Amazon EKS

User Guide

nEKS

Amazon EKS: User Guide

Table of Contents

What is Amazon EKS?

Amazon Elastic Kubernetes Service (Amazon EKS) is a managed service that makes it easy for you to run Kubernetes on AWS without needing to stand up or maintain your own Kubernetes control plane. Kubernetes is an open-source system for automating the deployment, scaling, and management of containerized applications.

Amazon EKS runs Kubernetes control plane instances across multiple Availability Zone to ensure high availability. Amazon EKS automatically detects and replaces unhealthy control plane instances, and it provides automated version upgrades and patching for them.

Amazon EKS is also integrated with many AWS services to provide scalability and security for your applications, including the following:

- Amazon ECR for container images
- Elastic Load Balancing for load distribution
- IAM for authentication
- Amazon VPC for isolation

Amazon EKS runs up-to-date versions of the open-source Kubernetes software, so you can use all the existing plugins and tooling from the Kubernetes community. Applications running on Amazon EKS are fully compatible with applications running on any standard Kubernetes environment, whether running in on-premises data centers or public clouds. This means that you can easily migrate any standard Kubernetes application to Amazon EKS without any code modification required.

Amazon EKS control plane architecture

Amazon EKS runs a single tenant Kubernetes control plane for each cluster, and control plane infrastructure is not shared across clusters or AWS accounts.

This control plane consists of at least two API server nodes and three `etcd` nodes that run across three Availability Zones within a Region. Amazon EKS automatically detects and replaces unhealthy control plane instances, restarting them across the Availability Zones within the Region as needed. Amazon EKS leverages the architecture of AWS Regions in order to maintain high availability. Because of this, Amazon EKS is able to offer an SLA for API server endpoint availability.

Amazon EKS uses Amazon VPC network policies to restrict traffic between control plane components to within a single cluster. Control plane components for a cluster cannot view or receive communication from other clusters or other AWS accounts, except as authorized with Kubernetes RBAC policies.

This secure and highly-available configuration makes Amazon EKS reliable and recommended for production workloads.

How does Amazon EKS work?
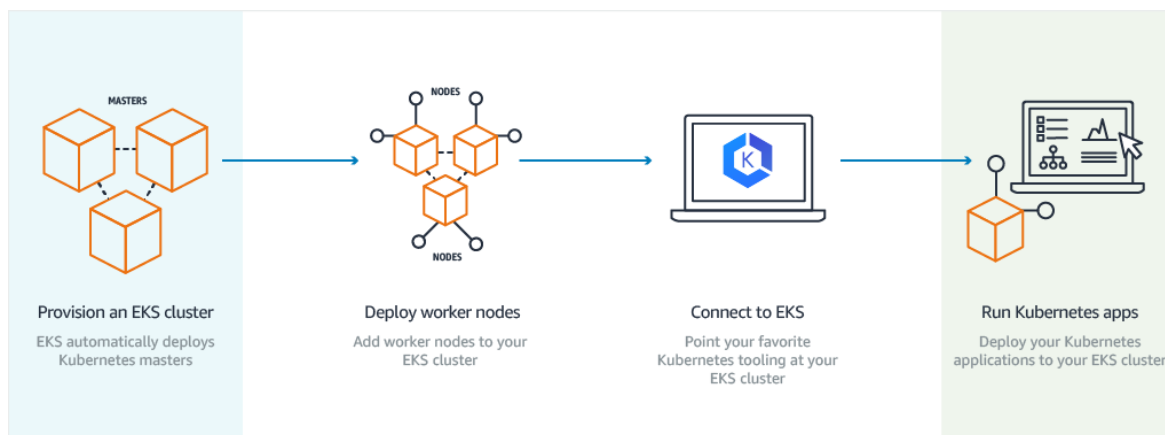
Getting started with Amazon EKS is easy:

1. First, create an Amazon EKS cluster in the AWS Management Console or with the AWS CLI or one of the AWS SDKs.
2. Then, launch worker nodes that register with the Amazon EKS cluster. We provide you with an AWS CloudFormation template that automatically configures your nodes.
3. When your cluster is ready, you can configure your favorite Kubernetes tools (such as **kubectl**) to communicate with your cluster.
4. Deploy and manage applications on your Amazon EKS cluster the same way that you would with any other Kubernetes environment.

For more information about creating your required resources and your first Amazon EKS cluster, see .

Getting started with Amazon EKS

There are two getting started guides available for creating a new Kubernetes cluster with worker nodes in Amazon EKS:

- Getting started with `eksctl` (p. 3): This getting started guide helps you to install all of the required resources to get started with Amazon EKS using `eksctl`, a simple command line utility for creating and managing Kubernetes clusters on Amazon EKS. At the end of this tutorial, you will have a running Amazon EKS cluster with worker nodes, and the `kubectl` command line utility will be configured to use your new cluster. This is the fastest and simplest way to get started with Amazon EKS.
- Getting started with the AWS Management Console (p. 10): This getting started guide helps you to create all of the required resources to get started with Amazon EKS in the AWS Management Console. In this guide, you manually create each resource in the Amazon EKS or AWS CloudFormation consoles, and the workflow described here gives you complete visibility into how each resource is created and how they interact with each other.

Getting started with `eksctl`

This getting started guide helps you to install all of the required resources to get started with Amazon EKS using `eksctl`, a simple command line utility for creating and managing Kubernetes clusters on Amazon EKS. At the end of this tutorial, you will have a running Amazon EKS cluster with a managed node group, and the `kubectl` command line utility will be configured to use your new cluster.

Prerequisites

This section helps you to install and configure the binaries you need to create and manage an Amazon EKS cluster.

Install the latest AWS CLI

To use `kubectl` with your Amazon EKS clusters, you must install a binary that can create the required client security token for cluster API server communication. The **aws eks get-token** command, available in version 1.18.49 or later of the AWS CLI, supports client security token creation. To install or upgrade the AWS CLI, see Installing the AWS command line interface in the *AWS Command Line Interface User Guide*.

If you already have pip and a supported version of Python, you can install or upgrade the AWS CLI with the following command:

```
pip install awscli --upgrade --user
```

> **Note**
> Your system's Python version must be 2.7.9 or later. Otherwise, you receive `hostname doesn't match` errors with AWS CLI calls to Amazon EKS. For more information, see What are "hostname doesn't match" errors? in the Python Requests FAQ.

For more information about other methods of installing or upgrading the AWS CLI for your platform, see the following topics in the *AWS Command Line Interface User Guide*.

- Install the AWS Command Line Interface on macOS
- Install the AWS Command Line Interface on Linux
- Install the AWS Command Line Interface on Microsoft Windows

If you are unable to install version 1.18.49 or later of the AWS CLI on your system, you must ensure that the AWS IAM Authenticator for Kubernetes is installed on your system. For more information, see Installing `aws-iam-authenticator` (p. 187).

Configure your AWS CLI credentials

Both `eksctl` and the AWS CLI require that you have AWS credentials configured in your environment. The **aws configure** command is the fastest way to set up your AWS CLI installation for general use.

```
$ aws configure
AWS Access Key ID [None]: AKIAIOSFODNN7EXAMPLE
AWS Secret Access Key [None]: wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY
Default region name [None]: region-code
Default output format [None]: json
```

When you type this command, the AWS CLI prompts you for four pieces of information: access key, secret access key, AWS Region, and output format. This information is stored in a profile (a collection of settings) named `default`. This profile is used unless you specify another one.

For more information, see Configuring the AWS CLI in the *AWS Command Line Interface User Guide*.

Install `eksctl`

This section helps you to install the `eksctl` command line utility. For more information, see the https://eksctl.io/.

Choose the tab below that best represents your client setup.

macOS

To install or upgrade `eksctl` on macOS using homebrew

The easiest way to get started with Amazon EKS and macOS is by installing `eksctl` with Homebrew. The `eksctl` Homebrew recipe installs `eksctl` and any other dependencies that are required for Amazon EKS, such as `kubectl` and the `aws-iam-authenticator`.

1. If you do not already have Homebrew installed on macOS, install it with the following command.

   ```
   /bin/bash -c "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/
   master/install.sh)"
   ```

2. Install the Weaveworks Homebrew tap.

   ```
   brew tap weaveworks/tap
   ```

3. Install or upgrade `eksctl`.

   - Install `eksctl` with the following command:

     ```
     brew install weaveworks/tap/eksctl
     ```

- If `eksctl` is already installed, run the following command to upgrade:

```
brew upgrade eksctl && brew link --overwrite eksctl
```

4. Test that your installation was successful with the following command.

```
eksctl version
```

> **Note**
> The `GitTag` version should be at least `0.19.0`. If not, check your terminal output for any installation or upgrade errors, or manually download an archive of the release from https://github.com/weaveworks/eksctl/releases/download/0.19.0/eksctl_Darwin_amd64.tar.gz, extract `eksctl`, and then execute it.

Linux

To install or upgrade `eksctl` on Linux using `curl`

1. Download and extract the latest release of `eksctl` with the following command.

```
curl --silent --location "https://github.com/weaveworks/eksctl/releases/latest/download/eksctl_$(uname -s)_amd64.tar.gz" | tar xz -C /tmp
```

2. Move the extracted binary to `/usr/local/bin`.

```
sudo mv /tmp/eksctl /usr/local/bin
```

3. Test that your installation was successful with the following command.

```
eksctl version
```

> **Note**
> The `GitTag` version should be at least `0.19.0`. If not, check your terminal output for any installation or upgrade errors, or replace the address in step 1 with `https://github.com/weaveworks/eksctl/releases/download/0.19.0/eksctl_$(uname -s)_amd64.tar.gz` and complete steps 1-3 again.

Windows

To install or upgrade `eksctl` on Windows using chocolatey

1. If you do not already have Chocolatey installed on your Windows system, see Installing chocolatey.
2. Install or upgrade `eksctl` and the `aws-iam-authenticator`.

   - Install the binaries with the following command:

```
chocolatey install -y eksctl aws-iam-authenticator
```

   - If they are already installed, run the following command to upgrade:

```
chocolatey upgrade -y eksctl aws-iam-authenticator
```

3. Test that your installation was successful with the following command.

```
eksctl version
```

> **Note**
> The `GitTag` version should be at least `0.19.0`. If not, check your terminal output
> for any installation or upgrade errors, or manually download an archive of the
> release from https://github.com/weaveworks/eksctl/releases/download/0.19.0/
> eksctl_Windows_amd64.zip, extract `eksctl`, and then execute it.

Install and configure **kubectl** for Amazon EKS

Kubernetes uses the `kubectl` command-line utility for communicating with the cluster API server.

> **Note**
> If you used the preceding Homebrew instructions to install `eksctl` on macOS, then `kubectl`
> and the `aws-iam-authenticator` have already been installed on your system. You can skip to
> Create your Amazon EKS cluster and worker nodes (p. 6).

To install kubectl for Amazon EKS

- You have multiple options to download and install **kubectl** for your operating system.
  - The `kubectl` binary is available in many operating system package managers, and this option is
    often much easier than a manual download and install process. You can follow the instructions
    for your specific operating system or package manager in the Kubernetes documentation to
    install.
  - Amazon EKS also vends **kubectl** binaries that you can use that are identical to the upstream
    kubectl binaries with the same version. To install the Amazon EKS-vended binary for your
    operating system, see Installing `kubectl` (p. 182).

Create your Amazon EKS cluster and worker nodes

Now you can create your Amazon EKS cluster and a worker node group with the `eksctl` command line
utility.

To create your cluster with `eksctl`

1. Choose a tab below that matches your workload requirements. If you want to create a cluster that
   only runs pods on AWS Fargate, choose **AWS Fargate-only cluster**. If you only intend to run Linux
   workloads on your cluster, choose **Cluster with Linux-only workloads**. If you want to run Linux and
   Windows workloads on your cluster, choose **Cluster with Linux and Windows workloads**.

   AWS Fargate-only cluster

   This procedure assumes that you have installed `eksctl`, and that your `eksctl` version is at
   least `0.19.0`. You can check your version with the following command:

   ```
   eksctl version
   ```

   For more information on installing or upgrading `eksctl`, see Installing or upgrading
   `eksctl` (p. 197).

   Create your Amazon EKS cluster with Fargate support with the following command. Replace the
   example *values* with your own values. For `--region`, specify a supported region (p. 110).

   ```
   eksctl create cluster \
   ```

```
--name prod \
--region region-code \
--fargate
```

Your new Amazon EKS cluster is created without a worker node group. However, `eksctl` creates a pod execution role, a Fargate profile for the `default` and `kube-system` namespaces, and it patches the `coredns` deployment so that it can run on Fargate. For more information see AWS Fargate (p. 110).

Cluster with Linux-only workloads

This procedure assumes that you have installed `eksctl`, and that your `eksctl` version is at least `0.19.0`. You can check your version with the following command:

```
eksctl version
```

For more information on installing or upgrading `eksctl`, see Installing or upgrading eksctl (p. 197).

Create your Amazon EKS cluster and Linux worker nodes with the following command. Replace the example *values* with your own values.

```
eksctl create cluster \
--name prod \
--region region-code \
--nodegroup-name standard-workers \
--node-type t3.medium \
--nodes 3 \
--nodes-min 1 \
--nodes-max 4 \
--ssh-access \
--ssh-public-key my-public-key.pub \
--managed
```

Note

- The `--managed` option for Amazon EKS Managed node groups (p. 85) is currently only supported on Kubernetes 1.14 and later clusters. We recommend that you use the latest version of Kubernetes that is available in Amazon EKS to take advantage of the latest features. If you choose to use an earlier Kubernetes version, you must remove the `--managed` option.

  For more information on the available options for **eksctl create cluster**, see the project README on GitHub or view the help page with the following command.

  ```
  eksctl create cluster --help
  ```

- Though `--ssh-public-key` is optional, we highly recommend that you specify it when you create your node group with a cluster. This option enables SSH access to the nodes in your managed node group. Enabling SSH access allows you to connect to your instances and gather diagnostic information if there are issues. You cannot enable remote access after the node group is created. For more information, about keys, see Amazon EC2 key pairs in the Amazon EC2 User Guide for Linux Instances.

Output:

You'll see several lines of output as the cluster and worker nodes are created. The last line of output is similar to the following example line.

```
[#]  EKS cluster "prod" in "region-code" region is ready
```

Cluster with Linux and Windows workloads

This procedure assumes that you have installed `eksctl`, and that your `eksctl` version is at least `0.19.0`. You can check your version with the following command:

```
eksctl version
```

For more information on installing or upgrading `eksctl`, see Installing or upgrading eksctl (p. 197).

Familiarize yourself with the Windows support considerations (p. 54), which include supported values for `instanceType` in the example text below. Replace the example *values* with your own values. Save the text below to a file named `cluster-spec.yaml`. The configuration file is used to create a cluster and both Linux and Windows worker node groups. Even if you only want to run Windows workloads in your cluster, all Amazon EKS clusters must contain at least one Linux worker node. We recommend that you create at least two worker nodes in each node group for availability purposes. The minimum required Kubernetes version for Windows workloads is 1.14.

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

metadata:
  name: windows-prod
  region: region-code
managedNodeGroups:
  - name: linux-ng
    instanceType: t2.large
    minSize: 2

nodeGroups:
  - name: windows-ng
    instanceType: m5.large
    minSize: 2
    volumeSize: 100
    amiFamily: WindowsServer2019FullContainer
```

Create your Amazon EKS cluster and Windows and Linux worker nodes with the following command.

```
eksctl create cluster -f cluster-spec.yaml --install-vpc-controllers
```

> **Note**
> The `managedNodeGroups` option for Amazon EKS Managed node groups (p. 85) is currently only supported on Kubernetes 1.14 and later clusters. We recommend that you use the latest version of Kubernetes that is available in Amazon EKS to take advantage of the latest features. If you choose to use an earlier Kubernetes version, you must remove the `--managed` option.
> For more information on the available options for **eksctl create cluster**, see the project README on GitHub or view the help page with the following command.

```
eksctl create cluster --help
```

Output:

You'll see several lines of output as the cluster and worker nodes are created. The last line of output is similar to the following example line.

```
[#]  EKS cluster "windows-prod" in "region-code" region is ready
```

2. Cluster provisioning usually takes between 10 and 15 minutes. When your cluster is ready, test that your `kubectl` configuration is correct.

```
kubectl get svc
```

> **Note**
> If you receive the error "aws-iam-authenticator": executable file not found
> in $PATH, your **kubectl** isn't configured for Amazon EKS. For more information, see
> Installing aws-iam-authenticator (p. 187).
> If you receive any other authorization or resource type errors, see Unauthorized or access
> denied (kubectl) (p. 288) in the troubleshooting section.

Output:

```
NAME              TYPE         CLUSTER-IP     EXTERNAL-IP    PORT(S)    AGE
svc/kubernetes    ClusterIP    10.100.0.1     <none>         443/TCP    1m
```

3. (Linux GPU workers only) If you chose a GPU instance type and the Amazon EKS-optimized AMI with GPU support, you must apply the NVIDIA device plugin for Kubernetes as a DaemonSet on your cluster with the following command.

```
kubectl apply -f https://raw.githubusercontent.com/NVIDIA/k8s-device-plugin/1.0.0-beta/
nvidia-device-plugin.yml
```

Next steps

Now that you have a working Amazon EKS cluster with worker nodes, you are ready to start installing Kubernetes add-ons and deploying applications to your cluster. The following documentation topics help you to extend the functionality of your cluster.

*   Cluster Autoscaler (p. 135) — Configure the Kubernetes Cluster autoscaler to automatically adjust the number of nodes in your node groups.
*   Launch a guest book application (p. 199) — Create a sample guest book application to test your cluster and Linux worker nodes.
*   Deploy a Windows sample application (p. 59) — Deploy a sample application to test your cluster and Windows worker nodes.
*   Tutorial: Deploy the Kubernetes Dashboard (web UI) (p. 208) — This tutorial guides you through deploying the Kubernetes dashboard to your cluster.
*   Using Helm with Amazon EKS (p. 207) — The `helm` package manager for Kubernetes helps you install and manage applications on your cluster.
*   Installing the Kubernetes Metrics Server (p. 202) — The Kubernetes metrics server is an aggregator of resource usage data in your cluster.
*   Control plane metrics with Prometheus (p. 203) — This topic helps you deploy Prometheus into your cluster with `helm`.

Getting started with the AWS Management Console

This getting started guide helps you to create all of the required resources to get started with Amazon EKS in the AWS Management Console. In this guide, you manually create each resource in the Amazon EKS or AWS CloudFormation consoles, and the workflow described here gives you complete visibility into how each resource is created and how they interact with each other.

You can also choose to use the `eksctl` CLI to create your cluster and worker nodes. For more information, see Getting started with `eksctl` (p. 3).

Amazon EKS prerequisites

Before you can create an Amazon EKS cluster, you must create an IAM role that Kubernetes can assume to create AWS resources. For example, when a load balancer is created, Kubernetes assumes the role to create an Elastic Load Balancing load balancer in your account. This only needs to be done one time and can be used for multiple EKS clusters.

You must also create a VPC and a security group for your cluster to use. Although the VPC and security groups can be used for multiple EKS clusters, we recommend that you use a separate VPC for each EKS cluster to provide better network isolation.

This section also helps you to install the **kubectl** binary and configure it to work with Amazon EKS.

Create your Amazon EKS cluster IAM role

You can create the role using the AWS Management Console or AWS CloudFormation. Select the tab with the name of the tool that you'd like to use to create the role.

AWS Management Console

To create your Amazon EKS cluster role in the IAM console

1. Open the IAM console at https://console.aws.amazon.com/iam/.
2. Choose **Roles**, then **Create role**.
3. Choose **EKS** from the list of services, then **EKS - Cluster** for your use case, and then **Next: Permissions**.
4. Choose **Next: Tags**.
5. (Optional) Add metadata to the role by attaching tags as key–value pairs. For more information about using tags in IAM, see Tagging IAM Entities in the *IAM User Guide*.
6. Choose **Next: Review**.
7. For **Role name**, enter a unique name for your role, such as `eksClusterRole`, then choose **Create role**.

AWS CloudFormation

To create your Amazon EKS cluster role with AWS CloudFormation

1. Save the following AWS CloudFormation template to a text file on your local system.

   ```
   ---
   AWSTemplateFormatVersion: '2010-09-09'
   Description: 'Amazon EKS Cluster Role'
   ```

```
Resources:

  eksClusterRole:
    Type: AWS::IAM::Role
    Properties:
      AssumeRolePolicyDocument:
        Version: '2012-10-17'
        Statement:
        - Effect: Allow
          Principal:
            Service:
            - eks.amazonaws.com
          Action:
          - sts:AssumeRole
      ManagedPolicyArns:
        - arn:aws:iam::aws:policy/AmazonEKSClusterPolicy

Outputs:

  RoleArn:
    Description: The role that Amazon EKS will use to create AWS resources for
 Kubernetes clusters
    Value: !GetAtt eksClusterRole.Arn
    Export:
      Name: !Sub "${AWS::StackName}-RoleArn"
```

> **Note**
> Prior to April 16, 2020, `ManagedPolicyArns` had an entry for
> `arn:aws:iam::aws:policy/AmazonEKSServicePolicy`. With the
> `AWSServiceRoleForAmazonEKS` service-linked role, that policy is no longer required.

2. Open the AWS CloudFormation console at https://console.aws.amazon.com/cloudformation.
3. Choose **Create stack**.
4. For **Specify template**, select **Upload a template file**, and then choose **Choose file**.
5. Choose the file you created earlier, and then choose **Next**.
6. For **Stack name**, enter a name for your role, such as `eksClusterRole`, and then choose **Next**.
7. On the **Configure stack options** page, choose **Next**.
8. On the **Review** page, review your information, acknowledge that the stack might create IAM resources, and then choose **Create stack**.

Create your Amazon EKS cluster VPC

This section guides you through creating a VPC with either two public subnets and two private subnets or a VPC with three public subnets.

When you create an Amazon EKS cluster, you specify the VPC subnets for your cluster to use. Amazon EKS requires subnets in at least two Availability Zones. We recommend a VPC with public and private subnets so that Kubernetes can create public load balancers in the public subnets that load balance traffic to pods running on worker nodes that are in private subnets.

For more information about both VPC types, see .

Choose the tab below that represents your desired VPC configuration.

Public and private subnets

To create your cluster VPC with public and private subnets

1. Open the AWS CloudFormation console at https://console.aws.amazon.com/cloudformation.

2. From the navigation bar, select a Region that supports Amazon EKS.

3. Choose **Create stack**.

4. For **Choose a template**, select **Specify an Amazon S3 template URL**.

5. Paste the following URL into the text area and choose **Next**:

```
https://amazon-eks.s3.us-west-2.amazonaws.com/cloudformation/2020-05-08/amazon-eks-
vpc-private-subnets.yaml
```

6. On the **Specify Details** page, fill out the parameters accordingly, and then choose **Next**.

   - **Stack name**: Choose a stack name for your AWS CloudFormation stack. For example, you can call it **eks-vpc**.
   - **VpcBlock**: Choose a CIDR range for your VPC. You can keep the default value.
   - **PublicSubnet01Block**: Specify a CIDR range for public subnet 1. We recommend that you keep the default value so that you have plenty of IP addresses for load balancers to use.
   - **PublicSubnet02Block**: Specify a CIDR range for public subnet 2. We recommend that you keep the default value so that you have plenty of IP addresses for load balancers to use.
   - **PrivateSubnet01Block**: Specify a CIDR range for private subnet 1. We recommend that you keep the default value so that you have plenty of IP addresses for pods and load balancers to use.
   - **PrivateSubnet02Block**: Specify a CIDR range for private subnet 2. We recommend that you keep the default value so that you have plenty of IP addresses for pods and load balancers to use.

7. (Optional) On the **Options** page, tag your stack resources. Choose **Next**.

8. On the **Review** page, choose **Create**.

9. When your stack is created, select it in the console and choose **Outputs**.

10. Record the **SecurityGroups** value for the security group that was created. When you add worker nodes to your cluster, you must specify the ID of the security group. The security group is applied to the cross-account elastic network interfaces that are created in your subnets that allow the Amazon EKS control plane to communicate with your worker nodes.

11. Record the **VpcId** for the VPC that was created. You need this when you launch your worker node group template.

12. Record the **SubnetIds** for the subnets that were created and whether you created them as public or private subnets. When you add worker nodes to your cluster, you must specify the IDs of the subnets that you want to launch the worker nodes into.

Only public subnets

To create your cluster VPC with only public subnets

1. Open the AWS CloudFormation console at https://console.aws.amazon.com/cloudformation.

2. From the navigation bar, select a Region that supports Amazon EKS.

3. Choose **Create stack**.

4. For **Choose a template**, select **Specify an Amazon S3 template URL**.

5. Paste the following URL into the text area and choose **Next**:

```
https://amazon-eks.s3.us-west-2.amazonaws.com/cloudformation/2020-05-08/amazon-eks-
vpc-sample.yaml
```

6. On the **Specify Details** page, fill out the parameters accordingly, and then choose **Next**.

   - **Stack name**: Choose a stack name for your AWS CloudFormation stack. For example, you can call it **eks-vpc**.

- **VpcBlock**: Choose a CIDR range for your VPC. You can keep the default value.
- **Subnet01Block**: Specify a CIDR range for subnet 1. We recommend that you keep the default value so that you have plenty of IP addresses for pods and load balancers to use.
- **Subnet02Block**: Specify a CIDR range for subnet 2. We recommend that you keep the default value so that you have plenty of IP addresses for pods and load balancers to use.
- **Subnet03Block**: Specify a CIDR range for subnet 3. We recommend that you keep the default value so that you have plenty of IP addresses for pods and load balancers to use.

7. (Optional) On the **Options** page, tag your stack resources. Choose **Next**.
8. On the **Review** page, choose **Create**.
9. When your stack is created, select it in the console and choose **Outputs**.
10. Record the **SecurityGroups** value for the security group that was created. When you add worker nodes to your cluster, you must specify the ID of the security group. The security group is applied to the cross-account elastic network interfaces that are created in your subnets that allow the Amazon EKS control plane to communicate with your worker nodes.
11. Record the **VpcId** for the VPC that was created. You need this when you launch your worker node group template.
12. Record the **SubnetIds** for the subnets that were created. When you add worker nodes to your cluster, you must specify the IDs of the subnets that you want to launch the worker nodes into.

Install and configure **kubectl** for Amazon EKS

Kubernetes uses a command-line utility called `kubectl` for communicating with the cluster API server.

To install kubectl for Amazon EKS

- You have multiple options to download and install **kubectl** for your operating system.
  - The `kubectl` binary is available in many operating system package managers, and this option is often much easier than a manual download and install process. You can follow the instructions for your specific operating system or package manager in the Kubernetes documentation to install.
  - Amazon EKS also vends **kubectl** binaries that you can use that are identical to the upstream **kubectl** binaries with the same version. To install the Amazon EKS-vended binary for your operating system, see Installing `kubectl` (p. 182).

Install the latest AWS CLI

To use `kubectl` with your Amazon EKS clusters, you must install a binary that can create the required client security token for cluster API server communication. The **aws eks get-token** command, available in version 1.18.49 or later of the AWS CLI, supports client security token creation. To install or upgrade the AWS CLI, see Installing the AWS command line interface in the *AWS Command Line Interface User Guide*.

> **Important**
> Package managers such **yum**, **apt-get**, or Homebrew for macOS are often behind several versions of the AWS CLI. To ensure that you have the latest version, see Installing the AWS Command Line Interface in the *AWS Command Line Interface User Guide*.

You can check your AWS CLI version with the following command:

```
aws --version
```

> **Note**
> Your system's Python version must be 2.7.9 or later. Otherwise, you receive `hostname doesn't match` errors with AWS CLI calls to Amazon EKS.

If you are unable to install version 1.18.49 or later of the AWS CLI on your system, you must ensure that the AWS IAM Authenticator for Kubernetes is installed on your system. For more information, see Installing `aws-iam-authenticator` (p. 187).

Step 1: Create your Amazon EKS cluster

Now you can create your Amazon EKS cluster. This section helps you to create a cluster with the latest version of Kubernetes that is available in Amazon EKS to take advantage of all of the latest features. Some features are not available on older versions of Kubernetes.

> **Important**
> When an Amazon EKS cluster is created, the IAM entity (user or role) that creates the cluster is added to the Kubernetes RBAC authorization table as the administrator (with `system:master` permissions. Initially, only hat IAM user can make calls to the Kubernetes API server using **kubectl**. For more information, see Managing users or IAM roles for your cluster (p. 193). If you use the console to create the cluster, you must ensure that the same IAM user credentials are in the AWS SDK credential chain when you are running **kubectl** commands on your cluster.
> If you install and configure the AWS CLI, you can configure the IAM credentials for your user. If the AWS CLI is configured properly for your user, then `eksctl` and the AWS IAM Authenticator for Kubernetes can find those credentials as well. For more information, see Configuring the AWS CLI in the *AWS Command Line Interface User Guide*.

To create your cluster with the console

1. Open the Amazon EKS console at https://console.aws.amazon.com/eks/home#/clusters.

2. Choose **Create cluster**.

   > **Note**
   > If your IAM user doesn't have administrative privileges, you must explicitly add permissions for that user to call the Amazon EKS API operations. For more information, see Amazon EKS identity-based policy examples (p. 242).

3. On the **Configure cluster** page, fill in the following fields:

   - **Name** – A unique name for your cluster.

   - **Kubernetes version** – The version of Kubernetes to use for your cluster.

     > **Important**
     > This guide requires that you select the latest available version.

   - **Cluster service role** – Select the IAM role that you created with Create your Amazon EKS cluster IAM role (p. 10).

   - **Secrets encryption** – (Optional) Choose to enable envelope encryption of Kubernetes secrets using the AWS Key Management Service (AWS KMS). If you enable envelope encryption, the Kubernetes secrets are encrypted using the customer master key (CMK) that you select. The CMK must be symmetric, created in the same region as the cluster, and if the CMK was created in a different account, the user must have access to the CMK. For more information, see Allowing users in other accounts to use a CMK in the *AWS Key Management Service Developer Guide.* Kubernetes secrets encryption with an AWS KMS CMK requires Kubernetes version 1.13 or later. If no keys are listed, you must create one first. For more information, see Creating keys.

   - **Tags** – (Optional) Add any tags to your cluster. For more information, see Tagging your Amazon EKS resources (p. 274).

4. Select **Next**.

5. On the **Specify networking** page, select values for the following fields:

   - **VPC** – The VPC that you created previously in the section called "Create your Amazon EKS cluster VPC" (p. 11). You can find the name of your VPC in the drop-down list.

- **Subnets** – By default, the available subnets in the VPC specified in the previous field are preselected. Select any subnet that you don't want to host cluster resources, such as worker nodes or load balancers.

- **Security groups** – The **SecurityGroups** value from the AWS CloudFormation output that you generated with Create your Amazon EKS cluster VPC (p. 11). This security group has **ControlPlaneSecurityGroup** in the drop-down name.

  > **Important**
  > The worker node AWS CloudFormation template modifies the security group that you specify here, so **Amazon EKS strongly recommends that you use a dedicated security group for each cluster control plane (one per cluster)**. If this security group is shared with other resources, you might block or disrupt connections to those resources.

- For **Cluster endpoint access** – Choose one of the following options:

  - **Public** – Enables only public access to your cluster's Kubernetes API server endpoint. Kubernetes API requests that originate from outside of your cluster's VPC use the public endpoint. By default, access is allowed from any source IP address. You can optionally restrict access to one or more CIDR ranges such as 192.168.0.0/16, for example, by selecting **Advanced settings** and then selecting **Add source**.

  - **Private** – Enables only private access to your cluster's Kubernetes API server endpoint. Kubernetes API requests that originate from within your cluster's VPC use the private VPC endpoint.

  - **Public and private** – Enables public and private access.

  For more information about the previous options, see ??? (p. 35).

6. Select **Next**.

7. On the **Configure logging** page, you can optionally choose which log types that you want to enable. By default, each log type is **Disabled**. For more information, see Amazon EKS control plane logging (p. 40).

8. Select **Next**.

9. On the **Review and create** page, review the information that you entered or selected on the previous pages. Select **Edit** if you need to make changes to any of your selections. Once you're satisfied with your settings, select **Create**. The **Status** field shows **CREATING** until the cluster provisioning process completes.

   > **Note**
   > You might receive an error that one of the Availability Zones in your request doesn't have sufficient capacity to create an Amazon EKS cluster. If this happens, the error output contains the Availability Zones that can support a new cluster. Retry creating your cluster with at least two subnets that are located in the supported Availability Zones for your account. For more information, see Insufficient capacity (p. 287).

   When your cluster provisioning is complete (usually between 10 and 15 minutes), note the **API server endpoint** and **Certificate authority** values. These are used in your **kubectl** configuration.

Step 2: Create a `kubeconfig` file

In this section, you create a `kubeconfig` file for your cluster with the AWS CLI **update-kubeconfig** command. If you do not want to install the AWS CLI, or if you would prefer to create or update your kubeconfig manually, see Create a `kubeconfig` for Amazon EKS (p. 190).

To create your `kubeconfig` file with the AWS CLI

1. Ensure that you have version 1.18.49 or later of the AWS CLI installed. To install or upgrade the AWS CLI, see Installing the AWS Command Line Interface in the *AWS Command Line Interface User Guide*.

Note
Your system's Python version must be 2.7.9 or later. Otherwise, you receive `hostname doesn't match` errors with AWS CLI calls to Amazon EKS.

You can check your AWS CLI version with the following command:

```
aws --version
```

> **Important**
> Package managers such **yum**, **apt-get**, or Homebrew for macOS are often behind several versions of the AWS CLI. To ensure that you have the latest version, see Installing the AWS Command Line Interface in the *AWS Command Line Interface User Guide*.

2. Use the AWS CLI **update-kubeconfig** command to create or update your kubeconfig for your cluster.

   - By default, the resulting configuration file is created at the default kubeconfig path (`.kube/config`) in your home directory or merged with an existing kubeconfig at that location. You can specify another path with the `--kubeconfig` option.

   - You can specify an IAM role ARN with the `--role-arn` option to use for authentication when you issue **kubectl** commands. Otherwise, the IAM entity in your default AWS CLI or SDK credential chain is used. You can view your default AWS CLI or SDK identity by running the **aws sts get-caller-identity** command.

   - For more information, see the help page with the **aws eks update-kubeconfig help** command or see update-kubeconfig in the *AWS CLI Command Reference*.

   > **Note**
   > To run the following command, your account must be assigned the `eks:DescribeCluster` IAM permission for the cluster name that you specify.

   ```
   aws eks --region region-code update-kubeconfig --name cluster_name
   ```

3. Test your configuration.

   ```
   kubectl get svc
   ```

   > **Note**
   > If you receive the error `"aws-iam-authenticator": executable file not found in $PATH`, your **kubectl** isn't configured for Amazon EKS. For more information, see Installing `aws-iam-authenticator` (p. 187).
   > If you receive any other authorization or resource type errors, see Unauthorized or access denied (`kubectl`) (p. 288) in the troubleshooting section.

   Output:

   ```
   NAME            TYPE        CLUSTER-IP    EXTERNAL-IP    PORT(S)    AGE
   svc/kubernetes  ClusterIP   10.100.0.1    <none>         443/TCP    1m
   ```

Step 3: Launch a managed node group

Now that your VPC and Kubernetes control plane are created, you can launch and configure a managed node group.

Important
Amazon EKS worker nodes are standard Amazon EC2 instances, and you are billed for them based on normal Amazon EC2 instance prices. For more information, see Amazon EC2 pricing.

The Amazon EKS worker node `kubelet` daemon makes calls to AWS APIs on your behalf. Worker nodes receive permissions for these API calls through an IAM instance profile and associated policies. Before you can launch worker nodes and register them into a cluster, you must create an IAM role for those worker nodes to use when they are launched. For more information, see Amazon EKS worker node IAM role (p. 250). You can create the role using the AWS Management Console or AWS CloudFormation. Select the tab with the name of the tool that you'd like to use to create the role.

Note
We recommend that you create a new worker node IAM role for each cluster. Otherwise, a node from one cluster could authenticate with another cluster that it does not belong to.

AWS Management Console

To create your Amazon EKS worker node role in the IAM console

1. Open the IAM console at https://console.aws.amazon.com/iam/.
2. Choose **Roles**, then **Create role**.
3. Choose **EC2** from the list of services, then **Next: Permissions**.
4. In the **Filter policies** box, enter `AmazonEKSWorkerNodePolicy`. Check the box to the left of **AmazonEKSWorkerNodePolicy.**
5. In the **Filter policies** box, enter `AmazonEKS_CNI_Policy`. Check the box to the left of **AmazonEKS_CNI_Policy.**
6. In the **Filter policies** box, enter `AmazonEC2ContainerRegistryReadOnly`. Check the box to the left of **AmazonEC2ContainerRegistryReadOnly.**
7. Choose **Next: Tags**.
8. (Optional) Add metadata to the role by attaching tags as key–value pairs. For more information about using tags in IAM, see Tagging IAM Entities in the *IAM User Guide*.
9. Choose **Next: Review**.
10. For **Role name**, enter a unique name for your role, such as `NodeInstanceRole`. For **Role description**, replace the current text with descriptive text such as `Amazon EKS - Node Group Role`, then choose **Create role**.

AWS CloudFormation

To create your Amazon EKS worker node role using AWS CloudFormation

1. Open the AWS CloudFormation console at https://console.aws.amazon.com/cloudformation.
2. Choose **Create stack** and then choose **With new resources (standard)**.
3. For **Specify template**, select **Amazon S3 URL**.
4. Paste the following URL into the **Amazon S3 URL** text area and choose **Next** twice:

```
https://amazon-eks.s3.us-west-2.amazonaws.com/cloudformation/2020-05-08/amazon-eks-
nodegroup-role.yaml
```

5. On the **Specify stack details** page, for **Stack name** enter a name such as **eks-node-group-instance-role** and choose **Next**.
6. (Optional) On the **Configure stack options** page, you can choose to tag your stack resources. Choose **Next**.
7. On the **Review** page, check the box in the **Capabilities** section and choose **Create stack**.

8. When your stack is created, select it in the console and choose **Outputs**.

9. Record the **NodeInstanceRole** value for the IAM role that was created. You need this when you create your node group.

To launch your managed node group

1. Wait for your cluster status to show as `ACTIVE`. You cannot create a managed node group for a cluster that is not yet `ACTIVE`.

2. Open the Amazon EKS console at https://console.aws.amazon.com/eks/home#/clusters.

3. Choose the name of the cluster that you want to create your managed node group in.

4. On the cluster page, choose **Add node group**.

5. On the **Configure node group** page, fill out the parameters accordingly, and then choose **Next**.

   - **Name** — Enter a unique name for your managed node group.
   - **Node IAM role name** — Choose the node instance role to use with your node group. For more information, see Amazon EKS worker node IAM role (p. 250).

     Important
     We recommend using a role that is not currently in use by any self-managed node group, or that you plan to use with a new self-managed node group. For more information, see ??? (p. 91).

   - **Subnets** — Choose the subnets to launch your managed nodes into.

     Important
     If you are running a stateful application across multiple Availability Zones that is backed by Amazon EBS volumes and using the Kubernetes Cluster Autoscaler (p. 135), you should configure multiple node groups, each scoped to a single Availability Zone. In addition, you should enable the `--balance-similar-node-groups` feature.

     Important
     If you choose a public subnet, then the subnet must have `MapPublicIpOnLaunch` set to true for the instances to be able to successfully join a cluster. If the subnet was created using `eksctl` or the Amazon EKS-vended AWS CloudFormation templates (p. 152) on or after 03/26/2020, then this setting is already set to true. If the subnets were created with `eksctl` or the AWS CloudFormation templates before 03/26/2020, then you need to change the setting manually. For more information, see Modifying the public IPv4 addressing attribute for your subnet.

   - **Remote Access** — (Optional) You can enable SSH access to the nodes in your managed node group. Enabling SSH allows you to connect to your instances and gather diagnostic information if there are issues. Complete the following steps to enable remote access.

     Note
     We highly recommend enabling remote access when you create your node group. You cannot enable remote access after the node group is created.

     1. Select the check box to **Allow remote access to nodes**.

     2. For **SSH key pair**, choose an Amazon EC2 SSH key to use. For more information, see Amazon EC2 key pairs in the Amazon EC2 User Guide for Linux Instances.

     3. For **Allow remote access from**, choose **All** to allow SSH access from anywhere on the Internet (0.0.0.0/0), or select a security group to allow SSH access from instances that belong to that security group.

   - **Tags** — (Optional) You can choose to tag your Amazon EKS managed node group. These tags do not propagate to other resources in the node group, such as Auto Scaling groups or instances. For more information, see Tagging your Amazon EKS resources (p. 274).

   - **Kubernetes labels** — (Optional) You can choose to apply Kubernetes labels to the nodes in your managed node group.

6. On the **Set compute configuration** page, fill out the parameters accordingly, and then choose **Next**.

   - **AMI type** — Choose **Amazon Linux 2 (AL2_x86_64)** for non-GPU instances, or **Amazon Linux 2 GPU Enabled (AL2_x86_64_GPU)** for GPU instances.
   - **Instance type** — Choose the instance type to use in your managed node group. Larger instance types can accommodate more pods.
   - **Disk size** — Enter the disk size (in GiB) to use for your worker node root volume.

7. On the **Setup scaling policies** page, fill out the parameters accordingly, and then choose **Next**.

   > **Note**
   > Amazon EKS does not automatically scale your node group in or out. However, you can configure the Kubernetes Cluster Autoscaler (p. 135) to do this for you.

   - **Minimum size** — Specify the minimum number of worker nodes that the managed node group can scale in to.
   - **Maximum size** — Specify the maximum number of worker nodes that the managed node group can scale out to.
   - **Desired size** — Specify the current number of worker nodes that the managed node group should maintain at launch.

8. On the **Review and create** page, review your managed node group configuration and choose **Create**.

9. Watch the status of your nodes and wait for them to reach the `Ready` status.

   ```
   kubectl get nodes --watch
   ```

10. (GPU workers only) If you chose a GPU instance type and the Amazon EKS-optimized AMI with GPU support, you must apply the NVIDIA device plugin for Kubernetes as a DaemonSet on your cluster with the following command.

    ```
    kubectl apply -f https://raw.githubusercontent.com/NVIDIA/k8s-device-plugin/1.0.0-beta/
    nvidia-device-plugin.yml
    ```

(Optional) To launch Windows worker nodes

Add Windows support to your cluster and launch Windows worker nodes. For more information, see Windows support (p. 54). All Amazon EKS clusters must contain at least one Linux worker node, even if you only want to run Windows workloads in your cluster.

Next steps

Now that you have a working Amazon EKS cluster with worker nodes, you are ready to start installing Kubernetes add-ons and deploying applications to your cluster. The following documentation topics help you to extend the functionality of your cluster.

- Cluster Autoscaler (p. 135) — Configure the Kubernetes Cluster autoscaler to automatically adjust the number of nodes in your node groups.
- Launch a guest book application (p. 199) — Create a sample guest book application to test your cluster and Linux worker nodes.
- Deploy a Windows sample application (p. 59) — Deploy a sample application to test your cluster and Windows worker nodes.
- Tutorial: Deploy the Kubernetes Dashboard (web UI) (p. 208) — This tutorial guides you through deploying the Kubernetes dashboard to your cluster.
- Using Helm with Amazon EKS (p. 207) — The `helm` package manager for Kubernetes helps you install and manage applications on your cluster.

- Installing the Kubernetes Metrics Server (p. 202) — The Kubernetes metrics server is an aggregator of resource usage data in your cluster.
- Control plane metrics with Prometheus (p. 203) — This topic helps you deploy Prometheus into your cluster with `helm`.

Amazon EKS clusters

An Amazon EKS cluster consists of two primary components:

- The Amazon EKS control plane
- Amazon EKS worker nodes that are registered with the control plane

The Amazon EKS control plane consists of control plane nodes that run the Kubernetes software, such as `etcd` and the Kubernetes API server. The control plane runs in an account managed by AWS, and the Kubernetes API is exposed via the Amazon EKS endpoint associated with your cluster. Each Amazon EKS cluster control plane is single-tenant and unique, and runs on its own set of Amazon EC2 instances.

All of the data stored by the `etcd` nodes and associated Amazon EBS volumes is encrypted. Amazon EKS uses master encryption keys that generate volume encryption keys which are managed by the Amazon EKS service.

The cluster control plane is provisioned across multiple Availability Zones and fronted by an Elastic Load Balancing Network Load Balancer. Amazon EKS also provisions elastic network interfaces in your VPC subnets to provide connectivity from the control plane instances to the worker nodes (for example, to support **kubectl exec**, **logs**, and **proxy** data flows).

Amazon EKS worker nodes run in your AWS account and connect to your cluster's control plane via the API server endpoint and a certificate file that is created for your cluster.

Creating an Amazon EKS cluster

This topic walks you through creating an Amazon EKS cluster.

If this is your first time creating an Amazon EKS cluster, we recommend that you follow one of our Getting started with Amazon EKS (p. 3) guides instead. They provide complete end-to-end walkthroughs for creating an Amazon EKS cluster with worker nodes.

> **Important**
> When an Amazon EKS cluster is created, the IAM entity (user or role) that creates the cluster is added to the Kubernetes RBAC authorization table as the administrator (with `system:master` permissions. Initially, only hat IAM user can make calls to the Kubernetes API server using **kubectl**. For more information, see Managing users or IAM roles for your cluster (p. 193). If you use the console to create the cluster, you must ensure that the same IAM user credentials are in the AWS SDK credential chain when you are running **kubectl** commands on your cluster.
> If you install and configure the AWS CLI, you can configure the IAM credentials for your user. If the AWS CLI is configured properly for your user, then `eksctl` and the AWS IAM Authenticator for Kubernetes can find those credentials as well. For more information, see Configuring the AWS CLI in the *AWS Command Line Interface User Guide*.

Choose the tab below that corresponds to your desired cluster creation method.

eksctl

This procedure assumes that you have installed `eksctl`, and that your `eksctl` version is at least `0.19.0`. You can check your version with the following command:

```
eksctl version
```

For more information on installing or upgrading `eksctl`, see Installing or upgrading eksctl (p. 197).

To create your cluster with `eksctl`

1. Create a cluster with the Amazon EKS default Kubernetes version in your default region. Replace *my-cluster* with your own value.

```
eksctl create cluster \
 --name my-cluster \
 --without-nodegroup
```

Cluster provisioning takes several minutes. During cluster creation, you'll see several lines of output. The last line of output is similar to the following example line.

```
[#]  EKS cluster "my-cluster" in "region-code" region is ready
```

2. When your cluster is ready, test that your `kubectl` configuration is correct.

```
kubectl get svc
```

> **Note**
> If you receive the error `"aws-iam-authenticator": executable file not found in $PATH`, your **kubectl** isn't configured for Amazon EKS. For more information, see Installing `aws-iam-authenticator` (p. 187).
> If you receive any other authorization or resource type errors, see Unauthorized or access denied (`kubectl`) (p. 288) in the troubleshooting section.

Output:

```
NAME             TYPE        CLUSTER-IP    EXTERNAL-IP    PORT(S)    AGE
svc/kubernetes   ClusterIP   10.100.0.1    <none>         443/TCP    1m
```

3. (Optional) If you want to run pods on AWS Fargate in your cluster, then you must ??? (p. 113) and ??? (p. 113).

4. Follow the procedures in Launching Amazon EKS Linux worker nodes (p. 92) to add Linux worker nodes to your cluster to support your workloads.

5. (Optional) After you add Linux worker nodes to your cluster, follow the procedures in Windows support (p. 54) to add Windows support to your cluster and to add Windows worker nodes. All Amazon EKS clusters must contain at least one Linux worker node, even if you only want to run Windows workloads in your cluster.

AWS Management Console

This procedure has the following prerequisites:

- You have created a VPC and a dedicated security group that meet the requirements for an Amazon EKS cluster. For more information, see Cluster VPC considerations (p. 155) and Amazon EKS security group considerations (p. 157). The Getting started with the AWS Management Console (p. 10) guide creates a VPC that meets the requirements, or you can also follow Creating a VPC for your Amazon EKS cluster (p. 152) to create one.

- You have created an Amazon EKS cluster IAM role to apply to your cluster. The Getting started with Amazon EKS (p. 3) guide creates a service role for you, or you can also follow Amazon EKS IAM roles (p. 241) to create one manually.

To create your cluster with the console

1. Open the Amazon EKS console at https://console.aws.amazon.com/eks/home#/clusters.

2. Choose **Create cluster**.

> **Note**
> If your IAM user doesn't have administrative privileges, you must explicitly add
> permissions for that user to call the Amazon EKS API operations. For more information,
> see Amazon EKS identity-based policy examples (p. 242).

3. On the **Configure cluster** page, fill in the following fields:

   - **Name** – A unique name for your cluster.
   - **Kubernetes version** – The version of Kubernetes to use for your cluster.

     Unless you require a specific Kubernetes version for your application, we recommend that you
     use the latest version available in Amazon EKS.

     > **Important**
     > Kubernetes version 1.13 is now deprecated on Amazon EKS. On **June 30th, 2020**,
     > Kubernetes version 1.13 will no longer be supported on Amazon EKS. On or after this
     > date, you will no longer be able to create new 1.13 clusters, and all existing Amazon
     > EKS clusters running Kubernetes version 1.13 will eventually be automatically
     > updated to version 1.14. We recommend that you update any 1.13 clusters to version
     > 1.14 or later in order to avoid service interruption. For more information, see Amazon
     > EKS version deprecation (p. 48).
     > Kubernetes API versions available through Amazon EKS are officially supported by
     > AWS, until we remove the ability to create clusters using that version. This is true
     > even if upstream Kubernetes is no longer supporting a version available on Amazon
     > EKS. We backport security fixes that are applicable to the Kubernetes versions
     > supported on Amazon EKS. Existing clusters are always supported, and Amazon EKS
     > will automatically update your cluster to a supported version if you have not done so
     > manually by the version end of life date.

   - **Cluster service role** – Choose the Amazon EKS cluster role to allow the Kubernetes control
     plane to manage AWS resources on your behalf. For more information, see Amazon EKS
     cluster IAM role (p. 248).
   - **Secrets encryption** – (Optional) Choose to enable envelope encryption of Kubernetes secrets
     using the AWS Key Management Service (AWS KMS). If you enable envelope encryption, the
     Kubernetes secrets are encrypted using the customer master key (CMK) that you select. The
     CMK must be symmetric, created in the same region as the cluster, and if the CMK was created
     in a different account, the user must have access to the CMK. For more information, see
     Allowing users in other accounts to use a CMK in the *AWS Key Management Service Developer
     Guide*. Kubernetes secrets encryption with an AWS KMS CMK requires Kubernetes version 1.13
     or later. If no keys are listed, you must create one first. For more information, see Creating
     keys.
   - **Tags** – (Optional) Add any tags to your cluster. For more information, see Tagging your
     Amazon EKS resources (p. 274).

4. Select **Next**.

5. On the **Specify networking** page, select values for the following fields:

   - **VPC** – Select an existing VPC to use for your cluster. If none are listed, then you need to create
     one first. For more information, see ??? (p. 152).
   - **Subnets** – By default, the available subnets in the VPC specified in the previous field are
     preselected. Select any subnet that you don't want to host cluster resources, such as worker
     nodes or load balancers. The subnets must meet the requirements for an Amazon EKS cluster.
     For more information, see Cluster VPC considerations (p. 155).

     > **Important**
     > If you select subnets that were created before 03/26/2020 using one of the Amazon
     > EKS AWS CloudFormation VPC templates, be aware of a default setting change that
     > was introduced on 03/26/2020. For more information, see ??? (p. 152).

- **Security groups** – The **SecurityGroups** value from the AWS CloudFormation output that you generated with Create your Amazon EKS cluster VPC (p. 11). This security group has **ControlPlaneSecurityGroup** in the drop-down name.

    Important
    The worker node AWS CloudFormation template modifies the security group that you specify here, so **Amazon EKS strongly recommends that you use a dedicated security group for each cluster control plane (one per cluster)**. If this security group is shared with other resources, you might block or disrupt connections to those resources.

- For **Cluster endpoint access** – Choose one of the following options:

    - **Public** – Enables only public access to your cluster's Kubernetes API server endpoint. Kubernetes API requests that originate from outside of your cluster's VPC use the public endpoint. By default, access is allowed from any source IP address. You can optionally restrict access to one or more CIDR ranges such as 192.168.0.0/16, for example, by selecting **Advanced settings** and then selecting **Add source**.

    - **Private** – Enables only private access to your cluster's Kubernetes API server endpoint. Kubernetes API requests that originate from within your cluster's VPC use the private VPC endpoint.

    - **Public and private** – Enables public and private access.

    For more information about the previous options, see ??? (p. 35).

6. Select **Next**.

7. On the **Configure logging** page, you can optionally choose which log types that you want to enable. By default, each log type is **Disabled**. For more information, see Amazon EKS control plane logging (p. 40).

8. Select **Next**.

9. On the **Review and create** page, review the information that you entered or selected on the previous pages. Select **Edit** if you need to make changes to any of your selections. Once you're satisfied with your settings, select **Create**. The **Status** field shows **CREATING** until the cluster provisioning process completes.

    Note
    You might receive an error that one of the Availability Zones in your request doesn't have sufficient capacity to create an Amazon EKS cluster. If this happens, the error output contains the Availability Zones that can support a new cluster. Retry creating your cluster with at least two subnets that are located in the supported Availability Zones for your account. For more information, see Insufficient capacity (p. 287).

    Cluster provisioning usually takes between 10 and 15 minutes.

10. Now that you have created your cluster, follow the procedures in Installing `aws-iam-authenticator` (p. 187) and Create a `kubeconfig` for Amazon EKS (p. 190) to enable communication with your new cluster.

11. (Optional) If you want to run pods on AWS Fargate in your cluster, see Getting started with AWS Fargate on Amazon EKS (p. 111).

12. After you enable communication, follow the procedures in Launching Amazon EKS Linux worker nodes (p. 92) to add Linux worker nodes to your cluster to support your workloads.

13. (Optional) After you add Linux worker nodes to your cluster, follow the procedures in Windows support (p. 54) to add Windows support to your cluster and to add Windows worker nodes. All Amazon EKS clusters must contain at least one Linux worker node, even if you only want to run Windows workloads in your cluster.

AWS CLI

To create your cluster with the AWS CLI

This procedure has the following prerequisites:

- You have created a VPC and a dedicated security group that meets the requirements for an Amazon EKS cluster. For more information, see Cluster VPC considerations (p. 155) and Amazon EKS security group considerations (p. 157). The Getting started with the AWS Management Console (p. 10) guide creates a VPC that meets the requirements, or you can also follow Creating a VPC for your Amazon EKS cluster (p. 152) to create one.
- You have created an Amazon EKS cluster IAM role to apply to your cluster. The Getting started with Amazon EKS (p. 3) guide creates a service role for you, or you can also follow Amazon EKS IAM roles (p. 241) to create one manually.

1. Create your cluster with the following command. Substitute your cluster name, the Amazon Resource Name (ARN) of your Amazon EKS cluster IAM role that you created in Create your Amazon EKS cluster IAM role (p. 10), and the subnet and security group IDs for the VPC that you created in Create your Amazon EKS cluster VPC (p. 11).

   Important
   Kubernetes version 1.13 is now deprecated on Amazon EKS. On **June 30th, 2020**, Kubernetes version 1.13 will no longer be supported on Amazon EKS. On or after this date, you will no longer be able to create new 1.13 clusters, and all existing Amazon EKS clusters running Kubernetes version 1.13 will eventually be automatically updated to version 1.14. We recommend that you update any 1.13 clusters to version 1.14 or later in order to avoid service interruption. For more information, see Amazon EKS version deprecation (p. 48).
   Kubernetes API versions available through Amazon EKS are officially supported by AWS, until we remove the ability to create clusters using that version. This is true even if upstream Kubernetes is no longer supporting a version available on Amazon EKS. We backport security fixes that are applicable to the Kubernetes versions supported on Amazon EKS. Existing clusters are always supported, and Amazon EKS will automatically update your cluster to a supported version if you have not done so manually by the version end of life date.

   ```
   aws eks --region region-code create-cluster \
       --name devel --kubernetes-version 1.16 \
       --role-arn \
         arn:aws:iam::111122223333:role/eks-service-role-AWSServiceRoleForAmazonEKS-EXAMPLEBKZRQR \
       --resources-vpc-config \
         subnetIds=subnet-a9189fe2,subnet-50432629,securityGroupIds=sg-f5c54184
   ```

   Important
   If you receive a syntax error similar to the following, you might be using a preview version of the AWS CLI for Amazon EKS. The syntax for many Amazon EKS commands has changed since the public service launch. Update your AWS CLI version to the latest available and delete the custom service model directory at `~/.aws/models/eks`.

   ```
   aws: error: argument --cluster-name is required
   ```

   Note
   If your IAM user doesn't have administrative privileges, you must explicitly add permissions for that user to call the Amazon EKS API operations. For more information, see Amazon EKS identity-based policy examples (p. 242).

Output:

```
{
    "cluster": {
        "name": "devel",
        "arn": "arn:aws:eks:region-code:111122223333:cluster/devel",
        "createdAt": 1527785885.159,
        "version": "1.16",
        "roleArn": "arn:aws:iam::111122223333:role/eks-service-role-
AWSServiceRoleForAmazonEKS-AFNL4H8HB71F",
        "resourcesVpcConfig": {
            "subnetIds": [
                "subnet-a9189fe2",
                "subnet-50432629"
            ],
            "securityGroupIds": [
                "sg-f5c54184"
            ],
            "vpcId": "vpc-a54041dc",
            "endpointPublicAccess": true,
            "endpointPrivateAccess": false
        },
        "status": "CREATING",
        "certificateAuthority": {}
    }
}
```

Note
You might receive an error that one of the Availability Zones in your request doesn't
have sufficient capacity to create an Amazon EKS cluster. If this happens, the error
output contains the Availability Zones that can support a new cluster. Retry creating
your cluster with at least two subnets that are located in the supported Availability
Zones for your account. For more information, see Insufficient capacity (p. 287).

To encrypt the Kubernetes secrets with a customer master key (CMK) from AWS Key
Management Service (AWS KMS), first create a CMK using the create-key operation.

```
MY_KEY_ARN=$(aws kms create-key --query KeyMetadata.Arn --output text)
```

Add the `--encryption-config` parameter to the `aws eks create-cluster` command.
Encryption of Kubernetes secrets can only be enabled when the cluster is created.

```
--encryption-config '[{"resources":["secrets"],"provider":
{"keyArn":"$MY_KEY_ARN"}}]'
```

The `keyArn` member can contain either the alias or ARN of your CMK. The CMK must be
symmetric, created in the same Region as the cluster, and if the CMK was created in a different
account, the user must have access to the CMK. For more information, see Allowing users in
other accounts to use a CMK in the *AWS Key Management Service Developer Guide*. Kubernetes
secrets encryption with an AWS KMS CMK requires Kubernetes version 1.13 or later.

2. Cluster provisioning usually takes between 10 and 15 minutes. You can query the status of your
cluster with the following command. When your cluster status is `ACTIVE`, you can proceed.

```
aws eks --region region-code describe-cluster --name devel --query "cluster.status"
```

3. When your cluster provisioning is complete, retrieve the `endpoint` and `certificateAuthority.data` values with the following commands. You must add these values to your **kubectl** configuration so that you can communicate with your cluster.

    a. Retrieve the `endpoint`.

    ```
    aws eks --region region-code describe-cluster --name devel  --query
     "cluster.endpoint" --output text
    ```

    b. Retrieve the `certificateAuthority.data`.

    ```
    aws eks --region region-code describe-cluster --name devel  --query
     "cluster.certificateAuthority.data" --output text
    ```

4. Now that you have created your cluster, follow the procedures in Installing `aws-iam-authenticator` (p. 187) and Create a `kubeconfig` for Amazon EKS (p. 190) to enable communication with your new cluster.

5. (Optional) If you want to run pods on AWS Fargate in your cluster, see Getting started with AWS Fargate on Amazon EKS (p. 111).

6. After you enable communication, follow the procedures in Launching Amazon EKS Linux worker nodes (p. 92) to add worker nodes to your cluster to support your workloads.

7. (Optional) After you add Linux worker nodes to your cluster, follow the procedures in Windows support (p. 54) to add Windows support to your cluster and to add Windows worker nodes. All Amazon EKS clusters must contain at least one Linux worker node, even if you only want to run Windows workloads in your cluster.

Updating an Amazon EKS cluster Kubernetes version

When a new Kubernetes version is available in Amazon EKS, you can update your cluster to the latest version.

Important
We recommend that before updating to a new Kubernetes version that you review the information in ??? (p. 45) and in the update steps in this topic.

New Kubernetes versions introduce significant changes, so we recommend that you test the behavior of your applications against a new Kubernetes version before performing the update on your production clusters. You can achieve this by building a continuous integration workflow to test your application behavior end-to-end before moving to a new Kubernetes version.

The update process consists of Amazon EKS launching new API server nodes with the updated Kubernetes version to replace the existing ones. Amazon EKS performs standard infrastructure and readiness health checks for network traffic on these new nodes to verify that they are working as expected. If any of these checks fail, Amazon EKS reverts the infrastructure deployment, and your cluster remains on the prior Kubernetes version. Running applications are not affected, and your cluster is never left in a non-deterministic or unrecoverable state. Amazon EKS regularly backs up all managed clusters, and mechanisms exist to recover clusters if necessary. We are constantly evaluating and improving our Kubernetes infrastructure management processes.

In order to upgrade the cluster, Amazon EKS requires 2-3 free IP addresses from the subnets which were provided when you created the cluster. If these subnets do not have available IP addresses, then the upgrade can fail. Additionally, if any of the subnets or security groups that were provided during cluster creation have been deleted, the cluster upgrade process can fail.

Note
Although Amazon EKS runs a highly available control plane, you might experience minor service interruptions during an update. For example, if you attempt to connect to an API server just before or just after it's terminated and replaced by a new API server running the new version of Kubernetes, you might experience API call errors or connectivity issues. If this happens, retry your API operations until they succeed.

Amazon EKS does not modify any of your Kubernetes add-ons when you update a cluster. After updating your cluster, we recommend that you update your add-ons to the versions listed in the following table for the new Kubernetes version that you're updating to. Steps to accomplish this are included in the update procedures.

Kubernetes version	1.16	1.15	1.14	1.13
Amazon VPC CNI plug-in	1.6.1	1.6.1	1.6.1	1.6.1
DNS (CoreDNS)	1.6.6	1.6.6	1.6.6	1.6.6
KubeProxy	1.16.8	1.15.11	1.14.9	1.13.12

If you're using additional add-ons for your cluster that aren't listed in the previous table, update them to the latest compatible versions after updating your cluster.

Update an existing cluster

Update the cluster and Kubnernete add-ons.

To update an existing cluster

1. Compare the Kubernetes version of your cluster control plane to the Kubernetes version of your worker nodes.

    - Get the Kubernetes version of your cluster control plane with the following command.

    ```
    kubectl version --short
    ```

    - Get the Kubernetes version of your worker nodes with the following command.

    ```
    kubectl get nodes
    ```

    If your worker nodes are more than one Kubernetes minor version older than your control plane, then you must upgrade your worker nodes to a newer Kubernetes minor version before you update your cluster's Kubernetes version. For more information, see Kubernetes version and version skew support policy in the Kubernetes documentation.

    We recommend that you update your worker nodes to your cluster's current pre-update Kubernetes minor version prior to your cluster update. Your worker nodes must not run a newer Kubernetes version than your control plane. For example, if your control plane is running version 1.15 and your workers are running version 1.13, update your worker nodes to version 1.14 or 1.15 (recommended) before you update your cluster's Kubernetes version to 1.16. For more information, see Self-managed worker node updates (p. 101).

2. The pod security policy admission controller is enabled on Amazon EKS clusters running Kubernetes version 1.13 or later. If you are upgrading your cluster to Kubernetes version 1.13 or later, ensure that the proper pod security policies are in place before you update to avoid any issues. You can check for the default policy with the following command:

```
kubectl get psp eks.privileged
```

If you receive the following error, see To install or restore the default pod security policy (p. 271) before proceeding.

```
Error from server (NotFound): podsecuritypolicies.extensions "eks.privileged" not found
```

3. Update your cluster. For instructions, select the tab with the name of the tool that you want to use to update your cluster.

eksctl

This procedure assumes that you have installed `eksctl`, and that your `eksctl` version is at least `0.19.0`. You can check your version with the following command:

```
eksctl version
```

For more information on installing or upgrading `eksctl`, see Installing or upgrading eksctl (p. 197).

Note
This procedure only works for clusters that were created with `eksctl`.

Update your Amazon EKS cluster Kubernetes version one minor version later than its current version with the following command, replacing *dev* with your cluster name. Because Amazon EKS runs a highly available control plane, you can update only one minor version at a time. See Kubernetes Version and Version Skew Support Policy for the rationale behind this requirement.

Important
You may need to update some of your deployed resources before you can update to 1.16. For more information, see the section called "Kubernetes 1.16 upgrade prerequisites" (p. 34).

```
eksctl update cluster --name dev --approve
```

This process takes several minutes to complete.

AWS Management Console

1. Open the Amazon EKS console at https://console.aws.amazon.com/eks/home#/clusters.

2. Choose the name of the cluster to update and choose **Update cluster version**.

3. For **Kubernetes version**, select the version to update your cluster to and choose **Update**.

Important
You may need to update some of your deployed resources before you can update to 1.16. For more information, see the section called "Kubernetes 1.16 upgrade prerequisites" (p. 34).

Important
Kubernetes version 1.13 is now deprecated on Amazon EKS. On **June 30th, 2020**, Kubernetes version 1.13 will no longer be supported on Amazon EKS. On or after this date, you will no longer be able to create new 1.13 clusters, and all existing Amazon EKS clusters running Kubernetes version 1.13 will eventually be automatically updated to version 1.14. We recommend that you update any 1.13 clusters to version 1.14 or later in order to avoid service interruption. For more information, see Amazon EKS version deprecation (p. 48).

Kubernetes API versions available through Amazon EKS are officially supported by AWS, until we remove the ability to create clusters using that version. This is true even if upstream Kubernetes is no longer supporting a version available on Amazon EKS. We backport security fixes that are applicable to the Kubernetes versions supported on Amazon EKS. Existing clusters are always supported, and Amazon EKS will automatically update your cluster to a supported version if you have not done so manually by the version end of life date.

Important
Because Amazon EKS runs a highly available control plane, you can update only one minor version at a time. See Kubernetes Version and Version Skew Support Policy for the rationale behind this requirement. Therefore, if your current version is 1.14 and you want to upgrade to 1.16, you must first upgrade your cluster to 1.15 and then upgrade it from 1.15 to 1.16. If you try to update directly from 1.14 to 1.16, the update version command throws an error.

4. For **Cluster name**, type the name of your cluster and choose **Confirm**.

Note
The cluster update should finish in a few minutes.

AWS CLI

1. Update your cluster with the following AWS CLI command. Substitute your cluster name and desired Kubernetes minor version.

Important
You may need to update some of your deployed resources before you can update to 1.16. For more information, see the section called "Kubernetes 1.16 upgrade prerequisites" (p. 34).

Important
Kubernetes version 1.13 is now deprecated on Amazon EKS. On **June 30th, 2020**, Kubernetes version 1.13 will no longer be supported on Amazon EKS. On or after this date, you will no longer be able to create new 1.13 clusters, and all existing Amazon EKS clusters running Kubernetes version 1.13 will eventually be automatically updated to version 1.14. We recommend that you update any 1.13 clusters to version 1.14 or later in order to avoid service interruption. For more information, see Amazon EKS version deprecation (p. 48).
Kubernetes API versions available through Amazon EKS are officially supported by AWS, until we remove the ability to create clusters using that version. This is true even if upstream Kubernetes is no longer supporting a version available on Amazon EKS. We backport security fixes that are applicable to the Kubernetes versions supported on Amazon EKS. Existing clusters are always supported, and Amazon EKS will automatically update your cluster to a supported version if you have not done so manually by the version end of life date.

Important
Because Amazon EKS runs a highly available control plane, you can update only one minor version at a time. See Kubernetes Version and Version Skew Support Policy for the rationale behind this requirement. Therefore, if your current version is 1.14 and you want to upgrade to 1.16, you must first upgrade your cluster to 1.15 and then upgrade it from 1.15 to 1.16. If you try to update directly from 1.14 to 1.16, the update version command throws an error.

```
aws eks --region region-code update-cluster-version --name prod --kubernetes-
version 1.16
```

Output:

```
{
    "update": {
        "id": "b5f0ba18-9a87-4450-b5a0-825e6e84496f",
        "status": "InProgress",
        "type": "VersionUpdate",
        "params": [
            {
                "type": "Version",
                "value": "1.16"
            },
            {
                "type": "PlatformVersion",
                "value": "eks.1"
            }
        ],
        "createdAt": 1577485455.5,
        "errors": []
    }
}
```

2. Monitor the status of your cluster update with the following command, using the cluster name and update ID that the previous command returned. Your update is complete when the status appears as `Successful`.

> **Note**
> The cluster update should finish in a few minutes.

```
aws eks --region region-code describe-update --name prod --update-
id b5f0ba18-9a87-4450-b5a0-825e6e84496f
```

Output:

```
{
    "update": {
        "id": "b5f0ba18-9a87-4450-b5a0-825e6e84496f",
        "status": "Successful",
        "type": "VersionUpdate",
        "params": [
            {
                "type": "Version",
                "value": "1.16"
            },
            {
                "type": "PlatformVersion",
                "value": "eks.1"
            }
        ],
        "createdAt": 1577485455.5,
        "errors": []
    }
}
```

4. Patch the `kube-proxy` daemonset to use the image that corresponds to your cluster's Region and current Kubernetes version (in this example, *1.16.8*).

Kubernetes version	1.16	1.15	1.14	1.13
KubeProxy	1.16.8	1.15.11	1.14.9	1.13.12

First, retrieve your current `kube-proxy` image:

```
kubectl get daemonset kube-proxy --namespace kube-system -
o=jsonpath='{$.spec.template.spec.containers[:1].image}'
```

Update `kube-proxy` to the recommended version by taking the output from the previous step and replacing the version tag with your cluster's recommended `kube-proxy` version:

```
kubectl set image daemonset.apps/kube-proxy \
    -n kube-system \
    kube-proxy=602401143452.dkr.ecr.us-west-2.amazonaws.com/eks/kube-proxy:v1.16.8
```

Your account ID and region may differ from the example above.

5. Check your cluster's DNS provider. Clusters that were created with Kubernetes version 1.10 shipped with `kube-dns` as the default DNS and service discovery provider. If you have updated a 1.10 cluster to a newer version and you want to use CoreDNS for DNS and service discovery, then you must install CoreDNS and remove `kube-dns`.

   To check if your cluster is already running CoreDNS, use the following command.

   ```
   kubectl get pod -n kube-system -l k8s-app=kube-dns
   ```

   If the output shows `coredns` in the pod names, you're already running CoreDNS in your cluster. If not, see Installing or upgrading CoreDNS (p. 175) to install CoreDNS on your cluster, update it to the recommended version, return here, and skip steps 6-8.

6. Check the current version of your cluster's `coredns` deployment.

   ```
   kubectl describe deployment coredns --namespace kube-system | grep Image | cut -d "/" -
   f 3
   ```

   Output:

   ```
   coredns:v1.1.3
   ```

   The recommended `coredns` versions for the corresponding Kubernetes versions are as follows:

Kubernetes version	1.16	1.15	1.14	1.13
CoreDNS	1.6.6	1.6.6	1.6.6	1.6.6

7. If your current `coredns` version is 1.5.0 or later, but earlier than the recommended version, then skip this step. If your current version is earlier than 1.5.0, then you need to modify the config map for `coredns` to use the `forward` plug-in, rather than the `proxy` plug-in.

   a. Open the configmap with the following command.

   ```
   kubectl edit configmap coredns -n kube-system
   ```

   b. Replace `proxy` in the following line with `forward`. Save the file and exit the editor.

   ```
   proxy . /etc/resolv.conf
   ```

8. Retrieve your current `coredns` image:

```
kubectl get deployment coredns --namespace kube-system -
o=jsonpath='{$.spec.template.spec.containers[:1].image}'
```

9. Update `coredns` to the recommended version by taking the output from the previous step and replacing the version tag with your cluster's recommended `coredns` version:

```
kubectl set image --namespace kube-system deployment.apps/coredns \
          coredns=602401143452.dkr.ecr.us-west-2.amazonaws.com/eks/coredns:v1.6.6
```

10. Check the version of your cluster's Amazon VPC CNI Plugin for Kubernetes. Use the following command to print your cluster's CNI version.

```
kubectl describe daemonset aws-node --namespace kube-system | grep Image | cut -d "/" -
f 2
```

Output:

```
amazon-k8s-cni:1.5.7
```

If your CNI version is earlier than 1.6.1, then use the appropriate command below to update your CNI version to the latest recommended version:

If your cluster is in China (Beijing) (`cn-north-1`) or China (Ningxia) (`cn-northwest-1`)

```
kubectl apply -f https://raw.githubusercontent.com/aws/amazon-vpc-cni-k8s/master/
config/v1.6/aws-k8s-cni-cn.yaml
```

If your cluster is in AWS GovCloud (US-East) (`us-gov-east-1`)

```
kubectl apply -f https://raw.githubusercontent.com/aws/amazon-vpc-cni-k8s/
release-1.6/config/v1.6/aws-k8s-cni-us-gov-east-1.yaml
```

If your cluster is in AWS GovCloud (US-West) (`us-gov-west-1`)

```
kubectl apply -f https://raw.githubusercontent.com/aws/amazon-vpc-cni-k8s/
release-1.6/config/v1.6/aws-k8s-cni-us-gov-west-1.yaml
```

For all other Regions

```
kubectl apply -f https://raw.githubusercontent.com/aws/amazon-vpc-cni-k8s/
release-1.6/config/v1.6/aws-k8s-cni.yaml
```

11. (Optional) If you deployed the Kubernetes Cluster Autoscaler to your cluster prior to upgrading the cluster, update the Cluster Autoscaler to the latest version that matches the Kubernetes major and minor version that you upgraded to.

a. Open the Cluster Autoscaler releases page in a web browser and find the latest Cluster Autoscaler version that matches your cluster's Kubernetes major and minor version. For example, if your cluster's Kubernetes version is 1.16 find the latest Cluster Autoscaler release that begins with 1.16. Record the semantic version number (1.16.n) for that release to use in the next step.

b. Set the Cluster Autoscaler image tag to the version that you recorded in the previous step with the following command. Replace `1.16.n` with your own value. You can replace `us` with `asia` or `eu`.

```
kubectl -n kube-system set image deployment.apps/cluster-autoscaler cluster-autoscaler=us.gcr.io/k8s-artifacts-prod/autoscaling/cluster-autoscaler:v1.16.n
```

> **Note**
> Depending on the version that you need, you may need to change the previous address to `gcr.io/google-containers/cluster-autoscaler:v1.n.n`. The image address is listed on the releases page.

12. (Clusters with GPU workers only) If your cluster has worker node groups with GPU support (for example, `p3.2xlarge`), you must update the NVIDIA device plugin for Kubernetes DaemonSet on your cluster with the following command.

```
kubectl apply -f https://raw.githubusercontent.com/NVIDIA/k8s-device-plugin/1.0.0-beta/nvidia-device-plugin.yml
```

13. After your cluster update is complete, update your worker nodes to the same Kubernetes version of your updated cluster. For more information, see Self-managed worker node updates (p. 101). Any new pods launched on Fargate will have a `kubelet` version that matches your cluster version. Existing Fargate pods will not be changed.

Kubernetes 1.16 upgrade prerequisites

As noted in the Kubernetes 1.15 changelog, and Deprecated APIs Removed In 1.16: Here's What You Need To Know documents, if you have an existing cluster, API changes are required for the following deployed resources before upgrading a cluster to 1.16.

> **Warning**
> If you do not change these APIs before upgrading to 1.16, workloads will fail after the upgrade is complete.

- NetworkPolicy resources will no longer be served from `extensions/v1beta1` in v1.16. Migrate use to the `networking.k8s.io/v1` API, available since v1.8. Existing persisted data can be retrieved through the `networking.k8s.io/v1` API.
- PodSecurityPolicy resources will no longer be served from `extensions/v1beta1` in v1.16. Migrate to the `policy/v1beta1` API, available since v1.10. Existing persisted data can be retrieved through the `policy/v1beta1` API.
- DaemonSet, Deployment, StatefulSet, and ReplicaSet resources will no longer be served from `extensions/v1beta1`, `apps/v1beta1`, or `apps/v1beta2` in v1.16. Migrate to the `apps/v1` API, available since v1.9. Existing persisted data can be retrieved through the `apps/v1` API. For example, to convert a Deployment that currently uses `apps/v1beta1`, enter the following command.

```
kubectl convert -f ./my-deployment.yaml --output-version apps/v1
```

> **Note**
> The previous command may use different default values from what is set in your current manifest file. To learn more about a specific resource, see the Kubernetes API reference.

What you need to do before upgrading to 1.16

- Change your YAML files to reference the new APIs.
- Update custom integrations and controllers to call the new APIs.

- Ensure that you use an updated version of any third party tools, such as ingress controllers, continuous delivery systems, and others, that call the new APIs.

To easily check for deprecated API usage in your cluster, make sure that the `audit` control plane log (p. 40) is enabled, and specify `v1beta` as a filter for the events. All of the replacement APIs are in Kubernetes versions later than 1.10. Applications on any supported version of Amazon EKS can begin using the updated APIs now.

Amazon EKS cluster endpoint access control

This topic helps you to enable private access for your Amazon EKS cluster's Kubernetes API server endpoint and limit, or completely disable, public access from the internet.

When you create a new cluster, Amazon EKS creates an endpoint for the managed Kubernetes API server that you use to communicate with your cluster (using Kubernetes management tools such as `kubectl`). By default, this API server endpoint is public to the internet, and access to the API server is secured using a combination of AWS Identity and Access Management (IAM) and native Kubernetes Role Based Access Control (RBAC).

You can enable private access to the Kubernetes API server so that all communication between your worker nodes and the API server stays within your VPC. You can limit the IP addresses that can access your API server from the internet, or completely disable internet access to the API server.

> **Note**
> Because this endpoint is for the Kubernetes API server and not a traditional AWS PrivateLink endpoint for communicating with an AWS API, it doesn't appear as an endpoint in the Amazon VPC console.

When you enable endpoint private access for your cluster, Amazon EKS creates a Route 53 private hosted zone on your behalf and associates it with your cluster's VPC. This private hosted zone is managed by Amazon EKS, and it doesn't appear in your account's Route 53 resources. In order for the private hosted zone to properly route traffic to your API server, your VPC must have `enableDnsHostnames` and `enableDnsSupport` set to `true`, and the DHCP options set for your VPC must include `AmazonProvidedDNS` in its domain name servers list. For more information, see Updating DNS support for your VPC in the *Amazon VPC User Guide*.

> **Note**
> In addition to standard Amazon EKS permissions, your IAM user or role must have `route53:AssociateVPCWithHostedZone` permissions to enable the cluster's endpoint private access.

You can define your API server endpoint access requirements when you create a new cluster, and you can update the API server endpoint access for a cluster at any time.

Modifying cluster endpoint access

Use the procedures in this section to modify the endpoint access for an existing cluster. The following table shows the supported API server endpoint access combinations and their associated behavior.

API server endpoint access options

Endpoint public access	Endpoint private access	Behavior
Enabled	Disabled	- This is the default behavior for new Amazon EKS clusters. - Kubernetes API requests that originate from within

Endpoint public access	Endpoint private access	Behavior
		your cluster's VPC (such as worker node to control plane communication) leave the VPC but not Amazon's network. • Your cluster API server is accessible from the internet. You can, optionally, limit the CIDR blocks that can access the public endpoint. If you limit access to specific CIDR blocks, then it is recommended that you also enable the private endpoint, or ensure that the CIDR blocks that you specify include the addresses that worker nodes and Fargate pods (if you use them) access the public endpoint from.
Enabled	Enabled	• Kubernetes API requests within your cluster's VPC (such as worker node to control plane communication) use the private VPC endpoint. • Your cluster API server is accessible from the internet. You can, optionally, limit the CIDR blocks that can access the public endpoint.

Endpoint public access	Endpoint private access	Behavior
Disabled	Enabled	- All traffic to your cluster API server must come from within your cluster's VPC or a connected network. - There is no public access to your API server from the internet. Any `kubectl` commands must come from within the VPC or a connected network. For connectivity options, see Accessing a private only API server (p. 39). - The cluster's API server endpoint is resolved by public DNS servers to a private IP address from the VPC. In the past, the endpoint could only be resolved from within the VPC. If your endpoint does not resolve to a private IP address within the VPC for an existing cluster, you can: - Enable public access and then disable it again. You only need to do so once for a cluster and the endpoint will resolve to a private IP address from that point forward. - Update (p. 27) your cluster.

You can modify your cluster API server endpoint access using the AWS Management Console or AWS CLI. For instructions, select the tab for the tool that you want to use.

AWS Management Console

1. Open the Amazon EKS console at https://console.aws.amazon.com/eks/home#/clusters.
2. Choose the name of the cluster to display your cluster information.
3. Under **Networking**, choose **Update**.
4. For **Private access**, choose whether to enable or disable private access for your cluster's Kubernetes API server endpoint. If you enable private access, Kubernetes API requests that originate from within your cluster's VPC use the private VPC endpoint. You must enable private access to disable public access.
5. For **Public access**, choose whether to enable or disable public access for your cluster's Kubernetes API server endpoint. If you disable public access, your cluster's Kubernetes API server can only receive requests from within the cluster VPC.
6. (Optional) If you've enabled **Public access**, you can specify which addresses from the Internet can communicate to the public endpoint. Select **Advanced Settings**. Enter a CIDR block, such

as `203.0.113.5/32`. The block cannot include reserved addresses. You can enter additional blocks by selecting **Add Source**. There is a maximum number of CIDR blocks that you can specify. For more information, see Amazon EKS service quotas (p. 294). If you specify no blocks, then the public API server endpoint receives requests from all (0.0.0.0/0) IP addresses. If you restrict access to your public endpoint using CIDR blocks, it is recommended that you also enable private endpoint access so that worker nodes and Fargate pods (if you use them) can communicate with the cluster. Without the private endpoint enabled, your public access endpoint CIDR sources must include the egress sources from your VPC. For example, if you have a worker node in a private subnet that communicates to the internet through a NAT Gateway, you will need to add the outbound IP address of the NAT gateway as part of a whitelisted CIDR block on your public endpoint.

7. Choose **Update** to finish.

AWS CLI

Complete the following steps using the AWS CLI version 1.18.49 or later. You can check your current version with `aws --version`. To install or upgrade the AWS CLI, see Installing the AWS CLI.

1. Update your cluster API server endpoint access with the following AWS CLI command. Substitute your cluster name and desired endpoint access values. If you set `endpointPublicAccess=true`, then you can (optionally) enter single CIDR block, or a comma-separated list of CIDR blocks for `publicAccessCidrs`. The blocks cannot include reserved addresses. If you specify CIDR blocks, then the public API server endpoint will only receive requests from the listed blocks. There is a maximum number of CIDR blocks that you can specify. For more information, see Amazon EKS service quotas (p. 294). If you restrict access to your public endpoint using CIDR blocks, it is recommended that you also enable private endpoint access so that worker nodes and Fargate pods (if you use them) can communicate with the cluster. Without the private endpoint enabled, your public access endpoint CIDR sources must include the egress sources from your VPC. For example, if you have a worker node in a private subnet that communicates to the internet through a NAT Gateway, you will need to add the outbound IP address of the NAT gateway as part of a whitelisted CIDR block on your public endpoint. If you specify no CIDR blocks, then the public API server endpoint receives requests from all (0.0.0.0/0) IP addresses.

   Note
   The following command enables private access and public access from a single IP address for the API server endpoint. Replace `203.0.113.5/32` with a single CIDR block, or a comma-separated list of CIDR blocks that you want to restrict network access to.

```
aws eks update-cluster-config \
    --region region-code \
    --name dev \
    --resources-vpc-config
 endpointPublicAccess=true,publicAccessCidrs="203.0.113.5/32",endpointPrivateAccess=true
```

Output:

```
{
    "update": {
        "id": "e6f0905f-a5d4-4a2a-8c49-EXAMPLE00000",
        "status": "InProgress",
        "type": "EndpointAccessUpdate",
        "params": [
            {
                "type": "EndpointPublicAccess",
                "value": "true"
            },
```

```
            {
                "type": "EndpointPrivateAccess",
                "value": "true"
            },
            {
                "type": "publicAccessCidrs",
                "value": "[\203.0.113.5/32\"]"
            }
        ],
        "createdAt": 1576874258.137,
        "errors": []
    }
}
```

2. Monitor the status of your endpoint access update with the following command, using the cluster name and update ID that was returned by the previous command. Your update is complete when the status is shown as `Successful`.

```
aws eks describe-update \
    --region region-code \
    --name dev \
    --update-id e6f0905f-a5d4-4a2a-8c49-EXAMPLE00000
```

Output:

```
{
    "update": {
        "id": "e6f0905f-a5d4-4a2a-8c49-EXAMPLE00000",
        "status": "Successful",
        "type": "EndpointAccessUpdate",
        "params": [
            {
                "type": "EndpointPublicAccess",
                "value": "true"
            },
            {
                "type": "EndpointPrivateAccess",
                "value": "true"
            },
            {
                "type": "publicAccessCidrs",
                "value": "[\203.0.113.5/32\"]"
            }
        ],
        "createdAt": 1576874258.137,
        "errors": []
    }
}
```

Accessing a private only API server

If you have disabled public access for your cluster's Kubernetes API server endpoint, you can only access the API server from within your VPC or a connected network. Here are a few possible ways to access the Kubernetes API server endpoint:

- **Connected network** – Connect your network to the VPC with an AWS transit gateway or other connectivity option and then use a computer in the connected network. You must ensure that your Amazon EKS control plane security group contains rules to allow ingress traffic on port 443 from your connected network.

- **Amazon EC2 bastion host** – You can launch an Amazon EC2 instance into a public subnet in your cluster's VPC and then log in via SSH into that instance to run `kubectl` commands. For more information, see Linux bastion hosts on AWS. You must ensure that your Amazon EKS control plane security group contains rules to allow ingress traffic on port 443 from your bastion host. For more information, see Amazon EKS security group considerations (p. 157).

  When you configure `kubectl` for your bastion host, be sure to use AWS credentials that are already mapped to your cluster's RBAC configuration, or add the IAM user or role that your bastion will use to the RBAC configuration before you remove endpoint public access. For more information, see Managing users or IAM roles for your cluster (p. 193) and Unauthorized or access denied (`kubectl`) (p. 288).

- **AWS Cloud9 IDE** – AWS Cloud9 is a cloud-based integrated development environment (IDE) that lets you write, run, and debug your code with just a browser. You can create an AWS Cloud9 IDE in your cluster's VPC and use the IDE to communicate with your cluster. For more information, see Creating an environment in AWS Cloud9. You must ensure that your Amazon EKS control plane security group contains rules to allow ingress traffic on port 443 from your IDE security group. For more information, see Amazon EKS security group considerations (p. 157).

  When you configure `kubectl` for your AWS Cloud9 IDE, be sure to use AWS credentials that are already mapped to your cluster's RBAC configuration, or add the IAM user or role that your IDE will use to the RBAC configuration before you remove endpoint public access. For more information, see Managing users or IAM roles for your cluster (p. 193) and Unauthorized or access denied (`kubectl`) (p. 288).

Amazon EKS control plane logging

Amazon EKS control plane logging provides audit and diagnostic logs directly from the Amazon EKS control plane to CloudWatch Logs in your account. These logs make it easy for you to secure and run your clusters. You can select the exact log types you need, and logs are sent as log streams to a group for each Amazon EKS cluster in CloudWatch.

You can start using Amazon EKS control plane logging by choosing which log types you want to enable for each new or existing Amazon EKS cluster. You can enable or disable each log type on a per-cluster basis using the AWS Management Console, AWS CLI (version 1.16.139 or higher), or through the Amazon EKS API. When enabled, logs are automatically sent from the Amazon EKS cluster to CloudWatch Logs in the same account.

When you use Amazon EKS control plane logging, you're charged standard Amazon EKS pricing for each cluster that you run. You are charged the standard CloudWatch Logs data ingestion and storage costs for any logs sent to CloudWatch Logs from your clusters. You are also charged for any AWS resources, such as Amazon EC2 instances or Amazon EBS volumes, that you provision as part of your cluster.

The following cluster control plane log types are available. Each log type corresponds to a component of the Kubernetes control plane. To learn more about these components, see Kubernetes Components in the Kubernetes documentation.

- **Kubernetes API server component logs (`api`)** – Your cluster's API server is the control plane component that exposes the Kubernetes API. For more information, see kube-apiserver in the Kubernetes documentation.
- **Audit (`audit`)** – Kubernetes audit logs provide a record of the individual users, administrators, or system components that have affected your cluster. For more information, see Auditing in the Kubernetes documentation.
- **Authenticator (`authenticator`)** – Authenticator logs are unique to Amazon EKS. These logs represent the control plane component that Amazon EKS uses for Kubernetes Role Based Access Control (RBAC) authentication using IAM credentials. For more information, see Managing cluster authentication (p. 182).

- **Controller manager (`controllerManager`)** – The controller manager manages the core control loops that are shipped with Kubernetes. For more information, see kube-controller-manager in the Kubernetes documentation.
- **Scheduler (`scheduler`)** – The scheduler component manages when and where to run pods in your cluster. For more information, see kube-scheduler in the Kubernetes documentation.

Enabling and disabling control plane logs

By default, cluster control plane logs aren't sent to CloudWatch Logs. You must enable each log type individually to send logs for your cluster. CloudWatch Logs ingestion, archive storage, and data scanning rates apply to enabled control plane logs. For more information, see CloudWatch pricing.

When you enable a log type, the logs are sent with a log verbosity level of 2.

To enable or disable control plane logs with the console

1. Open the Amazon EKS console at https://console.aws.amazon.com/eks/home#/clusters.
2. Choose the name of the cluster to display your cluster information.
3. Under **Logging**, choose **Update**.
4. For each individual log type, choose whether the log type should be **Enabled** or **Disabled**. By default, each log type is **Disabled**.
5. Choose **Update** to finish.

To enable or disable control plane logs with the AWS CLI

1. Check your AWS CLI version with the following command.

   ```
   aws --version
   ```

   If your AWS CLI version is below 1.16.139, you must first update to the latest version. To install or upgrade the AWS CLI, see Installing the AWS command line interface in the *AWS Command Line Interface User Guide*.

2. Update your cluster's control plane log export configuration with the following AWS CLI command. Substitute your cluster name and desired endpoint access values.

   > **Note**
   > The following command sends all available log types to CloudWatch Logs.

   ```
   aws eks --region region-code update-cluster-config --name prod \
   --logging '{"clusterLogging":[{"types":
   ["api","audit","authenticator","controllerManager","scheduler"],"enabled":true}]}'
   ```

   Output:

   ```
   {
       "update": {
           "id": "883405c8-65c6-4758-8cee-2a7c1340a6d9",
           "status": "InProgress",
           "type": "LoggingUpdate",
           "params": [
               {
                   "type": "ClusterLogging",
                   "value": "{\"clusterLogging\":[{\"types\":[\"api\",\"audit\",
   \"authenticator\",\"controllerManager\",\"scheduler\"],\"enabled\":true}]}"
   ```

```
            }
        ],
        "createdAt": 1553271814.684,
        "errors": []
    }
}
```

3. Monitor the status of your log configuration update with the following command, using the cluster name and the update ID that were returned by the previous command. Your update is complete when the status appears as `Successful`.

```
aws eks --region region-code describe-update --name prod --update-
id 883405c8-65c6-4758-8cee-2a7c1340a6d9
```

Output:

```
{
    "update": {
        "id": "883405c8-65c6-4758-8cee-2a7c1340a6d9",
        "status": "Successful",
        "type": "LoggingUpdate",
        "params": [
            {
                "type": "ClusterLogging",
                "value": "{\"clusterLogging\":[{\"types\":[\"api\",\"audit\",
\"authenticator\",\"controllerManager\",\"scheduler\"],\"enabled\":true}]}"
            }
        ],
        "createdAt": 1553271814.684,
        "errors": []
    }
}
```

Viewing cluster control plane logs

After you have enabled any of the control plane log types for your Amazon EKS cluster, you can view them on the CloudWatch console.

To learn more about viewing, analyzing, and managing logs in CloudWatch, see the Amazon CloudWatch Logs User Guide.

To view your cluster control plane logs on the CloudWatch console

1. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/home#logs:prefix=/ aws/eks. This URL displays your current available log groups and filters them with the `/aws/eks` prefix.
2. Choose the cluster that you want to view logs for. The log group name format is `/aws/ eks/cluster-name/cluster`.
3. Choose the log stream to view. The following list describes the log stream name format for each log type.

   Note
   As log stream data grows, the log stream names are rotated. When multiple log streams exist for a particular log type, you can view the latest log stream by looking for the log stream name with the latest **Last Event Time**.

   - **Kubernetes API server component logs (api)** – `kube-apiserver-nnn...`
   - **Audit (audit)** – `kube-apiserver-audit-nnn...`

- **Authenticator (`authenticator`)** – `authenticator-`*`nnn...`*
- **Controller manager (`controllerManager`)** – `kube-controller-manager-`*`nnn...`*
- **Scheduler (`scheduler`)** – `kube-scheduler-`*`nnn...`*

Deleting a cluster

When you're done using an Amazon EKS cluster, you should delete the resources associated with it so that you don't incur any unnecessary costs.

> **Important**
> If you have active services in your cluster that are associated with a load balancer, you must delete those services before deleting the cluster so that the load balancers are deleted properly. Otherwise, you can have orphaned resources in your VPC that prevent you from being able to delete the VPC.

Choose the tab below that corresponds to your preferred cluster deletion method.

eksctl

To delete an Amazon EKS cluster and worker nodes with `eksctl`

This procedure assumes that you have installed `eksctl`, and that your `eksctl` version is at least `0.19.0`. You can check your version with the following command:

```
eksctl version
```

For more information on installing or upgrading `eksctl`, see Installing or upgrading `eksctl` (p. 197).

> **Note**
> This procedure only works for clusters that were created with `eksctl`.

1. List all services running in your cluster.

```
kubectl get svc --all-namespaces
```

2. Delete any services that have an associated `EXTERNAL-IP` value. These services are fronted by an Elastic Load Balancing load balancer, and you must delete them in Kubernetes to allow the load balancer and associated resources to be properly released.

```
kubectl delete svc service-name
```

3. Delete the cluster and its associated worker nodes with the following command, replacing *prod* with your cluster name.

```
eksctl delete cluster --name prod
```

Output:

```
[#]   using region region-code
[#]   deleting EKS cluster "prod"
[#]   will delete stack "eksctl-prod-nodegroup-standard-workers"
[#]   waiting for stack "eksctl-prod-nodegroup-standard-workers" to get deleted
[#]   will delete stack "eksctl-prod-cluster"
```

```
[#]   the following EKS cluster resource(s) for "prod" will be deleted: cluster. If
 in doubt, check CloudFormation console
```

AWS Management Console

To delete an Amazon EKS cluster with the AWS Management Console

1. List all services running in your cluster.

    ```
    kubectl get svc --all-namespaces
    ```

2. Delete any services that have an associated EXTERNAL-IP value. These services are fronted by an Elastic Load Balancing load balancer, and you must delete them in Kubernetes to allow the load balancer and associated resources to be properly released.

    ```
    kubectl delete svc service-name
    ```

3. Delete the worker node AWS CloudFormation stack.

    a. Open the AWS CloudFormation console at https://console.aws.amazon.com/cloudformation.

    b. Select the worker node stack to delete and then choose **Actions**, **Delete Stack**.

    c. On the **Delete Stack** confirmation screen, choose **Yes, Delete**.

4. Delete the cluster.

    a. Open the Amazon EKS console at https://console.aws.amazon.com/eks/home#/clusters.

    b. Select the cluster to delete and choose **Delete**.

    c. On the delete cluster confirmation screen, choose **Delete**.

5. (Optional) Delete the VPC AWS CloudFormation stack.

    a. Select the VPC stack to delete and choose **Actions** and then **Delete Stack**.

    b. On the **Delete Stack** confirmation screen, choose **Yes, Delete**.

AWS CLI

To delete an Amazon EKS cluster with the AWS CLI

1. List all services running in your cluster.

    ```
    kubectl get svc --all-namespaces
    ```

2. Delete any services that have an associated EXTERNAL-IP value. These services are fronted by an Elastic Load Balancing load balancer, and you must delete them in Kubernetes to allow the load balancer and associated resources to be properly released.

    ```
    kubectl delete svc service-name
    ```

3. Delete the worker node AWS CloudFormation stack.

    a. List your available AWS CloudFormation stacks with the following command. Find the worker node template name in the resulting output.

    ```
    aws cloudformation list-stacks --query "StackSummaries[].StackName"
    ```

b. Delete the worker node stack with the following command, replacing *worker-node-stack* with your worker node stack name.

```
aws cloudformation delete-stack --stack-name worker-node-stack
```

4. Delete the cluster with the following command, replacing *my-cluster* with your cluster name.

```
aws eks delete-cluster --name my-cluster
```

5. (Optional) Delete the VPC AWS CloudFormation stack.

a. List your available AWS CloudFormation stacks with the following command. Find the VPC template name in the resulting output.

```
aws cloudformation list-stacks --query "StackSummaries[].StackName"
```

b. Delete the VPC stack with the following command, replacing *my-vpc-stack* with your VPC stack name.

```
aws cloudformation delete-stack --stack-name my-vpc-stack
```

Amazon EKS Kubernetes versions

The Kubernetes project is rapidly evolving with new features, design updates, and bug fixes. The community releases new Kubernetes minor versions, such as 1.16, as generally available approximately every three months, and each minor version is supported for approximately nine months after it is first released.

Available Amazon EKS Kubernetes versions

The following Kubernetes versions are currently available for new clusters in Amazon EKS:

- 1.16.8
- 1.15.11
- 1.14.9
- 1.13.12

Important
Kubernetes version 1.13 is now deprecated on Amazon EKS. On **June 30th, 2020**, Kubernetes version 1.13 will no longer be supported on Amazon EKS. On or after this date, you will no longer be able to create new 1.13 clusters, and all existing Amazon EKS clusters running Kubernetes version 1.13 will eventually be automatically updated to version 1.14. We recommend that you update any 1.13 clusters to version 1.14 or later in order to avoid service interruption. For more information, see Amazon EKS version deprecation (p. 48).
Kubernetes API versions available through Amazon EKS are officially supported by AWS, until we remove the ability to create clusters using that version. This is true even if upstream Kubernetes is no longer supporting a version available on Amazon EKS. We backport security fixes that are applicable to the Kubernetes versions supported on Amazon EKS. Existing clusters are always supported, and Amazon EKS will automatically update your cluster to a supported version if you have not done so manually by the version end of life date.

Unless your application requires a specific version of Kubernetes, we recommend that you choose the latest available Kubernetes version supported by Amazon EKS for your clusters. As new Kubernetes

versions become available in Amazon EKS, we recommend that you proactively update your clusters to use the latest available version. For more information, see Updating an Amazon EKS cluster Kubernetes version (p. 27).

Kubernetes 1.16

Kubernetes 1.16 is now available in Amazon EKS. For more information about Kubernetes 1.16, see the official release announcement.

> **Important**
>
> - Kubernetes 1.16 removes a number of deprecated APIs. Changes to your applications may be required before upgrading your cluster to 1.16. Carefully follow the 1.16 upgrade prerequisites (p. 34) before upgrading.
> - Starting with 1.16, the Amazon EKS certificate authority will honor certificate signing requests with SAN X.509 extensions, which resolves the EKS CA should honor SAN x509 extension feature request from GitHub.

The following Kubernetes features are now supported in Kubernetes 1.16 Amazon EKS clusters:

- Volume expansion in the CSI specification has moved to beta, which allows for any CSI spec volume plugin to be resizeable. For more information, see Volume Expansion in the Kubernetes CSI documentation. The latest version of the EBS CSI driver supports volume expansion when running on an Amazon EKS 1.16 cluster.
- Windows GMSA support has graduated from alpha to beta, and is now supported by Amazon EKS. For more information, see Configure GMSA for Windows Pods and containers in the Kubernetes documentation.
- A new annotation: `service.beta.kubernetes.io/aws-load-balancer-eip-allocations` is available on service type `LoadBalancer` to assign an elastic IP address to Network Load Balancers. For more information, see the Support EIP Allocations with AWS NLB GitHub issue.
- Finalizer protection for service load balancers is now in beta and enabled by default. Service load balancer finalizer protection ensures that any load balancer resources allocated for a Kubernetes Service object, such as the AWS Network Load Balancer, will be destroyed or released when the service is deleted. For more information, see Garbage Collecting Load Balancers in the Kubernetes documentation.
- The Kubernetes custom resource definitions and admission webhooks extensibility mechanisms have both reached general availability. For more information, see Custom Resources and Dynamic Admission Control in the Kubernetes documentation.
- The server-side apply feature has reached beta status and is enabled by default. For more information, see Server Side Apply in the Kubernetes documentation.
- The `CustomResourceDefaulting` feature is promoted to beta and enabled by default. Defaults may be specified in structural schemas through the `apiextensions.k8s.io/v1` API. For more information, see Specifying a structural schema in the Kubernetes documentation.

For the complete Kubernetes 1.16 changelog, see https://github.com/kubernetes/kubernetes/blob/master/CHANGELOG/CHANGELOG-1.16.md.

Kubernetes 1.15

Kubernetes 1.15 is now available in Amazon EKS. For more information about Kubernetes 1.15, see the official release announcement.

> **Important**
> Starting with 1.15, Amazon EKS no longer tags the VPC containing your cluster.

- Subnets within the VPC of your cluster are still tagged.
- VPC tags will not be modified on existing cluster upgrades to 1.15.

- For more information about VPC tagging, see .

Important
Amazon EKS has set the re-invocation policy for the Pod Identity Webhook to `IfNeeded`. This allows the webhook to be re-invoked if objects are changed by other mutating admission webhooks like the App Mesh sidecar injector. For more information about the App Mesh sidecar injector, see Install the sidecar injector.

The following features are now supported in Kubernetes 1.15 Amazon EKS clusters:

- EKS now supports configuring transport layer security (TLS) termination, access logs, and source ranges for network load balancers. For more information, see Network Load Balancer support on AWS on GitHub.
- Improved flexibility of Custom Resource Definitions (CRD), including the ability to convert between versions on the fly. For more information, see Extend the Kubernetes API with CustomResourceDefinitions on GitHub.
- NodeLocal DNSCache is in beta for Kubernetes version 1.15 clusters. This feature can help improve cluster DNS performance by running a DNS caching agent on cluster nodes as a DaemonSet. For more information, see Using NodeLocal DNSCache in Kubernetes clusters on GitHub.

For the complete Kubernetes 1.15 changelog, see https://github.com/kubernetes/kubernetes/blob/master/CHANGELOG/CHANGELOG-1.15.md.

Kubernetes 1.14

Kubernetes 1.14 is now available in Amazon EKS. For more information about Kubernetes 1.14, see the official release announcement.

Important
The `--allow-privileged` flag has been removed from `kubelet` on Amazon EKS 1.14 worker nodes. If you have modified or restricted the Amazon EKS default pod security policy (p. 269) on your cluster, you should verify that your applications have the permissions they need on 1.14 worker nodes.

The following features are now supported in Kubernetes 1.14 Amazon EKS clusters:

- Container Storage Interface Topology is in beta for Kubernetes version 1.14 clusters. For more information, see CSI Topology Feature in the Kubernetes CSI Developer Documentation. The following CSI drivers provide a CSI interface for container orchestrators like Kubernetes to manage the life cycle of Amazon EBS volumes, Amazon EFS file systems, and Amazon FSx for Lustre file systems:
  - Amazon Elastic Block Store (EBS) CSI driver
  - Amazon EFS CSI driver
  - Amazon FSx for Lustre CSI driver
- Process ID (PID) limiting is in beta for Kubernetes version 1.14 clusters. This feature allows you to set quotas for how many processes a pods can create, which can prevent resource starvation for other applications on a cluster. For more information, see Process ID limiting for stability improvements in Kubernetes 1.14.
- Persistent Local Volumes are now GA and make locally attached storage available as a persistent volume source. For more information, see Kubernetes 1.14: Local persistent volumes GA.
- Pod Priority and Preemption is now GA and allows pods to be assigned a scheduling priority level. For more information, see Pod Priority and Preemption in the Kubernetes documentation.

- Windows worker node support is GA with Kubernetes 1.14.

For the complete Kubernetes 1.14 changelog, see https://github.com/kubernetes/kubernetes/blob/master/CHANGELOG/CHANGELOG-1.14.md.

Kubernetes 1.13

The following features are now supported in Kubernetes 1.13 Amazon EKS clusters:

- The `PodSecurityPolicy` admission controller is now enabled. This admission controller allows fine-grained control over pod creation and updates. For more information, see Pod security policy (p. 269). If you do not have any pod security policies defined in your cluster when you upgrade to 1.13, then Amazon EKS creates a default policy for you.

  Important
  If you have **any** pod security policies defined in your cluster, the default policy is not created when you upgrade to Kubernetes 1.13. If your cluster does not have the default Amazon EKS pod security policy, your pods may not be able to launch if your existing pod security policies are too restrictive. You can check for any existing pod security policies with the following command:

  ```
  kubectl get psp
  ```

  If you cluster has any pod security policies defined, you should also make sure that you have the default Amazon EKS pod security policy (`eks.privileged`) defined. If not, you can apply it by following the steps in To install or restore the default pod security policy (p. 271).
- Amazon ECR interface VPC endpoints (AWS PrivateLink) are supported. When you enable these endpoints in your VPC, all network traffic between your VPC and Amazon ECR is restricted to the Amazon network. For more information, see Amazon ECR interface VPC endpoints (AWS PrivateLink) in the *Amazon Elastic Container Registry User Guide*.
- The `DryRun` feature is in beta in Kubernetes 1.13 and is enabled by default for Amazon EKS clusters. For more information, see Dry run in the Kubernetes documentation.
- The `TaintBasedEvictions` feature is in beta in Kubernetes 1.13 and is enabled by default for Amazon EKS clusters. For more information, see Taint based evictions in the Kubernetes documentation.
- Raw Block Volume Support is in beta in Kubernetes 1.13 and is enabled by default for Amazon EKS clusters. This is accessible via the `volumeDevices` container field in pod specs, and the `volumeMode` field in persistent volume and persistent volume claim definitions. For more information, see Raw Block Volume Support in the Kubernetes documentation.
- Node lease renewal is treated as the heartbeat signal from the node, in addition to its `NodeStatus` update. This reduces load on the control plane for large clusters. For more information, see https://github.com/kubernetes/kubernetes/pull/69241.

For the complete Kubernetes 1.13 changelog, see https://github.com/kubernetes/kubernetes/blob/master/CHANGELOG/CHANGELOG-1.13.md.

Amazon EKS version deprecation

In line with the Kubernetes community support for Kubernetes versions, Amazon EKS is committed to running at least three production-ready versions of Kubernetes at any given time, with a fourth version in deprecation.

We will announce the deprecation of a given Kubernetes minor version at least 60 days before the end of support date. Because of the Amazon EKS qualification and release process for new Kubernetes versions,

the deprecation of a Kubernetes version on Amazon EKS will be on or after the date the Kubernetes project stops supporting the version upstream.

On the end of support date, Amazon EKS clusters running the deprecated version will begin to be automatically updated to the next Amazon EKS-supported version of Kubernetes. For example, if the deprecated version is 1.13, clusters will eventually be automatically updated to version 1.14. If a cluster is automatically updated by Amazon EKS, you must update the version of your worker nodes after the update is complete. For more information, see Self-managed worker node updates (p. 101).

Kubernetes supports compatibility between masters and workers for at least two minor versions, so 1.13 workers will continue to operate when orchestrated by a 1.14 control plane. For more information, see Kubernetes version and version skew support policy in the Kubernetes documentation.

Platform versions

Amazon EKS platform versions represent the capabilities of the cluster control plane, such as which Kubernetes API server flags are enabled, as well as the current Kubernetes patch version. Each Kubernetes minor version has one or more associated Amazon EKS platform versions. The platform versions for different Kubernetes minor versions are independent.

When a new Kubernetes minor version is available in Amazon EKS, such as 1.16, the initial Amazon EKS platform version for that Kubernetes minor version starts at `eks.1`. However, Amazon EKS releases new platform versions periodically to enable new Kubernetes control plane settings and to provide security fixes.

When new Amazon EKS platform versions become available for a minor version:

- The Amazon EKS platform version number is incremented (`eks.`*n+1*).
- Amazon EKS automatically upgrades all existing clusters to the latest Amazon EKS platform version for their corresponding Kubernetes minor version.
- Amazon EKS might publish a new worker AMI with a corresponding patch version. However, all patch versions are compatible between the EKS control plane and worker AMIs for a given Kubernetes minor version.

New Amazon EKS platform versions don't introduce breaking changes or cause service interruptions.

> **Note**
> Automatic upgrades of existing Amazon EKS platform versions are rolled out incrementally. The roll-out process might take some time. If you need the latest Amazon EKS platform version features immediately, you should create a new Amazon EKS cluster.

Clusters are always created with the latest available Amazon EKS platform version (`eks.`*n*) for the specified Kubernetes version. If you update your cluster to a new Kubernetes minor version, your cluster receives the current Amazon EKS platform version for the Kubernetes minor version that you updated to.

The current and recent Amazon EKS platform versions are described in the following tables.

Kubernetes version 1.16

Kubernetes version	Amazon EKS platform version	Enabled admission controllers	Release notes
1.16.8	eks.1	NamespaceLifecycle, LimitRanger,	Initial release of Kubernetes 1.16 for

Kubernetes version	Amazon EKS platform version	Enabled admission controllers	Release notes
		ServiceAccount, DefaultStorageClass, ResourceQuota, DefaultTolerationSeconds, NodeRestriction, MutatingAdmissionWebhook, ValidatingAdmissionWebhook, PodSecurityPolicy, TaintNodesByCondition, Priority, StorageObjectInUseProtection, PersistentVolumeClaimResize	Amazon EKS. For more information, see Kubernetes 1.15 (p. 46).

Kubernetes version 1.15

Kubernetes version	Amazon EKS platform version	Enabled admission controllers	Release notes
1.15.11	eks.2	NamespaceLifecycle, LimitRanger, ServiceAccount, DefaultStorageClass, ResourceQuota, DefaultTolerationSeconds, NodeRestriction, MutatingAdmissionWebhook, ValidatingAdmissionWebhook, PodSecurityPolicy, TaintNodesByCondition, Priority, StorageObjectInUseProtection, PersistentVolumeClaimResize	New platform version with bug fixes and enhancements, including an update to the server side AWS IAM Authenticator, with IAM role traceability improvements.
1.15.10	eks.1	NamespaceLifecycle, LimitRanger, ServiceAccount, DefaultStorageClass, ResourceQuota, DefaultTolerationSeconds, NodeRestriction, MutatingAdmissionWebhook, ValidatingAdmissionWebhook, PodSecurityPolicy, TaintNodesByCondition, Priority, StorageObjectInUseProtection, PersistentVolumeClaimResize	Initial release of Kubernetes 1.15 for Amazon EKS. For more information, see Kubernetes 1.15 (p. 46).

Kubernetes version 1.14

Kubernetes version	Amazon EKS platform version	Enabled admission controllers	Release notes
1.14.9	eks.9	NamespaceLifecycle, LimitRanger, ServiceAccount, DefaultStorageClass, ResourceQuota, DefaultTolerationSeconds, NodeRestriction, MutatingAdmissionWebhook, ValidatingAdmissionWebhook, PodSecurityPolicy	New platform version with enhancements and additional feature support.
1.14.9	eks.8	NamespaceLifecycle, LimitRanger, ServiceAccount, DefaultStorageClass, ResourceQuota, DefaultTolerationSeconds, NodeRestriction, MutatingAdmissionWebhook, ValidatingAdmissionWebhook, PodSecurityPolicy	New platform version with security and bug fixes.
1.14.9	eks.7	NamespaceLifecycle, LimitRanger, ServiceAccount, DefaultStorageClass, ResourceQuota, DefaultTolerationSeconds, NodeRestriction, MutatingAdmissionWebhook, ValidatingAdmissionWebhook, PodSecurityPolicy	New platform version with security fixes.
1.14.9	eks.6	NamespaceLifecycle, LimitRanger, ServiceAccount, DefaultStorageClass, ResourceQuota, DefaultTolerationSeconds, NodeRestriction, MutatingAdmissionWebhook, ValidatingAdmissionWebhook, PodSecurityPolicy	New platform version updating Amazon EKS Kubernetes 1.14 clusters to 1.14.9, various bug fixes, and performance improvements.
1.14.8	eks.5	NamespaceLifecycle, LimitRanger, ServiceAccount, DefaultStorageClass, ResourceQuota, DefaultTolerationSeconds, NodeRestriction, MutatingAdmissionWebhook,	New platform version adding support for AWS Fargate (p. 110).

Kubernetes version	Amazon EKS platform version	Enabled admission controllers	Release notes
		ValidatingAdmissionWebhook, PodSecurityPolicy	
1.14.8	eks.4	NamespaceLifecycle, LimitRanger, ServiceAccount, DefaultStorageClass, ResourceQuota, DefaultTolerationSeconds, NodeRestriction, MutatingAdmissionWebhook, ValidatingAdmissionWebhook, PodSecurityPolicy	New platform version for various bug fixes and performance improvements.
1.14.8	eks.3	NamespaceLifecycle, LimitRanger, ServiceAccount, DefaultStorageClass, ResourceQuota, DefaultTolerationSeconds, NodeRestriction, MutatingAdmissionWebhook, ValidatingAdmissionWebhook, PodSecurityPolicy	New platform version adding support for Managed node groups (p. 85).
1.14.8	eks.2	NamespaceLifecycle, LimitRanger, ServiceAccount, DefaultStorageClass, ResourceQuota, DefaultTolerationSeconds, NodeRestriction, MutatingAdmissionWebhook, ValidatingAdmissionWebhook, PodSecurityPolicy	New platform version updating Amazon EKS Kubernetes 1.14 clusters to 1.14.8 to address CVE-2019-11253.
1.14.6	eks.1	NamespaceLifecycle, LimitRanger, ServiceAccount, DefaultStorageClass, ResourceQuota, DefaultTolerationSeconds, NodeRestriction, MutatingAdmissionWebhook, ValidatingAdmissionWebhook, PodSecurityPolicy	Initial release of Kubernetes 1.14 for Amazon EKS. For more information, see Kubernetes 1.14 (p. 47).

Kubernetes version 1.13

Kubernetes version	Amazon EKS platform version	Enabled admission controllers	Release notes
1.13.12	eks.8	NamespaceLifecycle, LimitRanger, ServiceAccount, DefaultStorageClass, ResourceQuota, DefaultTolerationSeconds, NodeRestriction, MutatingAdmissionWebhook, ValidatingAdmissionWebhook, PodSecurityPolicy	New platform version with enhancements and additional feature support.
1.13.12	eks.7	NamespaceLifecycle, LimitRanger, ServiceAccount, DefaultStorageClass, ResourceQuota, DefaultTolerationSeconds, NodeRestriction, MutatingAdmissionWebhook, ValidatingAdmissionWebhook, PodSecurityPolicy	New platform version with security fixes.
1.13.12	eks.6	NamespaceLifecycle, LimitRanger, ServiceAccount, DefaultStorageClass, ResourceQuota, DefaultTolerationSeconds, NodeRestriction, MutatingAdmissionWebhook, ValidatingAdmissionWebhook, PodSecurityPolicy	New platform version updating Amazon EKS Kubernetes 1.13 clusters to 1.13.12 to address CVE-2019-11253.
1.13.11	eks.5	NamespaceLifecycle, LimitRanger, ServiceAccount, DefaultStorageClass, ResourceQuota, DefaultTolerationSeconds, NodeRestriction, MutatingAdmissionWebhook, ValidatingAdmissionWebhook, PodSecurityPolicy	New platform version for various bug fixes and performance improvements.
1.13.10	eks.4	NamespaceLifecycle, LimitRanger, ServiceAccount, DefaultStorageClass, ResourceQuota, DefaultTolerationSeconds, NodeRestriction, MutatingAdmissionWebhook,	New platform version to support IAM roles for service accounts. For more information, see IAM roles for service accounts (p. 253).

Kubernetes version	Amazon EKS platform version	Enabled admission controllers	Release notes
		`ValidatingAdmissionWebhook,` `PodSecurityPolicy`	
`1.13.10`	`eks.3`	`NamespaceLifecycle,` `LimitRanger,` `ServiceAccount,` `DefaultStorageClass,` `ResourceQuota,` `DefaultTolerationSeconds,` `NodeRestriction,` `MutatingAdmissionWebhook,` `ValidatingAdmissionWebhook,` `PodSecurityPolicy`	New platform version updating Amazon EKS Kubernetes 1.13 clusters to a patched version of 1.13.10 to address CVE-2019-9512 and CVE-2019-9514.
`1.13.8`	`eks.2`	`NamespaceLifecycle,` `LimitRanger,` `ServiceAccount,` `DefaultStorageClass,` `ResourceQuota,` `DefaultTolerationSeconds,` `NodeRestriction,` `MutatingAdmissionWebhook,` `ValidatingAdmissionWebhook,` `PodSecurityPolicy`	New platform version updating Amazon EKS Kubernetes 1.13 clusters to a patched version of 1.13.8 to address CVE-2019-11247 and CVE-2019-11249.
`1.13.7`	`eks.1`	`NamespaceLifecycle,` `LimitRanger,` `ServiceAccount,` `DefaultStorageClass,` `ResourceQuota,` `DefaultTolerationSeconds,` `NodeRestriction,` `MutatingAdmissionWebhook,` `ValidatingAdmissionWebhook,` `PodSecurityPolicy`	Initial release of Kubernetes 1.13 for Amazon EKS. For more information, see Kubernetes 1.13 (p. 48).

Windows support

This topic describes how to add Windows support to Amazon EKS clusters.

Considerations

Before deploying Windows worker nodes, be aware of the following considerations.

- Windows workloads are supported with Amazon EKS clusters running Kubernetes version 1.14 or later.
- Amazon EC2 instance types C3, C4, D2, I2, M4 (excluding m4.16xlarge), and R3 instances are not supported for Windows workloads.
- Host networking mode is not supported for Windows workloads.
- Amazon EKS clusters must contain one or more Linux worker nodes to run core system pods that only run on Linux, such as `coredns` and the VPC resource controller.
- The `kubelet` and `kube-proxy` event logs are redirected to the `EKS` Windows Event Log and are set to a 200 MB limit.

- Windows worker nodes support one elastic network interface per node. The number of pods that you can run per Windows worker node is equal to the number of IP addresses available per elastic network interface for the node's instance type, minus one. For more information, see IP addresses per network interface per instance type in the *Amazon EC2 User Guide for Linux Instances*.
- Calico network policy enforcement has not been tested with Amazon EKS Windows nodes.
- Group Managed Service Accounts (GMSA) for Windows pods and containers is a Kubernetes 1.14 alpha feature that is not supported by Amazon EKS. You can follow the instructions in the Kubernetes documentation to enable and test this alpha feature on your clusters.

Enabling Windows support

The following steps help you to enable Windows support for your Amazon EKS cluster. Choose the tab below to use standard tools on your client operating system.

eksctl

To enable Windows support for your cluster with `eksctl`

This procedure only works for clusters that were created with `eksctl` and assumes that your `eksctl` version is `0.19.0` or later. You can check your version with the following command.

```
eksctl version
```

For more information about installing or upgrading `eksctl`, see Installing or upgrading `eksctl` (p. 197).

1. Enable Windows support for your Amazon EKS cluster with the following `eksctl` command. This command deploys the VPC resource controller and VPC admission controller webhook that are required on Amazon EKS clusters to run Windows workloads.

```
eksctl utils install-vpc-controllers --cluster cluster_name --approve
```

2. After you have enabled Windows support, you can launch a Windows node group into your cluster. For more information, see Launching Amazon EKS Windows worker nodes (p. 97).

After you add Windows support to your cluster, you must specify node selectors on your applications so that the pods land on a node with the appropriate operating system. For Linux pods, use the following node selector text in your manifests.

```
nodeSelector:
       kubernetes.io/os: linux
       kubernetes.io/arch: amd64
```

For Windows pods, use the following node selector text in your manifests.

```
nodeSelector:
       kubernetes.io/os: windows
       kubernetes.io/arch: amd64
```

Windows

To enable Windows support for your cluster with a Windows client

In the following steps, replace the `region-code` with the Region that your cluster resides in.

1. Deploy the VPC resource controller to your cluster.

    ```
    kubectl apply -f https://amazon-eks.s3.us-west-2.amazonaws.com/manifests/region-code/vpc-resource-controller/latest/vpc-resource-controller.yaml
    ```

2. Deploy the VPC admission controller webhook to your cluster.

    a. Download the required scripts and deployment files.

    ```
    curl -o vpc-admission-webhook-deployment.yaml https://amazon-eks.s3.us-west-2.amazonaws.com/manifests/region-code/vpc-admission-webhook/latest/vpc-admission-webhook-deployment.yaml;
    curl -o Setup-VPCAdmissionWebhook.ps1 https://amazon-eks.s3.us-west-2.amazonaws.com/manifests/region-code/vpc-admission-webhook/latest/Setup-VPCAdmissionWebhook.ps1;
    curl -o webhook-create-signed-cert.ps1 https://amazon-eks.s3.us-west-2.amazonaws.com/manifests/region-code/vpc-admission-webhook/latest/webhook-create-signed-cert.ps1;
    curl -o webhook-patch-ca-bundle.ps1 https://amazon-eks.s3.us-west-2.amazonaws.com/manifests/region-code/vpc-admission-webhook/latest/webhook-patch-ca-bundle.ps1;
    ```

    b. Install OpenSSL and jq.

    c. Set up and deploy the VPC admission webhook.

    ```
    ./Setup-VPCAdmissionWebhook.ps1 -DeploymentTemplate ".\vpc-admission-webhook-deployment.yaml"
    ```

3. Determine if your cluster has the required cluster role binding.

    ```
    kubectl get clusterrolebinding eks:kube-proxy-windows
    ```

    If output similar to the following example output is returned, then the cluster has the necessary role binding.

    ```
    NAME                       AGE
    eks:kube-proxy-windows     10d
    ```

    If the output includes `Error from server (NotFound)`, then the cluster does not have the necessary cluster role binding. Add the binding by creating a file named `eks-kube-proxy-windows-crb.yaml` with the following content.

    ```
    kind: ClusterRoleBinding
    apiVersion: rbac.authorization.k8s.io/v1beta1
    metadata:
      name: eks:kube-proxy-windows
      labels:
        k8s-app: kube-proxy
        eks.amazonaws.com/component: kube-proxy
    subjects:
      - kind: Group
        name: "eks:kube-proxy-windows"
    roleRef:
      kind: ClusterRole
      name: system:node-proxier
      apiGroup: rbac.authorization.k8s.io
    ```

    Apply the configuration to the cluster.

```
kubectl apply -f eks-kube-proxy-windows-crb.yaml
```

4. After you have enabled Windows support, you can launch a Windows node group into your
    cluster. For more information, see .

After you add Windows support to your cluster, you must specify node selectors on your applications
so that the pods land on a node with the appropriate operating system. For Linux pods, use the
following node selector text in your manifests.

```
nodeSelector:
        kubernetes.io/os: linux
        kubernetes.io/arch: amd64
```

For Windows pods, use the following node selector text in your manifests.

```
nodeSelector:
        kubernetes.io/os: windows
        kubernetes.io/arch: amd64
```

macOS and Linux

To enable Windows support for your cluster with a macOS or Linux client

This procedure requires that the `openssl` library and `jq` JSON processor are installed on your client
system.

In the following steps, replace `region-code` with the Region that your cluster resides in.

1. Deploy the VPC resource controller to your cluster.

    ```
    kubectl apply -f https://amazon-eks.s3.us-west-2.amazonaws.com/manifests/region-code/vpc-resource-controller/latest/vpc-resource-controller.yaml
    ```

2. Create the VPC admission controller webhook manifest for your cluster.

    a. Download the required scripts and deployment files.

    ```
    curl -o webhook-create-signed-cert.sh https://amazon-eks.s3.us-west-2.amazonaws.com/manifests/region-code/vpc-admission-webhook/latest/webhook-create-signed-cert.sh
    curl -o webhook-patch-ca-bundle.sh https://amazon-eks.s3.us-west-2.amazonaws.com/manifests/region-code/vpc-admission-webhook/latest/webhook-patch-ca-bundle.sh
    curl -o vpc-admission-webhook-deployment.yaml https://amazon-eks.s3.us-west-2.amazonaws.com/manifests/region-code/vpc-admission-webhook/latest/vpc-admission-webhook-deployment.yaml
    ```

    b. Add permissions to the shell scripts so that they can be executed.

    ```
    chmod +x webhook-create-signed-cert.sh webhook-patch-ca-bundle.sh
    ```

    c. Create a secret for secure communication.

    ```
    ./webhook-create-signed-cert.sh
    ```

    d. Verify the secret.

```
kubectl get secret -n kube-system vpc-admission-webhook-certs
```

e. Configure the webhook and create a deployment file.

```
cat ./vpc-admission-webhook-deployment.yaml | ./webhook-patch-ca-bundle.sh >
 vpc-admission-webhook.yaml
```

3. Deploy the VPC admission webhook.

```
kubectl apply -f vpc-admission-webhook.yaml
```

4. Determine if your cluster has the required cluster role binding.

```
kubectl get clusterrolebinding eks:kube-proxy-windows
```

If output similar to the following example output is returned, then the cluster has the necessary role binding.

```
NAME                      AGE
eks:kube-proxy-windows    10d
```

If the output includes `Error from server (NotFound)`, then the cluster does not have the necessary cluster role binding. Add the binding by creating a file named `eks-kube-proxy-windows-crb.yaml` with the following content.

```
kind: ClusterRoleBinding
apiVersion: rbac.authorization.k8s.io/v1beta1
metadata:
  name: eks:kube-proxy-windows
  labels:
    k8s-app: kube-proxy
    eks.amazonaws.com/component: kube-proxy
subjects:
  - kind: Group
    name: "eks:kube-proxy-windows"
roleRef:
  kind: ClusterRole
  name: system:node-proxier
  apiGroup: rbac.authorization.k8s.io
```

Apply the configuration to the cluster.

```
kubectl apply -f eks-kube-proxy-windows-crb.yaml
```

5. After you have enabled Windows support, you can launch a Windows node group into your cluster. For more information, see Launching Amazon EKS Windows worker nodes (p. 97).

After you add Windows support to your cluster, you must specify node selectors on your applications so that the pods land on a node with the appropriate operating system. For Linux pods, use the following node selector text in your manifests.

```
nodeSelector:
      kubernetes.io/os: linux
      kubernetes.io/arch: amd64
```

For Windows pods, use the following node selector text in your manifests.

```
nodeSelector:
        kubernetes.io/os: windows
        kubernetes.io/arch: amd64
```

Deploy a Windows sample application

To deploy a Windows sample application

1. Create a file named `windows-server-iis.yaml` with the following content.

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: windows-server-iis
spec:
  selector:
    matchLabels:
        app: windows-server-iis
        tier: backend
        track: stable
  replicas: 1
  template:
    metadata:
      labels:
        app: windows-server-iis
        tier: backend
        track: stable
    spec:
      containers:
      - name: windows-server-iis
        image: mcr.microsoft.com/windows/servercore:1809
        ports:
        - name: http
          containerPort: 80
        imagePullPolicy: IfNotPresent
        command:
        - powershell.exe
        - -command
        - "Add-WindowsFeature Web-Server; Invoke-WebRequest -UseBasicParsing
 -Uri 'https://dotnetbinaries.blob.core.windows.net/servicemonitor/2.0.1.6/
ServiceMonitor.exe' -OutFile 'C:\\ServiceMonitor.exe'; echo '<html><body><br/
><br/><marquee><H1>Hello EKS!!!<H1><marquee></body><html>' > C:\\inetpub\\wwwroot\
\default.html; C:\\ServiceMonitor.exe 'w3svc'; "
      nodeSelector:
        kubernetes.io/os: windows
---
apiVersion: v1
kind: Service
metadata:
  name: windows-server-iis-service
  namespace: default
spec:
  ports:
  - port: 80
    protocol: TCP
    targetPort: 80
  selector:
    app: windows-server-iis
    tier: backend
```

```
      track: stable
  sessionAffinity: None
  type: LoadBalancer
```

2. Deploy the application to the cluster.

```
kubectl apply -f windows-server-iis.yaml
```

3. Get the status of the pod.

```
kubectl get pods -o wide --watch
```

Wait for the pod to transition to the `Running` state.

4. Query the services in your cluster and wait until the **External IP** column for the `windows-server-iis-service` service is populated.

> **Note**
> It might take several minutes for the IP address to become available.

```
kubectl get services -o wide
```

5. After your external IP address is available, point a web browser to that address to view the IIS home page.

> **Note**
> It might take several minutes for DNS to propagate and for your sample application to load in your web browser.

Arm support

This topic describes how to create an Amazon EKS cluster and add worker nodes running on Amazon EC2 A1 instances to Amazon EKS clusters. Amazon EC2 A1 instances deliver significant cost savings for scale-out and Arm-based applications such as web servers, containerized microservices, caching fleets, and distributed data stores.

> **Note**
> These instructions and the assets that they reference are offered as a beta feature that is administered by AWS. Use of these instructions and assets is governed as a beta under the AWS service terms. While in beta, Amazon EKS does not support using Amazon EC2 A1 instances for production Kubernetes workloads. Submit comments or questions in a GitHub issue.

Considerations

- Worker nodes can be any A1 instance type, but all worker nodes must be an A1 instance type.
- Worker nodes must be deployed with Kubernetes version 1.13 or 1.14.
- To use A1 instance worker nodes, you must setup a new Amazon EKS cluster. You cannot add worker nodes to a cluster that has existing worker nodes.

Prerequisites

- Have `eksctl` installed on your computer. If you don't have it installed, see Install `eksctl` (p. 4) for installation instructions.
- Have `kubectl` and the AWS IAM authenticator installed on your computer. If you don't have them installed, see Installing `kubectl` (p. 182) for installation instructions.

Create a cluster

1. Run the following command to create an Amazon EKS cluster with no worker nodes. If you want to create a cluster running Kubernetes version 1.13, then replace *1.14* with `1.13` in your command. You can replace *region-code* with any Region that Amazon EKS is available in.

    ```
    eksctl create cluster \
    --name a1-preview \
    --version 1.14 \
    --region region-code \
    --without-nodegroup
    ```

    Launching an Amazon EKS cluster using `eksctl` creates an AWS CloudFormation stack. The launch process for this stack typically takes 10 to 15 minutes. You can monitor the progress in the Amazon EKS console.

2. When the cluster creation completes, open the AWS CloudFormation console. You will see a stack named `eksctl-`*a1-preview*`-cluster`. Select this stack. Select the **Resources** tab. Record the values of the IDs for the `ControlPlaneSecurityGroup` and `VPC` resources.

3. Confirm that the cluster is running with the `kubectl get svc` command. The command returns output similar to the following example output.

    ```
    NAME         TYPE        CLUSTER-IP    EXTERNAL-IP    PORT(S)    AGE
    kubernetes   ClusterIP   10.100.0.1    <none>         443/TCP    20m
    ```

Enable Arm support

To support having only A1 nodes in an Amazon EKS cluster, you need to update some of the Kubernetes components. Complete the following steps to update CoreDNS and `kube-proxy`, and install the Amazon VPC ARM64 CNI Plugin for Kubernetes.

1. Update the CoreDNS image ID using the command that corresponds to the version of the cluster that you installed in a previous step.

    Kubernetes 1.14

    ```
    kubectl apply -f https://raw.githubusercontent.com/aws/containers-roadmap/master/
    preview-programs/eks-arm-preview/dns-arm-1.14.yaml
    ```

    Kubernetes 1.13

    ```
    kubectl apply -f https://raw.githubusercontent.com/aws/containers-roadmap/master/
    preview-programs/eks-arm-preview/dns-arm-1.13.yaml
    ```

2. Update the `kube-proxy` image ID using the command that corresponds to the version of the cluster that you installed in a previous step.

    Kubernetes 1.14

    ```
    kubectl apply -f https://raw.githubusercontent.com/aws/containers-roadmap/master/
    preview-programs/eks-arm-preview/kube-proxy-arm-1.14.yaml
    ```

    Kubernetes 1.13

```
kubectl apply -f https://raw.githubusercontent.com/aws/containers-roadmap/master/
preview-programs/eks-arm-preview/kube-proxy-arm-1.13.yaml
```

3. Deploy the Amazon VPC ARM64 CNI Plugin for Kubernetes.

```
kubectl apply -f https://raw.githubusercontent.com/aws/containers-roadmap/master/
preview-programs/eks-arm-preview/aws-k8s-cni-arm64.yaml
```

Launch worker nodes

1. Open the AWS CloudFormation console. Ensure that you are in the AWS Region that you created your Amazon EKS cluster in.

2. Choose **Create stack**, and then choose **With new resources (standard).**

3. For **Specify template**, select **Amazon S3 URL**, enter the following URL into the **Amazon S3 URL** box, and then choose **Next** twice.

```
https://amazon-eks.s3.us-west-2.amazonaws.com/cloudformation/2020-05-08/amazon-eks-arm-
nodegroup.yaml
```

4. On the **Specify stack details** page, fill out the following parameters accordingly:

- **Stack name** – Choose a stack name for your AWS CloudFormation stack. For example, you can name it *a1-preview*-**worker-nodes**.
- **KubernetesVersion** – Select the version of Kubernetes that you chose when launching your Amazon EKS cluster.
- **ClusterName** – Enter the name that you used when you created your Amazon EKS cluster.

    Important
    This name must exactly match the name you used in Step 1: Create your Amazon EKS cluster (p. 14); otherwise, your worker nodes cannot join the cluster.

- **ClusterControlPlaneSecurityGroup** – Choose the `ControlPlaneSecurityGroup` ID value from the AWS CloudFormation output that you generated with the section called "Create a cluster" (p. 61).
- **NodeGroupName** – Enter a name for your node group. This name can be used later to identify the Auto Scaling node group that is created for your worker nodes.
- **NodeAutoScalingGroupMinSize** – Enter the minimum number of nodes that your worker node Auto Scaling group can scale in to.
- **NodeAutoScalingGroupDesiredCapacity** – Enter the desired number of nodes to scale to when your stack is created.
- **NodeAutoScalingGroupMaxSize** – Enter the maximum number of nodes that your worker node Auto Scaling group can scale out to.
- **NodeInstanceType** – Choose one of the A1 instance types for your worker nodes, such as `a1.large`.
- **NodeVolumeSize** – Specify a root volume size for your worker nodes, in GiB.
- **KeyName** – Enter the name of an Amazon EC2 SSH key pair that you can use to connect using SSH into your worker nodes with after they launch. If you don't already have an Amazon EC2 key pair, you can create one in the AWS Management Console. For more information, see Amazon EC2 key pairs in the *Amazon EC2 User Guide for Linux Instances*.

    Note
    If you do not provide a key pair here, the AWS CloudFormation stack creation fails.

- **BootstrapArguments** – Arguments to pass to the bootstrap script. For details, see https://github.com/awslabs/amazon-eks-ami/blob/master/files/bootstrap.sh.

- **VpcId** – Enter the ID for the VPC that you created in the section called "Create a cluster" (p. 61).

- **Subnets** – Choose the subnets that you created in the section called "Create a cluster" (p. 61).

  > **Important**
  > If any of the subnets are public subnets, then they must have the automatic public IP address assignment setting enabled. If the setting is not enabled for the public subnet, then any worker nodes that you deploy to that public subnet will not be assigned a public IP address and will not be able to communicate with the cluster or other AWS services. If the subnet was deployed before 03/26/2020 using either of the Amazon EKS AWS CloudFormation VPC templates (p. 152), or by using `eksctl`, then automatic public IP address assignment is disabled for public subnets. For information about how to enable public IP address assignment for a subnet, see Modifying the Public IPv4 Addressing Attribute for Your Subnet. If the worker node is deployed to a private subnet, then it is able to communicate with the cluster and other AWS services through a NAT gateway.

- **NodeImageAMI113** – The Amazon EC2 Systems Manager parameter for the 1.13 AMI image ID. This value is ignored if you selected `1.14` for `KubernetesVersion`.

- **NodeImageAMI114** – The Amazon EC2 Systems Manager parameter for the 1.14 AMI image ID. This value is ignored if you selected `1.13` for `KubernetesVersion`.

5. Choose **Next** and then choose **Next** again.

6. Acknowledge that the stack might create IAM resources, and then choose **Create stack**.

7. When your stack has finished creating, select it in the console and choose **Outputs**.

8. Record the **NodeInstanceRole** for the node group that was created. You need this when you configure your Amazon EKS worker nodes.

Join worker nodes to a cluster

1. Download, edit, and apply the AWS IAM Authenticator configuration map.

   a. Use the following command to download the configuration map:

   ```
   wget https://amazon-eks.s3.us-west-2.amazonaws.com/cloudformation/2019-11-15/aws-
   auth-cm.yaml
   ```

   b. Open the file with your favorite text editor. Replace the *<ARN of instance role (not instance profile)>* snippet with the **NodeInstanceRole** value that you recorded in the previous procedure, and save the file.

   > **Important**
   > Do not modify any other lines in this file.

   ```
   apiVersion: v1
   kind: ConfigMap
   metadata:
     name: aws-auth
     namespace: kube-system
   data:
     mapRoles: |
       - rolearn: <ARN of instance role (not instance profile)>
         username: system:node:{{EC2PrivateDNSName}}
         groups:
           - system:bootstrappers
           - system:nodes
   ```

   c. Apply the configuration. This command may take a few minutes to finish.

```
kubectl apply -f aws-auth-cm.yaml
```

> **Note**
> If you receive the error `"aws-iam-authenticator": executable file not found in $PATH`, your **kubectl** isn't configured for Amazon EKS. For more information, see Installing `aws-iam-authenticator` (p. 187).
> If you receive any other authorization or resource type errors, see Unauthorized or access denied (`kubectl`) (p. 288) in the troubleshooting section.

2. Watch the status of your nodes and wait for them to reach the `Ready` status.

```
kubectl get nodes --watch
```

(Optional) Deploy an application

To confirm that you can deploy and run an application on the worker nodes, complete the following steps.

1. Deploy the CNI metrics helper with the following command.

```
kubectl apply -f https://raw.githubusercontent.com/aws/containers-roadmap/master/
preview-programs/eks-arm-preview/cni-metrics-helper-arm64.yaml
```

The output returned is similar to the following example output.

```
clusterrole.rbac.authorization.k8s.io/cni-metrics-helper created
serviceaccount/cni-metrics-helper created
clusterrolebinding.rbac.authorization.k8s.io/cni-metrics-helper created
deployment.extensions/cni-metrics-helper created
```

2. Confirm that the CNI metrics helper is running with the following command.

```
kubectl -n kube-system get pods -o wide
```

The pod is running if you see the `cni-metrics-helper` pod returned in the output.

Viewing API server flags

You can use the control plane logging feature for Amazon EKS clusters to view the API server flags that were enabled when a cluster was created. For more information, see Amazon EKS control plane logging (p. 40). This topic shows you how to view the API server flags for an Amazon EKS cluster in the Amazon CloudWatch console.

When a cluster is first created, the initial API server logs include the flags that were used to start the API server. If you enable API server logs when you launch the cluster, or shortly thereafter, these logs are sent to CloudWatch Logs and you can view them there.

To view API server flags for a cluster

1. If you have not already done so, enable API server logs for your Amazon EKS cluster.

    a. Open the Amazon EKS console at https://console.aws.amazon.com/eks/home#/clusters.

b. Choose the name of the cluster to display your cluster information.

c. Under **Logging**, choose **Update**.

d. For **API server**, make sure that the log type is **Enabled**.

e. Choose **Update** to finish.

2. In the Amazon EKS console, scroll down to the **Logging** section of the cluster detail page. Choose the link under **CloudWatch** to open the CloudWatch console page for your cluster's logs.

3. In the list of log streams, find the earliest version of the `kube-apiserver-`*`example-`* *`ID-288ec988b77a59d70ec77`* log stream. Use the **Last Event Time** column to determine the log stream ages.

4. Scroll up to the earliest events (the beginning of the log stream). You should see the initial API server flags for the cluster.

> **Note**
> If you don't see the API server logs at the beginning of the log stream, then it is likely that the API server log file was rotated on the server before you enabled API server logging on the server. Any log files that are rotated before API server logging is enabled cannot be exported to CloudWatch.
> However, you can create a new cluster with the same Kubernetes version and enable the API server logging when you create the cluster. Clusters with the same platform version have the same flags enabled, so your flags should match the new cluster's flags. When you finish viewing the flags for the new cluster in CloudWatch, you can delete the new cluster.

Worker nodes

Worker machines in Kubernetes are called nodes. Amazon EKS worker nodes run in your AWS account and connect to your cluster's control plane via the cluster API server endpoint. You deploy one or more worker nodes into a node group. A node group is one or more Amazon EC2 instances that are deployed in an Amazon EC2 Auto Scaling group. All instances in a node group must:

- Be the same instance type
- Be running the same Amazon Machine Image (AMI)
- Use the same Amazon EKS worker node IAM role (p. 250)

A cluster can contain several node groups, and each node group can contain several worker nodes. If you deploy managed node groups (p. 85), then there is a maximum number of nodes that can be in a node group and a maximum number of node groups that you can have within a cluster. See service quotas (p. 294) for details. With this information, you can determine how many node groups you may need in a cluster to meet your requirements.

Amazon EKS worker nodes are standard Amazon EC2 instances, and you are billed for them based on normal EC2 prices. For more information, see Amazon EC2 pricing.

Amazon EKS provides a specialized Amazon Machine Image (AMI) called the Amazon EKS-optimized AMI. This AMI is built on top of Amazon Linux 2, and is configured to serve as the base image for Amazon EKS worker nodes. The AMI is configured to work with Amazon EKS out of the box, and it includes Docker, **kubelet**, and the AWS IAM Authenticator. The AMI also contains a specialized bootstrap script that allows it to discover and connect to your cluster's control plane automatically.

> **Note**
> You can track security or privacy events for Amazon Linux 2 at the Amazon Linux security center or subscribe to the associated RSS feed. Security and privacy events include an overview of the issue, what packages are affected, and how to update your instances to correct the issue.

Beginning with Kubernetes version 1.14 and platform version (p. 49) `eks.3`, Amazon EKS clusters support Managed node groups (p. 85), which automate the provisioning and lifecycle management of nodes. Earlier versions of Amazon EKS clusters can launch worker nodes with an Amazon EKS-provided AWS CloudFormation template.

If you restrict access to your cluster's public endpoint using CIDR blocks, it is recommended that you also enable private endpoint access so that worker nodes can communicate with the cluster. Without the private endpoint enabled, the CIDR blocks that you specify for public access must include the egress sources from your VPC. For more information, see Amazon EKS cluster endpoint access control (p. 35).

To add worker nodes to your Amazon EKS cluster, see Launching Amazon EKS Linux worker nodes (p. 92). If you follow the steps in the guide, the required tag is added to the worker node for you. If you launch workers manually, you must add the following tag to each worker node. For more information, see Adding and deleting tags on an individual resource.

Key	Value
`kubernetes.io/cluster/`*`<cluster-name>`*	`owned`

For more information about worker nodes from a general Kubernetes perspective, see Nodes in the Kubernetes documentation.

Topics

Amazon EKS-optimized Linux AMI

The Amazon EKS-optimized Linux AMI is built on top of Amazon Linux 2, and is configured to serve as the base image for Amazon EKS worker nodes. The AMI is configured to work with Amazon EKS out of the box, and it includes Docker, **kubelet**, and the AWS IAM Authenticator.

> **Note**
> You can track security or privacy events for Amazon Linux 2 at the Amazon Linux security center or subscribe to the associated RSS feed. Security and privacy events include an overview of the issue, what packages are affected, and how to update your instances to correct the issue.

The AMI IDs for the latest Amazon EKS-optimized AMI (with and without GPU support (p. 72)) are shown in the following table. You can also retrieve the IDs with an AWS Systems Manager parameter using different tools. For more information, see Retrieving Amazon EKS-optimized AMI IDs (p. 80).

> **Note**
> The Amazon EKS-optimized AMI with GPU support only supports GPU instance types. Be sure to specify these instance types in your worker node AWS CloudFormation template. By using the Amazon EKS-optimized AMI with GPU support, you agree to NVIDIA's end user license agreement (EULA).

Kubernetes version 1.16.8

Region	Amazon EKS-optimized AMI	with GPU support
US East (Ohio) (`us-east-2`)	View AMI ID	View AMI ID
US East (N. Virginia) (`us-east-1`)	View AMI ID	View AMI ID
US West (Oregon) (`us-west-2`)	View AMI ID	View AMI ID
Asia Pacific (Hong Kong) (`ap-east-1`)	View AMI ID	View AMI ID
Asia Pacific (Mumbai) (`ap-south-1`)	View AMI ID	View AMI ID
Asia Pacific (Tokyo) (`ap-northeast-1`)	View AMI ID	View AMI ID
Asia Pacific (Seoul) (`ap-northeast-2`)	View AMI ID	View AMI ID
Asia Pacific (Singapore) (`ap-southeast-1`)	View AMI ID	View AMI ID

Region	Amazon EKS-optimized AMI	with GPU support
Asia Pacific (Sydney) (`ap-southeast-2`)	View AMI ID	View AMI ID
Canada (Central) (`ca-central-1`)	View AMI ID	View AMI ID
China (Beijing) (`cn-north-1`)	View AMI ID	View AMI ID
China (Ningxia) (`cn-northwest-1`)	View AMI ID	View AMI ID
Europe (Frankfurt) (`eu-central-1`)	View AMI ID	View AMI ID
Europe (Ireland) (`eu-west-1`)	View AMI ID	View AMI ID
Europe (London) (`eu-west-2`)	View AMI ID	View AMI ID
Europe (Paris) (`eu-west-3`)	View AMI ID	View AMI ID
Europe (Stockholm) (`eu-north-1`)	View AMI ID	View AMI ID
Middle East (Bahrain) (`me-south-1`)	View AMI ID	View AMI ID
South America (São Paulo) (`sa-east-1`)	View AMI ID	View AMI ID
AWS GovCloud (US-East) (`us-gov-east-1`)	View AMI ID	View AMI ID
AWS GovCloud (US-West) (`us-gov-west-1`)	View AMI ID	View AMI ID

Kubernetes version 1.15.11

Region	Amazon EKS-optimized AMI	with GPU support
US East (Ohio) (`us-east-2`)	View AMI ID	View AMI ID
US East (N. Virginia) (`us-east-1`)	View AMI ID	View AMI ID
US West (Oregon) (`us-west-2`)	View AMI ID	View AMI ID
Asia Pacific (Hong Kong) (`ap-east-1`)	View AMI ID	View AMI ID
Asia Pacific (Mumbai) (`ap-south-1`)	View AMI ID	View AMI ID
Asia Pacific (Tokyo) (`ap-northeast-1`)	View AMI ID	View AMI ID

Region	Amazon EKS-optimized AMI	with GPU support
Asia Pacific (Seoul) (ap-northeast-2)	View AMI ID	View AMI ID
Asia Pacific (Singapore) (ap-southeast-1)	View AMI ID	View AMI ID
Asia Pacific (Sydney) (ap-southeast-2)	View AMI ID	View AMI ID
Canada (Central) (ca-central-1)	View AMI ID	View AMI ID
China (Beijing) (cn-north-1)	View AMI ID	View AMI ID
China (Ningxia) (cn-northwest-1)	View AMI ID	View AMI ID
Europe (Frankfurt) (eu-central-1)	View AMI ID	View AMI ID
Europe (Ireland) (eu-west-1)	View AMI ID	View AMI ID
Europe (London) (eu-west-2)	View AMI ID	View AMI ID
Europe (Paris) (eu-west-3)	View AMI ID	View AMI ID
Europe (Stockholm) (eu-north-1)	View AMI ID	View AMI ID
Middle East (Bahrain) (me-south-1)	View AMI ID	View AMI ID
South America (São Paulo) (sa-east-1)	View AMI ID	View AMI ID
AWS GovCloud (US-East) (us-gov-east-1)	View AMI ID	View AMI ID
AWS GovCloud (US-West) (us-gov-west-1)	View AMI ID	View AMI ID

Kubernetes version 1.14.9

Region	Amazon EKS-optimized AMI	with GPU support
US East (Ohio) (us-east-2)	View AMI ID	View AMI ID
US East (N. Virginia) (us-east-1)	View AMI ID	View AMI ID
US West (Oregon) (us-west-2)	View AMI ID	View AMI ID
Asia Pacific (Hong Kong) (ap-east-1)	View AMI ID	View AMI ID

Region	Amazon EKS-optimized AMI	with GPU support
Asia Pacific (Mumbai) (`ap-south-1`)	View AMI ID	View AMI ID
Asia Pacific (Tokyo) (`ap-northeast-1`)	View AMI ID	View AMI ID
Asia Pacific (Seoul) (`ap-northeast-2`)	View AMI ID	View AMI ID
Asia Pacific (Singapore) (`ap-southeast-1`)	View AMI ID	View AMI ID
Asia Pacific (Sydney) (`ap-southeast-2`)	View AMI ID	View AMI ID
Canada (Central) (`ca-central-1`)	View AMI ID	View AMI ID
China (Beijing) (`cn-north-1`)	View AMI ID	View AMI ID
China (Ningxia) (`cn-northwest-1`)	View AMI ID	View AMI ID
Europe (Frankfurt) (`eu-central-1`)	View AMI ID	View AMI ID
Europe (Ireland) (`eu-west-1`)	View AMI ID	View AMI ID
Europe (London) (`eu-west-2`)	View AMI ID	View AMI ID
Europe (Paris) (`eu-west-3`)	View AMI ID	View AMI ID
Europe (Stockholm) (`eu-north-1`)	View AMI ID	View AMI ID
Middle East (Bahrain) (`me-south-1`)	View AMI ID	View AMI ID
South America (São Paulo) (`sa-east-1`)	View AMI ID	View AMI ID
AWS GovCloud (US-East) (`us-gov-east-1`)	View AMI ID	View AMI ID
AWS GovCloud (US-West) (`us-gov-west-1`)	View AMI ID	View AMI ID

Kubernetes version 1.13.12

Region	Amazon EKS-optimized AMI	with GPU support
US East (Ohio) (`us-east-2`)	View AMI ID	View AMI ID
US East (N. Virginia) (`us-east-1`)	View AMI ID	View AMI ID

Region	Amazon EKS-optimized AMI	with GPU support
US West (Oregon) (`us-west-2`)	View AMI ID	View AMI ID
Asia Pacific (Hong Kong) (`ap-east-1`)	View AMI ID	View AMI ID
Asia Pacific (Mumbai) (`ap-south-1`)	View AMI ID	View AMI ID
Asia Pacific (Tokyo) (`ap-northeast-1`)	View AMI ID	View AMI ID
Asia Pacific (Seoul) (`ap-northeast-2`)	View AMI ID	View AMI ID
Asia Pacific (Singapore) (`ap-southeast-1`)	View AMI ID	View AMI ID
Asia Pacific (Sydney) (`ap-southeast-2`)	View AMI ID	View AMI ID
Canada (Central) (`ca-central-1`)	View AMI ID	View AMI ID
China (Beijing) (`cn-north-1`)	View AMI ID	View AMI ID
China (Ningxia) (`cn-northwest-1`)	View AMI ID	View AMI ID
Europe (Frankfurt) (`eu-central-1`)	View AMI ID	View AMI ID
Europe (Ireland) (`eu-west-1`)	View AMI ID	View AMI ID
Europe (London) (`eu-west-2`)	View AMI ID	View AMI ID
Europe (Paris) (`eu-west-3`)	View AMI ID	View AMI ID
Europe (Stockholm) (`eu-north-1`)	View AMI ID	View AMI ID
Middle East (Bahrain) (`me-south-1`)	View AMI ID	View AMI ID
South America (São Paulo) (`sa-east-1`)	View AMI ID	View AMI ID
AWS GovCloud (US-East) (`us-gov-east-1`)	View AMI ID	View AMI ID
AWS GovCloud (US-West) (`us-gov-west-1`)	View AMI ID	View AMI ID

Important
These AMIs require the latest AWS CloudFormation worker node template. You can't use these AMIs with a previous version of the worker node template; they will fail to join your cluster. Be sure to upgrade any existing AWS CloudFormation worker stacks with the latest template (URL shown below) before you attempt to use these AMIs.

```
https://amazon-eks.s3.us-west-2.amazonaws.com/cloudformation/2020-05-08/amazon-eks-
nodegroup.yaml
```

The AWS CloudFormation worker node template launches your worker nodes with Amazon EC2 user data that triggers a specialized bootstrap script. This script allows your worker nodes to discover and connect to your cluster's control plane automatically. For more information, see Launching Amazon EKS Linux worker nodes (p. 92).

Amazon EKS-optimized AMI build scripts

Amazon Elastic Kubernetes Service (Amazon EKS) has open-sourced the build scripts that are used to build the Amazon EKS-optimized AMI. These build scripts are now available on GitHub.

The Amazon EKS-optimized AMI is built on top of Amazon Linux 2, specifically for use as a worker node in Amazon EKS clusters. You can use this repository to view the specifics of how the Amazon EKS team configures **kubelet**, Docker, the AWS IAM Authenticator for Kubernetes, and more.

The build scripts repository includes a HashiCorp packer template and build scripts to generate an AMI. These scripts are the source of truth for Amazon EKS-optimized AMI builds, so you can follow the GitHub repository to monitor changes to our AMIs. For example, perhaps you want your own AMI to use the same version of Docker that the EKS team uses for the official AMI.

The GitHub repository also contains the specialized bootstrap script that runs at boot time to configure your instance's certificate data, control plane endpoint, cluster name, and more.

Additionally, the GitHub repository contains our Amazon EKS worker node AWS CloudFormation templates. These templates make it easier to spin up an instance running the Amazon EKS-optimized AMI and register it with a cluster.

For more information, see the repositories on GitHub at https://github.com/awslabs/amazon-eks-ami.

Amazon EKS-optimized AMI with GPU support

The Amazon EKS-optimized AMI with GPU support is built on top of the standard Amazon EKS-optimized AMI, and is configured to serve as an optional image for Amazon EKS worker nodes to support GPU workloads.

In addition to the standard Amazon EKS-optimized AMI configuration, the GPU AMI includes the following:

- NVIDIA drivers
- The `nvidia-docker2` package
- The `nvidia-container-runtime` (as the default runtime)

The AMI IDs for the latest Amazon EKS-optimized AMI with GPU support are shown in the following table. You can also retrieve the IDs with an AWS Systems Manager parameter using different tools. For more information, see Retrieving Amazon EKS-optimized AMI IDs (p. 80).

> **Note**
> The Amazon EKS-optimized AMI with GPU support only supports GPU instance types. Be sure to specify these instance types in your worker node AWS CloudFormation template. By using the Amazon EKS-optimized AMI with GPU support, you agree to NVIDIA's end user license agreement (EULA).

Kubernetes version 1.16.8

Region	Amazon EKS-optimized AMI with GPU support
US East (Ohio) (us-east-2)	View AMI ID
US East (N. Virginia) (us-east-1)	View AMI ID
US West (Oregon) (us-west-2)	View AMI ID
Asia Pacific (Hong Kong) (ap-east-1)	View AMI ID
Asia Pacific (Mumbai) (ap-south-1)	View AMI ID
Asia Pacific (Tokyo) (ap-northeast-1)	View AMI ID
Asia Pacific (Seoul) (ap-northeast-2)	View AMI ID
Asia Pacific (Singapore) (ap-southeast-1)	View AMI ID
Asia Pacific (Sydney) (ap-southeast-2)	View AMI ID
Canada (Central) (ca-central-1)	View AMI ID
China (Beijing) (cn-north-1)	View AMI ID
China (Ningxia) (cn-northwest-1)	View AMI ID
Europe (Frankfurt) (eu-central-1)	View AMI ID
Europe (Ireland) (eu-west-1)	View AMI ID
Europe (London) (eu-west-2)	View AMI ID
Europe (Paris) (eu-west-3)	View AMI ID
Europe (Stockholm) (eu-north-1)	View AMI ID
Middle East (Bahrain) (me-south-1)	View AMI ID
South America (São Paulo) (sa-east-1)	View AMI ID
AWS GovCloud (US-East) (us-gov-east-1)	View AMI ID
AWS GovCloud (US-West) (us-gov-west-1)	View AMI ID

Kubernetes version 1.15.11

Region	Amazon EKS-optimized AMI with GPU support
US East (Ohio) (us-east-2)	View AMI ID
US East (N. Virginia) (us-east-1)	View AMI ID
US West (Oregon) (us-west-2)	View AMI ID
Asia Pacific (Hong Kong) (ap-east-1)	View AMI ID
Asia Pacific (Mumbai) (ap-south-1)	View AMI ID
Asia Pacific (Tokyo) (ap-northeast-1)	View AMI ID

Region	Amazon EKS-optimized AMI with GPU support
Asia Pacific (Seoul) (`ap-northeast-2`)	View AMI ID
Asia Pacific (Singapore) (`ap-southeast-1`)	View AMI ID
Asia Pacific (Sydney) (`ap-southeast-2`)	View AMI ID
Canada (Central) (`ca-central-1`)	View AMI ID
China (Beijing) (`cn-north-1`)	View AMI ID
China (Ningxia) (`cn-northwest-1`)	View AMI ID
Europe (Frankfurt) (`eu-central-1`)	View AMI ID
Europe (Ireland) (`eu-west-1`)	View AMI ID
Europe (London) (`eu-west-2`)	View AMI ID
Europe (Paris) (`eu-west-3`)	View AMI ID
Europe (Stockholm) (`eu-north-1`)	View AMI ID
Middle East (Bahrain) (`me-south-1`)	View AMI ID
South America (São Paulo) (`sa-east-1`)	View AMI ID
AWS GovCloud (US-East) (`us-gov-east-1`)	View AMI ID
AWS GovCloud (US-West) (`us-gov-west-1`)	View AMI ID

Kubernetes version 1.14.9

Region	Amazon EKS-optimized AMI with GPU support
US East (Ohio) (`us-east-2`)	View AMI ID
US East (N. Virginia) (`us-east-1`)	View AMI ID
US West (Oregon) (`us-west-2`)	View AMI ID
Asia Pacific (Hong Kong) (`ap-east-1`)	View AMI ID
Asia Pacific (Mumbai) (`ap-south-1`)	View AMI ID
Asia Pacific (Tokyo) (`ap-northeast-1`)	View AMI ID
Asia Pacific (Seoul) (`ap-northeast-2`)	View AMI ID
Asia Pacific (Singapore) (`ap-southeast-1`)	View AMI ID
Asia Pacific (Sydney) (`ap-southeast-2`)	View AMI ID
Canada (Central) (`ca-central-1`)	View AMI ID
China (Beijing) (`cn-north-1`)	View AMI ID
China (Ningxia) (`cn-northwest-1`)	View AMI ID
Europe (Frankfurt) (`eu-central-1`)	View AMI ID

Region	Amazon EKS-optimized AMI with GPU support
Europe (Ireland) (`eu-west-1`)	View AMI ID
Europe (London) (`eu-west-2`)	View AMI ID
Europe (Paris) (`eu-west-3`)	View AMI ID
Europe (Stockholm) (`eu-north-1`)	View AMI ID
Middle East (Bahrain) (`me-south-1`)	View AMI ID
South America (São Paulo) (`sa-east-1`)	View AMI ID
AWS GovCloud (US-East) (`us-gov-east-1`)	View AMI ID
AWS GovCloud (US-West) (`us-gov-west-1`)	View AMI ID

Kubernetes version 1.13.12

Region	Amazon EKS-optimized AMI with GPU support
US East (Ohio) (`us-east-2`)	View AMI ID
US East (N. Virginia) (`us-east-1`)	View AMI ID
US West (Oregon) (`us-west-2`)	View AMI ID
Asia Pacific (Hong Kong) (`ap-east-1`)	View AMI ID
Asia Pacific (Mumbai) (`ap-south-1`)	View AMI ID
Asia Pacific (Tokyo) (`ap-northeast-1`)	View AMI ID
Asia Pacific (Seoul) (`ap-northeast-2`)	View AMI ID
Asia Pacific (Singapore) (`ap-southeast-1`)	View AMI ID
Asia Pacific (Sydney) (`ap-southeast-2`)	View AMI ID
Canada (Central) (`ca-central-1`)	View AMI ID
China (Beijing) (`cn-north-1`)	View AMI ID
China (Ningxia) (`cn-northwest-1`)	View AMI ID
Europe (Frankfurt) (`eu-central-1`)	View AMI ID
Europe (Ireland) (`eu-west-1`)	View AMI ID
Europe (London) (`eu-west-2`)	View AMI ID
Europe (Paris) (`eu-west-3`)	View AMI ID
Europe (Stockholm) (`eu-north-1`)	View AMI ID
Middle East (Bahrain) (`me-south-1`)	View AMI ID
South America (São Paulo) (`sa-east-1`)	View AMI ID
AWS GovCloud (US-East) (`us-gov-east-1`)	View AMI ID

Region	Amazon EKS-optimized AMI with GPU support
AWS GovCloud (US-West) (us-gov-west-1)	View AMI ID

Important
These AMIs require the latest AWS CloudFormation worker node template. You can't use these AMIs with a previous version of the worker node template; they will fail to join your cluster. Be sure to upgrade any existing AWS CloudFormation worker stacks with the latest template (URL shown below) before you attempt to use these AMIs.

```
https://amazon-eks.s3.us-west-2.amazonaws.com/cloudformation/2020-05-08/amazon-eks-
nodegroup.yaml
```

The AWS CloudFormation worker node template launches your worker nodes with Amazon EC2 user data that triggers a specialized bootstrap script. This script allows your worker nodes to discover and connect to your cluster's control plane automatically. For more information, see Launching Amazon EKS Linux worker nodes (p. 92).

After your GPU worker nodes join your cluster, you must apply the NVIDIA device plugin for Kubernetes as a DaemonSet on your cluster with the following command.

```
kubectl apply -f https://raw.githubusercontent.com/NVIDIA/k8s-device-plugin/1.0.0-beta/
nvidia-device-plugin.yml
```

You can verify that your nodes have allocatable GPUs with the following command:

```
kubectl get nodes "-o=custom-columns=NAME:.metadata.name,GPU:.status.allocatable.nvidia
\.com/gpu"
```

Example GPU manifest

This section provides an example pod manifest for you to test that your GPU workers are configured properly.

Example Get `nvidia-smi` output

This example pod manifest launches a Cuda container that runs `nvidia-smi` on a worker node. Create a file called `nvidia-smi.yaml`, copy and paste the following manifest into it, and save the file.

```
apiVersion: v1
kind: Pod
metadata:
  name: nvidia-smi
spec:
  restartPolicy: OnFailure
  containers:
  - name: nvidia-smi
    image: nvidia/cuda:9.2-devel
    args:
    - "nvidia-smi"
    resources:
      limits:
        nvidia.com/gpu: 1
```

Apply the manifest with the following command:

```
kubectl apply -f nvidia-smi.yaml
```

After the pod has finished running, view its logs with the following command:

```
kubectl logs nvidia-smi
```

Output:

```
Mon Aug  6 20:23:31 2018
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 396.26                 Driver Version: 396.26                     |
|-------------------------------+----------------------+----------------------+
| GPU  Name        Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M. |
|===============================+======================+======================|
|   0  Tesla V100-SXM2...  On   | 00000000:00:1C.0 Off |                    0 |
| N/A   46C    P0    47W / 300W |      0MiB / 16160MiB |      0%      Default |
+-------------------------------+----------------------+----------------------+

+-----------------------------------------------------------------------------+
| Processes:                                                       GPU Memory |
|  GPU       PID   Type   Process name                             Usage      |
|=============================================================================|
|  No running processes found                                                 |
+-----------------------------------------------------------------------------+
```

Amazon EKS-optimized Linux AMI versions

This topic lists versions of the Amazon EKS-optimized Linux AMIs and their corresponding versions of `kubelet`, Docker, the Linux kernel, and the Packer build script (p. 72) configuration.

The Amazon EKS-optimized AMI metadata, including the AMI ID, for each variant can be retrieved programmatically. For more information, see Retrieving Amazon EKS-optimized AMI IDs (p. 80).

AMIs are versioned by Kubernetes version and the release date of the AMI in the following format:

```
k8s_major_version.k8s_minor_version.k8s_patch_version-release_date
```

Amazon EKS-optimized AMI

The table below lists the current and previous versions of the Amazon EKS-optimized AMI.

Kubernetes version 1.16

AMI version	`kubelet` version	Docker version	Kernel version	Packer version
1.16.8-20200507	1.16.8	18.09.9-ce	4.14.177	v20200507
1.16.8-20200423	1.16.8	18.09.9-ce	4.14.173	v20200423

Kubernetes version 1.15

AMI version	`kubelet` version	Docker version	Kernel version	Packer version
1.15.11-20200507	1.15.11	18.09.9-ce	4.14.177	v20200507

AMI version	**`kubelet`** **version**	Docker version	Kernel version	Packer version
1.15.11-20200423	1.15.11	18.09.9-ce	4.14.173	v20200423
1.15.10-20200406	1.15.10	18.09.9-ce	4.14.173	v20200406
1.15.10-20200228	1.15.10	18.09.9-ce	4.14.165	v20200228

Kubernetes version 1.14

AMI version	**`kubelet`** **version**	Docker version	Kernel version	Packer version
1.14.9-20200507	1.14.9	18.09.9-ce	4.14.177	v20200507
1.14.9-20200423	1.14.9	18.09.9-ce	4.14.173	v20200423
1.14.9-20200406	1.14.9	18.09.9-ce	4.14.173	v20200406
1.14.9-20200406	1.14.9	18.09.9-ce	4.14.173	v20200406
1.14.9-20200228	1.14.9	18.09.9-ce	4.14.165	v20200228
1.14.9-20200122	1.14.9	18.09.9-ce	4.14.158	v20200122
1.14.8-20191213	1.14.8	18.09.9-ce	4.14.154	v20191213
1.14.7-20191119	1.14.7	18.09.9-ce	4.14.152	v20191119
1.14.7-20190927	1.14.7	18.06.1-ce	4.14.146	v20190927

Kubernetes version 1.13

AMI version	**`kubelet`** **version**	Docker version	Kernel version	Packer version
1.13.12-20200507	1.13.12	18.09.9-ce	4.14.177	v20200507
1.13.12-20200423	1.13.12	18.09.9-ce	4.14.173	v20200423
1.13.12-20200406	1.13.12	18.09.9-ce	4.14.173	v20200406
1.13.12-20200228	1.13.12	18.09.9-ce	4.14.165	v20200228
1.13.12-20200228	1.13.12	18.09.9-ce	4.14.165	v20200228
1.13.12-20200122	1.13.12	18.09.9-ce	4.14.158	v20200122
1.13.12-20191213	1.13.12	18.09.9-ce	4.14.154	v20191213
1.13.11-20191119	1.13.11	18.09.9-ce	4.14.152	v20191119
1.13.11-20190927	1.13.11	18.06.1-ce	4.14.146	v20190927

Amazon EKS-optimized AMI with GPU support

The table below lists the current and previous versions of the Amazon EKS-optimized AMI with GPU support.

Kubernetes version 1.16

AMI version	**kubelet version**	**Docker version**	**Kernel version**	**Packer version**	**NVIDIA driver version**
1.16.8-20200507	1.16.8	18.09.9-ce	4.14.177	20200507	418.87.00
1.16.8-20200423	1.16.8	18.09.9-ce	4.14.173	20200423	418.87.00

Kubernetes version 1.15

AMI version	**kubelet version**	**Docker version**	**Kernel version**	**Packer version**	**NVIDIA driver version**
1.15.11-20200507	1.15.11	18.09.9-ce	4.14.177	v20200507	418.87.00
1.15.11-20200423	1.15.11	18.09.9-ce	4.14.173	v20200423	418.87.00
1.15.10-20200406	1.15.10	18.09.9-ce	4.14.173	v20200406	418.87.00
1.15.10-20200228	1.15.10	18.09.9-ce	4.14.165	v20200228	418.87.00

Kubernetes version 1.14

AMI version	**kubelet version**	**Docker version**	**Kernel version**	**Packer version**	**NVIDIA driver version**
1.14.9-20200507	1.14.9	18.09.9-ce	4.14.177	v20200507	418.87.00
1.14.9-20200423	1.14.9	18.09.9-ce	4.14.173	v20200423	418.87.00
1.14.9-20200406	1.14.9	18.09.9-ce	4.14.173	v20200406	418.87.00
1.14.9-20200228	1.14.9	18.09.9-ce	4.14.165	v20200228	418.87.00
1.14.9-20200122	1.14.9	18.09.9-ce	4.14.158	v20200122	418.87.00
1.14.8-20191213	1.14.8	18.09.9-ce	4.14.154	v20191213	418.87.00
1.14.7-20191119	1.14.7	18.09.9-ce	4.14.152	v20191119	418.87.00
1.14.7-20190927	1.14.7	18.06.1-ce	4.14.146	v20190927	418.87.00

Kubernetes version 1.13

AMI version	**kubelet version**	**Docker version**	**Kernel version**	**Packer version**	**NVIDIA driver version**
1.13.12-20200507	1.13.12	18.09.9-ce	4.14.177	v20200507	418.87.00
1.13.12-20200423	1.13.12	18.09.9-ce	4.14.173	v20200423	418.87.00
1.13.12-20200406	1.13.12	18.09.9-ce	4.14.173	v20200406	418.87.00
1.13.12-20200228	1.13.12	18.09.9-ce	4.14.165	v20200228	418.87.00

AMI version	kubelet version	Docker version	Kernel version	Packer version	NVIDIA driver version
1.13.12-20200122	1.13.12	18.09.9-ce	4.14.158	v20200122	418.87.00
1.13.12-20191213	1.13.12	18.09.9-ce	4.14.154	v20191213	418.87.00
1.13.11-20191119	1.13.11	18.09.9-ce	4.14.152	v20191119	418.87.00
1.13.11-20190927	1.13.11	18.06.1-ce	4.14.146	v20190927	418.87.00

Retrieving Amazon EKS-optimized AMI IDs

You can programmatically retrieve the Amazon Machine Image (AMI) ID for Amazon EKS-optimized AMIs by querying the AWS Systems Manager Parameter Store API. This parameter eliminates the need for you to manually look up Amazon EKS-optimized AMI IDs. For more information about the Systems Manager Parameter Store API, see GetParameter. Your user account must have the `ssm:GetParameter` IAM permission to retrieve the Amazon EKS-optimized AMI metadata.

Select the name of the tool that you want to retrieve the AMI ID with.

AWS CLI

You can retrieve the image ID of the latest recommended Amazon EKS-optimized Amazon Linux AMI with the following command by using the sub-parameter `image_id`. Replace *1.16* with a supported version (p. 49) and *region-code* with an Amazon EKS-supported Region for which you want the AMI ID. Replace *amazon-linux-2* with `amazon-linux-2-gpu` to see the AMI with GPU ID.

```
aws ssm get-parameter --name /aws/service/eks/optimized-ami/1.16/amazon-linux-2/
recommended/image_id --region region-code --query "Parameter.Value" --output text
```

Example output:

```
ami-abcd1234efgh5678i
```

AWS Management Console

You can query for the recommended Amazon EKS-optimized AMI ID using a URL. The URL opens the Amazon EC2 Systems Manager console with the value of the ID for the parameter. In the following URL, replace *1.16* with a supported version (p. 49) and *us-west-2* with an Amazon EKS-supported Region for which you want the AMI ID. Replace *amazon-linux-2* with `amazon-linux-2-gpu` to see the AMI with GPU ID.

```
https://console.aws.amazon.com/systems-manager/parameters/%252Faws%252Fservice
%252Feks%252Foptimized-ami%252F1.16%252Famazon-linux-2%252Frecommended%252Fimage_id/
description?region=region-code
```

Amazon EKS-optimized Windows AMI

The Amazon EKS-optimized AMI is built on top of Windows Server 2019, and is configured to serve as the base image for Amazon EKS worker nodes. The AMI is configured to work with Amazon EKS out of the box, and it includes Docker, **kubelet**, and the AWS IAM Authenticator.

Note
You can track security or privacy events for Windows Server with the Microsoft security update guide.

The AMI IDs for the latest Amazon EKS-optimized AMI are shown in the following table. You can also retrieve the IDs with an AWS Systems Manager parameter using different tools. For more information, see Retrieving Amazon EKS-optimized Windows AMI IDs (p. 85).

Kubernetes version 1.16.8

Region	Amazon EKS-optimized Windows Server 2019 Full	Amazon EKS-optimized Windows Server 2019 Core	Amazon EKS-optimized Windows Server 1909 Core
US East (Ohio) (us-east-2)	View AMI ID	View AMI ID	View AMI ID
US East (N. Virginia) (us-east-1)	View AMI ID	View AMI ID	View AMI ID
US West (Oregon) (us-west-2)	View AMI ID	View AMI ID	View AMI ID
Asia Pacific (Hong Kong) (ap-east-1)	View AMI ID	View AMI ID	View AMI ID
Asia Pacific (Mumbai) (ap-south-1)	View AMI ID	View AMI ID	View AMI ID
Asia Pacific (Tokyo) (ap-northeast-1)	View AMI ID	View AMI ID	View AMI ID
Asia Pacific (Seoul) (ap-northeast-2)	View AMI ID	View AMI ID	View AMI ID
Asia Pacific (Singapore) (ap-southeast-1)	View AMI ID	View AMI ID	View AMI ID
Asia Pacific (Sydney) (ap-southeast-2)	View AMI ID	View AMI ID	View AMI ID
Canada (Central) (ca-central-1)	View AMI ID	View AMI ID	View AMI ID
China (Beijing) (cn-north-1)	View AMI ID	View AMI ID	View AMI ID
China (Ningxia) (cn-northwest-1)	View AMI ID	View AMI ID	View AMI ID
Europe (Frankfurt) (eu-central-1)	View AMI ID	View AMI ID	View AMI ID
Europe (Ireland) (eu-west-1)	View AMI ID	View AMI ID	View AMI ID
Europe (London) (eu-west-2)	View AMI ID	View AMI ID	View AMI ID

Region	Amazon EKS-optimized Windows Server 2019 Full	Amazon EKS-optimized Windows Server 2019 Core	Amazon EKS-optimized Windows Server 1909 Core
Europe (Paris) (eu-west-3)	View AMI ID	View AMI ID	View AMI ID
Europe (Stockholm) (eu-north-1)	View AMI ID	View AMI ID	View AMI ID
Middle East (Bahrain) (me-south-1)	View AMI ID	View AMI ID	View AMI ID
South America (São Paulo) (sa-east-1)	View AMI ID	View AMI ID	View AMI ID
AWS GovCloud (US-East) (us-gov-east-1)	View AMI ID	View AMI ID	View AMI ID
AWS GovCloud (US-West) (us-gov-west-1)	View AMI ID	View AMI ID	View AMI ID

Kubernetes version 1.15.11

Region	Amazon EKS-optimized Windows Server 2019 Full	Amazon EKS-optimized Windows Server 2019 Core	Amazon EKS-optimized Windows Server 1909 Core
US East (Ohio) (us-east-2)	View AMI ID	View AMI ID	View AMI ID
US East (N. Virginia) (us-east-1)	View AMI ID	View AMI ID	View AMI ID
US West (Oregon) (us-west-2)	View AMI ID	View AMI ID	View AMI ID
Asia Pacific (Hong Kong) (ap-east-1)	View AMI ID	View AMI ID	View AMI ID
Asia Pacific (Mumbai) (ap-south-1)	View AMI ID	View AMI ID	View AMI ID
Asia Pacific (Tokyo) (ap-northeast-1)	View AMI ID	View AMI ID	View AMI ID
Asia Pacific (Seoul) (ap-northeast-2)	View AMI ID	View AMI ID	View AMI ID
Asia Pacific (Singapore) (ap-southeast-1)	View AMI ID	View AMI ID	View AMI ID
Asia Pacific (Sydney) (ap-southeast-2)	View AMI ID	View AMI ID	View AMI ID

Region	Amazon EKS-optimized Windows Server 2019 Full	Amazon EKS-optimized Windows Server 2019 Core	Amazon EKS-optimized Windows Server 1909 Core
Canada (Central) (`ca-central-1`)	View AMI ID	View AMI ID	View AMI ID
China (Beijing) (`cn-north-1`)	View AMI ID	View AMI ID	View AMI ID
China (Ningxia) (`cn-northwest-1`)	View AMI ID	View AMI ID	View AMI ID
Europe (Frankfurt) (`eu-central-1`)	View AMI ID	View AMI ID	View AMI ID
Europe (Ireland) (`eu-west-1`)	View AMI ID	View AMI ID	View AMI ID
Europe (London) (`eu-west-2`)	View AMI ID	View AMI ID	View AMI ID
Europe (Paris) (`eu-west-3`)	View AMI ID	View AMI ID	View AMI ID
Europe (Stockholm) (`eu-north-1`)	View AMI ID	View AMI ID	View AMI ID
Middle East (Bahrain) (`me-south-1`)	View AMI ID	View AMI ID	View AMI ID
South America (São Paulo) (`sa-east-1`)	View AMI ID	View AMI ID	View AMI ID
AWS GovCloud (US-East) (`us-gov-east-1`)	View AMI ID	View AMI ID	View AMI ID
AWS GovCloud (US-West) (`us-gov-west-1`)	View AMI ID	View AMI ID	View AMI ID

Kubernetes version 1.14.9

Region	Amazon EKS-optimized Windows Server 2019 Full	Amazon EKS-optimized Windows Server 2019 Core	Amazon EKS-optimized Windows Server 1909 Core
US East (Ohio) (`us-east-2`)	View AMI ID	View AMI ID	View AMI ID
US East (N. Virginia) (`us-east-1`)	View AMI ID	View AMI ID	View AMI ID
US West (Oregon) (`us-west-2`)	View AMI ID	View AMI ID	View AMI ID

Region	Amazon EKS-optimized Windows Server 2019 Full	Amazon EKS-optimized Windows Server 2019 Core	Amazon EKS-optimized Windows Server 1909 Core
Asia Pacific (Hong Kong) (`ap-east-1`)	View AMI ID	View AMI ID	View AMI ID
Asia Pacific (Mumbai) (`ap-south-1`)	View AMI ID	View AMI ID	View AMI ID
Asia Pacific (Tokyo) (`ap-northeast-1`)	View AMI ID	View AMI ID	View AMI ID
Asia Pacific (Seoul) (`ap-northeast-2`)	View AMI ID	View AMI ID	View AMI ID
Asia Pacific (Singapore) (`ap-southeast-1`)	View AMI ID	View AMI ID	View AMI ID
Asia Pacific (Sydney) (`ap-southeast-2`)	View AMI ID	View AMI ID	View AMI ID
Canada (Central) (`ca-central-1`)	View AMI ID	View AMI ID	View AMI ID
China (Beijing) (`cn-north-1`)	View AMI ID	View AMI ID	View AMI ID
China (Ningxia) (`cn-northwest-1`)	View AMI ID	View AMI ID	View AMI ID
Europe (Frankfurt) (`eu-central-1`)	View AMI ID	View AMI ID	View AMI ID
Europe (Ireland) (`eu-west-1`)	View AMI ID	View AMI ID	View AMI ID
Europe (London) (`eu-west-2`)	View AMI ID	View AMI ID	View AMI ID
Europe (Paris) (`eu-west-3`)	View AMI ID	View AMI ID	View AMI ID
Europe (Stockholm) (`eu-north-1`)	View AMI ID	View AMI ID	View AMI ID
Middle East (Bahrain) (`me-south-1`)	View AMI ID	View AMI ID	View AMI ID
South America (São Paulo) (`sa-east-1`)	View AMI ID	View AMI ID	View AMI ID
AWS GovCloud (US-East) (`us-gov-east-1`)	View AMI ID	View AMI ID	View AMI ID
AWS GovCloud (US-West) (`us-gov-west-1`)	View AMI ID	View AMI ID	View AMI ID

Retrieving Amazon EKS-optimized Windows AMI IDs

You can programmatically retrieve the Amazon Machine Image (AMI) ID for Amazon EKS-optimized AMIs by querying the AWS Systems Manager Parameter Store API. This parameter eliminates the need for you to manually look up Amazon EKS-optimized AMI IDs. For more information about the Systems Manager Parameter Store API, see GetParameter. Your user account must have the `ssm:GetParameter` IAM permission to retrieve the Amazon EKS-optimized AMI metadata.

Select the name of the tool that you want to retrieve the AMI ID with.

AWS CLI

> You can retrieve the image ID of the latest recommended Amazon EKS-optimized Windows AMI with the following command by using the sub-parameter `image_id`. You can replace *1.15* with `1.14` and *region-code* with an Amazon EKS-supported Region for which you want the AMI ID. Replace *Core* with `Full` to see the Windows Server full AMI ID. You can also replace *2019* with `1909` for the `Core` version only.
>
> ```
> aws ssm get-parameter --name /aws/service/ami-windows-latest/Windows_Server-2019-
> English-Core-EKS_Optimized-1.15/image_id --region region-code --query "Parameter.Value"
>  --output text
> ```
>
> Example output:
>
> ```
> ami-ami-00a053f1635fffea0
> ```

AWS Management Console

> You can query for the recommended Amazon EKS-optimized AMI ID using a URL. The URL opens the Amazon EC2 Systems Manager console with the value of the ID for the parameter. In the following URL, You can replace *1.15* with `1.14` and *region-code* with an Amazon EKS-supported Region for which you want the AMI ID. Replace *Core* with `Full` to see the Windows Server full AMI ID. You can also replace *2019* with `1909` for the `Core` version only.
>
> ```
> https://console.aws.amazon.com/systems-manager/parameters/%252Faws%252Fservice%252Fami-
> windows-latest%252FWindows_Server-2019-English-Core-EKS_Optimized-1.15%252Fimage_id/
> description?region=region-code
> ```

Managed node groups

Amazon EKS managed node groups automate the provisioning and lifecycle management of nodes (Amazon EC2 instances) for Amazon EKS Kubernetes clusters.

> **Note**
> Managed node groups (p. 85) are supported on Amazon EKS clusters beginning with Kubernetes version 1.14 and platform version (p. 49) `eks.3`. Existing clusters can update to version 1.14 or later to take advantage of this feature. For more information, see Updating an Amazon EKS cluster Kubernetes version (p. 27).

With Amazon EKS managed node groups, you don't need to separately provision or register the Amazon EC2 instances that provide compute capacity to run your Kubernetes applications. You can create, update, or terminate nodes for your cluster with a single operation. Nodes run using the latest Amazon EKS-optimized AMIs in your AWS account while node updates and terminations gracefully drain nodes to ensure that your applications stay available.

All managed nodes are provisioned as part of an Amazon EC2 Auto Scaling group that is managed for you by Amazon EKS. All resources including the instances and Auto Scaling groups run within your AWS

account. Each node group uses the Amazon EKS-optimized Amazon Linux 2 AMI and can run across multiple Availability Zones that you define.

You can add a managed node group to new or existing clusters using the Amazon EKS console, `eksctl`, AWS CLI, AWS API, or infrastructure as code tools including AWS CloudFormation. Nodes launched as part of a managed node group are automatically tagged for auto-discovery by the Kubernetes cluster autoscaler and you can use the node group to apply Kubernetes labels to nodes and update them at any time.

There are no additional costs to use Amazon EKS managed node groups, you only pay for the AWS resources you provision. These include Amazon EC2 instances, Amazon EBS volumes, Amazon EKS cluster hours, and any other AWS infrastructure. There are no minimum fees and no upfront commitments.

To get started with a new Amazon EKS cluster and managed node group, see Getting started with the AWS Management Console (p. 10).

To add a managed node group to an existing cluster, see Creating a managed node group (p. 87).

Managed node groups concepts

- Amazon EKS managed node groups create and manage Amazon EC2 instances for you.
- All managed nodes are provisioned as part of an Amazon EC2 Auto Scaling group that is managed for you by Amazon EKS and all resources including Amazon EC2 instances and Auto Scaling groups run within your AWS account.
- A managed node group's Auto Scaling group spans all of the subnets that you specify when you create the group.
- Amazon EKS tags managed node group resources so that they are configured to use the Kubernetes Cluster Autoscaler (p. 135).

  > **Important**
  > If you are running a stateful application across multiple Availability Zones that is backed by Amazon EBS volumes and using the Kubernetes Cluster Autoscaler (p. 135), you should configure multiple node groups, each scoped to a single Availability Zone. In addition, you should enable the `--balance-similar-node-groups` feature.

- Instances in a managed node group use the latest version of the Amazon EKS-optimized Amazon Linux 2 AMI for its cluster's Kubernetes version. You can choose between standard and GPU variants of the Amazon EKS-optimized Amazon Linux 2 AMI.
- Amazon EKS follows the shared responsibility model for CVEs and security patches on managed node groups. Because managed nodes run the Amazon EKS-optimized AMIs, Amazon EKS is responsible for building patched versions of these AMIs when bugs or issues are reported and we are able to publish a fix. However, you are responsible for deploying these patched AMI versions to your managed node groups. When updates become available, see Updating a managed node group (p. 89).
- Amazon EKS managed node groups can be launched in both public and private subnets. If you launch a managed node group in a public subnet on or after 04/22/2020, the subnet must have `MapPublicIpOnLaunch` set to true for the instances to be able to successfully join a cluster. If the public subnet was created using `eksctl` or the Amazon EKS-vended AWS CloudFormation templates (p. 152) on or after 03/26/2020, then this setting is already set to true. If the public subnets were created before 03/26/2020, then you need to change the setting manually. For more information, see Modifying the public IPv4 addressing attribute for your subnet.
- When using VPC endpoints in private subnets, you must create endpoints for `com.amazonaws.region.ecr.api`, `com.amazonaws.region.ecr.dkr`, and a gateway endpoint for Amazon S3. For more information, see Amazon ECR interface VPC endpoints (AWS PrivateLink).
- You can create multiple managed node groups within a single cluster. For example, you could create one node group with the standard Amazon EKS-optimized Amazon Linux 2 AMI for some workloads and another with the GPU variant for workloads that require GPU support.

- If your managed node group encounters a health issue, Amazon EKS returns an error message to help you to diagnose the issue. For more information, see Managed node group errors (p. 289).
- Amazon EKS adds Kubernetes labels to managed node group instances. These Amazon EKS-provided labels are prefixed with `eks.amazon.com`.
- Amazon EKS automatically drains nodes using the Kubernetes API during terminations or updates. Updates respect the pod disruption budgets that you set for your pods.
- There are no additional costs to use Amazon EKS managed node groups, you only pay for the AWS resources you provision.

Creating a managed node group

This topic helps you to launch an Amazon EKS managed node group of Linux worker nodes that register with your Amazon EKS cluster. After the nodes join the cluster, you can deploy Kubernetes applications to them.

Managed node groups (p. 85) are supported on Amazon EKS clusters beginning with Kubernetes version 1.14 and platform version (p. 49) `eks.3`. Existing clusters can update to version 1.14 or later to take advantage of this feature. For more information, see Updating an Amazon EKS cluster Kubernetes version (p. 27).

If this is your first time launching an Amazon EKS managed node group, we recommend that you follow one of our Getting started with Amazon EKS (p. 3) guides instead. The guides provide complete end-to-end walkthroughs for creating an Amazon EKS cluster with worker nodes.

> **Important**
> Amazon EKS worker nodes are standard Amazon EC2 instances, and you are billed for them based on normal Amazon EC2 prices. For more information, see Amazon EC2 Pricing.

To launch your managed node group

1. Wait for your cluster status to show as `ACTIVE`. You cannot create a managed node group for a cluster that is not yet `ACTIVE`.
2. Open the Amazon EKS console at https://console.aws.amazon.com/eks/home#/clusters.
3. Choose the name of the cluster that you want to create your managed node group in.
4. On the cluster page, choose **Add node group**.
5. On the **Configure node group** page, fill out the parameters accordingly, and then choose **Next**.

   - **Name** — Enter a unique name for your managed node group.
   - **Node IAM role name** — Choose the node instance role to use with your node group. For more information, see Amazon EKS worker node IAM role (p. 250).

     > **Important**
     > We recommend using a role that is not currently in use by any self-managed node group, or that you plan to use with a new self-managed node group. For more information, see ??? (p. 91).

   - **Subnets** — Choose the subnets to launch your managed nodes into.

     > **Important**
     > If you are running a stateful application across multiple Availability Zones that is backed by Amazon EBS volumes and using the Kubernetes Cluster Autoscaler (p. 135), you should configure multiple node groups, each scoped to a single Availability Zone. In addition, you should enable the `--balance-similar-node-groups` feature.

     > **Important**
     > If you choose a public subnet, then the subnet must have `MapPublicIpOnLaunch` set to true for the instances to be able to successfully join a cluster. If the subnet was created

using `eksctl` or the Amazon EKS-vended AWS CloudFormation templates (p. 152) on or after 03/26/2020, then this setting is already set to true. If the subnets were created with `eksctl` or the AWS CloudFormation templates before 03/26/2020, then you need to change the setting manually. For more information, see Modifying the public IPv4 addressing attribute for your subnet.

- **Remote Access** — (Optional) You can enable SSH access to the nodes in your managed node group. Enabling SSH allows you to connect to your instances and gather diagnostic information if there are issues. Complete the following steps to enable remote access.

  Note
  We highly recommend enabling remote access when you create your node group. You cannot enable remote access after the node group is created.

  1. Select the check box to **Allow remote access to nodes**.

  2. For **SSH key pair**, choose an Amazon EC2 SSH key to use. For more information, see Amazon EC2 key pairs in the Amazon EC2 User Guide for Linux Instances.

  3. For **Allow remote access from**, choose **All** to allow SSH access from anywhere on the Internet (0.0.0.0/0), or select a security group to allow SSH access from instances that belong to that security group.

- **Tags** — (Optional) You can choose to tag your Amazon EKS managed node group. These tags do not propagate to other resources in the node group, such as Auto Scaling groups or instances. For more information, see Tagging your Amazon EKS resources (p. 274).

- **Kubernetes labels** — (Optional) You can choose to apply Kubernetes labels to the nodes in your managed node group.

6. On the **Set compute configuration** page, fill out the parameters accordingly, and then choose **Next**.

- **AMI type** — Choose **Amazon Linux 2 (AL2_x86_64)** for non-GPU instances, or **Amazon Linux 2 GPU Enabled (AL2_x86_64_GPU)** for GPU instances.

- **Instance type** — Choose the instance type to use in your managed node group. Larger instance types can accommodate more pods.

- **Disk size** — Enter the disk size (in GiB) to use for your worker node root volume.

7. On the **Setup scaling policies** page, fill out the parameters accordingly, and then choose **Next**.

  Note
  Amazon EKS does not automatically scale your node group in or out. However, you can configure the Kubernetes Cluster Autoscaler (p. 135) to do this for you.

- **Minimum size** — Specify the minimum number of worker nodes that the managed node group can scale in to.

- **Maximum size** — Specify the maximum number of worker nodes that the managed node group can scale out to.

- **Desired size** — Specify the current number of worker nodes that the managed node group should maintain at launch.

8. On the **Review and create** page, review your managed node group configuration and choose **Create**.

9. Watch the status of your nodes and wait for them to reach the `Ready` status.

```
kubectl get nodes --watch
```

10. (GPU workers only) If you chose a GPU instance type and the Amazon EKS-optimized AMI with GPU support, you must apply the NVIDIA device plugin for Kubernetes as a DaemonSet on your cluster with the following command.

```
kubectl apply -f https://raw.githubusercontent.com/NVIDIA/k8s-device-plugin/1.0.0-beta/
nvidia-device-plugin.yml
```

Now that you have a working Amazon EKS cluster with worker nodes, you are ready to start installing Kubernetes add-ons and deploying applications to your cluster. The following documentation topics help you to extend the functionality of your cluster.

- Cluster Autoscaler (p. 135) — Configure the Kubernetes Cluster autoscaler to automatically adjust the number of nodes in your node groups.
- Launch a guest book application (p. 199) — Create a sample guest book application to test your cluster and Linux worker nodes.
- Deploy a Windows sample application (p. 59) — Deploy a sample application to test your cluster and Windows worker nodes.
- Tutorial: Deploy the Kubernetes Dashboard (web UI) (p. 208) — This tutorial guides you through deploying the Kubernetes dashboard to your cluster.
- Using Helm with Amazon EKS (p. 207) — The `helm` package manager for Kubernetes helps you install and manage applications on your cluster.
- Installing the Kubernetes Metrics Server (p. 202) — The Kubernetes metrics server is an aggregator of resource usage data in your cluster.
- Control plane metrics with Prometheus (p. 203) — This topic helps you deploy Prometheus into your cluster with `helm`.

Updating a managed node group

There are several use cases for updating your Amazon EKS managed node group's version or configuration:

- You have updated the Kubernetes version for your Amazon EKS cluster, and you want to update your worker nodes to use the same Kubernetes version.
- A new AMI release version is available for your managed node group. For more information, see Amazon EKS-optimized Linux AMI versions (p. 77).
- You want to adjust the minimum, maximum, or desired count of the instances in your managed node group.
- You want to add or remove Kubernetes labels from the instances in your managed node group.
- You want to add or remove AWS tags from your managed node group.

If there is a newer AMI release version for your managed node group's Kubernetes version than the one your node group is running, you can update it to use that new AMI version. If your cluster is running a newer Kubernetes version than your node group, you can update the node group to use the latest AMI release version that matches your cluster's Kubernetes version.

> **Note**
> You cannot roll back a node group to an earlier Kubernetes version or AMI version.

When a node in a managed node group is terminated due to a scaling action or update, the pods in that node are drained first. For more information, see Managed node update behavior (p. 91).

Update a node group version

Select the tab with the name of the tool that you'd like to upgrade the version with.

AWS Management Console

To update a node group version

1. Open the Amazon EKS console at https://console.aws.amazon.com/eks/home#/clusters.
2. Choose the cluster that contains the node group to update.

3. If at least one of your node groups has an update available, you'll see a notification under the cluster name letting your know how many of your node groups have an update available. In the **Node Groups** table you will see **Update now** to the right of the value in the **AMI release version** column for each node group that can be updated. Select **Update now** for a node group that you want to update.

    If you select a node group from the table and an update is available for it, you'll receive a notification on the **Node Group configuration** page. If so, you can select the **Update now** button on the **Node Group configuration** page.

    > **Note**
    > **Update now** only appears if there is an update available. If you do not see this text, then your node group is running the latest available version.

4. On the **Update AMI release version** page, select the **Available AMI release version** that you want to update to, select one of the following options for **Update strategy**, and choose **Update**.

    - **Rolling update** — This option respects pod disruption budgets for your cluster and the update fails if Amazon EKS is unable to gracefully drain the pods that are running on this node group due to a pod disruption budget issue.
    - **Force update** — This option does not respect pod disruption budgets and it forces node restarts.

eksctl

Upgrade a managed nodegroup to the latest AMI release of the same Kubernetes version that is currently deployed on the worker nodes with the following command.

```
eksctl upgrade nodegroup --name=node-group-name --cluster=cluster-name
```

You can upgrade a nodegroup to a version that is one major release later than the nodegroup's current Kubernetes version, up to the cluster's Kubernetes version. For example, if you have a cluster running Kubernetes 1.16, you can upgrade workers currently running Kubernetes 1.15 to version 1.16 with the following command.

```
eksctl upgrade nodegroup --name=node-group-name --cluster=cluster-name --kubernetes-
version=1.16
```

Edit a node group configuration

You can change some of the configuration of a managed node group.

To edit a node group configuration

1. Open the Amazon EKS console at https://console.aws.amazon.com/eks/home#/clusters.
2. Choose the cluster that contains the node group to edit.
3. Select the node group to edit, and choose **Edit**.
4. On the **Edit node group** page edit the **Group configuration** if necessary.

    - **Tags** — Add tags to or remove tags from your node group resource. These tags are only applied to the Amazon EKS node group, and they do not propagate to other resources, such as subnets or Amazon EC2 instances in the node group.
    - **Kubernetes labels** — Add or remove Kubernetes labels to the nodes in your node group. The labels shown here are only the labels that you have applied with Amazon EKS. Other labels may exist on your nodes that are not shown here.

5. On the **Edit node group** page edit the **Group size** if necessary.

    - **Minimum size** — Specify the current number of worker nodes that the managed node group should maintain.
    - **Maximum size** — Specify the maximum number of worker nodes that the managed node group can scale out to. Managed node groups can support up to 100 nodes by default.
    - **Desired size** — Specify the current number of worker nodes that the managed node group should maintain.

6. When you are finished editing, choose **Save changes**.

Managed node update behavior

When you update a managed node group version to the latest AMI release version for your node group's Kubernetes version or to a newer Kubernetes version to match your cluster, Amazon EKS triggers the following logic:

1. Amazon EKS creates a new Amazon EC2 launch template version for the Auto Scaling group associated with your node group. The new template uses the target AMI for the update.

2. The Auto Scaling group is updated to use the latest launch template with the new AMI.

3. The Auto Scaling group maximum size and desired size are incremented by twice the number of distinct Availability Zones of the Auto Scaling group to ensure at least one new instance comes up in every Availability Zone of your node group.

4. The Auto Scaling group launches a new instance with the new AMI to satisfy the increased desired size of the node group.

5. Amazon EKS checks the nodes in the node group for the `eks.amazonaws.com/nodegroup-image` label, and it cordons all of the nodes in the node group that are not labeled with the latest AMI ID. This prevents nodes that have already been updated from a previous failed update from being cordoned.

6. Amazon EKS randomly selects a node in your node group and sends a termination signal to the Auto Scaling group, Then Amazon EKS sends a signal to drain the pods from the node.* After the node is drained, it is terminated. This step is repeated until all of the nodes are using the new AMI version.

7. The Auto Scaling group maximum size and desired size are decremented by 1 to return to your pre-update values.

* If pods do not drain from a node (for example, if a pod disruption budget is too restrictive) for 15 minutes, then one of two things happens:

- If the update is not forced, then the update fails and reports an error.
- If the update is forced, then the pods that could not be drained are deleted.

Deleting a managed node group

This topic helps you to delete an Amazon EKS managed node group.

When you delete a managed node group, Amazon EKS randomly selects a node in your node group and sends a termination signal to the Auto Scaling group. Then Amazon EKS sends a signal to drain the pods from the node. If pods do not drain from a node (for example, if a pod disruption budget is too restrictive) for 15 minutes, then the pods are deleted. After the node is drained, it is terminated. This step is repeated until all of the nodes in the Auto Scaling group are terminated, and then the Auto Scaling group is deleted.

Important
If you delete a managed node group that uses a worker node IAM role that is not used by
any other managed node group in the cluster, the role is removed from the `aws-auth`
ConfigMap (p. 193). If any self-managed node groups in the cluster are using the same worker
node IAM role, the self-managed nodes will move to the `NotReady` status and cluster operation
will be disrupted. You can add the mapping back to the ConfigMap to minimize disruption.

To delete a managed node group

1. Open the Amazon EKS console at https://console.aws.amazon.com/eks/home#/clusters.
2. Choose the cluster that contains the node group to delete.
3. Select the node group to delete, and choose **Delete**.
4. On the **Delete** *nodegroup* page, type the name of the cluster in the text field and choose **Confirm**

Launching Amazon EKS Linux worker nodes

This topic helps you to launch an Auto Scaling group of Linux worker nodes that register with your
Amazon EKS cluster. After the nodes join the cluster, you can deploy Kubernetes applications to them.

If this is your first time launching Amazon EKS Linux worker nodes, we recommend that you follow one
of our Getting started with Amazon EKS (p. 3) guides instead. The guides provide complete end-to-end
walkthroughs for creating an Amazon EKS cluster with worker nodes.

Important
Amazon EKS worker nodes are standard Amazon EC2 instances, and you are billed for them
based on normal Amazon EC2 prices. For more information, see Amazon EC2 pricing.

Choose the tab below that corresponds to your desired worker node creation method:

Amazon EKS managed node groups

Managed node groups (p. 85) are supported on Amazon EKS clusters beginning with Kubernetes
version 1.14 and platform version (p. 49) `eks.3`. Existing clusters can update to version 1.14 or
later to take advantage of this feature. For more information, see Updating an Amazon EKS cluster
Kubernetes version (p. 27).

To launch your managed node group

1. Wait for your cluster status to show as `ACTIVE`. You cannot create a managed node group for a
    cluster that is not yet `ACTIVE`.
2. Open the Amazon EKS console at https://console.aws.amazon.com/eks/home#/clusters.
3. Choose the name of the cluster that you want to create your managed node group in.
4. On the cluster page, choose **Add node group**.
5. On the **Configure node group** page, fill out the parameters accordingly, and then choose **Next**.

    *   **Name** — Enter a unique name for your managed node group.
    *   **Node IAM role name** — Choose the node instance role to use with your node group. For more
        information, see Amazon EKS worker node IAM role (p. 250).

        Important
        We recommend using a role that is not currently in use by any self-managed node
        group, or that you plan to use with a new self-managed node group. For more
        information, see ??? (p. 91).
    *   **Subnets** — Choose the subnets to launch your managed nodes into.

> **Important**
> If you are running a stateful application across multiple Availability Zones
> that is backed by Amazon EBS volumes and using the Kubernetes Cluster
> Autoscaler (p. 135), you should configure multiple node groups, each scoped to a
> single Availability Zone. In addition, you should enable the `--balance-similar-`
> `node-groups` feature.

> **Important**
> If you choose a public subnet, then the subnet must have `MapPublicIpOnLaunch`
> set to true for the instances to be able to successfully join a cluster. If the subnet
> was created using `eksctl` or the Amazon EKS-vended AWS CloudFormation
> templates (p. 152) on or after 03/26/2020, then this setting is already set to true. If
> the subnets were created with `eksctl` or the AWS CloudFormation templates before
> 03/26/2020, then you need to change the setting manually. For more information,
> see Modifying the public IPv4 addressing attribute for your subnet.

- **Remote Access** — (Optional) You can enable SSH access to the nodes in your managed
  node group. Enabling SSH allows you to connect to your instances and gather diagnostic
  information if there are issues. Complete the following steps to enable remote access.

  > **Note**
  > We highly recommend enabling remote access when you create your node group. You
  > cannot enable remote access after the node group is created.

  1. Select the check box to **Allow remote access to nodes**.

  2. For **SSH key pair**, choose an Amazon EC2 SSH key to use. For more information, see
     Amazon EC2 key pairs in the Amazon EC2 User Guide for Linux Instances.

  3. For **Allow remote access from**, choose **All** to allow SSH access from anywhere on the
     Internet (0.0.0.0/0), or select a security group to allow SSH access from instances that
     belong to that security group.

- **Tags** — (Optional) You can choose to tag your Amazon EKS managed node group. These
  tags do not propagate to other resources in the node group, such as Auto Scaling groups or
  instances. For more information, see Tagging your Amazon EKS resources (p. 274).

- **Kubernetes labels** — (Optional) You can choose to apply Kubernetes labels to the nodes in
  your managed node group.

6. On the **Set compute configuration** page, fill out the parameters accordingly, and then choose
   Next.

   - **AMI type** — Choose **Amazon Linux 2 (AL2_x86_64)** for non-GPU instances, or **Amazon Linux
     2 GPU Enabled (AL2_x86_64_GPU)** for GPU instances.

   - **Instance type** — Choose the instance type to use in your managed node group. Larger
     instance types can accommodate more pods.

   - **Disk size** — Enter the disk size (in GiB) to use for your worker node root volume.

7. On the **Setup scaling policies** page, fill out the parameters accordingly, and then choose **Next**.

   > **Note**
   > Amazon EKS does not automatically scale your node group in or out. However, you can
   > configure the Kubernetes Cluster Autoscaler (p. 135) to do this for you.

   - **Minimum size** — Specify the minimum number of worker nodes that the managed node
     group can scale in to.

   - **Maximum size** — Specify the maximum number of worker nodes that the managed node
     group can scale out to.

   - **Desired size** — Specify the current number of worker nodes that the managed node group
     should maintain at launch.

8. On the **Review and create** page, review your managed node group configuration and choose
   Create.

9. Watch the status of your nodes and wait for them to reach the `Ready` status.

```
kubectl get nodes --watch
```

10. (GPU workers only) If you chose a GPU instance type and the Amazon EKS-optimized AMI with GPU support, you must apply the NVIDIA device plugin for Kubernetes as a DaemonSet on your cluster with the following command.

```
kubectl apply -f https://raw.githubusercontent.com/NVIDIA/k8s-device-plugin/1.0.0-
beta/nvidia-device-plugin.yml
```

eksctl

To launch worker nodes with `eksctl`

This procedure assumes that you have installed `eksctl`, and that your `eksctl` version is at least `0.19.0`. You can check your version with the following command:

```
eksctl version
```

For more information on installing or upgrading `eksctl`, see Installing or upgrading eksctl (p. 197).

> **Note**
> This procedure only works for clusters that were created with `eksctl`.

1. Create your worker node group with the following command. Replace the *example values* with your own values.

```
eksctl create nodegroup \
--cluster default \
--version auto \
--name standard-workers \
--node-type t3.medium \
--node-ami auto \
--nodes 3 \
--nodes-min 1 \
--nodes-max 4
```

> **Note**
> For more information on the available options for **eksctl create nodegroup**, see the project README on GitHub or view the help page with the following command.
>
> ```
> eksctl create nodegroup --help
> ```

Output:

You'll see several lines of output as the worker nodes are created. The last line of output is similar to the following example line.

```
[#]  all nodegroups have up-to-date configuration
```

2. (Optional) Launch a guest book application (p. 199) — Deploy a sample application to test your cluster and Linux worker nodes.

Self-managed nodes

These procedures have the following prerequisites:

- You have created a VPC and security group that meet the requirements for an Amazon EKS cluster. For more information, see Cluster VPC considerations (p. 155) and Amazon EKS security group considerations (p. 157). The Getting started with Amazon EKS (p. 3) guide creates a VPC that meets the requirements, or you can also follow Creating a VPC for your Amazon EKS cluster (p. 152) to create one manually.
- You have created an Amazon EKS cluster and specified that it use the VPC and security group that meet the requirements of an Amazon EKS cluster. For more information, see Creating an Amazon EKS cluster (p. 21).

To launch your self-managed worker nodes with the AWS Management Console

1. Wait for your cluster status to show as ACTIVE. If you launch your worker nodes before the cluster is active, the worker nodes will fail to register with the cluster and you will have to relaunch them.

2. Open the AWS CloudFormation console at https://console.aws.amazon.com/cloudformation

3. Choose **Create stack**.

4. For **Specify template**, select **Amazon S3 URL**, then copy the following URL, paste it into **Amazon S3 URL**, and select **Next** twice.

```
https://amazon-eks.s3.us-west-2.amazonaws.com/cloudformation/2020-05-08/amazon-eks-
nodegroup.yaml
```

5. On the **Quick create stack** page, fill out the following parameters accordingly:

- **Stack name**: Choose a stack name for your AWS CloudFormation stack. For example, you can call it **<cluster-name>-worker-nodes**.
- **ClusterName**: Enter the name that you used when you created your Amazon EKS cluster.

   Important
   This name must exactly match the name you used in Step 1: Create your Amazon EKS cluster (p. 14); otherwise, your worker nodes cannot join the cluster.

- **ClusterControlPlaneSecurityGroup**: Choose the **SecurityGroups** value from the AWS CloudFormation output that you generated with Create your Amazon EKS cluster VPC (p. 11).
- **NodeGroupName**: Enter a name for your node group. This name can be used later to identify the Auto Scaling node group that is created for your worker nodes.
- **NodeAutoScalingGroupMinSize**: Enter the minimum number of nodes that your worker node Auto Scaling group can scale in to.
- **NodeAutoScalingGroupDesiredCapacity**: Enter the desired number of nodes to scale to when your stack is created.
- **NodeAutoScalingGroupMaxSize**: Enter the maximum number of nodes that your worker node Auto Scaling group can scale out to.
- **NodeInstanceType**: Choose an instance type for your worker nodes.

   Note
   The supported instance types for the latest version of the Amazon VPC CNI plugin for Kubernetes are shown here. You may need to update your CNI version to take advantage of the latest supported instance types. For more information, see Amazon VPC CNI plugin for Kubernetes upgrades (p. 174).

   Important
   Some instance types might not be available in all Regions.

- **NodeImageIdSSMParam**: Pre-populated with the Amazon EC2 Systems Manager parameter of the current recommended Amazon EKS-Optimized Linux AMI ID. If you want to use the AMI with GPU, then replace *amazon-linux-2* with `amazon-linux-2-gpu`. If you want to use a different Kubernetes minor version supported with Amazon EKS, then you can replace *1.16* with either `1.15`, `1.14`, or `1.13`.

    Note
    The Amazon EKS worker node AMI is based on Amazon Linux 2. You can track security or privacy events for Amazon Linux 2 at the Amazon Linux Security Center or subscribe to the associated RSS feed. Security and privacy events include an overview of the issue, what packages are affected, and how to update your instances to correct the issue.

- **NodeImageId**: (Optional) If you are using your own custom AMI (instead of the Amazon EKS-optimized AMI), enter a worker node AMI ID for your Region. If you specify a value here, it overrides any values in the **NodeImageIdSSMParam** field.

- **NodeVolumeSize**: Specify a root volume size for your worker nodes, in GiB.

- **KeyName**: Enter the name of an Amazon EC2 SSH key pair that you can use to connect using SSH into your worker nodes with after they launch. If you don't already have an Amazon EC2 keypair, you can create one in the AWS Management Console. For more information, see Amazon EC2 key pairs in the *Amazon EC2 User Guide for Linux Instances*.

    Note
    If you do not provide a keypair here, the AWS CloudFormation stack creation fails.

- **BootstrapArguments**: Specify any optional arguments to pass to the worker node bootstrap script, such as extra **kubelet** arguments. For more information, view the bootstrap script usage information at https://github.com/awslabs/amazon-eks-ami/blob/master/files/bootstrap.sh.

- **VpcId**: Enter the ID for the VPC that you created in Create your Amazon EKS cluster VPC (p. 11).

- **Subnets**: Choose the subnets that you created in Create your Amazon EKS cluster VPC (p. 11). If you created your VPC using the steps described at Creating a VPC for your Amazon EKS cluster (p. 152), then specify only the private subnets within the VPC for your worker nodes to launch into.

    Important
    If any of the subnets are public subnets, then they must have the automatic public IP address assignment setting enabled. If the setting is not enabled for the public subnet, then any worker nodes that you deploy to that public subnet will not be assigned a public IP address and will not be able to communicate with the cluster or other AWS services. If the subnet was deployed before 03/26/2020 using either of the Amazon EKS AWS CloudFormation VPC templates (p. 152), or by using `eksctl`, then automatic public IP address assignment is disabled for public subnets. For information about how to enable public IP address assignment for a subnet, see Modifying the Public IPv4 Addressing Attribute for Your Subnet. If the worker node is deployed to a private subnet, then it is able to communicate with the cluster and other AWS services through a NAT gateway.

6. Acknowledge that the stack might create IAM resources, and then choose **Create stack**.

7. When your stack has finished creating, select it in the console and choose **Outputs**.

8. Record the **NodeInstanceRole** for the node group that was created. You need this when you configure your Amazon EKS worker nodes.

To enable worker nodes to join your cluster

1. Download, edit, and apply the AWS IAM Authenticator configuration map.

a. Use the following command to download the configuration map:

```
curl -o aws-auth-cm.yaml https://amazon-eks.s3.us-west-2.amazonaws.com/
cloudformation/2020-05-08/aws-auth-cm.yaml
```

b. Open the file with your favorite text editor. Replace the *<ARN of instance role (not instance profile)>* snippet with the **NodeInstanceRole** value that you recorded in the previous procedure, and save the file.

> **Important**
> Do not modify any other lines in this file.

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: aws-auth
  namespace: kube-system
data:
  mapRoles: |
    - rolearn: <ARN of instance role (not instance profile)>
      username: system:node:{{EC2PrivateDNSName}}
      groups:
        - system:bootstrappers
        - system:nodes
```

c. Apply the configuration. This command may take a few minutes to finish.

```
kubectl apply -f aws-auth-cm.yaml
```

> **Note**
> If you receive the error `"aws-iam-authenticator": executable file not found in $PATH`, your **kubectl** isn't configured for Amazon EKS. For more information, see Installing `aws-iam-authenticator` (p. 187).
> If you receive any other authorization or resource type errors, see Unauthorized or access denied (`kubectl`) (p. 288) in the troubleshooting section.

2. Watch the status of your nodes and wait for them to reach the `Ready` status.

```
kubectl get nodes --watch
```

3. (GPU workers only) If you chose a GPU instance type and the Amazon EKS-optimized AMI with GPU support, you must apply the NVIDIA device plugin for Kubernetes as a DaemonSet on your cluster with the following command.

```
kubectl apply -f https://raw.githubusercontent.com/NVIDIA/k8s-device-plugin/1.0.0-
beta/nvidia-device-plugin.yml
```

4. (Optional) Launch a guest book application (p. 199) — Deploy a sample application to test your cluster and Linux worker nodes.

Launching Amazon EKS Windows worker nodes

This topic helps you to launch an Auto Scaling group of Windows worker nodes that register with your Amazon EKS cluster. After the nodes join the cluster, you can deploy Kubernetes applications to them.

> **Important**
> Amazon EKS worker nodes are standard Amazon EC2 instances, and you are billed for them based on normal Amazon EC2 prices. For more information, see Amazon EC2 pricing.

You must enable Windows support for your cluster and we recommend that you review important considerations before you launch a Windows worker node group. For more information, see Enabling Windows support (p. 55).

Choose the tab below that corresponds to your desired worker node creation method:

eksctl

If you don't already have an Amazon EKS cluster and a Linux worker node group to add a Windows worker node group to, then we recommend that you follow the Getting started with `eksctl` (p. 3) guide instead. The guide provides a complete end-to-end walkthrough for creating an Amazon EKS cluster with Linux and Windows worker nodes. If you have an existing Amazon EKS cluster and a Linux worker node group to add a Windows worker node group to, then complete the following steps to add the Windows worker node group.

To launch Windows worker nodes with `eksctl`

This procedure assumes that you have installed `eksctl`, and that your `eksctl` version is at least `0.19.0`. You can check your version with the following command:

```
eksctl version
```

For more information on installing or upgrading `eksctl`, see Installing or upgrading `eksctl` (p. 197).

> **Note**
> This procedure only works for clusters that were created with `eksctl`.

1. Create your worker node group with the following command. Replace the *example values* with your own values.

```
eksctl create nodegroup \
--region region-code \
--cluster windows \
--name windows-ng \
--node-type t2.large \
--nodes 3 \
--nodes-min 1 \
--nodes-max 4 \
--node-ami-family WindowsServer2019FullContainer
```

> **Note**
> For more information on the available options for **eksctl create nodegroup**, see the project README on GitHub or view the help page with the following command.
>
> ```
> eksctl create nodegroup --help
> ```

Output:

You'll see several lines of output as the worker nodes are created. The last line of output is similar to the following example line.

```
[#]  all nodegroups have up-to-date configuration
```

2. (Optional) Deploy a Windows sample application (p. 59) — Deploy a sample application to test your cluster and Windows worker nodes.

AWS Management Console

To launch your worker nodes with the AWS Management Console

These procedures have the following prerequisites:

- You have an existing Amazon EKS cluster and a Linux worker node group. If you don't have these resources, we recommend that you follow one of our Getting started with Amazon EKS (p. 3) guides to create them. The guides provide a complete end-to-end walkthrough for creating an Amazon EKS cluster with Linux worker nodes.
- You have created a VPC and security group that meet the requirements for an Amazon EKS cluster. For more information, see Cluster VPC considerations (p. 155) and Amazon EKS security group considerations (p. 157). The Getting started with Amazon EKS (p. 3) guide creates a VPC that meets the requirements, or you can also follow Creating a VPC for your Amazon EKS cluster (p. 152) to create one manually.

1. Wait for your cluster status to show as `ACTIVE`. If you launch your worker nodes before the cluster is active, the worker nodes will fail to register with the cluster and you will have to relaunch them.
2. Open the AWS CloudFormation console at https://console.aws.amazon.com/cloudformation
3. Choose **Create stack**.
4. For **Specify template**, select **Amazon S3 URL**, then copy the following URL, paste it into **Amazon S3 URL**, and select **Next** twice.

```
https://amazon-eks.s3.us-west-2.amazonaws.com/cloudformation/2020-05-08/amazon-eks-
windows-nodegroup.yaml
```

5. On the **Quick create stack** page, fill out the following parameters accordingly:

   - **Stack name**: Choose a stack name for your AWS CloudFormation stack. For example, you can call it *cluster-name*-**worker-nodes**.
   - **ClusterName**: Enter the name that you used when you created your Amazon EKS cluster.

     Important
     This name must exactly match the name you used in Step 1: Create your Amazon EKS cluster (p. 14); otherwise, your worker nodes cannot join the cluster.
   - **ClusterControlPlaneSecurityGroup**: Choose the **SecurityGroups** value from the AWS CloudFormation output that you generated with Create your Amazon EKS cluster VPC (p. 11).
   - **NodeGroupName**: Enter a name for your node group. This name can be used later to identify the Auto Scaling node group that is created for your worker nodes.
   - **NodeAutoScalingGroupMinSize**: Enter the minimum number of nodes that your worker node Auto Scaling group can scale in to.
   - **NodeAutoScalingGroupDesiredCapacity**: Enter the desired number of nodes to scale to when your stack is created.
   - **NodeAutoScalingGroupMaxSize**: Enter the maximum number of nodes that your worker node Auto Scaling group can scale out to.
   - **NodeInstanceType**: Choose an instance type for your worker nodes.

     Note
     The supported instance types for the latest version of the Amazon VPC CNI plugin for Kubernetes are shown here. You may need to update your CNI version to take advantage of the latest supported instance types. For more information, see Amazon VPC CNI plugin for Kubernetes upgrades (p. 174).

     Important
     Some instance types might not be available in all Regions.

- **NodeImageIdSSMParam**: Pre-populated with the Amazon EC2 Systems Manager parameter of the current recommended Amazon EKS-Optimized Windows Core AMI ID. If you want to use the full version of Windows, then replace *Core* with `Full`.

- **NodeImageId**: (Optional) If you are using your own custom AMI (instead of the Amazon EKS-optimized AMI), enter a worker node AMI ID for your Region. If you specify a value here, it overrides any values in the **NodeImageIdSSMParam** field.

- **NodeVolumeSize**: Specify a root volume size for your worker nodes, in GiB.

- **KeyName**: Enter the name of an Amazon EC2 SSH key pair that you can use to connect using SSH into your worker nodes with after they launch. If you don't already have an Amazon EC2 keypair, you can create one in the AWS Management Console. For more information, see Amazon EC2 key pairs in the *Amazon EC2 User Guide for Windows Instances*.

    > **Note**
    > If you do not provide a keypair here, the AWS CloudFormation stack creation fails.

- **BootstrapArguments**: Specify any optional arguments to pass to the worker node bootstrap script, such as extra `kubelet` arguments using `-KubeletExtraArgs`.

- **VpcId**: Select the ID for the VPC that you created in Create your Amazon EKS cluster VPC (p. 11).

- **NodeSecurityGroups**: Select the security group that was created for your Linux worker node group in Create your Amazon EKS cluster VPC (p. 11). If your Linux worker nodes have more than one security group attached to them (for example, if the Linux worker node group was created with `eksctl`), specify all of them here.

- **Subnets**: Choose the subnets that you created in Create your Amazon EKS cluster VPC (p. 11). If you created your VPC using the steps described at Creating a VPC for your Amazon EKS cluster (p. 152), then specify only the private subnets within the VPC for your worker nodes to launch into.

    > **Important**
    > If any of the subnets are public subnets, then they must have the automatic public IP address assignment setting enabled. If the setting is not enabled for the public subnet, then any worker nodes that you deploy to that public subnet will not be assigned a public IP address and will not be able to communicate with the cluster or other AWS services. If the subnet was deployed before 03/26/2020 using either of the Amazon EKS AWS CloudFormation VPC templates (p. 152), or by using `eksctl`, then automatic public IP address assignment is disabled for public subnets. For information about how to enable public IP address assignment for a subnet, see Modifying the Public IPv4 Addressing Attribute for Your Subnet. If the worker node is deployed to a private subnet, then it is able to communicate with the cluster and other AWS services through a NAT gateway.

6. Acknowledge that the stack might create IAM resources, and then choose **Create stack**.

7. When your stack has finished creating, select it in the console and choose **Outputs**.

8. Record the **NodeInstanceRole** for the node group that was created. You need this when you configure your Amazon EKS Windows worker nodes.

To enable worker nodes to join your cluster

1. Download, edit, and apply the AWS IAM Authenticator configuration map.

    a. Use the following command to download the configuration map:

    ```
    curl -o aws-auth-cm-windows.yaml https://amazon-eks.s3.us-west-2.amazonaws.com/
    cloudformation/2020-05-08/aws-auth-cm-windows.yaml
    ```

    b. Open the file with your favorite text editor. Replace the *<ARN of instance role (not instance profile) of **Linux** worker node>* and *<ARN of instance role*

(not instance profile) of **Windows** worker node> snippets with the
NodeInstanceRole values that you recorded for your Linux and Windows worker nodes, and
save the file.

> **Important**
> Do not modify any other lines in this file.

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: aws-auth
  namespace: kube-system
data:
  mapRoles: |
    - rolearn: <ARN of instance role (not instance profile) of **Linux** worker
 node>
      username: system:node:{{EC2PrivateDNSName}}
      groups:
        - system:bootstrappers
        - system:nodes
    - rolearn: <ARN of instance role (not instance profile) of **Windows**
 worker node>
      username: system:node:{{EC2PrivateDNSName}}
      groups:
        - system:bootstrappers
        - system:nodes
        - eks:kube-proxy-windows
```

c. Apply the configuration. This command may take a few minutes to finish.

```
kubectl apply -f aws-auth-cm-windows.yaml
```

> **Note**
> If you receive the error `"aws-iam-authenticator": executable file
> not found in $PATH`, your **kubectl** isn't configured for Amazon EKS. For more
> information, see Installing `aws-iam-authenticator` (p. 187).
> If you receive any other authorization or resource type errors, see Unauthorized or
> access denied (`kubectl`) (p. 288) in the troubleshooting section.

2. Watch the status of your nodes and wait for them to reach the `Ready` status.

```
kubectl get nodes --watch
```

3. (Optional) Deploy a Windows sample application (p. 59) — Deploy a sample application to test
    your cluster and Windows worker nodes.

Self-managed worker node updates

When a new Amazon EKS-optimized AMI is released, you should consider replacing the nodes in your
self-managed worker node group with the new AMI. Likewise, if you have updated the Kubernetes
version for your Amazon EKS cluster, you should also update the worker nodes to use worker nodes with
the same Kubernetes version.

Important
This topic covers worker node updates for self-managed node groups. If you are using Managed
node groups (p. 85), see Updating a managed node group (p. 89).

There are two basic ways to update self-managed node groups in your clusters to use a new AMI:

- the section called "Migrating to a new worker node group" (p. 102) – Create a new worker node group and migrate your pods to that group. Migrating to a new worker node group is more graceful than simply updating the AMI ID in an existing AWS CloudFormation stack, because the migration process taints the old node group as `NoSchedule` and drains the nodes after a new stack is ready to accept the existing pod workload.
- the section called "Updating an existing worker node group" (p. 107) – Update the AWS CloudFormation stack for an existing worker node group to use the new AMI. This method is not supported for worker node groups that were created with `eksctl`.

Migrating to a new worker node group

This topic helps you to create a new worker node group, gracefully migrate your existing applications to the new group, and then remove the old worker node group from your cluster.

eksctl

To migrate your applications to a new worker node group with `eksctl`

This procedure assumes that you have installed `eksctl`, and that your `eksctl` version is at least `0.19.0`. You can check your version with the following command:

```
eksctl version
```

For more information on installing or upgrading `eksctl`, see Installing or upgrading eksctl (p. 197).

> **Note**
> This procedure only works for clusters and worker node groups that were created with `eksctl`.

1. Retrieve the name of your existing worker node groups, substituting *default* with your cluster name.

```
eksctl get nodegroups --cluster=default
```

Output:

```
CLUSTER        NODEGROUP         CREATED                MIN SIZE      MAX SIZE
 DESIRED CAPACITY       INSTANCE TYPE     IMAGE ID
default        standard-workers  2019-05-01T22:26:58Z   1             4            3
                 t3.medium           ami-05a71d034119ffc12
```

2. Launch a new worker node group with `eksctl` with the following command, substituting the *example* values with your own values.

> **Note**
> For more available flags and their descriptions, see https://eksctl.io/.

```
eksctl create nodegroup \
--cluster default \
--version 1.16 \
--name standard-workers-new \
--node-type t3.medium \
--nodes 3 \
--nodes-min 1 \
--nodes-max 4 \
--node-ami auto
```

3. When the previous command completes, verify that all of your worker nodes have reached the `Ready` state with the following command:

```
kubectl get nodes
```

4. Delete the original node group with the following command, substituting the *example* values with your cluster and nodegroup names:

```
eksctl delete nodegroup --cluster default --name standard-workers
```

AWS Management Console

To migrate your applications to a new worker node group with the AWS Management Console

1. Launch a new worker node group by following the steps outlined in Launching Amazon EKS Linux worker nodes (p. 92).

2. When your stack has finished creating, select it in the console and choose **Outputs**.

3. Record the **NodeInstanceRole** for the node group that was created. You need this to add the new Amazon EKS worker nodes to your cluster.

    Note
    If you have attached any additional IAM policies to your old node group IAM role, such as adding permissions for the Kubernetes Cluster Autoscaler, you should attach those same policies to your new node group IAM role to maintain that functionality on the new group.

4. Update the security groups for both worker node groups so that they can communicate with each other. For more information, see Amazon EKS security group considerations (p. 157).

    a. Record the security group IDs for both worker node groups. This is shown as the **NodeSecurityGroup** value in the AWS CloudFormation stack outputs.

    You can use the following AWS CLI commands to get the security group IDs from the stack names. In these commands, `oldNodes` is the AWS CloudFormation stack name for your older worker node stack, and `newNodes` is the name of the stack that you are migrating to.

```
oldNodes="<old_node_CFN_stack_name>"
newNodes="<new_node_CFN_stack_name>"

oldSecGroup=$(aws cloudformation describe-stack-resources --stack-name
 $oldNodes \
--query 'StackResources[?
ResourceType==`AWS::EC2::SecurityGroup`].PhysicalResourceId' \
--output text)
newSecGroup=$(aws cloudformation describe-stack-resources --stack-name
 $newNodes \
--query 'StackResources[?
ResourceType==`AWS::EC2::SecurityGroup`].PhysicalResourceId' \
--output text)
```

    b. Add ingress rules to each worker node security group so that they accept traffic from each other.

    The following AWS CLI commands add ingress rules to each security group that allow all traffic on all protocols from the other security group. This configuration allows pods in each worker node group to communicate with each other while you are migrating your workload to the new group.

```
aws ec2 authorize-security-group-ingress --group-id $oldSecGroup \
--source-group $newSecGroup --protocol -1
aws ec2 authorize-security-group-ingress --group-id $newSecGroup \
--source-group $oldSecGroup --protocol -1
```

5. Edit the `aws-auth` configmap to map the new worker node instance role in RBAC.

```
kubectl edit configmap -n kube-system aws-auth
```

Add a new `mapRoles` entry for the new worker node group.

```
apiVersion: v1
data:
  mapRoles: |
    - rolearn: <ARN of instance role (not instance profile)>
      username: system:node:{{EC2PrivateDNSName}}
      groups:
        - system:bootstrappers
        - system:nodes
    - rolearn: arn:aws:iam::111122223333:role/workers-1-10-NodeInstanceRole-
U11V27W93CX5
      username: system:node:{{EC2PrivateDNSName}}
      groups:
        - system:bootstrappers
        - system:nodes
```

Replace the *<ARN of instance role (not instance profile)>* snippet with the **NodeInstanceRole** value that you recorded in Step 3 (p. 103), then save and close the file to apply the updated configmap.

6. Watch the status of your nodes and wait for your new worker nodes to join your cluster and reach the `Ready` status.

```
kubectl get nodes --watch
```

7. (Optional) If you are using the Kubernetes Cluster Autoscaler, scale the deployment down to 0 replicas to avoid conflicting scaling actions.

```
kubectl scale deployments/cluster-autoscaler --replicas=0 -n kube-system
```

8. Use the following command to taint each of the nodes that you want to remove with `NoSchedule` so that new pods are not scheduled or rescheduled on the nodes you are replacing:

```
kubectl taint nodes node_name key=value:NoSchedule
```

If you are upgrading your worker nodes to a new Kubernetes version, you can identify and taint all of the nodes of a particular Kubernetes version (in this case, 1.13.12) with the following code snippet.

```
K8S_VERSION=1.13.12
nodes=$(kubectl get nodes -o jsonpath="{.items[?(@.status.nodeInfo.kubeletVersion==
\"v$K8S_VERSION\")].metadata.name}")
for node in ${nodes[@]}
do
    echo "Tainting $node"
    kubectl taint nodes $node key=value:NoSchedule
```

```
done
```

9. Determine your cluster's DNS provider.

```
kubectl get deployments -l k8s-app=kube-dns -n kube-system
```

Output (this cluster is using `kube-dns` for DNS resolution, but your cluster may return `coredns` instead):

```
NAME        DESIRED    CURRENT    UP-TO-DATE    AVAILABLE    AGE
kube-dns    1          1          1             1            31m
```

10. If your current deployment is running fewer than two replicas, scale out the deployment to two replicas. Substitute `coredns` for `kube-dns` if your previous command output returned that instead.

```
kubectl scale deployments/kube-dns --replicas=2 -n kube-system
```

11. Drain each of the nodes that you want to remove from your cluster with the following command:

```
kubectl drain node_name --ignore-daemonsets --delete-local-data
```

If you are upgrading your worker nodes to a new Kubernetes version, you can identify and drain all of the nodes of a particular Kubernetes version (in this case, 1.13.12) with the following code snippet.

```
K8S_VERSION=1.13.12
nodes=$(kubectl get nodes -o jsonpath="{.items[?(@.status.nodeInfo.kubeletVersion==
\"v$K8S_VERSION\")].metadata.name}")
for node in ${nodes[@]}
do
    echo "Draining $node"
    kubectl drain $node --ignore-daemonsets --delete-local-data
done
```

12. After your old worker nodes have finished draining, revoke the security group ingress rules you authorized earlier, and then delete the AWS CloudFormation stack to terminate the instances.

   Note
   If you have attached any additional IAM policies to your old node group IAM role, such as adding permissions for the Kubernetes Cluster Autoscaler), you must detach those additional policies from the role before you can delete your AWS CloudFormation stack.

   a. Revoke the ingress rules that you created for your worker node security groups earlier. In these commands, `oldNodes` is the AWS CloudFormation stack name for your older worker node stack, and `newNodes` is the name of the stack that you are migrating to.

   ```
   oldNodes="<old_node_CFN_stack_name>"
   newNodes="<new_node_CFN_stack_name>"

   oldSecGroup=$(aws cloudformation describe-stack-resources --stack-name
    $oldNodes \
   --query 'StackResources[?
   ResourceType==`AWS::EC2::SecurityGroup`].PhysicalResourceId' \
   --output text)
   newSecGroup=$(aws cloudformation describe-stack-resources --stack-name
    $newNodes \
   ```

```
--query 'StackResources[?
ResourceType==`AWS::EC2::SecurityGroup`].PhysicalResourceId' \
--output text)
aws ec2 revoke-security-group-ingress --group-id $oldSecGroup \
--source-group $newSecGroup --protocol -1
aws ec2 revoke-security-group-ingress --group-id $newSecGroup \
--source-group $oldSecGroup --protocol -1
```

    b.   Open the AWS CloudFormation console at https://console.aws.amazon.com/
cloudformation.

    c.   Select your old worker node stack.

    d.   Choose **Actions**, then **Delete stack**.

13. Edit the `aws-auth` configmap to remove the old worker node instance role from RBAC.

```
kubectl edit configmap -n kube-system aws-auth
```

Delete the `mapRoles` entry for the old worker node group.

```
apiVersion: v1
data:
  mapRoles: |
    - rolearn: arn:aws:iam::111122223333:role/workers-1-11-NodeInstanceRole-
W70725MZQFF8
      username: system:node:{{EC2PrivateDNSName}}
      groups:
        - system:bootstrappers
        - system:nodes
    - rolearn: arn:aws:iam::111122223333:role/workers-1-10-NodeInstanceRole-
U11V27W93CX5
      username: system:node:{{EC2PrivateDNSName}}
      groups:
        - system:bootstrappers
        - system:nodes
```

Save and close the file to apply the updated configmap.

14. (Optional) If you are using the Kubernetes Cluster Autoscaler, scale the deployment back to one
replica.

> **Note**
> You must also tag your new Auto Scaling group appropriately (for example, `k8s.io/`
> `cluster-autoscaler/enabled,k8s.io/cluster-autoscaler/<YOUR CLUSTER`
> `NAME>`) and update your Cluster Autoscaler deployment's command to point to the
> newly tagged Auto Scaling group. For more information, see Cluster Autoscaler on
> AWS.

```
kubectl scale deployments/cluster-autoscaler --replicas=1 -n kube-system
```

15. (Optional) Verify that you are using the latest version of the Amazon VPC CNI plugin for
Kubernetes. You may need to update your CNI version to take advantage of the latest
supported instance types. For more information, see Amazon VPC CNI plugin for Kubernetes
upgrades (p. 174).

16. If your cluster is using `kube-dns` for DNS resolution (see step Step 9 (p. 105)), scale in the
`kube-dns` deployment to one replica.

```
kubectl scale deployments/kube-dns --replicas=1 -n kube-system
```

Updating an existing worker node group

This topic helps you to update an existing AWS CloudFormation self-managed worker node stack with a new AMI. You can use this procedure to update your worker nodes to a new version of Kubernetes following a cluster update, or you can update to the latest Amazon EKS-optimized AMI for an existing Kubernetes version.

> **Important**
> This topic covers worker node updates for self-managed node groups. If you are using Managed node groups (p. 85), see Updating a managed node group (p. 89).

The latest default Amazon EKS worker node AWS CloudFormation template is configured to launch an instance with the new AMI into your cluster before removing an old one, one at a time. This configuration ensures that you always have your Auto Scaling group's desired count of active instances in your cluster during the rolling update.

> **Note**
> This method is not supported for worker node groups that were created with `eksctl`. If you created your cluster or worker node group with `eksctl`, see Migrating to a new worker node group (p. 102).

To update an existing worker node group

1. Determine your cluster's DNS provider.

    ```
    kubectl get deployments -l k8s-app=kube-dns -n kube-system
    ```

    Output (this cluster is using `kube-dns` for DNS resolution, but your cluster may return `coredns` instead):

    ```
    NAME        DESIRED    CURRENT    UP-TO-DATE    AVAILABLE    AGE
    kube-dns    1          1          1             1            31m
    ```

2. If your current deployment is running fewer than two replicas, scale out the deployment to two replicas. Substitute `coredns` for `kube-dns` if your previous command output returned that instead.

    ```
    kubectl scale deployments/kube-dns --replicas=2 -n kube-system
    ```

3. (Optional) If you are using the Kubernetes Cluster Autoscaler, scale the deployment down to zero replicas to avoid conflicting scaling actions.

    ```
    kubectl scale deployments/cluster-autoscaler --replicas=0 -n kube-system
    ```

4. Determine the instance type and desired instance count of your current worker node group. You will enter these values later when you update the AWS CloudFormation template for the group.

    a. Open the Amazon EC2 console at https://console.aws.amazon.com/ec2/.

    b. Choose **Launch Configurations** in the left navigation, and note the instance type for your existing worker node launch configuration.

    c. Choose **Auto Scaling Groups** in the left navigation and note the **Desired** instance count for your existing worker node Auto Scaling group.

5. Open the AWS CloudFormation console at https://console.aws.amazon.com/cloudformation.

6. Select your worker node group stack, and then choose **Update**.

7. Select **Replace current template** and select **Amazon S3 URL**.

8. For **Amazon S3 URL**, paste the following URL into the text area to ensure that you are using the latest version of the worker node AWS CloudFormation template, and then choose **Next**:

```
https://amazon-eks.s3.us-west-2.amazonaws.com/cloudformation/2020-05-08/amazon-eks-
nodegroup.yaml
```

9. On the **Specify stack details** page, fill out the following parameters, and choose **Next**:

   - **NodeAutoScalingGroupDesiredCapacity** – Enter the desired instance count that you recorded in Step 4 (p. 107), or enter a new desired number of nodes to scale to when your stack is updated.

   - **NodeAutoScalingGroupMaxSize** – Enter the maximum number of nodes to which your worker node Auto Scaling group can scale out. **This value must be at least one node greater than your desired capacity so that you can perform a rolling update of your worker nodes without reducing your node count during the update.**

   - **NodeInstanceType** – Choose the instance type your recorded in Step 4 (p. 107), or choose a different instance type for your worker nodes.

     Note
     The supported instance types for the latest version of the Amazon VPC CNI plugin for Kubernetes are shown here. You may need to update your CNI version to take advantage of the latest supported instance types. For more information, see Amazon VPC CNI plugin for Kubernetes upgrades (p. 174).

     Important
     Some instance types might not be available in all Regions.

   - **NodeImageIdSSMParam** – The Amazon EC2 Systems Manager parameter of the AMI ID that you want to update to. The following value uses the latest Amazon EKS-optimized AMI for Kubernetes version 1.16.

   ```
   /aws/service/eks/optimized-ami/1.16/amazon-linux-2/recommended/image_id
   ```

   You can change the *1.16* value to any supported Kubernetes version (p. 49). If you want to use the Amazon EKS-optimized AMI with GPU support, then change *amazon-linux-2* to *amazon-linux-2-gpu*.

     Note
     Using the Amazon EC2 Systems Manager parameter enables you to update your worker nodes in the future without having to lookup and specify an AMI ID. If your AWS CloudFormation stack is using this value, any stack update will always launch the latest recommended Amazon EKS-optimized AMI for your specified Kubernetes version, even if you don't change any values in the template.

   - **NodeImageId** – To use your own custom AMI, enter the ID for the AMI to use.

     Important
     This value overrides any value specified for **NodeImageIdSSMParam**. If you want to use the **NodeImageIdSSMParam** value, ensure that the value for **NodeImageId** is blank.

10. (Optional) On the **Options** page, tag your stack resources. Choose **Next**.

11. On the **Review** page, review your information, acknowledge that the stack might create IAM resources, and then choose **Update stack**.

     Note
     The update of each node in the cluster takes several minutes. Wait for the update of all nodes to complete before performing the next steps.

12. If your cluster's DNS provider is `kube-dns`, scale in the `kube-dns` deployment to one replica.

    ```
    kubectl scale deployments/kube-dns --replicas=1 -n kube-system
    ```

13. (Optional) If you are using the Kubernetes Cluster Autoscaler, scale the deployment back to one replica.

```
kubectl scale deployments/cluster-autoscaler --replicas=1 -n kube-system
```

14. (Optional) Verify that you are using the latest version of the Amazon VPC CNI plugin for Kubernetes. You may need to update your CNI version to take advantage of the latest supported instance types. For more information, see Amazon VPC CNI plugin for Kubernetes upgrades (p. 174).

Ubuntu Amazon EKS-optimized AMIs

In addition to the official Amazon EKS-optimized, Canonical has partnered with Amazon EKS to create worker node AMIs that you can use in your clusters.

Canonical delivers a built-for-purpose Kubernetes Node OS image. This minimized Ubuntu image is optimized for Amazon EKS and includes the custom AWS kernel that is jointly developed with AWS. For more information, see Ubuntu and Amazon Elastic Kubernetes Service and Optimized support for Amazon EKS on Ubuntu 18.04.

AWS Fargate

This chapter discusses using Amazon EKS to run Kubernetes pods on AWS Fargate.

AWS Fargate is a technology that provides on-demand, right-sized compute capacity for containers. With AWS Fargate, you no longer have to provision, configure, or scale groups of virtual machines to run containers. This removes the need to choose server types, decide when to scale your node groups, or optimize cluster packing.

You can control which pods start on Fargate and how they run with Fargate profiles (p. 115), which are defined as part of your Amazon EKS cluster.

Amazon EKS integrates Kubernetes with AWS Fargate by using controllers that are built by AWS using the upstream, extensible model provided by Kubernetes. These controllers run as part of the Amazon EKS managed Kubernetes control plane and are responsible for scheduling native Kubernetes pods onto Fargate. The Fargate controllers include a new scheduler that runs alongside the default Kubernetes scheduler in addition to several mutating and validating admission controllers. When you start a pod that meets the criteria for running on Fargate, the Fargate controllers running in the cluster recognize, update, and schedule the pod onto Fargate.

Each pod running on Fargate has its own isolation boundary and does not share the underlying kernel, CPU resources, memory resources, or elastic network interface with another pod.

This chapter describes the different components of pods running on Fargate, and calls out special considerations for using Fargate with Amazon EKS.

AWS Fargate with Amazon EKS is currently only available in the following Regions:

Region name	Region
US East (Ohio)	us-east-2
US East (N. Virginia)	us-east-1
US West (Oregon)	us-west-2
Asia Pacific (Singapore)	ap-southeast-1
Asia Pacific (Sydney)	ap-southeast-2
Asia Pacific (Tokyo)	ap-northeast-1
Europe (Frankfurt)	eu-central-1
Europe (Ireland)	eu-west-1

Topics

AWS Fargate considerations

Here's some things to consider about using Fargate on Amazon EKS.

- Classic Load Balancers and Network Load Balancers are not supported on pods running on Fargate. For ingress, we recommend that you use the ALB Ingress Controller on Amazon EKS (p. 147) (minimum version v1.1.4).
- Pods must match a Fargate profile at the time that they are scheduled in order to run on Fargate. Pods which do not match a Fargate profile may be stuck as `Pending`. If a matching Fargate profile exists, you can delete pending pods that you have created to reschedule them onto Fargate.
- Daemonsets are not supported on Fargate. If your application requires a daemon, you should reconfigure that daemon to run as a sidecar container in your pods.
- Privileged containers are not supported on Fargate.
- Pods running on Fargate cannot specify HostPort or HostNetwork in the pod manifest.
- GPUs are currently not available on Fargate.
- Pods running on Fargate are only supported on private subnets (with NAT gateway access to AWS services, but not a direct route to an Internet Gateway), so your cluster's VPC must have private subnets available.
- We recommend using the Vertical Pod Autoscaler (p. 141) with pods running on Fargate to optimize the CPU and memory used for your applications. However, because changing the resource allocation for a pod requires the pod to be restarted, you must set the pod update policy to either `Auto` or `Recreate` to ensure correct functionality.
- Stateful applications are not recommended for pods running on Fargate. Instead, we recommend that you use AWS solutions such as Amazon S3 or DynamoDB for pod data storage.
- Fargate runs each pod in a VM-isolated environment without sharing resources with other pods. However, because Kubernetes is a single-tenant orchestrator, Fargate cannot guarantee pod-level security isolation. You should run sensitive workloads or untrusted workloads that need complete security isolation using separate Amazon EKS clusters.
- DNS resolution and DNS hostnames must be enabled for your VPC. For more information, see Viewing and updating DNS support for your VPC.
- You cannot use ??? (p. 168) with Fargate.

Getting started with AWS Fargate on Amazon EKS

This topic helps you to get started running pods on AWS Fargate with your Amazon EKS cluster.

Note
AWS Fargate with Amazon EKS is currently only available in the following Regions:

Region name	Region
US East (Ohio)	us-east-2
US East (N. Virginia)	us-east-1
US West (Oregon)	us-west-2
Asia Pacific (Singapore)	ap-southeast-1
Asia Pacific (Sydney)	ap-southeast-2
Asia Pacific (Tokyo)	ap-northeast-1

Region name	Region
Europe (Frankfurt)	eu-central-1
Europe (Ireland)	eu-west-1

If you restrict access to your cluster's public endpoint using CIDR blocks, it is recommended that you also enable private endpoint access so that Fargate pods can communicate with the cluster. Without the private endpoint enabled, the CIDR blocks that you specify for public access must include the egress sources from your VPC. For more information, see Amazon EKS cluster endpoint access control (p. 35).

(Optional) Create a cluster

Pods running on Fargate are supported on Amazon EKS clusters beginning with Kubernetes version 1.14 and platform version (p. 49) `eks.5`. Existing clusters can update to version 1.14 or later to take advantage of this feature. For more information, see Updating an Amazon EKS cluster Kubernetes version (p. 27).

If you do not already have an Amazon EKS cluster that supports Fargate, you can create one with the following `eksctl` command.

> **Note**
> This procedure assumes that you have installed `eksctl`, and that your `eksctl` version is at least `0.19.0`. You can check your version with the following command:
>
> ```
> eksctl version
> ```
>
> For more information on installing or upgrading `eksctl`, see Installing or upgrading `eksctl` (p. 197).

```
eksctl create cluster --name my-cluster --version 1.16 --fargate
```

Adding the `--fargate` option in the command above creates a cluster without a node group. However, `eksctl` creates a pod execution role, a Fargate profile for the `default` and `kube-system` namespaces, and it patches the `coredns` deployment so that it can run on Fargate.

Ensure that existing nodes can communicate with Fargate pods

If you are working with a new cluster with no worker nodes, or a cluster with only managed node groups (p. 85), you can skip to Create a Fargate pod execution role (p. 113).

If you are working with an existing cluster that already has worker nodes associated with it, you need to make sure that pods on these nodes can communicate freely with pods running on Fargate. Pods running on Fargate are automatically configured to use the cluster security group for the cluster that they are associated with. You must ensure that any existing worker nodes in your cluster can send and receive traffic to and from the cluster security group. Managed node groups (p. 85) are automatically configured to use the cluster security group as well, so you do not need to modify or check them for this compatibility.

For existing node groups that were created with `eksctl` or the Amazon EKS-managed AWS CloudFormation templates, you can add the cluster security group to the nodes manually, or you can modify the node group's Auto Scaling group launch template to attach the cluster security group to the instances. For more information, see Changing an instance's security groups in the *Amazon VPC User Guide*.

You can check for a cluster security group for your cluster in the AWS Management Console under the cluster's **Networking** section, or with the following AWS CLI command:

```
aws eks describe-cluster --name cluster_name --query
 cluster.resourcesVpcConfig.clusterSecurityGroupId
```

Create a Fargate pod execution role

When your cluster creates pods on AWS Fargate, the pods need to make calls to AWS APIs on your behalf to do things like pull container images from Amazon ECR. The Amazon EKS pod execution role provides the IAM permissions to do this.

> **Note**
> If you created your cluster with `eksctl` using the `--fargate` option, then your cluster already has a pod execution role and you can skip ahead to Create a Fargate profile for your cluster (p. 113). Similarly, if you use `eksctl` to create your Fargate profiles, `eksctl` will create your pod execution role if one does not already exist.

When you create a Fargate profile, you must specify a pod execution role to use with your pods. This role is added to the cluster's Kubernetes Role based access control (RBAC) for authorization. This allows the `kubelet` that is running on the Fargate infrastructure to register with your Amazon EKS cluster so that it can appear in your cluster as a node. For more information, see Pod execution role (p. 252).

1. Open the IAM console at https://console.aws.amazon.com/iam/.
2. Choose **Roles**, then **Create role**.
3. Choose **EKS** from the list of services, **EKS - Fargate pod** for your use case, and then **Next: Permissions**.
4. Choose **Next: Tags**.
5. (Optional) Add metadata to the role by attaching tags as key–value pairs. For more information about using tags in IAM, see Tagging IAM Entities in the *IAM User Guide*.
6. Choose **Next: Review**.
7. For **Role name**, enter a unique name for your role, such as `AmazonEKSFargatePodExecutionRole`, then choose **Create role**.

Create a Fargate profile for your cluster

Before you can schedule pods running on Fargate in your cluster, you must define a Fargate profile that specifies which pods should use Fargate when they are launched. For more information, see AWS Fargate profile (p. 115).

> **Note**
> If you created your cluster with `eksctl` using the `--fargate` option, then a Fargate profile has already been created for your cluster with selectors for all pods in the `kube-system` and `default` namespaces. Use the following procedure to create Fargate profiles for any other namespaces you would like to use with Fargate.

Choose the tab below that corresponds to your preferred Fargate profile creation method.

eksctl

To create a Fargate profile for a cluster with `eksctl`

This procedure assumes that you have installed `eksctl`, and that your `eksctl` version is at least `0.19.0`. You can check your version with the following command:

```
eksctl version
```

For more information on installing or upgrading `eksctl`, see Installing or upgrading `eksctl` (p. 197).

- Create your Fargate profile with the following `eksctl` command, replacing the *variable text* with your own values. You must specify a namespace, but the labels option is not required.

```
eksctl create fargateprofile --cluster cluster_name --name fargate_profile_name --
namespace kubernetes_namespace --labels key=value
```

AWS Management Console

To create a Fargate profile for a cluster with the AWS Management Console

1. Open the Amazon EKS console at https://console.aws.amazon.com/eks/home#/clusters.
2. Choose the cluster to create a Fargate profile for.
3. Under **Fargate profiles**, choose **Add Fargate profile**.
4. On the **Configure Fargate profile** page, enter the following information and choose **Next**.

   a. For **Name**, enter a unique name for your Fargate profile.

   b. For **Pod execution role**, choose the pod execution role to use with your Fargate profile. Only IAM roles with the `eks-fargate-pods.amazonaws.com` service principal are shown. If you do not see any roles listed here, you must create one. For more information, see Pod execution role (p. 252).

   c. For **Subnets**, choose the subnets to use for your pods. By default, all subnets in your cluster's VPC are selected. Only private subnets are supported for pods running on Fargate; you must deselect any public subnets.

   d. For **Tags**, you can optionally tag your Fargate profile. These tags do not propagate to other resources associated with the profile, such as its pods.

5. On the **Configure pods selection** page, enter the following information and choose **Next**.

   a. For **Namespace**, enter a namespace to match for pods, such as `kube-system` or `default`.

   b. (Optional) Add Kubernetes labels to the selector that pods in the specified namespace must have to match the selector. For example, you could add the label `infrastructure: fargate` to the selector so that only pods in the specified namespace that also have the `infrastructure: fargate` Kubernetes label match the selector.

6. On the **Review and create** page, review the information for your Fargate profile and choose **Create**.

(Optional) Update CoreDNS

By default, CoreDNS is configured to run on Amazon EC2 infrastructure on Amazon EKS clusters. If you want to *only* run your pods on Fargate in your cluster, you need to modify the CoreDNS deployment to remove the `eks.amazonaws.com/compute-type : ec2` annotation. You would also need to create a Fargate profile to target the CoreDNS pods. The following Fargate profile JSON file does this.

> **Note**
> If you created your cluster with `eksctl` using the `--fargate` option, then `coredns` has already been patched to run on Fargate and you can skip ahead to Next steps (p. 115).

```
{
    "fargateProfileName": "coredns",
```

```
    "clusterName": "dev",
    "podExecutionRoleArn": "arn:aws:iam::111122223333:role/
AmazonEKSFargatePodExecutionRole",
    "subnets": [
        "subnet-0b64dd020cdff3864",
        "subnet-00b03756df55e2b87",
        "subnet-0418fcb68ed294abf"
    ],
    "selectors": [
        {
            "namespace": "kube-system",
            "labels": {
                "k8s-app": "kube-dns"
            }
        }
    ]
}
```

You could apply this Fargate profile to your cluster with the following AWS CLI command. First, create a file called `coredns.json` and paste the JSON file from the previous step into it, replacing the *variable text* with your own cluster values.

```
aws eks create-fargate-profile --cli-input-json file://coredns.json
```

Then, use the following `kubectl` command to remove the `eks.amazonaws.com/compute-type : ec2` annotation from the CoreDNS pods.

```
kubectl patch deployment coredns -n kube-system --type json \
-p='[{"op": "remove", "path": "/spec/template/metadata/annotations/
eks.amazonaws.com~1compute-type"}]'
```

Next steps

- You can start migrating your existing applications to run on Fargate with the following workflow.
  1. Create a Fargate profile (p. 117) that matches your application's Kubernetes namespace and Kubernetes labels.
  2. Delete and re-create any existing pods so that they are scheduled on Fargate. For example, the following command triggers a rollout of the `coredns` Deployment. You can modify the namespace and deployment type to update your specific pods.

     ```
     kubectl rollout restart -n kube-system deployment coredns
     ```

- Deploy the ALB Ingress Controller on Amazon EKS (p. 147) (version v1.1.4 or later) to allow Ingress objects for your pods running on Fargate.
- Deploy the Vertical Pod Autoscaler (p. 141) for your pods running on Fargate to optimize the CPU and memory used for your applications. Be sure to set the pod update policy to either `Auto` or `Recreate` to ensure correct functionality.

AWS Fargate profile

Before you can schedule pods on Fargate in your cluster, you must define at least one Fargate profile that specifies which pods should use Fargate when they are launched.

The Fargate profile allows an administrator to declare which pods run on Fargate. This declaration is done through the profile's selectors. Each profile can have up to five selectors that contain a namespace and optional labels. You must define a namespace for every selector. The label field consists of multiple

optional key-value pairs. Pods that match a selector (by matching a namespace for the selector and all of the labels specified in the selector) are scheduled on Fargate. If a namespace selector is defined without any labels, Amazon EKS will attempt to schedule all pods that run in that namespace onto Fargate using the profile. If a to-be-scheduled pod matches any of the selectors in the Fargate profile, then that pod is scheduled on Fargate.

If a pod matches multiple Fargate profiles, Amazon EKS picks one of the matches at random. In this case, you can specify which profile a pod should use by adding the following Kubernetes label to the pod specification: `eks.amazonaws.com/fargate-profile:` *`profile_name`*. However, the pod must still match a selector in that profile in order to be scheduled onto Fargate.

When you create a Fargate profile, you must specify a pod execution role for the pods that run on Fargate using the profile. This role is added to the cluster's Kubernetes Role Based Access Control (RBAC) for authorization so that the `kubelet` that is running on the Fargate infrastructure can register with your Amazon EKS cluster and appear in your cluster as a node. The pod execution role also provides IAM permissions to the Fargate infrastructure to allow read access to Amazon ECR image repositories. For more information, see Pod execution role (p. 252).

Fargate profiles are immutable. However, you can create a new updated profile to replace an existing profile and then delete the original after the updated profile has finished creating.

> **Note**
> Any pods that are running using a Fargate profile will be stopped and put into pending when the profile is deleted.

If any Fargate profiles in a cluster are in the `DELETING` status, you must wait for that Fargate profile to finish deleting before you can create any other profiles in that cluster.

Fargate profile components

The following components are contained in a Fargate profile.

```
{
    "fargateProfileName": "",
    "clusterName": "",
    "podExecutionRoleArn": "",
    "subnets": [
        ""
    ],
    "selectors": [
        {
            "namespace": "",
            "labels": {
                "KeyName": ""
            }
        }
    ],
    "clientRequestToken": "",
    "tags": {
        "KeyName": ""
    }
}
```

Pod execution role

When your cluster creates pods on AWS Fargate, the pod needs to make calls to AWS APIs on your behalf, for example, to pull container images from Amazon ECR. The Amazon EKS pod execution role provides the IAM permissions to do this.

When you create a Fargate profile, you must specify a pod execution role to use with your pods. This role is added to the cluster's Kubernetes Role Based Access Control (RBAC) for authorization, so that

the `kubelet` that is running on the Fargate infrastructure can register with your Amazon EKS cluster and appear in your cluster as a node. For more information, see Pod execution role (p. 252).

Subnets

The IDs of subnets to launch pods into that use this profile. At this time, pods running on Fargate are not assigned public IP addresses, so only private subnets (with no direct route to an Internet Gateway) are accepted for this parameter.

Selectors

The selectors to match for pods to use this Fargate profile. Each selector must have an associated namespace. Optionally, you can also specify labels for a namespace. You may specify up to five selectors in a Fargate profile. A pod only needs to match one selector to run using the Fargate profile.

Namespace

You must specify a namespace for a selector. The selector only matches pods that are created in this namespace, but you can create multiple selectors to target multiple namespaces.

Labels

You can optionally specify Kubernetes labels to match for the selector. The selector only matches pods that have all of the labels that are specified in the selector.

Creating a Fargate profile

This topic helps you to create a Fargate profile. Your cluster must support Fargate (beginning with Kubernetes version 1.14 and platform version (p. 49) `eks.5`). You also must have created a pod execution role to use for your Fargate profile. For more information, see Pod execution role (p. 252). Pods running on Fargate are only supported on private subnets (with NAT gateway access to AWS services, but not a direct route to an Internet Gateway), so your cluster's VPC must have private subnets available.

eksctl

To create a Fargate profile for a cluster with `eksctl`

This procedure assumes that you have installed `eksctl`, and that your `eksctl` version is at least `0.19.0`. You can check your version with the following command:

```
eksctl version
```

For more information on installing or upgrading `eksctl`, see Installing or upgrading eksctl (p. 197).

- Create your Fargate profile with the following `eksctl` command, replacing the *variable text* with your own values. You must specify a namespace, but the labels option is not required.

```
eksctl create fargateprofile --cluster cluster_name --name fargate_profile_name --
namespace kubernetes_namespace --labels key=value
```

AWS Management Console

To create a Fargate profile for a cluster with the AWS Management Console

1. Open the Amazon EKS console at https://console.aws.amazon.com/eks/home#/clusters.

2. Choose the cluster to create a Fargate profile for.

3. Under **Fargate profiles**, choose **Add Fargate profile**.

4. On the **Configure Fargate profile** page, enter the following information and choose **Next**.

   a. For **Name**, enter a unique name for your Fargate profile.

   b. For **Pod execution role**, choose the pod execution role to use with your Fargate profile. Only IAM roles with the `eks-fargate-pods.amazonaws.com` service principal are shown. If you do not see any roles listed here, you must create one. For more information, see Pod execution role (p. 252).

   c. For **Subnets**, choose the subnets to use for your pods. By default, all subnets in your cluster's VPC are selected. Only private subnets are supported for pods running on Fargate; you must deselect any public subnets.

   d. For **Tags**, you can optionally tag your Fargate profile. These tags do not propagate to other resources associated with the profile, such as its pods.

5. On the **Configure pods selection** page, enter the following information and choose **Next**.

   a. For **Namespace**, enter a namespace to match for pods, such as `kube-system` or `default`.

   b. (Optional) Add Kubernetes labels to the selector that pods in the specified namespace must have to match the selector. For example, you could add the label `infrastructure: fargate` to the selector so that only pods in the specified namespace that also have the `infrastructure: fargate` Kubernetes label match the selector.

6. On the **Review and create** page, review the information for your Fargate profile and choose **Create**.

Deleting a Fargate profile

This topic helps you to delete a Fargate profile.

When you delete a Fargate profile, any pods that were scheduled onto Fargate with the profile are deleted. If those pods match another Fargate profile, then they are scheduled on Fargate with that profile. If they no longer match any Fargate profiles, then they are not scheduled onto Fargate and may remain as pending.

Only one Fargate profile in a cluster can be in the `DELETING` status at a time. You must wait for a Fargate profile to finish deleting before you can delete any other profiles in that cluster.

To delete a Fargate profile from a cluster

1. Open the Amazon EKS console at https://console.aws.amazon.com/eks/home#/clusters.

2. Choose the cluster that you want to delete the Fargate profile from.

3. Choose the Fargate profile to delete and then **Delete**.

4. On the **Delete** *cluster_name* page, type the name of the cluster and choose **Confirm** to delete.

Fargate pod configuration

This section describes some of the unique pod configuration details for running Kubernetes pods on AWS Fargate.

Pod CPU and memory

Kubernetes allows you to define requests, a minimum amount of vCPU and memory resources that are allocated to each container in a pod. Pods are scheduled by Kubernetes to ensure that at least the

requested resources for each pod are available on the compute resource. For more information, see Managing compute resources for containers in the Kubernetes documentation.

When pods are scheduled on Fargate, the vCPU and memory reservations within the pod specification determine how much CPU and memory to provision for the pod.

- The maximum request out of any Init containers is used to determine the Init request vCPU and memory requirements.
- Requests for all long-running containers are added up to determine the long-running request vCPU and memory requirements.
- The larger of the above two values is chosen for the vCPU and memory request to use for your pod.
- Fargate adds 256 MB to each pod's memory reservation for the required Kubernetes components (`kubelet`, `kube-proxy`, and `containerd`).

Fargate rounds up to the compute configuration shown below that most closely matches the sum of vCPU and memory requests in order to ensure pods always have the resources that they need to run.

If you do not specify a vCPU and memory combination, then the smallest available combination is used (.25 vCPU and 0.5 GB memory).

The table below shows the vCPU and memory combinations that are available for pods running on Fargate.

vCPU value	Memory value
.25 vCPU	0.5 GB, 1 GB, 2 GB
.5 vCPU	1 GB, 2 GB, 3 GB, 4 GB
1 vCPU	2 GB, 3 GB, 4 GB, 5 GB, 6 GB, 7 GB, 8 GB
2 vCPU	Between 4 GB and 16 GB in 1-GB increments
4 vCPU	Between 8 GB and 30 GB in 1-GB increments

For pricing information on these compute configurations, see AWS Fargate pricing.

Fargate storage

When provisioned, each pod running on Fargate receives 20 GB of container image layer storage. Pod storage is ephemeral. After a pod stops, the storage is deleted.

Note
The usable storage for Amazon EKS pods running on Fargate is less than 20GB because some space is used by the `kubelet` and other Kubernetes modules that are loaded inside the pod.

Storage

This chapter covers storage options for Amazon EKS clusters.

The Storage classes (p. 120) topic uses the in-tree Amazon EBS storage provisioner. The Amazon EBS CSI driver (p. 121) is available for managing storage in Kubernetes 1.14 and later clusters.

> **Note**
> The existing in-tree Amazon EBS plugin is still supported, but by using a CSI driver, you benefit from the decoupling of Kubernetes upstream release cycle and CSI driver release cycle. Eventually, the in-tree plugin will be deprecated in favor of the CSI driver.

Topics

- Storage classes (p. 120)
- Amazon EBS CSI driver (p. 121)
- Amazon EFS CSI driver (p. 125)
- Amazon FSx for Lustre CSI driver (p. 129)

Storage classes

Amazon EKS clusters that were created prior to Kubernetes version 1.11 were not created with any storage classes. You must define storage classes for your cluster to use and you should define a default storage class for your persistent volume claims. For more information, see Storage classes in the Kubernetes documentation.

> **Note**
> This topic uses the in-tree Amazon EBS storage provisioner. For Kubernetes 1.14 and later clusters, the Amazon EBS CSI driver (p. 121) is available for managing storage. The existing in-tree Amazon EBS plugin is still supported, but by using a CSI driver, you benefit from the decoupling of Kubernetes upstream release cycle and CSI driver release cycle. Eventually, the in-tree plugin will be deprecated in favor of the CSI driver.

To create an AWS storage class for your Amazon EKS cluster

1. Create an AWS storage class manifest file for your storage class. The `gp2-storage-class.yaml` example below defines a storage class called `gp2` that uses the Amazon EBS `gp2` volume type.

   For more information about the options available for AWS storage classes, see AWS EBS in the Kubernetes documentation.

   ```
   kind: StorageClass
   apiVersion: storage.k8s.io/v1
   metadata:
     name: gp2
     annotations:
       storageclass.kubernetes.io/is-default-class: "true"
   provisioner: kubernetes.io/aws-ebs
   parameters:
     type: gp2
   ```

```
    fsType: ext4
```

2. Use **kubectl** to create the storage class from the manifest file.

```
kubectl create -f gp2-storage-class.yaml
```

Output:

```
storageclass "gp2" created
```

To define a default storage class

1. List the existing storage classes for your cluster. A storage class must be defined before you can set it as a default.

```
kubectl get storageclass
```

Output:

```
NAME        PROVISIONER            AGE
gp2         kubernetes.io/aws-ebs   8m
```

2. Choose a storage class and set it as your default by setting the `storageclass.kubernetes.io/is-default-class=true` annotation.

```
kubectl patch storageclass gp2 -p '{"metadata": {"annotations":
{"storageclass.kubernetes.io/is-default-class":"true"}}}'
```

Output:

```
storageclass "gp2" patched
```

3. Verify that the storage class is now set as default.

```
kubectl get storageclass
```

Output:

```
gp2 (default)    kubernetes.io/aws-ebs    12m
```

Amazon EBS CSI driver

The Amazon EBS Container Storage Interface (CSI) driver provides a CSI interface that allows Amazon EKS clusters to manage the lifecycle of Amazon EBS volumes for persistent volumes.

This topic shows you how to deploy the Amazon EBS CSI Driver to your Amazon EKS cluster and verify that it works. We recommend using version v0.4.0 of the driver.

Note
This driver is only supported on Kubernetes version 1.14 and later Amazon EKS clusters and worker nodes. The driver is not supported on Fargate. Alpha features of the Amazon EBS CSI Driver are not supported on Amazon EKS clusters. The driver is in Beta release. It is well tested

and supported by Amazon EKS for production use. Support for the driver will not be dropped, though details may change. If the schema or schematics of the driver changes, instructions for migrating to the next version will be provided.

For detailed descriptions of the available parameters and complete examples that demonstrate the driver's features, see the Amazon EBS Container Storage Interface (CSI) driver project on GitHub.

To deploy the Amazon EBS CSI driver to an Amazon EKS cluster

1. Create an IAM policy called `Amazon_EBS_CSI_Driver` for your worker node instance profile that allows the Amazon EBS CSI Driver to make calls to AWS APIs on your behalf. Use the following AWS CLI commands to create the IAM policy in your AWS account. You can view the policy document on GitHub.

   a. Download the policy document from GitHub.

   ```
   curl -O https://raw.githubusercontent.com/kubernetes-sigs/aws-ebs-csi-driver/
   v0.4.0/docs/example-iam-policy.json
   ```

   b. Create the policy.

   ```
   aws iam create-policy --policy-name Amazon_EBS_CSI_Driver \
   --policy-document file://example-iam-policy.json
   ```

   Take note of the policy ARN that is returned.

2. Get the IAM role name for your worker nodes. Use the following command to print the `aws-auth` configmap.

   ```
   kubectl -n kube-system describe configmap aws-auth
   ```

   Output:

   ```
   Name:         aws-auth
   Namespace:    kube-system
   Labels:       <none>
   Annotations:  <none>

   Data
   ====
   mapRoles:
   ----
   - groups:
     - system:bootstrappers
     - system:nodes
     rolearn: arn:aws:iam::111122223333:role/eksctl-alb-nodegroup-ng-b1f603c5-
   NodeInstanceRole-GKNS581EASPU
     username: system:node:{{EC2PrivateDNSName}}

   Events:   <none>
   ```

   Record the role name for any `rolearn` values that have the `system:nodes` group assigned to them. In the previous example output, the role name is *eksctl-alb-nodegroup-ng-b1f603c5-NodeInstanceRole-GKNS581EASPU*. You should have one value for each node group in your cluster.

3. Attach the new `Amazon_EBS_CSI_Driver` IAM policy to each of the worker node IAM roles you identified earlier with the following command, substituting the red text with your own AWS account number and worker node IAM role name.

```
aws iam attach-role-policy \
--policy-arn arn:aws:iam::111122223333:policy/Amazon_EBS_CSI_Driver \
--role-name eksctl-alb-nodegroup-ng-b1f603c5-NodeInstanceRole-GKNS581EASPU
```

4. Deploy the Amazon EBS CSI Driver with the following command.

   Note
   This command requires version 1.14 or later of `kubectl`. You can see your `kubectl` version
   with the following command. To install or upgrade your `kubectl` version, see Installing
   kubectl (p. 182).

   ```
   kubectl version --client --short
   ```

   ```
   kubectl apply -k "github.com/kubernetes-sigs/aws-ebs-csi-driver/deploy/kubernetes/
   overlays/stable/?ref=master"
   ```

To deploy a sample application and verify that the CSI driver is working

This procedure uses the Dynamic volume provisioning example from the Amazon EBS Container Storage
Interface (CSI) driver GitHub repository to consume a dynamically-provisioned Amazon EBS volume.

1. Clone the Amazon EBS Container Storage Interface (CSI) driver GitHub repository to your local
   system.

   ```
   git clone https://github.com/kubernetes-sigs/aws-ebs-csi-driver.git
   ```

2. Navigate to the `dynamic-provisioning` example directory.

   ```
   cd aws-ebs-csi-driver/examples/kubernetes/dynamic-provisioning/
   ```

3. Deploy the `ebs-sc` storage class, `ebs-claim` persistent volume claim, and `app` sample application
   from the `specs` directory.

   ```
   kubectl apply -f specs/
   ```

4. Describe the `ebs-sc` storage class.

   ```
   kubectl describe storageclass ebs-sc
   ```

   Output:

   ```
   Name:            ebs-sc
   IsDefaultClass:  No
   Annotations:     kubectl.kubernetes.io/last-applied-
   configuration={"apiVersion":"storage.k8s.io/v1","kind":"StorageClass","metadata":
   {"annotations":{},"name":"ebs-
   sc"},"provisioner":"ebs.csi.aws.com","volumeBindingMode":"WaitForFirstConsumer"}

   Provisioner:           ebs.csi.aws.com
   Parameters:            <none>
   AllowVolumeExpansion:  <unset>
   MountOptions:          <none>
   ReclaimPolicy:         Delete
   VolumeBindingMode:     WaitForFirstConsumer
   Events:                <none>
   ```

Note that the storage class uses the `WaitForFirstConsumer` volume binding mode. This means that volumes are not dynamically provisioned until a pod makes a persistent volume claim. For more information, see Volume Binding Mode in the Kubernetes documentation.

5. Watch the pods in the default namespace and wait for the `app` pod to become ready.

```
kubectl get pods --watch
```

6. List the persistent volumes in the default namespace. Look for a persistent volume with the `default/ebs-claim` claim.

```
kubectl get pv
```

Output:

```
NAME                                           CAPACITY   ACCESS MODES   RECLAIM POLICY
  STATUS    CLAIM               STORAGECLASS   REASON   AGE
pvc-37717cd6-d0dc-11e9-b17f-06fad4858a5a    4Gi        RWO            Delete
  Bound     default/ebs-claim   ebs-sc                  30s
```

7. Describe the persistent volume.

```
kubectl describe pv pvc-37717cd6-d0dc-11e9-b17f-06fad4858a5a
```

Output:

```
Name:              pvc-37717cd6-d0dc-11e9-b17f-06fad4858a5a
Labels:            <none>
Annotations:       pv.kubernetes.io/provisioned-by: ebs.csi.aws.com
Finalizers:        [kubernetes.io/pv-protection external-attacher/ebs-csi-aws-com]
StorageClass:      ebs-sc
Status:            Bound
Claim:             default/ebs-claim
Reclaim Policy:    Delete
Access Modes:      RWO
VolumeMode:        Filesystem
Capacity:          4Gi
Node Affinity:
  Required Terms:
    Term 0:        topology.ebs.csi.aws.com/zone in [regiona]
Message:
Source:
    Type:          CSI (a Container Storage Interface (CSI) volume source)
    Driver:        ebs.csi.aws.com
    VolumeHandle:  vol-0d651e157c6d93445
    ReadOnly:      false
    VolumeAttributes:      storage.kubernetes.io/
csiProvisionerIdentity=1567792483192-8081-ebs.csi.aws.com
Events:                    <none>
```

The Amazon EBS volume ID is listed as the `VolumeHandle`.

8. Verify that the pod is successfully writing data to the volume.

```
kubectl exec -it app cat /data/out.txt
```

Output:

```
Fri Sep 6 19:26:53 UTC 2019
Fri Sep 6 19:26:58 UTC 2019
Fri Sep 6 19:27:03 UTC 2019
Fri Sep 6 19:27:08 UTC 2019
Fri Sep 6 19:27:13 UTC 2019
Fri Sep 6 19:27:18 UTC 2019
```

9. When you finish experimenting, delete the resources for this sample application to clean up.

```
kubectl delete -f specs/
```

Amazon EFS CSI driver

The Amazon EFS Container Storage Interface (CSI) driver provides a CSI interface that allows Amazon EKS clusters to manage the lifecycle of Amazon EFS file systems.

This topic shows you how to deploy the Amazon EFS CSI Driver to your Amazon EKS cluster and verify that it works.

Note
This driver is supported on Kubernetes version 1.14 and later Amazon EKS clusters and worker nodes. The driver is not supported on Fargate. Alpha features of the Amazon EFS CSI Driver are not supported on Amazon EKS clusters. The driver is in Beta release. It is well tested and supported by Amazon EKS for production use. Support for the driver will not be dropped, though details may change. If the schema or schematics of the driver changes, instructions for migrating to the next version will be provided.

For detailed descriptions of the available parameters and complete examples that demonstrate the driver's features, see the Amazon EFS Container Storage Interface (CSI) driver project on GitHub.

To deploy the Amazon EFS CSI driver to an Amazon EKS cluster

- Deploy the Amazon EFS CSI Driver with the following command.

    Note
    This command requires `kubectl` version 1.14 or later. You can see your `kubectl` version with the following command. To install or upgrade your `kubectl` version, see Installing kubectl (p. 182).

    ```
    kubectl version --client --short
    ```

    ```
    kubectl apply -k "github.com/kubernetes-sigs/aws-efs-csi-driver/deploy/kubernetes/
    overlays/stable/?ref=master"
    ```

To create an Amazon EFS file system for your Amazon EKS cluster

1. Locate the VPC ID for your Amazon EKS cluster. You can find this ID in the Amazon EKS console, or you can use the following AWS CLI command.

    ```
    aws eks describe-cluster --name cluster_name --query "cluster.resourcesVpcConfig.vpcId"
     --output text
    ```

    Output:

```
vpc-exampledb76d3e813
```

2. Locate the CIDR range for your cluster's VPC. You can find this in the Amazon VPC console, or you can use the following AWS CLI command.

```
aws ec2 describe-vpcs --vpc-ids vpc-exampledb76d3e813 --query "Vpcs[].CidrBlock" --
output text
```

Output:

```
192.168.0.0/16
```

3. Create a security group that allows inbound NFS traffic for your Amazon EFS mount points.

   a. Open the Amazon VPC console at https://console.aws.amazon.com/vpc/.

   b. Choose **Security Groups** in the left navigation pane, and then **Create security group**.

   c. Enter a name and description for your security group, and choose the VPC that your Amazon EKS cluster is using.

   d. Choose **Create** and then **Close** to finish.

4. Add a rule to your security group to allow inbound NFS traffic from your VPC CIDR range.

   a. Choose the security group that you created in the previous step.

   b. Choose the **Inbound Rules** tab and then choose **Edit rules**.

   c. Choose **Add Rule**, fill out the following fields, and then choose **Save rules**.

      - **Type**: NFS
      - **Source**: Custom. Paste the VPC CIDR range.
      - **Description**: Add a description, such as "Allows inbound NFS traffic from within the VPC."

5. Create the Amazon EFS file system for your Amazon EKS cluster.

   a. Open the Amazon Elastic File System console at https://console.aws.amazon.com/efs/.

   b. Choose **Create file system**.

   c. On the **Configure file system access** page, choose the VPC that your Amazon EKS cluster is using.

   d. For **Security groups**, add the security group that you created in the previous step to each mount target and choose **Next step**.

   e. Configure any optional settings for your file system, and then choose **Next step** and **Create File System** to finish.

      > **Important**
      > By default, new Amazon EFS file systems are owned by `root:root`, and only the `root` user (UID 0) has read-write-execute permissions. If your containers are not running as `root`, you must change the Amazon EFS file system permissions to allow other users to modify the file system. For more information, see Working with users, groups, and permissions at the Network File System (NFS) level in the *Amazon Elastic File System User Guide*.

To deploy a sample application and verify that the CSI driver is working

This procedure uses the Multiple Pods Read Write Many example from the Amazon EFS Container Storage Interface (CSI) driver GitHub repository to consume a statically provisioned Amazon EFS persistent volume and access it from multiple pods with the `ReadWriteMany` access mode.

1. Clone the Amazon EFS Container Storage Interface (CSI) driver GitHub repository to your local system.

   ```
   git clone https://github.com/kubernetes-sigs/aws-efs-csi-driver.git
   ```

2. Navigate to the `multiple_pods` example directory.

   ```
   cd aws-efs-csi-driver/examples/kubernetes/multiple_pods/
   ```

3. Retrieve your Amazon EFS file system ID. You can find this in the Amazon EFS console, or use the following AWS CLI command.

   ```
   aws efs describe-file-systems --query "FileSystems[*].FileSystemId" --output text
   ```

   Output:

   ```
   fs-582a03f3
   ```

4. Edit the `specs/pv.yaml` file and replace the `volumeHandle` value with your Amazon EFS file system ID.

   ```
   apiVersion: v1
   kind: PersistentVolume
   metadata:
     name: efs-pv
   spec:
     capacity:
       storage: 5Gi
     volumeMode: Filesystem
     accessModes:
       - ReadWriteMany
     persistentVolumeReclaimPolicy: Retain
     storageClassName: efs-sc
     csi:
       driver: efs.csi.aws.com
       volumeHandle: fs-582a03f3
   ```

   Note
   Because Amazon EFS is an elastic file system, it does not enforce any file system capacity limits. The actual storage capacity value in persistent volumes and persistent volume claims is not used when creating the file system. However, since storage capacity is a required field in Kubernetes, you must specify a valid value, such as, `5Gi` in this example. This value does not limit the size of your Amazon EFS file system.

5. Deploy the `efs-sc` storage class, `efs-claim` persistent volume claim, `efs-pv` persistent volume, and `app1` and `app2` sample applications from the `specs` directory.

   ```
   kubectl apply -f specs/
   ```

6. Watch the pods in the default namespace and wait for the `app1` and `app2` pods to become ready.

   ```
   kubectl get pods --watch
   ```

7. List the persistent volumes in the default namespace. Look for a persistent volume with the `default/efs-claim` claim.

   ```
   kubectl get pv
   ```

Output:

```
NAME       CAPACITY    ACCESS MODES    RECLAIM POLICY    STATUS    CLAIM
 STORAGECLASS   REASON    AGE
efs-pv    5Gi          RWX              Retain           Bound    default/efs-claim    efs-sc
               2m50s
```

8. Describe the persistent volume.

    ```
    kubectl describe pv efs-pv
    ```

    Output:

    ```
    Name:           efs-pv
    Labels:         <none>
    Annotations:    kubectl.kubernetes.io/last-applied-configuration:
                      {"apiVersion":"v1","kind":"PersistentVolume","metadata":
    {"annotations":{},"name":"efs-pv"},"spec":{"accessModes":["ReadWriteMany"],"capaci...
                      pv.kubernetes.io/bound-by-controller: yes
    Finalizers:     [kubernetes.io/pv-protection]
    StorageClass:   efs-sc
    Status:         Bound
    Claim:          default/efs-claim
    Reclaim Policy: Retain
    Access Modes:   RWX
    VolumeMode:     Filesystem
    Capacity:       5Gi
    Node Affinity:  <none>
    Message:
    Source:
        Type:           CSI (a Container Storage Interface (CSI) volume source)
        Driver:         efs.csi.aws.com
        VolumeHandle:   fs-582a03f3
        ReadOnly:       false
        VolumeAttributes:  <none>
    Events:             <none>
    ```

    The Amazon EFS file system ID is listed as the `VolumeHandle`.

9. Verify that the `app1` pod is successfully writing data to the volume.

    ```
    kubectl exec -ti app1 -- tail /data/out1.txt
    ```

    Output:

    ```
    Wed Sep 18 20:30:48 UTC 2019
    Wed Sep 18 20:30:53 UTC 2019
    Wed Sep 18 20:30:58 UTC 2019
    Wed Sep 18 20:31:03 UTC 2019
    Wed Sep 18 20:31:08 UTC 2019
    Wed Sep 18 20:31:13 UTC 2019
    ```

10. Verify that the `app2` pod is shows the same data in the volume.

    ```
    kubectl exec -ti app2 -- tail /data/out1.txt
    ```

    Output:

```
Wed Sep 18 20:30:48 UTC 2019
Wed Sep 18 20:30:53 UTC 2019
Wed Sep 18 20:30:58 UTC 2019
Wed Sep 18 20:31:03 UTC 2019
Wed Sep 18 20:31:08 UTC 2019
Wed Sep 18 20:31:13 UTC 2019
```

11. When you finish experimenting, delete the resources for this sample application to clean up.

```
kubectl delete -f specs/
```

Amazon FSx for Lustre CSI driver

The Amazon FSx for Lustre Container Storage Interface (CSI) driver provides a CSI interface that allows Amazon EKS clusters to manage the lifecycle of Amazon FSx for Lustre file systems.

This topic shows you how to deploy the Amazon FSx for Lustre CSI Driver to your Amazon EKS cluster and verify that it works. We recommend using version 0.3.0 of the driver.

> **Note**
> This driver is supported on Kubernetes version 1.16 and later Amazon EKS clusters and worker nodes. The driver is not supported on Fargate. Alpha features of the Amazon FSx for Lustre CSI Driver are not supported on Amazon EKS clusters. The driver is in Beta release. It is well tested and supported by Amazon EKS for production use. Support for the driver will not be dropped, though details may change. If the schema or schematics of the driver changes, instructions for migrating to the next version will be provided.

For detailed descriptions of the available parameters and complete examples that demonstrate the driver's features, see the Amazon FSx for Lustre Container Storage Interface (CSI) driver project on GitHub.

Prerequisites

You must have:

- Version 1.18.49 or later of the AWS CLI installed. You can check your currently-installed version with the `aws --version` command. To install or upgrade the AWS CLI, see Installing the AWS CLI.
- An existing Amazon EKS cluster. If you don't currently have a cluster, see ??? (p. 3) to create one.
- Version 0.19.0 or later of `eksctl` installed. You can check your currently-installed version with the `eksctl version` command. To install or upgrade `eksctl`, see Installing or upgrading eksctl (p. 197).
- The latest version of `kubectl` installed that aligns to your cluster version. You can check your currently-installed version with the `kubectl version --short --client` command. For more information, see Installing kubectl (p. 182).

To deploy the Amazon FSx for Lustre CSI driver to an Amazon EKS cluster

1. Create an AWS Identity and Access Management OIDC provider and associate it with your cluster.

```
eksctl utils associate-iam-oidc-provider \
    --region region-code \
    --cluster prod \
    --approve
```

2. Create an IAM policy and service account that allows the driver to make calls to AWS APIs on your behalf.

   a. Copy the following text and save it to a file named `fsx-csi-driver.json`.

```
{
    "Version":"2012-10-17",
    "Statement":[
        {
            "Effect":"Allow",
            "Action":[
                "iam:CreateServiceLinkedRole",
                "iam:AttachRolePolicy",
                "iam:PutRolePolicy"
            ],
            "Resource":"arn:aws:iam::*:role/aws-service-role/s3.data-
source.lustre.fsx.amazonaws.com/*"
        },
        {
            "Action":"iam:CreateServiceLinkedRole",
            "Effect":"Allow",
            "Resource":"*",
            "Condition":{
                "StringLike":{
                    "iam:AWSServiceName":[
                        "fsx.amazonaws.com"
                    ]
                }
            }
        },
        {
            "Effect":"Allow",
            "Action":[
                "s3:ListBucket",
                "fsx:CreateFileSystem",
                "fsx:DeleteFileSystem",
                "fsx:DescribeFileSystems"
            ],
            "Resource":[
                "*"
            ]
        }
    ]
}
```

   b. Create the policy.

```
aws iam create-policy \
    --policy-name Amazon_FSx_Lustre_CSI_Driver \
    --policy-document file://fsx-csi-driver.json
```

   Take note of the policy Amazon Resource Name (ARN) that is returned.

3. Create a Kubernetes service account for the driver and attach the policy to the service account. Replacing the ARN of the policy with the ARN returned in the previous step.

```
eksctl create iamserviceaccount \
    --region region-code \
    --name fsx-csi-controller-sa \
    --namespace kube-system \
    --cluster prod \
    --attach-policy-arn arn:aws:iam::111122223333:policy/Amazon_FSx_Lustre_CSI_Driver \
```

```
    --approve
```

Output:

You'll see several lines of output as the service account is created. The last line of output is similar to the following example line.

```
[#]  created serviceaccount "kube-system/fsx-csi-controller-sa"
```

Note the name of the AWS CloudFormation stack that was deployed. In the example output above, the stack is named `eksctl-prod-addon-iamserviceaccount-kube-system-fsx-csi-controller-sa`.

4. Note the **Role ARN** for the role that was created.

   a. Open the AWS CloudFormation console at https://console.aws.amazon.com/cloudformation.

   b. Ensure that the console is set to the region that you created your IAM role in and then select **Stacks**.

   c. Select the stack named `eksctl-prod-addon-iamserviceaccount-kube-system-fsx-csi-controller-sa`.

   d. Select the **Outputs** tab. The **Role ARN** is listed on the **Output(1)** page.

5. Deploy the driver with the following command.

```
kubectl apply -k "github.com/kubernetes-sigs/aws-fsx-csi-driver/deploy/kubernetes/
overlays/stable/?ref=master"
```

Output

```
Warning: kubectl apply should be used on resource created by either kubectl create --
save-config or kubectl apply
serviceaccount/fsx-csi-controller-sa configured
clusterrole.rbac.authorization.k8s.io/fsx-csi-external-provisioner-role created
clusterrolebinding.rbac.authorization.k8s.io/fsx-csi-external-provisioner-binding
 created
deployment.apps/fsx-csi-controller created
daemonset.apps/fsx-csi-node created
csidriver.storage.k8s.io/fsx.csi.aws.com created
```

6. Patch the driver deployment to add the service account that you created in step 3, replacing the ARN with the ARN that you noted in step 4.

```
kubectl annotate serviceaccount -n kube-system fsx-csi-controller-sa \
 eks.amazonaws.com/role-arn=arn:aws:iam::111122223333:role/eksctl-prod-addon-
iamserviceaccount-kube-sys-Role1-NPFTLHJ5PJF5 --overwrite=true
```

To deploy a Kubernetes storage class, persistent volume claim, and sample application to verify that the CSI driver is working

This procedure uses the Dynamic volume provisioning for Amazon S3 from the Amazon FSx for Lustre Container Storage Interface (CSI) driver GitHub repository to consume a dynamically-provisioned Amazon FSx for Lustre volume.

1. Create an Amazon S3 bucket and a folder within it named `export` by creating and copying a file to the bucket.

```
aws s3 mb s3://fsx-csi
```

```
echo test-file >> testfile
aws s3 cp testfile s3://fsx-csi/export/testfile
```

2. Download the `storageclass` manifest with the following command.

```
curl -o storageclass.yaml https://raw.githubusercontent.com/kubernetes-sigs/aws-fsx-
csi-driver/master/examples/kubernetes/dynamic_provisioning_s3/specs/storageclass.yaml
```

3. Edit the file and replace the existing, `alternate-colored` values with your own.

```
parameters:
  subnetId: subnet-056da83524edbe641
  securityGroupIds: sg-086f61ea73388fb6b
  s3ImportPath: s3://ml-training-data-000
  s3ExportPath: s3://ml-training-data-000/export
  deploymentType: SCRATCH_2
```

- **subnetId** – The subnet ID that the Amazon FSx for Lustre file system should be created in. Amazon FSx for Lustre is not supported in all Availability Zones. Open the Amazon FSx for Lustre console at https://console.aws.amazon.com/fsx/ to confirm that the subnet that you want to use is in a supported Availability Zone. The subnet can include your worker nodes, or can be a different subnet or VPC. If the subnet that you specify is not the same subnet that you have worker nodes in, then your VPCs must be connected, and you must ensure that you have the necessary ports open in your security groups.

- **securityGroupIds** – The security group ID for your worker nodes.

- **s3ImportPath** – The Amazon Simple Storage Service data repository that you want to copy data from to the persistent volume. Specify the `fsx-csi` bucket that you created in step 1.

- **s3ExportPath** – The Amazon S3 data repository that you want to export new or modified files to. Specify the `fsx-csi/export` folder that you created in step 1.

- **deploymentType** – The file system deployment type. Valid values are `SCRATCH_1`, `SCRATCH_2`, and `PERSISTENT_1`. For more information about deployment types, see Create your Amazon FSx for Lustre file system.

    Note
    The Amazon S3 bucket for `s3ImportPath` and `s3ExportPath` must be the same, otherwise the driver cannot create the Amazon FSx for Lustre file system. The `s3ImportPath` can stand alone. A random path will be created automatically like `s3://ml-training-data-000/FSxLustre20190308T012310Z`. The `s3ExportPath` cannot be used without specifying a value for `S3ImportPath`.

4. Create the `storageclass`.

```
kubectl apply -f storageclass.yaml
```

5. Download the persistent volume claim manifest.

```
curl -o claim.yaml https://raw.githubusercontent.com/kubernetes-sigs/aws-fsx-csi-
driver/master/examples/kubernetes/dynamic_provisioning_s3/specs/claim.yaml
```

6. (Optional) Edit the `claim.yaml` file. Change the following `value` to one of the increment values listed below, based on your storage requirements and the `deploymentType` that you selected in a previous step.

```
storage: 1200Gi
```

- `SCRATCH_2` and `PERSISTENT` – 1.2 TiB, 2.4 TiB, or increments of 2.4 TiB over 2.4 TiB.

- `SCRATCH_1` – 1.2 TiB, 2.4 TiB, 3.6 TiB, or increments of 3.6 TiB over 3.6 TiB.

7. Create the persistent volume claim.

```
kubectl apply -f claim.yaml
```

8. Confirm that the file system is provisioned.

```
kubectl get pvc
```

Output.

```
NAME         STATUS   VOLUME                                   CAPACITY   ACCESS MODES
   STORAGECLASS   AGE
fsx-claim    Bound    pvc-15dad3c1-2365-11ea-a836-02468c18769e   1200Gi     RWX
   fsx-sc         7m37s
```

> **Note**
> The `STATUS` may show as `Pending` for 5-10 minutes, before changing to `Bound`. Don't
> continue with the next step until the `STATUS` is `Bound`.

9. Deploy the sample application.

```
kubectl apply -f https://raw.githubusercontent.com/kubernetes-sigs/aws-fsx-csi-driver/
master/examples/kubernetes/dynamic_provisioning_s3/specs/pod.yaml
```

10. Verify that the sample application is running.

```
kubectl get pods
```

Output

```
NAME       READY   STATUS       RESTARTS   AGE
fsx-app    1/1     Running      0          8s
```

Access Amazon S3 files from the Amazon FSx for Lustre file system

If you only want to import data and read it without any modification and creation, then you don't need a
value for `s3ExportPath` in your `storageclass.yaml` file. Verify that data was written to the Amazon
FSx for Lustre file system by the sample app.

```
kubectl exec -it fsx-app ls /data
```

Output.

```
export  out.txt
```

The sample app wrote the `out.txt` file to the file system.

Archive files to the `s3ExportPath`

For new files and modified files, you can use the Lustre user space tool to archive the data back to
Amazon S3 using the value that you specified for `s3ExportPath`.

1. Export the file back to Amazon S3.

```
kubectl exec -ti fsx-app -- lfs hsm_archive /data/out.txt
```

Note

- New files aren't synced back to Amazon S3 automatically. In order to sync files to the `s3ExportPath`, you need to install the Lustre client in your container image and manually run the `lfs hsm_archive` command. The container should run in privileged mode with the `CAP_SYS_ADMIN` capability.
- This example uses a lifecycle hook to install the Lustre client for demonstration purpose. A normal approach is building a container image with the Lustre client.

2. Confirm that the `out.txt` file was written to the `s3ExportPath` folder in Amazon S3.

```
aws s3 ls fsx-csi/export/
```

Output

```
2019-12-23 12:11:35        4553 out.txt
2019-12-23 11:41:21          10 testfile
```

Autoscaling

This chapter covers various autoscaling configurations for your Amazon EKS cluster. There are several types of Kubernetes autoscaling supported in Amazon EKS:

- Cluster Autoscaler (p. 135) — The Kubernetes Cluster Autoscaler automatically adjusts the number of nodes in your cluster when pods fail to launch due to lack of resources or when nodes in the cluster are underutilized and their pods can be rescheduled on to other nodes in the cluster.
- Horizontal Pod Autoscaler (p. 139) — The Kubernetes Horizontal Pod Autoscaler automatically scales the number of pods in a deployment, replication controller, or replica set based on that resource's CPU utilization.
- Vertical Pod Autoscaler (p. 141) — The Kubernetes Vertical Pod Autoscaler automatically adjusts the CPU and memory reservations for your pods to help "right size" your applications. This can help you to better use your cluster resources and free up CPU and memory for other pods.

Cluster Autoscaler

The Kubernetes Cluster Autoscaler automatically adjusts the number of nodes in your cluster when pods fail to launch due to lack of resources or when nodes in the cluster are underutilized and their pods can be rescheduled onto other nodes in the cluster.

This topic shows you how to deploy the Cluster Autoscaler to your Amazon EKS cluster and how to configure it to modify your Amazon EC2 Auto Scaling groups. The Cluster Autoscaler modifies your worker node groups so that they scale out when you need more resources and scale in when you have underutilized resources.

Create an Amazon EKS cluster

This section helps you to create a cluster and node group or groups. If you already have a cluster, you can skip ahead to Cluster Autoscaler node group considerations (p. 136).

If you are running a stateful application across multiple Availability Zones that is backed by Amazon EBS volumes and using the Kubernetes Cluster Autoscaler (p. 135), you should configure multiple node groups, each scoped to a single Availability Zone. In addition, you should enable the `--balance-similar-node-groups` feature. Otherwise, you can create a single node group that spans multiple Availability Zones.

Choose one of the cluster creation procedures below that meets your requirements.

To create a cluster with a single managed group that spans multiple Availability Zones

- Create an Amazon EKS cluster with a single managed node group with the following `eksctl` command. For more information, see Creating an Amazon EKS cluster (p. 21). Substitute the *variable text* with your own values.

```
eksctl create cluster --name my-cluster --version 1.16 --managed --asg-access
```

Portions of the output showing the Availability Zones:

```
...
[#]  using region region-code
[#]  setting availability zones to [region-codea region-codeb region-codec]
[#]  subnets for region-codea - public:192.168.0.0/19 private:192.168.96.0/19
```

```
[#]   subnets for region-codeb - public:192.168.32.0/19 private:192.168.128.0/19
[#]   subnets for region-codec - public:192.168.64.0/19 private:192.168.160.0/19
...
[#]   nodegroup "ng-6bcca56a" has 2 node(s)
[#]   node "ip-192-168-28-68.region-code.compute.internal" is ready
[#]   node "ip-192-168-61-153.region-code.compute.internal" is ready
[#]   waiting for at least 2 node(s) to become ready in "ng-6bcca56a"
[#]   nodegroup "ng-6bcca56a" has 2 node(s)
[#]   node "ip-192-168-28-68.region-code.compute.internal" is ready
[#]   node "ip-192-168-61-153.region-code.compute.internal" is ready
...
[#]   EKS cluster "my-cluster" in "region-code" region-code is ready
```

To create a cluster with a dedicated managed node group for each Availability Zone

1. Create an Amazon EKS cluster with no node groups with the following `eksctl` command. For more information, see Creating an Amazon EKS cluster (p. 21). Note the Availability Zones that the cluster is created in. You will use these Availability Zones when you create your node groups. Substitute the red variable text with your own values.

    ```
    eksctl create cluster --name my-cluster --version 1.16 --without-nodegroup
    ```

    Portions of the output showing the Availability Zones:

    ```
    ...
    [#]   using region region-code
    [#]   setting availability zones to [region-codea region-codec region-codeb]
    [#]   subnets for region-codea - public:192.168.0.0/19 private:192.168.96.0/19
    [#]   subnets for region-codec - public:192.168.32.0/19 private:192.168.128.0/19
    [#]   subnets for region-codeb - public:192.168.64.0/19 private:192.168.160.0/19
    ...
    [#]   EKS cluster "my-cluster" in "region-code" region is ready
    ```

    This cluster was created in the following Availability Zones: *region-code*a, *region-code*c, and *region-code*b.

2. For each Availability Zone in your cluster, use the following `eksctl` command to create a node group. Substitute the *variable text* with your own values. This command creates an Auto Scaling group with a minimum count of one and a maximum count of ten.

    ```
    eksctl create nodegroup --cluster my-cluster --node-zones region-codea --name region-
    codea --asg-access --nodes-min 1 --nodes 5 --nodes-max 10 --managed
    ```

Cluster Autoscaler node group considerations

The Cluster Autoscaler requires additional IAM and resource tagging considerations that are explained in this section.

Node group IAM policy

The Cluster Autoscaler requires the following IAM permissions to make calls to AWS APIs on your behalf.

If you used the previous `eksctl` commands to create your node groups, these permissions are automatically provided and attached to your worker node IAM roles. If you did not use `eksctl`, you must create an IAM policy with the following document and attach it to your worker node IAM roles. For more information, see Modifying a role in the *IAM User Guide*.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Action": [
                "autoscaling:DescribeAutoScalingGroups",
                "autoscaling:DescribeAutoScalingInstances",
                "autoscaling:DescribeLaunchConfigurations",
                "autoscaling:DescribeTags",
                "autoscaling:SetDesiredCapacity",
                "autoscaling:TerminateInstanceInAutoScalingGroup",
                "ec2:DescribeLaunchTemplateVersions"
            ],
            "Resource": "*",
            "Effect": "Allow"
        }
    ]
}
```

Auto Scaling group tags

The Cluster Autoscaler requires the following tags on your node group Auto Scaling groups so that they can be auto-discovered.

If you used the previous `eksctl` commands to create your node groups, these tags are automatically applied. If not, you must manually tag your Auto Scaling groups with the following tags. For more information, see Tagging your Amazon EC2 resources in the *Amazon EC2 User Guide for Linux Instances*.

Key	Value
`k8s.io/cluster-autoscaler/`*`<cluster-name>`*	`owned`
`k8s.io/cluster-autoscaler/enabled`	`true`

Deploy the Cluster Autoscaler

To deploy the Cluster Autoscaler

1. Deploy the Cluster Autoscaler to your cluster with the following command.

    ```
    kubectl apply -f https://raw.githubusercontent.com/kubernetes/autoscaler/master/
    cluster-autoscaler/cloudprovider/aws/examples/cluster-autoscaler-autodiscover.yaml
    ```

2. Add the `cluster-autoscaler.kubernetes.io/safe-to-evict` annotation to the deployment with the following command.

    ```
    kubectl -n kube-system annotate deployment.apps/cluster-autoscaler cluster-
    autoscaler.kubernetes.io/safe-to-evict="false"
    ```

3. Edit the Cluster Autoscaler deployment with the following command.

    ```
    kubectl -n kube-system edit deployment.apps/cluster-autoscaler
    ```

    Edit the `cluster-autoscaler` container command to replace *`<YOUR CLUSTER NAME>`* with your cluster's name, and add the following options.

- *--balance-similar-node-groups*
- *--skip-nodes-with-system-pods=false*

```
    spec:
      containers:
      - command:
        - ./cluster-autoscaler
        - --v=4
        - --stderrthreshold=info
        - --cloud-provider=aws
        - --skip-nodes-with-local-storage=false
        - --expander=least-waste
        - --node-group-auto-discovery=asg:tag=k8s.io/cluster-autoscaler/enabled,k8s.io/
cluster-autoscaler/<YOUR CLUSTER NAME>
        - --balance-similar-node-groups
        - --skip-nodes-with-system-pods=false
```

Save and close the file to apply the changes.

4. Open the Cluster Autoscaler releases page in a web browser and find the latest Cluster Autoscaler version that matches your cluster's Kubernetes major and minor version. For example, if your cluster's Kubernetes version is 1.16 find the latest Cluster Autoscaler release that begins with 1.16. Record the semantic version number (1.16.*n*) for that release to use in the next step.

5. Set the Cluster Autoscaler image tag to the version that you recorded in the previous step with the following command. Replace *1.15.n* with your own value. You can replace us with *asia* or *eu*.

```
kubectl -n kube-system set image deployment.apps/cluster-autoscaler cluster-
autoscaler=us.gcr.io/k8s-artifacts-prod/autoscaling/cluster-autoscaler:v1.15.n
```

> **Note**
> Depending on the version that you need, you may need to change the previous address to `gcr.io/google-containers/cluster-autoscaler:v1.n.n`. The image address is listed on the releases page.

View your Cluster Autoscaler logs

After you have deployed the Cluster Autoscaler, you can view the logs and verify that it is monitoring your cluster load.

View your Cluster Autoscaler logs with the following command.

```
kubectl -n kube-system logs -f deployment.apps/cluster-autoscaler
```

Output:

```
I0926 23:15:55.165842       1 static_autoscaler.go:138] Starting main loop
I0926 23:15:55.166279       1 utils.go:595] No pod using affinity / antiaffinity found in
 cluster, disabling affinity predicate for this loop
I0926 23:15:55.166293       1 static_autoscaler.go:294] Filtering out schedulables
I0926 23:15:55.166330       1 static_autoscaler.go:311] No schedulable pods
I0926 23:15:55.166338       1 static_autoscaler.go:319] No unschedulable pods
I0926 23:15:55.166345       1 static_autoscaler.go:366] Calculating unneeded nodes
I0926 23:15:55.166357       1 utils.go:552] Skipping ip-192-168-3-111.region-
code.compute.internal - node group min size reached
I0926 23:15:55.166365       1 utils.go:552] Skipping ip-192-168-71-83.region-
code.compute.internal - node group min size reached
```

```
I0926 23:15:55.166373        1 utils.go:552] Skipping ip-192-168-60-191.region-
code.compute.internal - node group min size reached
I0926 23:15:55.166435        1 static_autoscaler.go:393] Scale down status:
 unneededOnly=false lastScaleUpTime=2019-09-26 21:42:40.908059094 ...
I0926 23:15:55.166458        1 static_autoscaler.go:403] Starting scale down
I0926 23:15:55.166488        1 scale_down.go:706] No candidates for scale down
```

Horizontal Pod Autoscaler

The Kubernetes Horizontal Pod Autoscaler automatically scales the number of pods in a deployment, replication controller, or replica set based on that resource's CPU utilization. This can help your applications scale out to meet increased demand or scale in when resources are not needed, thus freeing up your worker nodes for other applications. When you set a target CPU utilization percentage, the Horizontal Pod Autoscaler scales your application in or out to try to meet that target.

The Horizontal Pod Autoscaler is a standard API resource in Kubernetes that simply requires that a metrics source (such as the Kubernetes metrics server) is installed on your Amazon EKS cluster to work. You do not need to deploy or install the Horizontal Pod Autoscaler on your cluster to begin scaling your applications. For more information, see Horizontal pod autoscaler in the Kubernetes documentation.

Use this topic to prepare the Horizontal Pod Autoscaler for your Amazon EKS cluster and to verify that it is working with a sample application.

> **Note**
> This topic is based on the Horizontal pod autoscaler walkthrough in the Kubernetes documentation.

Install the metrics server

The Kubernetes metrics server is an aggregator of resource usage data in your cluster. The metrics server is not deployed by default in Amazon EKS clusters, but it provides metrics that are required by the Horizontal Pod Autoscaler. This topic explains how to deploy the Kubernetes metrics server on your Amazon EKS cluster.

If you have already deployed the metrics server to your cluster, you can move on to the next section. You can check for the metrics server with the following command.

```
kubectl -n kube-system get deployment/metrics-server
```

If this command returns a `NotFound` error, then you must deploy the metrics server to your Amazon EKS cluster.

To deploy the Metrics Server

1. Deploy the Metrics Server with the following command:

   ```
   kubectl apply -f https://github.com/kubernetes-sigs/metrics-server/releases/download/
   v0.3.6/components.yaml
   ```

2. Verify that the `metrics-server` deployment is running the desired number of pods with the following command.

   ```
   kubectl get deployment metrics-server -n kube-system
   ```

   Output

```
NAME            READY    UP-TO-DATE    AVAILABLE    AGE
metrics-server  1/1      1             1            6m
```

Run a Horizontal Pod Autoscaler test application

In this section, you deploy a sample application to verify that the Horizontal Pod Autoscaler is working.

Note

This example is based on the Horizontal pod autoscaler walkthrough in the Kubernetes documentation.

To test your Horizontal Pod Autoscaler installation

1. Create a simple Apache web server application with the following command.

```
kubectl run httpd --image=httpd --requests=cpu=100m --limits=cpu=200m --expose --
port=80
```

This Apache web server pod is given a 200 millicpu CPU limit and it is serving on port 80.

2. Create a Horizontal Pod Autoscaler resource for the `httpd` deployment.

```
kubectl autoscale deployment httpd --cpu-percent=50 --min=1 --max=10
```

This command creates an autoscaler that targets 50 percent CPU utilization for the deployment, with a minimum of one pod and a maximum of ten pods. When the average CPU load is below 50 percent, the autoscaler tries to reduce the number of pods in the deployment, to a minimum of one. When the load is greater than 50 percent, the autoscaler tries to increase the number of pods in the deployment, up to a maximum of ten. For more information, see How does the Horizontal Pod Autoscaler work? in the Kubernetes documentation.

3. Describe the autoscaler with the following command to view its details.

```
kubectl describe hpa/httpd
```

Output:

```
Name:                                                    httpd
Namespace:                                               default
Labels:                                                  <none>
Annotations:                                             <none>
CreationTimestamp:                                       Fri, 27 Sep 2019 13:32:15 -0700
Reference:                                               Deployment/httpd
Metrics:                                                 ( current / target )
  resource cpu on pods  (as a percentage of request):   1% (1m) / 50%
Min replicas:                                            1
Max replicas:                                            10
Deployment pods:                                         1 current / 1 desired
Conditions:
  Type            Status   Reason              Message
  ----            ------   ------              -------
  AbleToScale     True     ReadyForNewScale    recommended size matches current size
  ScalingActive   True     ValidMetricFound    the HPA was able to successfully
 calculate a replica count from cpu resource utilization (percentage of request)
  ScalingLimited  False    DesiredWithinRange  the desired count is within the
 acceptable range
Events:            <none>
```

As you can see, the current CPU load is only one percent, but the pod count is already at its lowest boundary (one), so it cannot scale in.

4. Create a load for the web server. The following command uses the Apache Bench program to send hundreds of thousands of requests to the `httpd` server. This should significantly increase the load and cause the autoscaler to scale out the deployment.

```
kubectl run apache-bench -i --tty --rm --image=httpd -- ab -n 500000 -c 1000 http://
httpd.default.svc.cluster.local/
```

5. Watch the `httpd` deployment scale out while the load is generated. To watch the deployment and the autoscaler, periodically run the following command.

```
kubectl get horizontalpodautoscaler.autoscaling/httpd
```

Output:

```
NAME     REFERENCE          TARGETS    MINPODS   MAXPODS   REPLICAS   AGE
httpd    Deployment/httpd   76%/50%    1         10        10         4m50s
```

When the load finishes, the deployment should scale back down to 1.

6. When you are done experimenting with your sample application, delete the `httpd` resources.

```
kubectl delete deployment.apps/httpd service/httpd horizontalpodautoscaler.autoscaling/
httpd
```

Vertical Pod Autoscaler

The Kubernetes Vertical Pod Autoscaler automatically adjusts the CPU and memory reservations for your pods to help "right size" your applications. This adjustment can improve cluster resource utilization and free up CPU and memory for other pods. This topic helps you to deploy the Vertical Pod Autoscaler to your cluster and verify that it is working.

Install the metrics server

The Kubernetes metrics server is an aggregator of resource usage data in your cluster. It is not deployed by default in Amazon EKS clusters, but it provides metrics that are required by the Vertical Pod Autoscaler. This topic explains how to deploy the Kubernetes metrics server on your Amazon EKS cluster.

> **Note**
> You can also use Prometheus to provide metrics for the Vertical Pod Autoscaler. For more information, see Control plane metrics with Prometheus (p. 203).

If you have already deployed the metrics server to your cluster, you can move on to the next section. You can check for the metrics server with the following command.

```
kubectl -n kube-system get deployment/metrics-server
```

If this command returns a `NotFound` error, then you must deploy the metrics server to your Amazon EKS cluster.

To deploy the Metrics Server

1. Deploy the Metrics Server with the following command:

```
kubectl apply -f https://github.com/kubernetes-sigs/metrics-server/releases/download/
v0.3.6/components.yaml
```

2. Verify that the `metrics-server` deployment is running the desired number of pods with the following command.

```
kubectl get deployment metrics-server -n kube-system
```

Output

```
NAME             READY    UP-TO-DATE    AVAILABLE    AGE
metrics-server   1/1      1             1            6m
```

Deploy the Vertical Pod Autoscaler

In this section, you deploy the Vertical Pod Autoscaler to your cluster.

To deploy the Vertical Pod Autoscaler

1. Open a terminal window and navigate to a directory where you would like to download the Vertical Pod Autoscaler source code.
2. Clone the kubernetes/autoscaler GitHub repository.

```
git clone https://github.com/kubernetes/autoscaler.git
```

3. Change to the `vertical-pod-autoscaler` directory.

```
cd autoscaler/vertical-pod-autoscaler/
```

4. (Optional) If you have already deployed another version of the Vertical Pod Autoscaler, remove it with the following command.

```
./hack/vpa-down.sh
```

5. Deploy the Vertical Pod Autoscaler to your cluster with the following command.

```
./hack/vpa-up.sh
```

6. Verify that the Vertical Pod Autoscaler pods have been created successfully.

```
kubectl get pods -n kube-system
```

Output:

```
NAME                                        READY    STATUS     RESTARTS    AGE
aws-node-949vx                              1/1      Running    0           122m
aws-node-b4nj8                              1/1      Running    0           122m
coredns-6c75b69b98-r9x68                    1/1      Running    0           133m
coredns-6c75b69b98-rt9bp                    1/1      Running    0           133m
kube-proxy-bkm6b                            1/1      Running    0           122m
kube-proxy-hpqm2                            1/1      Running    0           122m
metrics-server-8459fc497-kfj8w              1/1      Running    0           83m
vpa-admission-controller-68c748777d-ppspd   1/1      Running    0           7s
vpa-recommender-6fc8c67d85-gljpl            1/1      Running    0           8s
```

```
vpa-updater-786b96955c-bgp9d                   1/1      Running   0          8s
```

Test your Vertical Pod Autoscaler installation

In this section, you deploy a sample application to verify that the Vertical Pod Autoscaler is working.

To test your Vertical Pod Autoscaler installation

1. Deploy the `hamster.yaml` Vertical Pod Autoscaler example with the following command.

   ```
   kubectl apply -f examples/hamster.yaml
   ```

2. Get the pods from the `hamster` example application.

   ```
   kubectl get pods -l app=hamster
   ```

   Output:

   ```
   hamster-c7d89d6db-rglf5   1/1      Running   0          48s
   hamster-c7d89d6db-znvz5   1/1      Running   0          48s
   ```

3. Describe one of the pods to view its CPU and memory reservation.

   ```
   kubectl describe pod hamster-c7d89d6db-rglf5
   ```

   Output:

   ```
   Name:          hamster-c7d89d6db-rglf5
   Namespace:     default
   Priority:      0
   Node:          ip-192-168-9-44.region-code.compute.internal/192.168.9.44
   Start Time:    Fri, 27 Sep 2019 10:35:15 -0700
   Labels:        app=hamster
                  pod-template-hash=c7d89d6db
   Annotations:   kubernetes.io/psp: eks.privileged
                  vpaUpdates: Pod resources updated by hamster-vpa: container 0:
   Status:        Running
   IP:            192.168.23.42
   IPs:           <none>
   Controlled By: ReplicaSet/hamster-c7d89d6db
   Containers:
     hamster:
       Container ID:  docker://
   e76c2413fc720ac395c33b64588c82094fc8e5d590e373d5f818f3978f577e24
       Image:         k8s.gcr.io/ubuntu-slim:0.1
       Image ID:      docker-pullable://k8s.gcr.io/ubuntu-
   slim@sha256:b6f8c3885f5880a4f1a7cf717c07242eb4858fdd5a84b5ffe35b1cf680ea17b1
       Port:          <none>
       Host Port:     <none>
       Command:
         /bin/sh
       Args:
         -c
         while true; do timeout 0.5s yes >/dev/null; sleep 0.5s; done
       State:         Running
         Started:     Fri, 27 Sep 2019 10:35:16 -0700
       Ready:         True
       Restart Count: 0
   ```

```
      Requests:
        cpu:          100m
        memory:       50Mi
...
```

You can see that the original pod reserves 100 millicpu of CPU and 50 Mebibytes of memory. For this example application, 100 millicpu is less than the pod needs to run, so it is CPU-constrained. It also reserves much less memory than it needs. The Vertical Pod Autoscaler `vpa-recommender` deployment analyzes the `hamster` pods to see if the CPU and memory requirements are appropriate. If adjustments are needed, the `vpa-updater` relaunches the pods with updated values.

4. Wait for the `vpa-updater` to launch a new `hamster` pod. This should take a minute or two. You can monitor the pods with the following command.

   Note
   If you are not sure that a new pod has launched, compare the pod names with your previous list. When the new pod launches, you will see a new pod name.

```
kubectl get --watch pods -l app=hamster
```

5. When a new `hamster` pod is started, describe it and view the updated CPU and memory reservations.

```
kubectl describe pod hamster-c7d89d6db-jxgfv
```

Output:

```
Name:          hamster-c7d89d6db-jxgfv
Namespace:     default
Priority:      0
Node:          ip-192-168-9-44.region-code.compute.internal/192.168.9.44
Start Time:    Fri, 27 Sep 2019 10:37:08 -0700
Labels:        app=hamster
               pod-template-hash=c7d89d6db
Annotations:   kubernetes.io/psp: eks.privileged
               vpaUpdates: Pod resources updated by hamster-vpa: container 0: cpu
 request, memory request
Status:        Running
IP:            192.168.3.140
IPs:           <none>
Controlled By: ReplicaSet/hamster-c7d89d6db
Containers:
  hamster:
    Container ID:
 docker://2c3e7b6fb7ce0d8c86444334df654af6fb3fc88aad4c5d710eac3b1e7c58f7db
    Image:         k8s.gcr.io/ubuntu-slim:0.1
    Image ID:      docker-pullable://k8s.gcr.io/ubuntu-
slim@sha256:b6f8c3885f5880a4f1a7cf717c07242eb4858fdd5a84b5ffe35b1cf680ea17b1
    Port:          <none>
    Host Port:     <none>
    Command:
      /bin/sh
    Args:
      -c
      while true; do timeout 0.5s yes >/dev/null; sleep 0.5s; done
    State:         Running
      Started:     Fri, 27 Sep 2019 10:37:08 -0700
    Ready:         True
    Restart Count: 0
    Requests:
      cpu:         587m
      memory:      262144k
```

```
...
```

Here you can see that the CPU reservation has increased to 587 millicpu, which is over five times the original value. The memory has increased to 262,144 Kilobytes, which is around 250 Mebibytes, or five times the original value. This pod was under-resourced, and the Vertical Pod Autoscaler corrected our estimate with a much more appropriate value.

6. Describe the `hamster-vpa` resource to view the new recommendation.

```
kubectl describe vpa/hamster-vpa
```

Output:

```
Name:          hamster-vpa
Namespace:     default
Labels:        <none>
Annotations:   kubectl.kubernetes.io/last-applied-configuration:
                 {"apiVersion":"autoscaling.k8s.io/
v1beta2","kind":"VerticalPodAutoscaler","metadata":{"annotations":{},"name":"hamster-
vpa","namespace":"d...
API Version:   autoscaling.k8s.io/v1beta2
Kind:          VerticalPodAutoscaler
Metadata:
  Creation Timestamp:  2019-09-27T18:22:51Z
  Generation:          23
  Resource Version:    14411
  Self Link:           /apis/autoscaling.k8s.io/v1beta2/namespaces/default/
verticalpodautoscalers/hamster-vpa
  UID:                 d0d85fb9-e153-11e9-ae53-0205785d75b0
Spec:
  Target Ref:
    API Version:  apps/v1
    Kind:         Deployment
    Name:         hamster
Status:
  Conditions:
    Last Transition Time:  2019-09-27T18:23:28Z
    Status:                True
    Type:                  RecommendationProvided
  Recommendation:
    Container Recommendations:
      Container Name:  hamster
      Lower Bound:
        Cpu:      550m
        Memory:   262144k
      Target:
        Cpu:      587m
        Memory:   262144k
      Uncapped Target:
        Cpu:      587m
        Memory:   262144k
      Upper Bound:
        Cpu:      21147m
        Memory:   387863636
Events:          <none>
```

7. When you finish experimenting with the example application, you can delete it with the following command.

```
kubectl delete -f examples/hamster.yaml
```

Load balancing and ingress

This chapter covers the following load balancing and ingress configuration options for Amazon EKS clusters:

- the section called "Load balancing" (p. 146) – Amazon EKS supports the AWS Network Load Balancer and the Classic Load Balancer for pods running on Amazon EC2 instance worker nodes through the Kubernetes `LoadBalancer` service type.
- the section called "ALB Ingress Controller on Amazon EKS" (p. 147) – The AWS ALB Ingress Controller for Kubernetes is a controller that triggers the creation of an AWS Application Load Balancer and the necessary supporting AWS resources whenever an ingress resource is created on a cluster with the `kubernetes.io/ingress.class: alb` annotation.

Load balancing

Amazon EKS supports the Network Load Balancer and the Classic Load Balancer for pods running on Amazon EC2 instance worker nodes through the Kubernetes service of type `LoadBalancer`. Classic Load Balancers and Network Load Balancers are not supported for pods running on AWS Fargate (Fargate). For Fargate ingress, we recommend that you use the ALB Ingress Controller (p. 147) on Amazon EKS (minimum version v1.1.4).

The configuration of your load balancer is controlled by annotations that are added to the manifest for your service. By default, Classic Load Balancers are used for `LoadBalancer` type services. To use the Network Load Balancer instead, apply the following annotation to your service:

```
service.beta.kubernetes.io/aws-load-balancer-type: nlb
```

For an example service manifest that specifies a load balancer, see Type LoadBalancer in the Kubernetes documentation. For more information about using Network Load Balancer with Kubernetes, see Network Load Balancer support on AWS in the Kubernetes documentation.

By default, services of type `LoadBalancer` create public-facing load balancers. To use an internal load balancer, apply the following annotation to your service:

```
service.beta.kubernetes.io/aws-load-balancer-internal: "true"
```

For internal load balancers, your Amazon EKS cluster must be configured to use at least one private subnet in your VPC. Kubernetes examines the route table for your subnets to identify whether they are public or private. Public subnets have a route directly to the internet using an internet gateway, but private subnets do not.

Subnet tagging for load balancers

You must tag the public subnets in your VPC so that Kubernetes knows to use only those subnets for external load balancers instead of choosing a public subnet in each Availability Zone (in lexicographical order by subnet ID). If you use an Amazon EKS AWS CloudFormation template to create your VPC after 03/26/2020, then the subnets created by the template are tagged when they're created. For more information about the Amazon EKS AWS CloudFormation VPC templates, see ??? (p. 152).

Key	Value
`kubernetes.io/role/elb`	1

Private subnets must be tagged in the following way so that Kubernetes knows it can use the subnets for internal load balancers. If you use an Amazon EKS AWS CloudFormation template to create your VPC after 03/26/2020, then the subnets created by the template are tagged when they're created. For more information about the Amazon EKS AWS CloudFormation VPC templates, see .

Key	Value
`kubernetes.io/role/internal-elb`	1

ALB Ingress Controller on Amazon EKS

The AWS ALB Ingress Controller for Kubernetes is a controller that triggers the creation of an Application Load Balancer (ALB) and the necessary supporting AWS resources whenever an Ingress resource is created on the cluster with the `kubernetes.io/ingress.class: alb` annotation. The Ingress resource configures the ALB to route HTTP or HTTPS traffic to different pods within the cluster. The ALB Ingress Controller is supported for production workloads running on Amazon EKS clusters.

To ensure that your ingress objects use the ALB Ingress Controller, add the following annotation to your Ingress specification. For more information, see Ingress specification in the documentation.

```
annotations:
    kubernetes.io/ingress.class: alb
```

The ALB Ingress controller supports the following traffic modes:

- **Instance** – Registers nodes within your cluster as targets for the ALB. Traffic reaching the ALB is routed to `NodePort` for your service and then proxied to your pods. This is the default traffic mode. You can also explicitly specify it with the `alb.ingress.kubernetes.io/target-type: instance` annotation.

    Note
    Your Kubernetes service must specify the `NodePort` type to use this traffic mode.

- **IP** – Registers pods as targets for the ALB. Traffic reaching the ALB is directly routed to pods for your service. You must specify the `alb.ingress.kubernetes.io/target-type: ip` annotation to use this traffic mode.

For other available annotations supported by the ALB Ingress Controller, see Ingress annotations.

This topic shows you how to configure the ALB Ingress Controller to work with your Amazon EKS cluster.

To deploy the ALB Ingress Controller to an Amazon EKS cluster

1. Tag the subnets in your VPC that you want to use for your load balancers so that the ALB Ingress Controller knows that it can use them. For more information, see Subnet tagging requirement (p. 157). If you deployed your cluster with `ekctl`, then the tags are already applied.

    - All subnets in your VPC should be tagged accordingly so that Kubernetes can discover them.

Key	Value
`kubernetes.io/cluster/`*`<cluster-`* *`name>`*	`shared`

- Public subnets in your VPC should be tagged accordingly so that Kubernetes knows to use only those subnets for external load balancers.

Key	Value
`kubernetes.io/role/elb`	1

- Private subnets must be tagged in the following way so that Kubernetes knows it can use the subnets for internal load balancers. If you use an Amazon EKS AWS CloudFormation template to create your VPC after 03/26/2020, then the subnets created by the template are tagged when they're created. For more information about the Amazon EKS AWS CloudFormation VPC templates, see ??? (p. 152).

Key	Value
`kubernetes.io/role/internal-elb`	1

2. Create an IAM OIDC provider and associate it with your cluster. If you don't have `eksctl` version 0.19.0 or later installed, complete the instructions in Installing or upgrading `eksctl` (p. 197) to install or upgrade it. You can check your installed version with `eksctl version`.

```
eksctl utils associate-iam-oidc-provider \
    --region region-code \
    --cluster prod \
    --approve
```

3. Create an IAM policy called `ALBIngressControllerIAMPolicy` for the ALB Ingress Controller pod that allows it to make calls to AWS APIs on your behalf. Use the following AWS CLI command to create the IAM policy in your AWS account. You can view the policy document on GitHub.

```
aws iam create-policy \
    --policy-name ALBIngressControllerIAMPolicy \
    --policy-document https://raw.githubusercontent.com/kubernetes-sigs/aws-alb-
ingress-controller/v1.1.4/docs/examples/iam-policy.json
```

Take note of the policy ARN that is returned.

4. Create a Kubernetes service account named `alb-ingress-controller` in the `kube-system` namespace, a cluster role, and a cluster role binding for the ALB Ingress Controller to use with the following command. If you don't have `kubectl` installed, complete the instructions in Installing kubectl (p. 182) to install it.

```
kubectl apply -f https://raw.githubusercontent.com/kubernetes-sigs/aws-alb-ingress-
controller/v1.1.4/docs/examples/rbac-role.yaml
```

5. Create an IAM role for the ALB Ingress Controller and attach the role to the service account created in the previous step. If you didn't create your cluster with `eksctl`, then use the instructions on the AWS Management Console or AWS CLI tabs.

eksctl

The command that follows only works for clusters that were created with `eksctl`.

```
eksctl create iamserviceaccount \
    --region region-code \
    --name alb-ingress-controller \
    --namespace kube-system \
    --cluster prod \
```

```
    --attach-policy-arn
 arn:aws:iam::111122223333:policy/ALBIngressControllerIAMPolicy \
    --override-existing-serviceaccounts \
    --approve
```

AWS Management Console

1. Using the instructions on the AWS Management Console tab in Create an IAM role (p. 260), create an IAM role named *eks-alb-ingress-controller* and attach the *ALBIngressControllerIAMPolicy* IAM policy that you created in a previous step to it. Note the Amazon Resource Name (ARN) of the role, once you've created it.

2. Annotate the Kubernetes service account with the ARN of the role that you created with the following command.

```
kubectl annotate serviceaccount -n kube-system alb-ingress-controller \
eks.amazonaws.com/role-arn=arn:aws:iam::111122223333:role/eks-alb-ingress-
controller
```

AWS CLI

1. Using the instructions on the AWS CLI tab in Create an IAM role (p. 260), create an IAM role named *eks-alb-ingress-controller* and attach the *ALBIngressControllerIAMPolicy* IAM policy that you created in a previous step to it. Note the Amazon Resource Name (ARN) of the role, once you've created it.

2. Annotate the Kubernetes service account with the ARN of the role that you created with the following command.

```
kubectl annotate serviceaccount -n kube-system alb-ingress-controller \
eks.amazonaws.com/role-arn=arn:aws:iam::111122223333:role/eks-alb-ingress-
controller
```

6. Deploy the ALB Ingress Controller with the following command.

```
kubectl apply -f https://raw.githubusercontent.com/kubernetes-sigs/aws-alb-ingress-
controller/v1.1.4/docs/examples/alb-ingress-controller.yaml
```

7. Open the ALB Ingress Controller deployment manifest for editing with the following command.

```
kubectl edit deployment.apps/alb-ingress-controller -n kube-system
```

8. Add a line for the cluster name after the `--ingress-class=alb` line. If you're running the ALB Ingress Controller on Fargate, then you must also add the lines for the VPC ID, and AWS Region name of your cluster. Once you've added the appropriate lines, save and close the file.

```
    spec:
      containers:
      - args:
        - --ingress-class=alb
        - --cluster-name=prod
        - --aws-vpc-id=vpc-03468a8157edca5bd
        - --aws-region=region-code
```

9. Confirm that the ALB Ingress Controller is running with the following command.

```
kubectl get pods -n kube-system
```

Expected output:

```
NAME                                         READY   STATUS     RESTARTS   AGE
alb-ingress-controller-55b5bbcb5b-bc8q9      1/1     Running    0          56s
```

To deploy a sample application

1. Deploy the game 2048 as a sample application to verify that the ALB Ingress Controller creates an Application Load Balancer as a result of the Ingress object. You can run the sample application on a cluster that has Amazon EC2 worker nodes only, one or more Fargate pods, or a combination of the two. If your cluster has Amazon EC2 worker nodes and no Fargate pods, then select the **Amazon EC2 worker nodes only** tab. If your cluster has any existing Fargate pods, or you want to deploy the application to new Fargate pods, then select the **Fargate** tab. For more information about Fargate pods, see Getting started with AWS Fargate on Amazon EKS (p. 111) .

   Amazon EC2 worker nodes only

   Deploy the application with the following commands.

   ```
   kubectl apply -f https://raw.githubusercontent.com/kubernetes-sigs/aws-alb-ingress-
   controller/v1.1.4/docs/examples/2048/2048-namespace.yaml
   kubectl apply -f https://raw.githubusercontent.com/kubernetes-sigs/aws-alb-ingress-
   controller/v1.1.4/docs/examples/2048/2048-deployment.yaml
   kubectl apply -f https://raw.githubusercontent.com/kubernetes-sigs/aws-alb-ingress-
   controller/v1.1.4/docs/examples/2048/2048-service.yaml
   kubectl apply -f https://raw.githubusercontent.com/kubernetes-sigs/aws-alb-ingress-
   controller/v1.1.4/docs/examples/2048/2048-ingress.yaml
   ```

   Fargate

   Ensure that the cluster that you want to use Fargate in is in the list of supported Regions (p. 110).

   a. Create a Fargate profile that includes the sample application's namespace with the following command. Replace the *example-values* with your own values.

      > **Note**
      > The command that follows only works for clusters that were created with `eksctl`. If you didn't create your cluster with `eksctl`, then you can create the profile with the the AWS Management Console (p. 117), using the same values for `name` and `namespace` that are in the command below.

      ```
      eksctl create fargateprofile --cluster prod --region region-code --name alb-
      sample-app --namespace 2048-game
      ```

   b. Download and apply the manifest files to create the Kubernetes namespace, deployment, and service with the following commands.

      ```
      kubectl apply -f https://raw.githubusercontent.com/kubernetes-sigs/aws-alb-
      ingress-controller/v1.1.4/docs/examples/2048/2048-namespace.yaml
      kubectl apply -f https://raw.githubusercontent.com/kubernetes-sigs/aws-alb-
      ingress-controller/v1.1.4/docs/examples/2048/2048-deployment.yaml
      kubectl apply -f https://raw.githubusercontent.com/kubernetes-sigs/aws-alb-
      ingress-controller/v1.1.4/docs/examples/2048/2048-service.yaml
      ```

   c. Download the ingress manifest file with the following command.

```
curl -o 2048-ingress.yaml https://raw.githubusercontent.com/kubernetes-sigs/aws-
alb-ingress-controller/v1.1.4/docs/examples/2048/2048-ingress.yaml
```

    d. Edit the `2048-ingress.yaml` file. Under the existing `alb.ingress.kubernetes.io/scheme: internet-facing` line , add the line `alb.ingress.kubernetes.io/target-type: ip`.

    e. Apply the ingress manifest file with the following command.

```
kubectl apply -f 2048-ingress.yaml
```

2. After a few minutes, verify that the Ingress resource was created with the following command.

```
kubectl get ingress/2048-ingress -n 2048-game
```

Output:

```
NAME            HOSTS    ADDRESS
         PORTS        AGE
2048-ingress    *        example-2048game-2048ingr-6fa0-352729433.region-
code.elb.amazonaws.com    80       24h
```

> **Note**
> If your Ingress has not been created after several minutes, run the following command to view the Ingress controller logs. These logs may contain error messages that can help you diagnose any issues with your deployment.

```
kubectl logs -n kube-system    deployment.apps/alb-ingress-controller
```

3. Open a browser and navigate to the `ADDRESS` URL from the previous command output to see the sample application.

4. When you finish experimenting with your sample application, delete it with the following commands.

```
kubectl delete -f https://raw.githubusercontent.com/kubernetes-sigs/aws-alb-ingress-
controller/v1.1.4/docs/examples/2048/2048-ingress.yaml
kubectl delete -f https://raw.githubusercontent.com/kubernetes-sigs/aws-alb-ingress-
controller/v1.1.4/docs/examples/2048/2048-service.yaml
kubectl delete -f https://raw.githubusercontent.com/kubernetes-sigs/aws-alb-ingress-
controller/v1.1.4/docs/examples/2048/2048-deployment.yaml
kubectl delete -f https://raw.githubusercontent.com/kubernetes-sigs/aws-alb-ingress-
controller/v1.1.4/docs/examples/2048/2048-namespace.yaml
```

Amazon EKS networking

This chapter covers the following networking topics for running Kubernetes on Amazon EKS.

- the section called "Creating a VPC for Amazon EKS" (p. 153) – An Amazon Virtual Private Cloud (Amazon VPC) enables you to launch AWS resources into a virtual network that you've defined.
- the section called "Cluster VPC considerations" (p. 155) – Learn about requirements for your VPC and subnets.
- the section called "Amazon EKS security group considerations" (p. 157) – Learn about requirements for security groups.
- the section called "Pod networking (CNI)" (p. 160) – Using the CNI plugin allows Kubernetes pods to have the same IP address inside the pod as they do on the VPC network.
- the section called "Installing or upgrading CoreDNS" (p. 175) – CoreDNS is supported on Amazon EKS clusters with Kubernetes version 1.13 or later.
- the section called "Installing Calico on Amazon EKS" (p. 177) – Project Calico is a network policy engine for Kubernetes. With Calico network policy enforcement, you can implement network segmentation and tenant isolation.

Creating a VPC for your Amazon EKS cluster

Amazon Virtual Private Cloud (Amazon VPC) enables you to launch AWS resources into a virtual network that you've defined. This virtual network closely resembles a traditional network that you'd operate in your own data center, with the benefits of using the scalable infrastructure of AWS. For more information, see the Amazon VPC User Guide and De-mystifying cluster networking for Amazon EKS worker nodes.

If you want to use an existing VPC, then it must meet specific requirements for use with Amazon EKS. For more information, see ??? (p. 155). This topic guides you through creating a VPC for your cluster using one of the following configurations:

- **Public and private subnets** – This VPC has two public and two private subnets. One public and one private subnet are deployed to the same Availability Zone. The other public and private subnets are deployed to a second Availability Zone in the same Region. We recommend this option for all production deployments. This option allows you to deploy your worker nodes to private subnets and allows Kubernetes to deploy load balancers to the public subnets that can load balance traffic to pods running on worker nodes in the private subnets.

  Public IP addresses are automatically assigned to resources deployed to one of the public subnets, but public IP addresses are not assigned to any resources deployed to the private subnets. The worker nodes in private subnets can communicate with the cluster and other AWS services, and pods can communicate outbound to the internet through a NAT gateway that is deployed in each Availability Zone. A security group is deployed that denies all inbound traffic and allows all outbound traffic. The subnets are tagged so that Kubernetes is able to deploy load balancers to them. For more information about subnet tagging, see ??? (p. 157). For more information about this type of VPC, see VPC with public and private subnets (NAT).

- **Only public subnets** – This VPC has three public subnets that are deployed into different Availability Zones in the region. All worker nodes are automatically assigned public IP addresses and can send

and receive internet traffic through an internet gateway. A security group is deployed that denies all inbound traffic and allows all outbound traffic. The subnets are tagged so that Kubernetes can deploy load balancers to them. For more information about subnet tagging, see ??? (p. 157). For more information about this type of VPC, see VPC with a single public subnet.

> **Important**
>
> If you deployed a VPC using `eksctl` or by using either of the Amazon EKS AWS CloudFormation VPC templates:
>
> - On or after 03/26/2020 – Public IPv4 addresses are automatically assigned by public subnets to new worker nodes deployed to public subnets.
>
> - Before 03/26/2020 – Public IPv4 addresses are not automatically assigned by public subnets to new worker nodes deployed to public subnets.
>
> This change impacts new node groups deployed to public subnets in the following ways:
>
> - Managed node groups (p. 87) – If the node group is deployed to a public subnet on or after 04/22/2020, the public subnet must have automatic assignment of public IP addresses enabled. For more information, see Modifying the public IPv4 addressing attribute for your subnet.
>
> - Linux (p. 92), Windows (p. 97), or Arm (p. 62) self-managed node groups – If the node group is deployed to a public subnet on or after 03/26/2020, the public subnet must have automatic assignment of public IP addresses enabled or the worker nodes must be launched with a public IP address. For more information, see Modifying the public IPv4 addressing attribute for your subnet or Assigning a public IPv4 address during instance launch.

Choose the tab below that represents your desired VPC configuration.

Creating a VPC for your Amazon EKS cluster

Public and private subnets

To create your cluster VPC with public and private subnets

1. Open the AWS CloudFormation console at https://console.aws.amazon.com/cloudformation.
2. From the navigation bar, select a Region that supports Amazon EKS.
3. Choose **Create stack**.
4. For **Choose a template**, select **Specify an Amazon S3 template URL**.
5. Paste the following URL into the text area and choose **Next**:

   ```
   https://amazon-eks.s3.us-west-2.amazonaws.com/cloudformation/2020-05-08/amazon-eks-
   vpc-private-subnets.yaml
   ```

6. On the **Specify Details** page, fill out the parameters accordingly, and then choose **Next**.

   - **Stack name**: Choose a stack name for your AWS CloudFormation stack. For example, you can call it **eks-vpc**.
   - **VpcBlock**: Choose a CIDR range for your VPC. You can keep the default value.
   - **PublicSubnet01Block**: Specify a CIDR range for public subnet 1. We recommend that you keep the default value so that you have plenty of IP addresses for load balancers to use.
   - **PublicSubnet02Block**: Specify a CIDR range for public subnet 2. We recommend that you keep the default value so that you have plenty of IP addresses for load balancers to use.

- **PrivateSubnet01Block**: Specify a CIDR range for private subnet 1. We recommend that you keep the default value so that you have plenty of IP addresses for pods and load balancers to use.

- **PrivateSubnet02Block**: Specify a CIDR range for private subnet 2. We recommend that you keep the default value so that you have plenty of IP addresses for pods and load balancers to use.

7. (Optional) On the **Options** page, tag your stack resources. Choose **Next**.

8. On the **Review** page, choose **Create**.

9. When your stack is created, select it in the console and choose **Outputs**.

10. Record the **SecurityGroups** value for the security group that was created. When you add worker nodes to your cluster, you must specify the ID of the security group. The security group is applied to the cross-account elastic network interfaces that are created in your subnets that allow the Amazon EKS control plane to communicate with your worker nodes.

11. Record the **VpcId** for the VPC that was created. You need this when you launch your worker node group template.

12. Record the **SubnetIds** for the subnets that were created and whether you created them as public or private subnets. When you add worker nodes to your cluster, you must specify the IDs of the subnets that you want to launch the worker nodes into.

Only public subnets

To create your cluster VPC with only public subnets

1. Open the AWS CloudFormation console at https://console.aws.amazon.com/cloudformation.

2. From the navigation bar, select a Region that supports Amazon EKS.

3. Choose **Create stack**.

4. For **Choose a template**, select **Specify an Amazon S3 template URL**.

5. Paste the following URL into the text area and choose **Next**:

```
https://amazon-eks.s3.us-west-2.amazonaws.com/cloudformation/2020-05-08/amazon-eks-
vpc-sample.yaml
```

6. On the **Specify Details** page, fill out the parameters accordingly, and then choose **Next**.

- **Stack name**: Choose a stack name for your AWS CloudFormation stack. For example, you can call it **eks-vpc**.

- **VpcBlock**: Choose a CIDR range for your VPC. You can keep the default value.

- **Subnet01Block**: Specify a CIDR range for subnet 1. We recommend that you keep the default value so that you have plenty of IP addresses for pods and load balancers to use.

- **Subnet02Block**: Specify a CIDR range for subnet 2. We recommend that you keep the default value so that you have plenty of IP addresses for pods and load balancers to use.

- **Subnet03Block**: Specify a CIDR range for subnet 3. We recommend that you keep the default value so that you have plenty of IP addresses for pods and load balancers to use.

7. (Optional) On the **Options** page, tag your stack resources. Choose **Next**.

8. On the **Review** page, choose **Create**.

9. When your stack is created, select it in the console and choose **Outputs**.

10. Record the **SecurityGroups** value for the security group that was created. When you add worker nodes to your cluster, you must specify the ID of the security group. The security group is applied to the cross-account elastic network interfaces that are created in your subnets that allow the Amazon EKS control plane to communicate with your worker nodes.

11. Record the **VpcId** for the VPC that was created. You need this when you launch your worker node group template.

12. Record the **SubnetIds** for the subnets that were created. When you add worker nodes to your cluster, you must specify the IDs of the subnets that you want to launch the worker nodes into.

Next steps

After you have created your VPC, you can try the Getting started with Amazon EKS (p. 3) walkthrough, but you can skip the Create your Amazon EKS cluster VPC (p. 11) section and use these subnets and security groups for your cluster.

Cluster VPC considerations

When you create an Amazon EKS cluster, you specify the VPC subnets for your cluster to use. Amazon EKS requires subnets in at least two Availability Zones. We recommend a VPC with public and private subnets so that Kubernetes can create public load balancers in the public subnets that load balance traffic to pods running on worker nodes that are in private subnets.

When you create your cluster, specify all of the subnets that will host resources for your cluster (such as worker nodes and load balancers).

> **Note**
> Internet-facing load balancers require a public subnet in your cluster. Worker nodes also require outbound internet access to the Amazon EKS APIs for cluster introspection and node registration at launch time. To pull container images, they require access to the Amazon S3 and Amazon ECR APIs (and any other container registries, such as DockerHub). For more information, see Amazon EKS security group considerations (p. 157) and AWS IP Address Ranges in the *AWS General Reference*.

The subnets that you pass when you create the cluster influence where Amazon EKS places elastic network interfaces that are used for the control plane to worker node communication.

It is possible to specify only public or private subnets when you create your cluster, but there are some limitations associated with these configurations:

- **Private-only**: Everything runs in a private subnet and Kubernetes cannot create internet-facing load balancers for your pods.
- **Public-only**: Everything runs in a public subnet, including your worker nodes.

Amazon EKS creates an elastic network interface in your private subnets to facilitate communication to your worker nodes. This communication channel supports Kubernetes functionality such as **kubectl exec** and **kubectl logs**. The security group that you specify when you create your cluster is applied to the elastic network interfaces that are created for your cluster control plane.

Your VPC must have DNS hostname and DNS resolution support. Otherwise, your worker nodes cannot register with your cluster. For more information, see Using DNS with Your VPC in the *Amazon VPC User Guide*.

VPC IP addressing

Your worker nodes must be able to access the internet using a public IP address to function properly. If your worker nodes are deployed in a private subnet, then the subnet must have a default route to a

NAT gateway. The NAT gateway must be assigned a public IP address to provide internet access for the worker nodes. If self-managed worker nodes are deployed to a public subnet, then the subnet must be configured to auto-assign public IP addresses or your worker node instances must be assigned a public IP address when they're launched. If managed worker nodes are deployed to a public subnet, then the subnet must be configured to auto-assign public IP addresses or the worker nodes will not be assigned a public IP address. Determine whether your public subnets are configured to auto-assign public IP addresses with the following command.

```
aws ec2 describe-subnets \
    --filters "Name=vpc-id,Values=VPC-ID" | grep 'SubnetId\|MapPublicIpOnLaunch'
```

Output

```
"MapPublicIpOnLaunch": false,
"SubnetId": "subnet-aaaaaaaaaaaaaaaaa",
"MapPublicIpOnLaunch": false,
"SubnetId": "subnet-bbbbbbbbbbbbbbbbb",
```

For any subnets that have `MapPublicIpOnLaunch` set to `false`, change the setting to `true`.

```
aws ec2 modify-subnet-attribute --map-public-ip-on-launch --subnet-id subnet-aaaaaaaaaaaaaaaaa
```

> **Important**
> If you used an Amazon EKS AWS CloudFormation template (p. 152) to deploy your VPC prior to 03/26/2020, then you need to change the setting for your public subnets.

You can define both private (RFC 1918) and public (non-RFC 1918) CIDR ranges within the VPC used for your Amazon EKS cluster. For more information, see VPCs and Subnets and IP Addressing in Your VPC in the *Amazon VPC User Guide*.

The Amazon EKS control plane creates up to 4 cross-account elastic network interfaces in your VPC for each cluster. Be sure that the subnets you specify have enough available IP addresses for the cross-account elastic network interfaces and your pods.

> **Important**
> Docker runs in the `172.17.0.0/16` CIDR range in Amazon EKS clusters. We recommend that your cluster's VPC subnets do not overlap this range. Otherwise, you will receive the following error:
>
> ```
> Error: : error upgrading connection: error dialing backend: dial tcp
>   172.17.nn.nn:10250: getsockopt: no route to host
> ```

VPC tagging requirement

When you create an Amazon EKS cluster earlier than version 1.15, Amazon EKS tags the VPC containing the subnets you specify in the following way so that Kubernetes can discover it:

Key	Value
kubernetes.io/cluster/<cluster-name>	shared

- **Key**: The *<cluster-name>* value matches your Amazon EKS cluster's name.

- **Value**: The `shared` value allows more than one cluster to use this VPC.

This tag is not required or created by Amazon EKS for 1.15 clusters. If you deploy a 1.15 cluster to a VPC that already has this tag, the tag is not removed.

Subnet tagging requirement

When you create your Amazon EKS cluster, Amazon EKS tags the subnets you specify in the following way so that Kubernetes can discover them:

> **Note**
> All subnets (public and private) that your cluster uses for resources should have this tag.

Key	Value
`kubernetes.io/cluster/`*`<cluster-name>`*	`shared`

- **Key**: The *`<cluster-name>`* value matches your Amazon EKS cluster.
- **Value**: The `shared` value allows more than one cluster to use this subnet.

Private subnet tagging requirement for internal load balancers

Private subnets must be tagged in the following way so that Kubernetes knows it can use the subnets for internal load balancers. If you use an Amazon EKS AWS CloudFormation template to create your VPC after 03/26/2020, then the subnets created by the template are tagged when they're created. For more information about the Amazon EKS AWS CloudFormation VPC templates, see .

Key	Value
`kubernetes.io/role/internal-elb`	`1`

Public subnet tagging option for external load balancers

You must tag the public subnets in your VPC so that Kubernetes knows to use only those subnets for external load balancers instead of choosing a public subnet in each Availability Zone (in lexicographical order by subnet ID). If you use an Amazon EKS AWS CloudFormation template to create your VPC after 03/26/2020, then the subnets created by the template are tagged when they're created. For more information about the Amazon EKS AWS CloudFormation VPC templates, see .

Key	Value
`kubernetes.io/role/elb`	`1`

Amazon EKS security group considerations

The following sections describe the minimum required and recommended security group settings for the cluster, control plane, and worker node security groups for your cluster, depending on your Kubernetes version and Amazon EKS platform version.

Amazon EKS User Guide
Cluster security group (available starting
with Amazon EKS clusters running
Kubernetes 1.14 and `eks.3` platform version)

Cluster security group (available starting with Amazon EKS clusters running Kubernetes 1.14 and `eks.3` platform version)

Amazon EKS clusters beginning with Kubernetes version 1.14 and platform version (p. 49) `eks.3` create a cluster security group as part of cluster creation (or when a cluster is upgraded to this Kubernetes version and platform version). This security group is designed to allow all traffic from the control plane and managed node groups (p. 85) to flow freely between each other. By assigning the cluster security group to the control plane cross-account elastic network interfaces and the managed node group instances, you do not need to configure complex security group rules to allow this communication. Any instance or network interface that is assigned this security group can freely communicate with other resources with this security group.

You can check for a cluster security group for your cluster in the AWS Management Console under the cluster's **Networking** section, or with the following AWS CLI command:

```
aws eks describe-cluster --name cluster_name --query
 cluster.resourcesVpcConfig.clusterSecurityGroupId
```

If your cluster is running Kubernetes version 1.14 and platform version (p. 49) `eks.3` or later, we recommend that you add the cluster security group to all existing and future worker node groups. For more information, see Security Groups for Your VPC in the *Amazon VPC User Guide*. Amazon EKS managed node groups (p. 85) are automatically configured to use the cluster security group.

	Protocol	Port range	Source	Destination
Recommended inbound traffic	All	All	Self	
Recommended outbound traffic	All	All		0.0.0.0/0

Control plane and worker node security groups (for Amazon EKS clusters earlier than Kubernetes version 1.14 and platform version (p. 49) `eks.3`)

For Amazon EKS clusters earlier than Kubernetes version 1.14 and platform version (p. 49) `eks.3`, control plane to worker node communication is configured by manually creating a control plane security group and specifying that security group when you create the cluster. At cluster creation, this security group is then attached to the cross-account elastic network interfaces for the cluster.

You can check the control plane security group for your cluster in the AWS Management Console under the cluster's **Networking** section (listed as **Additional security groups**), or with the following AWS CLI command:

```
aws eks describe-cluster --name cluster_name --query
 cluster.resourcesVpcConfig.securityGroupIds
```

If you launch worker nodes with the AWS CloudFormation template in the Getting started with Amazon EKS (p. 3) walkthrough, AWS CloudFormation modifies the control plane security group to allow

Amazon EKS User Guide
Control plane and worker node security groups
(for Amazon EKS clusters earlier than Kubernetes
version 1.14 and platform version `eks.3`)

communication with the worker nodes. **Amazon EKS strongly recommends that you use a dedicated security group for each control plane (one per cluster)**. If you share a control plane security group with other Amazon EKS clusters or resources, you may block or disrupt connections to those resources.

The security group for the worker nodes and the security group for the control plane communication to the worker nodes have been set up to prevent communication to privileged ports in the worker nodes. If your applications require added inbound or outbound access from the control plane or worker nodes, you must add these rules to the security groups associated with your cluster. For more information, see Security Groups for Your VPC in the *Amazon VPC User Guide*.

> **Note**
> To allow proxy functionality on privileged ports or to run the CNCF conformance tests yourself, you must edit the security groups for your control plane and the worker nodes. The security group on the worker nodes' side needs to allow inbound access for ports 0-65535 from the control plane, and the control plane side needs to allow outbound access to the worker nodes on ports 0-65535.

Control Plane Security Group

	Protocol	Port range	Source	Destination
Minimum inbound traffic	TCP	443	All worker node security groups **When cluster endpoint private access (p. 35) is enabled:** Any security groups that generate API server client traffic (such as `kubectl` commands on a bastion host within your cluster's VPC)	
Recommended inbound traffic	TCP	443	All worker node security groups **When cluster endpoint private access (p. 35) is enabled:** Any security groups that generate API server client traffic (such as `kubectl` commands on a bastion host within your cluster's VPC)	
Minimum outbound traffic	TCP	10250		All worker node security groups
Recommended outbound traffic	TCP	1025-65535		All worker node security groups

Worker Node Security Group

	Protocol	Port range	Source	Destination
Minimum inbound traffic (from other worker nodes)	Any protocol you expect your worker nodes to use for inter-worker communication	Any ports you expect your worker nodes to use for inter-worker communication	All worker node security groups	
Minimum inbound traffic (from control plane)	TCP	10250	Control plane security group	
Recommended inbound traffic	All TCP	All 443, 1025-65535	All worker node security groups Control plane security group	
Minimum outbound traffic*	TCP	443		Control plane security group
Recommended outbound traffic	All	All		0.0.0.0/0

* Worker nodes also require outbound internet access to the Amazon EKS APIs for cluster introspection and node registration at launch time. To pull container images, they require access to the Amazon S3 and Amazon ECR APIs (and any other container registries, such as DockerHub). For more information, see AWS IP Address Ranges in the *AWS General Reference*.

If you have more than one security group associated to your worker nodes, then one of the security groups must have the following tag applied to it. If you have only one security group associated to your worker nodes, then the tag is optional. For more information about tagging, see Working with tags using the console (p. 275).

Key	Value
`kubernetes.io/cluster/`*`<cluster-name>`*	`owned`

Pod networking (CNI)

Amazon EKS supports native VPC networking via the Amazon VPC Container Network Interface (CNI) plugin for Kubernetes. Using this CNI plugin allows Kubernetes pods to have the same IP address inside the pod as they do on the VPC network. This CNI plugin is an open-source project that is maintained on GitHub.
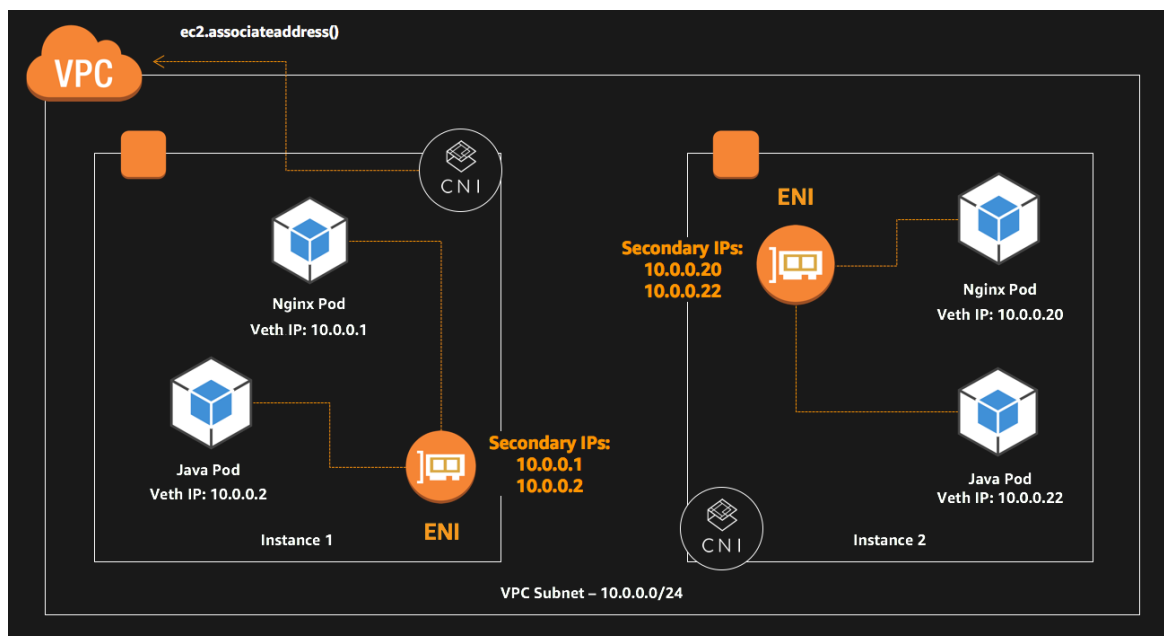
The CNI plugin is responsible for allocating VPC IP addresses to Kubernetes nodes and configuring the necessary networking for pods on each node. The plugin consists of two primary components:

- The L-IPAM daemon is responsible for attaching elastic network interfaces to instances, assigning secondary IP addresses to elastic network interfaces, and maintaining a "warm pool" of IP addresses on each node for assignment to Kubernetes pods when they are scheduled.
- The CNI plugin itself is responsible for wiring the host network (for example, configuring the interfaces and virtual Ethernet pairs) and adding the correct interface to the pod namespace.

For more information about the design and networking configuration, see CNI plugin for Kubernetes networking over AWS VPC.

Elastic network interface and secondary IP address limitations by Amazon EC2 instance types are applicable. In general, larger instances can support more IP addresses. For more information, see IP Addresses Per Network Interface Per Instance Type in the *Amazon EC2 User Guide for Linux Instances*.

Topics

CNI configuration variables

The Amazon VPC CNI plugin for Kubernetes supports a number of configuration options, which are set through environment variables. The following environment variables are available, and all of them are optional.

AWS_VPC_CNI_NODE_PORT_SUPPORT

> **Type** – Boolean

Default – `true`

Specifies whether `NodePort` services are enabled on a worker node's primary network interface. This requires additional `iptables` rules and that the kernel's reverse path filter on the primary interface is set to `loose`.

AWS_VPC_K8S_CNI_CUSTOM_NETWORK_CFG

Type – Boolean

Default – `false`

Specifies that your pods may use subnets and security groups, within the same VPC as your control plane resources, that are independent of your cluster's `resourcesVpcConfig`. By default, pods share the same subnet and security groups as the worker node's primary interface. Setting this variable to `true` causes `ipamD` to use the security groups and subnets in a worker node's `ENIConfig` for elastic network interface allocation. You must create an `ENIConfig` custom resource definition for each subnet that your pods will reside in, and then annotate each worker node to use a specific `ENIConfig` (multiple worker nodes can be annotated with the same `ENIConfig`). Worker nodes can only be annotated with a single `ENIConfig` at a time, and the subnet in the `ENIConfig` must belong to the same Availability Zone that the worker node resides in. For more information, see CNI custom networking (p. 168).

ENI_CONFIG_ANNOTATION_DEF

Type – String

Default – `k8s.amazonaws.com/eniConfig`

Specifies a node annotation key name. This should be used when `AWS_VPC_K8S_CNI_CUSTOM_NETWORK_CFG=true`. The annotation value will be used to set `ENIConfig` name. Annotations take precedence over labels.

ENI_CONFIG_LABEL_DEF

Type – String

Default – `k8s.amazonaws.com/eniConfig`

Specifies a node label key name. This should be used when `AWS_VPC_K8S_CNI_CUSTOM_NETWORK_CFG=true`. The label value will be used to set `ENIConfig` name. Annotations will take precedence over labels. To use labels, ensure that an annotation with the key `k8s.amazonaws.com/eniConfig` is defined and that a value for the annotation key `ENI_CONFIG_ANNOTATION_DEF` is not set on the node. To select an `ENIConfig` based upon Availability Zone, set this to `failure-domain.beta.kubernetes.io/zone` and create an `ENIConfig` custom resource for each Availability Zone, such as `us-east-1a`. For more information, see CNI custom networking (p. 168).

AWS_VPC_ENI_MTU – v1.6.0 and later

Type – Integer

Default – `9001`

Used to configure the MTU size for attached ENIs. The valid range is from `576` to `9001`.

AWS_VPC_K8S_CNI_EXTERNALSNAT

Type – Boolean

Default – `false`

Specifies whether an external NAT gateway should be used to provide SNAT of secondary ENI IP addresses. If set to `true`, the SNAT `iptables` rule and off-VPC IP rule are not applied, and these rules are removed if they have already been applied.

Disable SNAT if you need to allow inbound communication to your pods from external VPNs, direct connections, and external VPCs, and your pods do not need to access the internet directly via an Internet Gateway. Your nodes must be running in a private subnet and connected to the internet through an AWS NAT Gateway or another external NAT device.

For more information, see External source network address translation (SNAT) (p. 167).

`AWS_VPC_K8S_CNI_RANDOMIZESNAT`

Type – String

Default – `hashrandom`

Valid values – `hashrandom, prng, none`

Specifies whether the SNAT `iptables` rule should randomize the outgoing ports for connections. This should be used when `AWS_VPC_K8S_CNI_EXTERNALSNAT=false`. When enabled (`hashrandom`) the `--random` flag will be added to the SNAT `iptables` rule. To use a pseudo random number generation, rather than hash-based (`--random-fully`), use `prng` for the environment variable. For old versions of `iptables` that do not support `--random-fully`, this option will fall back to `--random`. Disable (`none`) this functionality if you rely on sequential port allocation for outgoing connections.

> **Note**
> Any options other than `none` will cause outbound connections to be assigned a source port that's not necessarily part of the ephemeral port range set at the OS level (`/proc/sys/net/ipv4/ip_local_port_range`). This is relevant if you have NACLs restricting traffic based on the port range found in `ip_local_port_range`.

`AWS_VPC_K8S_CNI_EXCLUDE_SNAT_CIDRS` – v1.6.0 and later

Type – String

Default – empty

Specify a comma-separated list of IPv4 CIDRs to exclude from SNAT. For every item in the list, an `iptables` rule and off-VPC IP rule will be applied. If an item is not a valid IPv4 range it will be skipped. This should be used when `AWS_VPC_K8S_CNI_EXTERNALSNAT=false`.

`WARM_ENI_TARGET`

Type – Integer

Default – `1`

Specifies the number of free elastic network interfaces (and all of their available IP addresses) that the `ipamD` daemon should attempt to keep available for pod assignment on the node. By default, `ipamD` attempts to keep one elastic network interface and all of its IP addresses available for pod assignment.

> **Note**
> The number of IP addresses per network interface varies by instance type. For more information, see IP addresses per network interface per instance type in the *Amazon EC2 User Guide for Linux Instances*.

For example, an `m4.4xlarge` launches with one network interface and 30 IP addresses. If five pods are placed on the node and five free IP addresses are removed from the IP address warm pool, then `ipamD` attempts to allocate more interfaces until `WARM_ENI_TARGET` free interfaces are available on the node.

> **Note**
> If `WARM_IP_TARGET` is set, then this environment variable is ignored and the `WARM_IP_TARGET` behavior is used instead.

`WARM_IP_TARGET`

Type – Integer

Default – None

Specifies the number of free IP addresses that the `ipamD` daemon should attempt to keep available for pod assignment on the node. For example, if `WARM_IP_TARGET` is set to 10, then `ipamD` attempts to keep 10 free IP addresses available at all times. If the elastic network interfaces on the node are unable to provide these free addresses, `ipamD` attempts to allocate more interfaces until `WARM_IP_TARGET` free IP addresses are available.

> **Note**
> This environment variable overrides `WARM_ENI_TARGET` behavior.

`MINIMUM_IP_TARGET` – v1.6.0 and later

Type – Integer

Default – None

Specifies the number of total IP addresses that the `ipamD` daemon should attempt to allocate for pod assignment on the node. `MINIMUM_IP_TARGET` behaves identically to `WARM_IP_TARGET`, except that instead of setting a target number of free IP addresses to keep available at all times, it sets a target number for a floor on how many total IP addresses are allocated.

`MINIMUM_IP_TARGET` is for pre-scaling and `WARM_IP_TARGET` is for dynamic scaling. For example, suppose a cluster has an expected pod density of approximately 30 pods per node. If `WARM_IP_TARGET` is set to 30 to ensure there are enough IPs initially allocated by the CNI, then 30 pods are deployed to the node. The CNI will allocate an additional 30 IPs, for a total of 60, accelerating IP exhaustion in the relevant subnets. If instead, `MINIMUM_IP_TARGET` is set to 30 and `WARM_IP_TARGET` to 2, after the 30 pods are deployed, the CNI would allocate an additional 2 IPs. This still provides elasticity, but uses approximately half as many IPs as using `WARM_IP_TARGET` alone (32 IP addresses versus 60 IP addresses).

This also improves reliability of the cluster by reducing the number of calls necessary to allocate or deallocate private IP addresses, which may be throttled, especially at scaling-related times.

`MAX_ENI`

Type – Integer

Default – None

Specifies the maximum number of ENIs that will be attached to the node. When `MAX_ENI` is unset or less than or equal to `0`, the setting is not used, and the maximum number of ENIs is always equal to the maximum number for the instance type in question. Even when `MAX_ENI` is a positive number, it is limited by the maximum number for the instance type.

`AWS_VPC_K8S_CNI_LOGLEVEL`

Type – String

Default – DEBUG

Valid values – `DEBUG`, `INFO`, `WARN`, `ERROR`, or `FATAL` (not case sensitive)

Specifies the loglevel for `ipamd`.

`AWS_VPC_K8S_CNI_LOG_FILE`

Type – String

Default – Unset

Valid values: `stdout` or a file path

Specifies where to write the logging output of `ipamd`. You can specify `stdout` or override the default file, such as `/var/log/aws-routed-eni/ipamd.log`.

AWS_VPC_K8S_PLUGIN_LOG_FILE

    Type – String

    Default – Unset

    Valid values – `stdout` or a file path.

    Specifies where to write the logging output for the `aws-cni` plugin. You can specify `stdout` or override the default file, such as `/var/log/aws-routed-eni/plugin.log`.

AWS_VPC_K8S_PLUGIN_LOG_LEVEL

    Type – String

    Default – DEBUG

    Valid values – `DEBUG`, `INFO`, `WARN`, `ERROR`, or `FATAL` (not case sensitive)

    Specifies the log level for the `aws-cni` plugin.

INTROSPECTION_BIND_ADDRESS

    Type – String

    Default – `127.0.0.1:61679`

    Specifies the bind address for the introspection endpoint. A Unix domain socket can be specified with the `unix:` prefix before the socket path.

DISABLE_INTROSPECTION

    Type – Boolean

    Default – false

    Specifies whether introspection endpoints are disabled on a worker node. Setting this to `true` will reduce the debugging information you can get from the node when running the `aws-cni-support.sh` script.

DISABLE_METRICS

    Type – Boolean

    Default – false

    Specifies whether the Prometheus metrics endpoint is disabled or not for `ipamd`. By default metrics are published on `:61678/metrics`.

AWS_VPC_K8S_CNI_VETHPREFIX

    Type – String

    Default – `eni`

    Specifies the `veth` prefix used to generate the host-side `veth` device name for the CNI. The prefix can be a maximum of four characters long.

ADDITIONAL_ENI_TAGS – v1.6.0 and later

> **Type** – String

> **Default** – { }

> **Example values** – {"tag_key": "tag_val"}

> Metadata applied to ENIs help you categorize and organize your resources for billing or other purposes. Each tag consists of a custom-defined key and an optional value. Tag keys can have a maximum character length of 128 characters. Tag values can have a maximum length of 256 characters. The tags will be added to all ENIs on the host.

> > **Important**
> > Custom tags should not contain the `k8s.amazonaws.com` prefix, because it is reserved. If the tag contains `k8s.amazonaws.com`, the tag addition will be ignored.

CLUSTER_NAME

> **Type** – String

> **Default** – ""

> Specifies the cluster name to tag allocated ENIs with.

> ### ENI tags related to allocation

> This plugin interacts with the following tags on ENIs:

> - `cluster.k8s.amazonaws.com/name`
> - `node.k8s.amazonaws.com/instance_id`
> - `node.k8s.amazonaws.com/no_manage`

> #### Cluster name tag

> The tag `cluster.k8s.amazonaws.com/name` will be set to the cluster name of the `aws-node` daemonset which created the ENI.

> #### Instance ID tag

> The tag `node.k8s.amazonaws.com/instance_id` will be set to the instance ID of the `aws-node` instance that allocated this ENI.

> #### No manage tag

> The `node.k8s.amazonaws.com/no_manage` tag is read by the `aws-node` daemonset to determine whether an ENI attached to the machine should not be configured or used for private IP addresses. This tag is not set by the CNI plugin itself, but rather may be set by a user to indicate that an ENI is intended for host networking pods, or for some other process unrelated to Kubernetes.

> > **Note**
> > Attaching an ENI with the `no_manage` tag will result in an incorrect value for the `kubelet`'s `--max-pods` configuration option. Consider also updating the `MAX_ENI` and `--max-pods` configuration options on this plugin and the `kubelet`, respectively, if you are using of this tag.

Notes

The `L-IPAMD` (`aws-node` daemonSet) running on every worker node requires access to the Kubernetes API server. If it can not reach the Kubernetes API server, `ipamD` will exit and the CNI will not be able to get any IP addresses for pods. To confirm whether `L-IPAMD` has access to the Kubernetes API server.

```
kubectl get svc kubernetes
```

Output

```
NAME         TYPE         CLUSTER-IP     EXTERNAL-IP    PORT(S)    AGE
kubernetes   ClusterIP    10.0.0.1       <none>         443/TCP    29d
```

SSH into a worker node to check whether the worker node can reach the API server.

```
telnet 10.0.0.1 443
```

Output

```
Trying 10.0.0.1...
Connected to 10.0.0.1.
Escape character is '^]'.
```

If you receive the last line of output, then the Kubernetes API server is reachable.

External source network address translation (SNAT)

Communication within a VPC (such as pod to pod) is direct between private IP addresses and requires no source network address translation (SNAT). When traffic is destined for an address outside of the VPC, the Amazon VPC CNI plugin for Kubernetes translates the private IP address of each pod to the primary private IP address assigned to the primary elastic network interface (network interface) of the Amazon EC2 instance worker node that the pod is running on, by default. SNAT:

- Enables pods to communicate bi-directionally with the internet. The worker node must be in a public subnet and have a public or elastic IP address assigned to the primary private IP address of its primary network interface. The traffic is translated to and from the public or elastic IP address and routed to and from the internet by an internet gateway, as shown in the following picture.
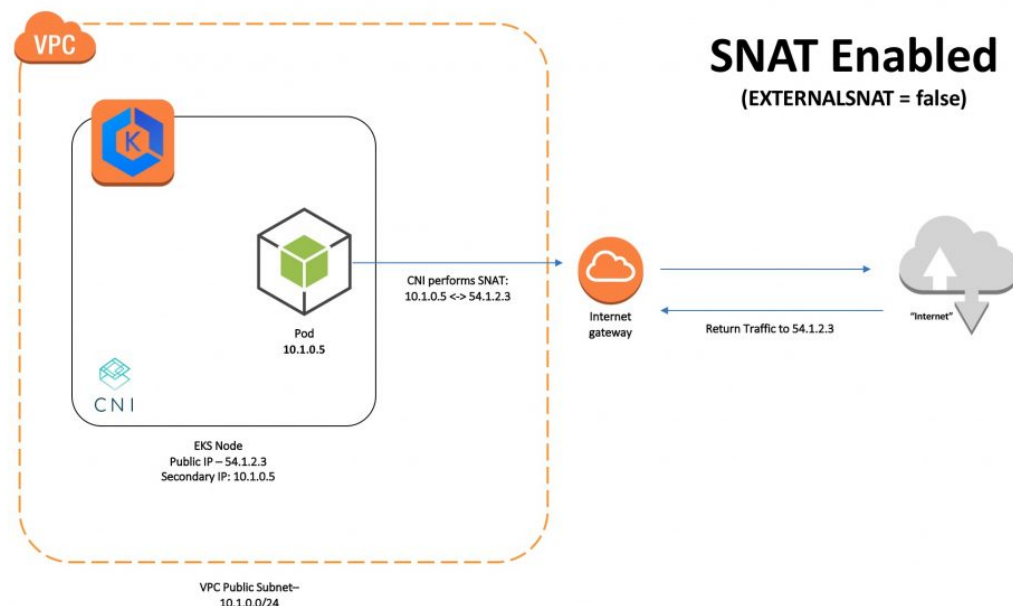
SNAT is necessary because the internet gateway only knows how to translate between the primary private and public or elastic IP address assigned to the primary elastic network interface of the Amazon EC2 instance worker node that pods are running on.

- Prevents a device in other private IP address spaces (for example, VPC peering, Transit VPC, or Direct Connect) from communicating directly to a pod that is not assigned the primary private IP address of the primary elastic network interface of the Amazon EC2 instance worker node.

If the internet or devices in other private IP address spaces need to communicate with a pod that isn't assigned the primary private IP address assigned to the primary elastic network interface of the Amazon EC2 instance worker node that the pod is running on, then:

- The worker node must be deployed in a private subnet that has a route to a NAT device in a public subnet.
- You need to enable external SNAT in the CNI plugin `aws-node` DaemonSet with the following command:

```
kubectl set env daemonset -n kube-system aws-node AWS_VPC_K8S_CNI_EXTERNALSNAT=true
```

Once external SNAT is enabled, the CNI plugin does not translate a pod's private IP address to the primary private IP address assigned to the primary elastic network interface of the Amazon EC2 instance worker node that the pod is running on when traffic is destined for an adddress outside of the VPC. Traffic from the pod to the internet is externally translated to and from the public IP address of the NAT device and routed to and from the internet by an internet gateway, as shown in the following picture.
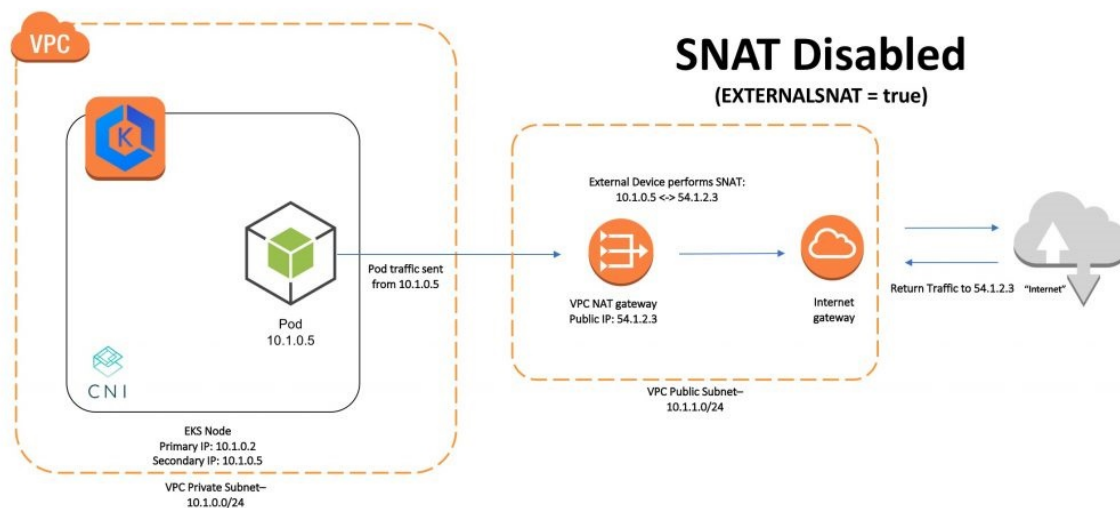
CNI custom networking

By default, when new network interfaces are allocated for pods, ipamD uses the worker node's primary elastic network interface's (ENI) security groups and subnet. However, there are use cases where your pod network interfaces should use a different security group or subnet, within the same VPC as your control plane security group. For example:

- There are a limited number of IP addresses available in a subnet. This limits the number of pods that can be created in the cluster. Using different subnets for pod groups allows you to increase the number of available IP addresses.

- For security reasons, your pods must use different security groups or subnets than the node's primary network interface.
- The worker nodes are configured in public subnets and you want the pods to be placed in private subnets using a NAT Gateway. For more information, see External source network address translation (SNAT) (p. 167).

> **Note**
> You can configure custom networking for self-managed node groups, but not for managed node groups or Fargate. The use cases discussed in this topic require the Amazon VPC CNI plugin for Kubernetes version 1.4.0 or later. To check your CNI version, and upgrade if necessary, see Amazon VPC CNI plugin for Kubernetes upgrades (p. 174).

Enabling a custom network effectively removes an available elastic network interface (and all of its available IP addresses for pods) from each worker node that uses it. The primary network interface for the worker node is not used for pod placement when a custom network is enabled.

To configure CNI custom networking

1. Associate a secondary CIDR block to your cluster's VPC. For more information, see Associating a Secondary IPv4 CIDR Block with Your VPC in the *Amazon VPC User Guide*.

2. Create a subnet in your VPC for each Availability Zone, using your secondary CIDR block. Your custom subnets must be from a different VPC CIDR block than the subnet that your worker nodes were launched into. For more information, see Creating a Subnet in Your VPC in the *Amazon VPC User Guide*.

3. Set the `AWS_VPC_K8S_CNI_CUSTOM_NETWORK_CFG=true` environment variable to `true` in the `aws-node` DaemonSet:

```
kubectl set env daemonset aws-node -n kube-system
 AWS_VPC_K8S_CNI_CUSTOM_NETWORK_CFG=true
```

4. View the currently-installed CNI version.

```
kubectl describe daemonset aws-node --namespace kube-system | grep Image | cut -d "/" -
f 2
```

Output:

```
amazon-k8s-cni:1.5.7
```

5. If you have version 1.3 or later of the CNI installed, you can skip to step 6. Define a new `ENIConfig` custom resource for your cluster.

    a. Create a file called `ENIConfig.yaml` and paste the following content into it:

```
apiVersion: apiextensions.k8s.io/v1beta1
kind: CustomResourceDefinition
metadata:
  name: eniconfigs.crd.k8s.amazonaws.com
spec:
  scope: Cluster
  group: crd.k8s.amazonaws.com
  version: v1alpha1
  names:
    plural: eniconfigs
    singular: eniconfig
    kind: ENIConfig
```

b.   Apply the file to your cluster with the following command:

```
kubectl apply -f ENIConfig.yaml
```

6.   Create an `ENIConfig` custom resource for each subnet that you want to schedule pods in.

a.   Create a unique file for each elastic network interface configuration. Each file must include the contents below with a unique value for `name`. We highly recommend using a value for `name` that matches the Availability Zone of the subnet, as this makes deployment of multi-AZ Auto Scaling groups simpler (see step 6c below). In this example, a file named `us-west-2a.yaml` is created. Replace the *example values* for `name`, `subnet`, and `securityGroups` with your own values. In this example, we follow best practices and set the value for `name` to the Availability Zone that the subnet is in. If you don't have a specific security group that you want to attach for your pods, you can leave that value empty for now. Later, you will specify the worker node security group in the ENIConfig.

> **Note**
> Each subnet and security group combination requires its own custom resource.

```
apiVersion: crd.k8s.amazonaws.com/v1alpha1
kind: ENIConfig
metadata:
  name: us-west-2a
spec:
  securityGroups:
    - sg-0dff111a1d11c1c11
  subnet: subnet-011b111c1f11fdf11
```

b.   Apply each custom resource file that you created to your cluster with the following command:

```
kubectl apply -f us-west-2a.yaml
```

c.   (Optional, but recommended for multi-AZ worker node groups) By default, Kubernetes applies the Availability Zone of a node to the `failure-domain.beta.kubernetes.io/zone` label. If you named your ENIConfig custom resources after each Availability Zone in your VPC, as recommended in step 6a above, then you can enable Kubernetes to automatically apply the corresponding ENIConfig for the worker node's Availability Zone with the following command.

```
kubectl set env daemonset aws-node -n kube-system ENI_CONFIG_LABEL_DEF=failure-
domain.beta.kubernetes.io/zone
```

> **Note**
> Ensure that an annotation with the key `k8s.amazonaws.com/eniConfig` for the `ENI_CONFIG_ANNOTATION_DEF` environment variable doesn't exist in the container spec for the `aws-node` daemonset. If it exists, it overrides the `ENI_CONFIG_LABEL_DEF` value, and should be removed. You can check to see if the variable is set with the `kubectl describe daemonset aws-node -n kube-system | grep ENI_CONFIG_ANNOTATION_DEF` command. If no output is returned, then the variable is not set.

7.   Create a new self-managed worker node group for each `ENIConfig` that you configured.

a.   Determine the maximum number of pods that can be scheduled on each worker node using the following formula.

```
maxPods = (number of interfaces - 1) * (max IPv4 address per interface - 1) + 2
```

For example, the `m5.large` instance type supports three network interfaces and ten IPv4 addresses per interface. Inserting the values into the formula, the instance can support a maximum of 20 pods, as shown in the following calculation.

```
maxPods = (3 - 1) * (10 - 1) + 2 = 20
```

For more information about the the maximum number of network interfaces per instance type, see Elastic Network Interfaces in the Amazon EC2 User Guide for Linux Instances.

b. Follow the steps in the **Self-managed nodes** tab of Launching Amazon EKS Linux worker nodes (p. 92) to create each new self-managed worker node group. After you've opened the AWS CloudFormation template, enter values as described in the instructions. For the **BootstrapArguments** field, enter the following value.

```
--use-max-pods false --kubelet-extra-args '--max-pods=20'
```

8. After your worker node groups are created, record the security group that was created for each worker node group and apply it to its associated `ENIConfig`. Edit each `ENIConfig` with the following command, replacing *eniconfig-name* with your value:

```
kubectl edit eniconfig.crd.k8s.amazonaws.com/eniconfig-name
```

If you followed best practices from steps 6a and 6c above, the `eniconfig-name` corresponds to the Availability Zone name.

The `spec` section should look like this:

```
spec:
  securityGroups:
  - sg-0dff222a2d22c2c22
  subnet: subnet-022b222c2f22fdf22
```

9. If you have any worker nodes in your cluster that had pods placed on them before you completed this procedure, you should terminate them. Only new nodes that are registered with the `k8s.amazonaws.com/eniConfig` label will use the new custom networking feature.

CNI metrics helper

The CNI metrics helper is a tool that you can use to scrape elastic network interface and IP address information, aggregate metrics at the cluster level, and publish the metrics to Amazon CloudWatch.

When managing an Amazon EKS cluster, you may want to know how many IP addresses have been assigned and how many are available. The CNI metrics helper helps you to:

- Track these metrics over time
- Troubleshoot and diagnose issues related to IP assignment and reclamation
- Provide insights for capacity planning

When a worker node is provisioned, the CNI plugin automatically allocates a pool of secondary IP addresses from the node's subnet to the primary elastic network interface (`eth0`). This pool of IP addresses is known as the *warm pool*, and its size is determined by the worker node's instance type. For example, a `c4.large` instance can support three elastic network interfaces and nine IP addresses per interface. The number of IP addresses available for a given pod is one less than the maximum (of ten) because one of the IP addresses is reserved for the elastic network interface itself. For more information,

see IP Addresses Per Network Interface Per Instance Type in the *Amazon EC2 User Guide for Linux Instances*.

As the pool of IP addresses is depleted, the plugin automatically attaches another elastic network interface to the instance and allocates another set of secondary IP addresses to that interface. This process continues until the node can no longer support additional elastic network interfaces.

The following metrics are collected for your cluster and exported to CloudWatch:

- The maximum number of elastic network interfaces that the cluster can support
- The number of elastic network interfaces have been allocated to pods
- The number of IP addresses currently assigned to pods
- The total and maximum numbers of IP addresses available
- The number of ipamD errors

Deploying the CNI metrics helper

The CNI metrics helper requires `cloudwatch:PutMetricData` permissions to send metric data to CloudWatch. This section helps you to create an IAM policy with those permissions, apply it to your worker node instance role, and then deploy the CNI metrics helper.

To create an IAM policy for the CNI metrics helper

1. Create a file called `allow_put_metrics_data.json` and populate it with the following policy document.

   ```
   {
     "Version": "2012-10-17",
     "Statement": [
       {
         "Effect": "Allow",
         "Action": "cloudwatch:PutMetricData",
         "Resource": "*"
       }
     ]
   }
   ```

2. Create an IAM policy called `CNIMetricsHelperPolicy` for your worker node instance profile that allows the CNI metrics helper to make calls to AWS APIs on your behalf. Use the following AWS CLI command to create the IAM policy in your AWS account.

   ```
   aws iam create-policy --policy-name CNIMetricsHelperPolicy \
   --description "Grants permission to write metrics to CloudWatch" \
   --policy-document file://allow_put_metrics_data.json
   ```

   Take note of the policy ARN that is returned.

3. Get the IAM role name for your worker nodes. Use the following command to print the `aws-auth` configmap.

   ```
   kubectl -n kube-system describe configmap aws-auth
   ```

   Output:

   ```
   Name:          aws-auth
   Namespace:     kube-system
   ```

```
Labels:         <none>
Annotations:    <none>

Data
====
mapRoles:
----
- groups:
  - system:bootstrappers
  - system:nodes
  rolearn: arn:aws:iam::111122223333:role/eksctl-prod-nodegroup-standard-wo-
NodeInstanceRole-GKNS581EASPU
  username: system:node:{{EC2PrivateDNSName}}

Events:   <none>
```

Record the role name for any `rolearn` values that have the `system:nodes` group assigned to them. In the above example output, the role name is *eksctl-prod-nodegroup-standard-wo-NodeInstanceRole-GKNS581EASPU*. You should have one value for each node group in your cluster.

4. Attach the new `CNIMetricsHelperPolicy` IAM policy to each of the worker node IAM roles you identified earlier with the following command, substituting the red text with your own AWS account number and worker node IAM role name.

```
aws iam attach-role-policy \
--policy-arn arn:aws:iam::111122223333:policy/CNIMetricsHelperPolicy \
--role-name eksctl-prod-nodegroup-standard-wo-NodeInstanceRole-GKNS581EASPU
```

To deploy the CNI metrics helper

- Apply the CNI metrics helper manifest with the following command.

```
kubectl apply -f https://raw.githubusercontent.com/aws/amazon-vpc-cni-k8s/release-1.6/
config/v1.6/cni-metrics-helper.yaml
```

Creating a metrics dashboard

After you have deployed the CNI metrics helper, you can view the CNI metrics in the CloudWatch console. This topic helps you to create a dashboard for viewing your cluster's CNI metrics.

To create a CNI metrics dashboard

1. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.
2. In the left navigation, choose **Metrics**.
3. Under **Custom Namespaces**, choose **Kubernetes**.
4. Choose **CLUSTER_ID**.
5. On the **All metrics** tab, select the metrics you want to add to the dashboard.
6. Choose **Actions**, and then **Add to dashboard**.
7. In the **Select a dashboard** section, choose **Create new** and enter a name for your dashboard, such as "EKS-CNI-metrics".
8. In the **Select a widget type** section, choose **Number**.
9. In the **Customize the widget title** section, enter a logical name for your dashboard title, such as "EKS CNI metrics".

10. Choose **Add to dashboard** to finish. Now your CNI metrics are added to a dashboard that you can monitor, as shown below.
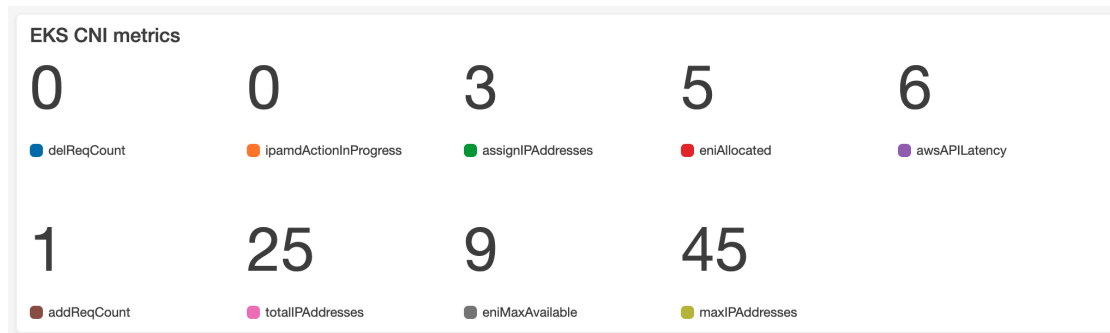
Amazon VPC CNI plugin for Kubernetes upgrades

When you launch an Amazon EKS cluster, we apply a recent version of the Amazon VPC CNI plugin for Kubernetes to your cluster. The absolute latest version of the plugin is available on GitHub for a short grace period before new clusters are switched over to use it. Amazon EKS does not automatically upgrade the CNI plugin on your cluster when new versions are released. To get a newer version of the CNI plugin on existing clusters, you must manually upgrade the plugin.

The latest version that we recommend is version 1.6.1. You can view the different releases available for the plugin, and read the release notes for each version on GitHub.

Use the following procedures to check your CNI plugin version and upgrade to the latest recommended version.

To check your Amazon VPC CNI plugin for Kubernetes version

- Use the following command to print your cluster's CNI version:

```
kubectl describe daemonset aws-node --namespace kube-system | grep Image | cut -d "/" -f 2
```

Output:

```
amazon-k8s-cni:1.5.7
```

In this example output, the CNI version is 1.5.7, which is earlier than the current recommended version, 1.6.1. Use the following procedure to upgrade the CNI.

To upgrade the Amazon VPC CNI plugin for Kubernetes

- If your CNI version is earlier than 1.6.1, then use the appropriate command below to update your CNI version to the latest recommended version:

  If your cluster is in China (Beijing) (`cn-north-1`) or China (Ningxia) (`cn-northwest-1`)

  ```
  kubectl apply -f https://raw.githubusercontent.com/aws/amazon-vpc-cni-k8s/master/config/v1.6/aws-k8s-cni-cn.yaml
  ```

If your cluster is in AWS GovCloud (US-East) (`us-gov-east-1`)

```
kubectl apply -f https://raw.githubusercontent.com/aws/amazon-vpc-cni-k8s/
release-1.6/config/v1.6/aws-k8s-cni-us-gov-east-1.yaml
```

If your cluster is in AWS GovCloud (US-West) (`us-gov-west-1`)

```
kubectl apply -f https://raw.githubusercontent.com/aws/amazon-vpc-cni-k8s/
release-1.6/config/v1.6/aws-k8s-cni-us-gov-west-1.yaml
```

For all other Regions

```
kubectl apply -f https://raw.githubusercontent.com/aws/amazon-vpc-cni-k8s/
release-1.6/config/v1.6/aws-k8s-cni.yaml
```

Installing or upgrading CoreDNS

CoreDNS is supported on Amazon EKS clusters with Kubernetes version 1.13 or later. Clusters that were created with Kubernetes version 1.10 shipped with `kube-dns` as the default DNS and service discovery provider. If you have updated from a 1.10 cluster and you want to use CoreDNS for DNS and service discovery, then you must install CoreDNS and remove `kube-dns`.

To check if your cluster is already running CoreDNS, use the following command.

```
kubectl get pod -n kube-system -l k8s-app=kube-dns
```

If the output shows `coredns` in the pod names, then you're already running CoreDNS in your cluster. If not, use the following procedure to update your DNS and service discovery provider to CoreDNS.

Note
The service for CoreDNS is still called `kube-dns` for backward compatibility.

To install CoreDNS on an updated Amazon EKS cluster with `kubectl`

1. Add the `{"eks.amazonaws.com/component": "kube-dns"}` selector to the `kube-dns` deployment for your cluster. This prevents the two DNS deployments from competing for control of the same set of labels.

    ```
    kubectl patch -n kube-system deployment/kube-dns --patch \
    '{"spec":{"selector":{"matchLabels":{"eks.amazonaws.com/component":"kube-dns"}}}}'
    ```

2. Deploy CoreDNS to your cluster.

    a. Set your cluster's DNS IP address to the `DNS_CLUSTER_IP` environment variable.

    ```
    export DNS_CLUSTER_IP=$(kubectl get svc -n kube-system kube-dns -o
      jsonpath='{.spec.clusterIP}')
    ```

    b. Set the `REGION` environment variable to your cluster's AWS *region-code*.

    ```
    export REGION="region-code"
    ```

    c. Download the CoreDNS manifest from the Amazon EKS resource bucket.

```
curl -o dns.yaml https://amazon-eks.s3.us-west-2.amazonaws.com/
cloudformation/2020-05-08/dns.yaml
```

d. Replace the variable placeholders in the `dns.yaml` file with your environment variable values and apply the updated manifest to your cluster. The following command completes this in one step.

```
cat dns.yaml | sed -e "s/REGION/$REGION/g" | sed -e "s/DNS_CLUSTER_IP/
$DNS_CLUSTER_IP/g" | kubectl apply -f -
```

e. Fetch the `coredns` pod name from your cluster.

```
COREDNS_POD=$(kubectl get pod -n kube-system -l eks.amazonaws.com/component=coredns
 \
-o jsonpath='{.items[0].metadata.name}')
```

f. Query the `coredns` pod to ensure that it's receiving requests.

```
kubectl get --raw /api/v1/namespaces/kube-system/pods/$COREDNS_POD:9153/proxy/
metrics \
| grep 'coredns_dns_request_count_total'
```

> **Note**
> It might take several minutes for the expected output to return properly, depending on the rate of DNS requests in your cluster.

In the following expected output, the number `23` is the DNS request count total.

```
# HELP coredns_dns_request_count_total Counter of DNS requests made per zone,
 protocol and family.
# TYPE coredns_dns_request_count_total counter
coredns_dns_request_count_total{family="1",proto="udp",server="dns://:53",zone="."}
 23
```

3. Upgrade CoreDNS to the recommended version for your cluster by completing the steps in the section called "Upgrading CoreDNS" (p. 176).

4. Scale down the `kube-dns` deployment to zero replicas.

```
kubectl scale -n kube-system deployment/kube-dns --replicas=0
```

5. Clean up the old `kube-dns` resources.

```
kubectl delete -n kube-system deployment/kube-dns serviceaccount/kube-dns configmap/
kube-dns
```

Upgrading CoreDNS

1. Check the current version of your cluster's `coredns` deployment.

```
kubectl describe deployment coredns --namespace kube-system | grep Image | cut -d "/" -
f 3
```

Output:

```
coredns:v1.1.3
```

The recommended `coredns` versions for the corresponding Kubernetes versions are as follows:

Kubernetes version	1.16	1.15	1.14	1.13
CoreDNS	1.6.6	1.6.6	1.6.6	1.6.6

2. If your current `coredns` version is 1.5.0 or later, but earlier than the recommended version, then skip this step. If your current version is earlier than 1.5.0, then you need to modify the config map for `coredns` to use the `forward` plug-in, rather than the `proxy` plug-in.

   a. Open the configmap with the following command.

   ```
   kubectl edit configmap coredns -n kube-system
   ```

   b. Replace `proxy` in the following line with `forward`. Save the file and exit the editor.

   ```
   proxy . /etc/resolv.conf
   ```

3. Retrieve your current `coredns` image:

   ```
   kubectl get deployment coredns --namespace kube-system -
   o=jsonpath='{$.spec.template.spec.containers[:1].image}'
   ```

4. Update `coredns` to the recommended version by taking the output from the previous step and replacing the version tag with your cluster's recommended `coredns` version:

   ```
   kubectl set image --namespace kube-system deployment.apps/coredns \
           coredns=602401143452.dkr.ecr.us-west-2.amazonaws.com/eks/coredns:v1.6.6
   ```

Installing Calico on Amazon EKS

Project Calico is a network policy engine for Kubernetes. With Calico network policy enforcement, you can implement network segmentation and tenant isolation. This is useful in multi-tenant environments where you must isolate tenants from each other or when you want to create separate environments for development, staging, and production. Network policies are similar to AWS security groups in that you can create network ingress and egress rules. Instead of assigning instances to a security group, you assign network policies to pods using pod selectors and labels. The following procedure shows you how to install Calico on your Amazon EKS cluster.

Note
Calico is not supported when using Fargate with Amazon EKS.

To install Calico on your Amazon EKS cluster

1. Apply the Calico manifest from the `aws/amazon-vpc-cni-k8s` GitHub project. This manifest creates DaemonSets in the `kube-system` namespace.

   ```
   kubectl apply -f https://raw.githubusercontent.com/aws/amazon-vpc-cni-k8s/release-1.6/
   config/v1.6/calico.yaml
   ```

2. Watch the `kube-system` DaemonSets and wait for the `calico-node` DaemonSet to have the `DESIRED` number of pods in the `READY` state. When this happens, Calico is working.

```
kubectl get daemonset calico-node --namespace kube-system
```

Output:

```
NAME            DESIRED    CURRENT    READY      UP-TO-DATE    AVAILABLE    NODE SELECTOR
 AGE
calico-node     3          3          3          3             3            <none>
 38s
```

To delete Calico from your Amazon EKS cluster

- If you are done using Calico in your Amazon EKS cluster, you can delete the DaemonSet with the following command:

```
kubectl delete -f https://raw.githubusercontent.com/aws/amazon-vpc-cni-k8s/release-1.6/
config/v1.6/calico.yaml
```

Stars policy demo

This section walks through the Stars policy demo provided by the Project Calico documentation. The demo creates a frontend, backend, and client service on your Amazon EKS cluster. The demo also creates a management GUI that shows the available ingress and egress paths between each service.

Before you create any network policies, all services can communicate bidirectionally. After you apply the network policies, you can see that the client can only communicate with the frontend service, and the backend can only communicate with the frontend.

To run the Stars policy demo

1. Apply the frontend, backend, client, and management UI services:

```
kubectl apply -f https://docs.projectcalico.org/v3.3/getting-started/kubernetes/
tutorials/stars-policy/manifests/00-namespace.yaml
kubectl apply -f https://docs.projectcalico.org/v3.3/getting-started/kubernetes/
tutorials/stars-policy/manifests/01-management-ui.yaml
kubectl apply -f https://docs.projectcalico.org/v3.3/getting-started/kubernetes/
tutorials/stars-policy/manifests/02-backend.yaml
kubectl apply -f https://docs.projectcalico.org/v3.3/getting-started/kubernetes/
tutorials/stars-policy/manifests/03-frontend.yaml
kubectl apply -f https://docs.projectcalico.org/v3.3/getting-started/kubernetes/
tutorials/stars-policy/manifests/04-client.yaml
```

2. Wait for all of the pods to reach the `Running` status:

```
kubectl get pods --all-namespaces --watch
```

3. To connect to the management UI, forward your local port 9001 to the `management-ui` service running on your cluster:

```
kubectl port-forward service/management-ui -n management-ui 9001
```

4. Open a browser on your local system and point it to http://localhost:9001/. You should see the management UI. The **C** node is the client service, the **F** node is the frontend service, and the **B** node

is the backend service. Each node has full communication access to all other nodes (as indicated by the bold, colored lines).
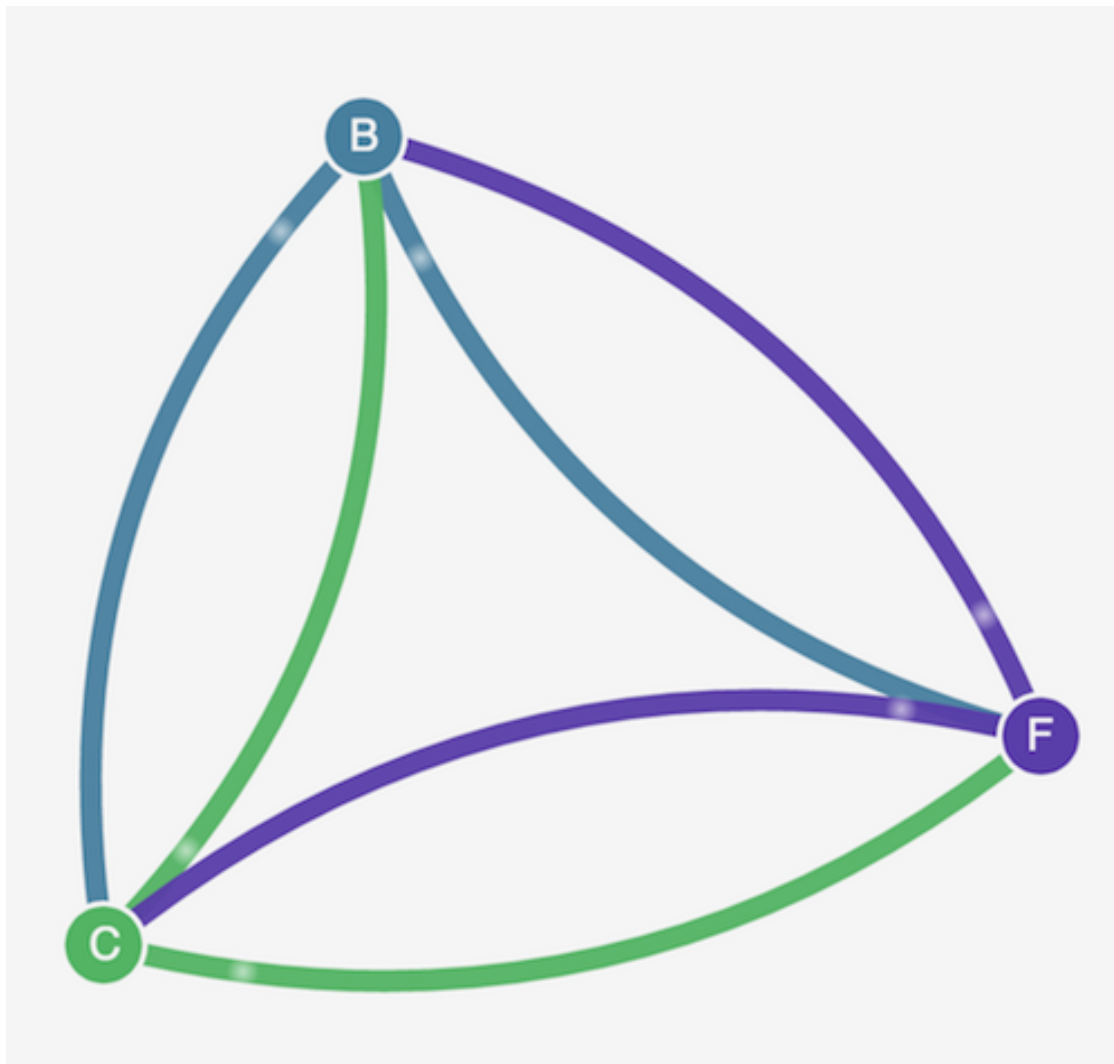
5. Apply the following network policies to isolate the services from each other:

    ```
    kubectl apply -n stars -f https://docs.projectcalico.org/v3.3/getting-started/
    kubernetes/tutorials/stars-policy/policies/default-deny.yaml
    kubectl apply -n client -f https://docs.projectcalico.org/v3.3/getting-started/
    kubernetes/tutorials/stars-policy/policies/default-deny.yaml
    ```

6. Refresh your browser. You see that the management UI can no longer reach any of the nodes, so they don't show up in the UI.

7. Apply the following network policies to allow the management UI to access the services:

    ```
    kubectl apply -f https://docs.projectcalico.org/v3.3/getting-started/kubernetes/
    tutorials/stars-policy/policies/allow-ui.yaml
    kubectl apply -f https://docs.projectcalico.org/v3.3/getting-started/kubernetes/
    tutorials/stars-policy/policies/allow-ui-client.yaml
    ```

8. Refresh your browser. You see that the management UI can reach the nodes again, but the nodes cannot communicate with each other.
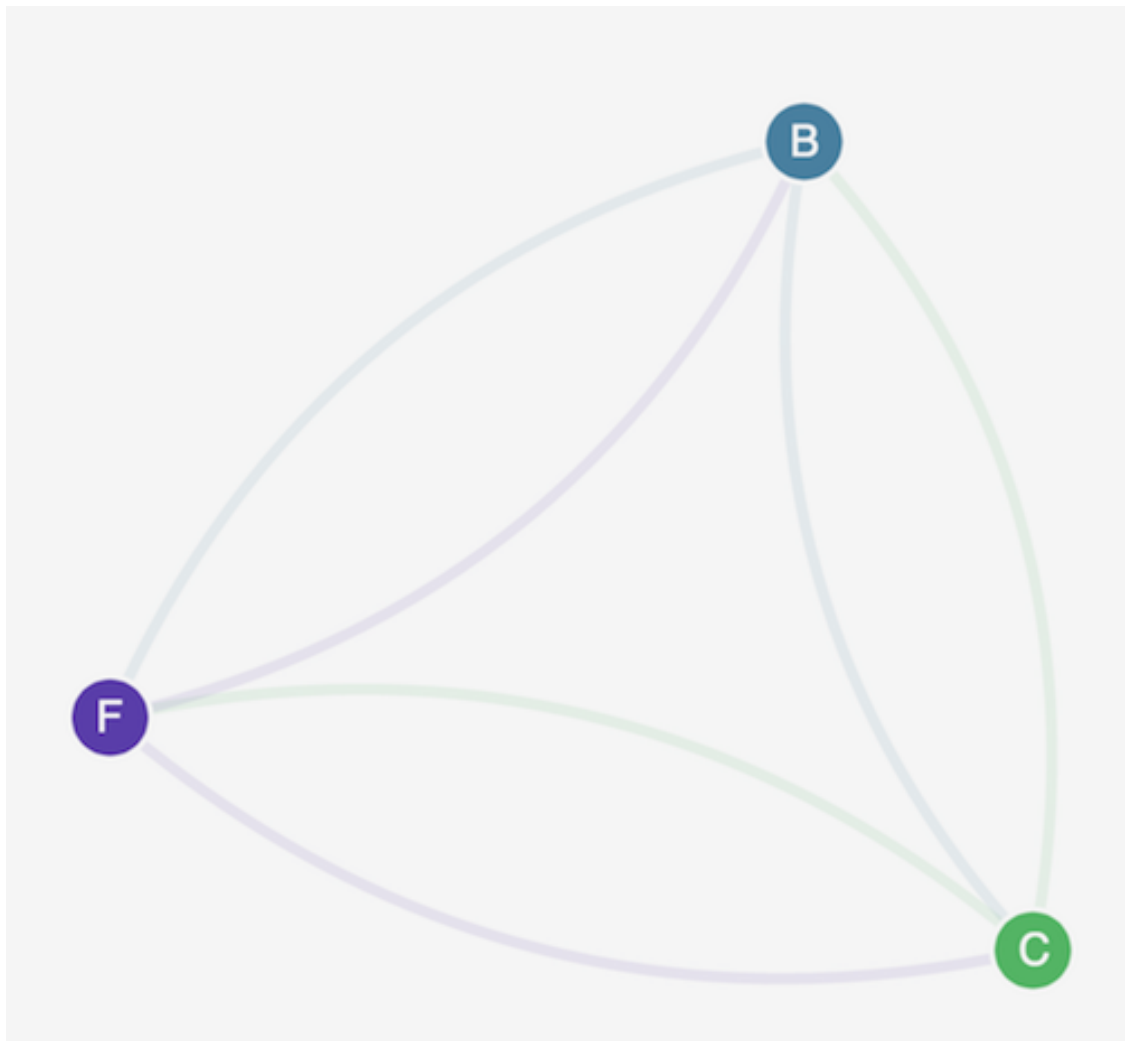
9. Apply the following network policy to allow traffic from the frontend service to the backend service:

```
kubectl apply -f https://docs.projectcalico.org/v3.3/getting-started/kubernetes/
tutorials/stars-policy/policies/backend-policy.yaml
```

10. Apply the following network policy to allow traffic from the `client` namespace to the frontend service:

```
kubectl apply -f https://docs.projectcalico.org/v3.3/getting-started/kubernetes/
tutorials/stars-policy/policies/frontend-policy.yaml
```
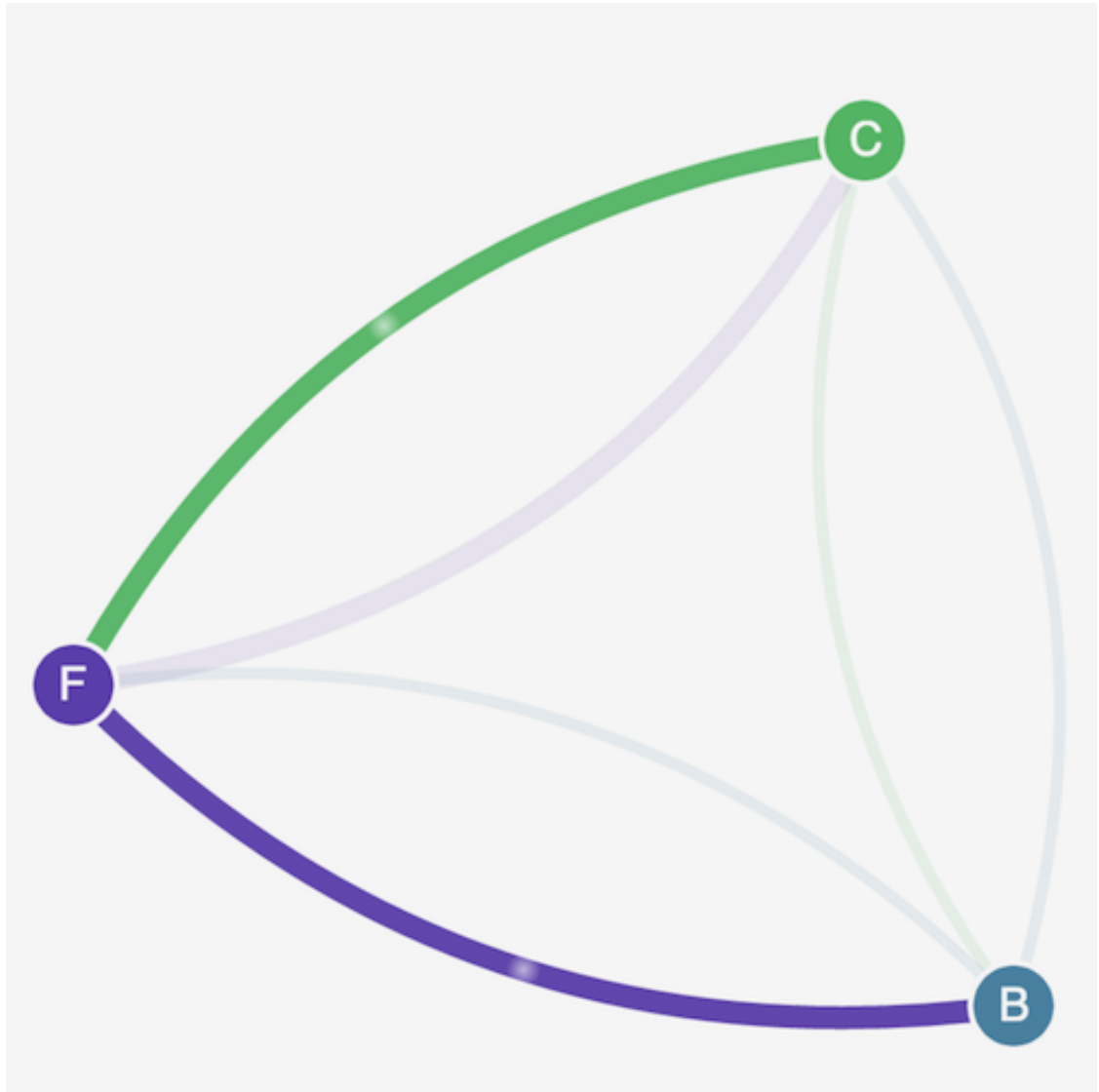
11. (Optional) When you are done with the demo, you can delete its resources with the following commands:

```
kubectl delete -f https://docs.projectcalico.org/v3.3/getting-started/kubernetes/
tutorials/stars-policy/manifests/04-client.yaml
kubectl delete -f https://docs.projectcalico.org/v3.3/getting-started/kubernetes/
tutorials/stars-policy/manifests/03-frontend.yaml
kubectl delete -f https://docs.projectcalico.org/v3.3/getting-started/kubernetes/
tutorials/stars-policy/manifests/02-backend.yaml
kubectl delete -f https://docs.projectcalico.org/v3.3/getting-started/kubernetes/
tutorials/stars-policy/manifests/01-management-ui.yaml
kubectl delete -f https://docs.projectcalico.org/v3.3/getting-started/kubernetes/
tutorials/stars-policy/manifests/00-namespace.yaml
```

Managing cluster authentication

Amazon EKS uses IAM to provide authentication to your Kubernetes cluster (through the **aws eks get-token** command, available in version 1.18.49 or later of the AWS CLI, or the AWS IAM Authenticator for Kubernetes), but it still relies on native Kubernetes Role Based Access Control (RBAC) for authorization. This means that IAM is only used for authentication of valid IAM entities. All permissions for interacting with your Amazon EKS cluster's Kubernetes API is managed through the native Kubernetes RBAC system.
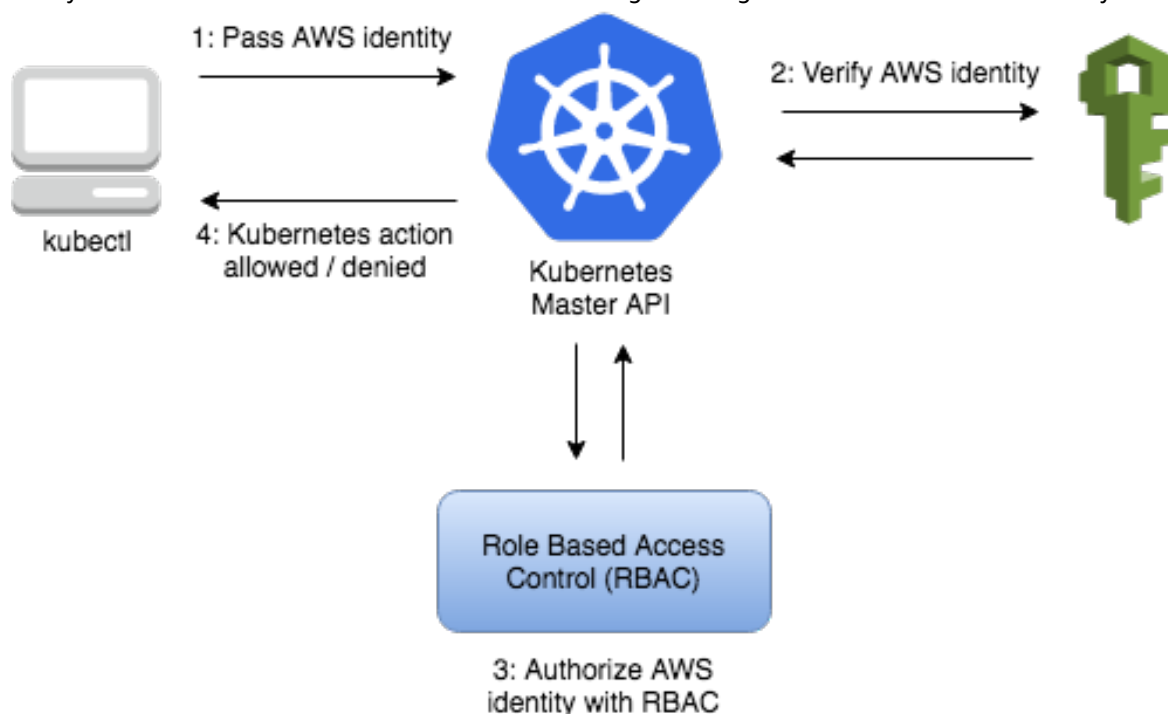
Topics

Installing `kubectl`

Kubernetes uses a command line utility called `kubectl` for communicating with the cluster API server. The `kubectl` binary is available in many operating system package managers, and this option is often much easier than a manual download and install process. You can follow the instructions for your specific operating system or package manager in the Kubernetes documentation to install.

This topic helps you to download and install the Amazon EKS-vended **kubectl** binaries for macOS, Linux, and Windows operating systems. These binaries are identical to the upstream community versions, and are not unique to Amazon EKS or AWS.

Note
You must use a `kubectl` version that is within one minor version difference of your Amazon EKS cluster control plane . For example, a 1.15 `kubectl` client should work with Kubernetes 1.14, 1.15, and 1.16 clusters.

macOS

To install `kubectl` on macOS

1. Download the Amazon EKS-vended **kubectl** binary for your cluster's Kubernetes version from Amazon S3:

    *   **Kubernetes 1.16:**

        ```
        curl -o kubectl https://amazon-eks.s3.us-west-2.amazonaws.com/1.16.8/2020-04-16/
        bin/darwin/amd64/kubectl
        ```

    *   **Kubernetes 1.15:**

        ```
        curl -o kubectl https://amazon-eks.s3.us-west-2.amazonaws.com/1.15.10/2020-02-22/
        bin/darwin/amd64/kubectl
        ```

        Note
        You can use this version with 1.15.10 and 1.15.11 clusters.

    *   **Kubernetes 1.14:**

        ```
        curl -o kubectl https://amazon-eks.s3.us-west-2.amazonaws.com/1.14.6/2019-08-22/
        bin/darwin/amd64/kubectl
        ```

    *   **Kubernetes 1.13:**

        ```
        curl -o kubectl https://amazon-eks.s3.us-west-2.amazonaws.com/1.13.8/2019-08-14/
        bin/darwin/amd64/kubectl
        ```

2. (Optional) Verify the downloaded binary with the SHA-256 sum for your binary.

    a. Download the SHA-256 sum for your cluster's Kubernetes version for macOS:

        *   **Kubernetes 1.16:**

            ```
            curl -o kubectl.sha256 https://amazon-eks.s3.us-
            west-2.amazonaws.com/1.16.8/2020-04-16/bin/darwin/amd64/kubectl.sha256
            ```

        *   **Kubernetes 1.15:**

            ```
            curl -o kubectl.sha256 https://amazon-eks.s3.us-
            west-2.amazonaws.com/1.15.10/2020-02-22/bin/darwin/amd64/kubectl.sha256
            ```

        *   **Kubernetes 1.14:**

            ```
            curl -o kubectl.sha256 https://amazon-eks.s3.us-
            west-2.amazonaws.com/1.14.6/2019-08-22/bin/darwin/amd64/kubectl.sha256
            ```

        *   **Kubernetes 1.13:**

            ```
            curl -o kubectl.sha256 https://amazon-eks.s3.us-
            west-2.amazonaws.com/1.13.8/2019-08-14/bin/darwin/amd64/kubectl.sha256
            ```

    b.   Check the SHA-256 sum for your downloaded binary.

```
openssl sha1 -sha256 kubectl
```

    c.   Compare the generated SHA-256 sum in the command output against your downloaded SHA-256 file. The two should match.

3.   Apply execute permissions to the binary.

```
chmod +x ./kubectl
```

4.   Copy the binary to a folder in your `PATH`. If you have already installed a version of **kubectl**, then we recommend creating a `$HOME/bin/kubectl` and ensuring that `$HOME/bin` comes first in your `$PATH`.

```
mkdir -p $HOME/bin && cp ./kubectl $HOME/bin/kubectl && export PATH=$PATH:$HOME/bin
```

5.   (Optional) Add the `$HOME/bin` path to your shell initialization file so that it is configured when you open a shell.

```
echo 'export PATH=$PATH:$HOME/bin' >> ~/.bash_profile
```

6.   After you install **kubectl**, you can verify its version with the following command:

```
kubectl version --short --client
```

Linux

To install `kubectl` on Linux

1.   Download the Amazon EKS-vended **kubectl** binary for your cluster's Kubernetes version from Amazon S3:

- **Kubernetes 1.16:**

```
curl -o kubectl https://amazon-eks.s3.us-west-2.amazonaws.com/1.16.8/2020-04-16/
bin/linux/amd64/kubectl
```

- **Kubernetes 1.15:**

```
curl -o kubectl https://amazon-eks.s3.us-west-2.amazonaws.com/1.15.10/2020-02-22/
bin/linux/amd64/kubectl
```

    Note
    You can use this version with 1.15.10 and 1.15.11 clusters.

- **Kubernetes 1.14:**

```
curl -o kubectl https://amazon-eks.s3.us-west-2.amazonaws.com/1.14.6/2019-08-22/
bin/linux/amd64/kubectl
```

- **Kubernetes 1.13:**

```
curl -o kubectl https://amazon-eks.s3.us-west-2.amazonaws.com/1.13.8/2019-08-14/
bin/linux/amd64/kubectl
```

2.   (Optional) Verify the downloaded binary with the SHA-256 sum for your binary.

a. Download the SHA-256 sum for your cluster's Kubernetes version for Linux:

- **Kubernetes 1.16:**

```
curl -o kubectl.sha256 https://amazon-eks.s3.us-
west-2.amazonaws.com/1.16.8/2020-04-16/bin/linux/amd64/kubectl.sha256
```

- **Kubernetes 1.15:**

```
curl -o kubectl.sha256 https://amazon-eks.s3.us-
west-2.amazonaws.com/1.15.10/2020-02-22/bin/linux/amd64/kubectl.sha256
```

- **Kubernetes 1.14:**

```
curl -o kubectl.sha256 https://amazon-eks.s3.us-
west-2.amazonaws.com/1.14.6/2019-08-22/bin/linux/amd64/kubectl.sha256
```

- **Kubernetes 1.13:**

```
curl -o kubectl.sha256 https://amazon-eks.s3.us-
west-2.amazonaws.com/1.13.8/2019-08-14/bin/linux/amd64/kubectl.sha256
```

b. Check the SHA-256 sum for your downloaded binary.

```
openssl sha1 -sha256 kubectl
```

c. Compare the generated SHA-256 sum in the command output against your downloaded SHA-256 file. The two should match.

3. Apply execute permissions to the binary.

```
chmod +x ./kubectl
```

4. Copy the binary to a folder in your `PATH`. If you have already installed a version of **kubectl**, then we recommend creating a `$HOME/bin/kubectl` and ensuring that `$HOME/bin` comes first in your `$PATH`.

```
mkdir -p $HOME/bin && cp ./kubectl $HOME/bin/kubectl && export PATH=$PATH:$HOME/bin
```

5. (Optional) Add the `$HOME/bin` path to your shell initialization file so that it is configured when you open a shell.

> **Note**
> This step assumes you are using the Bash shell; if you are using another shell, change the command to use your specific shell initialization file.

```
echo 'export PATH=$PATH:$HOME/bin' >> ~/.bashrc
```

6. After you install **kubectl**, you can verify its version with the following command:

```
kubectl version --short --client
```

Windows

To install `kubectl` on Windows

1. Open a PowerShell terminal.

2. Download the Amazon EKS-vended **kubectl** binary for your cluster's Kubernetes version from Amazon S3:

- **Kubernetes 1.16:**

```
curl -o kubectl.exe https://amazon-eks.s3.us-
west-2.amazonaws.com/1.16.8/2020-04-16/bin/windows/amd64/kubectl.exe
```

- **Kubernetes 1.15:**

```
curl -o kubectl.exe https://amazon-eks.s3.us-
west-2.amazonaws.com/1.15.10/2020-02-22/bin/windows/amd64/kubectl.exe
```

> **Note**
> You can use this version with 1.15.10 and 1.15.11 clusters.

- **Kubernetes 1.14:**

```
curl -o kubectl.exe https://amazon-eks.s3.us-
west-2.amazonaws.com/1.14.6/2019-08-22/bin/windows/amd64/kubectl.exe
```

- **Kubernetes 1.13:**

```
curl -o kubectl.exe https://amazon-eks.s3.us-
west-2.amazonaws.com/1.13.8/2019-08-14/bin/windows/amd64/kubectl.exe
```

3. (Optional) Verify the downloaded binary with the SHA-256 sum for your binary.

   a. Download the SHA-256 sum for your cluster's Kubernetes version for Windows:

   - **Kubernetes 1.16:**

   ```
   curl -o kubectl.exe.sha256 https://amazon-eks.s3.us-
   west-2.amazonaws.com/1.16.8/2020-04-16/bin/windows/amd64/kubectl.exe.sha256
   ```

   - **Kubernetes 1.15:**

   ```
   curl -o kubectl.exe.sha256 https://amazon-eks.s3.us-
   west-2.amazonaws.com/1.15.10/2020-02-22/bin/windows/amd64/kubectl.exe.sha256
   ```

   - **Kubernetes 1.14:**

   ```
   curl -o kubectl.exe.sha256 https://amazon-eks.s3.us-
   west-2.amazonaws.com/1.14.6/2019-08-22/bin/windows/amd64/kubectl.exe.sha256
   ```

   - **Kubernetes 1.13:**

   ```
   curl -o kubectl.exe.sha256 https://amazon-eks.s3.us-
   west-2.amazonaws.com/1.13.8/2019-08-14/bin/windows/amd64/kubectl.exe.sha256
   ```

   b. Check the SHA-256 sum for your downloaded binary.

   ```
   Get-FileHash kubectl.exe
   ```

   c. Compare the generated SHA-256 sum in the command output against your downloaded SHA-256 file. The two should match, although the PowerShell output will be uppercase.

4. Copy the binary to a folder in your `PATH`. If you have an existing directory in your PATH that you use for command line utilities, copy the binary to that directory. Otherwise, complete the following steps.

   a. Create a new directory for your command line binaries, such as `C:\bin`.

   b. Copy the `kubectl.exe` binary to your new directory.

   c. Edit your user or system PATH environment variable to add the new directory to your PATH.

   d. Close your PowerShell terminal and open a new one to pick up the new PATH variable.

5. After you install **kubectl**, you can verify its version with the following command:

```
kubectl version --short --client
```

Installing aws-iam-authenticator

Amazon EKS uses IAM to provide authentication to your Kubernetes cluster through the AWS IAM authenticator for Kubernetes. You can configure the stock **kubectl** client to work with Amazon EKS by installing the AWS IAM authenticator for Kubernetes and modifying your **kubectl** configuration file to use it for authentication.

macOS

To install `aws-iam-authenticator` with Homebrew

The easiest way to install the `aws-iam-authenticator` is with Homebrew.

1. If you do not already have Homebrew installed on your Mac, install it with the following command.

```
/bin/bash -c "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/
master/install.sh)"
```

2. Install the `aws-iam-authenticator` with the following command.

```
brew install aws-iam-authenticator
```

3. Test that the `aws-iam-authenticator` binary works.

```
aws-iam-authenticator help
```

To install `aws-iam-authenticator` on macOS

You can also install the AWS-vended version of the `aws-iam-authenticator` by following these steps.

1. Download the Amazon EKS-vended `aws-iam-authenticator` binary from Amazon S3:

```
curl -o aws-iam-authenticator https://amazon-eks.s3.us-
west-2.amazonaws.com/1.16.8/2020-04-16/bin/darwin/amd64/aws-iam-authenticator
```

2. (Optional) Verify the downloaded binary with the SHA-256 sum provided in the same bucket prefix.

   a. Download the SHA-256 sum for your system.

```
curl -o aws-iam-authenticator.sha256 https://amazon-eks.s3.us-
west-2.amazonaws.com/1.16.8/2020-04-16/bin/darwin/amd64/aws-iam-
authenticator.sha256
```

b. Check the SHA-256 sum for your downloaded binary.

```
openssl sha1 -sha256 aws-iam-authenticator
```

c. Compare the generated SHA-256 sum in the command output against your downloaded `aws-iam-authenticator.sha256` file. The two should match.

3. Apply execute permissions to the binary.

```
chmod +x ./aws-iam-authenticator
```

4. Copy the binary to a folder in your `$PATH`. We recommend creating a `$HOME/bin/aws-iam-authenticator` and ensuring that `$HOME/bin` comes first in your `$PATH`.

```
mkdir -p $HOME/bin && cp ./aws-iam-authenticator $HOME/bin/aws-iam-authenticator &&
 export PATH=$PATH:$HOME/bin
```

5. Add `$HOME/bin` to your `PATH` environment variable.

```
echo 'export PATH=$PATH:$HOME/bin' >> ~/.bash_profile
```

6. Test that the `aws-iam-authenticator` binary works.

```
aws-iam-authenticator help
```

Linux

To install `aws-iam-authenticator` on Linux

1. Download the Amazon EKS-vended `aws-iam-authenticator` binary from Amazon S3:

```
curl -o aws-iam-authenticator https://amazon-eks.s3.us-
west-2.amazonaws.com/1.16.8/2020-04-16/bin/linux/amd64/aws-iam-authenticator
```

2. (Optional) Verify the downloaded binary with the SHA-256 sum provided in the same bucket prefix.

a. Download the SHA-256 sum for your system.

```
curl -o aws-iam-authenticator.sha256 https://amazon-eks.s3.us-
west-2.amazonaws.com/1.16.8/2020-04-16/bin/linux/amd64/aws-iam-
authenticator.sha256
```

b. Check the SHA-256 sum for your downloaded binary.

```
openssl sha1 -sha256 aws-iam-authenticator
```

c. Compare the generated SHA-256 sum in the command output against your downloaded `aws-iam-authenticator.sha256` file. The two should match.

3. Apply execute permissions to the binary.

```
chmod +x ./aws-iam-authenticator
```

4. Copy the binary to a folder in your `$PATH`. We recommend creating a `$HOME/bin/aws-iam-authenticator` and ensuring that `$HOME/bin` comes first in your `$PATH`.

```
mkdir -p $HOME/bin && cp ./aws-iam-authenticator $HOME/bin/aws-iam-authenticator &&
 export PATH=$PATH:$HOME/bin
```

5. Add `$HOME/bin` to your `PATH` environment variable.

```
echo 'export PATH=$PATH:$HOME/bin' >> ~/.bashrc
```

6. Test that the `aws-iam-authenticator` binary works.

```
aws-iam-authenticator help
```

Windows

To install `aws-iam-authenticator` on Windows with Chocolatey

1. If you do not already have Chocolatey installed on your Windows system, see Installing chocolatey.
2. Open a PowerShell terminal window and install the `aws-iam-authenticator` package with the following command:

```
choco install -y aws-iam-authenticator
```

3. Test that the `aws-iam-authenticator` binary works.

```
aws-iam-authenticator help
```

To install `aws-iam-authenticator` on Windows

1. Open a PowerShell terminal window and download the Amazon EKS-vended `aws-iam-authenticator` binary from Amazon S3:

```
curl -o aws-iam-authenticator.exe https://amazon-eks.s3.us-
west-2.amazonaws.com/1.16.8/2020-04-16/bin/windows/amd64/aws-iam-authenticator.exe
```

2. (Optional) Verify the downloaded binary with the SHA-256 sum provided in the same bucket prefix.

   a. Download the SHA-256 sum for your system.

   ```
   curl -o aws-iam-authenticator.sha256 https://amazon-eks.s3.us-
   west-2.amazonaws.com/1.16.8/2020-04-16/bin/windows/amd64/aws-iam-
   authenticator.exe.sha256
   ```

   b. Check the SHA-256 sum for your downloaded binary.

   ```
   Get-FileHash aws-iam-authenticator.exe
   ```

   c. Compare the generated SHA-256 sum in the command output against your downloaded SHA-256 file. The two should match, although the PowerShell output will be uppercase.

3. Copy the binary to a folder in your `PATH`. If you have an existing directory in your PATH that you use for command line utilities, copy the binary to that directory. Otherwise, complete the following steps.

   a. Create a new directory for your command line binaries, such as `C:\bin`.

   b. Copy the `aws-iam-authenticator.exe` binary to your new directory.

   c. Edit your user or system PATH environment variable to add the new directory to your PATH.

   d. Close your PowerShell terminal and open a new one to pick up the new PATH variable.

4. Test that the `aws-iam-authenticator` binary works.

```
aws-iam-authenticator help
```

If you have an existing Amazon EKS cluster, create a `kubeconfig` file for that cluster. For more information, see Create a `kubeconfig` for Amazon EKS (p. 190). Otherwise, see Creating an Amazon EKS cluster (p. 21) to create a new Amazon EKS cluster.

Create a `kubeconfig` for Amazon EKS

In this section, you create a `kubeconfig` file for your cluster (or update an existing one).

This section offers two procedures to create or update your kubeconfig. You can quickly create or update a kubeconfig with the AWS CLI **update-kubeconfig** command by using the first procedure, or you can create a kubeconfig manually with the second procedure.

Amazon EKS uses the **aws eks get-token** command, available in version 1.18.49 or later of the AWS CLI or the AWS IAM Authenticator for Kubernetes with **kubectl** for cluster authentication. If you have installed the AWS CLI on your system, then by default the AWS IAM Authenticator for Kubernetes will use the same credentials that are returned with the following command:

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

For more information, see Configuring the AWS CLI in the *AWS Command Line Interface User Guide*.

To create your `kubeconfig` file with the AWS CLI

1. Ensure that you have version 1.18.49 or later of the AWS CLI installed. To install or upgrade the AWS CLI, see Installing the AWS Command Line Interface in the *AWS Command Line Interface User Guide*.

   Note
   Your system's Python version must be 2.7.9 or later. Otherwise, you receive `hostname doesn't match` errors with AWS CLI calls to Amazon EKS.

   You can check your AWS CLI version with the following command:

```
aws --version
```

   Important
   Package managers such **yum**, **apt-get**, or Homebrew for macOS are often behind several versions of the AWS CLI. To ensure that you have the latest version, see Installing the AWS Command Line Interface in the *AWS Command Line Interface User Guide*.

2. Use the AWS CLI **update-kubeconfig** command to create or update your kubeconfig for your cluster.

- By default, the resulting configuration file is created at the default kubeconfig path (`.kube/config`) in your home directory or merged with an existing kubeconfig at that location. You can specify another path with the `--kubeconfig` option.

- You can specify an IAM role ARN with the `--role-arn` option to use for authentication when you issue **kubectl** commands. Otherwise, the IAM entity in your default AWS CLI or SDK credential chain is used. You can view your default AWS CLI or SDK identity by running the **aws sts get-caller-identity** command.

- For more information, see the help page with the **aws eks update-kubeconfig help** command or see update-kubeconfig in the *AWS CLI Command Reference*.

   Note
   To run the following command, your account must be assigned the `eks:DescribeCluster` IAM permission for the cluster name that you specify.

   ```
   aws eks --region region-code update-kubeconfig --name cluster_name
   ```

3. Test your configuration.

   ```
   kubectl get svc
   ```

   Note
   If you receive the error `"aws-iam-authenticator": executable file not found in $PATH`, your **kubectl** isn't configured for Amazon EKS. For more information, see Installing `aws-iam-authenticator` (p. 187).
   If you receive any other authorization or resource type errors, see Unauthorized or access denied (`kubectl`) (p. 288) in the troubleshooting section.

   Output:

   ```
   NAME            TYPE        CLUSTER-IP    EXTERNAL-IP   PORT(S)   AGE
   svc/kubernetes  ClusterIP   10.100.0.1    <none>        443/TCP   1m
   ```

To create your `kubeconfig` file manually

1. Create the default ~/.kube directory if it does not already exist.

   ```
   mkdir -p ~/.kube
   ```

2. Open your favorite text editor and copy one of the `kubeconfig` code blocks below into it, depending on your preferred client token method.

   - To use the AWS CLI **aws eks get-token** command (requires version 1.18.49 or later of the AWS CLI):

     ```
     apiVersion: v1
     clusters:
     - cluster:
         server: <endpoint-url>
         certificate-authority-data: <base64-encoded-ca-cert>
       name: kubernetes
     contexts:
     - context:
         cluster: kubernetes
         user: aws
     ```

```
      name: aws
current-context: aws
kind: Config
preferences: {}
users:
- name: aws
  user:
    exec:
      apiVersion: client.authentication.k8s.io/v1alpha1
      command: aws
      args:
        - "eks"
        - "get-token"
        - "--cluster-name"
        - "<cluster-name>"
        # - "--role"
        # - "<role-arn>"
      # env:
        # - name: AWS_PROFILE
        #   value: "<aws-profile>"
```

- To use the AWS IAM authenticator for Kubernetes:

```
apiVersion: v1
clusters:
- cluster:
    server: <endpoint-url>
    certificate-authority-data: <base64-encoded-ca-cert>
  name: kubernetes
contexts:
- context:
    cluster: kubernetes
    user: aws
  name: aws
current-context: aws
kind: Config
preferences: {}
users:
- name: aws
  user:
    exec:
      apiVersion: client.authentication.k8s.io/v1alpha1
      command: aws-iam-authenticator
      args:
        - "token"
        - "-i"
        - "<cluster-name>"
        # - "-r"
        # - "<role-arn>"
      # env:
        # - name: AWS_PROFILE
        #   value: "<aws-profile>"
```

3. Replace the `<endpoint-url>` with the endpoint URL that was created for your cluster.

4. Replace the `<base64-encoded-ca-cert>` with the `certificateAuthority.data` that was created for your cluster.

5. Replace the `<cluster-name>` with your cluster name.

6. (Optional) To assume an IAM role to perform cluster operations instead of the default AWS credential provider chain, uncomment the `-r` or `--role` and `<role-arn>` lines and substitute an IAM role ARN to use with your user.

7. (Optional) To always use a specific named AWS credential profile (instead of the default AWS credential provider chain), uncomment the env lines and substitute *<aws-profile>* with the profile name to use.

8. Save the file to the default **kubectl** folder, with your cluster name in the file name. For example, if your cluster name is *devel*, save the file to ~/.kube/config-*devel*.

9. Add that file path to your KUBECONFIG environment variable so that **kubectl** knows where to look for your cluster configuration.

   - For Bash shells on macOS or Linux:

   ```
   export KUBECONFIG=$KUBECONFIG:~/.kube/config-devel
   ```

   - For PowerShell on Windows:

   ```
   $ENV:KUBECONFIG="{0};{1}" -f  $ENV:KUBECONFIG, "$ENV:userprofile\.kube\config-devel"
   ```

10. (Optional) Add the configuration to your shell initialization file so that it is configured when you open a shell.

    - For Bash shells on macOS:

    ```
    echo 'export KUBECONFIG=$KUBECONFIG:~/.kube/config-devel' >> ~/.bash_profile
    ```

    - For Bash shells on Linux:

    ```
    echo 'export KUBECONFIG=$KUBECONFIG:~/.kube/config-devel' >> ~/.bashrc
    ```

    - For PowerShell on Windows:

    ```
    [System.Environment]::SetEnvironmentVariable('KUBECONFIG', $ENV:KUBECONFIG,
      'Machine')
    ```

11. Test your configuration.

    ```
    kubectl get svc
    ```

    Note
    If you receive the error "aws-iam-authenticator": executable file not found in $PATH, your **kubectl** isn't configured for Amazon EKS. For more information, see Installing aws-iam-authenticator (p. 187).
    If you receive any other authorization or resource type errors, see Unauthorized or access denied (kubectl) (p. 288) in the troubleshooting section.

    Output:

    ```
    NAME              TYPE        CLUSTER-IP    EXTERNAL-IP    PORT(S)    AGE
    svc/kubernetes    ClusterIP   10.100.0.1    <none>         443/TCP    1m
    ```

Managing users or IAM roles for your cluster

When you create an Amazon EKS cluster, the IAM entity user or role, such as a federated user that creates the cluster, is automatically granted system:masters permissions in the cluster's RBAC configuration. To grant additional AWS users or roles the ability to interact with your cluster, you must edit the aws-auth ConfigMap within Kubernetes.

Note
For more information about different IAM identities, see Identities (Users, Groups, and Roles) in the *IAM User Guide*. For more information on Kubernetes RBAC configuration, see Using RBAC Authorization. For all ConfigMap settings, see Full Configuration Format on GitHub.

The `aws-auth` ConfigMap is applied as part of the Getting started with Amazon EKS (p. 3) guide which provides a complete end-to-end walkthrough from creating an Amazon EKS cluster to deploying a sample Kubernetes application. It is initially created to allow your worker nodes to join your cluster, but you also use this ConfigMap to add RBAC access to IAM users and roles. If you have not launched worker nodes and applied the `aws-auth` ConfigMap, you can do so with the following procedure.

To apply the `aws-auth` ConfigMap to your cluster

1. Check to see if you have already applied the `aws-auth` ConfigMap.

   ```
   kubectl describe configmap -n kube-system aws-auth
   ```

   If you receive an error stating "`Error from server (NotFound): configmaps "aws-auth" not found`", then proceed with the following steps to apply the stock ConfigMap.

2. Download, edit, and apply the AWS authenticator configuration map.

   a. Download the configuration map:

   ```
   curl -o aws-auth-cm.yaml https://amazon-eks.s3.us-west-2.amazonaws.com/
   cloudformation/2020-05-08/aws-auth-cm.yaml
   ```

   b. Open the file with your favorite text editor. Replace the *<ARN of instance role (not instance profile)>* snippet with the Amazon Resource Name (ARN) of the IAM role that is associated with your worker nodes, and save the file. You can inspect the AWS CloudFormation stack outputs for your worker node groups and look for the following values:

   - **InstanceRoleARN** (for worker node groups that were created with `eksctl`)
   - **NodeInstanceRole** (for worker node groups that were created with Amazon EKS-vended AWS CloudFormation templates in the AWS Management Console)

     Important
     Do not modify any other lines in this file.

   ```
   apiVersion: v1
   kind: ConfigMap
   metadata:
     name: aws-auth
     namespace: kube-system
   data:
     mapRoles: |
       - rolearn: <ARN of instance role (not instance profile)>
         username: system:node:{{EC2PrivateDNSName}}
         groups:
           - system:bootstrappers
           - system:nodes
   ```

   c. Apply the configuration. This command may take a few minutes to finish.

   ```
   kubectl apply -f aws-auth-cm.yaml
   ```

Note
If you receive the error `"aws-iam-authenticator": executable file not found in $PATH`, your **kubectl** isn't configured for Amazon EKS. For more information, see Installing `aws-iam-authenticator` (p. 187).
If you receive any other authorization or resource type errors, see Unauthorized or access denied (`kubectl`) (p. 288) in the troubleshooting section.

3. Watch the status of your nodes and wait for them to reach the `Ready` status.

```
kubectl get nodes --watch
```

To add an IAM user or role to an Amazon EKS cluster

1. Ensure that the AWS credentials that **kubectl** is using are already authorized for your cluster. The IAM user that created the cluster has these permissions by default.

2. Open the `aws-auth` ConfigMap.

```
kubectl edit -n kube-system configmap/aws-auth
```

Note
If you receive an error stating `"Error from server (NotFound): configmaps "aws-auth" not found"`, then use the previous procedure to apply the stock ConfigMap.

Example ConfigMap:

```
# Please edit the object below. Lines beginning with a '#' will be ignored,
# and an empty file will abort the edit. If an error occurs while saving this file will
 be
# reopened with the relevant failures.
#
apiVersion: v1
data:
  mapRoles: |
    - rolearn: arn:aws:iam::111122223333:role/doc-test-worker-nodes-NodeInstanceRole-
WDO5P42N3ETB
      username: system:node:{{EC2PrivateDNSName}}
      groups:
        - system:bootstrappers
        - system:nodes
kind: ConfigMap
metadata:
  annotations:
    kubectl.kubernetes.io/last-applied-configuration: |
      {"apiVersion":"v1","data":{"mapRoles":"- rolearn: arn:aws:iam::111122223333:role/
doc-test-worker-nodes-NodeInstanceRole-WDO5P42N3ETB\n  username: system:node:
{{EC2PrivateDNSName}}\n  groups:\n    - system:bootstrappers\n    -
 system:nodes\n"},"kind":"ConfigMap","metadata":{"annotations":{},"name":"aws-
auth","namespace":"kube-system"}}
  creationTimestamp: 2018-04-04T18:49:10Z
  name: aws-auth
  namespace: kube-system
  resourceVersion: "780"
  selfLink: /api/v1/namespaces/kube-system/configmaps/aws-auth
  uid: dcc31de5-3838-11e8-af26-02e00430057c
```

3. Add your IAM users, roles, or AWS accounts to the configMap.

- **To add an IAM user:** add the user details to the `mapUsers` section of the ConfigMap, under `data`. Add this section if it does not already exist in the file. Each entry supports the following parameters:
  - **userarn**: The ARN of the IAM user to add.
  - **username**: The user name within Kubernetes to map to the IAM user.
  - **groups**: A list of groups within Kubernetes to which the user is mapped to. For more information, see Default Roles and Role Bindings in the Kubernetes documentation.
- **To add an IAM role (for example, for federated users):** add the role details to the `mapRoles` section of the ConfigMap, under `data`. Add this section if it does not already exist in the file. Each entry supports the following parameters:
  - **rolearn**: The ARN of the IAM role to add.
  - **username**: The user name within Kubernetes to map to the IAM role.
  - **groups**: A list of groups within Kubernetes to which the role is mapped. For more information, see Default Roles and Role Bindings in the Kubernetes documentation.

For example, the block below contains:

- A `mapRoles` section that adds the worker node instance role so that worker nodes can register themselves with the cluster.
- A `mapUsers` section with the AWS users `admin` from the default AWS account, and `ops-user` from another AWS account. Both users are added to the `system:masters` group.

```
# Please edit the object below. Lines beginning with a '#' will be ignored,
# and an empty file will abort the edit. If an error occurs while saving this file will
 be
# reopened with the relevant failures.
#
apiVersion: v1
data:
  mapRoles: |
    - rolearn: arn:aws:iam::555555555555:role/devel-worker-nodes-
NodeInstanceRole-74RF4UBDUKL6
      username: system:node:{{EC2PrivateDNSName}}
      groups:
        - system:bootstrappers
        - system:nodes
  mapUsers: |
    - userarn: arn:aws:iam::555555555555:user/admin
      username: admin
      groups:
        - system:masters
    - userarn: arn:aws:iam::111122223333:user/ops-user
      username: ops-user
      groups:
        - system:masters
```

4. Save the file and exit your text editor.

The `eksctl` command line utility

This chapter covers `eksctl`, a simple command line utility for creating and managing Kubernetes clusters on Amazon EKS. The `eksctl` command line utility provides the fastest and easiest way to create a new cluster with worker nodes for Amazon EKS.

For more information and to see the official documentation, visit https://eksctl.io/.

Installing or upgrading `eksctl`

This section helps you to install or upgrade the latest version of the `eksctl` command line utility.

Choose the tab below that best represents your client setup.

macOS

To install or upgrade `eksctl` on macOS using homebrew

The easiest way to get started with Amazon EKS and macOS is by installing `eksctl` with Homebrew. The `eksctl` Homebrew recipe installs `eksctl` and any other dependencies that are required for Amazon EKS, such as `kubectl` and the `aws-iam-authenticator`.

1. If you do not already have Homebrew installed on macOS, install it with the following command.

    ```
    /bin/bash -c "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/master/install.sh)"
    ```

2. Install the Weaveworks Homebrew tap.

    ```
    brew tap weaveworks/tap
    ```

3. Install or upgrade `eksctl`.

    *   Install `eksctl` with the following command:

        ```
        brew install weaveworks/tap/eksctl
        ```

    *   If `eksctl` is already installed, run the following command to upgrade:

        ```
        brew upgrade eksctl && brew link --overwrite eksctl
        ```

4. Test that your installation was successful with the following command.

    ```
    eksctl version
    ```

    > **Note**
    > The `GitTag` version should be at least `0.19.0`. If not, check your terminal output for any installation or upgrade errors, or manually download an archive of the release from https://github.com/weaveworks/eksctl/releases/download/0.19.0/eksctl_Darwin_amd64.tar.gz, extract `eksctl`, and then execute it.

Linux

To install or upgrade `eksctl` on Linux using `curl`

1. Download and extract the latest release of `eksctl` with the following command.

    ```
    curl --silent --location "https://github.com/weaveworks/eksctl/releases/latest/
    download/eksctl_$(uname -s)_amd64.tar.gz" | tar xz -C /tmp
    ```

2. Move the extracted binary to `/usr/local/bin`.

    ```
    sudo mv /tmp/eksctl /usr/local/bin
    ```

3. Test that your installation was successful with the following command.

    ```
    eksctl version
    ```

    > **Note**
    > The `GitTag` version should be at least `0.19.0`. If not, check your terminal output for
    > any installation or upgrade errors, or replace the address in step 1 with `https://`
    > `github.com/weaveworks/eksctl/releases/download/0.19.0/eksctl_`
    > `$(uname -s)_amd64.tar.gz` and complete steps 1-3 again.

Windows

To install or upgrade `eksctl` on Windows using chocolatey

1. If you do not already have Chocolatey installed on your Windows system, see Installing chocolatey.
2. Install or upgrade `eksctl` and the `aws-iam-authenticator`.

    *   Install the binaries with the following command:

        ```
        chocolatey install -y eksctl aws-iam-authenticator
        ```

    *   If they are already installed, run the following command to upgrade:

        ```
        chocolatey upgrade -y eksctl aws-iam-authenticator
        ```

3. Test that your installation was successful with the following command.

    ```
    eksctl version
    ```

    > **Note**
    > The `GitTag` version should be at least `0.19.0`. If not, check your terminal output
    > for any installation or upgrade errors, or manually download an archive of the
    > release from https://github.com/weaveworks/eksctl/releases/download/0.19.0/
    > eksctl_Windows_amd64.zip, extract `eksctl`, and then execute it.

Launch a guest book application

In this topic, you create a sample guest book application to test your Amazon EKS cluster.

> **Note**
> For more information about setting up the guest book example, see https://github.com/
> kubernetes/examples/blob/master/guestbook-go/README.md in the Kubernetes
> documentation.

To create your guest book application

1. Create the Redis master replication controller.

```
kubectl apply -f https://raw.githubusercontent.com/kubernetes/examples/master/
guestbook-go/redis-master-controller.json
```

> **Note**
> If you receive the error "aws-iam-authenticator": executable file not found
> in $PATH, your **kubectl** isn't configured for Amazon EKS. For more information, see
> Installing aws-iam-authenticator (p. 187).
> If you receive any other authorization or resource type errors, see Unauthorized or access
> denied (kubectl) (p. 288) in the troubleshooting section.

Output:

```
replicationcontroller "redis-master" created
```

2. Create the Redis master service.

```
kubectl apply -f https://raw.githubusercontent.com/kubernetes/examples/master/
guestbook-go/redis-master-service.json
```

Output:

```
service "redis-master" created
```

3. Create the Redis slave replication controller.

```
kubectl apply -f https://raw.githubusercontent.com/kubernetes/examples/master/
guestbook-go/redis-slave-controller.json
```

Output:

```
replicationcontroller "redis-slave" created
```

4. Create the Redis slave service.

```
kubectl apply -f https://raw.githubusercontent.com/kubernetes/examples/master/
guestbook-go/redis-slave-service.json
```

Output:

```
service "redis-slave" created
```

5. Create the guestbook replication controller.

```
kubectl apply -f https://raw.githubusercontent.com/kubernetes/examples/master/
guestbook-go/guestbook-controller.json
```

Output:

```
replicationcontroller "guestbook" created
```

6. Create the guestbook service.

```
kubectl apply -f https://raw.githubusercontent.com/kubernetes/examples/master/
guestbook-go/guestbook-service.json
```

Output:

```
service "guestbook" created
```

7. Query the services in your cluster and wait until the **External IP** column for the `guestbook` service is populated.

    Note
    It might take several minutes before the IP address is available.

```
kubectl get services -o wide
```

8. After your external IP address is available, point a web browser to that address at port 3000 to view your guest book. For example, *http://a7a95c2b9e69711e7b1a3022fdcfdf2e-1985673473.region.elb.amazonaws.com:3000*

    Note
    It might take several minutes for DNS to propagate and for your guest book to show up.

Important
If you are unable to connect to the external IP address with your browser, be sure that your corporate firewall is not blocking non-standards ports, like 3000. You can try switching to a guest network to verify.

To clean up your guest book application

When you are finished experimenting with your guest book application, you should clean up the resources that you created for it.

- The following command deletes all of the services and replication controllers for the guest book application:

```
kubectl delete rc/redis-master rc/redis-slave rc/guestbook svc/redis-master svc/redis-slave svc/guestbook
```

Note
If you receive the error `"aws-iam-authenticator": executable file not found in $PATH`, your **kubectl** isn't configured for Amazon EKS. For more information, see Installing `aws-iam-authenticator` (p. 187).
If you receive any other authorization or resource type errors, see Unauthorized or access denied (`kubectl`) (p. 288) in the troubleshooting section.

If you are done with your Amazon EKS cluster, you should delete it and its resources so that you do not incur additional charges. For more information, see Deleting a cluster (p. 43).

Installing the Kubernetes Metrics Server

The Kubernetes Metrics Server is an aggregator of resource usage data in your cluster, and it is not deployed by default in Amazon EKS clusters. The Metrics Server is commonly used by other Kubernetes add ons, such as the Horizontal Pod Autoscaler (p. 139) or the Kubernetes Dashboard (p. 208). For more information, see Resource metrics pipeline in the Kubernetes documentation. This topic explains how to deploy the Kubernetes Metrics Server on your Amazon EKS cluster.

To deploy the Metrics Server

1. Deploy the Metrics Server with the following command:

    ```
    kubectl apply -f https://github.com/kubernetes-sigs/metrics-server/releases/download/
    v0.3.6/components.yaml
    ```

2. Verify that the `metrics-server` deployment is running the desired number of pods with the following command.

    ```
    kubectl get deployment metrics-server -n kube-system
    ```

    Output

    ```
    NAME             READY    UP-TO-DATE    AVAILABLE    AGE
    metrics-server   1/1      1             1            6m
    ```

Control plane metrics with Prometheus

The Kubernetes API server exposes a number of metrics that are useful for monitoring and analysis. These metrics are exposed internally through a metrics endpoint that refers to the `/metrics` HTTP API. Like other endpoints, this endpoint is exposed on the Amazon EKS control plane. This topic explains some of the ways you can use this endpoint to view and analyze what your cluster is doing.

Viewing the raw metrics

To view the raw metrics output, use `kubectl` with the `--raw` flag. This command allows you to pass any HTTP path and returns the raw response.

```
kubectl get --raw /metrics
```

Example output:

```
...
# HELP rest_client_requests_total Number of HTTP requests, partitioned by status code,
 method, and host.
# TYPE rest_client_requests_total counter
rest_client_requests_total{code="200",host="127.0.0.1:21362",method="POST"} 4994
rest_client_requests_total{code="200",host="127.0.0.1:443",method="DELETE"} 1
rest_client_requests_total{code="200",host="127.0.0.1:443",method="GET"} 1.326086e+06
rest_client_requests_total{code="200",host="127.0.0.1:443",method="PUT"} 862173
rest_client_requests_total{code="404",host="127.0.0.1:443",method="GET"} 2
rest_client_requests_total{code="409",host="127.0.0.1:443",method="POST"} 3
rest_client_requests_total{code="409",host="127.0.0.1:443",method="PUT"} 8
# HELP ssh_tunnel_open_count Counter of ssh tunnel total open attempts
# TYPE ssh_tunnel_open_count counter
ssh_tunnel_open_count 0
# HELP ssh_tunnel_open_fail_count Counter of ssh tunnel failed open attempts
# TYPE ssh_tunnel_open_fail_count counter
ssh_tunnel_open_fail_count 0
```

This raw output returns verbatim what the API server exposes. These metrics are represented in a Prometheus format. This format allows the API server to expose different metrics broken down by line. Each line includes a metric name, tags, and a value.

```
metric_name{"tag"="value"[,...]} value
```

While this endpoint is useful if you are looking for a specific metric, you typically want to analyze these metrics over time. To do this, you can deploy Prometheus into your cluster. Prometheus is a monitoring and time series database that scrapes exposed endpoints and aggregates data, allowing you to filter, graph, and query the results.

Deploying Prometheus

This topic helps you deploy Prometheus into your cluster with Helm V3. If you already have Helm installed, you can check your version with the `helm version` command. Helm is a package manager

for Kubernetes clusters. For more information about Helm and how to install it, see Using Helm with Amazon EKS (p. 207).

After you configure Helm for your Amazon EKS cluster, you can use it to deploy Prometheus with the following steps.

To deploy Prometheus using Helm

1. Create a Prometheus namespace.

   ```
   kubectl create namespace prometheus
   ```

2. Deploy Prometheus.

   ```
   helm install prometheus stable/prometheus \
       --namespace prometheus \
       --set
    alertmanager.persistentVolume.storageClass="gp2",server.persistentVolume.storageClass="gp2"
   ```

3. Verify that all of the pods in the `prometheus` namespace are in the `READY` state.

   ```
   kubectl get pods -n prometheus
   ```

   Output:

   ```
   NAME                                            READY    STATUS     RESTARTS    AGE
   prometheus-alertmanager-59b4c8c744-r7bgp        1/2      Running    0           48s
   prometheus-kube-state-metrics-7cfd87cf99-jkz2f  1/1      Running    0           48s
   prometheus-node-exporter-jcjqz                  1/1      Running    0           48s
   prometheus-node-exporter-jxv2h                  1/1      Running    0           48s
   prometheus-node-exporter-vbdks                  1/1      Running    0           48s
   prometheus-pushgateway-76c444b68c-82tnw         1/1      Running    0           48s
   prometheus-server-775957f748-mmht9              1/2      Running    0           48s
   ```

4. Use `kubectl` to port forward the Prometheus console to your local machine.

   ```
   kubectl --namespace=prometheus port-forward deploy/prometheus-server 9090
   ```

5. Point a web browser to localhost:9090 to view the Prometheus console.

6. Choose a metric from the **- insert metric at cursor** menu, then choose **Execute**. Choose the **Graph** tab to show the metric over time. The following image shows `container_memory_usage_bytes` over time.
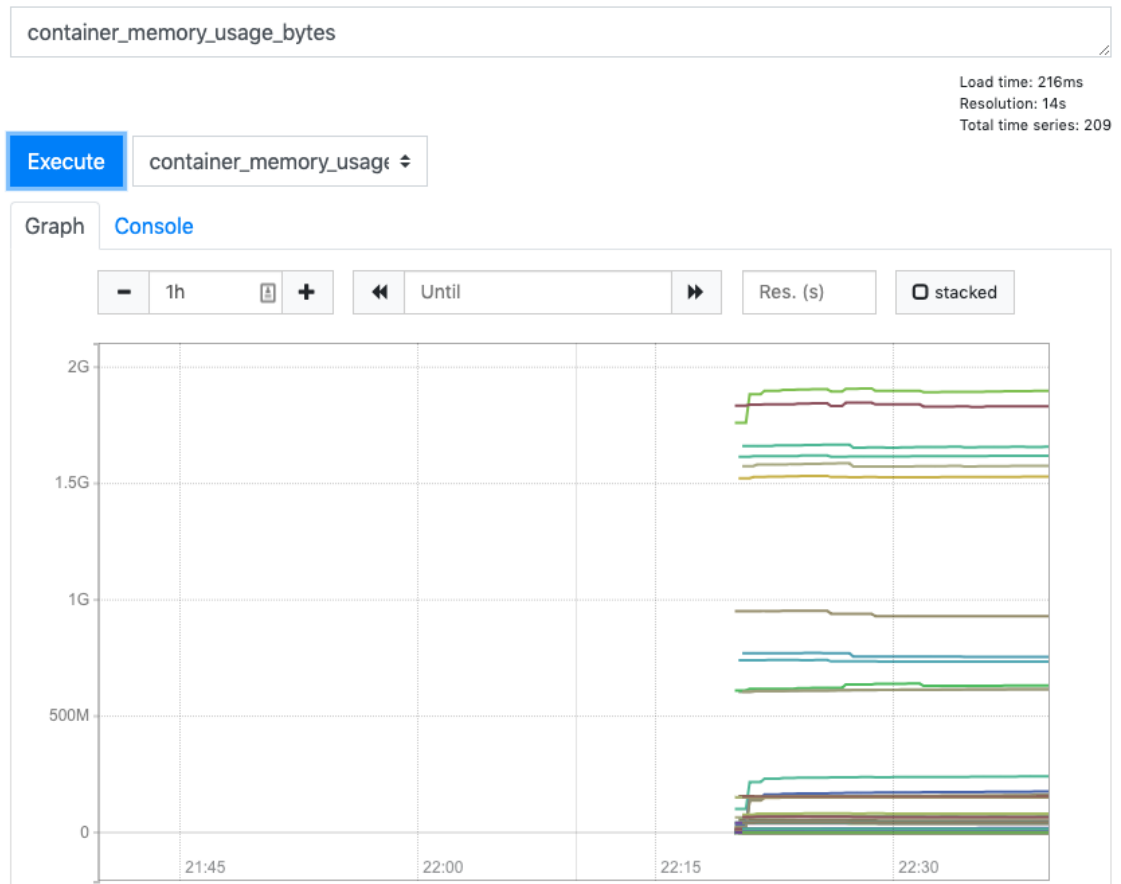
7. From the top navigation bar, choose **Status**, then **Targets**.
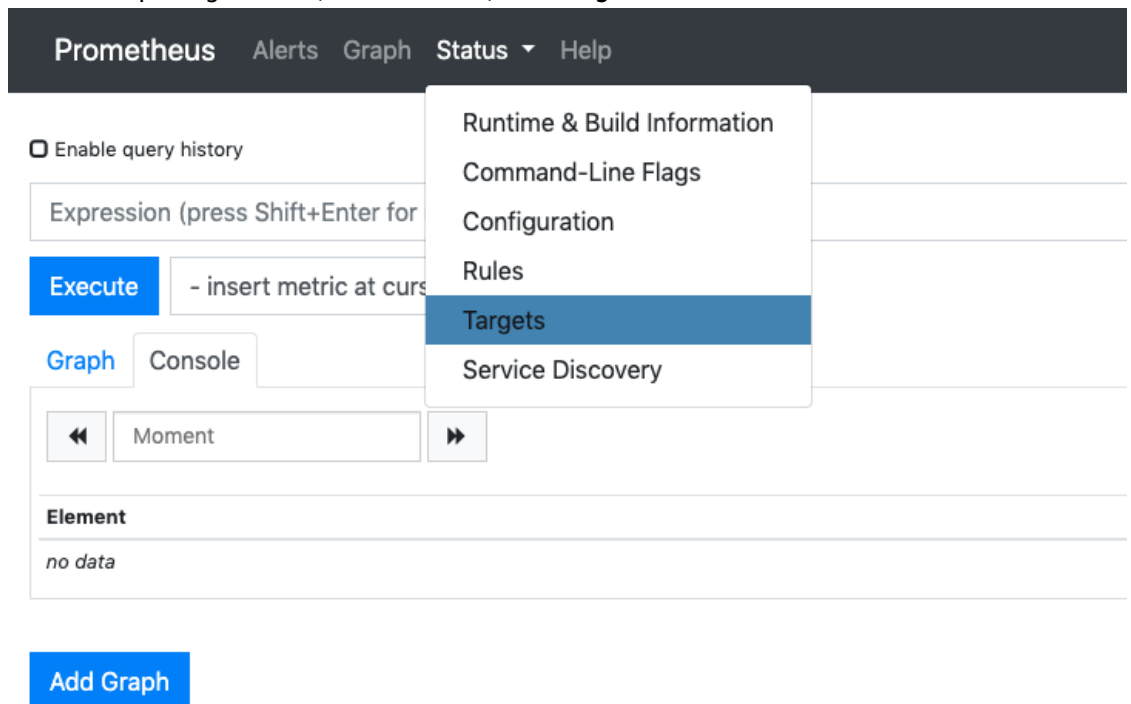
All of the Kubernetes endpoints that are connected to Prometheus using service discovery are displayed.

Using Helm with Amazon EKS

The Helm package manager for Kubernetes helps you install and manage applications on your Kubernetes cluster. For more information, see the Helm documentation. This topic helps you install and run the Helm binaries so that you can install and manage charts using the Helm CLI on your local system.

> **Important**
> Before you can install Helm charts on your Amazon EKS cluster, you must configure **kubectl** to work for Amazon EKS. If you have not already done this, see Installing aws-iam-authenticator (p. 187) and Create a `kubeconfig` for Amazon EKS (p. 190) before proceeding. If the following command succeeds for your cluster, you're properly configured.

```
kubectl get svc
```

To install the Helm binaries on your local system

1. Run the appropriate command for your client operating system.

   - If you're using macOS with Homebrew, install the binaries with the following command.

   ```
   brew install helm
   ```

   - If you're using Windows with Chocolatey, install the binaries with the following command.

   ```
   choco install kubernetes-helm
   ```

   - If you're using Linux, install the binaries with the following commands.

   ```
   curl https://raw.githubusercontent.com/helm/helm/master/scripts/get-helm-3 >
    get_helm.sh
   chmod 700 get_helm.sh
   ./get_helm.sh
   ```

2. To pick up the new binary in your `PATH`, Close your current terminal window and open a new one.
3. Confirm that Helm is running with the following command.

```
helm help
```

4. At this point, you can run any Helm commands (such as `helm install chart_name`) to install, modify, delete, or query Helm charts in your cluster. If you're new to Helm and don't have a specific chart to install, you can:

   - Experiment by installing an example chart. See Install an example chart in the Helm Quickstart guide.
   - Install an Amazon EKS chart from the eks-charts GitHub repo or from Helm Hub.

Tutorial: Deploy the Kubernetes Dashboard (web UI)

This tutorial guides you through deploying the Kubernetes Dashboard to your Amazon EKS cluster, complete with CPU and memory metrics. It also helps you to create an Amazon EKS administrator service account that you can use to securely connect to the dashboard to view and control your cluster.
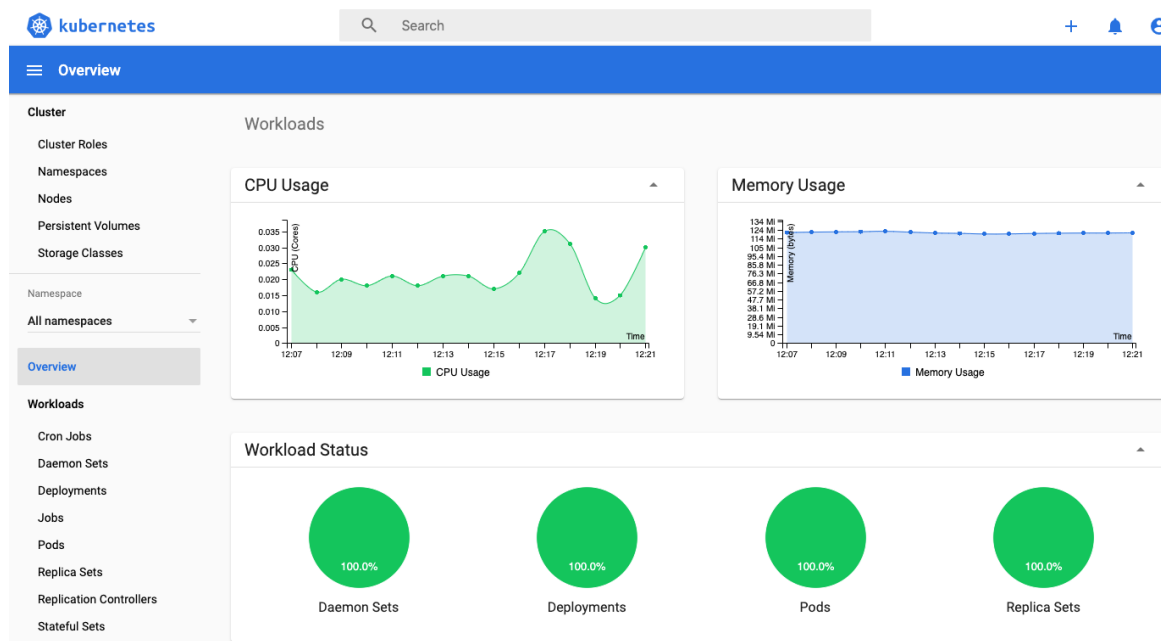
Prerequisites

This tutorial assumes the following:

- You have created an Amazon EKS cluster by following the steps in Getting started with Amazon EKS (p. 3).
- The security groups for your control plane elastic network interfaces and worker nodes follow the recommended settings in Amazon EKS security group considerations (p. 157).
- You are using a **kubectl** client that is configured to communicate with your Amazon EKS cluster (p. 15).

Step 1: Deploy the Kubernetes Metrics Server

The Kubernetes Metrics Server is an aggregator of resource usage data in your cluster, and it is not deployed by default in Amazon EKS clusters. The Kubernetes Dashboard uses the metrics server to gather metrics for your cluster, such as CPU and memory usage over time.

To deploy the Metrics Server

1. Deploy the Metrics Server with the following command:

```
kubectl apply -f https://github.com/kubernetes-sigs/metrics-server/releases/download/
v0.3.6/components.yaml
```

2. Verify that the `metrics-server` deployment is running the desired number of pods with the following command.

```
kubectl get deployment metrics-server -n kube-system
```

Output

```
NAME             READY    UP-TO-DATE    AVAILABLE    AGE
metrics-server   1/1      1             1            6m
```

Step 2: Deploy the dashboard

Use the following command to deploy the Kubernetes Dashboard.

```
kubectl apply -f https://raw.githubusercontent.com/kubernetes/dashboard/v2.0.0-beta8/aio/
deploy/recommended.yaml
```

Output:

```
namespace/kubernetes-dashboard created
serviceaccount/kubernetes-dashboard created
service/kubernetes-dashboard created
secret/kubernetes-dashboard-certs created
secret/kubernetes-dashboard-csrf created
secret/kubernetes-dashboard-key-holder created
configmap/kubernetes-dashboard-settings created
role.rbac.authorization.k8s.io/kubernetes-dashboard created
clusterrole.rbac.authorization.k8s.io/kubernetes-dashboard created
rolebinding.rbac.authorization.k8s.io/kubernetes-dashboard created
clusterrolebinding.rbac.authorization.k8s.io/kubernetes-dashboard created
deployment.apps/kubernetes-dashboard created
service/dashboard-metrics-scraper created
deployment.apps/dashboard-metrics-scraper created
```

Step 3: Create an `eks-admin` service account and cluster role binding

By default, the Kubernetes Dashboard user has limited permissions. In this section, you create an `eks-admin` service account and cluster role binding that you can use to securely connect to the dashboard with admin-level permissions. For more information, see Managing Service Accounts in the Kubernetes documentation.

To create the `eks-admin` service account and cluster role binding

> **Important**
> The example service account created with this procedure has full `cluster-admin` (superuser) privileges on the cluster. For more information, see Using RBAC authorization in the Kubernetes documentation.

1. Create a file called `eks-admin-service-account.yaml` with the text below. This manifest defines a service account and cluster role binding called `eks-admin`.

```
apiVersion: v1
kind: ServiceAccount
metadata:
  name: eks-admin
  namespace: kube-system
---
apiVersion: rbac.authorization.k8s.io/v1beta1
kind: ClusterRoleBinding
metadata:
  name: eks-admin
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: cluster-admin
subjects:
- kind: ServiceAccount
  name: eks-admin
  namespace: kube-system
```

2. Apply the service account and cluster role binding to your cluster.

```
kubectl apply -f eks-admin-service-account.yaml
```

Output:

```
serviceaccount "eks-admin" created
clusterrolebinding.rbac.authorization.k8s.io "eks-admin" created
```

Step 4: Connect to the dashboard

Now that the Kubernetes Dashboard is deployed to your cluster, and you have an administrator service account that you can use to view and control your cluster, you can connect to the dashboard with that service account.

To connect to the Kubernetes dashboard

1. Retrieve an authentication token for the `eks-admin` service account. Copy the `<authentication_token>` value from the output. You use this token to connect to the dashboard.

```
kubectl -n kube-system describe secret $(kubectl -n kube-system get secret | grep eks-admin | awk '{print $1}')
```

Output:

```
Name:         eks-admin-token-b5zv4
Namespace:    kube-system
Labels:       <none>
Annotations:  kubernetes.io/service-account.name=eks-admin
              kubernetes.io/service-account.uid=bcfe66ac-39be-11e8-97e8-026dce96b6e8

Type:  kubernetes.io/service-account-token

Data
====
```
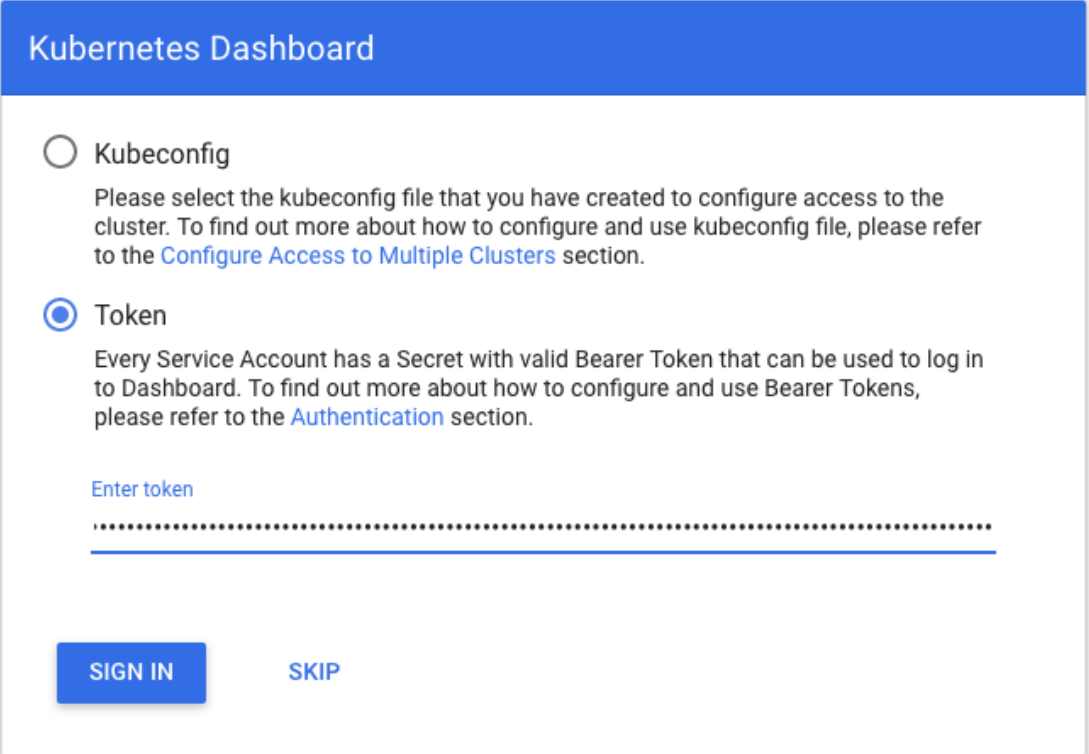
```
ca.crt:      1025 bytes
namespace:   11 bytes
token:       <authentication_token>
```

2. Start the **kubectl proxy**.

```
kubectl proxy
```

3. To access the dashboard endpoint, open the following link with a web browser: http://
    localhost:8001/api/v1/namespaces/kubernetes-dashboard/services/https:kubernetes-dashboard:/
    proxy/#!/login.

4. Choose **Token**, paste the `<authentication_token>` output from the previous command into the
    Token field, and choose **SIGN IN**.

> **Note**
> It may take a few minutes before CPU and memory metrics appear in the dashboard.

Step 5: Next steps

After you have connected to your Kubernetes Dashboard, you can view and control your cluster using
your `eks-admin` service account. For more information about using the dashboard, see the project
documentation on GitHub.

Getting started with AWS App Mesh and Kubernetes

AWS App Mesh is a service mesh based on the Envoy proxy that helps you monitor and control services. App Mesh standardizes how your services communicate, giving you end-to-end visibility into and helping to ensure high-availability for your applications. App Mesh gives you consistent visibility and network traffic controls for every service in an application. For more information, see the AWS App Mesh User Guide.

This topic helps you use AWS App Mesh with an actual service that is running on Kubernetes. You can either integrate Kubernetes with App Mesh resources by completing the steps in this topic or by installing the App Mesh Kubernetes integration components. The integration components automatically complete the tasks in this topic for you, enabling you to integrate with App Mesh directly from Kubernetes. For more information, see Configure App Mesh Integration with Kubernetes.

Scenario

To illustrate how to use App Mesh with Kubernetes, assume that you have an application with the following characteristics:

- Includes two services named `serviceA` and `serviceB`.
- Both services are registered to a namespace named `apps.local`.
- `ServiceA` communicates with `serviceB` over HTTP/2, port 80.
- You've already deployed version 2 of `serviceB` and registered it with the name `serviceBv2` in the `apps.local` namespace.

You have the following requirements:

- You want to send 75 percent of the traffic from `serviceA` to `serviceB` and 25 percent of the traffic to `serviceBv2` to ensure that `serviceBv2` is bug free before you send 100 percent of the traffic from `serviceA` to it.
- You want to be able to easily adjust the traffic weighting so that 100 percent of the traffic goes to `serviceBv2` once it's proven to be reliable. Once all traffic is being sent to `serviceBv2`, you want to deprecate `serviceB`.
- You don't want to have to change any existing application code or service discovery registration for your actual services to meet the previous requirements.

To meet your requirements, you've decided to create an App Mesh service mesh with virtual services, virtual nodes, a virtual router, and a route. After implementing your mesh, you update the pod specs for your services to use the Envoy proxy. Once updated, your services communicate with each other through the Envoy proxy rather than directly with each other.

Prerequisites

App Mesh supports Linux services that are registered with DNS, AWS Cloud Map, or both. To use this getting started guide, we recommend that you have three existing services that are registered with DNS. You can create a service mesh and its resources even if the services don't exist, but you can't use the mesh until you have deployed actual services.

If you don't already have Kubernetes running, then you can create an Amazon EKS cluster. For more information, see Getting Started with Amazon EKS using `eksctl`. If you don't already have some services running on Kubernetes, you can deploy a test application. For more information, see Launch a Guest Book Application.

The remaining steps assume that the actual services are named `serviceA`, `serviceB`, and `serviceBv2` and that all services are discoverable through a namespace named `apps.local`.

Step 1: Create a mesh and virtual service

A service mesh is a logical boundary for network traffic between the services that reside within it. For more information, see Service Meshes in the *AWS App Mesh User Guide*. A virtual service is an abstraction of an actual service. For more information, see Virtual Services in the *AWS App Mesh User Guide*.

Create the following resources:

- A mesh named `apps`, since all of the services in the scenario are registered to the `apps.local` namespace.
- A virtual service named `serviceb.apps.local`, since the virtual service represents a service that is discoverable with that name, and you don't want to change your code to reference another name. A virtual service named `servicea.apps.local` is added in a later step.

You can use the AWS Management Console or the AWS CLI version 1.18.16 or higher to complete the following steps. If using the AWS CLI, use the `aws --version` command to check your installed AWS CLI version. If you don't have version 1.18.16 or higher installed, you must install or update the AWS CLI. Select the tab for the tool that you want to use.

AWS Management Console

1. Open the App Mesh console first-run wizard at https://console.aws.amazon.com/appmesh/get-started.
2. For **Mesh name**, enter `apps`.
3. For **Virtual service name**, enter `serviceb.apps.local`.
4. To continue, choose **Next**.

AWS CLI

1. Create a mesh with the `create-mesh` command.

   ```
   aws appmesh create-mesh --mesh-name apps
   ```

2. Create a virtual service with the `create-virtual-service` command.

   ```
   aws appmesh create-virtual-service --mesh-name apps --virtual-service-name
    serviceb.apps.local --spec {}
   ```

Step 2: Create a virtual node

A virtual node acts as a logical pointer to an actual service. For more information, see Virtual Nodes *in the AWS App Mesh User Guide*.

Create a virtual node named `serviceB`, since one of the virtual nodes represents the actual service named `serviceB`. The actual service that the virtual node represents is discoverable through `DNS` with a hostname of `serviceb.apps.local`. Alternately, you can discover actual services using AWS Cloud Map. The virtual node will listen for traffic using the HTTP/2 protocol on port 80. Other protocols are also supported, as are health checks. You will create virtual nodes for `serviceA` and `serviceBv2` in a later step.

AWS Management Console

1. For **Virtual node name**, enter `serviceB`.

2. For **Service discovery method**, choose `DNS` and enter `serviceb.apps.local` for **DNS hostname**.

3. Under **Listener**, enter `80` for **Port** and choose `http2` for **Protocol**.

4. To continue, choose **Next**.

AWS CLI

1. Create a file named `create-virtual-node-serviceb.json` with the following contents:

```
{
    "meshName": "apps",
    "spec": {
        "listeners": [
            {
                "portMapping": {
                    "port": 80,
                    "protocol": "http2"
                }
            }
        ],
        "serviceDiscovery": {
            "dns": {
                "hostname": "serviceB.apps.local"
            }
        }
    },
    "virtualNodeName": "serviceB"
}
```

2. Create the virtual node with the create-virtual-node command using the JSON file as input.

```
aws appmesh create-virtual-node --cli-input-json file://create-virtual-node-
serviceb.json
```

Step 3: Create a virtual router and route

Virtual routers route traffic for one or more virtual services within your mesh. For more information, see Virtual Routers and Routes in the *AWS App Mesh User Guide*.

Create the following resources:

- A virtual router named `serviceB`, since the `serviceB.apps.local` virtual service doesn't initiate outbound communication with any other service. Remember that the virtual service that you created previously is an abstraction of your actual `serviceb.apps.local` service. The virtual service sends traffic to the virtual router. The virtual router will listen for traffic using the HTTP/2 protocol on port 80. Other protocols are also supported.

- A route named `serviceB`. It will route 100 percent of its traffic to the `serviceB` virtual node. You'll change the weight in a later step once you've added the `serviceBv2` virtual node. Though not covered in this guide, you can add additional filter criteria for the route and add a retry policy to cause the Envoy proxy to make multiple attempts to send traffic to a virtual node when it experiences a communication problem.

AWS Management Console

1. For **Virtual router name,** enter **serviceB**.

2. Under **Listener**, specify **80** for **Port** and choose `http2` for **Protocol**.

3. For **Route name**, enter **serviceB**.

4. For **Route type**, choose `http2`.

5. For **Virtual node name**, select `serviceB` and enter **100** for **Weight**.

6. To continue, choose **Next**.

AWS CLI

1. Create a virtual router.

   a. Create a file named `create-virtual-router.json` with the following contents:

   ```
   {
       "meshName": "apps",
       "spec": {
           "listeners": [
               {
                   "portMapping": {
                       "port": 80,
                       "protocol": "http2"
                   }
               }
           ]
       },
       "virtualRouterName": "serviceB"
   }
   ```

   b. Create the virtual router with the create-virtual-router command using the JSON file as input.

   ```
   aws appmesh create-virtual-router --cli-input-json file://create-virtual-router.json
   ```

2. Create a route.

   a. Create a file named `create-route.json` with the following contents:

   ```
   {
       "meshName" : "apps",
       "routeName" : "serviceB",
       "spec" : {
           "httpRoute" : {
               "action" : {
                   "weightedTargets" : [
                       {
                           "virtualNode" : "serviceB",
                           "weight" : 100
                       }
                   ]
   ```

```
            },
            "match" : {
                "prefix" : "/"
            }
        }
    },
    "virtualRouterName" : "serviceB"
}
```

b. Create the route with the create-route command using the JSON file as input.

```
aws appmesh create-route --cli-input-json file://create-route.json
```

Step 4: review and create

Review the settings against the previous instructions.

AWS Management Console

Choose **Edit** if you need to make any changes in any section. Once you're satisfied with the settings, choose **Create mesh service**.

AWS CLI

Review the settings of the mesh you created with the describe-mesh command.

```
aws appmesh describe-mesh --mesh-name apps
```

Review the settings of the virtual service that you created with the describe-virtual-service command.

```
aws appmesh describe-virtual-service --mesh-name apps --virtual-service-name
  serviceb.apps.local
```

Review the settings of the virtual node that you created with the describe-virtual-node command.

```
aws appmesh describe-virtual-node --mesh-name apps --virtual-node-name serviceB
```

Review the settings of the virtual router that you created with the describe-virtual-router command.

```
aws appmesh describe-virtual-router --mesh-name apps --virtual-router-name serviceB
```

Review the settings of the route that you created with the describe-route command.

```
aws appmesh describe-route --mesh-name apps \
    --virtual-router-name serviceB  --route-name serviceB
```

Step 5: Create additional resources

To complete the scenario, you need to:

- Create one virtual node named serviceBv2 and another named serviceA. Both virtual nodes listen for requests over HTTP/2 port 80. For the serviceA virtual node, configure a backend of

serviceb.apps.local, since all outbound traffic from the serviceA virtual node is sent to the virtual service named serviceb.apps.local. Though not covered in this guide, you can also specify a file path to write access logs to for a virtual node.

• Create one additional virtual service named servicea.apps.local, which will send all traffic directly to the serviceA virtual node.

• Update the serviceB route that you created in a previous step to send 75 percent of its traffic to the serviceB virtual node and 25 percent of its traffic to the serviceBv2 virtual node. Over time, you can continue to modify the weights until serviceBv2 receives 100 percent of the traffic. Once all traffic is sent to serviceBv2, you can deprecate the serviceB virtual node and actual service. As you change weights, your code doesn't require any modification, because the serviceb.apps.local virtual and actual service names don't change. Recall that the serviceb.apps.local virtual service sends traffic to the virtual router, which routes the traffic to the virtual nodes. The service discovery names for the virtual nodes can be changed at any time.

AWS Management Console

1. In the left navigation pane, select **Meshes**.
2. Select the apps mesh that you created in a previous step.
3. In the left navigation pane, select **Virtual nodes**.
4. Choose **Create virtual node**.
5. For **Virtual node name**, enter **serviceBv2**, for **Service discovery method**, choose DNS, and for **DNS hostname**, enter **servicebv2.apps.local**.
6. For **Listener**, enter **80** for **Port** and select http2 for **Protocol**.
7. Choose **Create virtual node**.
8. Choose **Create virtual node** again, and enter **serviceA** for the **Virtual node name**, for **Service discovery method**, choose DNS, and for **DNS hostname**, enter **servicea.apps.local**.
9. Expand **Additional configuration**.
10. Select **Add backend**. Enter **serviceb.apps.local**.
11. Enter **80** for **Port**, choose http2 for **Protocol**, and then choose **Create virtual node**.
12. In the left navigation pane, select **Virtual routers** and then select the serviceB virtual router from the list.
13. Under **Routes**, select the route named ServiceB that you created in a previous step, and choose **Edit**.
14. Under **Virtual node name**, change the value of **Weight** for serviceB to **75**.
15. Choose **Add target**, choose serviceBv2 from the drop-down list, and set the value of **Weight** to **25**.
16. Choose **Save**.
17. In the left navigation pane, select **Virtual services** and then choose **Create virtual service**.
18. Enter **servicea.apps.local** for **Virtual service name**, select Virtual node for **Provider**, select serviceA for **Virtual node**, and then choose **Create virtual service.**

AWS CLI

1. Create the serviceBv2 virtual node.

   a. Create a file named create-virtual-node-servicebv2.json with the following contents:

   ```
   {
       "meshName": "apps",
       "spec": {
           "listeners": [
   ```

```
            {
                "portMapping": {
                    "port": 80,
                    "protocol": "http2"
                }
            }
        ],
        "serviceDiscovery": {
            "dns": {
                "hostname": "serviceBv2.apps.local"
            }
        }
    },
    "virtualNodeName": "serviceBv2"
}
```

b. Create the virtual node.

```
aws appmesh create-virtual-node --cli-input-json file://create-virtual-node-
servicebv2.json
```

2. Create the `serviceA` virtual node.

a. Create a file named `create-virtual-node-servicea.json` with the following
    contents:

```
{
    "meshName" : "apps",
    "spec" : {
        "backends" : [
            {
                "virtualService" : {
                    "virtualServiceName" : "serviceb.apps.local"
                }
            }
        ],
        "listeners" : [
            {
                "portMapping" : {
                    "port" : 80,
                    "protocol" : "http2"
                }
            }
        ],
        "serviceDiscovery" : {
            "dns" : {
                "hostname" : "servicea.apps.local"
            }
        }
    },
    "virtualNodeName" : "serviceA"
}
```

b. Create the virtual node.

```
aws appmesh create-virtual-node --cli-input-json file://create-virtual-node-
servicea.json
```

3. Update the `serviceb.apps.local` virtual service that you created in a previous step to send
    its traffic to the `serviceB` virtual router. When the virtual service was originally created, it
    didn't send traffic anywhere, since the `serviceB` virtual router hadn't been created yet.

a. Create a file named `update-virtual-service.json` with the following contents:

```
{
    "meshName" : "apps",
    "spec" : {
        "provider" : {
            "virtualRouter" : {
                "virtualRouterName" : "serviceB"
            }
        }
    },
    "virtualServiceName" : "serviceb.apps.local"
}
```

b.   Update the virtual service with the update-virtual-service command.

```
aws appmesh update-virtual-service --cli-input-json file://update-virtual-
service.json
```

4.   Update the `serviceB` route that you created in a previous step.

a.   Create a file named `update-route.json` with the following contents:

```
{
    "meshName" : "apps",
    "routeName" : "serviceB",
    "spec" : {
        "http2Route" : {
            "action" : {
                "weightedTargets" : [
                    {
                        "virtualNode" : "serviceB",
                        "weight" : 75
                    },
                    {
                        "virtualNode" : "serviceBv2",
                        "weight" : 25
                    }
                ]
            },
            "match" : {
                "prefix" : "/"
            }
        }
    },
    "virtualRouterName" : "serviceB"
}
```

b.   Update the route with the update-route command.

```
aws appmesh update-route --cli-input-json file://update-route.json
```

5.   Create the `serviceA` virtual service.

a.   Create a file named `create-virtual-servicea.json` with the following contents:

```
{
    "meshName" : "apps",
    "spec" : {
        "provider" : {
            "virtualNode" : {
                "virtualNodeName" : "serviceA"
            }
```

```
            }
        },
        "virtualServiceName" : "servicea.apps.local"
    }
```

b. Create the virtual service.

```
aws appmesh create-virtual-service --cli-input-json file://create-virtual-
servicea.json
```

Mesh summary

Before you created the service mesh, you had three actual services named `servicea.apps.local`,
`serviceb.apps.local`, and `servicebv2.apps.local`. In addition to the actual services, you now
have a service mesh that contains the following resources that represent the actual services:

*   Two virtual services. The proxy sends all traffic from the `servicea.apps.local` virtual service to the
    `serviceb.apps.local` virtual service through a virtual router.
*   Three virtual nodes named `serviceA`, `serviceB`, and `serviceBv2`. The Envoy proxy uses the service
    discovery information configured for the virtual nodes to look up the IP addresses of the actual
    services.
*   One virtual router with one route that instructs the Envoy proxy to route 75 percent of inbound traffic
    to the `serviceB` virtual node and 25 percent of the traffic to the `serviceBv2` virtual node.

Step 6: Update services

After creating your mesh, you need to complete the following tasks:

*   Authorize the Envoy proxy that you deploy with each Kubernetes pod to read the configuration
    of one or more virtual nodes. For more information about how to authorize the proxy, see Proxy
    authorization.
*   Update each of your existing Kubernetes pod specs to use the Envoy proxy.

App Mesh vends the following custom container images that you must add to your Kubernetes pod
specifications:

*   Specify one of the following App Mesh Envoy container images, depending on which region you want
    to pull the image from.
    *   All supported Regions other than `me-south-1` and `ap-east-1`. You can replace *us-west-2* with
        any Region other than `me-south-1` and `ap-east-1`.

        ```
        840364872350.dkr.ecr.us-west-2.amazonaws.com/aws-appmesh-envoy:v1.12.3.0-prod
        ```

    *   `me-south-1` Region:

        ```
        772975370895.dkr.ecr.me-south-1.amazonaws.com/aws-appmesh-envoy:v1.12.3.0-prod
        ```

    *   `ap-east-1` Region:

        ```
        856666278305.dkr.ecr.ap-east-1.amazonaws.com/aws-appmesh-envoy:v1.12.3.0-prod
        ```

Envoy uses the configuration defined in the App Mesh control plane to determine where to send your
application traffic.

You must use the App Mesh Envoy container image until the Envoy project team merges changes that support App Mesh. For additional details, see the GitHub roadmap issue.

- App Mesh proxy route manager – `111345817488.dkr.ecr.us-west-2.amazonaws.com/aws-appmesh-proxy-route-manager:v2`. The route manager sets up a pod's network namespace with `iptables` rules that route ingress and egress traffic through Envoy.

Update each pod specification in your application to include these containers, as shown in the following example. Once updated, deploy the new specifications to update your services and start using App Mesh with your Kubernetes application. The following example shows updating the `serviceB` pod specification, that aligns to the scenario. To complete the scenario, you also need to update the `serviceBv2` and `serviceA` pod specifications by changing the values appropriately. For your own applications, substitute your mesh name and virtual node name for the `APPMESH_VIRTUAL_NODE_NAME` value, and add a list of ports that your application listens on for the `APPMESH_APP_PORTS` value. Substitute the Amazon EC2 instance AWS Region for the `AWS_REGION` value.

Example Kubernetes pod spec

```
spec:
  containers:
    - name: envoy
      image: 840364872350.dkr.ecr.us-west-2.amazonaws.com/aws-appmesh-envoy:v1.12.3.0-prod
      securityContext:
        runAsUser: 1337
      env:
        - name: "APPMESH_VIRTUAL_NODE_NAME"
          value: "mesh/apps/virtualNode/serviceB"
        - name: "ENVOY_LOG_LEVEL"
          value: "info"
        - name: "AWS_REGION"
          value: "aws_region_name"
  initContainers:
    - name: proxyinit
      image: 111345817488.dkr.ecr.us-west-2.amazonaws.com/aws-appmesh-proxy-route-
manager:v2
      securityContext:
        capabilities:
          add:
            - NET_ADMIN
      env:
        - name: "APPMESH_START_ENABLED"
          value: "1"
        - name: "APPMESH_IGNORE_UID"
          value: "1337"
        - name: "APPMESH_ENVOY_INGRESS_PORT"
          value: "15000"
        - name: "APPMESH_ENVOY_EGRESS_PORT"
          value: "15001"
        - name: "APPMESH_APP_PORTS"
          value: "application_port_list"
        - name: "APPMESH_EGRESS_IGNORED_IP"
          value: "169.254.169.254"
        - name: "APPMESH_EGRESS_IGNORED_PORTS"
          value: "22"
```

Tutorial: Configure App Mesh integration with Kubernetes

AWS App Mesh is a service mesh based on the Envoy proxy that makes it easy to monitor and control services. App Mesh standardizes how your services communicate, giving you end-to-end visibility and helping to ensure high availability for your applications.

App Mesh gives you consistent visibility and network traffic controls for every service in an application. For more information, see the App Mesh User Guide.

When you integrate AWS App Mesh with Kubernetes, you manage App Mesh resources, such as virtual services and virtual nodes, through Kubernetes. You also automatically add the App Mesh sidecar container images to Kubernetes pod specifications. This tutorial guides you through the installation of the following open source components that enable this integration:

- **App Mesh controller for Kubernetes** – The controller is accompanied by the deployment of three Kubernetes custom resource definitions: `mesh`, `virtual service`, and `virtual node`. The controller watches for creation, modification, and deletion of the custom resources and makes changes to the corresponding App Mesh `mesh`, `virtual service` (including `virtual router` and `route`), and `virtual node` resources through the App Mesh API. To learn more or contribute to the controller, see the GitHub project.
- **App Mesh sidecar injector for Kubernetes** – The injector installs as a webhook and injects the following containers into Kubernetes pods that are running in specific, labeled namespaces. To learn more or contribute, see the GitHub project.
  - **App Mesh Envoy proxy** –Envoy uses the configuration defined in the App Mesh control plane to determine where to send your application traffic..
  - **App Mesh proxy route manager** – The route manager sets up a pod's network namespace with `iptables` rules that route ingress and egress traffic through Envoy.

> *The features discussed in this topic are available as an open-source beta. This means that these features are well tested. Support for the features will not be dropped, though details may change. If the schema or schematics of a feature changes, instructions for migrating to the next version will be provided. This migration may require deleting, editing, and re-creating Kubernetes API objects.*

Prerequisites

- An existing understanding of App Mesh concepts. For more information, see What is AWS App Mesh.
- An existing Kubernetes cluster running version 1.13 or later. If you don't have an existing cluster, you can deploy one using the Getting Started with Amazon EKS guide.
- The AWS CLI version 1.18.16 or later installed. To install or upgrade the, see Installing the AWS CLI.
- A `kubectl` client that is configured to communicate with your Kubernetes cluster. If you're using Amazon Elastic Kubernetes Service, you can use the instructions for installing `kubectl` and configuring a `kubeconfig` file.
- Helm version 3.0 or later installed. If you don't have Helm installed, you can install it by completing the instructions in Using Helm with Amazon EKS.

Step 1: Install the integration components

Install the integration components one time to each cluster that hosts pods that you want to use with App Mesh

To install the integration components

1. Add the `eks-charts` repository to Helm.

    ```
    helm repo add eks https://aws.github.io/eks-charts
    ```

2. Install the App Mesh Kubernetes custom resource definitions (CRD).

    ```
    kubectl apply -k https://github.com/aws/eks-charts/stable/appmesh-controller/crds?
    ref=master
    ```

3. Create a Kubernetes namespace for the controller.

    ```
    kubectl create ns appmesh-system
    ```

4. Set the following variables. Replace *cluster-name* and *region-code* with the values for your existing cluster.

    ```
    export CLUSTER_NAME=cluster-name
    export AWS_REGION=region-code
    ```

5. Create an OpenID Connect (OIDC) identity provider for your cluster. If you don't have `eksctl` installed, you can install it with the instructions in Installing or Upgrading `eksctl`. If you'd prefer to create the provider using the console, see Enabling IAM Roles for Service Accounts on your Cluster.

    ```
    eksctl utils associate-iam-oidc-provider \
        --region=$AWS_REGION \
        --cluster $CLUSTER_NAME \
        --approve
    ```

6. Create an IAM role, attach the AWSAppMeshFullAccess and AWSCloudMapFullAccess AWS managed policies to it, and bind it to the `appmesh-controller` Kubernetes service account. The role enables the controller to add, remove, and change App Mesh resources.

    > **Note**
    > The command creates an AWS IAM role with an auto-generated name. You are not able to specify the IAM role name that is created.

    ```
    eksctl create iamserviceaccount \
        --cluster $CLUSTER_NAME \
        --namespace appmesh-system \
        --name appmesh-controller \
        --attach-policy-arn  arn:aws:iam::aws:policy/
    AWSCloudMapFullAccess,arn:aws:iam::aws:policy/AWSAppMeshFullAccess \
        --override-existing-serviceaccounts \
        --approve
    ```

    If you prefer to create the service account using the AWS Management Console or AWS CLI, see Creating an IAM Role and Policy for your Service Account. If you use the AWS Management Console or AWS CLI to create the account, you also need to map the role to a Kubernetes service account. For more information, see Specifying an IAM Role for your Service Account.

7. Deploy the App Mesh controller. For a list of all configuration options, see Configuration on GitHub.

```
helm upgrade -i appmesh-controller eks/appmesh-controller \
    --namespace appmesh-system \
    --set region=$AWS_REGION \
    --set serviceAccount.create=false \
    --set serviceAccount.name=appmesh-controller
```

8. Confirm that the controller version is `v0.3.0` or later. You can view the change log on GitHub.

```
kubectl get deployment -n appmesh-system appmesh-controller -o json  | jq -r
 ".spec.template.spec.containers[].image" | cut -f2 -d ':'
```

9. Install the App Mesh sidecar injector and create a mesh with the name *my-mesh*. For a list of all configuration options, see Configuration on GitHub.

```
helm upgrade -i appmesh-inject eks/appmesh-inject \
    --namespace appmesh-system \
    --set mesh.name=my-mesh \
    --set mesh.create=true
```

> **Note**
> If you view the log for the running container, you may see a line that includes the following text, which can be safely ignored.
>
> ```
> Neither --kubeconfig nor --master was specified. Using the inClusterConfig. This
>  might not work.
> ```

Step 2: Deploy App Mesh resources

When you deploy an application in Kubernetes, you also create the Kubernetes custom resources so that the controller can create the corresponding App Mesh resources.

To deploy App Mesh resources

1. Create a Kubernetes namespace to deploy App Mesh resources to.

```
kubectl create ns my-app-1
```

2. Create an App Mesh virtual node. A virtual node acts as a logical pointer to a Kubernetes deployment.

   a. Create a file named *virtual-node.yaml* with the following contents. The file will be used to create an App Mesh virtual node named *my-service-a* in the *my-app-1* namespace. The virtual node represents a Kubernetes service that is created in a later step. The virtual node will communicate to a backend virtual service named *my-service-b.my-app-1.svc.cluster.local*, but it is not created in this tutorial.

```
apiVersion: appmesh.k8s.aws/v1beta1
kind: VirtualNode
metadata:
  name: my-service-a
  namespace: my-app-1
spec:
  meshName: my-mesh
  listeners:
    - portMapping:
        port: 80
```

```
        protocol: http
  serviceDiscovery:
    dns:
      hostName: my-service-a.my-app-1.svc.cluster.local
  backends:
    - virtualService:
        virtualServiceName: my-service-b.my-app-1.svc.cluster.local
```

Virtual nodes have capabilities, such as end-to-end encryption and health checks, that aren't covered in this tutorial. For more information, see Virtual nodes. To see all available settings for a virtual node that you can set in the preceding spec, run the following command.

```
aws appmesh create-virtual-node --generate-cli-skeleton yaml-input
```

b. Deploy the virtual node.

```
kubectl apply -f virtual-node.yaml
```

c. View all of the resources in the my-app-1 namespace.

```
kubectl -n my-app-1 get all
```

Output

```
NAME                          AGE
mesh.appmesh.k8s.aws/my-mesh    17m

NAME                                            AGE
virtualnode.appmesh.k8s.aws/my-service-a     16m
```

This output shows the mesh and the virtual node that exist in the my-app-1 namespace.

d. Confirm that the virtual node was created in App Mesh.

Note
Even though the name of the virtual node created in Kubernetes is my-service-a, the name of the virtual node created in App Mesh is my-service-a-my-app-1. The controller appends the Kubernetes namespace name to the App Mesh virtual node name when it creates the App Mesh resource. The namespace name is added because in Kubernetes you can create virtual nodes with the same name in different namespaces, but in App Mesh a virtual node name must be unique within a mesh.

```
aws appmesh describe-virtual-node --mesh-name my-mesh --virtual-node-name my-service-a-my-app-1
```

Output

```
{
    "virtualNode": {
        "meshName": "my-mesh",
        "metadata": {
            "arn": "arn:aws:appmesh:us-west-2:111122223333:mesh/my-mesh/
virtualNode/my-service-a-my-app-1",
            "createdAt": "2020-03-20T08:18:19.510000-05:00",
            "lastUpdatedAt": "2020-03-20T08:18:19.510000-05:00",
            "meshOwner": "111122223333",
            "resourceOwner": "111122223333",
            "uid": "111a11b1-c11d-1e1f-gh1i-j11k1l111m711",
            "version": 1
```

```
            },
            "spec": {
                "backends": [
                    {
                        "virtualService": {
                            "virtualServiceName": "my-service-b.my-
app-1.svc.cluster.local"
                        }
                    }
                ],
                "listeners": [
                    {
                        "portMapping": {
                            "port": 80,
                            "protocol": "http"
                        }
                    }
                ],
                "serviceDiscovery": {
                    "dns": {
                        "hostname": "my-service-a.my-app-1.svc.cluster.local"
                    }
                }
            },
            "status": {
                "status": "ACTIVE"
            },
            "virtualNodeName": "my-service-a-my-app-1"
    }
}
```

Note the value of `arn` in the preceding output. You will use it in a later step.

3. Create an App Mesh virtual service, virtual router, and route. A virtual service is an abstraction of a real service that is provided by a virtual node directly or indirectly by means of a virtual router. Dependent services call your virtual service by its name. The requests are routed to the virtual node or virtual router that is specified as the provider for the virtual service.

   a. Create a file named *virtual-service.yaml* with the following contents. The file will be used to create a virtual service that uses a virtual router provider to route traffic to the virtual node named *my-service-a* that was created in the previous step. The value for `name` is the fully qualified domain name (FQDN) of the actual Kubernetes service that this virtual service abstracts. The service is created in . The controller will create the App Mesh virtual service, virtual router, and route resources. You can specify many more capabilities for your routes and use protocols other than `http`. For more information, see the App Mesh Virtual services, Virtual router, and Route documentation.

```
apiVersion: appmesh.k8s.aws/v1beta1
kind: VirtualService
metadata:
  name: my-service-a.my-app-1.svc.cluster.local
  namespace: my-app-1
spec:
  meshName: my-mesh
  virtualRouter:
    name: my-service-a-virtual-router
    listeners:
      - portMapping:
          port: 80
          protocol: http
  routes:
    - name: my-service-a-route
```

```
      http:
        match:
          prefix: /
        action:
          weightedTargets:
            - virtualNodeName: my-service-a
              weight: 1
```

To see all available settings for a virtual service, virtual router, and route that you can set in the preceding spec, run any of the following commands.

```
aws appmesh create-virtual-service --generate-cli-skeleton yaml-input
aws appmesh create-virtual-router --generate-cli-skeleton yaml-input
aws appmesh create-route --generate-cli-skeleton yaml-input
```

b. Create the virtual service.

```
kubectl apply -f virtual-service.yaml
```

c. View the virtual service resource.

```
kubectl describe virtualservice my-service-a.my-app-1.svc.cluster.local -n my-app-1
```

Abbreviated output

```
Name:          my-service-a.my-app-1.svc.cluster.local
Namespace:     my-app-1
Labels:        <none>
Annotations:   kubectl.kubernetes.io/last-applied-configuration:
                 {"apiVersion":"appmesh.k8s.aws/
v1beta1","kind":"VirtualService","metadata":{"annotations":{},"name":"my-service-
a.my-app-1.svc.cluster.loc...
API Version:   appmesh.k8s.aws/v1beta1
Kind:          VirtualService
...
Spec:
  Mesh Name:  my-mesh
  Routes:
    Http:
      Action:
        Weighted Targets:
          Virtual Node Name:  my-service-a
          Weight:             1
      Match:
        Prefix:  /
    Name:        my-service-a-route
  Virtual Router:
    Listeners:
      Port Mapping:
        Port:      80
        Protocol:  http
    Name:          my-service-a-virtual-router
Status:
  Conditions:
    Last Transition Time:  2020-03-20T13:24:37Z
    Status:                True
    Type:                  VirtualRouterActive
    Last Transition Time:  2020-03-20T13:24:37Z
    Status:                True
    Type:                  RoutesActive
    Last Transition Time:  2020-03-20T13:24:38Z
    Status:                True
```

```
    Type:               VirtualServiceActive
Events:                 <none>
```

d. Confirm that the virtual service was created in your mesh. The Kubernetes controller did not append the Kubernetes namespace name to the App Mesh virtual service name when it created the virtual service in App Mesh because the virtual service's name is a unique FQDN.

```
aws appmesh describe-virtual-service --virtual-service-name my-service-a.my-
app-1.svc.cluster.local --mesh-name=my-mesh
```

Output

```
{
    "virtualService": {
        "meshName": "my-mesh",
        "metadata": {
            "arn": "arn:aws:appmesh:us-west-2:111122223333:mesh/my-mesh/
virtualService/my-service-a.my-app-1.svc.cluster.local",
            "createdAt": "2020-03-20T08:24:37.434000-05:00",
            "lastUpdatedAt": "2020-03-20T08:24:37.434000-05:00",
            "meshOwner": "111122223333",
            "resourceOwner": "111122223333",
            "uid": "111a11b1-c11d-1e1f-gh1i-j11k1l111m711",
            "version": 1
        },
        "spec": {
            "provider": {
                "virtualRouter": {
                    "virtualRouterName": "my-service-a-virtual-router-my-app-1"
                }
            }
        },
        "status": {
            "status": "ACTIVE"
        },
        "virtualServiceName": "my-service-a.my-app-1.svc.cluster.local"
    }
}
```

e. Confirm that the virtual router was created in your mesh.

> **Note**
> Though the virtual router created in Kubernetes is *my-service-a-virtual-router*, the name of the virtual router created in App Mesh is *my-service-a-virtual-router-my-app-1*.

```
aws appmesh describe-virtual-router --virtual-router-name my-service-a-virtual-
router-my-app-1 --mesh-name=my-mesh
```

Output

```
{
    "virtualRouter": {
        "meshName": "my-mesh",
        "metadata": {
            "arn": "arn:aws:appmesh:us-west-2:111122223333:mesh/my-mesh/
virtualRouter/my-service-a-virtual-router-my-app-1",
            "createdAt": "2020-03-20T08:24:37.285000-05:00",
            "lastUpdatedAt": "2020-03-20T08:24:37.285000-05:00",
            "meshOwner": "111122223333",
            "resourceOwner": "111122223333",
```

```
                    "uid": "111a11b1-c11d-1e1f-gh1i-j11k1l111m711",
                    "version": 1
                },
                "spec": {
                    "listeners": [
                        {
                            "portMapping": {
                                "port": 80,
                                "protocol": "http"
                            }
                        }
                    ]
                },
                "status": {
                    "status": "ACTIVE"
                },
                "virtualRouterName": "my-service-a-virtual-router-my-app-1"
        }
}
```

f.   Confirm that the route was created in your mesh. The Kubernetes controller did not append the Kubernetes namespace name to the App Mesh route name when it created the route in App Mesh because route names are unique to a virtual router.

```
aws appmesh describe-route --route-name my-service-a-route --virtual-router-
name my-service-a-virtual-router-my-app-1 --mesh-name my-mesh
```

Output

```
{
    "route": {
        "meshName": "my-mesh",
        "metadata": {
            "arn": "arn:aws:appmesh:us-west-2:111122223333:mesh/my-mesh/
virtualRouter/my-service-a-virtual-router-my-app-1/route/my-service-a-route",
            "createdAt": "2020-03-20T08:24:37.331000-05:00",
            "lastUpdatedAt": "2020-03-20T08:24:37.331000-05:00",
            "meshOwner": "111122223333",
            "resourceOwner": "111122223333",
            "uid": "111a11b1-c11d-1e1f-gh1i-j11k1l111m711",
            "version": 1
        },
        "routeName": "my-service-a-route",
        "spec": {
            "httpRoute": {
                "action": {
                    "weightedTargets": [
                        {
                            "virtualNode": "my-service-a-my-app-1",
                            "weight": 1
                        }
                    ]
                },
                "match": {
                    "prefix": "/"
                }
            }
        },
        "status": {
            "status": "ACTIVE"
        },
        "virtualRouterName": "my-service-a-virtual-router-my-app-1"
    }
```

```
}
```

Step 3: Create or update services

Any pods that you want to use with App Mesh must have the App Mesh sidecar containers added to them. The injector automatically adds the sidecar containers to any pod deployed into a namespace that you specify.

To create or update services

1. To enable sidecar injection for the namespace, label the namespace.

   ```
   kubectl label namespace my-app-1 appmesh.k8s.aws/sidecarInjectorWebhook=enabled
   ```

2. Enable proxy authorization. We recommend that you enable each Kubernetes deployment to stream the configuration for its own App Mesh virtual node.

   a. Create a file named *proxy-auth.json* with the following contents.

   ```
   {
       "Version": "2012-10-17",
       "Statement": [
           {
               "Effect": "Allow",
               "Action": "appmesh:StreamAggregatedResources",
               "Resource": [
                   "arn:aws:appmesh:region-code:111122223333:mesh/my-mesh/
   virtualNode/my-service-a-my-app-1"
               ]
           }
       ]
   }
   ```

   b. Create the policy.

   ```
   aws iam create-policy --policy-name my-policy --policy-document file://proxy-
   auth.json
   ```

   Note the ARN of the policy in the output returned. You'll use it in the next step.

   c. Create an IAM role, attach the policy you created in the previous step to it, create a Kubernetes service account and bind the policy to the Kubernetes service account. The role enables the controller to add, remove, and change App Mesh resources.

   ```
   eksctl create iamserviceaccount \
       --cluster $CLUSTER_NAME \
       --namespace my-app-1 \
       --name my-service-a \
       --attach-policy-arn  arn:aws:iam::111122223333:policy/my-policy \
       --override-existing-serviceaccounts \
       --approve
   ```

   If you prefer to create the service account using the AWS Management Console or AWS CLI, see Creating an IAM Role and Policy for your Service Account. If you use the AWS Management Console or AWS CLI to create the account, you also need to map the role to a Kubernetes service account. For more information, see Specifying an IAM Role for your Service Account.

3. Create a Kubernetes service and deployment. If you have an existing deployment that you want to use with App Mesh, you need to update its namespace to *my-app-1* so that the sidecar containers are automatically added to the pods and the pods are redeployed.

   a. Create a file named *example-service.yaml* with the following contents.

   ```
   apiVersion: v1
   kind: Service
   metadata:
     name: my-service-a
     namespace: my-app-1
     labels:
       app: nginx
   spec:
     selector:
       app: nginx
     ports:
       - protocol: TCP
         port: 80
         targetPort: 80
   ---
   apiVersion: apps/v1
   kind: Deployment
   metadata:
     name: my-service-a
     namespace: my-app-1
     labels:
       app: nginx
   spec:
     replicas: 3
     selector:
       matchLabels:
         app: nginx
     template:
       metadata:
         labels:
           app: nginx
       spec:
         serviceAccountName: my-service-a
         containers:
         - name: nginx
           image: nginx:1.14.2
           ports:
           - containerPort: 80
   ```

   You can override the default behavior of the injector for individual pods. For example, notice in the preceding spec that the name of the deployment is *my-service-a*. By default, this name must be the same as the name of the virtual node that you created in the section called "Step 2: Deploy App Mesh resources" (p. 224). If you want the name of the virtual node to be different than the name of the deployment, then you must add an annotation to your spec for the `virtualNode` setting. To familiarize yourself with the settings that you can override, see Default behavior and how to override on GitHub.

   b. Deploy the service.

   ```
   kubectl apply -f example-service.yaml
   ```

   c. View the service and deployment.

   ```
   kubectl -n my-app-1 get pods
   ```

```
NAME                              READY   STATUS    RESTARTS   AGE
my-service-a-658c47c864-g9p9l     2/2     Running   0          26s
my-service-a-658c47c864-grn8w     2/2     Running   0          26s
my-service-a-658c47c864-lc5qk     2/2     Running   0          26s
```

d. View the details for one of the pods that was deployed.

```
kubectl -n my-app-1 describe pod my-service-a-7fd6966748-79674
```

Abbreviated output

```
Name:           my-service-a-658c47c864-g9p9l
Namespace:      my-app-1
...
Init Containers:
  proxyinit:
    Container ID:
 docker://5f45b51566681be12eb851d48a47199e5a09d8661a16296bca826c93d3703ae6
    Image:          111345817488.dkr.ecr.us-west-2.amazonaws.com/aws-appmesh-proxy-
route-manager:v2
    Image ID:       docker-pullable://111345817488.dkr.ecr.us-
west-2.amazonaws.com/aws-appmesh-proxy-route-
manager@sha256:1111111111111111111111111111111111111111111111111111111111111111
    Port:           <none>
    Host Port:      <none>
    State:          Terminated
      Reason:       Completed
      Exit Code:    0
      Started:      Fri, 20 Mar 2020 08:37:05 -0500
      Finished:     Fri, 20 Mar 2020 08:37:06 -0500
    Ready:          True
    Restart Count:  0
    Requests:
      cpu:      10m
      memory:   32Mi
    Environment:
      APPMESH_START_ENABLED:          1
      APPMESH_IGNORE_UID:             1337
      APPMESH_ENVOY_INGRESS_PORT:     15000
      APPMESH_ENVOY_EGRESS_PORT:      15001
      APPMESH_APP_PORTS:              80
      APPMESH_EGRESS_IGNORED_IP:      169.254.169.254
      APPMESH_EGRESS_IGNORED_PORTS:   22
      AWS_ROLE_ARN:                   arn:aws:iam::111122223333:role/eksctl-
appmesh-1-addon-iamserviceaccount-my-Role1-MVK5HIHHL4WT
      AWS_WEB_IDENTITY_TOKEN_FILE:    /var/run/secrets/eks.amazonaws.com/
serviceaccount/token
    Mounts:
      /var/run/secrets/eks.amazonaws.com/serviceaccount from aws-iam-token (ro)
      /var/run/secrets/kubernetes.io/serviceaccount from my-service-a-token-s29ls
 (ro)
Containers:
  nginx:
    Container ID:
 docker://022c01578fa71129213a0de9df0748ff410f468739edc872bb1888843020bd88
    Image:          nginx:1.14.2
    Image ID:       docker-pullable://
nginx@sha256:1111111111111111111111111111111111111111111111111111111111111111
    Port:           80/TCP
    Host Port:      0/TCP
```

```
      State:          Running
        Started:      Fri, 20 Mar 2020 08:37:11 -0500
      Ready:          True
      Restart Count:  0
      Environment:
        AWS_ROLE_ARN:                  arn:aws:iam::111122223333:role/eksctl-
appmesh-1-addon-iamserviceaccount-my-Role1-MVK5HIHHL4WT
        AWS_WEB_IDENTITY_TOKEN_FILE:  /var/run/secrets/eks.amazonaws.com/
serviceaccount/token
      Mounts:
        /var/run/secrets/eks.amazonaws.com/serviceaccount from aws-iam-token (ro)
        /var/run/secrets/kubernetes.io/serviceaccount from my-service-a-token-s29ls
 (ro)
  envoy:
    Container ID:
 docker://742934a0722734762890ae925a4c143b739ba114ab33acf14919c4069773f492
      Image:          840364872350.dkr.ecr.us-west-2.amazonaws.com/aws-appmesh-
envoy:v1.12.2.1-prod
      Image ID:       docker-pullable://840364872350.dkr.ecr.us-west-2.amazonaws.com/
aws-appmesh-
envoy@sha256:111111111111111111111111111111111111111111111111111111111111111111
      Port:           9901/TCP
      Host Port:      0/TCP
      State:          Running
        Started:      Fri, 20 Mar 2020 08:37:18 -0500
      Ready:          True
      Restart Count:  0
      Requests:
        cpu:      10m
        memory:   32Mi
      Environment:
        APPMESH_VIRTUAL_NODE_NAME:    mesh/my-mesh/virtualNode/my-service-a-my-app-1
        APPMESH_PREVIEW:              0
        ENVOY_LOG_LEVEL:              info
        AWS_REGION:                   us-west-2
        AWS_ROLE_ARN:                  arn:aws:iam::111122223333:role/eksctl-
appmesh-1-addon-iamserviceaccount-my-Role1-MVK5HIHHL4WT
        AWS_WEB_IDENTITY_TOKEN_FILE:  /var/run/secrets/eks.amazonaws.com/
serviceaccount/token
      Mounts:
        /var/run/secrets/eks.amazonaws.com/serviceaccount from aws-iam-token (ro)
        /var/run/secrets/kubernetes.io/serviceaccount from my-service-a-token-s29ls
 (ro)
...
```

In the preceding output, you can see that the `proxyinit` and `envoy` containers were added to the pod.

Step 4: Clean up

Remove all of the example resources created in this tutorial. The controller also removes the resources that were created in App Mesh.

```
kubectl delete namespace my-app-1
```

(Optional) You can remove the Kubernetes integration components.

```
helm delete appmesh-controller -n appmesh-system
helm delete appmesh-inject -n appmesh-system
```

Deep Learning Containers

AWS Deep Learning Containers are a set of Docker images for training and serving models in TensorFlow on Amazon EKS and Amazon Elastic Container Service (Amazon ECS). Deep Learning Containers provide optimized environments with TensorFlow, Nvidia CUDA (for GPU instances), and Intel MKL (for CPU instances) libraries and are available in Amazon ECR.

To get started using AWS Deep Learning Containers on Amazon EKS, see AWS Deep Learning Containers on Amazon EKS in the *AWS Deep Learning AMI Developer Guide*.

Security in Amazon EKS

Cloud security at AWS is the highest priority. As an AWS customer, you benefit from a data center and network architecture that is built to meet the requirements of the most security-sensitive organizations.

Security is a shared responsibility between AWS and you. The shared responsibility model describes this as security *of* the cloud and security *in* the cloud:

- **Security of the cloud** – AWS is responsible for protecting the infrastructure that runs AWS services in the AWS Cloud. For Amazon EKS, AWS is responsible for the Kubernetes control plane, which includes the control plane nodes and `etcd` database. Third-party auditors regularly test and verify the effectiveness of our security as part of the AWS compliance programs. To learn about the compliance programs that apply to Amazon EKS, see AWS Services in Scope by Compliance Program.
- **Security in the cloud** – Your responsibility includes the following areas.
  - The security configuration of the data plane, including the configuration of the security groups that allow traffic to pass from the Amazon EKS control plane into the customer VPC
  - The configuration of the worker nodes and the containers themselves
  - The worker node guest operating system (including updates and security patches)
  - Other associated application software:
    - Setting up and managing network controls, such as firewall rules
    - Managing platform-level identity and access management, either with or in addition to IAM
  - The sensitivity of your data, your company's requirements, and applicable laws and regulations

This documentation helps you understand how to apply the shared responsibility model when using Amazon EKS. The following topics show you how to configure Amazon EKS to meet your security and compliance objectives. You also learn how to use other AWS services that help you to monitor and secure your Amazon EKS resources.

Topics

Identity and access management for Amazon EKS

AWS Identity and Access Management (IAM) is an AWS service that helps an administrator securely control access to AWS resources. IAM administrators control who can be *authenticated* (signed in) and *authorized* (have permissions) to use Amazon EKS resources. IAM is an AWS service that you can use with no additional charge.

Audience

How you use AWS Identity and Access Management (IAM) differs, depending on the work you do in Amazon EKS.

Service user – If you use the Amazon EKS service to do your job, then your administrator provides you with the credentials and permissions that you need. As you use more Amazon EKS features to do your work, you might need additional permissions. Understanding how access is managed can help you request the right permissions from your administrator. If you cannot access a feature in Amazon EKS, see Troubleshooting Amazon EKS identity and access (p. 266).

Service administrator – If you're in charge of Amazon EKS resources at your company, you probably have full access to Amazon EKS. It's your job to determine which Amazon EKS features and resources your employees should access. You must then submit requests to your IAM administrator to change the permissions of your service users. Review the information on this page to understand the basic concepts of IAM. To learn more about how your company can use IAM with Amazon EKS, see How Amazon EKS works with IAM (p. 239).

IAM administrator – If you're an IAM administrator, you might want to learn details about how you can write policies to manage access to Amazon EKS. To view example Amazon EKS identity-based policies that you can use in IAM, see Amazon EKS identity-based policy examples (p. 242).

Authenticating with identities

Authentication is how you sign in to AWS using your identity credentials. For more information about signing in using the AWS Management Console, see The IAM Console and Sign-in Page in the *IAM User Guide*.

You must be *authenticated* (signed in to AWS) as the AWS account root user, an IAM user, or by assuming an IAM role. You can also use your company's single sign-on authentication, or even sign in using Google or Facebook. In these cases, your administrator previously set up identity federation using IAM roles. When you access AWS using credentials from another company, you are assuming a role indirectly.

To sign in directly to the AWS Management Console, use your password with your root user email or your IAM user name. You can access AWS programmatically using your root user or IAM user access keys. AWS provides SDK and command line tools to cryptographically sign your request using your credentials. If you don't use AWS tools, you must sign the request yourself. Do this using *Signature Version 4*, a protocol for authenticating inbound API requests. For more information about authenticating requests, see Signature Version 4 Signing Process in the *AWS General Reference*.

Regardless of the authentication method that you use, you might also be required to provide additional security information. For example, AWS recommends that you use multi-factor authentication (MFA) to increase the security of your account. To learn more, see Using Multi-Factor Authentication (MFA) in AWS in the *IAM User Guide*.

AWS Account Root User

When you first create an AWS account, you begin with a single sign-in identity that has complete access to all AWS services and resources in the account. This identity is called the AWS account *root user* and is accessed by signing in with the email address and password that you used to create the account. We strongly recommend that you do not use the root user for your everyday tasks, even the administrative ones. Instead, adhere to the best practice of using the root user only to create your first IAM user. Then securely lock away the root user credentials and use them to perform only a few account and service management tasks.

IAM users and groups

An *IAM user* is an identity within your AWS account that has specific permissions for a single person or application. An IAM user can have long-term credentials such as a user name and password or a set of access keys. To learn how to generate access keys, see Managing Access Keys for IAM Users in the *IAM User Guide*. When you generate access keys for an IAM user, make sure you view and securely save the key

pair. You cannot recover the secret access key in the future. Instead, you must generate a new access key pair.

An *IAM group* is an identity that specifies a collection of IAM users. You can't sign in as a group. You can use groups to specify permissions for multiple users at a time. Groups make permissions easier to manage for large sets of users. For example, you could have a group named *IAMAdmins* and give that group permissions to administer IAM resources.

Users are different from roles. A user is uniquely associated with one person or application, but a role is intended to be assumable by anyone who needs it. Users have permanent long-term credentials, but roles provide temporary credentials. To learn more, see When to Create an IAM User (Instead of a Role) in the *IAM User Guide*.

IAM roles

An *IAM role* is an identity within your AWS account that has specific permissions. It is similar to an IAM user, but is not associated with a specific person. You can temporarily assume an IAM role in the AWS Management Console by switching roles. You can assume a role by calling an AWS CLI or AWS API operation or by using a custom URL. For more information about methods for using roles, see Using IAM Roles in the *IAM User Guide*.

IAM roles with temporary credentials are useful in the following situations:

- **Temporary IAM user permissions** – An IAM user can assume an IAM role to temporarily take on different permissions for a specific task.

- **Federated user access** – Instead of creating an IAM user, you can use existing identities from AWS Directory Service, your enterprise user directory, or a web identity provider. These are known as *federated users*. AWS assigns a role to a federated user when access is requested through an identity provider. For more information about federated users, see Federated Users and Roles in the *IAM User Guide*.

- **Cross-account access** – You can use an IAM role to allow someone (a trusted principal) in a different account to access resources in your account. Roles are the primary way to grant cross-account access. However, with some AWS services, you can attach a policy directly to a resource (instead of using a role as a proxy). To learn the difference between roles and resource-based policies for cross-account access, see How IAM Roles Differ from Resource-based Policies in the *IAM User Guide*.

- **AWS service access** – A service role is an IAM role that a service assumes to perform actions in your account on your behalf. When you set up some AWS service environments, you must define a role for the service to assume. This service role must include all the permissions that are required for the service to access the AWS resources that it needs. Service roles vary from service to service, but many allow you to choose your permissions as long as you meet the documented requirements for that service. Service roles provide access only within your account and cannot be used to grant access to services in other accounts. You can create, modify, and delete a service role from within IAM. For example, you can create a role that allows Amazon Redshift to access an Amazon S3 bucket on your behalf and then load data from that bucket into an Amazon Redshift cluster. For more information, see Creating a Role to Delegate Permissions to an AWS Service in the *IAM User Guide*.

- **Applications running on Amazon EC2** – You can use an IAM role to manage temporary credentials for applications that are running on an EC2 instance and making AWS CLI or AWS API requests. This is preferable to storing access keys within the EC2 instance. To assign an AWS role to an EC2 instance and make it available to all of its applications, you create an instance profile that is attached to the instance. An instance profile contains the role and enables programs that are running on the EC2 instance to get temporary credentials. For more information, see Using an IAM Role to Grant Permissions to Applications Running on Amazon EC2 Instances in the *IAM User Guide*.

To learn whether to use IAM roles, see When to Create an IAM Role (Instead of a User) in the *IAM User Guide*.

Managing access using policies

You control access in AWS by creating policies and attaching them to IAM identities or AWS resources. A policy is an object in AWS that, when associated with an identity or resource, defines their permissions. AWS evaluates these policies when an entity (root user, IAM user, or IAM role) makes a request. Permissions in the policies determine whether the request is allowed or denied. Most policies are stored in AWS as JSON documents. For more information about the structure and contents of JSON policy documents, see Overview of JSON Policies in the *IAM User Guide*.

An IAM administrator can use policies to specify who has access to AWS resources, and what actions they can perform on those resources. Every IAM entity (user or role) starts with no permissions. In other words, by default, users can do nothing, not even change their own password. To give a user permission to do something, an administrator must attach a permissions policy to a user. Or the administrator can add the user to a group that has the intended permissions. When an administrator gives permissions to a group, all users in that group are granted those permissions.

IAM policies define permissions for an action regardless of the method that you use to perform the operation. For example, suppose that you have a policy that allows the `iam:GetRole` action. A user with that policy can get role information from the AWS Management Console, the AWS CLI, or the AWS API.

Identity-based policies

Identity-based policies are JSON permissions policy documents that you can attach to an identity, such as an IAM user, role, or group. These policies control what actions that identity can perform, on which resources, and under what conditions. To learn how to create an identity-based policy, see Creating IAM Policies in the *IAM User Guide*.

Identity-based policies can be further categorized as *inline policies* or *managed policies*. Inline policies are embedded directly into a single user, group, or role. Managed policies are standalone policies that you can attach to multiple users, groups, and roles in your AWS account. Managed policies include AWS managed policies and customer managed policies. To learn how to choose between a managed policy or an inline policy, see Choosing Between Managed Policies and Inline Policies in the *IAM User Guide*.

Resource-based policies

Resource-based policies are JSON policy documents that you attach to a resource such as an Amazon S3 bucket. Service administrators can use these policies to define what actions a specified principal (account member, user, or role) can perform on that resource and under what conditions. Resource-based policies are inline policies. There are no managed resource-based policies.

Access control lists (ACLs)

Access control lists (ACLs) are a type of policy that controls which principals (account members, users, or roles) have permissions to access a resource. ACLs are similar to resource-based policies, although they do not use the JSON policy document format. Amazon S3, AWS WAF, and Amazon VPC are examples of services that support ACLs. To learn more about ACLs, see Access Control List (ACL) Overview in the *Amazon Simple Storage Service Developer Guide*.

Other policy types

AWS supports additional, less-common policy types. These policy types can set the maximum permissions granted to you by the more common policy types.

- **Permissions boundaries** – A permissions boundary is an advanced feature in which you set the maximum permissions that an identity-based policy can grant to an IAM entity (IAM user or role). You can set a permissions boundary for an entity. The resulting permissions are the intersection of

entity's identity-based policies and its permissions boundaries. Resource-based policies that specify the user or role in the `Principal` field are not limited by the permissions boundary. An explicit deny in any of these policies overrides the allow. For more information about permissions boundaries, see Permissions Boundaries for IAM Entities in the *IAM User Guide*.

- **Service control policies (SCPs)** – SCPs are JSON policies that specify the maximum permissions for an organization or organizational unit (OU) in AWS Organizations. AWS Organizations is a service for grouping and centrally managing multiple AWS accounts that your business owns. If you enable all features in an organization, then you can apply service control policies (SCPs) to any or all of your accounts. The SCP limits permissions for entities in member accounts, including each AWS account root user. For more information about Organizations and SCPs, see How SCPs Work in the *AWS Organizations User Guide*.

- **Session policies** – Session policies are advanced policies that you pass as a parameter when you programmatically create a temporary session for a role or federated user. The resulting session's permissions are the intersection of the user or role's identity-based policies and the session policies. Permissions can also come from a resource-based policy. An explicit deny in any of these policies overrides the allow. For more information, see Session Policies in the *IAM User Guide*.

Multiple policy types

When multiple types of policies apply to a request, the resulting permissions are more complicated to understand. To learn how AWS determines whether to allow a request when multiple policy types are involved, see Policy Evaluation Logic in the *IAM User Guide*.

How Amazon EKS works with IAM

Before you use IAM to manage access to Amazon EKS, you should understand what IAM features are available to use with Amazon EKS. To get a high-level view of how Amazon EKS and other AWS services work with IAM, see AWS services that work with IAM in the *IAM User Guide*.

Topics

Amazon EKS identity-based policies

With IAM identity-based policies, you can specify allowed or denied actions and resources as well as the conditions under which actions are allowed or denied. Amazon EKS supports specific actions, resources, and condition keys. To learn about all of the elements that you use in a JSON policy, see IAM JSON policy elements reference in the *IAM User Guide*.

Actions

The `Action` element of an IAM identity-based policy describes the specific action or actions that will be allowed or denied by the policy. Policy actions usually have the same name as the associated AWS API operation. The action is used in a policy to grant permissions to perform the associated operation.

Policy actions in Amazon EKS use the following prefix before the action: `eks:`. For example, to grant someone permission to get descriptive information about an Amazon EKS cluster, you include the `DescribeCluster` action in their policy. Policy statements must include either an `Action` or `NotAction` element.

To specify multiple actions in a single statement, separate them with commas as follows:

```
"Action": ["eks:action1", "eks:action2"]
```

You can specify multiple actions using wildcards (*). For example, to specify all actions that begin with the word `Describe`, include the following action:

```
"Action": "eks:Describe*"
```

To see a list of Amazon EKS actions, see Actions Defined by Amazon Elastic Kubernetes Service in the *IAM User Guide*.

Resources

The `Resource` element specifies the object or objects to which the action applies. Statements must include either a `Resource` or a `NotResource` element. You specify a resource using an ARN or using the wildcard (*) to indicate that the statement applies to all resources.

The Amazon EKS cluster resource has the following ARN:

```
arn:${Partition}:eks:${Region}:${Account}:cluster/${ClusterName}
```

For more information about the format of ARNs, see Amazon resource names (ARNs) and AWS service namespaces.

For example, to specify the `dev` cluster in your statement, use the following ARN:

```
"Resource": "arn:aws:eks:region-code:123456789012:cluster/dev"
```

To specify all clusters that belong to a specific account and Region, use the wildcard (*):

```
"Resource": "arn:aws:eks:region-code:123456789012:cluster/*"
```

Some Amazon EKS actions, such as those for creating resources, cannot be performed on a specific resource. In those cases, you must use the wildcard (*).

```
"Resource": "*"
```

To see a list of Amazon EKS resource types and their ARNs, see Resources Defined by Amazon Elastic Kubernetes Service in the *IAM User Guide*. To learn with which actions you can specify the ARN of each resource, see Actions Defined by Amazon Elastic Kubernetes Service.

Condition keys

Amazon EKS does not provide any service-specific condition keys, but it does support using some global condition keys. To see all AWS global condition keys, see AWS global condition context keys in the *IAM User Guide*.

Examples

To view examples of Amazon EKS identity-based policies, see Amazon EKS identity-based policy examples (p. 242).

When you create an Amazon EKS cluster, the IAM entity user or role, such as a federated user that creates the cluster, is automatically granted `system:masters` permissions in the cluster's RBAC

configuration. To grant additional AWS users or roles the ability to interact with your cluster, you must edit the `aws-auth` ConfigMap within Kubernetes.

For additional information about working with the ConfigMap, see Managing users or IAM roles for your cluster (p. 193).

Amazon EKS resource-based policies

Amazon EKS does not support resource-based policies.

Authorization based on Amazon EKS tags

You can attach tags to Amazon EKS resources or pass tags in a request to Amazon EKS. To control access based on tags, you provide tag information in the condition element of a policy using the `eks:ResourceTag/`*`key-name`*, `aws:RequestTag/`*`key-name`*, or `aws:TagKeys` condition keys. For more information about tagging Amazon EKS resources, see Tagging your Amazon EKS resources (p. 274).

Amazon EKS IAM roles

An IAM role is an entity within your AWS account that has specific permissions.

Using temporary credentials with Amazon EKS

You can use temporary credentials to sign in with federation, assume an IAM role, or to assume a cross-account role. You obtain temporary security credentials by calling AWS STS API operations such as AssumeRole or GetFederationToken.

Amazon EKS supports using temporary credentials.

Service-linked roles

Service-linked roles allow AWS services to access resources in other services to complete an action on your behalf. Service-linked roles appear in your IAM account and are owned by the service. An IAM administrator can view but not edit the permissions for service-linked roles.

Amazon EKS supports service-linked roles. For details about creating or managing Amazon EKS service-linked roles, see Using Service-Linked Roles for Amazon EKS (p. 244).

Service roles

This feature allows a service to assume a service role on your behalf. This role allows the service to access resources in other services to complete an action on your behalf. Service roles appear in your IAM account and are owned by the account. This means that an IAM administrator can change the permissions for this role. However, doing so might break the functionality of the service.

Amazon EKS supports service roles. For more information, see the section called "Cluster IAM role" (p. 248) and the section called "Worker node IAM role" (p. 250).

Choosing an IAM role in Amazon EKS

When you create a cluster resource in Amazon EKS, you must choose a role to allow Amazon EKS to access several other AWS resources on your behalf. If you have previously created a service role, then Amazon EKS provides you with a list of roles to choose from. It's important to choose a role that has the Amazon EKS managed policies attached to it. For more information, see the section called "Check for an existing cluster role" (p. 248) and the section called "Check for an existing worker node role" (p. 250).

Amazon EKS identity-based policy examples

By default, IAM users and roles don't have permission to create or modify Amazon EKS resources. They also can't perform tasks using the AWS Management Console, AWS CLI, or AWS API. An IAM administrator must create IAM policies that grant users and roles permission to perform specific API operations on the specified resources they need. The administrator must then attach those policies to the IAM users or groups that require those permissions.

To learn how to create an IAM identity-based policy using these example JSON policy documents, see Creating policies on the JSON tab in the *IAM User Guide*.

When you create an Amazon EKS cluster, the IAM entity user or role, such as a federated user that creates the cluster, is automatically granted `system:masters` permissions in the cluster's RBAC configuration. To grant additional AWS users or roles the ability to interact with your cluster, you must edit the `aws-auth` ConfigMap within Kubernetes.

For additional information about working with the ConfigMap, see Managing users or IAM roles for your cluster (p. 193).

Topics
- Policy best practices (p. 242)
- Using the Amazon EKS console (p. 242)
- Allow users to view their own permissions (p. 243)
- Update a Kubernetes cluster (p. 244)
- List or describe all clusters (p. 244)

Policy best practices

Identity-based policies are very powerful. They determine whether someone can create, access, or delete Amazon EKS resources in your account. These actions can incur costs for your AWS account. When you create or edit identity-based policies, follow these guidelines and recommendations:

- **Get Started Using AWS Managed Policies** – To start using Amazon EKS quickly, use AWS managed policies to give your employees the permissions they need. These policies are already available in your account and are maintained and updated by AWS. For more information, see Get Started Using Permissions With AWS Managed Policies in the *IAM User Guide*.
- **Grant Least Privilege** – When you create custom policies, grant only the permissions required to perform a task. Start with a minimum set of permissions and grant additional permissions as necessary. Doing so is more secure than starting with permissions that are too lenient and then trying to tighten them later. For more information, see Grant Least Privilege in the *IAM User Guide*.
- **Enable MFA for Sensitive Operations** – For extra security, require IAM users to use multi-factor authentication (MFA) to access sensitive resources or API operations. For more information, see Using Multi-Factor Authentication (MFA) in AWS in the *IAM User Guide*.
- **Use Policy Conditions for Extra Security** – To the extent that it's practical, define the conditions under which your identity-based policies allow access to a resource. For example, you can write conditions to specify a range of allowable IP addresses that a request must come from. You can also write conditions to allow requests only within a specified date or time range, or to require the use of SSL or MFA. For more information, see IAM JSON Policy Elements: Condition in the *IAM User Guide*.

Using the Amazon EKS console

To access the Amazon EKS console, you must have a minimum set of permissions. These permissions must allow you to list and view details about the Amazon EKS resources in your AWS account. If you

create an identity-based policy that is more restrictive than the minimum required permissions, the console won't function as intended for entities (IAM users or roles) with that policy.

To ensure that those entities can still use the Amazon EKS console, create a policy with your own unique name, such as `AmazonEKSAdminPolicy`. Attach the policy to the entities. For more information, see Adding permissions to a user in the *IAM User Guide*:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "eks:*"
            ],
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": "iam:PassRole",
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "iam:PassedToService": "eks.amazonaws.com"
                }
            }
        }
    ]
}
```

You don't need to allow minimum console permissions for users that are making calls only to the AWS CLI or the AWS API. Instead, allow access to only the actions that match the API operation that you're trying to perform.

Allow users to view their own permissions

This example shows how you might create a policy that allows IAM users to view the inline and managed policies that are attached to their user identity. This policy includes permissions to complete this action on the console or programmatically using the AWS CLI or AWS API.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "ViewOwnUserInfo",
            "Effect": "Allow",
            "Action": [
                "iam:GetUserPolicy",
                "iam:ListGroupsForUser",
                "iam:ListAttachedUserPolicies",
                "iam:ListUserPolicies",
                "iam:GetUser"
            ],
            "Resource": ["arn:aws:iam::*:user/${aws:username}"]
        },
        {
            "Sid": "NavigateInConsole",
            "Effect": "Allow",
            "Action": [
                "iam:GetGroupPolicy",
                "iam:GetPolicyVersion",
                "iam:GetPolicy",
```

```
                "iam:ListAttachedGroupPolicies",
                "iam:ListGroupPolicies",
                "iam:ListPolicyVersions",
                "iam:ListPolicies",
                "iam:ListUsers"
            ],
            "Resource": "*"
        }
    ]
}
```

Update a Kubernetes cluster

This example shows how you can create a policy that allows a user to update the Kubernetes version of any *dev* cluster for an account, in any region.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "eks:UpdateClusterVersion",
            "Resource": "arn:aws:eks:*:111122223333:cluster/dev"
        }
    ]
}
```

List or describe all clusters

This example shows how you can create a policy that allows a user read-only access to list or describe all clusters. An account must be able to list and describe clusters to use the `update-kubeconfig` AWS CLI command.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "eks:DescribeCluster",
                "eks:ListClusters"
            ],
            "Resource": "*"
        }
    ]
}
```

Using Service-Linked Roles for Amazon EKS

Amazon Elastic Kubernetes Service uses AWS Identity and Access Management (IAM) service-linked roles. A service-linked role is a unique type of IAM role that is linked directly to Amazon EKS. Service-linked roles are predefined by Amazon EKS and include all the permissions that the service requires to call other AWS services on your behalf.

Topics

Using Roles for Amazon EKS

Amazon Elastic Kubernetes Service uses AWS Identity and Access Management (IAM) service-linked roles. A service-linked role is a unique type of IAM role that is linked directly to Amazon EKS. Service-linked roles are predefined by Amazon EKS and include all the permissions that the service requires to call other AWS services on your behalf.

A service-linked role makes setting up Amazon EKS easier because you don't have to manually add the necessary permissions. Amazon EKS defines the permissions of its service-linked roles, and unless defined otherwise, only Amazon EKS can assume its roles. The defined permissions include the trust policy and the permissions policy, and that permissions policy cannot be attached to any other IAM entity.

You can delete a service-linked role only after first deleting their related resources. This protects your Amazon EKS resources because you can't inadvertently remove permission to access the resources.

For information about other services that support service-linked roles, see AWS Services That Work with IAM and look for the services that have **Yes** in the **Service-Linked Role** column. Choose a **Yes** with a link to view the service-linked role documentation for that service.

Service-Linked Role Permissions for Amazon EKS

Amazon EKS uses the service-linked role named **AWSServiceRoleForAmazonEKS** – These permissions are required for Amazon EKS to manage clusters in your account. These policies are related to management of the following resources: network interfaces, security groups, logs, and VPCs.

> **Note**
> The **AWSServiceRoleForAmazonEKS** service-linked role is distinct from the role required for cluster creation. For more information, see the section called "Cluster IAM role" (p. 248).

The AWSServiceRoleForAmazonEKS service-linked role trusts the following services to assume the role:

- `eks.amazonaws.com`

The role permissions policy allows Amazon EKS to complete the following actions on the specified resources:

- AWSServiceRoleForAmazonEKS

You must configure permissions to allow an IAM entity (such as a user, group, or role) to create, edit, or delete a service-linked role. For more information, see Service-Linked Role Permissions in the *IAM User Guide*.

Creating a Service-Linked Role for Amazon EKS

You don't need to manually create a service-linked role. When you create a cluster in the AWS Management Console, the AWS CLI, or the AWS API, Amazon EKS creates the service-linked role for you.

If you delete this service-linked role, and then need to create it again, you can use the same process to recreate the role in your account. When you create a cluster, Amazon EKS creates the service-linked role for you again.

Editing a Service-Linked Role for Amazon EKS

Amazon EKS does not allow you to edit the AWSServiceRoleForAmazonEKS service-linked role. After you create a service-linked role, you cannot change the name of the role because various entities might

reference the role. However, you can edit the description of the role using IAM. For more information, see Editing a Service-Linked Role in the *IAM User Guide*.

Deleting a Service-Linked Role for Amazon EKS

If you no longer need to use a feature or service that requires a service-linked role, we recommend that you delete that role. That way you don't have an unused entity that is not actively monitored or maintained. However, you must clean up your service-linked role before you can manually delete it.

Cleaning Up a Service-Linked Role

Before you can use IAM to delete a service-linked role, you must first delete any resources used by the role.

> **Note**
> If the Amazon EKS service is using the role when you try to delete the resources, then the deletion might fail. If that happens, wait for a few minutes and try the operation again.

To delete Amazon EKS resources used by the AWSServiceRoleForAmazonEKS

1. Open the Amazon EKS console at https://console.aws.amazon.com/eks/home#/clusters.
2. Choose a cluster.
3. On the cluster page, if there are any managed node groups in the **Node Groups** section, select each one individually, and then choose **Delete**.
4. Type the name of the node group in the deletion confirmation window, and then choose **Confirm** to delete.
5. Repeat this procedure for any other node groups in the cluster. Wait for all of the delete operations to finish.
6. On the cluster page choose **Delete**.
7. Repeat this procedure for any other clusters in your account.

Manually Delete the Service-Linked Role

Use the IAM console, the AWS CLI, or the AWS API to delete the AWSServiceRoleForAmazonEKS service-linked role. For more information, see Deleting a Service-Linked Role in the *IAM User Guide*.

Supported Regions for Amazon EKS Service-Linked Roles

Amazon EKS supports using service-linked roles in all of the regions where the service is available. For more information, see Amazon EKS Service Endpoints and Quotas.

Using Roles for Amazon EKS Nodegroups

Amazon EKS uses AWS Identity and Access Management (IAM) service-linked roles. A service-linked role is a unique type of IAM role that is linked directly to Amazon EKS. Service-linked roles are predefined by Amazon EKS and include all the permissions that the service requires to call other AWS services on your behalf.

A service-linked role makes setting up Amazon EKS easier because you don't have to manually add the necessary permissions. Amazon EKS defines the permissions of its service-linked roles, and unless defined otherwise, only Amazon EKS can assume its roles. The defined permissions include the trust policy and the permissions policy, and that permissions policy cannot be attached to any other IAM entity.

You can delete a service-linked role only after first deleting their related resources. This protects your Amazon EKS resources because you can't inadvertently remove permission to access the resources.

For information about other services that support service-linked roles, see AWS Services That Work with IAM and look for the services that have **Yes** in the **Service-Linked Role** column. Choose a **Yes** with a link to view the service-linked role documentation for that service.

Service-Linked Role Permissions for Amazon EKS

Amazon EKS uses the service-linked role named **AWSServiceRoleForAmazonEKSNodegroup** – These permissions are required for managing nodegroups in your account. These policies are related to management of the following resources: Auto Scaling groups, security groups, launch templates and IAM instance profiles..

The AWSServiceRoleForAmazonEKSNodegroup service-linked role trusts the following services to assume the role:

- `eks-nodegroup.amazonaws.com`

The role permissions policy allows Amazon EKS to complete the following actions on the specified resources:

- AWSServiceRoleForAmazonEKS

You must configure permissions to allow an IAM entity (such as a user, group, or role) to create, edit, or delete a service-linked role. For more information, see Service-Linked Role Permissions in the *IAM User Guide*.

Creating a Service-Linked Role for Amazon EKS

You don't need to manually create a service-linked role. When you CreateNodegroup in the AWS Management Console, the AWS CLI, or the AWS API, Amazon EKS creates the service-linked role for you.

> **Important**
> This service-linked role can appear in your account if you completed an action in another service that uses the features supported by this role. If you were using the Amazon EKS service before January 1, 2017, when it began supporting service-linked roles, then Amazon EKS created the AWSServiceRoleForAmazonEKSNodegroup role in your account. To learn more, see A New Role Appeared in My IAM Account.

Creating a Service-Linked Role in Amazon EKS (AWS API)

You don't need to manually create a service-linked role. When you create a managed node group in the AWS Management Console, the AWS CLI, or the AWS API, Amazon EKS creates the service-linked role for you.

If you delete this service-linked role, and then need to create it again, you can use the same process to recreate the role in your account. When you create another managed node group, Amazon EKS creates the service-linked role for you again.

Editing a Service-Linked Role for Amazon EKS

Amazon EKS does not allow you to edit the AWSServiceRoleForAmazonEKSNodegroup service-linked role. After you create a service-linked role, you cannot change the name of the role because various entities might reference the role. However, you can edit the description of the role using IAM. For more information, see Editing a Service-Linked Role in the *IAM User Guide*.

Deleting a Service-Linked Role for Amazon EKS

If you no longer need to use a feature or service that requires a service-linked role, we recommend that you delete that role. That way you don't have an unused entity that is not actively monitored or maintained. However, you must clean up your service-linked role before you can manually delete it.

Cleaning Up a Service-Linked Role

Before you can use IAM to delete a service-linked role, you must first delete any resources used by the role.

> **Note**
> If the Amazon EKS service is using the role when you try to delete the resources, then the deletion might fail. If that happens, wait for a few minutes and try the operation again.

To delete Amazon EKS resources used by the AWSServiceRoleForAmazonEKSNodegroup

1. Open the Amazon EKS console at https://console.aws.amazon.com/eks/home#/clusters.
2. Choose a cluster.
3. On the cluster page, if there are any managed node groups in the **Node Groups** section, select each one individually, and then choose **Delete**.
4. Type the name of the cluster in the deletion confirmation window, and then choose **Confirm** to delete.
5. Repeat this procedure for any other node groups in the cluster and for any other clusters in your account.

Manually Delete the Service-Linked Role

Use the IAM console, the AWS CLI, or the AWS API to delete the AWSServiceRoleForAmazonEKSNodegroup service-linked role. For more information, see Deleting a Service-Linked Role in the *IAM User Guide*.

Supported Regions for Amazon EKS Service-Linked Roles

Amazon EKS supports using service-linked roles in all of the regions where the service is available. For more information, see Amazon EKS Service Endpoints and Quotas.

Amazon EKS cluster IAM role

Kubernetes clusters managed by Amazon EKS make calls to other AWS services on your behalf to manage the resources that you use with the service. Before you can create Amazon EKS clusters, you must create an IAM role with the following IAM policies:

- `AmazonEKSClusterPolicy`

> **Note**
> Prior to April 16, 2020, AmazonEKSServicePolicy was also required and the suggested name was `eksServiceRole`. With the `AWSServiceRoleForAmazonEKS` service-linked role, that policy is no longer required.

Check for an existing cluster role

You can use the following procedure to check and see if your account already has the Amazon EKS cluster role.

To check for the `eksClusterRole` in the IAM console

1. Open the IAM console at https://console.aws.amazon.com/iam/.
2. In the navigation pane, choose **Roles**.
3. Search the list of roles for `eksClusterRole`. If a role that includes `eksClusterRole` does not exist, then see Creating the Amazon EKS cluster role (p. 249) to create the role. If a role that includes `eksClusterRole` does exist, then select the role to view the attached policies.

4. Choose **Permissions**.

5. Ensure that the **AmazonEKSClusterPolicy** managed policy is attached to the role. If the policy is attached, your Amazon EKS cluster role is properly configured.

6. Choose **Trust Relationships**, **Edit Trust Relationship**.

7. Verify that the trust relationship contains the following policy. If the trust relationship matches the policy below, choose **Cancel**. If the trust relationship does not match, copy the policy into the **Policy Document** window and choose **Update Trust Policy**.

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

Creating the Amazon EKS cluster role

You can use the AWS Management Console or AWS CloudFormation to create the cluster role if you do not already have one for your account. Select the name of the tool that you'd like to use to create the role.

AWS Management Console

To create your Amazon EKS cluster role in the IAM console

1. Open the IAM console at https://console.aws.amazon.com/iam/.

2. Choose **Roles**, then **Create role**.

3. Choose **EKS** from the list of services, then **EKS - Cluster** for your use case, and then **Next: Permissions**.

4. Choose **Next: Tags**.

5. (Optional) Add metadata to the role by attaching tags as key–value pairs. For more information about using tags in IAM, see Tagging IAM Entities in the *IAM User Guide*.

6. Choose **Next: Review**.

7. For **Role name**, enter a unique name for your role, such as `eksClusterRole`, then choose **Create role**.

AWS CloudFormation

To create your Amazon EKS cluster role with AWS CloudFormation

1. Save the following AWS CloudFormation template to a text file on your local system.

```
---
AWSTemplateFormatVersion: '2010-09-09'
Description: 'Amazon EKS Cluster Role'


Resources:
```

```
  eksClusterRole:
    Type: AWS::IAM::Role
    Properties:
      AssumeRolePolicyDocument:
        Version: '2012-10-17'
        Statement:
        - Effect: Allow
          Principal:
            Service:
            - eks.amazonaws.com
          Action:
          - sts:AssumeRole
      ManagedPolicyArns:
        - arn:aws:iam::aws:policy/AmazonEKSClusterPolicy

Outputs:

  RoleArn:
    Description: The role that Amazon EKS will use to create AWS resources for
 Kubernetes clusters
    Value: !GetAtt eksClusterRole.Arn
    Export:
      Name: !Sub "${AWS::StackName}-RoleArn"
```

> **Note**
> Prior to April 16, 2020, `ManagedPolicyArns` had an entry for
> `arn:aws:iam::aws:policy/AmazonEKSServicePolicy`. With the
> `AWSServiceRoleForAmazonEKS` service-linked role, that policy is no longer required.

2. Open the AWS CloudFormation console at https://console.aws.amazon.com/cloudformation.

3. Choose **Create stack**.

4. For **Specify template**, select **Upload a template file**, and then choose **Choose file**.

5. Choose the file you created earlier, and then choose **Next**.

6. For **Stack name**, enter a name for your role, such as `eksClusterRole`, and then choose **Next**.

7. On the **Configure stack options** page, choose **Next**.

8. On the **Review** page, review your information, acknowledge that the stack might create IAM resources, and then choose **Create stack**.

Amazon EKS worker node IAM role

The Amazon EKS worker node `kubelet` daemon makes calls to AWS APIs on your behalf. Worker nodes receive permissions for these API calls through an IAM instance profile and associated policies. Before you can launch worker nodes and register them into a cluster, you must create an IAM role for those worker nodes to use when they are launched. This requirement applies to worker nodes launched with the Amazon EKS-optimized AMI provided by Amazon, or with any other worker node AMIs that you intend to use. Before you create worker nodes, you must create an IAM role with the following IAM policies:

- AmazonEKSWorkerNodePolicy
- AmazonEKS_CNI_Policy
- AmazonEC2ContainerRegistryReadOnly

Check for an existing worker node role

You can use the following procedure to check and see if your account already has the Amazon EKS worker node role.

To check for the `NodeInstanceRole` in the IAM console

1. Open the IAM console at https://console.aws.amazon.com/iam/.
2. In the navigation pane, choose **Roles**.
3. Search the list of roles for `NodeInstanceRole`. If a role that contains `NodeInstanceRole` does not exist, then see Creating the Amazon EKS worker node IAM role (p. 251) to create the role. If a role that contains `NodeInstanceRole` does exist, then select the role to view the attached policies.
4. Choose **Permissions**.
5. Ensure that the **AmazonEKSWorkerNodePolicy**, **AmazonEKS_CNI_Policy**, and **AmazonEC2ContainerRegistryReadOnly** managed policies are attached to the role. If the policies are attached, your Amazon EKS worker node role is properly configured.
6. Choose **Trust Relationships**, **Edit Trust Relationship**.
7. Verify that the trust relationship contains the following policy. If the trust relationship matches the policy below, choose **Cancel**. If the trust relationship does not match, copy the policy into the **Policy Document** window and choose **Update Trust Policy**.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "ec2.amazonaws.com"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
```

Creating the Amazon EKS worker node IAM role

If you created your worker nodes by following the steps in the Getting started with the AWS Management Console (p. 10) or Getting started with `eksctl` (p. 3) topics, then the worker node role already exists and you don't need to manually create it. You can use the AWS Management Console or AWS CloudFormation to create the Amazon EKS worker node role if you do not already have one for your account. Select the name of the tool that you'd like to use to create the role.

AWS Management Console

To create your Amazon EKS worker node role in the IAM console

1. Open the IAM console at https://console.aws.amazon.com/iam/.
2. Choose **Roles**, then **Create role**.
3. Choose **EC2** from the list of services, then **Next: Permissions**.
4. In the **Filter policies** box, enter `AmazonEKSWorkerNodePolicy`. Check the box to the left of **AmazonEKSWorkerNodePolicy.**
5. In the **Filter policies** box, enter `AmazonEKS_CNI_Policy`. Check the box to the left of **AmazonEKS_CNI_Policy.**
6. In the **Filter policies** box, enter `AmazonEC2ContainerRegistryReadOnly`. Check the box to the left of **AmazonEC2ContainerRegistryReadOnly.**
7. Choose **Next: Tags**.
8. (Optional) Add metadata to the role by attaching tags as key–value pairs. For more information about using tags in IAM, see Tagging IAM Entities in the *IAM User Guide*.
9. Choose **Next: Review**.

10. For **Role name**, enter a unique name for your role, such as `NodeInstanceRole`. For **Role description**, replace the current text with descriptive text such as `Amazon EKS - Node Group Role`, then choose **Create role**.

AWS CloudFormation

To create your Amazon EKS worker node role using AWS CloudFormation

1. Open the AWS CloudFormation console at https://console.aws.amazon.com/cloudformation.
2. Choose **Create stack** and then choose **With new resources (standard)**.
3. For **Specify template**, select **Amazon S3 URL**.
4. Paste the following URL into the **Amazon S3 URL** text area and choose **Next** twice:

   ```
   https://amazon-eks.s3.us-west-2.amazonaws.com/cloudformation/2020-05-08/amazon-eks-
   nodegroup-role.yaml
   ```

5. On the **Specify stack details** page, for **Stack name** enter a name such as **eks-node-group-instance-role** and choose **Next**.
6. (Optional) On the **Configure stack options** page, you can choose to tag your stack resources. Choose **Next**.
7. On the **Review** page, check the box in the **Capabilities** section and choose **Create stack**.
8. When your stack is created, select it in the console and choose **Outputs**.
9. Record the **NodeInstanceRole** value for the IAM role that was created. You need this when you create your node group.

Pod execution role

The Amazon EKS pod execution role is required to run pods on AWS Fargate infrastructure.

When your cluster creates pods on AWS Fargate infrastructure, the pod needs to make calls to AWS APIs on your behalf, for example, to pull container images from Amazon ECR. The Amazon EKS pod execution role provides the IAM permissions to do this.

When you create a Fargate profile, you must specify a pod execution role to use with your pods. This role is added to the cluster's Kubernetes Role based access control (RBAC) for authorization, so that the `kubelet` that is running on the Fargate infrastructure can register with your Amazon EKS cluster. This is what allows Fargate infrastructure to appear in your cluster as nodes.

Before you create a Fargate profile, you must create an IAM role with the following IAM policy:

• `AmazonEKSFargatePodExecutionRolePolicy`

Check for an existing pod execution role

You can use the following procedure to check and see if your account already has the Amazon EKS pod execution role.

To check for the `AmazonEKSFargatePodExecutionRole` in the IAM console

1. Open the IAM console at https://console.aws.amazon.com/iam/.
2. In the navigation pane, choose **Roles**.
3. Search the list of roles for `AmazonEKSFargatePodExecutionRole`. If the role does not exist, see Creating the Amazon EKS pod execution role (p. 253) to create the role. If the role does exist, select the role to view the attached policies.

4. Choose **Permissions**.

5. Ensure that the **AmazonEKSFargatePodExecutionRolePolicy** Amazon managed policy is attached to the role. If the policy is attached, then your Amazon EKS pod execution role is properly configured.

6. Choose **Trust Relationships**, **Edit Trust Relationship**.

7. Verify that the trust relationship contains the following policy. If the trust relationship matches the policy below, choose **Cancel**. If the trust relationship does not match, copy the policy into the **Policy Document** window and choose **Update Trust Policy**.

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

Creating the Amazon EKS pod execution role

You can use the following procedure to create the Amazon EKS pod execution role if you do not already have one for your account.

1. Open the IAM console at https://console.aws.amazon.com/iam/.

2. Choose **Roles**, then **Create role**.

3. Choose **EKS** from the list of services, **EKS - Fargate pod** for your use case, and then **Next: Permissions**.

4. Choose **Next: Tags**.

5. (Optional) Add metadata to the role by attaching tags as key–value pairs. For more information about using tags in IAM, see Tagging IAM Entities in the *IAM User Guide*.

6. Choose **Next: Review**.

7. For **Role name**, enter a unique name for your role, such as `AmazonEKSFargatePodExecutionRole`, then choose **Create role**.

IAM roles for service accounts

With IAM roles for service accounts on Amazon EKS clusters, you can associate an IAM role with a Kubernetes service account. This service account can then provide AWS permissions to the containers in any pod that uses that service account. With this feature, you no longer need to provide extended permissions to the worker node IAM role so that pods on that node can call AWS APIs.

Applications must sign their AWS API requests with AWS credentials. This feature provides a strategy for managing credentials for your applications, similar to the way that Amazon EC2 instance profiles provide credentials to Amazon EC2 instances. Instead of creating and distributing your AWS credentials to the containers or using the Amazon EC2 instance's role, you can associate an IAM role with a Kubernetes service account. The applications in the pod's containers can then use an AWS SDK or the AWS CLI to make API requests to authorized AWS services.

The IAM roles for service accounts feature provides the following benefits:

- **Least privilege —** By using the IAM roles for service accounts feature, you no longer need to provide extended permissions to the worker node IAM role so that pods on that node can call AWS APIs. You can scope IAM permissions to a service account, and only pods that use that service account have access to those permissions. This feature also eliminates the need for third-party solutions such as `kiam` or `kube2iam`.
- **Credential isolation —** A container can only retrieve credentials for the IAM role that is associated with the service account to which it belongs. A container never has access to credentials that are intended for another container that belongs to another pod.
- **Auditability —** Access and event logging is available through CloudTrail to help ensure retrospective auditing.

To get started, see Enabling IAM roles for service accounts on your cluster (p. 258).

For an end-to-end walkthrough using `eksctl`, see Walkthrough: Updating a DaemonSet to use IAM for service accounts (p. 264).

IAM roles for service accounts technical overview

In 2014, AWS Identity and Access Management added support for federated identities using OpenID Connect (OIDC). This feature allows you to authenticate AWS API calls with supported identity providers and receive a valid OIDC JSON web token (JWT). You can pass this token to the AWS STS `AssumeRoleWithWebIdentity` API operation and receive IAM temporary role credentials. You can use these credentials to interact with any AWS service, like Amazon S3 and DynamoDB.

Kubernetes has long used service accounts as its own internal identity system. Pods can authenticate with the Kubernetes API server using an auto-mounted token (which was a non-OIDC JWT) that only the Kubernetes API server could validate. These legacy service account tokens do not expire, and rotating the signing key is a difficult process. In Kubernetes version 1.12, support was added for a new `ProjectedServiceAccountToken` feature, which is an OIDC JSON web token that also contains the service account identity, and supports a configurable audience.

Amazon EKS now hosts a public OIDC discovery endpoint per cluster containing the signing keys for the `ProjectedServiceAccountToken` JSON web tokens so external systems, like IAM, can validate and accept the OIDC tokens issued by Kubernetes.

IAM role configuration

In IAM, you create an IAM role with a trust relationship that is scoped to your cluster's OIDC provider, the service account namespace, and (optionally) the service account name, and then attach the IAM policy that you want to associate with the service account. You can add multiple entries in the `StringEquals` and `StringLike` conditions below to use multiple service accounts or namespaces with the role.

- To scope a role to a specific service account:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Federated": "arn:aws:iam::AWS_ACCOUNT_ID:oidc-provider/OIDC_PROVIDER"
      },
      "Action": "sts:AssumeRoleWithWebIdentity",
      "Condition": {
        "StringEquals": {
          "OIDC_PROVIDER:sub":
 "system:serviceaccount:SERVICE_ACCOUNT_NAMESPACE:SERVICE_ACCOUNT_NAME"
        }
      }
```

```
      }
    ]
}
```

- To scope a role to an entire namespace (to use the namespace as a boundary):

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "Federated": "arn:aws:iam::AWS_ACCOUNT_ID:oidc-provider/OIDC_PROVIDER"
            },
            "Action": "sts:AssumeRoleWithWebIdentity",
            "Condition": {
                "StringLike": {
                    "OIDC_PROVIDER:sub": "system:serviceaccount:SERVICE_ACCOUNT_NAMESPACE:*"
                }
            }
        }
    ]
}
```

Service account configuration

In Kubernetes, you define the IAM role to associate with a service account in your cluster by adding the `eks.amazonaws.com/role-arn` annotation to the service account.

```
apiVersion: v1
kind: ServiceAccount
metadata:
  annotations:
    eks.amazonaws.com/role-arn: arn:aws:iam::AWS_ACCOUNT_ID:role/IAM_ROLE_NAME
```

Pod configuration

The Amazon EKS Pod Identity Webhook on the cluster watches for pods that are associated with service accounts with this annotation and applies the following environment variables to them.

```
AWS_ROLE_ARN=arn:aws:iam::AWS_ACCOUNT_ID:role/IAM_ROLE_NAME
AWS_WEB_IDENTITY_TOKEN_FILE=/var/run/secrets/eks.amazonaws.com/serviceaccount/token
```

> **Note**
> Your cluster does not need to use the mutating web hook to configure the environment variables and token file mounts; you can choose to configure pods to add these environment variables manually.

Supported versions of the AWS SDK (p. 257) look for these environment variables first in the credential chain provider. The role credentials are used for pods that meet this criteria.

> **Note**
> When a pod uses AWS credentials from an IAM role associated with a service account, the AWS CLI or other SDKs in the containers for that pod use the credentials provided by that role exclusively. They no longer inherit any IAM permissions from the worker node IAM role.

By default, only containers that run as `root` have the proper file system permissions to read the web identity token file. You can provide these permissions by having your containers run as `root`, or by providing the following security context for the containers in your manifest. The `fsGroup` ID is arbitrary,

and you can choose any valid group ID. For more information about the implications of setting a security context for your pods, see Configure a Security Context for a Pod or Container in the Kubernetes documentation.

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: my-app
spec:
  template:
    metadata:
      labels:
        app: my-app
    spec:
      serviceAccountName: my-app
      containers:
      - name: my-app
        image: my-app:latest
      securityContext:
        fsGroup: 1337
...
```

Cross-account IAM permissions

You can configure cross-account IAM permissions either by creating an identity provider from another account's cluster or by using chained AssumeRole operations. In the following examples, Account A owns an Amazon EKS cluster that supports IAM roles for service accounts. Pods running on that cluster need to assume IAM permissions from Account B.

Example : Create an identity provider from another account's cluster

Example

In this example, Account A would provide Account B with the OIDC issuer URL from their cluster. Account B follows the instructions in Enabling IAM roles for service accounts on your cluster (p. 258) and Creating an IAM role and policy for your service account (p. 259) using the OIDC issuer URL from Account A's cluster. Then a cluster administrator annotates the service account in Account A's cluster to use the role from Account B.

```
apiVersion: v1
kind: ServiceAccount
metadata:
  annotations:
    eks.amazonaws.com/role-arn: arn:aws:iam::ACCOUNT_B_AWS_ACCOUNT_ID:role/IAM_ROLE_NAME
```

Example : Use chained `AssumeRole` operations

Example

In this example, Account B creates an IAM policy with the permissions to give to pods in Account A's cluster. Account B attaches that policy to an IAM role with a trust relationship that allows `AssumeRole` permissions to Account A (`111111111111`), as shown below.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "AWS": "arn:aws:iam::111111111111:root"
      },
```

```
      "Action": "sts:AssumeRole",
      "Condition": {}
    }
  ]
}
```

Account A creates a role with a trust policy that gets credentials from the identity provider created with the cluster's OIDC issuer URL, as shown below.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Federated": "arn:aws:iam::111111111111:oidc-provider/oidc.eks.region-code.amazonaws.com/id/EXAMPLEC061A78C479E31025A21AC4CDE191335D05820BE5CE"
      },
      "Action": "sts:AssumeRoleWithWebIdentity"
    }
  ]
}
```

Account A attaches a policy to that role with the following permissions to assume the role that Account B created.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "sts:AssumeRole",
            "Resource": "arn:aws:iam::222222222222:role/account-b-role"
        }
    ]
}
```

The application code for pods to assume Account B's role uses two profiles: `account_b_role` and `account_a_role`. The `account_b_role` profile uses the `account_a_role` profile as its source. For the AWS CLI, the `~/.aws/config` file would look like the following example.

```
[profile account_b_role]
source_profile = account_a_role
role_arn=arn:aws:iam::222222222222:role/account-b-role

[profile account_a_role]
web_identity_token_file = /var/run/secrets/eks.amazonaws.com/serviceaccount/token
role_arn=arn:aws:iam::111111111111:role/account-a-role
```

To specify chained profiles for other AWS SDKs, consult their documentation.

Using a supported AWS SDK

The containers in your pods must use an AWS SDK version that supports assuming an IAM role via an OIDC web identity token file. AWS SDKs that are included in Linux distribution package managers may not be new enough to support this feature. Be sure to use at least the minimum SDK versions listed below:

- Java (Version 2) — 2.10.11
- Java — 1.11.704

- Go — 1.23.13
- Python (Boto3) — 1.9.220
- Python (botocore) — 1.12.200
- AWS CLI — 1.16.232
- Node — 2.521.0
- Ruby — 2.11.345
- C++ — 1.7.174
- .NET — 3.3.659.1
- PHP — 3.110.7

Many popular Kubernetes add-ons, such as the Cluster Autoscaler and the ALB Ingress Controller support IAM roles for service accounts. The Amazon VPC CNI plugin for Kubernetes has been updated with a supported version of the AWS SDK for Go, and you can use the IAM roles for service accounts feature to provide the required permissions for the CNI to work.

To ensure that you are using a supported SDK, follow the installation instructions for your preferred SDK at Tools for Amazon Web Services when you build your containers.

Enabling IAM roles for service accounts on your cluster

The IAM roles for service accounts feature is available on new Amazon EKS Kubernetes version 1.14 and later clusters, and clusters that were updated to versions 1.13 or later on or after September 3rd, 2019. Existing clusters can update to version 1.13 or later to take advantage of this feature. For more information, see Updating an Amazon EKS cluster Kubernetes version (p. 27).

If your cluster supports IAM roles for service accounts, it will have an OpenID Connect issuer URL associated with it. You can view this URL in the Amazon EKS console, or you can use the following AWS CLI command to retrieve it.

> **Important**
> You must use at least version 1.18.49 of the AWS CLI to receive the proper output from this command. For more information, see Installing the AWS CLI in the *AWS Command Line Interface User Guide*.

```
aws eks describe-cluster --name cluster_name --query "cluster.identity.oidc.issuer" --output text
```

Output:

```
https://oidc.eks.region-code.amazonaws.com/id/EXAMPLED539D4633E53DE1B716D3041E
```

To use IAM roles for service accounts in your cluster, you must create an OIDC identity provider in the IAM console.

eksctl

To create an IAM OIDC identity provider for your cluster with `eksctl`

1. Check your `eksctl` version with the following command. This procedure assumes that you have installed `eksctl` and that your `eksctl` version is at least `0.19.0`.

   ```
   eksctl version
   ```

   For more information about installing or upgrading `eksctl`, see Installing or upgrading eksctl (p. 197).

2. Create your OIDC identity provider for your cluster with the following command. Substitute *cluster_name* with your own value.

```
eksctl utils associate-iam-oidc-provider --cluster cluster_name --approve
```

AWS Management Console

To create an IAM OIDC identity provider for your cluster with the AWS Management Console

1. Retrieve the OIDC issuer URL from the Amazon EKS console description of your cluster or use the following AWS CLI command.

   Important
   You must use at least version 1.18.49 of the AWS CLI to receive the proper output from this command. For more information, see Installing the AWS CLI in the *AWS Command Line Interface User Guide*.

   ```
   aws eks describe-cluster --name cluster_name --query "cluster.identity.oidc.issuer"
    --output text
   ```

2. Open the IAM console at https://console.aws.amazon.com/iam/.
3. In the navigation pane, choose **Identity Providers**, and then choose **Create Provider**.
4. For **Provider Type**, choose **Choose a provider type**, and then choose **OpenID Connect**.
5. For **Provider URL**, paste the OIDC issuer URL for your cluster.
6. For Audience, type `sts.amazonaws.com` and choose **Next Step**.
7. Verify that the provider information is correct, and then choose **Create** to create your identity provider.

After you have enabled the IAM OIDC identity provider for your cluster, you can create IAM roles to associate with a service account in your cluster. For more information, see Creating an IAM role and policy for your service account (p. 259)

Creating an IAM role and policy for your service account

You must create an IAM policy that specifies the permissions that you would like the containers in your pods to have. You have several ways to create a new IAM permission policy. One way is to copy a complete AWS managed policy that already does some of what you're looking for and then customize it to your specific requirements. For more information, see Creating a New Policy in the *IAM User Guide*.

You must also create an IAM role for your Kubernetes service accounts to use before you associate it with a service account. The trust relationship is scoped to your cluster and service account so that each cluster and service account combination requires its own role. You can then attach a specific IAM policy to the role that gives the containers in your pods the permissions you desire. The following procedures describe how to do this.

Create an IAM policy

In this procedure, we offer two example policies that you can use for your application:

- A policy to allow read-only access to an Amazon S3 bucket. You could store configuration information or a bootstrap script in this bucket, and the containers in your pod can read the file from the bucket and load it into your application.
- A policy to allow paid container images from AWS Marketplace.

1. Open the IAM console at https://console.aws.amazon.com/iam/.

2. In the navigation pane, choose **Policies** and then choose **Create policy**.

3. Choose the **JSON** tab.

4. In the **Policy Document** field, paste one of the following policies to apply to your service accounts, or paste your own policy document into the field. You can also use the visual editor to construct your own policy.

   The example below allows permission to the *my-pod-secrets-bucket* Amazon S3 bucket. You can modify the policy document to suit your specific needs.

   ```
   {
     "Version": "2012-10-17",
     "Statement": [
       {
         "Effect": "Allow",
         "Action": [
           "s3:GetObject"
         ],
         "Resource": [
           "arn:aws:s3:::my-pod-secrets-bucket/*"
         ]
       }
     ]
   }
   ```

   The example below gives the required permissions to use a paid container image from AWS Marketplace.

   ```
   {
     "Version": "2012-10-17",
     "Statement": [
       {
         "Action": [
           "aws-marketplace:RegisterUsage"
         ],
         "Effect": "Allow",
         "Resource": "*"
       }
     ]
   }
   ```

5. Choose **Review policy**.

6. Enter a name and description for your policy and then choose **Create policy**.

7. Record the Amazon Resource Name (ARN) of the policy to use later when you create your role.

Create an IAM role

Create an IAM role for your service accounts. Select the tab with the name of the tool that you want to use to create the role.

eksctl

Create the service account and IAM role with the following command. Substitute the *example values* with your own values.

> **Note**
> This command only works for clusters that were created with `eksctl`. If you didn't create your cluster with `eksctl`, then use the instructions on the AWS Management Console or AWS CLI tabs.

```
eksctl create iamserviceaccount \
    --name service_account_name \
    --namespace service_account_namespace \
    --cluster cluster_name \
    --attach-policy-arn IAM_policy_ARN \
    --approve \
    --override-existing-serviceaccounts
```

An AWS CloudFormation template was deployed that created an IAM role and attached the IAM policy to it. The role was associated with a Kubernetes service account.

AWS Management Console

1. Retrieve the OIDC issuer URL from the Amazon EKS console description of your cluster, or use the following AWS CLI command.

   Important
   You must use at least version 1.18.49 of the AWS CLI to receive the proper output from this command. For more information, see Installing the AWS CLI in the *AWS Command Line Interface User Guide*.

   ```
   aws eks describe-cluster --name cluster_name --query "cluster.identity.oidc.issuer"
    --output text
   ```

2. Open the IAM console at https://console.aws.amazon.com/iam/.

3. In the navigation pane, choose **Roles**, **Create Role**.

4. In the **Select type of trusted entity** section, choose **Web identity**.

5. In the **Choose a web identity provider** section:

   1. For **Identity provider**, choose the URL for your cluster.

   2. For **Audience**, choose `sts.amazonaws.com`.

6. Choose **Next: Permissions**.

7. In the **Attach Policy** section, select the policy to use for your service account. Choose **Next: Tags**.

8. On the **Add tags (optional)** screen, you can add tags for the account. Choose **Next: Review**.

9. For **Role Name**, enter a name for your role and then choose **Create Role**.

10. After the role is created, choose the role in the console to open it for editing.

11. Choose the **Trust relationships** tab, and then choose **Edit trust relationship**.

    1. Edit the OIDC provider suffix and change it from `:aud` to `:sub`.

    2. Replace `sts.amazonaws.com` with your service account ID.

    The resulting line should look like this.

    ```
    "oidc.eks.region-code.amazonaws.com/id/EXAMPLED539D4633E53DE1B716D3041E:sub":
     "system:serviceaccount:SERVICE_ACCOUNT_NAMESPACE:SERVICE_ACCOUNT_NAME"
    ```

12. Choose **Update Trust Policy** to finish.

13. Associate the IAM role with a Kubernetes service account. For more information, see Specifying an IAM role for your service account (p. 262).

AWS CLI

1. Set your AWS account ID to an environment variable with the following command.

```
AWS_ACCOUNT_ID=$(aws sts get-caller-identity --query "Account" --output text)
```

2. Set your OIDC identity provider to an environment variable with the following command, replacing your cluster name.

    Important
    You must use at least version 1.18.49 of the AWS CLI to receive the proper output from this command. For more information, see Installing the AWS CLI in the *AWS Command Line Interface User Guide*.

    ```
    OIDC_PROVIDER=$(aws eks describe-cluster --name cluster-name --query
     "cluster.identity.oidc.issuer" --output text | sed -e "s/^https:\/\///")
    ```

3. Copy the following code block to your computer and replace *namespace* and *service-account-name* with your own values.

    ```
    read -r -d '' TRUST_RELATIONSHIP <<EOF
    {
      "Version": "2012-10-17",
      "Statement": [
        {
          "Effect": "Allow",
          "Principal": {
            "Federated": "arn:aws:iam::${AWS_ACCOUNT_ID}:oidc-provider/
    ${OIDC_PROVIDER}"
          },
          "Action": "sts:AssumeRoleWithWebIdentity",
          "Condition": {
            "StringEquals": {
              "${OIDC_PROVIDER}:sub": "system:serviceaccount:namespace:service-account-
    name"
            }
          }
        }
      ]
    }
    EOF
    echo "${TRUST_RELATIONSHIP}" > trust.json
    ```

4. Run the modified code block from the previous step to create a file named `trust.json`.

5. Run the following AWS CLI command to create the role, replacing your IAM role name and description.

    ```
    aws iam create-role --role-name IAM_ROLE_NAME --assume-role-policy-document file://
    trust.json --description "IAM_ROLE_DESCRIPTION"
    ```

6. Run the following command to attach your IAM policy to your role, replacing your IAM role name and policy ARN.

    ```
    aws iam attach-role-policy --role-name IAM_ROLE_NAME --policy-arn=IAM_POLICY_ARN
    ```

7. Associate the IAM role with a Kubernetes service account. For more information, see Specifying an IAM role for your service account (p. 262).

Specifying an IAM role for your service account

In Kubernetes, you define the IAM role to associate with a service account in your cluster by adding the following annotation to the service account.

Note
If you created an IAM role to use with your service account using `eksctl`, this has already been done for you with the service account that you specified when creating the role.

```
apiVersion: v1
kind: ServiceAccount
metadata:
  annotations:
    eks.amazonaws.com/role-arn: arn:aws:iam::AWS_ACCOUNT_ID:role/IAM_ROLE_NAME
```

To patch a service account to use with IAM roles

1. Use the following command to annotate your service account with the ARN of the IAM role that you want to use with your service account. Be sure to substitute your own values for the *alternate-colored* example values to use with your pods.

    ```
    kubectl annotate serviceaccount -n SERVICE_ACCOUNT_NAMESPACE SERVICE_ACCOUNT_NAME \
    eks.amazonaws.com/role-arn=arn:aws:iam::AWS_ACCOUNT_ID:role/IAM_ROLE_NAME
    ```

2. Delete and re-create any existing pods that are associated with the service account to apply the credential environment variables. The mutating web hook does not apply them to pods that are already running. The following command deletes the existing the `aws-node` DaemonSet pods and deploys them with the service account annotation. You can modify the namespace, deployment type, and label to update your specific pods.

    ```
    kubectl delete pods -n kube-system -l k8s-app=aws-node
    ```

3. Confirm that the pods all restarted.

    ```
    kubectl get pods -n kube-system  -l k8s-app=aws-node
    ```

4. Describe one of the pods and verify that the `AWS_WEB_IDENTITY_TOKEN_FILE` and `AWS_ROLE_ARN` environment variables exist.

    ```
    kubectl exec -n kube-system aws-node-9rgzw env | grep AWS
    ```

    Output:

    ```
    AWS_VPC_K8S_CNI_LOGLEVEL=DEBUG
    AWS_ROLE_ARN=arn:aws:iam::AWS_ACCOUNT_ID:role/IAM_ROLE_NAME
    AWS_WEB_IDENTITY_TOKEN_FILE=/var/run/secrets/eks.amazonaws.com/serviceaccount/token
    ```

    The IAM role was created by `eksctl` when you created the Kubernetes service account in a previous step.

Restricting access to Amazon EC2 instance profile credentials

By default, all containers that are running on a worker node have all permissions assigned to the that is attached to the worker node. The Amazon EC2 instance metadata service provides the credentials to any process running on an instance. For more information, see Retrieving Security Credentials from Instance Metadata.

When you implement IAM roles for service accounts for a pod, the containers in the pod have all permissions assigned to the service account and the worker node IAM role. If you implement IAM roles for service accounts for all pods in a cluster, you may want to prevent the containers in the pods from using the permissions assigned to the worker node IAM role. Keep in mind however, that there may

be certain key permissions on the worker node IAM role that pods need to function. It's important to properly scope your service account IAM roles so that your pods have all of the necessary permissions. For example, the worker node IAM role is assigned permissions to pull container images from Amazon ECR. If a pod isn't assigned those permissions, then the pod can't pull container images from Amazon ECR.

To prevent all containers in all pods on a worker node from using the permissions assigned to the worker node IAM role (while still allowing the permissions that are assigned to the service account), run the following `iptables` commands on your worker nodes (as `root`) or include them in your instance bootstrap user data script.

> **Important**
>
> These commands completely block **all** containers running on a worker node from querying the instance metadata service for any metadata, not just the credentials for the worker node IAM role. Do not run these commands on worker nodes that run pods that you haven't implemented IAM roles for service accounts for or none of the containers on the node will have any of the permissions assigned to the worker node IAM role.

```
yum install -y iptables-services
iptables --insert FORWARD 1 --in-interface eni+ --destination 169.254.169.254/32 --jump
 DROP
iptables-save | tee /etc/sysconfig/iptables
systemctl enable --now iptables
```

Walkthrough: Updating a DaemonSet to use IAM for service accounts

The Amazon VPC CNI plugin for Kubernetes is the networking plugin for pod networking in Amazon EKS clusters. The CNI plugin is responsible for allocating VPC IP addresses to Kubernetes nodes and configuring the necessary networking for pods on each node. The plugin requires IAM permissions, provided by the AWS managed policy `AmazonEKS_CNI_Policy`, to make calls to AWS APIs on your behalf. By default, this policy is attached to your worker node IAM role. However, using this method, all pods on the worker nodes have the same permissions as the CNI plugin. You can use the IAM roles for service accounts feature to provide the `AmazonEKS_CNI_Policy` permissions, and then remove the policy from the worker node IAM role.

For ease of use, this topic uses `eksctl` to configure IAM roles for service accounts. However, if you would rather use the AWS Management Console, the AWS CLI, or one of the AWS SDKs, the same basic concepts apply, but you will have to modify the steps to use the procedures in Enabling IAM roles for service accounts on your cluster (p. 258)

To configure the CNI plugin to use IAM roles for service accounts

1. Check your `eksctl` version with the following command. This procedure assumes that you have installed `eksctl` and that your `eksctl` version is at least `0.19.0`.

    ```
    eksctl version
    ```

    For more information about installing or upgrading `eksctl`, see Installing or upgrading `eksctl` (p. 197).

2. Check the version of your cluster's Amazon VPC CNI Plugin for Kubernetes. Use the following command to print your cluster's CNI version.

    ```
    kubectl describe daemonset aws-node --namespace kube-system | grep Image | cut -d "/" -
    f 2
    ```

    Output:

```
amazon-k8s-cni:1.5.7
```

If your CNI version is earlier than 1.6.1, complete the following steps to create a service account and then upgrade your CNI version to the latest version:

a. Create an OIDC identity provider for your cluster with the following command. Substitute the cluster name with your own value.

```
eksctl utils associate-iam-oidc-provider --cluster cluster_name --approve
```

b. Create a Kubernetes service account with the following command. Substitute *cluster_name* with your own value. This command deploys an AWS CloudFormation stack that creates an IAM role, attaches the `AmazonEKS_CNI_Policy` AWS managed policy to it, and binds the IAM role to the service account.

```
eksctl create iamserviceaccount \
    --name aws-node \
    --namespace kube-system \
    --cluster cluster_name \
    --attach-policy-arn arn:aws:iam::aws:policy/AmazonEKS_CNI_Policy \
    --approve \
    --override-existing-serviceaccounts
```

c. Upgrade your CNI version to the latest version. The manifest specifies the `aws-node` service account that you created in the previous step.

```
kubectl apply -f https://raw.githubusercontent.com/aws/amazon-vpc-cni-k8s/
release-1.6/config/v1.6/aws-k8s-cni.yaml
```

3. Watch the roll out, and wait for the `DESIRED` count of the deployment to match the `UP-TO-DATE` count. Press **Ctrl + c** to exit.

```
kubectl get -n kube-system daemonset.apps/aws-node --watch
```

4. List the pods in the `aws-node` DaemonSet.

```
kubectl get pods -n kube-system  -l k8s-app=aws-node
```

Output:

```
NAME             READY    STATUS        RESTARTS    AGE
aws-node-mp88b   1/1      Running       0           17m
aws-node-n4tcd   1/1      Running       0           20s
aws-node-qt9dl   1/1      Running       0           17m
```

5. Check the version of your cluster's Amazon VPC CNI Plugin for Kubernetes again, confirming that the version is 1.6.1.

```
kubectl describe daemonset aws-node --namespace kube-system | grep Image | cut -d "/" -
f 2
```

Output:

```
amazon-k8s-cni:1.6.1
```

6. Describe one of the pods and verify that the `AWS_WEB_IDENTITY_TOKEN_FILE` and `AWS_ROLE_ARN` environment variables exist.

```
kubectl exec -n kube-system aws-node-9rgzw env | grep AWS
```

Output:

```
AWS_VPC_K8S_CNI_LOGLEVEL=DEBUG
AWS_ROLE_ARN=arn:aws:iam::111122223333:role/eksctl-prod-addon-iamserviceaccount-kube-
sys-Role1-V66K5I6JLDGK
AWS_WEB_IDENTITY_TOKEN_FILE=/var/run/secrets/eks.amazonaws.com/serviceaccount/token
```

The IAM role was created by `eksctl` when you created the Kubernetes service account in a previous step.

7. Remove the `AmazonEKS_CNI_Policy` policy from your worker node IAM role.

   a. Open the IAM console at https://console.aws.amazon.com/iam/.

   b. In the left navigation, choose **Roles**, and then search for your node instance role.

   c. Choose the **Permissions** tab for your node instance role and then choose the **X** to the right of the `AmazonEKS_CNI_Policy`.

   d. Choose **Detach** to finish.

Now your CNI plugin pods are getting their IAM permissions from their own role, and the instance role no longer can provide those permissions to other pods.

Troubleshooting Amazon EKS identity and access

To diagnose and fix common issues that you might encounter when working with Amazon EKS and IAM see Troubleshooting IAM (p. 291).

Logging and monitoring in Amazon EKS

Amazon EKS control plane logging provides audit and diagnostic logs directly from the Amazon EKS control plane to CloudWatch Logs in your account. These logs make it easy for you to secure and run your clusters. You can select the exact log types you need, and logs are sent as log streams to a group for each Amazon EKS cluster in CloudWatch. For more information, see Amazon EKS Control Plane Logging (p. 40).

Note
When you check the Amazon EKS authenticator logs in Amazon CloudWatch, you'll see entries that contain text similar to the following example text.

```
level=info msg="mapping IAM role" groups="[]"
 role="arn:aws:iam::111122223333:role/XXXXXXXXXXXXXXXXXX-NodeManagerRole-XXXXXXXX"
 username="eks:node-manager"
```

Entries that contain this text are expected. The `username` is an Amazon EKS internal service role that performs specific operations for managed node groups and Fargate.

Amazon EKS is integrated with AWS CloudTrail, a service that provides a record of actions taken by a user, role, or an AWS service in Amazon EKS. CloudTrail captures all API calls for Amazon EKS as events. The calls captured include calls from the Amazon EKS console and code calls to the Amazon EKS API operations. For more information, see Logging Amazon EKS API Calls with AWS CloudTrail (p. 278).

The Kubernetes API server exposes a number of metrics that are useful for monitoring and analysis. For more information, see .

Compliance validation for Amazon EKS

Third-party auditors assess the security and compliance of Amazon EKS as part of multiple AWS compliance programs. These include SOC, PCI, ISO, HIPAA, and others.

For a list of AWS services in scope of specific compliance programs, see AWS services in scope by compliance program. For general information, see AWS compliance programs.

You can download third-party audit reports using AWS Artifact. For more information, see Downloading reports in AWS Artifact.

Your compliance responsibility when using Amazon EKS is determined by the sensitivity of your data, your company's compliance objectives, and applicable laws and regulations. AWS provides the following resources to help with compliance:

- Security and compliance quick start guides – These deployment guides discuss architectural considerations and provide steps for deploying security- and compliance-focused baseline environments on AWS.
- Architecting for HIPAA security and compliance whitepaper – This whitepaper describes how companies can use AWS to create HIPAA-compliant applications.
- AWS compliance resources – This collection of workbooks and guides might apply to your industry and location.
- AWS Config – This AWS service assesses how well your resource configurations comply with internal practices, industry guidelines, and regulations.
- AWS Security Hub – This AWS service provides a comprehensive view of your security state within AWS that helps you check your compliance with security industry standards and best practices.

Resilience in Amazon EKS

The AWS global infrastructure is built around AWS Regions and Availability Zones. AWS Regions provide multiple physically separated and isolated Availability Zones, which are connected with low-latency, high-throughput, and highly redundant networking. With Availability Zones, you can design and operate applications and databases that automatically fail over between Availability Zones without interruption. Availability Zones are more highly available, fault tolerant, and scalable than traditional single or multiple data center infrastructures.

Amazon EKS runs Kubernetes control plane instances across multiple Availability Zone to ensure high availability. Amazon EKS automatically detects and replaces unhealthy control plane instances, and it provides automated version upgrades and patching for them.

This control plane consists of at least two API server nodes and three `etcd` nodes that run across three Availability Zones within a Region. Amazon EKS automatically detects and replaces unhealthy control plane instances, restarting them across the Availability Zones within the Region as needed. Amazon EKS leverages the architecture of AWS Regions in order to maintain high availability. Because of this, Amazon EKS is able to offer an SLA for API server endpoint availability.

For more information about AWS Regions and Availability Zones, see AWS global infrastructure.

Infrastructure security in Amazon EKS

As a managed service, Amazon EKS is protected by the AWS global network security procedures that are described in the Amazon Web Services: Overview of security processes whitepaper.

You use AWS published API calls to access Amazon EKS through the network. Clients must support Transport Layer Security (TLS) 1.0 or later. We recommend TLS 1.2 or later. Clients must also support cipher suites with perfect forward secrecy (PFS) such as Ephemeral Diffie-Hellman (DHE) or Elliptic Curve Ephemeral Diffie-Hellman (ECDHE). Most modern systems such as Java 7 and later support these modes.

Additionally, requests must be signed by using an access key ID and a secret access key that is associated with an IAM principal. Or you can use the AWS Security Token Service (AWS STS) to generate temporary security credentials to sign requests.

When you create an Amazon EKS cluster, you specify the VPC subnets for your cluster to use. Amazon EKS requires subnets in at least two Availability Zones. We recommend a VPC with public and private subnets so that Kubernetes can create public load balancers in the public subnets that load balance traffic to pods running on worker nodes that are in private subnets.

For more information about VPC considerations, see Cluster VPC considerations (p. 155).

If you create your VPC and worker node groups with the AWS CloudFormation templates provided in the Getting started with Amazon EKS (p. 3) walkthrough, then your control plane and worker node security groups are configured with our recommended settings.

For more information about security group considerations, see Amazon EKS security group considerations (p. 157).

When you create a new cluster, Amazon EKS creates an endpoint for the managed Kubernetes API server that you use to communicate with your cluster (using Kubernetes management tools such as `kubectl`). By default, this API server endpoint is public to the internet, and access to the API server is secured using a combination of AWS Identity and Access Management (IAM) and native Kubernetes Role Based Access Control (RBAC).

You can enable private access to the Kubernetes API server so that all communication between your worker nodes and the API server stays within your VPC. You can limit the IP addresses that can access your API server from the internet, or completely disable internet access to the API server.

For more information about modifying cluster endpoint access, see Modifying cluster endpoint access (p. 35).

You can implement network policies with tools such as Project Calico (p. 177). Project Calico is a third party open source project. For more information, see the Project Calico documentation.

Configuration and vulnerability analysis in Amazon EKS

Amazon EKS platform versions represent the capabilities of the cluster control plane, including which Kubernetes API server flags are enabled and the current Kubernetes patch version. New clusters are deployed with the latest platform version. For details, see Platform versions (p. 49).

You can update an Amazon EKS cluster (p. 27) to newer Kubernetes versions. As new Kubernetes versions become available in Amazon EKS, we recommend that you proactively update your clusters to use the latest available version. For more information about Kubernetes versions in EKS, see Amazon EKS Kubernetes Versions (p. 45).

Track security or privacy events for Amazon Linux 2 at the Amazon Linux Security Center or subscribe to the associated RSS feed. Security and privacy events include an overview of the issue affected, packages, and instructions for updating your instances to correct the issue.

You can use Amazon Inspector to check for unintended network accessibility of your worker nodes and for vulnerabilities on those Amazon EC2 instances.

Pod security policy

The Kubernetes pod security policy admission controller validates pod creation and update requests against a set of rules. By default, Amazon EKS clusters ship with a fully permissive security policy with no restrictions. For more information, see Pod Security Policies in the Kubernetes documentation.

> **Note**
> The pod security policy admission controller is only enabled on Amazon EKS clusters running Kubernetes version 1.13 or later. You must update your cluster's Kubernetes version to at least 1.13 to use pod security policies. For more information, see Updating an Amazon EKS cluster Kubernetes version (p. 27).

Amazon EKS default pod security policy

Amazon EKS clusters with Kubernetes version 1.13 and higher have a default pod security policy named `eks.privileged`. This policy has no restriction on what kind of pod can be accepted into the system, which is equivalent to running Kubernetes with the `PodSecurityPolicy` controller disabled.

> **Note**
> This policy was created to maintain backwards compatibility with clusters that did not have the `PodSecurityPolicy` controller enabled. You can create more restrictive policies for your cluster and for individual namespaces and service accounts and then delete the default policy to enable the more restrictive policies.

You can view the default policy with the following command.

```
kubectl get psp eks.privileged
```

Output:

```
NAME              PRIV    CAPS    SELINUX     RUNASUSER    FSGROUP     SUPGROUP    READONLYROOTFS
   VOLUMES
eks.privileged    true    *       RunAsAny    RunAsAny     RunAsAny    RunAsAny    false
   *
```

For more details, you can describe the policy with the following command.

```
kubectl describe psp eks.privileged
```

Output:

```
Name:  eks.privileged

Settings:
  Allow Privileged:                 true
  Allow Privilege Escalation:       0xc0004ce5f8
  Default Add Capabilities:         <none>
  Required Drop Capabilities:       <none>
```

```
  Allowed Capabilities:                    *
  Allowed Volume Types:                    *
  Allow Host Network:                      true
  Allow Host Ports:                        0-65535
  Allow Host PID:                          true
  Allow Host IPC:                          true
  Read Only Root Filesystem:               false
  SELinux Context Strategy: RunAsAny
    User:                                  <none>
    Role:                                  <none>
    Type:                                  <none>
    Level:                                 <none>
  Run As User Strategy: RunAsAny
    Ranges:                                <none>
  FSGroup Strategy: RunAsAny
    Ranges:                                <none>
  Supplemental Groups Strategy: RunAsAny
    Ranges:                                <none>
```

The following example shows the full YAML file for the `eks.privileged` pod security policy, its cluster role, and cluster role binding.

```
---
apiVersion: policy/v1beta1
kind: PodSecurityPolicy
metadata:
  name: eks.privileged
  annotations:
    kubernetes.io/description: 'privileged allows full unrestricted access to
      pod features, as if the PodSecurityPolicy controller was not enabled.'
    seccomp.security.alpha.kubernetes.io/allowedProfileNames: '*'
  labels:
    kubernetes.io/cluster-service: "true"
    eks.amazonaws.com/component: pod-security-policy
spec:
  privileged: true
  allowPrivilegeEscalation: true
  allowedCapabilities:
  - '*'
  volumes:
  - '*'
  hostNetwork: true
  hostPorts:
  - min: 0
    max: 65535
  hostIPC: true
  hostPID: true
  runAsUser:
    rule: 'RunAsAny'
  seLinux:
    rule: 'RunAsAny'
  supplementalGroups:
    rule: 'RunAsAny'
  fsGroup:
    rule: 'RunAsAny'
  readOnlyRootFilesystem: false

---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: eks:podsecuritypolicy:privileged
  labels:
    kubernetes.io/cluster-service: "true"
```

```
        eks.amazonaws.com/component: pod-security-policy
rules:
- apiGroups:
  - policy
  resourceNames:
  - eks.privileged
  resources:
  - podsecuritypolicies
  verbs:
  - use

---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: eks:podsecuritypolicy:authenticated
  annotations:
    kubernetes.io/description: 'Allow all authenticated users to create privileged pods.'
  labels:
    kubernetes.io/cluster-service: "true"
    eks.amazonaws.com/component: pod-security-policy
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: eks:podsecuritypolicy:privileged
subjects:
  - kind: Group
    apiGroup: rbac.authorization.k8s.io
    name: system:authenticated
```

To delete the default pod security policy

After you create custom pod security policies for your cluster, you can delete the default Amazon EKS `eks.privileged` pod security policy to enable your custom policies.

1. Create a file called `privileged-podsecuritypolicy.yaml` and paste the full `eks.privileged` YAML file contents from the preceding example into it (this allows you to delete the pod security policy, the `ClusterRole`, and the `ClusterRoleBinding` associated with it).

2. Delete the YAML with the following command.

    ```
    kubectl delete -f privileged-podsecuritypolicy.yaml
    ```

To install or restore the default pod security policy

If you are upgrading from an earlier version of Kubernetes, or have modified or deleted the default Amazon EKS `eks.privileged` pod security policy, you can restore it with the following steps.

1. Create a file called `privileged-podsecuritypolicy.yaml` and paste the YAML file contents below into it.

    ```
    ---
    apiVersion: policy/v1beta1
    kind: PodSecurityPolicy
    metadata:
      name: eks.privileged
      annotations:
        kubernetes.io/description: 'privileged allows full unrestricted access to
          pod features, as if the PodSecurityPolicy controller was not enabled.'
        seccomp.security.alpha.kubernetes.io/allowedProfileNames: '*'
      labels:
    ```

```
        kubernetes.io/cluster-service: "true"
        eks.amazonaws.com/component: pod-security-policy
spec:
  privileged: true
  allowPrivilegeEscalation: true
  allowedCapabilities:
  - '*'
  volumes:
  - '*'
  hostNetwork: true
  hostPorts:
  - min: 0
    max: 65535
  hostIPC: true
  hostPID: true
  runAsUser:
    rule: 'RunAsAny'
  seLinux:
    rule: 'RunAsAny'
  supplementalGroups:
    rule: 'RunAsAny'
  fsGroup:
    rule: 'RunAsAny'
  readOnlyRootFilesystem: false

---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: eks:podsecuritypolicy:privileged
  labels:
    kubernetes.io/cluster-service: "true"
    eks.amazonaws.com/component: pod-security-policy
rules:
- apiGroups:
  - policy
  resourceNames:
  - eks.privileged
  resources:
  - podsecuritypolicies
  verbs:
  - use

---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: eks:podsecuritypolicy:authenticated
  annotations:
    kubernetes.io/description: 'Allow all authenticated users to create privileged
 pods.'
  labels:
    kubernetes.io/cluster-service: "true"
    eks.amazonaws.com/component: pod-security-policy
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: eks:podsecuritypolicy:privileged
subjects:
  - kind: Group
    apiGroup: rbac.authorization.k8s.io
    name: system:authenticated
```

2. Apply the YAML with the following command.

```
kubectl apply -f privileged-podsecuritypolicy.yaml
```

Tagging your Amazon EKS resources

To help you manage your Amazon EKS resources, you can assign your own metadata to each resource in the form of *tags*. This topic describes tags and shows you how to create them.

Contents

Tag basics

A tag is a label that you assign to an AWS resource. Each tag consists of a *key* and an optional *value*, both of which you define.

Tags enable you to categorize your AWS resources by, for example, purpose, owner, or environment. When you have many resources of the same type, you can quickly identify a specific resource based on the tags you've assigned to it. For example, you can define a set of tags for your Amazon EKS clusters to help you track each cluster's owner and stack level. We recommend that you devise a consistent set of tag keys for each resource type. You can then search and filter the resources based on the tags that you add.

Tags are not automatically assigned to your resources. After you add a tag, you can edit tag keys and values or remove tags from a resource at any time. If you delete a resource, any tags for the resource are also deleted.

Tags don't have any semantic meaning to Amazon EKS and are interpreted strictly as a string of characters. You can set the value of a tag to an empty string, but you can't set the value of a tag to null. If you add a tag that has the same key as an existing tag on that resource, the new value overwrites the old value.

You can work with tags using the AWS Management Console, the AWS CLI, and the Amazon EKS API.

> **Note**
> Amazon EKS tags are not currently supported by `eksctl`.

If you're using AWS Identity and Access Management (IAM), you can control which users in your AWS account have permission to create, edit, or delete tags.

Tagging your resources

You can tag new or existing Amazon EKS clusters and managed node groups.

If you're using the Amazon EKS console, you can apply tags to new resources when they are created or to existing resources at any time using the **Tags** tab on the relevant resource page.

If you're using the Amazon EKS API, the AWS CLI, or an AWS SDK, you can apply tags to new resources using the `tags` parameter on the relevant API action or to existing resources using the `TagResource` API action. For more information, see TagResource.

Some resource-creating actions enable you to specify tags for a resource when the resource is created. If tags cannot be applied during resource creation, the resource creation process fails. This ensures that resources you intended to tag on creation are either created with specified tags or not created at all. If you tag resources at the time of creation, you don't need to run custom tagging scripts after resource creation.

The following table describes the Amazon EKS resources that can be tagged, and the resources that can be tagged on creation.

Tagging support for Amazon EKS resources

Resource	Supports tags	Supports tag propagation	Supports tagging on creation (Amazon EKS API, AWS CLI, AWS SDK)
Amazon EKS clusters	Yes	No. Cluster tags do not propagate to any other resources associated with the cluster.	Yes
Amazon EKS managed node groups	Yes	No. Managed node group tags do not propagate to any other resources associated with the node group.	Yes

Tag restrictions

The following basic restrictions apply to tags:

- Maximum number of tags per resource – 50
- For each resource, each tag key must be unique, and each tag key can have only one value.
- Maximum key length – 128 Unicode characters in UTF-8
- Maximum value length – 256 Unicode characters in UTF-8
- If your tagging schema is used across multiple AWS services and resources, remember that other services may have restrictions on allowed characters. Generally allowed characters are letters, numbers, spaces representable in UTF-8, and the following characters: + - = . _ : / @.
- Tag keys and values are case sensitive.
- Don't use `aws:`, `AWS:`, or any upper or lowercase combination of such as a prefix for either keys or values, as it is reserved for AWS use. You can't edit or delete tag keys or values with this prefix. Tags with this prefix do not count against your tags-per-resource limit.

Working with tags using the console

Using the Amazon EKS console, you can manage the tags associated with new or existing clusters and managed node groups.

When you select a resource-specific page in the Amazon EKS console, it displays a list of those resources. For example, if you select **Clusters** from the navigation pane, the console displays a list of Amazon EKS clusters. When you select a resource from one of these lists (for example, a specific cluster), if the resource supports tags, you can view and manage its tags on the **Tags** tab.

Adding tags on an individual resource on creation

You can add tags to Amazon EKS clusters and managed node groups when you create them. For more information, see Creating an Amazon EKS cluster (p. 21)

Adding and deleting tags on an individual resource

Amazon EKS allows you to add or delete tags associated with your clusters directly from the resource's page.

To add or delete a tag on an individual resource

1. Open the Amazon EKS console at https://console.aws.amazon.com/eks/home#/clusters.
2. From the navigation bar, select the Region to use.
3. In the navigation pane, choose **Clusters**.
4. Choose a specific cluster, then scroll down and choose **Manage tags**.
5. On the **Update tags** page, add or delete your tags as necessary.

   - To add a tag — choose **Add tag** and then specify the key and value for each tag.
   - To delete a tag — choose **Remove tag**.
6. Repeat this process for each tag you want to add or delete, and then choose **Update** to finish.

Working with tags using the CLI or API

Use the following AWS CLI commands or Amazon EKS API operations to add, update, list, and delete the tags for your resources.

Tagging support for Amazon EKS resources

Task	AWS CLI	AWS Tools for Windows PowerShell	API action
Add or overwrite one or more tags.	tag-resource	Add-EKSResourceTag	TagResource
Delete one or more tags.	untag-resource	Remove-EKSResourceTag	UntagResource

The following examples show how to tag or untag resources using the AWS CLI.

Example 1: Tag an existing cluster

The following command tags an existing cluster.

```
aws eks tag-resource --resource-arn resource_ARN --tags team=devs
```

Example 2: Untag an existing cluster

The following command deletes a tag from an existing cluster.

```
aws eks untag-resource --resource-arn resource_ARN --tag-keys tag_key
```

Example 3: List tags for a resource

The following command lists the tags associated with an existing resource.

```
aws eks list-tags-for-resource --resource-arn resource_ARN
```

Some resource-creating actions enable you to specify tags when you create the resource. The following actions support tagging on creation.

Task	AWS CLI	AWS Tools for Windows PowerShell	API action
Create a cluster	create-cluster	New-EKSCluster	CreateCluster
Create a managed node group	create-nodegroup	New-EKSNodegroup	CreateNodegroup

Logging Amazon EKS API calls with AWS CloudTrail

Amazon EKS is integrated with AWS CloudTrail, a service that provides a record of actions taken by a user, role, or an AWS service in Amazon EKS. CloudTrail captures all API calls for Amazon EKS as events. The calls captured include calls from the Amazon EKS console and code calls to the Amazon EKS API operations.

If you create a trail, you can enable continuous delivery of CloudTrail events to an Amazon S3 bucket, including events for Amazon EKS. If you don't configure a trail, you can still view the most recent events in the CloudTrail console in **Event history**. Using the information collected by CloudTrail, you can determine the request that was made to Amazon EKS, the IP address from which the request was made, who made the request, when it was made, and additional details.

To learn more about CloudTrail, see the AWS CloudTrail User Guide.

Amazon EKS information in CloudTrail

CloudTrail is enabled on your AWS account when you create the account. When activity occurs in Amazon EKS, that activity is recorded in a CloudTrail event along with other AWS service events in **Event history**. You can view, search, and download recent events in your AWS account. For more information, see Viewing events with CloudTrail event history.

For an ongoing record of events in your AWS account, including events for Amazon EKS, create a trail. A *trail* enables CloudTrail to deliver log files to an Amazon S3 bucket. By default, when you create a trail in the console, the trail applies to all AWS Regions. The trail logs events from all Regions in the AWS partition and delivers the log files to the Amazon S3 bucket that you specify. Additionally, you can configure other AWS services to further analyze and act upon the event data collected in CloudTrail logs. For more information, see the following:

- Overview for creating a trail
- CloudTrail supported services and integrations
- Configuring Amazon SNS notifications for CloudTrail
- Receiving CloudTrail log files from multiple regions and Receiving CloudTrail log files from multiple accounts

All Amazon EKS actions are logged by CloudTrail and are documented in the Amazon EKS API Reference. For example, calls to the `CreateCluster`, `ListClusters` and `DeleteCluster` sections generate entries in the CloudTrail log files.

Every event or log entry contains information about who generated the request. The identity information helps you determine the following:

- Whether the request was made with root or AWS Identity and Access Management (IAM) user credentials.
- Whether the request was made with temporary security credentials for a role or federated user.
- Whether the request was made by another AWS service.

For more information, see the CloudTrail userIdentity element.

Understanding Amazon EKS log file entries

A trail is a configuration that enables delivery of events as log files to an Amazon S3 bucket that you specify. CloudTrail log files contain one or more log entries. An event represents a single request from any source and includes information about the requested action, the date and time of the action, request parameters, and so on. CloudTrail log files aren't an ordered stack trace of the public API calls, so they don't appear in any specific order.

The following example shows a CloudTrail log entry that demonstrates the `CreateCluster` action.

```
{
  "eventVersion": "1.05",
  "userIdentity": {
    "type": "IAMUser",
    "principalId": "AKIAIOSFODNN7EXAMPLE",
    "arn": "arn:aws:iam::111122223333:user/username",
    "accountId": "111122223333",
    "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
    "userName": "username"
  },
  "eventTime": "2018-05-28T19:16:43Z",
  "eventSource": "eks.amazonaws.com",
  "eventName": "CreateCluster",
  "awsRegion": "region-code",
  "sourceIPAddress": "205.251.233.178",
  "userAgent": "PostmanRuntime/6.4.0",
  "requestParameters": {
    "resourcesVpcConfig": {
      "subnetIds": [
        "subnet-a670c2df",
        "subnet-4f8c5004"
      ]
    },
    "roleArn": "arn:aws:iam::111122223333:role/AWSServiceRoleForAmazonEKS-CAC1G1VH3ZKZ",
    "clusterName": "test"
  },
  "responseElements": {
    "cluster": {
      "clusterName": "test",
      "status": "CREATING",
      "createdAt": 1527535003.208,
      "certificateAuthority": {},
      "arn": "arn:aws:eks:region-code:111122223333:cluster/test",
      "roleArn": "arn:aws:iam::111122223333:role/AWSServiceRoleForAmazonEKS-CAC1G1VH3ZKZ",
      "version": "1.10",
      "resourcesVpcConfig": {
        "securityGroupIds": [],
        "vpcId": "vpc-21277358",
        "subnetIds": [
          "subnet-a670c2df",
          "subnet-4f8c5004"
        ]
      }
    }
  },
  "requestID": "a7a0735d-62ab-11e8-9f79-81ce5b2b7d37",
  "eventID": "eab22523-174a-499c-9dd6-91e7be3ff8e3",
  "readOnly": false,
  "eventType": "AwsApiCall",
  "recipientAccountId": "111122223333"
```

```
}
```

Log Entries for Amazon EKS Service Linked Roles

The Amazon EKS service linked roles make API calls to AWS resources. You will see
CloudTrail log entries with `username: AWSServiceRoleForAmazonEKS` and `username:
AWSServiceRoleForAmazonEKSNodegroup` for calls made by the Amazon EKS service linked roles.
For more information about Amazon EKS and service linked roles, see the section called "Using Service-
Linked Roles" (p. 244).

The following example shows a CloudTrail log entry that demonstrates a `DeleteInstanceProfile`
action made by the `AWSServiceRoleForAmazonEKSNodegroup` service linked role, noted in the
`sessionContext`.

```
{
    "eventVersion": "1.05",
    "userIdentity": {
        "type": "AssumedRole",
        "principalId": "AROA3WHGPEZ7SJ2CW55C5:EKS",
        "arn": "arn:aws:sts::111122223333:assumed-role/AWSServiceRoleForAmazonEKSNodegroup/
EKS",
        "accountId": "111122223333",
        "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
        "sessionContext": {
            "sessionIssuer": {
                "type": "Role",
                "principalId": "AROA3WHGPEZ7SJ2CW55C5",
                "arn": "arn:aws:iam::111122223333:role/aws-service-role/eks-
nodegroup.amazonaws.com/AWSServiceRoleForAmazonEKSNodegroup",
                "accountId": "111122223333",
                "userName": "AWSServiceRoleForAmazonEKSNodegroup"
            },
            "webIdFederationData": {},
            "attributes": {
                "mfaAuthenticated": "false",
                "creationDate": "2020-02-26T00:56:33Z"
            }
        },
        "invokedBy": "eks-nodegroup.amazonaws.com"
    },
    "eventTime": "2020-02-26T00:56:34Z",
    "eventSource": "iam.amazonaws.com",
    "eventName": "DeleteInstanceProfile",
    "awsRegion": "us-east-1",
    "sourceIPAddress": "eks-nodegroup.amazonaws.com",
    "userAgent": "eks-nodegroup.amazonaws.com",
    "requestParameters": {
        "instanceProfileName": "eks-11111111-2222-3333-4444-abcdef123456"
    },
    "responseElements": null,
    "requestID": "11111111-2222-3333-4444-abcdef123456",
    "eventID": "11111111-2222-3333-4444-abcdef123456",
    "eventType": "AwsApiCall",
    "recipientAccountId": "111122223333"
}
```

Amazon EKS on AWS Outposts

Beginning with Kubernetes version 1.14.8 with Amazon EKS platform version `eks.5` and Kubernetes version 1.13.12 with Amazon EKS platform version `eks.6`, you can create and run Amazon EKS nodes on AWS Outposts. AWS Outposts enables native AWS services, infrastructure, and operating models in on-premises facilities. In AWS Outposts environments, you can use the same AWS APIs, tools, and infrastructure that you use in the AWS Cloud. Amazon EKS worker nodes on AWS Outposts is ideal for low-latency workloads that need to be run in close proximity to on-premises data and applications. For more information about AWS Outposts, see the AWS Outposts User Guide.

Prerequisites

The following are the prerequisites for using Amazon EKS worker nodes on AWS Outposts:

- You must have installed and configured an Outpost in your on-premises data center.
- You must have a reliable network connection between your Outpost and its AWS Region.
- The AWS Region for the Outpost must support Amazon EKS. For a list of supported Regions, see Amazon EKS service endpoints in the *AWS General Reference*.

Limitations

The following are the limitations of using Amazon EKS on Outposts:

- AWS Identity and Access Management, Application Load Balancer, Network Load Balancer, Classic Load Balancer, and Amazon Route 53 run in the AWS Region, not on Outposts. This will increase latencies between the services and the containers.
- AWS Fargate is not available on AWS Outposts.

Network connectivity considerations

The following are network connectivity considerations for Amazon EKS AWS Outposts:

- If network connectivity between your Outpost and its AWS Region is lost, your nodes will continue to run. However, you cannot create new nodes or take new actions on existing deployments until connectivity is restored. In case of instance failures, the instance will not be automatically replaced. The Kubernetes master runs in the Region, and missing heartbeats caused by things like a loss of connectivity to the Availability Zone could lead to failures. The failed heartbeats will lead to pods on the Outposts being marked as unhealthy, and eventually the node status will time out and pods will be marked for eviction. For more information, see Node Controller in the Kubernetes documentation.
- We recommend that you provide reliable, highly available, and low-latency connectivity between your Outpost and its AWS Region.

Creating Amazon EKS nodes on an Outpost

Creating Amazon EKS nodes on an Outpost is similar to creating Amazon EKS nodes in the AWS Cloud. When you create an Amazon EKS node on an Outpost, you must specify a subnet associated with your Outpost.

An Outpost is an extension of an AWS Region, and you can extend a VPC in an account to span multiple Availability Zones and any associated Outpost locations. When you configure your Outpost, you associate a subnet with it to extend your Regional VPC environment to your on-premises facility. Instances on an Outpost appear as part of your Regional VPC, similar to an Availability Zone with associated subnets.
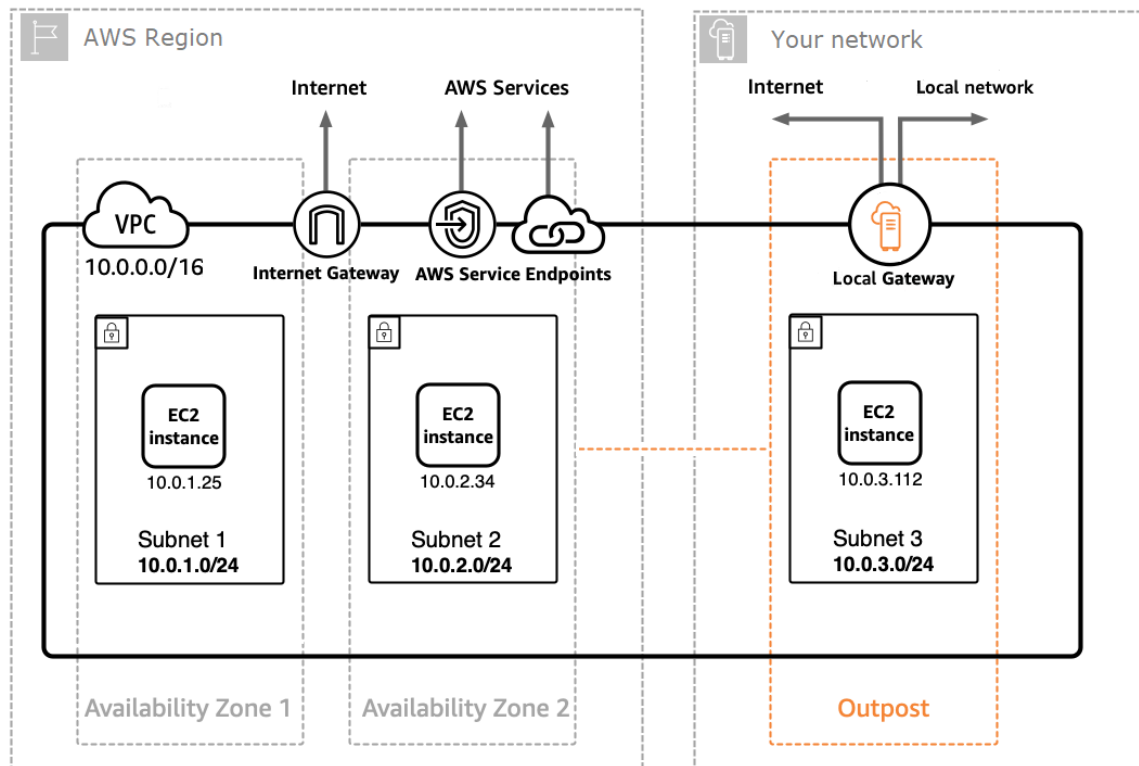
To create Amazon EKS nodes on an Outpost with the AWS CLI, specify a security group and a subnet associated with your Outpost.

To create an Amazon EKS node group on an Outpost

1. Create a VPC.

    ```
    aws ec2 create-vpc --cidr-block 10.0.0.0/16
    ```

2. Create Outpost subnets. The `--outpost-arn` parameter must be specified for the subnet to be created for the Outpost. (This step is different for AWS Outposts.)

    ```
    aws ec2 create-subnet --vpc-id vpc-xxxxxxxx --cidr-block 10.0.3.0/24 \
       --outpost-arn  arn:aws:outposts:us-west-2:123456789012:outpost/op-xxxxxxxxxxxxxxxx
    ```

3. Create a cluster, specifying the subnets for the Outpost. (This step is different for AWS Outposts.)

    ```
    aws eks --region region-code create-cluster --name eks-outpost --role-arn \
       arn:aws:iam::123456789012:role/eks-service-role-AWSServiceRoleForAmazonEKS-OUTPOST \
       --resources-vpc-config  subnetIds=subnet-xxxxxxxx,subnet-
    yyyyyyyy,securityGroupIds=sg-xxxxxxxx
    ```

4. Create the node group. Specify an instance type that is available on your Outpost. (This step is different for AWS Outposts.)

```
eksctl create nodegroup --cluster eks-outpost \
       --version    auto \
       --name       outpost-workers \
       --node-type  c5.large \
       --node-ami   auto \
       --nodes      3 \
       --nodes-min  1 \
       --nodes-max  4
```

5. Deploy applications and services.

```
kubectl apply -f kubernetes/deployment.yaml
```

Related projects

These open source projects extend the functionality of Kubernetes clusters running on AWS, including clusters managed by Amazon EKS.

Management tools

Related management tools for Amazon EKS and Kubernetes clusters.

eksctl

`eksctl` is a simple CLI tool for creating clusters on Amazon EKS.

- Project URL: https://eksctl.io/
- Project documentation: https://eksctl.io/
- AWS open source blog: eksctl: Amazon EKS cluster with one command

AWS service operator

AWS Service Operator allows you to create AWS resources using `kubectl`.

- Project URL: https://github.com/aws/aws-service-operator-k8s
- Project documentation: https://github.com/aws/aws-service-operator-k8s/blob/master/README.md
- AWS open source blog: AWS service operator for Kubernetes now available

Networking

Related networking projects for Amazon EKS and Kubernetes clusters.

Amazon VPC CNI plugin for Kubernetes

Amazon EKS supports native VPC networking via the Amazon VPC CNI plugin for Kubernetes. Using this CNI plugin allows Kubernetes pods to have the same IP address inside the pod as they do on the VPC network. For more information, see Pod networking (CNI) (p. 160) and CNI configuration variables (p. 161).

- Project URL: https://github.com/aws/amazon-vpc-cni-k8s
- Project documentation: https://github.com/aws/amazon-vpc-cni-k8s/blob/master/README.md

AWS Application Load Balancer (ALB) ingress controller for Kubernetes

The AWS ALB Ingress Controller satisfies Kubernetes ingress resources by provisioning Application Load Balancers.

- Project URL: https://github.com/kubernetes-sigs/aws-alb-ingress-controller
- Project documentation: https://github.com/kubernetes-sigs/aws-alb-ingress-controller/tree/master/docs
- AWS open source blog: Kubernetes ingress with AWS ALB ingress controller

ExternalDNS

ExternalDNS synchronizes exposed Kubernetes services and ingresses with DNS providers including Amazon Route 53 and AWS Service Discovery.

- Project URL: https://github.com/kubernetes-incubator/external-dns
- Project documentation: https://github.com/kubernetes-incubator/external-dns/blob/master/docs/tutorials/aws.md

Security

Related security projects for Amazon EKS and Kubernetes clusters.

AWS IAM authenticator

A tool to use AWS IAM credentials to authenticate to a Kubernetes cluster. For more information, see Installing `aws-iam-authenticator` (p. 187).

- Project URL: https://github.com/kubernetes-sigs/aws-iam-authenticator
- Project documentation: https://github.com/kubernetes-sigs/aws-iam-authenticator/blob/master/README.md
- AWS open source blog: Deploying the AWS IAM authenticator to kops

Machine learning

Related machine learning projects for Amazon EKS and Kubernetes clusters.

Kubeflow

A machine learning toolkit for Kubernetes.

- Project URL: https://www.kubeflow.org/
- Project documentation: https://www.kubeflow.org/docs/
- AWS open source blog: Kubeflow on Amazon EKS

Auto Scaling

Related auto scaling projects for Amazon EKS and Kubernetes clusters.

Cluster autoscaler

Cluster Autoscaler is a tool that automatically adjusts the size of the Kubernetes cluster based on CPU and memory pressure.

- Project URL: https://github.com/kubernetes/autoscaler/tree/master/cluster-autoscaler
- Project documentation: https://github.com/kubernetes/autoscaler/blob/master/cluster-autoscaler/cloudprovider/aws/README.md
- Amazon EKS workshop: https://eksworkshop.com/scaling/deploy_ca/

Escalator

Escalator is a batch or job optimized horizontal autoscaler for Kubernetes.

- Project URL: https://github.com/atlassian/escalator
- Project documentation: https://github.com/atlassian/escalator/blob/master/docs/README.md

Monitoring

Related monitoring projects for Amazon EKS and Kubernetes clusters.

Prometheus

Prometheus is an open-source systems monitoring and alerting toolkit.

- Project URL: https://prometheus.io/
- Project documentation: https://prometheus.io/docs/introduction/overview/
- Amazon EKS workshop: https://eksworkshop.com/intermediate/240_monitoring/

Continuous integration / continuous deployment

Related CI/CD projects for Amazon EKS and Kubernetes clusters.

Jenkins X

CI/CD solution for modern cloud applications on Amazon EKS and Kubernetes clusters.

- Project URL: https://jenkins-x.io/
- Project documentation: https://jenkins-x.io/documentation/
- AWS open source blog: Continuous delivery with Amazon EKS and Jenkins X

Amazon EKS troubleshooting

This chapter covers some common errors that you may see while using Amazon EKS and how to work around them.

Insufficient capacity

If you receive the following error while attempting to create an Amazon EKS cluster, then one of the Availability Zones you specified does not have sufficient capacity to support a cluster.

Cannot create cluster *'example-cluster'* because *region-1d*, the targeted Availability Zone, does not currently have sufficient capacity to support the cluster. Retry and choose from these Availability Zones: *region-1a*, *region-1b*, *region-1c*

Retry creating your cluster with subnets in your cluster VPC that are hosted in the Availability Zones returned by this error message.

`aws-iam-authenticator` Not found

If you receive the error `"aws-iam-authenticator": executable file not found in $PATH`, then your **kubectl** is not configured for Amazon EKS. For more information, see Installing aws-iam-authenticator (p. 187).

Worker nodes fail to join cluster

There are a few common reasons that prevent worker nodes from joining the cluster:

- The `aws-auth-cm.yaml` file does not have the correct IAM role ARN for your worker nodes. Ensure that the worker node IAM role ARN (not the instance profile ARN) is specified in your `aws-auth-cm.yaml` file. For more information, see Launching Amazon EKS Linux worker nodes (p. 92).
- The **ClusterName** in your worker node AWS CloudFormation template does not exactly match the name of the cluster you want your worker nodes to join. Passing an incorrect value to this field results in an incorrect configuration of the worker node's `/var/lib/kubelet/kubeconfig` file, and the nodes will not join the cluster.
- The worker node is not tagged as being *owned* by the cluster. Your worker nodes must have the following tag applied to them, where `<cluster_name>` is replaced with the name of your cluster.

Key	Value
`kubernetes.io/cluster/<cluster-name>`	owned

- The worker nodes may not be able to access the cluster using a public IP address. Ensure that worker nodes deployed in public subnets are assigned a public IP address. If not, you can associate an elastic IP address to a worker node after it's launched. For more information, see Associating an elastic IP address with a running instance or network interface. If the public subnet is not set to automatically assign public IP addresses to instances deployed to it, then we recommend enabling that setting. For more information, see Modifying the public IPv4 addressing attribute for your subnet. If the worker node is deployed to a private subnet, then the subnet must have a route to a NAT gateway that has a public IP address assigned to it.

Unauthorized or access denied (`kubectl`)

If you receive one of the following errors while running **kubectl** commands, then your **kubectl** is not configured properly for Amazon EKS or the IAM user or role credentials that you are using do not map to a Kubernetes RBAC user with sufficient permissions in your Amazon EKS cluster.

- `could not get token: AccessDenied: Access denied`
- `error: You must be logged in to the server (Unauthorized)`
- `error: the server doesn't have a resource type "svc"`

This could be because the cluster was created with one set of AWS credentials (from an IAM user or role), and **kubectl** is using a different set of credentials.

When an Amazon EKS cluster is created, the IAM entity (user or role) that creates the cluster is added to the Kubernetes RBAC authorization table as the administrator (with `system:master` permissions). Initially, only that IAM user can make calls to the Kubernetes API server using **kubectl**. For more information, see Managing users or IAM roles for your cluster (p. 193). Also, the AWS IAM authenticator for Kubernetes uses the AWS SDK for Go to authenticate against your Amazon EKS cluster. If you use the console to create the cluster, you must ensure that the same IAM user credentials are in the AWS SDK credential chain when you are running **kubectl** commands on your cluster.

If you install and configure the AWS CLI, you can configure the IAM credentials for your user. If the AWS CLI is configured properly for your user, then the AWS IAM authenticator for Kubernetes can find those credentials as well. For more information, see Configuring the AWS CLI in the *AWS Command Line Interface User Guide*.

If you assumed a role to create the Amazon EKS cluster, you must ensure that **kubectl** is configured to assume the same role. Use the following command to update your kubeconfig file to use an IAM role. For more information, see Create a `kubeconfig` for Amazon EKS (p. 190).

```
aws --region region-code eks update-kubeconfig --name cluster_name --role-arn
 arn:aws:iam::aws_account_id:role/role_name
```

To map an IAM user to a Kubernetes RBAC user, see Managing users or IAM roles for your cluster (p. 193) or watch a video about how to map a user.

hostname doesn't match

Your system's Python version must be 2.7.9 or later. Otherwise, you receive `hostname doesn't match` errors with AWS CLI calls to Amazon EKS. For more information, see What are "hostname doesn't match" errors? in the Python Requests FAQ.

getsockopt: no route to host

Docker runs in the `172.17.0.0/16` CIDR range in Amazon EKS clusters. We recommend that your cluster's VPC subnets do not overlap this range. Otherwise, you will receive the following error:

```
Error: : error upgrading connection: error dialing backend: dial tcp 172.17.nn.nn:10250:
 getsockopt: no route to host
```

Managed node group errors

If you receive the error "Instances failed to join the kubernetes cluster" in the AWS Management Console, ensure that either the cluster's private endpoint access is enabled, or that you have correctly configured CIDR blocks for public endpoint access. For more information, see Amazon EKS cluster endpoint access control (p. 35).

If your managed node group encounters a health issue, Amazon EKS returns an error message to help you to diagnose the issue. The following error messages and their associated descriptions are shown below.

- **AutoScalingGroupNotFound**: We couldn't find the Auto Scaling group associated with the managed node group. You may be able to recreate an Auto Scaling group with the same settings to recover.
- **Ec2SecurityGroupNotFound**: We couldn't find the cluster security group for the cluster. You must recreate your cluster.
- **Ec2SecurityGroupDeletionFailure**: We could not delete the remote access security group for your managed node group. Remove any dependencies from the security group.
- **Ec2LaunchTemplateNotFound**: We couldn't find the Amazon EC2 launch template for your managed node group. You may be able to recreate a launch template with the same settings to recover.
- **Ec2LaunchTemplateVersionMismatch**: The Amazon EC2 launch template version for your managed node group does not match the version that Amazon EKS created. You may be able to revert to the version that Amazon EKS created to recover.
- **IamInstanceProfileNotFound**: We couldn't find the IAM instance profile for your managed node group. You may be able to recreate an instance profile with the same settings to recover.
- **IamNodeRoleNotFound**: We couldn't find the IAM role for your managed node group. You may be able to recreate an IAM role with the same settings to recover.
- **AsgInstanceLaunchFailures**: Your Auto Scaling group is experiencing failures while attempting to launch instances.
- **NodeCreationFailure**: Your launched instances are unable to register with your Amazon EKS cluster. Common causes of this failure are insufficient worker node IAM role (p. 250) permissions or lack of outbound internet access for the nodes. Your worker nodes must be able to access the internet using a public IP address to function properly. For more information, see ??? (p. 155). Your worker nodes must also have ports open to the internet. For more information, see ??? (p. 157).
- **InstanceLimitExceeded**: Your AWS account is unable to launch any more instances of the specified instance type. You may be able to request an Amazon EC2 instance limit increase to recover.
- **InsufficientFreeAddresses**: One or more of the subnets associated with your managed node group does not have enough available IP addresses for new nodes.
- **AccessDenied**: Amazon EKS or one or more of your managed nodes is unable to communicate with your cluster API server.
- **InternalFailure**: These errors are usually caused by an Amazon EKS server-side issue.

CNI log collection tool

The Amazon VPC CNI plugin for Kubernetes has its own troubleshooting script (which is available on worker nodes at `/opt/cni/bin/aws-cni-support.sh`) that you can use to collect diagnostic logs for support cases and general troubleshooting.

Use the following command to run the script on your worker node:

```
sudo bash /opt/cni/bin/aws-cni-support.sh
```

> **Note**
> If the script is not present at that location, then the CNI container failed to run. You can manually download and run the script with the following command:
>
> ```
> curl -O https://raw.githubusercontent.com/awslabs/amazon-eks-ami/master/log-
> collector-script/linux/eks-log-collector.sh
> sudo bash eks-log-collector.sh
> ```

The script collects the following diagnostic information. The CNI version that you have deployed can be earlier than the script version.

```
        This is version 0.6.1. New versions can be found at https://github.com/awslabs/
amazon-eks-ami

Trying to collect common operating system logs...
Trying to collect kernel logs...
Trying to collect mount points and volume information...
Trying to collect SELinux status...
Trying to collect iptables information...
Trying to collect installed packages...
Trying to collect active system services...
Trying to collect Docker daemon information...
Trying to collect kubelet information...
Trying to collect L-IPAMD information...
Trying to collect sysctls information...
Trying to collect networking infomation...
Trying to collect CNI configuration information...
Trying to collect running Docker containers and gather container data...
Trying to collect Docker daemon logs...
Trying to archive gathered information...

 Done... your bundled logs are located in /var/log/eks_i-0717c9d54b6cfaa19_2020-03-24_0103-
UTC_0.6.1.tar.gz
```

The diagnostic information is collected and stored at

```
/var/log/eks_i-0717c9d54b6cfaa19_2020-03-24_0103-UTC_0.6.1.tar.gz
```

Container runtime network not ready

You may receive a `Container runtime network not ready` error and authorization errors similar to the following:

```
4191 kubelet.go:2130] Container runtime network not ready: NetworkReady=false
 reason:NetworkPluginNotReady message:docker: network plugin is not ready: cni config
 uninitialized
4191 reflector.go:205] k8s.io/kubernetes/pkg/kubelet/kubelet.go:452: Failed to list
 *v1.Service: Unauthorized
4191 kubelet_node_status.go:106] Unable to register node "ip-10-40-175-122.ec2.internal"
 with API server: Unauthorized
4191 reflector.go:205] k8s.io/kubernetes/pkg/kubelet/kubelet.go:452: Failed to list
 *v1.Service: Unauthorized
```

The errors are most likely related to the AWS IAM Authenticator configuration map not being applied to the worker nodes. The configuration map provides the `system:bootstrappers` and `system:nodes` Kubernetes RBAC permissions for worker nodes to register to the cluster. For more information, see **To enable worker nodes to join your cluster** on the **Self-managed nodes** tab of Launching Amazon

. Ensure that you specify the **Role ARN** of the instance role in the configuration map, not the **Instance Profile ARN**.

The authenticator does not recognize a **Role ARN** if it includes a path other than /, such as the following example:

```
arn:aws:iam::111122223333:role/development/apps/prod-iam-role-NodeInstanceRole-621LVEXAMPLE
```

When specifying a **Role ARN** in the configuration map that includes a path other than /, you must drop the path. The ARN above would be specified as the following:

```
arn:aws:iam::111122223333:role/prod-iam-role-NodeInstanceRole-621LVEXAMPLE
```

TLS handshake timeout

When a worker node is unable to establish a connection to the public API server endpoint, you may an error similar to the following error.

```
server.go:233] failed to run Kubelet: could not init cloud provider "aws": error finding
 instance i-1111f2222f333e44c: "error listing AWS instances: \"RequestError: send request
 failed\\ncaused by: Post  net/http: TLS handshake timeout\""
```

The `kubelet` process will continually respawn and test the API server endpoint. The error can also occur temporarily during any procedure that performs a rolling update of the cluster in the control plane, such as a configuration change or version update.

To resolve the issue, check the route table and security groups to ensure that traffic from the worker nodes can reach the public endpoint.

Error: ErrImagePull

If you have 1.12 worker nodes deployed into a China Region, you may see the following text in an error message in your `kubelet` logs:

```
Failed: Failed to pull image "xxxxxx.dkr.ecr.region-code.amazonaws.com.cn"
```

To resolve the issue, ensure that you pull the image from an Amazon Elastic Container Registry repository that is in the same region that your worker node is deployed in.

Troubleshooting IAM

This topic covers some common errors that you may see while using Amazon EKS with IAM and how to work around them.

AccessDeniedException

If you receive an `AccessDeniedException` when calling an AWS API operation, then the AWS Identity and Access Management (IAM) user or role credentials that you are using do not have the required permissions to make that call.

```
An error occurred (AccessDeniedException) when calling the DescribeCluster operation:
User: arn:aws:iam::111122223333:user/user_name is not authorized to perform:
eks:DescribeCluster on resource: arn:aws:eks:region:111122223333:cluster/cluster_name
```

In the above example message, the user does not have permissions to call the Amazon EKS `DescribeCluster` API operation. To provide Amazon EKS admin permissions to a user, see Amazon EKS identity-based policy examples (p. 242).

For more general information about IAM, see Controlling access using policies in the *IAM User Guide*.

I Am not authorized to perform iam:PassRole

If you receive an error that you're not authorized to perform the `iam:PassRole` action, then you must contact your administrator for assistance. Your administrator is the person that provided you with your user name and password. Ask that person to update your policies to allow you to pass a role to Amazon EKS.

Some AWS services allow you to pass an existing role to that service, instead of creating a new service role or service-linked role. To do this, you must have permissions to pass the role to the service.

The following example error occurs when an IAM user named `marymajor` tries to use the console to perform an action in Amazon EKS. However, the action requires the service to have permissions granted by a service role. Mary does not have permissions to pass the role to the service.

```
User: arn:aws:iam::123456789012:user/marymajor is not authorized to perform: iam:PassRole
```

In this case, Mary asks her administrator to update her policies to allow her to perform the `iam:PassRole` action.

I want to view my access keys

After you create your IAM user access keys, you can view your access key ID at any time. However, you can't view your secret access key again. If you lose your secret key, you must create a new access key pair.

Access keys consist of two parts: an access key ID (for example, `AKIAIOSFODNN7EXAMPLE`) and a secret access key (for example, `wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY`). Like a user name and password, you must use both the access key ID and secret access key together to authenticate your requests. Manage your access keys as securely as you do your user name and password.

> **Important**
> Do not provide your access keys to a third party, even to help find your canonical user ID. By doing this, you might give someone permanent access to your account.

When you create an access key pair, you are prompted to save the access key ID and secret access key in a secure location. The secret access key is available only at the time you create it. If you lose your secret access key, you must add new access keys to your IAM user. You can have a maximum of two access keys. If you already have two, you must delete one key pair before creating a new one. To view instructions, see Managing Access Keys in the *IAM User Guide*.

I'm an administrator and want to allow others to access Amazon EKS

To allow others to access Amazon EKS, you must create an IAM entity (user or role) for the person or application that needs access. They will use the credentials for that entity to access AWS. You must then attach a policy to the entity that grants them the correct permissions in Amazon EKS.

Amazon EKS User Guide
I want to allow people outside of my AWS
account to access my Amazon EKS resources

To get started right away, see Creating Your First IAM Delegated User and Group in the *IAM User Guide*.

I want to allow people outside of my AWS account to access my Amazon EKS resources

You can create a role that users in other accounts or people outside of your organization can use to access your resources. You can specify who is trusted to assume the role. For services that support resource-based policies or access control lists (ACLs), you can use those policies to grant people access to your resources.

To learn more, consult the following:

- To learn whether Amazon EKS supports these features, see How Amazon EKS works with IAM (p. 239).
- To learn how to provide access to your resources across AWS accounts that you own, see Providing Access to an IAM User in Another AWS Account That You Own in the *IAM User Guide*.
- To learn how to provide access to your resources to third-party AWS accounts, see Providing Access to AWS Accounts Owned by Third Parties in the *IAM User Guide*.
- To learn how to provide access through identity federation, see Providing Access to Externally Authenticated Users (Identity Federation) in the *IAM User Guide*.
- To learn the difference between using roles and resource-based policies for cross-account access, see How IAM Roles Differ from Resource-based Policies in the *IAM User Guide*.

Amazon EKS service quotas

The following table provides the default quotas for Amazon EKS for an AWS account that can be changed. To request a quota increase, open the AWS Support Center page, sign in if necessary, and choose **Create case**. Choose **Service limit increase**. Complete and submit the form.

Resource	Default quota
Maximum number of Amazon EKS clusters (per Region, per account)	100
Maximum number of managed node groups per cluster	30
Maximum number of nodes per managed node group	100
Maximum number Fargate profiles per cluster	10
Maximum number selectors per Fargate profile	5
Maximum number of label pairs per Fargate profile selector	5
Maximum number of concurrent Fargate pods (per Region, per account)	100
Maximum number Fargate pod launches per second (per Region, per account)	1, with temporary burst up to 10

The following table provides quotas for Amazon EKS that cannot be changed.

Resource	Default quota
Maximum number of control plane security groups per cluster (these are specified when you create the cluster)	4
Maximum number of public endpoint access CIDR ranges per cluster	40

Document history for Amazon EKS

The following table describes the major updates and new features for the Amazon EKS User Guide. We also update the documentation frequently to address the feedback that you send us.

update-history-change	update-history-description	update-history-date
Amazon EKS Region expansion (p. 295)	Amazon EKS is now available in the AWS GovCloud (US-East) (`us-gov-east-1`) and AWS GovCloud (US-West) (`us-gov-west-1`) Regions.	May 13, 2020
Kubernetes 1.12 deprecated on Amazon EKS	Kubernetes version 1.12 is no longer supported on Amazon EKS. Please update any 1.12 clusters to version 1.13 or higher in order to avoid service interruption.	May 12, 2020
Kubernetes version 1.16	Added Kubernetes version 1.16 support for new clusters and version upgrades.	April 30, 2020
Added the AWSServiceRoleForAmazonEKS service-linked role	Added the **AWSServiceRoleForAmazonEKS** service-linked role.	April 16, 2020
Kubernetes version 1.15	Added Kubernetes version 1.15 support for new clusters and version upgrades.	March 10, 2020
Amazon EKS Region expansion (p. 295)	Amazon EKS is now available in the Beijing (`cn-north-1`) and Ningxia (`cn-northwest-1`) Regions.	February 26, 2020
Amazon FSx for Lustre CSI driver	Added topic for installing the Amazon FSx for Lustre CSI Driver on Kubernetes 1.14 Amazon EKS clusters.	December 23, 2019
Restrict network access to the public access endpoint of a cluster	Amazon EKS now enables you to restrict the CIDR ranges that can communicate to the public access endpoint of the Kubernetes API server.	December 20, 2019
Resolve the private access endpoint address for a cluster from outside of a VPC	Amazon EKS now enables you to resolve the private access endpoint of the Kubernetes API server from outside of a VPC.	December 13, 2019
(Beta) Amazon EC2 A1 instance worker nodes	Launch Amazon EC2 A1 instance worker nodes that register with your Amazon EKS cluster.	December 4, 2019

Creating a cluster on AWS Outposts	Amazon EKS now supports creating clusters on an AWS Outpost.	December 3, 2019
AWS Fargate on Amazon EKS	Amazon EKS Kubernetes clusters now support running pods on Fargate.	December 3, 2019
Amazon EKS Region expansion (p. 295)	Amazon EKS is now available in the Canada (Central) (ca-central-1) Region.	November 21, 2019
Managed node groups	Amazon EKS managed node groups automate the provisioning and lifecycle management of nodes (Amazon EC2 instances) for Amazon EKS Kubernetes clusters.	November 18, 2019
Amazon EKS platform version update	New platform versions to address CVE-2019-11253.	November 6, 2019
Kubernetes 1.11 deprecated on Amazon EKS	Kubernetes version 1.11 is no longer supported on Amazon EKS. Please update any 1.11 clusters to version 1.12 or higher in order to avoid service interruption.	November 4, 2019
Amazon EKS Region expansion (p. 295)	Amazon EKS is now available in the South America (São Paulo) (sa-east-1) Region.	October 16, 2019
Windows support	Amazon EKS clusters running Kubernetes version 1.14 now support Windows workloads.	October 7, 2019
Autoscaling	Added a chapter to cover some of the different types of Kubernetes autoscaling that are supported on Amazon EKS clusters.	September 30, 2019
Kubernetes Dashboard update	Updated topic for installing the Kubernetes Dashboard on Amazon EKS clusters to use the beta 2.0 version.	September 28, 2019
Amazon EFS CSI driver	Added topic for installing the Amazon EFS CSI Driver on Kubernetes 1.14 Amazon EKS clusters.	September 19, 2019
Amazon EC2 Systems Manager parameter for Amazon EKS-optimized AMI ID	Added topic for retrieving the Amazon EKS-optimized AMI ID using an Amazon EC2 Systems Manager parameter. The parameter eliminates the need for you to look up AMI IDs.	September 18, 2019

Amazon EKS resource tagging	Manage tagging of your Amazon EKS clusters.	September 16, 2019
Amazon EBS CSI driver	Added topic for installing the Amazon EBS CSI driver on Kubernetes 1.14 Amazon EKS clusters.	September 9, 2019
New Amazon EKS-optimized AMI patched for CVE-2019-9512 and CVE-2019-9514	Amazon EKS has updated the Amazon EKS-optimized AMI to address CVE-2019-9512 and CVE-2019-9514.	September 6, 2019
Announcing deprecation of Kubernetes 1.11 in Amazon EKS	Amazon EKS will deprecate Kubernetes version 1.11 on November 4, 2019. On this day, you will no longer be able to create new 1.11 clusters and all Amazon EKS clusters running Kubernetes version 1.11 will be updated to the latest available platform version of Kubernetes version 1.12.	September 4, 2019
Kubernetes version 1.14	Added Kubernetes version 1.14 support for new clusters and version upgrades.	September 3, 2019
IAM roles for service accounts	With IAM roles for service accounts on Amazon EKS clusters, you can associate an IAM role with a Kubernetes service account. With this feature, you no longer need to provide extended permissions to the worker node IAM role so that pods on that node can call AWS APIs.	September 3, 2019
Amazon EKS Region expansion (p. 295)	Amazon EKS is now available in the Middle East (Bahrain) (`me-south-1`) Region.	August 29, 2019
Amazon EKS platform version update	New platform versions to address CVE-2019-9512 and CVE-2019-9514.	August 28, 2019
Amazon EKS platform version update	New platform versions to address CVE-2019-11247 and CVE-2019-11249.	August 5, 2019
Amazon EKS Region expansion (p. 295)	Amazon EKS is now available in the Asia Pacific (Hong Kong) (`ap-east-1`) Region.	July 31, 2019

Kubernetes 1.10 deprecated on Amazon EKS	Kubernetes version 1.10 is no longer supported on Amazon EKS. Please update any 1.10 clusters to version 1.11 or higher in order to avoid service interruption.	July 30, 2019
Added topic on ALB ingress controller	The AWS ALB Ingress Controller for Kubernetes is a controller that triggers the creation of an Application Load Balancer when ingress resources are created.	July 11, 2019
New Amazon EKS-optimized AMI	Removing unnecessary `kubectl` binary from AMIs.	July 3, 2019
Kubernetes version 1.13	Added Kubernetes version 1.13 support for new clusters and version upgrades.	June 18, 2019
New Amazon EKS-optimized AMI patched for AWS-2019-005	Amazon EKS has updated the Amazon EKS-optimized AMI to address the vulnerabilities described in AWS-2019-005.	June 17, 2019
Announcing deprecation of Kubernetes 1.10 in Amazon EKS	Amazon EKS will deprecate Kubernetes version 1.10 on July 22, 2019. On this day, you will no longer be able to create new 1.10 clusters and all Amazon EKS clusters running Kubernetes version 1.10 will be updated to the latest available platform version of Kubernetes version 1.11.	May 21, 2019
Amazon EKS platform version update	New platform version for Kubernetes 1.11 and 1.10 clusters to support custom DNS names in the Kubelet certificate and improve `etcd` performance.	May 21, 2019
Getting started with eksctl	This getting started guide helps you to install all of the required resources to get started with Amazon EKS using `eksctl`, a simple command line utility for creating and managing Kubernetes clusters on Amazon EKS.	May 10, 2019

AWS CLI get-token command (p. 295)	The **aws eks get-token** command was added to the AWS CLI so that you no longer need to install the AWS IAM Authenticator for Kubernetes to create client security tokens for cluster API server communication. Upgrade your AWS CLI installation to the latest version to take advantage of this new functionality. For more information, see Installing the AWS command line interface in the *AWS Command Line Interface User Guide*.	May 10, 2019
Amazon EKS platform version update	New platform version for Kubernetes 1.12 clusters to support custom DNS names in the Kubelet certificate and improve `etcd` performance. This fixes a bug that caused worker node Kubelet daemons to request a new certificate every few seconds.	May 8, 2019
Prometheus tutorial	Added topic for deploying Prometheus to your Amazon EKS cluster.	April 5, 2019
Amazon EKS control plane logging	Amazon EKS control plane logging makes it easy for you to secure and run your clusters by providing audit and diagnostic logs directly from the Amazon EKS control plane to CloudWatch Logs in your account.	April 4, 2019
Kubernetes version 1.12 (p. 295)	Added Kubernetes version 1.12 support for new clusters and version upgrades.	March 28, 2019
Added App Mesh getting started guide	Added documentation for getting started with App Mesh and Kubernetes.	March 27, 2019
Amazon EKS API server endpoint private access	Added documentation for disabling public access for your Amazon EKS cluster's Kubernetes API server endpoint.	March 19, 2019
Added topic for installing the Kubernetes Metrics Server	The Kubernetes Metrics Server is an aggregator of resource usage data in your cluster.	March 18, 2019

Added list of related open source projects	These open source projects extend the functionality of Kubernetes clusters running on AWS, including clusters managed by Amazon EKS.	March 15, 2019
Added topic for installing Helm locally	The `helm` package manager for Kubernetes helps you install and manage applications on your Kubernetes cluster. This topic helps you install and run the `helm` and `tiller` binaries locally so that you can install and manage charts using the `helm` CLI on your local system.	March 11, 2019
Amazon EKS platform version update	New platform version updating Amazon EKS Kubernetes 1.11 clusters to patch level 1.11.8 to address CVE-2019-1002100.	March 8, 2019
Increased cluster limit	Amazon EKS has increased the number of clusters that you can create in a Region from 3 to 50.	February 13, 2019
Amazon EKS Region expansion (p. 295)	Amazon EKS is now available in the Europe (London) (`eu-west-2`), Europe (Paris) (`eu-west-3`), and Asia Pacific (Mumbai) (`ap-south-1`) Regions.	February 13, 2019
New Amazon EKS-optimized AMI patched for ALAS-2019-1156	Amazon EKS has updated the Amazon EKS-optimized AMI to address the vulnerability described in ALAS-2019-1156.	February 11, 2019
New Amazon EKS-optimized AMI patched for ALAS2-2019-1141	Amazon EKS has updated the Amazon EKS-optimized AMI to address the CVEs referenced in ALAS2-2019-1141.	January 9, 2019
Amazon EKS Region expansion (p. 295)	Amazon EKS is now available in the Asia Pacific (Seoul) (`ap-northeast-2`) Region.	January 9, 2019
Amazon EKS region expansion (p. 295)	Amazon EKS is now available in the following additional regions: Europe (Frankfurt) (`eu-central-1`), Asia Pacific (Tokyo) (`ap-northeast-1`), Asia Pacific (Singapore) (`ap-southeast-1`), and Asia Pacific (Sydney) (`ap-southeast-2`).	December 19, 2018
Amazon EKS cluster updates	Added documentation for Amazon EKS cluster Kubernetes version updates and worker node replacement.	December 12, 2018

Amazon EKS Region expansion (p. 295)	Amazon EKS is now available in the Europe (Stockholm) (`eu-north-1`) Region.	December 11, 2018
Amazon EKS platform version update	New platform version updating Kubernetes to patch level 1.10.11 to address CVE-2018-1002105.	December 4, 2018
Added version 1.0.0 support for the Application Load Balancer ingress controller	The Application Load Balancer ingress controller releases version 1.0.0 with formal support from AWS.	November 20, 2018
Added support for CNI network configuration	The Amazon VPC CNI plugin for Kubernetes version 1.2.1 now supports custom network configuration for secondary pod network interfaces.	October 16, 2018
Added support for MutatingAdmissionWebhook and ValidatingAdmissionWebhook	Amazon EKS platform version `1.10-eks.2` now supports `MutatingAdmissionWebhook` and `ValidatingAdmissionWebhook` admission controllers.	October 10, 2018
Added partner AMI information	Canonical has partnered with Amazon EKS to create worker node AMIs that you can use in your clusters.	October 3, 2018
Added instructions for AWS CLI update-kubeconfig command	Amazon EKS has added the `update-kubeconfig` to the AWS CLI to simplify the process of creating a `kubeconfig` file for accessing your cluster.	September 21, 2018
New Amazon EKS-optimized AMIs	Amazon EKS has updated the Amazon EKS-optimized AMIs (with and without GPU support) to provide various security fixes and AMI optimizations.	September 13, 2018
Amazon EKS Region expansion (p. 295)	Amazon EKS is now available in the Europe (Ireland) (`eu-west-1`) region.	September 5, 2018
Amazon EKS platform version update	New platform version with support for Kubernetes aggregation layer and the Horizontal Pod Autoscaler(HPA).	August 31, 2018

New Amazon EKS-optimized AMIs and GPU support	Amazon EKS has updated the Amazon EKS-optimized AMI to use a new AWS CloudFormation worker node template and bootstrap script. In addition, a new Amazon EKS-optimized AMI with GPU support is available.	August 22, 2018
New Amazon EKS-optimized AMI patched for ALAS2-2018-1058	Amazon EKS has updated the Amazon EKS-optimized AMI to address the CVEs referenced in ALAS2-2018-1058.	August 14, 2018
Amazon EKS-optimized AMI build scripts	Amazon EKS has open-sourced the build scripts that are used to build the Amazon EKS-optimized AMI. These build scripts are now available on GitHub.	July 10, 2018
Amazon EKS initial release (p. 295)	Initial documentation for service launch	June 5, 2018