```

그러면 생성된 ClusterConfig를에 전달할 수 있습니다`eksctl create cluster`.

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

가시성 및 문서화를 위해 CLI 옵션을 나타내는 `instanceSelector` 필드도 ClusterConfig 파일에 추가됩니다. `--dry-run`를 생략하면이 필드가 무시되고 `instanceTypes` 필드가 사용됩니다. 그렇지 않으면에 대한 변경 사항이 eksctl에 의해 재정의`instanceTypes`됩니다.

ClusterConfig 파일이와 함께 전달되면 `--dry-run`eksctl은 각 노드 그룹의 인스턴스 선택기 리소스 기준을 확장한 후 동일한 노드 그룹 집합이 포함된 ClusterConfig 파일을 출력합니다.

```
# 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
# ...
```

스팟 인스턴스

관리형 노드 그룹

eksctl는 내결함성 애플리케이션이 있는 EKS 고객이 EKS 클러스터에 대한 EC2 스팟 인스턴스를 쉽게 프로비저닝하고 관리할 수 있는 기능인 EKS 관리형 노드 그룹을 사용하여 스팟 워커 노드를 지원합니다. EC2 EKS 관리형 노드 그룹은 스팟 모범 사례에 따라 스팟 인스턴스의 EC2 Autoscaling 그룹을 구성하고 시작하고 인스턴스가 AWS에 의해 중단되기 전에 스팟 작업자 노드를 자동으로 드레이닝합니다. 이 기능을 사용하는 데는 증분 요금이 부과되지 않으며 고객은 EC2 스팟 인스턴스 및 EBS 볼륨과 같은 AWS 리소스 사용에 대해서만 비용을 지불합니다.

스팟 인스턴스를 사용하여 관리형 노드 그룹으로 클러스터를 생성하려면 --spot 플래그와 선택적 인스턴스 유형 목록을 전달합니다.

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

기존 클러스터에서 스팟 인스턴스를 사용하여 관리형 노드 그룹을 생성하려면:

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

구성 파일을 통해 관리형 노드 그룹을 사용하여 스팟 인스턴스를 생성하려면:

```
# 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**
>
> 비관리형 노드 그룹은 `spot` 및 `instanceTypes` 필드를 지원하지 않으며, 대신 `instancesDistribution` 필드는 스팟 인스턴스를 구성하는 데 사용됩니다. 아래 참조

추가 정보

- EKS 스팟 노드 그룹
- EKS 관리형 노드 그룹 용량 유형

비관리형 노드 그룹

`eksctl`는 Auto Scaling 그룹에 대한 MixedInstancesPolicy를 통해 스팟 인스턴스를 지원합니다.

다음은 50% 스팟 인스턴스와 50% 온디맨드 인스턴스를 사용하는 노드 그룹의 예입니다.

```
nodeGroups:
  - name: ng-1
    minSize: 2
```

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

nodeGroups.X.instanceType 필드를 사용할 때는 instancesDistribution 필드를 설정해서는 안 됩니다.

이 예제에서는 GPU 인스턴스를 사용합니다.

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

이 예제에서는 용량 최적화 스팟 할당 전략을 사용합니다.

```
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"
```

이 예제에서는 capacity-optimized-prioritized 지정 스팟 할당 전략을 사용합니다.

```
nodeGroups:
```

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

`capacity-optimized-prioritized` 할당 전략을 사용한 다음 시작 템플릿 재정의 목록에서 인스턴스 유형 순서를 가장 높은 우선 순위에서 가장 낮은 우선 순위(목록에서 첫 번째부터 마지막까지)로 설정합니다. Amazon EC2 Auto Scaling은 최상의 노력으로 인스턴스 유형 우선순위를 준수하지만 먼저 용량을 최적화합니다. 이는 중단 가능성을 최소화해야 하지만 특정 인스턴스 유형에 대한 기본 설정도 중요한 워크로드에 적합한 옵션입니다. 자세한 내용은 ASG 구매 옵션을 참조하세요.

`spotInstancePools` 필드를 사용할 때는 `spotAllocationStrategy` 필드를 설정해서는 안 됩니다. `spotAllocationStrategy`를 지정하지 않으면 EC2는 기본적으로 `lowest-price` 전략을 사용합니다.

다음은 최소한의 예입니다.

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

스팟 인스턴스와 온디맨드 인스턴스 간에 노드를 구분하려면 유형에 `on-demand` 따라 `spot` 또는 값이 `node-lifecycle` 있는 kubernetes 레이블을 사용할 수 있습니다.

instancesDistribution의 파라미터

자세한 내용은 클러스터 구성 스키마를 참조하세요.

GPU 지원

Eksctl은 노드 그룹에 대한 GPU 인스턴스 유형 선택을 지원합니다. 호환되는 인스턴스 유형을 생성 명령에 제공하거나 구성 파일을 통해 제공하면 됩니다.

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

> **Note**
>
> 더 이상 EKS에서 GPU 지원을 위해 마켓플레이스 AMI를 구독할 필요가 없습니다.

AMI 해석기(`auto` 및 `auto-ssm`)는 GPU 인스턴스 유형을 사용하려는 것을 확인하고 올바른 EKS 최적화 가속 AMI를 선택합니다.

Eksctl은 GPU 지원 인스턴스 유형의 AMI가 선택되었음을 감지하고 NVIDIA Kubernetes 디바이스 플러그인을 자동으로 설치합니다.

> **Note**
>
> Windows 및 Ubuntu AMIs GPU 드라이버가 설치된 상태로 제공되지 않으므로 GPU 가속 워크로드 실행은 즉시 작동하지 않습니다.

자동 플러그인 설치를 비활성화하고 특정 버전을 수동으로 설치하려면 create 명령`--install-nvidia-plugin=false`과 함께를 사용합니다. 예제:

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

및 버전 0.15.0 이상의 경우

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

또는 이전 버전의 경우

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

Bottlerocket이 이미 디바이스 플러그인 실행을 처리하므로 클러스터에 Bottlerocket 노드 그룹만 포함된 경우 NVIDIA Kubernetes 디바이스 플러그인 설치를 건너뜁니다. 클러스터 구성에 다른 AMI 패밀리를 사용하는 경우 테인트와 허용치를 사용하여 디바이스 플러그인이 Bottlerocket 노드에서 실행되지 않도록 해야 할 수 있습니다.

ARM 지원

이 주제에서는 ARM 노드 그룹을 사용하여 클러스터를 생성하는 방법과 기존 클러스터에 ARM 노드 그룹을 추가하는 방법을 다룹니다.

EKS는 Graviton 프로세서를 사용하여 64비트 ARM 아키텍처를 지원합니다. 클러스터를 생성하려면 Graviton 기반 인스턴스 유형(a1, , t4g, m6g, m7g, m6gd, c6g, c7gc6gd, r6g, r7g, r6gd, m8g, r8gc8g) 중 하나를 선택하고 다음을 실행합니다.

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

또는 구성 파일 사용:

```
---
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
```

ARM은 관리형 노드 그룹에서도 지원됩니다.

```
---
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 해석기 `auto` 및 `auto-ssm`는 ARM 인스턴스 유형에 따라 올바른 AMI를 유추합니다. AmazonLinux2023, AmazonLinux2 및 Bottlerocket 패밀리에만 ARM용 EKS 최적화 AMIs.

> **Note**
>
> ARM은 버전 1.15 이상의 클러스터에서 지원됩니다.

Auto Scaling

Auto Scaling 활성화

클러스터 오토스케일러 사용을 허용하는 IAM 역할을 사용하여 클러스터(또는 기존 클러스터의 노드 그룹)를 생성할 수 있습니다.

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

이 플래그는 `k8s.io/cluster-autoscaler/enabled` 및 `k8s.io/cluster-autoscaler/<clusterName>` 태그도 설정하므로 노드 그룹 검색이 작동해야 합니다.

클러스터가 실행되면 Cluster Autoscaler 자체를 설치해야 합니다.

또한 관리형 또는 비관리형 노드 그룹 정의(들)에 다음을 추가하여 Cluster Autoscaler가 노드 그룹을 조정하는 데 필요한 태그를 추가해야 합니다.

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

0에서 확장

노드 그룹을 0에서 확장할 수 있고 노드 그룹에 레이블 및/또는 테인트가 정의되어 있는 경우 이를 Auto Scaling 그룹(ASGs.

이를 수행하는 한 가지 방법은 노드 그룹 정의의 tags 필드에 ASG 태그를 설정하는 것입니다. 예를 들어 다음과 같은 레이블과 테인트가 있는 노드 그룹을 지정합니다.

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

다음 ASG 태그를 추가해야 합니다.

```
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"
```

관리형 노드 그룹과 비관리형 노드 그룹 모두에 대해를 propagateASGTags로 설정하여 이를 자동으로 수행할 수 true있습니다. 그러면 레이블과 테인트가 Auto Scaling 그룹에 태그로 추가됩니다.

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

영역 인식 Auto Scaling

워크로드가 영역별 워크로드인 경우 각 영역에 대해 별도의 노드 그룹을 생성해야 합니다. 이는가 그룹의 모든 노드가 정확히 동일하다고 cluster-autoscaler 가정하기 때문입니다. 따라서 예를 들어

영역별 PVC가 필요한 포드(예: EBS 볼륨)에 의해 스케일 업 이벤트가 트리거되는 경우 새 노드가 잘못된 AZ에 예약되어 포드가 시작되지 않을 수 있습니다.

환경이 다음 기준을 충족하는 경우 각 AZ에 대해 별도의 노드 그룹이 필요하지 않습니다.

- 영역별 스토리지 요구 사항은 없습니다.
- 호스트 이외의 토폴로지가 있는 podAffinity는 필요하지 않습니다.
- 영역 레이블에 필요한 nodeAffinity가 없습니다.
- 영역 레이블에 nodeSelector가 없습니다.

(여기 및 여기에서 자세히 읽어보세요.)

위의 요구 사항(및 기타 요구 사항)을 모두 충족하는 경우 여러 AZs에 걸쳐 있는 단일 노드 그룹으로 안전해야 합니다. 그렇지 않으면 별도의 단일 AZ 노드 그룹을 생성해야 합니다.

이전:

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

이후:

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

사용자 지정 AMI 지원

노드 AMI ID 설정

`--node-ami` 플래그를 사용하면 사용자 지정 AMI를 사용하거나 AWS를 실시간으로 쿼리하여 사용할 AMI를 결정하는 등 다양한 고급 사용 사례를 사용할 수 있습니다. 플래그는 GPU가 아닌 이미지와 GPU 이미지 모두에 사용할 수 있습니다.

플래그는 명시적으로 사용할 이미지의 AMI 이미지 ID를 가져올 수 있습니다. 또한 다음과 같은 '특수' 키워드를 사용할 수 있습니다.

키워드	설명
auto	AWS EC2를 쿼리하여 노드에 사용할 AMI를 찾아야 함을 나타냅니다. 이는 자동 해석기와 관련이 있습니다.
auto-ssm	AWS SSM 파라미터 스토어를 쿼리하여 노드에 사용할 AMI를 찾아야 함을 나타냅니다.

> **Note**
>
> 현재 EKS 관리형 노드 그룹은 사용자 지정 AMI로 작업할 때 , `AmazonLinux2023`, `AmazonLinux2`, `Bottlerocket`, 및 AMIs 패밀리만 지원합니다`Ubuntu2004`. `UbuntuPro2004` `Ubuntu2204` `Ubuntu2404`

`--node-ami`를 ID 문자열로 설정할 때 `eksctl`는 사용자 지정 AMI가 요청되었다고 가정합니다. EKS 관리형 노드와 자체 관리형 노드인 AmazonLinux2 및 Ubuntu 노드의 경우 `overrideBootstrapCommand`가 필요합니다. AmazonLinux2023의 경우 노드 부트스트래핑에 대한 `/etc/eks/bootstrap.sh` 스크립트 사용을 중지하므로 nodeadm 초기화 프로세스(자세한 내용은 노드 부트스트래핑 문서 참조)`overrideBootstrapCommand`에 유리하게 지원되지 않습니다.

CLI 플래그 예제:

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

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

Config 파일 예제:

```
nodeGroups:
  - name: ng1
    instanceType: p2.xlarge
    amiFamily: AmazonLinux2
```

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

`--node-ami` 플래그는 에서도 사용할 수 있습니다`eksctl create nodegroup`.

노드 AMI 패밀리 설정

는 다음 키워드를 사용할 `--node-ami-family` 수 있습니다.

키워드	설명
AmazonLinux2	Amazon Linux 2 기반 EKS AMI 이미지를 사용해야 함을 나타냅니다(기본값).
AmazonLinux2023	Amazon Linux 2023 기반 EKS AMI 이미지를 사용해야 함을 나타냅니다.
Ubuntu2004	Ubuntu 20.04 LTS(Focal) 기반 EKS AMI 이미지를 사용해야 함을 나타냅니다(EKS """" 1.29 지원).
UbuntuPro2004	Ubuntu Pro 20.04 LTS(Focal) 기반 EKS AMI 이미지를 사용해야 함을 나타냅니다(EKS >= 1.27, "" 1.29에 사용 가능).
Ubuntu2204	Ubuntu 22.04 LTS(Jammy) 기반 EKS AMI 이미지를 사용해야 함을 나타냅니다(EKS >= 1.29에 사용 가능).

키워드	설명
UbuntuPro2204	Ubuntu Pro 22.04 LTS(Jammy) 기반 EKS AMI 이미지를 사용해야 함을 나타냅니다(EKS >= 1.29에 사용 가능).
Ubuntu2404	Ubuntu 24.04 LTS(Noble) 기반 EKS AMI 이미지를 사용해야 함을 나타냅니다(EKS >= 1.31에 사용 가능).
UbuntuPro2404	Ubuntu Pro 24.04 LTS(Noble) 기반 EKS AMI 이미지를 사용해야 함을 나타냅니다(EKS >= 1.31에 사용 가능).
Bottlerocket	Bottlerocket 기반 EKS AMI 이미지를 사용해야 함을 나타냅니다.
WindowsServer2019FullContainer	Windows Server 2019 Full Container 기반 EKS AMI 이미지를 사용해야 함을 나타냅니다.
WindowsServer2019CoreContainer	Windows Server 2019 Core Container 기반 EKS AMI 이미지를 사용해야 함을 나타냅니다.
WindowsServer2022FullContainer	Windows Server 2022 Full Container 기반 EKS AMI 이미지를 사용해야 함을 나타냅니다.
WindowsServer2022CoreContainer	Windows Server 2022 Core Container 기반 EKS AMI 이미지를 사용해야 함을 나타냅니다.

CLI 플래그 예제:

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

Config 파일 예제:

```
nodeGroups:
  - name: ng1
    instanceType: m5.large
    amiFamily: AmazonLinux2
```

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

`--node-ami-family` 플래그는 에서도 사용할 수 있습니다`eksctl create nodegroup`. `eksctl`에서는 사용자 지정 AMI로 작업할 때마다 구성 파일 또는 `--node-ami-family` CLI 플래그를 통해 AMI 패밀리를 명시적으로 설정해야 합니다.

> **Note**
>
> 현재 EKS 관리형 노드 그룹은 사용자 지정 AMI로 작업할 때 , `AmazonLinux2023`, `AmazonLinux2`, `Bottlerocket`, 및 AMIs 패밀리만 지원합니다`Ubuntu2004`. `UbuntuPro2004` `Ubuntu2204` `Ubuntu2404`

Windows 사용자 지정 AMI 지원

자체 관리형 Windows 노드 그룹만 사용자 지정 AMI를 지정할 수 있습니다.는 유효한 Windows AMI 패밀리로 설정해야 `amiFamily` 합니다.

부트스트랩 스크립트에 사용할 수 있는 PowerShell 변수는 다음과 같습니다.

```
$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 파일 예제:

```
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 사용자 지정 AMI 지원

Bottlerocket 노드의 경우 `overrideBootstrapCommand`는 지원되지 않습니다. 대신 자체 부트스트랩 컨테이너를 지정하려면 `bottlerocket` 필드를 구성 파일의 일부로 사용해야 합니다. 예:

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

Windows 작업자 노드

버전 1.14부터 Amazon EKS는 Windows 컨테이너 실행을 허용하는 Windows 노드를 지원합니다. Windows 노드가 있는 것 외에도 Microsoft는 아직 호스트 네트워킹 모드를 지원하지 않으므로 클러스터의 Linux 노드가 CoreDNS를 실행해야 합니다. 따라서 Windows EKS 클러스터는 Windows 노드와 하나 이상의 Linux 노드가 혼합되어 있습니다. Linux 노드는 클러스터의 작동에 매우 중요하므로 프로덕션 등급 클러스터의 경우 HA용 `t2.large` Linux 노드를 두 개 이상 보유하는 것이 좋습니다.

> **Note**
>
> 2021년 10월 22일 이후에 생성된 EKS 클러스터에서 Windows 워크로드를 실행하기 위해 Linux 작업자 노드에 VPC 리소스 컨트롤러를 더 이상 설치할 필요가 없습니다. ConfigMap 설정을 통해 EKS 컨트롤 플레인에서 Windows IP 주소 관리를 활성화할 수 있습니다(자세한 내용은 link:eks/latest/userguide/windows-support.html 참조). eksctl은 Windows 노드 그룹이 생성될 때 Windows IP 주소 관리를 활성화하기 위해 ConfigMap을 자동으로 패치합니다.

Windows 지원을 사용하여 새 클러스터 생성

구성 파일 구문을 사용하면 단일 명령으로 Windows를 지원하는 완전한 기능을 갖춘 클러스터를 생성할 수 있습니다.

```
# 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
```

구성 파일을 사용하지 않고 Windows 비관리형 노드 그룹을 사용하여 새 클러스터를 생성하려면 다음 명령을 실행합니다.

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

기존 Linux 클러스터에 Windows 지원 추가

Linux 노드(AmazonLinux2 AMI 패밀리)가 있는 기존 클러스터에서 Windows 워크로드 실행을 활성화하려면 Windows 노드 그룹을 추가해야 합니다.

NEW Windows 관리형 노드 그룹에 대한 지원이 추가되었습니다(--managed=true 또는 플래그 생략).

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

워크로드가 올바른 OS에 예약되도록 하려면 워크로드가 실행되어야 하는 OS를 nodeSelector 대상으로 하는이 있어야 합니다.

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

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

kubernetes.io/os 및 kubernetes.io/arch 레이블보다 오래된 클러스터1.19를 사용하는 경우 beta.kubernetes.io/arch 각각 beta.kubernetes.io/os 및 로 교체해야 합니다.

추가 정보

- EKS Windows 지원

추가 볼륨 매핑

추가 구성 옵션으로 볼륨 매핑을 처리할 때 노드 그룹이 생성될 때 추가 매핑을 구성할 수 있습니다.

이렇게 하려면 필드를 다음과 additionalVolumes 같이 설정합니다.

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

```
kind: ClusterConfig

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

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

volumeNames 선택에 대한 자세한 내용은 디바이스 이름 지정 설명서를 참조하세요. EBS 볼륨, 인스턴스 볼륨 제한 또는 블록 디바이스 매핑에 대한 자세한 내용은 이 페이지를 참조하세요.

EKS 하이브리드 노드

소개

AWS EKS는 AWS 클라우드에서 사용하는 것과 동일한 AWS EKS 클러스터, 기능 및 도구를 사용하여 고객 관리형 인프라에서 온프레미스 및 엣지 애플리케이션을 실행할 수 있는 새로운 기능인 하이브리드 노드를 도입합니다. AWS EKS 하이버드 노드는 고객이 온프레미스, 엣지 및 클라우드 환경에서 애플리케이션을 실행하는 방법을 간소화하고 표준화할 수 있도록 온프레미스 환경에 AWS 관리형 Kubernetes 환경을 제공합니다. 자세한 내용은 EKS Hybrid Nodes에서 확인하세요.

이 기능에 대한 지원을 용이하게 하기 위해 eksctl은 라는 새로운 최상위 필드를 도입합니다remoteNetworkConfig. 하이브리드 노드 관련 구성은 구성 파일의 일부로이 필드를 통해 설정해야 합니다. CLI 플래그는 해당되지 않습니다. 또한 시작 시 모든 원격 네트워크 구성은 클러스터 생성

중에만 설정할 수 있으며 나중에 업데이트할 수 없습니다. 즉, 하이브리드 노드를 사용하도록 기존 클러스터를 업데이트할 수 없습니다.

구성 파일의 `remoteNetworkConfig` 섹션을 사용하면 원격 노드를 EKS 클러스터에 조인할 때 네트워킹 및 자격 증명이라는 두 가지 핵심 영역을 설정할 수 있습니다.

네트워킹

EKS Hybrid Nodes는 온프레미스 네트워크(들)를 AWS VPC에 연결하는 선호하는 방법에 유연합니다. AWS Site-to-Site VPN 및 AWS Direct Connect를 포함하여 몇 가지 문서화된 옵션을 사용할 수 있으며 사용 사례에 가장 적합한 방법을 선택할 수 있습니다. 선택할 수 있는 대부분의 방법에서 VPC는 가상 프라이빗 게이트웨이(VGW) 또는 전송 게이트웨이(TGW)에 연결됩니다. eksctl을 사용하여 VPC를 생성하는 경우 eksctl은 EKS 컨트롤 플레인과 원격 노드 간의 통신을 용이하게 하기 위해 VPC 범위 내에서 네트워킹 관련 사전 조건도 구성합니다.

- 수신/송신 SG 규칙
- 프라이빗 서브넷의 라우팅 테이블의 라우팅
- 지정된 TGW 또는 VGW에 대한 VPC 게이트웨이 연결

구성 파일의 예:

```
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"]
```

선택한 연결 방법에 TGW 또는 VGW 사용이 포함되지 않은 경우 eksctl을 사용하여 VPC를 생성해서는 안 되며 대신 기존 VPC를 제공해야 합니다. 관련 참고로 기존 VPC를 사용하는 경우 eksctl은 이를 수정하지 않으며 모든 네트워킹 요구 사항이 적용되도록 하는 것은 사용자의 책임입니다.

> **Note**
>
> eksctl은 AWS VPC 외부의 네트워킹 인프라(예: VGW/TGW에서 원격 네트워크로의 인프라)를 설정하지 않습니다.

자격 증명

EKS Hybrid Nodes는 AWS SSM 또는 AWS IAM Roles Anywhere에서 프로비저닝한 AWS IAM Authenticator 및 임시 IAM 자격 증명을 사용하여 EKS 클러스터로 인증합니다. 자체 관리형 노드 그룹과 마찬가지로 달리 제공되지 않은 경우 eksctl은 원격 노드가 수임할 하이브리드 노드 IAM 역할을 생성합니다. 또한 IAM Roles Anywhere를 자격 증명 공급자로 사용하는 경우 eksctl은 지정된 인증 기관 번들(`iam.caBundleCert`)을 기반으로 프로필을 설정하고 앵커를 신뢰합니다. 예:

```
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
```

eksctl에서 생성한 하이브리드 노드 역할의 ARN은 나중에 원격 노드를 클러스터에 조인하고, `NodeConfig`를 설정하고`nodeadm`, 활성화를 생성하는 과정에서 필요합니다(SSM을 사용하는 경우). 가져오려면 다음을 사용합니다.

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

마찬가지로 IAM Roles Anywhere를 사용하는 경우 eksctl에서 생성한 트러스트 앵커 및 Anywhere 프로파일의 ARN을 가져와 각각 `RemoteNodesTrustAnchorARN` 또는 `RemoteNodesRoleARN`로 대체하여 이전 명령을 수정할 수 `RemoteNodesAnywhereProfileARN`있습니다.

기존 IAM Roles Anywhere 구성이 있거나 SSM을 사용하는 경우를 통해 하이브리드 노드에 대한 IAM 역할을 제공할 수 있습니다`remoteNetworkConfig.iam.roleARN`. 이 시나리오에서는 eksctl이 트러스트 앵커와 어디에나 프로필을 생성하지 않는다는 점에 유의하세요. 예:

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

역할을 Kubernetes 자격 증명에 매핑하고 원격 노드가 EKS 클러스터에 조인하도록 권한을 부여하기 위해 eksctl은 하이브리드 노드 IAM 역할을 보안 주체 ARN 및 유형 로 사용하여 액세스 항목을 생성합니다. HYBRID_LINUX예:

```
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"
        ]
    }
]
```

추가 기능 지원

컨테이너 네트워킹 인터페이스(CNI): AWS VPC CNI를 하이브리드 노드와 함께 사용할 수 없습니다. Cilium과 Calico의 핵심 기능은 하이브리드 노드와 함께 사용할 수 있습니다. Helm과 같은 도구를 선택하여 CNI를 관리할 수 있습니다. 자세한 내용은 하이브리드 노드에 대한 CNI 구성을 참조하세요.

> **Note**
>
> 자체 관리형 또는 EKS 관리형 노드 그룹의 클러스터에 VPC CNI를 설치하는 경우 하이브리드 노드에 설치되지 않도록 제외하기 위해 추가 기능의 데몬 세트에 대한 udpate가 포함되므로 v1.19.0-eksbuild.1 이상을 사용해야 합니다.

추가 참조

- EKS Hybrid Nodes UserDocs
- 시작 알림

EKS 관리형 노드 그룹에 대한 노드 복구 구성 지원

EKS 관리형 노드 그룹은 관리형 노드의 상태가 모니터링되고 비정상 작업자 노드가 응답으로 교체되거나 재부팅되는 노드 복구를 지원합니다. 이제 eksctl은 노드 복구 동작을 세밀하게 제어할 수 있는 포괄적인 구성 옵션을 제공합니다.

기본 노드 복구 구성

CLI 플래그 사용

기본 노드 복구를 사용하여 관리형 노드 그룹으로 클러스터를 생성하려면 `--enable-node-repair` 플래그를 전달합니다.

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

기존 클러스터에서 노드 복구를 사용하여 관리형 노드 그룹을 생성하려면:

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

구성 파일 사용

```
# 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
```

향상된 노드 복구 구성

임계값 구성

노드 복구 작업이 백분율 또는 개수 기반 임계값 사용을 중지하는 시기를 구성할 수 있습니다. 참고: 백분율 임계값과 개수 임계값을 동시에 사용할 수는 없습니다.

임계값에 대한 CLI 플래그

```
# 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
```

임계값에 대한 구성 파일

```
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
```

병렬 복구 제한

동시에 또는 병렬로 복구할 수 있는 최대 노드 수를 제어합니다. 이를 통해 노드 교체 속도를 더 세밀하게 제어할 수 있습니다. 참고: 백분율 및 개수 제한을 동시에 사용할 수는 없습니다.

병렬 제한에 대한 CLI 플래그

```
# 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
```

병렬 제한에 대한 구성 파일

```
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
```

사용자 지정 복구 재정의

특정 복구 작업에 대한 세분화된 재정의를 지정합니다. 이러한 재정의는 노드를 복구할 수 있는 항목으로 간주하기 전에 복구 작업과 복구 지연 시간을 제어합니다. 이를 사용하는 경우 각 재정의를 위한 모든 값을 지정해야 합니다.

```
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"
```

전체 구성 예제

모든 구성 옵션이 포함된 포괄적인 예제는 example/44-node-repair.yaml을 참조하세요.

예제 1: 백분율 임계값을 사용한 기본 복구

```
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
```

예제 2: 중요한 워크로드에 대한 보존적 복구

```
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"
```

예제 3: 특수 복구가 포함된 GPU 워크로드

```
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 참조

노드 복구 플래그

플래그	설명	예제
`--enable-node-repair`	자동 노드 복구 활성화	`--enable-node-repair`
`--node-repair-max-unhealthy-percentage`	복구 전 비정상 노드의 최대 비율	`--node-repair-max-unhealthy-percentage=20`
`--node-repair-max-unhealthy-count`	복구 전 비정상 노드의 최대 수	`--node-repair-max-unhealthy-count=5`

플래그	설명	예제
`--node-repair-max-parallel-percentage`	병렬로 복구할 노드의 최대 백분율	`--node-repair-max-parallel-percentage=15`
`--node-repair-max-parallel-count`	병렬로 복구할 최대 노드 수	`--node-repair-max-parallel-count=2`

참고: 노드 복구 구성 재정의는 복잡성으로 인해 YAML 구성 파일을 통해서만 지원됩니다.

구성 참조

nodeRepairConfig

Field	Type	설명	제약 조건	예제
`enabled`	부울	노드 복구 활성화/비활성화	-	`true`
`maxUnhealthyNodeThresholdPercentage`	정수	노드 자동 복구 작업이 중지되는 비정상 노드의 백분율 임계값	와 함께 사용할 수 없음 `maxUnhealthyNodeThresholdCount`	`20`
`maxUnhealthyNodeThresholdCount`	정수	노드 자동 복구 작업이 중지되는 비정상 노드의 임계값 계산	와 함께 사용할 수 없음 `maxUnhealthyNodeThresholdPercentage`	`5`
`maxParallelNodesRe`	정수	동시에 또는 병렬로 복구할 수 있는 비정상 노드의 최대 백분율	와 함께 사용할 수 없음 `maxParall`	`15`

Field	Type	설명	제약 조건	예제
pairedPercentage			elNodesRepairedCount	
maxParallelNodesRepairedCount	정수	동시 또는 병렬로 복구할 수 있는 비정상 노드의 최대 수	와 함께 사용할 수 없음 maxParallelNodesRepairedPercentage	2
nodeRepairConfigOverrides	배열	복구 작업 및 지연 시간을 제어하는 특정 복구 작업에 대한 세분화된 재정의	각 재정의에 대해 모든 값을 지정해야 합니다.	위의 예제 참조

nodeRepairConfigOverrides

Field	Type	설명	유효한 값
nodeMonitoringCondition	문자열	이 재정의가 적용되는 노드 모니터링 에이전트가 보고하는 비정상 조건	"AcceleratedInstanceNotReady" , "NetworkNotReady"
nodeUnhealthyReason	문자열	이 재정의가 적용되는 노드 모니터링 에이전트가 보고하는 이유	"NvidiaXID13Error" ,"InterfaceNotUp"
minRepairWaitTimeMins	정수	지정된 조건 및 이유로 노드 복구를 시도하기 전에 기다리는 최소 시간	모든 양의 정수

Field	Type	설명	유효한 값
`repairAction`	문자열	지정된 모든 조건이 충족될 때 노드에 대해 수행할 복구 작업	`"Terminate"` , `"Restart"` , `"NoAction"`

추가 정보

- EKS 관리형 노드 그룹 노드 상태

네트워킹

이 장에는 Eksctl이 EKS 클러스터용 Virtual Private Cloud(VPC) 네트워크를 생성하는 방법에 대한 정보가 포함되어 있습니다.

주제:

- the section called “VPC 구성”
  - VPC CIDR 범위 수정 및 IPv6 주소 지정 구성
  - 기존 VPC 사용
  - 새 EKS 클러스터의 VPC, 서브넷, 보안 그룹 및 NAT 게이트웨이 사용자 지정
- the section called “서브넷 설정”
  - 초기 노드 그룹에 프라이빗 서브넷을 사용하여 퍼블릭 인터넷에서 격리
  - 가용 영역당 여러 서브넷을 나열하고 노드 그룹 구성에서 서브넷을 지정하여 서브넷 토폴로지 사용자 지정
  - VPC 구성에서 노드 그룹을 이름이 지정된 특정 서브넷으로 제한
  - 노드 그룹에 프라이빗 서브넷을 사용하는 경우를 `privateNetworking`로 설정합니다. `true`
  - VPC 사양에서 `public` 및 `private` 구성을 모두 포함하는 전체 서브넷 사양 제공
  - 노드 그룹 구성에는 `subnets` 또는 중 하나만 제공할 `availabilityZones` 수 있습니다.
- the section called “클러스터 액세스”
  - EKS 클러스터의 Kubernetes API 서버 엔드포인트에 대한 퍼블릭 및 프라이빗 액세스 관리
  - 허용된 CIDR 범위를 지정하여 EKS Kubernetes 퍼블릭 API 엔드포인트에 대한 액세스 제한
  - 기존 클러스터에 대한 API 서버 엔드포인트 액세스 구성 및 퍼블릭 액세스 CIDR 제한 업데이트
- the section called “컨트롤 플레인 네트워킹”
  - 클러스터의 EKS 컨트롤 플레인에서 사용하는 서브넷 업데이트
- the section called “IPv6 지원”
  - EKS 클러스터로 VPC를 생성할 때 사용할 IP 버전(IPv4 또는 IPv6)을 지정합니다.

VPC 구성

클러스터 전용 VPC

기본적으로 `eksctl create cluster`는 클러스터에 대한 전용 VPC를 생성합니다. 이는 보안을 비롯한 다양한 이유로 기존 리소스에 대한 간섭을 방지하기 위한 것이지만 기존 VPC의 모든 설정을 감지하기가 어려우므로 가능합니다.

- 에서 사용하는 기본 VPC CIDR은 `eksctl`입니다`192.168.0.0/16`.
  - (/19) 서브넷 8개(프라이빗 3개, 퍼블릭 3개, 예약 2개)로 나뉩니다.
- 초기 노드 그룹은 퍼블릭 서브넷에 생성됩니다.
- 를 지정하지 않으면 SSH 액세스가 비활성화`--allow-ssh`됩니다.
- 노드 그룹은 기본적으로 포트 1025~65535에서 컨트롤 플레인 보안 그룹의 인바운드 트래픽을 허용합니다.

> **Note**
>
> `us-east-1` eksctl에서는 기본적으로 퍼블릭 서브넷 2개와 프라이빗 서브넷 2개만 생성합니다.

VPC CIDR 변경

다른 VPC와의 피어링을 설정해야 하거나 단순히 더 크거나 더 작은 범위의 IPs 필요한 경우 `--vpc-cidr` 플래그를 사용하여 변경할 수 있습니다. AWS VPC에서 사용할 수 있는 CIDR 블록 선택에 대한 지침은 AWS 문서를 참조하세요.

IPv6 클러스터를 생성하는 경우 cluser 구성 파일`VPC.IPv6Cidr`에서를 구성할 수 있습니다. 이 설정은 CLI 플래그가 아닌 구성 파일에만 있습니다.

IPv6 IP 주소 블록을 소유한 경우 자체 IPv6 풀을 가져올 수도 있습니다. 자체 풀을 가져오는 방법은 Amazon EC2로 자체 IP 주소 가져오기(BYOIP)를 참조하세요. 그런 다음 cluser config 파일의 `VPC.IPv6Cidr`를 사용하여 Eksctl을 구성합니다.

기존 VPC 사용: kops와 공유

kops에서 관리하는 기존 Kubernetes 클러스터의 VPC를 사용할 수 있습니다. 이 기능은 마이그레이션 및/또는 클러스터 피어링을 용이하게 하기 위해 제공됩니다.

다음과 유사한 명령을 사용하는 등 이전에 kops로 클러스터를 생성한 경우:

```
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
```

동일한 VPC 서브넷을 사용하여 동일한 AZs에서 EKS 클러스터를 생성할 수 있습니다(참고: 최소 2AZs/서브넷 필요).

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

기존 VPC 사용: 기타 사용자 지정 구성

`eksctl`는 사용자 지정 VPC 및 서브넷 토폴로지에 대한 일부 유연성을 제공하지만 완전하지는 않습니다.

`--vpc-private-subnets` 및 `--vpc-public-subnets` 플래그를 사용하여 프라이빗 및/또는 퍼블릭 서브넷을 제공하여 기존 VPC를 사용할 수 있습니다. 구성이 다양하기 때문에 서브넷이 실제로 프라이빗인지 퍼블릭인지 확인하는 간단한 방법이 없기 때문에 사용하는 서브넷이 올바르게 분류되었는지 확인하는 것은 사용자의 몫입니다.

이러한 플래그가 주어지면 `eksctl create cluster`는 VPC ID를 자동으로 결정하지만 인터넷/NAT 게이트웨이와 같은 라우팅 테이블이나 기타 리소스는 생성하지 않습니다. 그러나 초기 노드 그룹 및 컨트롤 플레인에 대한 전용 보안 그룹을 생성합니다.

서로 다른 AZs에 최소 2개의 서브넷을 제공해야 하며이 조건은 EKS에서 확인합니다. 기존 VPC를 사용하는 경우 EKS 또는 Eksctl에서 다음 요구 사항을 적용하거나 확인하지 않으며 EKS는 클러스터를 생성합니다. 클러스터의 일부 기본 함수는 이러한 요구 사항 없이 작동합니다. (예를 들어 태그 지정이 반드시 필요한 것은 아니며, 테스트 결과 서브넷에 태그가 설정되지 않은 상태에서 기능 클러스터를 생성할 수 있는 것으로 나타났습니다. 하지만 항상 이를 유지하고 태그를 지정하는 것은 권장되지 않습니다.)

표준 요구 사항:

- 지정된 모든 서브넷은 동일한 IP 블록 내에서 동일한 VPC에 있어야 IPs
- 필요에 따라 충분한 수의 IP 주소를 사용할 수 있음
- 필요에 따라 충분한 수의 서브넷(최소 2개)
- 서브넷에는 최소한 다음 태그가 지정됩니다.
  - `kubernetes.io/cluster/<name>` `shared` 또는 중 하나로 설정된 태그 `owned`
  - `kubernetes.io/role/internal-elb` 프라이빗 서브넷에 1 대해 로 설정된 태그
  - `kubernetes.io/role/elb` 퍼블릭 서브넷에 1 대해 로 설정된 태그
- 올바르게 구성된 인터넷 및/또는 NAT 게이트웨이
- 라우팅 테이블에 올바른 항목이 있고 네트워크가 작동함
- 신규: 모든 퍼블릭 서브넷에는 속성이 `MapPublicIpOnLaunch` 활성화되어 있어야 합니다(예: AWS 콘솔`Auto-assign public IPv4 address`에서). 관리형 노드 그룹과 Fargate는 퍼블릭 IPv4 주소를 할당하지 않으므로 서브넷에 속성을 설정해야 합니다.

EKS 또는 Kubernetes에서 부과하는 다른 요구 사항이 있을 수 있으며, 모든 요구 사항 및/또는 권장 사항을 up-to-date 상태로 유지하고 필요에 따라/가능한 경우 이를 구현하는 것은 전적으로 사용자의 몫입니다.

에서 적용되는 기본 보안 그룹 설정은 다른 보안 그룹의 리소스와 액세스를 공유하는 데 충분하지 않거나 충분하지 않을 수 `eksctl` 있습니다. 보안 그룹의 수신/송신 규칙을 수정하려면 다른 도구를 사용하여 변경을 자동화하거나 EC2 콘솔을 통해 변경해야 할 수 있습니다.

의심스러운 경우 사용자 지정 VPC를 사용하지 마십시오. `--vpc-*` 플래그 `eksctl create cluster` 없이를 사용하면 항상 완전한 기능을 갖춘 전용 VPC로 클러스터가 구성됩니다.

예시

프라이빗 서브넷 2개와 퍼블릭 서브넷 2개가 있는 사용자 지정 VPC를 사용하여 클러스터를 생성합니다.

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

또는 다음과 같은 동등한 구성 파일을 사용합니다.

```
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
```

3x 프라이빗 서브넷이 있는 사용자 지정 VPC를 사용하여 클러스터를 생성하고 초기 노드 그룹이 해당 서브넷을 사용하도록 합니다.

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

또는 다음과 같은 동등한 구성 파일을 사용합니다.

```
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
```

사용자 지정 VPC 4x 퍼블릭 서브넷을 사용하여 클러스터를 생성합니다.

```
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
```

리포지토리의 `examples` 폴더에서 더 많은 예제를 찾을 수 있습니다.

- 기존 VPC 사용
- 사용자 지정 VPC CIDR 사용

사용자 지정 공유 노드 보안 그룹

eksctl는 비관리형 노드와 클러스터 컨트롤 플레인 및 관리형 노드 간의 통신을 허용하는 공유 노드 보안 그룹을 생성하고 관리합니다.

대신 사용자 지정 보안 그룹을 제공하려는 경우 구성 파일의 sharedNodeSecurityGroup 필드를 재정의할 수 있습니다.

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

기본적으로 클러스터를 생성할 때 eksctl는 EKS가 생성하는 기본 클러스터 보안 그룹과의 통신을 허용하는 규칙을이 보안 그룹에 추가합니다. 기본 클러스터 보안 그룹은 EKS 컨트롤 플레인과 관리형 노드 그룹 모두에서 사용됩니다.

보안 그룹 규칙을 직접 관리하려는 경우 구성 파일false에서를 manageSharedNodeSecurityGroupRules로 설정하여 eksctl가 규칙을 생성하지 못하도록 할 수 있습니다.

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

NAT 게이트웨이

클러스터의 NAT 게이트웨이는 Disable, Single (기본값) 또는 로 구성할 수 있습니다HighlyAvailable. HighlyAvailable 옵션은 리전의 각 가용 영역에 NAT 게이트웨이를 배포하므로 AZ가 다운된 경우에도 다른 AZs의 노드는 인터넷과 통신할 수 있습니다.

CLI --vpc-nat-mode 플래그를 통해 지정하거나 아래 예제와 같이 클러스터 구성 파일에서 지정할 수 있습니다.

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

여기에서 전체 예제를 참조하세요.

> **Note**
>
> NAT 게이트웨이 지정은 클러스터 생성 중에만 지원됩니다. 클러스터 업그레이드 중에는 접촉되지 않습니다.

서브넷 설정

초기 노드 그룹에 프라이빗 서브넷 사용

퍼블릭 인터넷에서 초기 노드 그룹을 격리하려는 경우 `--node-private-networking` 플래그를 사용할 수 있습니다. `--ssh-access` 플래그와 함께 사용하는 경우 SSH 포트는 VPC 내부에서만 액세스할 수 있습니다.

> **Note**
>
> `--node-private-networking` 플래그를 사용하면 발신 트래픽이 탄력적 IP를 사용하여 NAT 게이트웨이를 통과합니다. 반면 노드가 퍼블릭 서브넷에 있는 경우 발신 트래픽은 NAT 게이트웨이를 통과하지 않으므로 발신 트래픽에는 각 개별 노드의 IP가 있습니다.

사용자 지정 서브넷 토폴로지

`eksctl` 버전에는 다음과 같은 기능을 갖춘 추가 서브넷 토폴로지 사용자 지정이 `0.32.0` 도입되었습니다.

- VPC 구성에서 AZ당 여러 서브넷 나열
- 노드 그룹 구성에서 서브넷 지정

이전 버전에서는 가용 영역에서 사용자 지정 서브넷을 제공해야 했습니다. 즉, AZ당 하나의 서브넷만 나열할 수 있습니다. 식별 키`0.32.0`는 임의적일 수 있습니다.

```
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**
>
> AZ를 식별 키로 사용하는 경우 `az` 값을 생략할 수 있습니다.

위와 같이 임의의 문자열을 식별 키로 사용하는 경우 다음 중 하나를 수행합니다.

- `id`를 설정해야 합니다(`az` 및 `cidr` 선택 사항).
- 또는를 설정해야 `az` 합니다(`cidr`선택 사항).

사용자가 CIDR 및 ID를 지정하지 않고 AZ에서 서브넷을 지정하는 경우 해당 AZ의 서브넷은 VPC에서 선택되며, 이러한 서브넷이 여러 개 있는 경우 임의로 선택됩니다.

> **Note**
>
> 전체 서브넷 사양, 즉 VPC 사양에 선언된 `public` 및 `private` 구성을 모두 제공해야 합니다.

노드 그룹은 구성을 통해 명명된 서브넷으로 제한할 수 있습니다. 노드 그룹 구성에서 서브넷을 지정할 때는 서브넷 ID가 아닌 VPC 사양에 지정된 식별 키를 사용합니다. 예제:

```
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**
>
> 노드 그룹 구성에는 `subnets` 또는 중 하나만 제공할 `availabilityZones` 수 있습니다.

프라이빗 서브넷 내에 노드 그룹을 배치할 때는 노드 그룹에서 `true`를 로 설정해야 `privateNetworking` 합니다.

```
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
```

전체 구성 예제는 eksctl GitHub 리포지토리의 24-nodegroup-subnets.yaml을 참조하세요.

클러스터 액세스

Kubernetes API 서버 엔드포인트에 대한 액세스 관리

기본적으로 EKS 클러스터는 Kubernetes API 서버를 공개적으로 노출하지만 VPC 서브넷 내에서 직접 노출하지는 않습니다(public=true, private=false). VPC 내에서 API 서버로 향하는 트래픽은 먼저 VPC 네트워크(Amazon의 네트워크 아님)를 종료한 다음 다시 입력하여 API 서버에 도달해야 합니다.

클러스터에 대한 Kubernetes API 서버 엔드포인트 액세스는 클러스터 구성 파일을 사용하여 클러스터를 생성할 때 퍼블릭 및 프라이빗 액세스를 위해 구성할 수 있습니다. 아래 예제:

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

Kubernetes API 엔드포인트 액세스를 구성할 때 몇 가지 추가 주의 사항이 있습니다.

1. EKS는 프라이빗 또는 퍼블릭 액세스가 활성화되지 않은 클러스터를 허용하지 않습니다.
2. EKS는 프라이빗 액세스만 활성화할 수 있는 구성을 생성할 수 있지만 eksctl은 클러스터 생성 중에 이를 지원하지 않습니다. eksctl이 작업자 노드를 클러스터에 조인할 수 없기 때문입니다.
3. 프라이빗 전용 Kubernetes API 엔드포인트 액세스 권한이 있도록 클러스터를 업데이트하면 기본적으로 Kubernetes 명령(예: `kubectl`)과 , `eksctl delete cluster` `eksctl utils write-kubeconfig`및 가능한 경우 명령을 클러스터 VPC 내에서 실행해야 `eksctl utils update-kube-proxy` 합니다.
   - 이를 위해서는 다양한 AWS 리소스를 일부 변경해야 합니다. 자세한 내용은 클러스터 API 서버 엔드포인트를 참조하세요.
   - `vpc.extraCIDRs`이 ControlPlaneSecurityGroup에 추가 CIDR 범위를 추가할를 제공하여 VPC 외부의 서브넷이 kubernetes API 엔드포인트에 도달할 수 있도록 할 수 있습니다. 마찬가지로 IPv6 CIDR 범위를 추가`vpc.extraIPv6CIDRs`하기 위해를 제공할 수도 있습니다.

다음은 `utils` 하위 명령을 사용하여 Kubernetes API 엔드포인트 액세스를 구성하는 방법의 예입니다.

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

`ClusterConfig` 파일을 사용하여 설정을 업데이트하려면 다음을 사용합니다.

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

플래그를 전달하지 않으면 현재 값이 유지됩니다. 제안된 변경 사항에 만족하면 `approve` 플래그를 추가하여 실행 중인 클러스터를 변경합니다.

EKS Kubernetes 퍼블릭 API 엔드포인트에 대한 액세스 제한

EKS 클러스터의 기본 생성은 Kubernetes API 서버를 공개적으로 노출합니다.

이 기능은 퍼블릭 엔드포인트에만 적용됩니다. API 서버 엔드포인트 액세스 구성 옵션은 변경되지 않으며, 클러스터가 인터넷에서 액세스할 수 없도록 퍼블릭 엔드포인트를 비활성화할 수 있습니다. (소스: https://github.com/aws/containers-roadmap/issues/108#issuecomment-552766489)

클러스터를 생성할 때 퍼블릭 API 엔드포인트에 대한 액세스를 CIDRs 세트로 제한하려면 **publicAccessCIDRs** 필드를 설정합니다.

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

기존 클러스터에 대한 제한을 업데이트하려면 다음을 사용합니다.

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

ClusterConfig 파일을 사용하여 제한을 업데이트하려면에서 새 CIDRs 설정하고 다음을 **vpc.publicAccessCIDRs** 실행합니다.

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

> **Important**
>
> 노드 그룹을 설정하고 publicAccessCIDRs 생성하는 경우를 로 privateAccess 설정true하거나 노드의 IPs publicAccessCIDRs 목록에 추가해야 합니다.

제한된 액세스로 인해 노드가 클러스터 API 엔드포인트에 액세스할 수 없는 경우 노드가 퍼블릭 엔드포인트에 액세스할 수 없고 클러스터에 조인하지 context deadline exceeded 못하여 클러스터 생성에 실패합니다.

단일 명령으로 클러스터에 대한 API 서버 엔드포인트 액세스 및 퍼블릭 액세스 CIDRs을 모두 업데이트하려면 다음을 실행합니다.

```
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
```

구성 파일을 사용하여 설정을 업데이트하려면:

```
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
```

컨트롤 플레인 서브넷 및 보안 그룹 업데이트

이 설명서에서는 초기 생성 후 EKS 클러스터 컨트롤 플레인의 네트워킹 구성을 수정하는 방법을 설명합니다. 여기에는 컨트롤 플레인 서브넷 및 보안 그룹 업데이트가 포함됩니다.

컨트롤 플레인 서브넷 업데이트

eksctl을 사용하여 클러스터를 생성하면 퍼블릭 및 프라이빗 서브넷 세트가 생성되어 EKS API로 전달됩니다. EKS는 해당 서브넷에 2~4개의 교차 계정 탄력적 네트워크 인터페이스(ENIs)를 생성하여 EKS 관리형 Kubernetes 컨트롤 플레인과 VPC 간의 통신을 가능하게 합니다.

EKS 컨트롤 플레인에서 사용하는 서브넷을 업데이트하려면 다음을 실행합니다.

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

구성 파일을 사용하여 설정을 업데이트하려면:

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

vpc:
  controlPlaneSubnetIDs: [subnet-1234, subnet-5678]
```

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

`--approve` 플래그가 없으면 eksctl은 제안된 변경 사항만 로깅합니다. 제안된 변경 사항에 만족하면 `--approve` 플래그를 사용하여 명령을 다시 실행합니다.

컨트롤 플레인 보안 그룹 업데이트

컨트롤 플레인과 작업자 노드 간의 트래픽을 관리하기 위해 EKS는 EKS에서 프로비저닝한 교차 계정 네트워크 인터페이스에 적용되는 추가 보안 그룹 전달을 지원합니다. EKS 컨트롤 플레인의 보안 그룹을 업데이트하려면 다음을 실행합니다.

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

구성 파일을 사용하여 설정을 업데이트하려면:

```
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
```

클러스터의 컨트롤 플레인 서브넷과 보안 그룹을 모두 업데이트하려면 다음을 실행합니다.

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

구성 파일을 사용하여 두 필드를 모두 업데이트하려면:

```
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
```

전체 예제는 cluster-subnets-sgs.yaml을 참조하세요.

--approve 플래그가 없으면 eksctl은 제안된 변경 사항만 로깅합니다. 제안된 변경 사항에 만족하면 --approve 플래그를 사용하여 명령을 다시 실행합니다.

IPv6 지원

IP 패밀리 정의

가 vpc를 eksctl 생성할 때 사용할 IP 버전을 정의할 수 있습니다. 다음 옵션을 구성할 수 있습니다.

- IPv4
- IPv6

기본값은 IPv4입니다.

정의하려면 다음 예제를 사용합니다.

```
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**
>
> 이 설정은 CLI 플래그가 아닌 구성 파일에만 있습니다.

IPv6를 사용하는 경우 다음 요구 사항을 구성해야 합니다.

- OIDC가 활성화됨
- 관리형 추가 기능은 위와 같이 정의됩니다.
- 클러스터 버전은 => 1.21이어야 합니다.
- vpc-cni 추가 기능 버전은 => 1.10.0이어야 합니다.
- 자체 관리형 노드 그룹은 IPv6 클러스터에서 지원되지 않습니다.
- 비소유 IPv6 클러스터에서는 관리형 노드 그룹이 지원되지 않습니다.
- `vpc.nat` 및 `serviceIPv4CIDR` 필드는 ipv6 클러스터용 eksctl에서 생성되며 지원되지 않는 구성 옵션입니다.
- AutoAllocateIPv6는 IPv6와 함께 지원되지 않습니다.
- IPv6 클러스터의 경우 vpc-cni의 IAM 역할에 IPv6 모드에 필요한 IAM 정책이 연결되어 있어야 합니다.

프라이빗 네트워킹은 IPv6 IP 패밀리로도 수행할 수 있습니다. EKS 프라이빗 클러스터에 설명된 지침을 따르세요.

IAM

이 장에는 AWS IAM 작업에 대한 정보가 포함되어 있습니다.

주제:

- the section called “IAM 사용자 및 역할 관리”
  - IAM 사용자 및 역할 매핑을 관리하여 EKS 클러스터에 대한 액세스 제어
  - 클러스터 구성 파일 또는 CLI 명령을 통해 IAM 자격 증명 매핑 구성
- the section called “서비스 계정에 대한 IAM 역할”
  - 다른 AWS 서비스를 사용하는 Amazon EKS에서 실행되는 애플리케이션에 대한 세분화된 권한 관리
  - eksctl을 사용하여 IAM 역할 및 Kubernetes 서비스 계정 페어 생성 및 구성
  - EKS 클러스터에 대한 IAM OpenID Connect 공급자를 활성화하여 서비스 계정에 대한 IAM 역할 활성화
- the section called “IAM 권한 경계”
  - 권한 경계를 설정하여 IAM 엔터티(사용자 또는 역할)에 부여된 최대 권한 제어
- the section called “EKS Pod Identity 연결”
  - 권장 포드 자격 증명 연결을 사용하여 EKS 추가 기능에 대한 IAM 권한 구성
  - Kubernetes 애플리케이션이 클러스터 외부의 AWS 서비스와 연결하는 데 필요한 IAM 권한을 수신하도록 활성화
  - 여러 EKS 클러스터에서 IAM 역할 및 서비스 계정을 자동화하는 프로세스 간소화
- the section called “IAM 정책”
  - 이미지 빌더, 오토 스케일러, 외부 DNS, 인증서 관리자 등과 같은 다양한 추가 기능 정책에 대한 지원을 포함하여 EKS 노드 그룹에 대한 IAM 정책을 관리합니다.
  - 추가 권한을 위해 사용자 지정 인스턴스 역할 또는 인라인 정책을 노드 그룹에 연결합니다.
  - AmazonEKSWorkerNodePolicy 및 AmazonEKS_CNI_Policy와 같은 필수 정책이 포함되도록 AmazonEKS별 특정 AWS 관리형 정책을 노드 그룹에 연결합니다.
- the section called “최소 IAM 정책”
  - 로드 밸런서, Auto Scaling 그룹 및 CloudWatch 모니터링을 포함한 AWS EC2 리소스 관리
  - AWS CloudFormation 스택 생성 및 관리

- Amazon Elastic Kubernetes Service(EKS) 클러스터, 노드 그룹 및 IAM 역할 및 정책과 같은 관련 리소스 관리

최소 IAM 정책

이 문서에서는 eksctl의 주요 사용 사례를 실행하는 데 필요한 최소 IAM 정책을 설명합니다. 통합 테스트를 실행하는 데 사용되는 항목입니다.

> **Note**
>
> 를 자신의 <account_id> 것으로 바꿔야 합니다.

> **Note**
>
> AWS 관리형 정책은 AWS에서 생성하고 관리합니다. AWS 관리형 정책에 정의된 권한은 변경할 수 없습니다.

AmazonEC2FullAccess(AWS 관리형 정책)

AmazonEC2FullAccess 정책 정의를 확인합니다.

AWSCloudFormationFullAccess(AWS 관리형 정책)

AWSCloudFormationFullAccess 정책 정의를 봅니다.

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 권한 경계

권한 경계는 자격 증명 기반 정책이 IAM 엔터티에 부여할 수 있는 최대 권한이 설정된 고급 AWS IAM 기능입니다. 여기서 해당 엔터티는 사용자 또는 역할입니다. 엔터티에 대한 권한 경계가 설정되면 해당 엔터티는 자격 증명 기반 정책과 권한 경계 모두에서 허용되는 작업만 수행할 수 있습니다.

eksctl에서 생성한 모든 자격 증명 기반 엔터티가 해당 경계 내에 생성되도록 권한 경계를 제공할 수 있습니다. 이 예제에서는 eksctl에서 생성한 다양한 자격 증명 기반 엔터티에 권한 경계를 제공하는 방법을 보여줍니다.

```
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
>
> 역할 ARN과 권한 경계를 모두 제공할 수는 없습니다.

VPC CNI 권한 경계 설정

OIDC가 활성화된 클러스터를 생성하면 eksctl은 보안상의 이유로 VPC-CNI에 `iamserviceaccount` 대한를 자동으로 생성합니다. 권한 경계를 추가하려면 구성 파일`iamserviceaccount`에서를 수동으로 지정해야 합니다.

```
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 정책

인스턴스 역할을 노드 그룹에 연결할 수 있습니다. 노드에서 실행되는 워크로드는 노드로부터 IAM 권한을 받습니다. mroe에 대한 자세한 내용은 Amazon EC2의 IAM 역할을 참조하세요.

이 페이지에는 eksctl에서 사용할 수 있는 사전 정의된 IAM 정책 템플릿이 나열되어 있습니다. 이러한 템플릿은 사용자 지정 IAM 정책을 수동으로 생성할 필요 없이 EKS 노드에 적절한 AWS 서비스 권한을 부여하는 프로세스를 간소화합니다.

지원되는 IAM 추가 기능 정책

지원되는 모든 추가 기능 정책의 예:

```
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 정책

이 `imageBuilder` 정책은 전체 ECR(Elastic Container Registry) 액세스를 허용합니다. 이는 예를 들어 이미지를 ECR로 푸시해야 하는 CI 서버를 구축하는 데 유용합니다.

EBS 정책

이 `ebs` 정책은 새 EBS CSI(Elastic Block Store Container Storage Interface) 드라이버를 활성화합니다.

Cert Manager 정책

이 `certManager` 정책을 사용하면 DNS01 문제를 해결하기 위해 Route 53에 레코드를 추가할 수 있습니다. 자세한 내용은 여기에서 확인할 수 있습니다.

사용자 지정 인스턴스 역할 추가

이 예제에서는 다른 클러스터의 기존 IAM 인스턴스 역할을 재사용하는 노드 그룹을 생성합니다.

```
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"
```

인라인 정책 연결

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

ARN별 정책 연결

```
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**
>
> 노드 그룹에이 포함된 경우이 예제`AmazonEC2ContainerRegistryPullOnly`에서는 `AmazonEKSWorkerNodePolicy` `AmazonEKS_CNI_Policy` 및와 같은 기본 노드 정책도 포함해야 `attachPolicyARNs` 합니다.

IAM 사용자 및 역할 관리

> **Note**
>
> AWS는 ConfigMapthe section called “EKS Pod Identity 연결”에서 `aws-auth` 로 마이그레이션할 것을 제안합니다.

EKS 클러스터는 IAM 사용자 및 역할을 사용하여 클러스터에 대한 액세스를 제어합니다. 규칙은 구성 맵에서 구현됩니다.

CLI 명령을 사용하여 ConfigMap 편집

라는 이름`aws-auth`. `eksctl`는이 구성 맵을 읽고 편집하는 명령을 제공합니다.

모든 자격 증명 매핑 가져오기:

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

ARN과 일치하는 모든 자격 증명 매핑 가져오기:

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

자격 증명 매핑 생성:

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

자격 증명 매핑 삭제:

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

> **Note**
>
> 위 명령은이 `--all` 제공되지 않는 한 단일 매핑 FIFO를 삭제합니다.이 경우 일치하는 항목을 모두 제거합니다. 이 역할과 일치하는 더 많은 매핑이 발견되면 경고합니다.

계정 매핑 생성:

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

계정 매핑 삭제:

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

ClusterConfig 파일을 사용하여 ConfigMap 편집

자격 증명 매핑은 ClusterConfig에서도 지정할 수 있습니다.

```
---
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
```

서비스 계정에 대한 IAM 역할

> **Tip**
>
> `eksctl`는 EKS Pod Identity Associations를 통해 앱을 실행하는 EKS에 대한 세분화된 권한 구성을 지원합니다.

Amazon EKS는 여기에서 클러스터 운영자가 AWS IAM 역할을 Kubernetes 서비스 계정에 매핑할 수 있는 서비스 계정에 대한 역할(IRSA)]을 지원합니다.

이를 통해 EKS에서 실행되고 다른 AWS 서비스를 사용하는 앱에 대한 세분화된 권한 관리를 제공합니다. 여기에는 S3, 기타 데이터 서비스(RDS, MQ, STS, DynamoDB) 또는 AWS Load Balancer 컨트롤러 또는 ExternalDNS와 같은 Kubernetes 구성 요소를 사용하는 앱이 포함될 수 있습니다.

를 사용하여 IAM 역할 및 서비스 계정 페어를 쉽게 생성할 수 있습니다`eksctl`.

> **Note**
>
> 인스턴스 역할을 사용하고 대신 IRSA를 사용하려는 경우 두 역할을 혼합해서는 안 됩니다.

작동 방식

EKS가 노출하는 IAM OpenID Connect Provider(OIDC)를 통해 작동하며, IAM 역할은 IAM OIDC Provider(특정 EKS 클러스터에만 해당)와 바인딩될 Kubernetes 서비스 계정에 대한 참조를 참조하여

구성해야 합니다. IAM 역할이 생성되면 서비스 계정에 해당 역할의 ARN이 주석()으로 포함되어야 합니다`eks.amazonaws.com/role-arn`. 기본적으로 서비스 계정은 역할 주석을 포함하도록 생성되거나 업데이트되며 플래그를 사용하여 비활성화할 수 있습니다`--role-only`.

EKS 내에는 포드에서 사용하는 서비스 계정의 주석을 기반으로 AWS 세션 자격 증명을 역할의 포드에 각각 주입하는 승인 컨트롤러가 있습니다. 자격 증명은 `AWS_ROLE_ARN` 및 `AWS_WEB_IDENTITY_TOKEN_FILE` 환경 변수에 의해 노출됩니다. 최신 버전의 AWS SDK가 사용되는 경우(정확한 버전에 대한 자세한 내용은 여기 참조) 애플리케이션은 이러한 자격 증명을 사용합니다.

리소스 `eksctl` 이름에는 IAM 역할 및 서비스 계정 페어를 나타내는 iamserviceaccount가 있습니다.

CLI에서 사용

> **Note**
>
> 서비스 계정에 대한 IAM 역할에는 Kubernetes 버전 1.13 이상이 필요합니다.

IAM OIDC Provider는 기본적으로 활성화되어 있지 않습니다. 다음 명령을 사용하여 활성화하거나 구성 파일을 사용할 수 있습니다(아래 참조).

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

IAM OIDC 공급자가 클러스터와 연결되면 서비스 계정에 바인딩된 IAM 역할을 생성하려면 다음을 실행합니다.

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

> **Note**
>
> 둘 이상의 정책을 사용하도록 `--attach-policy-arn` 여러 번 지정할 수 있습니다.

보다 구체적으로, 다음을 실행하여 S3에 대한 읽기 전용 액세스 권한이 있는 서비스 계정을 생성할 수 있습니다.

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

기본적으로 default 네임스페이스에 생성되지만 다음과 같은 다른 네임스페이스를 지정할 수 있습니다.

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

> **Note**
>
> 네임스페이스가 아직 없는 경우 네임스페이스가 생성됩니다.

클러스터에 서비스 계정이 이미 생성된 경우(IAM 역할 제외) --override-existing-serviceaccounts 플래그를 사용해야 합니다.

를 지정하여 IAM 역할에 사용자 지정 태그 지정을 적용할 수도 있습니다--tags.

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

CloudFormation은 임의의 문자열을 포함하는 역할 이름을 생성합니다. 미리 결정된 역할 이름을 선호하는 경우 --role-name를 지정할 수 있습니다.

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

helm과 같은 다른 도구에서 서비스 계정을 생성하고 관리하는 경우를 사용하여 충돌--role-only을 방지합니다. 그러면 다른 도구가 역할 ARN 주석 유지 관리를 담당합니다. --override-existing-serviceaccounts는 roleOnly/--role-only 서비스 계정에 영향을 주지 않으며 역할은 항상 생성됩니다.

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

서비스 계정과 함께 사용할 기존 역할이 있는 경우 정책을 제공하는 대신 --attach-role-arn 플래그를 제공할 수 있습니다. 지정된 서비스 계정에서만 역할을 수임할 수 있도록 하려면 여기 관계 정책 문서를 설정해야 합니다.]

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

서비스 계정 역할 권한을 업데이트하려면를 실행할 수 있습니다`eksctl update iamserviceaccount`.

> **Note**
>
> `eksctl delete iamserviceaccount`는에서 생성하지 않은 `ServiceAccounts` 경우에도 Kubernetes를 삭제합니다`eksctl`.

구성 파일 사용

구성 파일을 `iamserviceaccounts` 사용하여를 관리하려면에서 원하는 계정을 설정하고 `iam.withOIDC: true` 나열해야 합니다`iam.serviceAccount`.

모든 명령은를 지원`--config-file`하므로 노드 그룹과 동일한 방식으로 iamserviceaccount를 관리할 수 있습니다. `eksctl create iamserviceaccount` 명령은 `--include` 및 `--exclude` 플래그를 지원합니다(이러한 작동 방식에 대한 자세한 내용은 이 섹션 참조). 또한 `eksctl delete iamserviceaccount` 명령은 `--only-missing` 도 지원하므로 노드 그룹과 동일한 방식으로 삭제를 수행할 수 있습니다.

> **Note**
>
> IAM 서비스 계정은 네임스페이스 내에서 범위가 지정됩니다. 즉, 이름이 같은 두 개의 서비스 계정이 서로 다른 네임스페이스에 존재할 수 있습니다. 따라서 서비스 계정을 , `--include --exclude` 플래그의 일부로 고유하게 정의하려면 이름 문자열을 `namespace/name` 형식으로 전달해야 합니다. 예:

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

활성화 옵션은 `cluster-autoscaler` 및와 같은 잘 알려진 사용 사례와 함께 IRSA를 정책 목록에 대한 간단한 설명`cert-manager`으로 사용하기 위해 `wellKnownPolicies` 포함되어 있습니다.

지원되는 잘 알려진 정책 및의 기타 속성`serviceAccounts`은 구성 스키마에 문서화되어 있습니다.

에서 `eksctl create cluster`다음 구성 예제를 사용합니다.

```
# 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
```

이러한 필드를 설정하지 않고 클러스터를 생성하는 경우 다음 명령을 사용하여 필요한 모든 것을 활성화할 수 있습니다.

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

추가 정보

- 서비스 계정에 대한 세분화된 IAM 역할 소개
- EKS 사용 설명서 - 서비스 계정에 대한 IAM 역할
- IAM 사용자 및 역할을 Kubernetes RBAC 역할에 매핑

EKS Pod Identity 연결

AWS EKS는 클러스터 관리자가 클러스터 외부의 AWS 서비스와 연결하는 데 필요한 IAM 권한을 받도록 Kubernetes 애플리케이션을 구성하기 위해 Pod Identity Association이라는 새로운 향상된 메커니즘을 도입했습니다. 포드 자격 증명 연결은 IRSA를 활용하지만 EKS API를 통해 직접 구성할 수 있으므로 IAM API를 전혀 사용할 필요가 없습니다.

따라서 IAM 역할은 더 이상 OIDC 공급자를 참조할 필요가 없으므로 더 이상 단일 클러스터에 연결되지 않습니다. 즉, 이제 새 클러스터가 생성될 때마다 역할 신뢰 정책을 업데이트할 필요 없이 여러 EKS 클러스터에서 IAM 역할을 사용할 수 있습니다. 따라서 역할 중복이 필요하지 않으며 IRSA를 자동화하는 프로세스가 간소화됩니다.

사전 조건

백그라운드에서 포드 자격 증명 연결의 구현은 작업자 노드에서 에이전트를 데몬 세트로 실행하고 있습니다. 클러스터에서 사전 필수 에이전트를 실행하기 위해 EKS는 EKS Pod Identity Agent라는 새로운 추가 기능을 제공합니다. 따라서 포드 자격 증명 연결(일반적으로 및)을 생성하려면 클러스터에

eks-pod-identity-agent 추가 기능이 사전 설치되어 eksctl있어야 합니다. 이 추가 기능은 지원되는 다른 추가 기능과 동일한 방식으로 eksctl를 사용하여 생성할 수 있습니다.

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

또한 포드 자격 증명 연결을 생성할 때 기존 IAM 역할을 사용하는 경우 새로 도입된 EKS 서비스 보안 주체()를 신뢰하도록 역할을 구성해야 합니다pods.eks.amazonaws.com. IAM 신뢰 정책의 예는 아래에서 확인할 수 있습니다.

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

대신 생성 명령에 기존 역할의 ARN을 제공하지 않으면 eksctl는 백그라운드에서 ARN을 생성하고 위의 신뢰 정책을 구성합니다.

포드 자격 증명 연결 생성

포드 자격 증명 연결을 조작하기 위해 eksctl는 아래에 iam.podIdentityAssociations다음과 같은 새 필드를 추가했습니다.

```
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
```

전체 예제는 pod-identity-associations.yaml을 참조하세요.

> **Note**
>
> `permissionPolicy`가 인라인 정책 문서로 사용되는 것 외에도 다른 모든 필드에는 CLI 플래그가 있습니다.

다음과 같은 방법으로 포드 자격 증명 연결을 생성할 수 있습니다. 클러스터를 생성하는 동안 원하는 포드 자격 증명 연결을 구성 파일의 일부로 지정하고 실행하여 다음을 수행합니다.

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

구성 파일을 사용한 클러스터 생성 후 예:

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

CLI 플래그를 사용한 OR 예:

```
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**
>
> 한 번에 하나의 IAM 역할만 서비스 계정에 연결할 수 있습니다. 따라서 동일한 서비스 계정에 대한 두 번째 포드 자격 증명 연결을 생성하려고 하면 오류가 발생합니다.

포드 자격 증명 연결 가져오기

특정 클러스터에 대한 모든 포드 자격 증명 연결을 검색하려면 다음 명령 중 하나를 실행합니다.

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

또는

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

또한 지정된 네임스페이스 내에서 포드 자격 증명 연결만 검색하려면 `--namespace` 플래그를 사용합니다. 예:

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

마지막으로 특정 K8s 서비스 계정에 해당하는 단일 연결을 검색하려면 위의 명령`--service-account-name`에 대한 도 포함합니다.

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

포드 자격 증명 연결 업데이트

하나 이상의 포드 자격 증명 연결의 IAM 역할을 업데이트하려면 새 `roleARN(s)`를 구성 파일에 전달합니다. 예:

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

및 실행:

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

OR(단일 연결을 업데이트하는 경우)은 CLI 플래그를 `--role-arn` 통해 새를 전달합니다.

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

포드 자격 증명 연결 삭제

하나 이상의 포드 자격 증명 연결을 삭제하려면 구성 파일에 `namespace(s)` 및 `serviceAccountName(s)`를 전달합니다. 예:

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

및 실행:

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

OR(단일 연결을 삭제하려면)은 CLI 플래그를 `--service-account-name` 통해 `--namespace` 및를 전달합니다.

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

포드 자격 증명 연결에 대한 EKS 추가 기능 지원

EKS 추가 기능은 EKS Pod Identity Associations를 통한 IAM 권한 수신도 지원합니다. 구성 파일은 `addon.podIdentityAssociations`, `addonsConfig.autoApplyPodIdentityAssociations` 및를 구성할 수 있는 세 개의 필드를 표시합니다`addon.useDefaultPodIdentityAssociations`. 를 사용하여 원하는 포드 자격 증명 연결을 명시적으로 구성`addon.podIdentityAssociations`하거나 `addonsConfig.autoApplyPodIdentityAssociations` 또는를 사용하여 권장 포드 자격 증명 구성을 `eksctl` 자동으로 확인(및 적용)할 수 있습니다`addon.useDefaultPodIdentityAssociations`.

> **Note**
>
> 모든 EKS 추가 기능이 시작 시 포드 자격 증명 연결을 지원하는 것은 아닙니다. 이 경우 IRSA 설정을 사용하여 필요한 IAM 권한을 계속 제공해야 합니다.

IAM 권한을 사용하여 추가 기능 생성

IAM 권한이 필요한 추가 기능을 생성할 때 `eksctl`는 먼저 포드 자격 증명 연결 또는 IRSA 설정이 구성 파일의 일부로 명시적으로 구성되어 있는지 확인하고, 구성되어 있는 경우이 중 하나를 사용하여 추가 기능에 대한 권한을 구성합니다. 예:

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

및 실행

```
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**
>
> 포드 자격 증명과 IRSA를 동시에 설정하는 것은 허용되지 않으며 검증 오류가 발생합니다.

포드 ID를 지원하는 EKS 추가 기능의 경우는 추가 기능 생성 시 권장 IAM 권한을 자동으로 구성하는 옵션을 `eksctl` 제공합니다. 이는 구성 파일`addonsConfig.autoApplyPodIdentityAssociations: true`에서를 설정하면 가능합니다. 예:

```
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
```

및 실행

```
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
```

이와 마찬가지로 CLI 플래그를 통해 동일한 작업을 수행할 수 있습니다. 예:

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

권장 IAM 정책과 함께 포드 자격 증명을 사용하도록 기존 추가 기능을 마이그레이션하려면

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

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

IAM 권한으로 추가 기능 업데이트

추가 기능을 업데이트할 때를 지정addon.PodIdentityAssociations하면 업데이트 작업이 완료된 후 추가 기능의 상태에 대한 단일 실제 소스를 나타냅니다. 백그라운드에서는 원하는 상태를 달성하기 위해 다양한 유형의 작업이 수행됩니다. 즉,

- 구성 파일에는 있지만 클러스터에는 없는 포드 ID 생성
- 연결된 IAM 리소스와 함께 구성 파일에서 제거된 기존 포드 ID 삭제
- 구성 파일에도 있고 IAM 권한 집합이 변경된 기존 포드 자격 증명 업데이트

> **Note**
>
> EKS 추가 기능이 소유한 포드 자격 증명 연결의 수명 주기는 EKS 추가 기능 API에서 직접 처리합니다.

Amazon EKS 추가 기능과 함께 사용되는 연결에는 `eksctl update podidentityassociation` (IAM 권한을 업데이트하기 위해) 또는 `eksctl delete podidentityassociations` (연결을 제거하기 위해)를 사용할 수 없습니다. 대신 `eksctl update addon` 또는 `eksctl delete addon`를 사용해야 합니다.

추가 기능에 대한 초기 포드 자격 증명 구성을 분석하는 것부터 위의 예를 살펴보겠습니다.

```
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"
    }
]
```

이제 아래 구성을 사용합니다.

```
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
```

및 실행

```
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
```

이제 포드 자격 증명 구성이 올바르게 업데이트되었는지 확인합니다.

```
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"
    }
]
```

추가 기능에서 모든 포드 자격 증명 연결을 제거하려면를 []로 명시적으로 설정해야 addon.PodIdentityAssociations 합니다. 예:

```
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: []
```

및 실행

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

IAM 권한이 있는 추가 기능 삭제

추가 기능을 삭제하면 추가 기능과 연결된 모든 포드 자격 증명도 제거됩니다. 클러스터를 삭제하면 모든 추가 기능에 대해 동일한 효과가 적용됩니다. 에서 생성한 포드 자격 증명에 대한 모든 IAM 역할은 정상적으로 eksctl삭제됩니다.

기존 iamserviceaccounts 및 추가 기능을 포드 자격 증명 연결로 마이그레이션

서비스 계정의 기존 IAM 역할을 포드 자격 증명 연결로 마이그레이션하는 `eksctl` utils 명령이 있습니다.

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

백그라운드에서 명령은 다음 단계를 적용합니다.

- 클러스터에서 아직 활성화되지 않은 경우 `eks-pod-identity-agent` 추가 기능 설치
- iamserviceaccounts와 연결된 모든 IAM 역할 식별
- 포드 자격 증명 연결을 지원하는 EKS 추가 기능과 연결된 모든 IAM 역할 식별
- 새 EKS 서비스 보안 주체를 가리키는 신뢰할 수 있는 추가 엔터티를 사용하여 식별된 모든 역할의 IAM 신뢰 정책 업데이트(및 선택적으로 기존 OIDC 공급자 신뢰 관계 제거)
- iamserviceaccounts와 연결된 필터링된 역할에 대한 포드 자격 증명 연결 생성
- 포드 ID로 EKS 추가 기능 업데이트(EKS API는 백그라운드에서 포드 ID를 생성함)

`--approve` 플래그 없이 명령을 실행하면 위의 단계를 반영하는 작업 세트로 구성된 계획만 출력됩니다. 예:

```
[#]  (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
```

기존 OIDC 공급자 신뢰 관계는 항상 EKS 추가 기능과 연결된 IAM 역할에서 제거됩니다. 또한 iamserviceaccounts와 연결된 IAM 역할에서 기존 OIDC 공급자 신뢰 관계를 제거하려면 `--remove-oidc-provider-trust-relationship` 플래그를 사용하여 명령을 실행합니다. 예:

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

교차 계정 포드 자격 증명 지원

eksctl은 EKS Pod Identity 교차 계정 액세스를 지원합니다. 이 기능을 사용하면 EKS 클러스터에서 실행되는 포드가 다른 AWS 계정의 AWS 리소스에 액세스할 수 있습니다.

사용법

교차 계정 액세스와 포드 자격 증명 연결을 생성하려면 먼저 소스 AWS 계정(클러스터 사용)에서 대상 AWS 계정(클러스터가 액세스할 수 있는 리소스 사용)으로의 액세스를 허용하는 IAM 역할 및 정책을 설정합니다. 이에 대한 예는 "Amazon EKS Pod Identity는 교차 계정 액세스를 간소화합니다."를 참조하세요.

각 계정에 IAM 역할이 구성되면 eksctl을 사용하여 포드 자격 증명 연결을 생성합니다.

```
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
```

추가 참조

포드 ID에 대한 공식 AWS Userdocs for EKS 추가 기능 지원

포드 자격 증명 연결에 대한 공식 AWS 블로그 게시물

포드 자격 증명 연결을 위한 공식 AWS 사용자 설명서

배포 옵션

이 장에서는 eksctl을 사용하여 대체 환경에 배포된 EKS 클러스터를 관리하는 방법을 다룹니다.

EKS 배포 옵션에 대한 가장 정확한 정보는 EKS 사용 설명서의 클라우드 및 온프레미스 환경에 Amazon EKS 클러스터 배포를 참조하세요.

주제:

- the section called “EKS Anywhere”
  - Amazon EKS Anywhere 클러스터에서 eksctl을 사용합니다.
  - Amazon EKS Anywhere는 온프레미스 및 엣지에서 Kubernetes를 더 쉽게 실행하고 관리할 수 있도록 AWS에서 구축한 컨테이너 관리 소프트웨어입니다.
- the section called “AWS Outposts 지원”
  - AWS Outposts에서 EKS 클러스터와 함께 eksctl을 사용합니다.
  - AWS Outposts는 진정으로 일관된 하이브리드 경험을 위해 거의 모든 온프레미스 또는 엣지 로케이션에 AWS 인프라와 서비스를 제공하는 완전관리형 솔루션 제품군입니다.
  - eksctl에서 AWS Outposts 지원을 사용하면 AWS Outposts에서 로컬로 실행되는 EKS 컨트롤 플레인 및 작업자 노드를 포함한 전체 Kubernetes 클러스터로 로컬 클러스터를 생성할 수 있습니다.
- the section called “EKS 하이브리드 노드”
  - AWS 클라우드에서 사용하는 것과 동일한 AWS EKS 클러스터, 기능 및 도구를 사용하여 고객 관리형 인프라에서 온프레미스 및 엣지 애플리케이션을 실행합니다.

EKS Anywhere

`eksctl`는 하위 명령 `EKS Anywhere`를 사용하여 라는 AWS 기능에 대한 액세스를 제공합니다`eksctl anywhere`. 이를 위해서는에 있는 `eksctl-anywhere` 바이너리가 필요합니다`PATH`. Install eksctl-anywhere에 설명된 지침에 따라 설치하십시오.

완료되면 다음을 실행하여 어디서나 명령을 실행합니다.

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

EKS Anywhere에 대한 자세한 내용은 EKS Anywhere 웹 사이트를 참조하십시오.

AWS Outposts 지원

> **Warning**
>
> EKS 관리형 노드 그룹은 Outposts에서 지원되지 않습니다.

기존 클러스터를 AWS Outposts로 확장

다음과 같이 `nodeGroup.outpostARN` 새 노드 그룹이 Outposts에 노드 그룹을 생성하도록 설정하여 AWS 리전에서 실행되는 기존 EKS 클러스터를 AWS Outposts로 확장할 수 있습니다.

```
# 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
```

이 설정에서 EKS 컨트롤 플레인은 AWS 리전에서 실행되는 반면 `outpostARN` 설정된 노드 그룹은 지정된 Outpost에서 실행됩니다. Outposts에서 노드 그룹을 처음 생성하는 경우 eksctl은 지정된 Outpost에 서브넷을 생성하여 VPC를 확장합니다. 이러한 서브넷은가 `outpostARN` 설정된 노드 그룹을 생성하는 데 사용됩니다.

기존 VPC가 있는 고객은 다음과 `nodeGroup.subnets`같이 Outposts에서 서브넷을 생성하고에 전달해야 합니다.

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

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

vpc:
  id: vpc-1234
    subnets:
      private:
        outpost-subnet-1:
          id: subnet-1234

nodeGroups:
  # Nodegroup will be created in an AWS region.
  - name: ng

  # Nodegroup will be created on the specified Outpost.
  - name: outpost-ng
    privateNetworking: true
    # Subnet IDs for subnets created on Outpost.
    subnets: [subnet-5678]
    outpostARN: "arn:aws:outposts:us-west-2:1234:outpost/op-1234"
```

AWS Outposts에서 로컬 클러스터 생성

> **Note**
>
> 로컬 클러스터는 Outpost 랙만 지원합니다.

> **Note**
>
> 컨트롤 플레인이 Outposts에 있는 경우 노드 그룹에는 Amazon Linux 2만 지원됩니다. Outposts의 노드 그룹에는 EBS gp2 볼륨 유형만 지원됩니다.

eksctl에서 AWS Outposts 지원을 사용하면 AWS Outposts에서 로컬로 실행되는 EKS 컨트롤 플레인 및 작업자 노드를 포함한 전체 Kubernetes 클러스터로 로컬 클러스터를 생성할 수 있습니다. 고객은 AWS Outposts에서 로컬로 실행되는 EKS 컨트롤 플레인과 작업자 노드를 모두 사용하여 로컬 클러스터를 생성하거나, Outposts에서 작업자 노드를 생성하여 AWS 리전에서 실행되는 기존 EKS 클러스터를 AWS Outposts로 확장할 수 있습니다.

AWS Outposts에서 EKS 컨트롤 플레인 및 노드 그룹을 생성하려면 다음과 같이 `outpost.controlPlaneOutpostARN`를 Outpost ARN으로 설정합니다.

```
# 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
```

이렇게 하면 eksctl이 지정된 Outpost에 EKS 컨트롤 플레인과 서브넷을 생성하도록 지시합니다. Outpost 랙은 단일 가용 영역에 존재하므로 eksctl은 퍼블릭 및 프라이빗 서브넷을 하나만 생성합니다. eksctl은 생성된 VPC를 로컬 게이트웨이와 연결하지 않으므로 eksctl은 API 서버에 대한 연결이 부족하고 노드 그룹을 생성할 수 없습니다. 따라서 클러스터 생성 중에에 노드 그룹이 `ClusterConfig` 포함된 경우 다음과 `--without-nodegroup`같이 명령을 로 실행해야 합니다.

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

API 서버에 연결할 수 있도록 클러스터를 생성한 후 eksctl에서 생성한 VPC를 로컬 게이트웨이와 연결하는 것은 고객의 책임입니다. 이 단계 후에는를 사용하여 노드 그룹을 생성할 수 있습니다`eksctl create nodegroup`.

선택적으로의 컨트롤 플레인 노드 `outpost.controlPlaneInstanceType` 또는의 노드 그룹에 대한 인스턴스 유형을 지정할 수 있지만 `nodeGroup.instanceType`인스턴스 유형이 Outpost에 있어

야 합니다. 그렇지 않으면 eksctl이 오류를 반환합니다. 기본적으로 eksctl은 컨트롤 플레인 노드 및 노드 그룹에 대해 Outpost에서 사용 가능한 가장 작은 인스턴스 유형을 선택하려고 시도합니다.

컨트롤 플레인이 Outposts에 있으면 해당 Outpost에 노드 그룹이 생성됩니다. 선택적으로의 노드 그룹에 대한 Outpost ARN을 지정할 수 `nodeGroup.outpostARN` 있지만 컨트롤 플레인의 Outpost ARN과 일치해야 합니다.

```
# 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
```

기존 VPC

기존 VPC가 있는 고객은 다음과 `vpc.subnets`같이에서 서브넷 구성을 지정하여 AWS Outposts에서 로컬 클러스터를 생성할 수 있습니다.

```
# 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
```

서브넷은에 지정된 Outpost에 있어야 합니다. `outpost.controlPlaneOutpostARN` 그렇지 않으면 eksctl이 오류를 반환합니다. 서브넷의 로컬 게이트웨이에 액세스할 수 있거나 VPC 리소스에 연결되어 있는 경우 클러스터 생성 중에 노드 그룹을 지정할 수도 있습니다.

로컬 클러스터에서 지원되지 않는 기능

- 추가 기능
- 서비스 계정에 대한 IAM 역할

- IPv6
- 자격 증명 공급자
- Fargate
- KMS 암호화
- 로컬 영역
- Karpenter
- 인스턴스 선택기
- 가용 영역은 기본적으로 Outpost 가용 영역으로 지정되므로 지정할 수 없습니다.
- `vpc.publicAccessCIDRs`, `vpc.autoAllocateIPv6`은(는) 지원되지 않습니다.
- 로컬 클러스터는 프라이빗 전용 엔드포인트 액세스로만 생성할 수 있으므로 API 서버에 대한 퍼블릭 엔드포인트 액세스는 지원되지 않습니다.

추가 정보

- AWS Outposts의 Amazon EKS
- AWS Outposts의 Amazon EKS용 로컬 클러스터
- 로컬 클러스터 생성
- Outpost에서 자체 관리형 Amazon Linux 노드 시작

보안

`eksctl`는 EKS 클러스터의 보안을 개선할 수 있는 몇 가지 옵션을 제공합니다.

`withOIDC`

를 활성화`withOIDC`하여 Amazon CNI 플러그인에 대한 IRSA를 자동으로 생성하고 클러스터의 노드에 부여된 권한을 제한하는 대신 CNI 서비스 계정에만 필요한 권한을 부여합니다.

배경은 이 AWS 설명서에 설명되어 있습니다.

`disablePodIMDS`

관리형 및 비관리형 노드 그룹의 경우 `disablePodIMDS` 옵션을 사용하면이 노드 그룹에서 실행 중인 모든 비 호스트 네트워킹 포드가 IMDS 요청을 할 수 없습니다.

> **Note**
>
> 와 함께 사용할 수 없습니다`withAddonPolicies`.

EKS 클러스터에 대한 KMS Envelope 암호화

> **Note**
>
> Amazon Elastic Kubernetes Service(Amazon EKS)는 Kubernetes 버전 1.28 이상을 실행하는 EKS 클러스터의 모든 Kubernetes API 데이터에 기본 봉투 암호화를 제공합니다. 자세한 내용은 EKS 사용 설명서의 모든 Kubernetes API 데이터에 대한 기본 봉투 암호화를 참조하세요.

EKS는 AWS KMS 키를 사용하여 EKS에 저장된 Kubernetes 보안 암호의 봉투 암호화를 제공합니다. 봉투 암호화는 Kubernetes 클러스터 내에 저장된 애플리케이션 보안 암호 또는 사용자 데이터에 대한 고객 관리형 암호화 계층을 추가합니다.

이전에 Amazon EKS는 클러스터 생성 중에만 KMS 키를 사용한 봉투 암호화 활성화를 지원했습니다. 이제 Amazon EKS 클러스터에 대해 언제든지 봉투 암호화를 활성화할 수 있습니다.

AWS 컨테이너 블로그에서 defense-in-depth를 위한 EKS 암호화 공급자 지원 사용에 대해 자세히 알아보세요.

KMS 암호화가 활성화된 클러스터 생성

```
# 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
```

기존 클러스터에서 KMS 암호화 활성화

아직 활성화되지 않은 클러스터에서 KMS 암호화를 활성화하려면를 실행합니다.

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

또는 구성 파일 없음:

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

EKS 클러스터에서 KMS 암호화를 활성화하는 것 외에도 eksctl은 주석 로 새 KMS 키를 업데이트하여 기존 Kubernetes 보안 암호를 모두 다시 암호화합니다`eksctl.io/kms-encryption-timestamp`. 이 동작은 다음과 `--encrypt-existing-secrets=false`같이를 전달하여 비활성화할 수 있습니다.

```
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>
```

클러스터에 이미 KMS 암호화가 활성화된 경우 eksctl은 기존 보안 암호를 모두 다시 암호화합니다.

> **Note**
>
> KMS 암호화가 활성화되면 다른 KMS 키를 사용하도록 비활성화하거나 업데이트할 수 없습니다.

문제 해결

이 주제에는 Eksctl의 일반적인 오류를 해결하는 방법에 대한 지침이 포함되어 있습니다.

스택 생성 실패

`--cfn-disable-rollback` 플래그를 사용하여 Cloudformation이 실패한 스택을 롤백하지 못하도록 하여 디버깅을 더 쉽게 할 수 있습니다.

서브넷 ID "subnet-11111111"은 "subnet-22222222"과 동일하지 않습니다.

다음과 같이 VPC의 서브넷을 지정하는 구성 파일이 있습니다.

```
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: []
```

오류는 지정된 서브넷이 올바른 가용 영역에 배치되지 않음을 `subnet ID "subnet-11111111" is not the same as "subnet-22222222"` 의미합니다. 각 가용 영역에 적합한 서브넷 ID인 AWS 콘솔을 확인합니다.

이 예제에서 VPC에 대한 올바른 구성은 다음과 같습니다.

```
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}
```

삭제 문제

삭제가 작동하지 않거나 삭제 `--wait` 시 추가를 잊어버린 경우 amazon의 다른 도구를 사용하여 클라우드포밍 스택을 삭제해야 할 수 있습니다. 이는 gui 또는 aws cli를 통해 수행할 수 있습니다.

권한 부여 오류와 함께 kubectl 로그 및 kubectl 실행 실패

노드가 프라이빗 서브넷에 배포되고 `kubectl logs` `kubectl run` 실패하고 다음과 같은 오류가 발생하는 경우:

```
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)
```

그런 다음 enableDnsHostnames를 설정해야 할 수 있습니다. 자세한 내용은 이 문제에서 확인할 수 있습니다.

공지 사항

이 주제에서는 새로운 Eksctl 기능의 과거 발표를 다룹니다.

관리형 노드 그룹 기본값

eksctl v0.58.0부터 eksctl은 `eksctl create cluster` 및에 대해 `ClusterConfig` 파일이 지정되지 않은 경우 기본적으로 관리형 노드 그룹을 생성합니다`eksctl create nodegroup`. 자체 관리형 노드 그룹을 생성하려면를 전달합니다`--managed=false`. Windows 노드 그룹과 같은 관리형 노드 그룹에서 지원되지 않는 기능이 사용 중인 경우 구성 파일을 사용하지 않는 스크립트가 중단될 수 있습니다. 이 문제를 해결하려면를 전달`--managed=false`하거나 자체 관리형 노드 그룹을 생성하는 `nodeGroups` 필드를 사용하여 `ClusterConfig` 파일에 노드 그룹 구성을 지정합니다.

사용자 지정 AMIs

이 변경 사항은 Breaking: overrideBootstrapCommand 곧 발표되었습니다. 이제 이 PR을 전달합니다. 부트스트랩 스크립트가 없거나 부분 부트스트랩 스크립트가 있는 사용자 지정 AMIs를 지원하지 않기로 결정한 이유에 대해 첨부된 문제를 주의 깊게 읽어 보십시오.

여전히 도우미를 제공합니다! 마이그레이션은 그다지 어렵지 않을 것입니다.는 `eksctl` 여전히 스크립트를 제공하며,이 스크립트는 소싱될 때 몇 가지 유용한 환경 속성과 설정을 내보냅니다. 이 스크립트는 여기에 있습니다.

사용 가능한 환경 속성은 다음과 같습니다.

```
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
```

가 실패eksctl하지 않도록 재정의할 때 사용해야 하는 최소는 레이블입니다! eksctl는 노드에 있는 특정 레이블 집합에 의존하므로 노드를 찾을 수 있습니다. 재정의를 정의할 때이 베어 최소 재정의 명령을 제공하세요.

```
    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}"
```

아웃바운드 인터넷에 액세스할 수 없는 노드 그룹의 경우 다음과 같이 부트스트랩 스크립트--b64-cluster-ca에 --apiserver-endpoint 및를 제공해야 합니다.

```
    overrideBootstrapCommand: |
      #!/bin/bash

      source /var/lib/cloud/scripts/eksctl/bootstrap.helper.sh

      # Note "--node-labels=${NODE_LABELS}" needs the above helper sourced to work,
 otherwise will have to be defined manually.
      /etc/eks/bootstrap.sh ${CLUSTER_NAME} --container-runtime containerd --kubelet-
extra-args "--node-labels=${NODE_LABELS}" \
        --apiserver-endpoint ${API_SERVER_URL} --b64-cluster-ca ${B64_CLUSTER_CA}
```

`--node-labels' 설정을 기록해 둡니다. 정의되지 않은 경우 노드는 클러스터에 조인되지만 노드가가 될 때까지 기다리eksctl는 마지막 단계에서 시간이 초과됩니다Ready. 레이블이 인 노드에 대한 Kubernetes 조회를 수행하고 있습니다alpha.eksctl.io/nodegroup-name=<cluster-name>. 이는 비관리형 노드 그룹에만 해당됩니다. 관리형의 경우 다른 레이블을 사용하고 있습니다.

이 오버헤드를 방지하기 위해 관리형 노드 그룹으로 전환할 수 있는 경우 이제이 작업을 수행할 시간이 되었습니다. 모든 재정의가 훨씬 쉬워집니다.

기계 번역으로 제공되는 번역입니다. 제공된 번역과 원본 영어의 내용이 상충하는 경우에는 영어 버전이 우선합니다.