Amazon EKS

User Guide

Part 1

Amazon EKS: User Guide

Table of Contents

What is Amazon EKS?

Amazon Elastic Kubernetes Service (Amazon EKS) is a managed service that you can use to run Kubernetes on AWS without needing to install, operate, and maintain your own Kubernetes control plane or nodes. Kubernetes is an open-source system for automating the deployment, scaling, and management of containerized applications. Amazon EKS:

- Runs and scales the Kubernetes control plane across multiple AWS Availability Zones to ensure high availability.
- Automatically scales control plane instances based on load, detects and replaces unhealthy control plane instances, and it provides automated version updates and patching for them.
- Is integrated with many AWS services to provide scalability and security for your applications, including the following capabilities:
  - Amazon ECR for container images
  - Elastic Load Balancing for load distribution
  - IAM for authentication
  - Amazon VPC for isolation
- Runs up-to-date versions of the open-source Kubernetes software, so you can use all of the existing plugins and tooling from the Kubernetes community. Applications that are running on Amazon EKS are fully compatible with applications running on any standard Kubernetes environment, no matter whether they're 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.

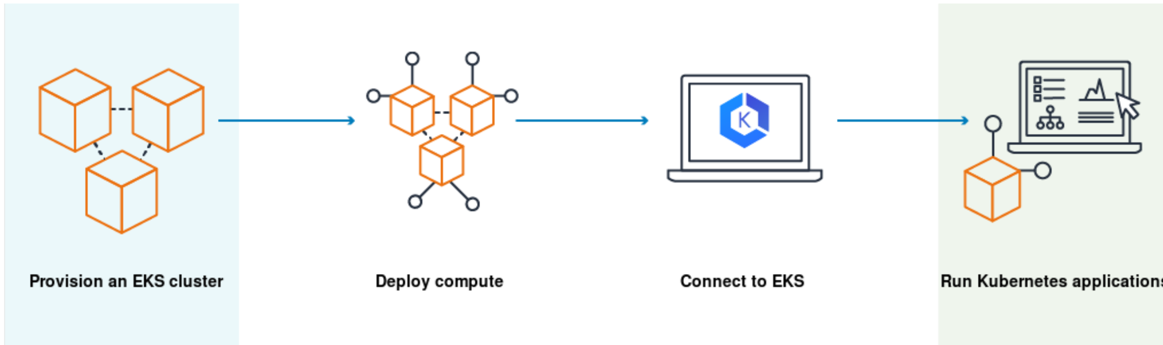
Amazon EKS control plane architecture

Amazon EKS runs a single tenant Kubernetes control plane for each cluster. The control plane infrastructure isn't shared across clusters or AWS accounts. The control plane consists of at least two API server instances and three `etcd` instances that run across three Availability Zones within an AWS Region. Amazon EKS:

- Actively monitors the load on control plane instances and automatically scales them to ensure high performance.
- Automatically detects and replaces unhealthy control plane instances, restarting them across the Availability Zones within the AWS Region as needed.
- 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 can't 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. Create an Amazon EKS cluster in the AWS Management Console or with the AWS CLI or one of the AWS SDKs.
2. Launch managed or self-managed Amazon EC2 nodes, or deploy your workloads to AWS Fargate.
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 workloads on your Amazon EKS cluster the same way that you would with any other Kubernetes environment. You can also view information about your workloads using the AWS Management Console.

To create your first cluster and its associated resources, see Getting started with Amazon EKS (p. 4). To learn about other Kubernetes deployment options, see Deployment options (p. 2).

Pricing

An Amazon EKS cluster consists of a control plane and the Amazon EC2 or AWS Fargate compute that you run pods on. For more information about pricing for the control plane, see Amazon EKS pricing. Both Amazon EC2 and Fargate provide:

- **On-Demand Instances** – Pay for the instances that you use by the second, with no long-term commitments or upfront payments. For more information, see Amazon EC2 On-Demand Pricing and AWS Fargate Pricing.
- **Savings Plans** – You can reduce your costs by making a commitment to a consistent amount of usage, in USD per hour, for a term of 1 or 3 years. For more information, see Pricing with Savings Plans.

Deployment options

You can use Amazon EKS with any, or all, of the following deployment options:

- **Amazon EKS** – Amazon Elastic Kubernetes Service (Amazon EKS) is a managed service that you can use to run Kubernetes on AWS without needing to install, operate, and maintain your own Kubernetes control plane or nodes. For more information, see What is Amazon EKS? (p. 1).
- **Amazon EKS on AWS Outposts** – Run Amazon EKS nodes on AWS Outposts. AWS Outposts enables native AWS services, infrastructure, and operating models in on-premises facilities. For more information, see Amazon EKS nodes on AWS Outposts (p. 222).

- **Amazon EKS Anywhere** – Amazon EKS Anywhere is a deployment option for Amazon EKS that enables you to easily create and operate Kubernetes clusters on-premises. Both Amazon EKS and Amazon EKS Anywhere are built on the Amazon EKS Distro. To learn more about Amazon EKS Anywhere, and its differences with Amazon EKS, see Overview and Comparing Amazon EKS Anywhere to Amazon EKS in the Amazon EKS Anywhere documentation.
- **Amazon EKS Distro** – Amazon EKS Distro is a distribution of the same open-source Kubernetes software and dependencies deployed by Amazon EKS in the cloud. Amazon EKS Distro follows the same Kubernetes version release cycle as Amazon EKS and is provided as an open-source project. To learn more, see Amazon EKS Distro. You can also view and download the source code for the Amazon EKS Distro on GitHub.

When choosing which deployment options to use for your Kubernetes cluster, consider the following:

Feature	Amazon EKS	Amazon EKS on AWS Outposts	Amazon EKS Anywhere	Amazon EKS Distro
Hardware	AWS-supplied	AWS-supplied	Supplied by you	Supplied by you
Deployment location	AWS cloud	Your data center	Your data center	Your datacenter
Kubernetes control plane location	AWS cloud	AWS cloud	Your data center	Your datacenter
Kubernetes data plane location	AWS cloud	Your data center	Your data center	Your datacenter
Support	AWS support	AWS support	AWS support	OSS community support

Frequently asked questions

- Q: Can I deploy Amazon EKS Anywhere in the AWS cloud?

  A: Amazon EKS Anywhere isn't designed to run in the AWS cloud. It doesn't integrate with the Kubernetes Cluster API Provider for AWS. If you plan to deploy Kubernetes clusters in the AWS cloud, we strongly recommend that you use Amazon EKS.
- Q: Can I deploy Amazon EKS Anywhere on AWS Outposts?

  A: Amazon EKS Anywhere isn't designed to run on AWS Outposts. If you're planning to deploy Kubernetes clusters on AWS Outposts, we strongly recommend that you use Amazon EKS on AWS Outposts.

Getting started with Amazon EKS

Many procedures of this user guide use the following command line tools:

- **`kubectl`** – A command line tool for working with Kubernetes clusters. For more information, see Installing `kubectl` (p. 4).
- **`eksctl`** – A command line tool for working with EKS clusters that automates many individual tasks. For more information, see Installing `eksctl` (p. 10).
- **AWS CLI** – A command line tool for working with AWS services, including Amazon EKS. For more information, see Installing, updating, and uninstalling the AWS CLI in the AWS Command Line Interface User Guide. After installing the AWS CLI, we recommend that you also configure it. For more information, see Quick configuration with `aws configure` in the AWS Command Line Interface User Guide.

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

- Getting started with Amazon EKS – `eksctl` (p. 12) – 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 the tutorial, you will have a running Amazon EKS cluster that you can deploy applications to. This is the fastest and simplest way to get started with Amazon EKS.
- Getting started with Amazon EKS – AWS Management Console and AWS CLI (p. 15) – This getting started guide helps you to create all of the required resources to get started with Amazon EKS using the AWS Management Console and AWS CLI. At the end of the tutorial, you will have a running Amazon EKS cluster that you can deploy applications to. In this guide, you manually create each resource required for an Amazon EKS cluster. The procedures give you visibility into how each resource is created and how they interact with each other.

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. Select the tab name of your operating system. 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 `1.21` `kubectl` client works with Kubernetes `1.20`, `1.21`, and `1.22` clusters.

Select the tab with the name of the operating system that you want to install `kubectl` on.

macOS

To install `kubectl` on macOS

1. Download the Amazon EKS vended `kubectl` binary for your cluster's Kubernetes version from Amazon S3.

- **Kubernetes** `1.22`

```
curl -o kubectl https://s3.us-west-2.amazonaws.com/amazon-eks/1.22.6/2022-03-09/
bin/darwin/amd64/kubectl
```

- **Kubernetes** `1.21`

```
curl -o kubectl https://s3.us-west-2.amazonaws.com/amazon-eks/1.21.2/2021-07-05/
bin/darwin/amd64/kubectl
```

- **Kubernetes** `1.20`

```
curl -o kubectl https://s3.us-west-2.amazonaws.com/amazon-eks/1.20.4/2021-04-12/
bin/darwin/amd64/kubectl
```

- **Kubernetes** `1.19`

```
curl -o kubectl https://s3.us-west-2.amazonaws.com/amazon-eks/1.19.6/2021-01-05/
bin/darwin/amd64/kubectl
```

- **Kubernetes** `1.18`

```
curl -o kubectl https://s3.us-west-2.amazonaws.com/amazon-eks/1.18.9/2020-11-02/
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** `1.22`

      ```
      curl -o kubectl.sha256 https://s3.us-west-2.amazonaws.com/amazon-
      eks/1.22.6/2022-03-09/bin/darwin/amd64/kubectl.sha256
      ```

      - **Kubernetes** `1.21`

      ```
      curl -o kubectl.sha256 https://s3.us-west-2.amazonaws.com/amazon-
      eks/1.21.2/2021-07-05/bin/darwin/amd64/kubectl.sha256
      ```

      - **Kubernetes** `1.20`

      ```
      curl -o kubectl.sha256 https://s3.us-west-2.amazonaws.com/amazon-
      eks/1.20.4/2021-04-12/bin/darwin/amd64/kubectl.sha256
      ```

      - **Kubernetes** `1.19`

      ```
      curl -o kubectl.sha256 https://s3.us-west-2.amazonaws.com/amazon-
      eks/1.19.6/2021-01-05/bin/darwin/amd64/kubectl.sha256
      ```

      - **Kubernetes** `1.18`

      ```
      curl -o kubectl.sha256 https://s3.us-west-2.amazonaws.com/amazon-
      eks/1.18.9/2020-11-02/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 $HOME/bin/kubectl && export PATH=$HOME/bin:$PATH
```

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 using the command for your hardware platform.

- **Kubernetes `1.22`**

```
curl -o kubectl https://s3.us-west-2.amazonaws.com/amazon-eks/1.22.6/2022-03-09/bin/linux/amd64/kubectl
```

```
curl -o kubectl https://s3.us-west-2.amazonaws.com/amazon-eks/1.22.6/2022-03-09/bin/linux/arm64/kubectl
```

- **Kubernetes `1.21`**

```
curl -o kubectl https://s3.us-west-2.amazonaws.com/amazon-eks/1.21.2/2021-07-05/bin/linux/amd64/kubectl
```

```
curl -o kubectl https://s3.us-west-2.amazonaws.com/amazon-eks/1.21.2/2021-07-05/bin/linux/arm64/kubectl
```

- **Kubernetes `1.20`**

```
curl -o kubectl https://s3.us-west-2.amazonaws.com/amazon-eks/1.20.4/2021-04-12/bin/linux/amd64/kubectl
```

```
curl -o kubectl https://s3.us-west-2.amazonaws.com/amazon-eks/1.20.4/2021-04-12/bin/linux/arm64/kubectl
```

- **Kubernetes `1.19`**

```
curl -o kubectl https://s3.us-west-2.amazonaws.com/amazon-eks/1.19.6/2021-01-05/
bin/linux/amd64/kubectl
```

```
curl -o kubectl https://s3.us-west-2.amazonaws.com/amazon-eks/1.19.6/2021-01-05/
bin/linux/arm64/kubectl
```

- **Kubernetes `1.18`**

```
curl -o kubectl https://s3.us-west-2.amazonaws.com/amazon-eks/1.18.9/2020-11-02/
bin/linux/amd64/kubectl
```

```
curl -o kubectl https://s3.us-west-2.amazonaws.com/amazon-eks/1.18.9/2020-11-02/
bin/linux/arm64/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 Linuxusing the command for your hardware platform.

   - **Kubernetes `1.22`**

```
curl -o kubectl.sha256 https://s3.us-west-2.amazonaws.com/amazon-
eks/1.22.6/2022-03-09/bin/linux/amd64/kubectl.sha256
```

```
curl -o kubectl.sha256 https://s3.us-west-2.amazonaws.com/amazon-
eks/1.23.6/2022-03-09/bin/linux/arm64/kubectl.sha256
```

   - **Kubernetes `1.21`**

```
curl -o kubectl.sha256 https://s3.us-west-2.amazonaws.com/amazon-
eks/1.21.2/2021-07-05/bin/linux/amd64/kubectl.sha256
```

```
curl -o kubectl.sha256 https://s3.us-west-2.amazonaws.com/amazon-
eks/1.21.2/2021-07-05/bin/linux/arm64/kubectl.sha256
```

   - **Kubernetes `1.20`**

```
curl -o kubectl.sha256 https://s3.us-west-2.amazonaws.com/amazon-
eks/1.20.4/2021-04-12/bin/linux/amd64/kubectl.sha256
```

```
curl -o kubectl.sha256 https://s3.us-west-2.amazonaws.com/amazon-
eks/1.20.4/2021-04-12/bin/linux/arm64/kubectl.sha256
```

   - **Kubernetes `1.19`**

```
curl -o kubectl.sha256 https://s3.us-west-2.amazonaws.com/amazon-
eks/1.19.6/2021-01-05/bin/linux/amd64/kubectl.sha256
```

```
curl -o kubectl.sha256 https://s3.us-west-2.amazonaws.com/amazon-
eks/1.19.6/2021-01-05/bin/linux/arm64/kubectl.sha256
```

   - **Kubernetes `1.18`**

```
curl -o kubectl.sha256 https://s3.us-west-2.amazonaws.com/amazon-eks/1.18.9/2020-11-02/bin/linux/amd64/kubectl.sha256
```

```
curl -o kubectl.sha256 https://s3.us-west-2.amazonaws.com/amazon-eks/1.18.9/2020-11-02/bin/linux/arm64/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 `1.22`**

```
curl -o kubectl.exe https://s3.us-west-2.amazonaws.com/amazon-eks/1.22.6/2022-03-09/bin/windows/amd64/kubectl.exe
```

- **Kubernetes `1.21`**

```
curl -o kubectl.exe https://s3.us-west-2.amazonaws.com/amazon-eks/1.21.2/2021-07-05/bin/windows/amd64/kubectl.exe
```

- **Kubernetes `1.20`**

```
curl -o kubectl.exe https://s3.us-west-2.amazonaws.com/amazon-eks/1.20.4/2021-04-12/bin/windows/amd64/kubectl.exe
```

- **Kubernetes 1.19**

```
curl -o kubectl.exe https://s3.us-west-2.amazonaws.com/amazon-eks/1.19.6/2021-01-05/bin/windows/amd64/kubectl.exe
```

- **Kubernetes 1.18**

```
curl -o kubectl.exe https://s3.us-west-2.amazonaws.com/amazon-eks/1.18.9/2020-11-02/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 1.22**

      ```
      curl -o kubectl.exe.sha256 https://s3.us-west-2.amazonaws.com/amazon-eks/1.22.6/2022-03-09/bin/windows/amd64/kubectl.exe.sha256
      ```

      - **Kubernetes 1.21**

      ```
      curl -o kubectl.exe.sha256 https://s3.us-west-2.amazonaws.com/amazon-eks/1.21.2/2021-07-05/bin/windows/amd64/kubectl.exe.sha256
      ```

      - **Kubernetes 1.20**

      ```
      curl -o kubectl.exe.sha256 https://s3.us-west-2.amazonaws.com/amazon-eks/1.20.4/2021-04-12/bin/windows/amd64/kubectl.exe.sha256
      ```

      - **Kubernetes 1.19**

      ```
      curl -o kubectl.exe.sha256 https://s3.us-west-2.amazonaws.com/amazon-eks/1.19.6/2021-01-05/bin/windows/amd64/kubectl.exe.sha256
      ```

      - **Kubernetes 1.18**

      ```
      curl -o kubectl.exe.sha256 https://s3.us-west-2.amazonaws.com/amazon-eks/1.18.9/2020-11-02/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 `eksctl`

This topic 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 nodes for Amazon EKS. For more information and to see the official documentation, visit https://eksctl.io/.

This topic helps you to download and install `eksctl` binaries for macOS, Linux, and Windows operating systems.

Prerequisite

The `kubectl` command line tool is installed on your computer or AWS CloudShell. The version can be the same as or up to one minor version earlier or later than the Kubernetes version of your cluster. For example, if your cluster version is `1.21`, you can use `kubectl` version `1.20`,`1.21`, or `1.22` with it. To install or upgrade `kubectl`, see Installing `kubectl` (p. 4).

Installing or upgrading `eksctl`

This section helps you install or upgrade to the latest version of the `eksctl` command line utility. Complete the procedure for your operating system.

macOS

To install or upgrade `eksctl` on macOS

The easiest way to get started with Amazon EKS and macOS is by installing `eksctl` with Homebrew, an open-source tool that can be installed using these instructions. The `eksctl` Homebrew recipe installs `eksctl` and any other dependencies that are required for Amazon EKS, such as `kubectl`. The recipe also installs the `aws-iam-authenticator` (p. 419), which is required if you don't have the AWS CLI version `1.16.156` or higher installed.

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 `0.104.0`. 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/v0.104.0/eksctl_Darwin_amd64.tar.gz, extract `eksctl`, and then run it.

Linux

To install or upgrade `eksctl` on Linux

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 `0.104.0`. 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/v0.104.0/eksctl_Linux_amd64.tar.gz` and complete steps 1-3 again.

Windows

To install or upgrade `eksctl` on Windows

1. If you do not already have Chocolatey installed on your Windows system, see Installing Chocolatey.
2. Install or upgrade `eksctl`.
   - Install the binaries with the following command:

```
choco install -y eksctl
```

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

```
choco upgrade -y eksctl
```

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

```
eksctl version
```

Note

The `GitTag` version should be at least `0.104.0`. 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/v0.104.0/eksctl_Windows_amd64.zip, extract `eksctl`, and then run it.

Getting started with Amazon EKS – `eksctl`

This guide helps you to create all of the required resources to get started with Amazon Elastic Kubernetes Service (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 that you can deploy applications to.

The procedures in this guide create several resources for you automatically that you have to create manually when you create your cluster using the AWS Management Console. If you'd rather manually create most of the resources to better understand how they interact with each other, then use the AWS Management Console to create your cluster and compute. For more information, see Getting started with Amazon EKS – AWS Management Console and AWS CLI (p. 15).

Prerequisites

Before starting this tutorial, you must install and configure the following tools and resources that you need to create and manage an Amazon EKS cluster.

- **`kubectl`** – A command line tool for working with Kubernetes clusters. This guide requires that you use version `1.22` or later. For more information, see Installing `kubectl` (p. 4).
- **`eksctl`** – A command line tool for working with EKS clusters that automates many individual tasks. This guide requires that you use version `0.104.0` or later. For more information, see Installing `eksctl` (p. 10).
- **Required IAM permissions** – The IAM security principal that you're using must have permissions to work with Amazon EKS IAM roles and service linked roles, AWS CloudFormation, and a VPC and related resources. For more information, see Actions, resources, and condition keys for Amazon Elastic Container Service for Kubernetes and Using service-linked roles in the IAM User Guide. You must complete all steps in this guide as the same user.

Step 1: Create your Amazon EKS cluster and nodes

Important

To get started as simply and quickly as possible, this topic includes steps to create a cluster and nodes with default settings. Before creating a cluster and nodes for production use, we recommend that you familiarize yourself with all settings and deploy a cluster and nodes with the settings that meet your requirements. For more information, see Creating an Amazon EKS cluster (p. 23) and Amazon EKS nodes (p. 101). Some settings can only be enabled when creating your cluster and nodes.

You can create a cluster with one of the following node types. To learn more about each type, see Amazon EKS nodes (p. 101). After your cluster is deployed, you can add other node types.

- **Fargate – Linux** – Select this type of node if you want to run Linux applications on AWS Fargate. Fargate is a serverless compute engine that lets you deploy Kubernetes pods without managing Amazon EC2 instances.
- **Managed nodes – Linux** – Select this type of node if you want to run Amazon Linux applications on Amazon EC2 instances. Though not covered in this guide, you can also add Windows self-managed (p. 136) and Bottlerocket (p. 134) nodes to your cluster.

Create your Amazon EKS cluster with the following command. You can replace *my-cluster* with your own value. The cluster name can contain only alphanumeric characters (case-sensitive) and hyphens. It must start with an alphabetic character and can't be longer than 128 characters. Replace *region-code* with any AWS Region that is supported by Amazon EKS. For a list of AWS Regions, see Amazon EKS endpoints and quotas in the AWS General Reference guide.

Fargate – Linux

```
eksctl create cluster --name my-cluster --region region-code --fargate
```

Managed nodes – Linux

```
eksctl create cluster --name my-cluster --region region-code
```

Cluster creation takes several minutes. During 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
```

`eksctl` created a `kubectl` `config` file in `~/.kube` or added the new cluster's configuration within an existing `config` file in `~/.kube` on your computer.

After cluster creation is complete, view the AWS CloudFormation stack named `eksctl-my-cluster-cluster` in the AWS CloudFormation console at https://console.aws.amazon.com/cloudformation to see all of the resources that were created.

Step 2: View Kubernetes resources

1. View your cluster nodes.

```
kubectl get nodes -o wide
```

The example output is as follows.

Fargate – Linux

```
NAME                                                  STATUS   ROLES    AGE
   VERSION              INTERNAL-IP        EXTERNAL-IP   OS-IMAGE         KERNEL-
VERSION                  CONTAINER-RUNTIME
fargate-ip-192-168-141-147.region-code.compute.internal Ready    <none>
  8m3s     v1.22.9-eks-7c9bda    192.168.141.147    <none>        Amazon Linux 2
  5.4.156-83.273.amzn2.x86_64    containerd://1.3.2
fargate-ip-192-168-164-53.region-code.compute.internal  Ready    <none>
  7m30s    v1.22.9-eks-7c9bda    192.168.164.53     <none>        Amazon Linux 2
  5.4.156-83.273.amzn2.x86_64    containerd://1.3.2
```

Managed nodes – Linux

```
NAME                                         STATUS   ROLES    AGE    VERSION
           INTERNAL-IP       EXTERNAL-IP     OS-IMAGE         KERNEL-VERSION
          CONTAINER-RUNTIME
ip-192-168-12-49.region-code.compute.internal    Ready    <none>   6m7s
  v1.22.9-eks-d1db3c    192.168.12.49     52.35.116.65    Amazon Linux 2
  5.4.156-83.273.amzn2.x86_64    docker://20.10.7
```

```
ip-192-168-72-129.region-code.compute.internal  Ready    <none>   6m4s
  v1.22.9-eks-d1db3c    192.168.72.129    44.242.140.21    Amazon Linux 2
  5.4.156-83.273.amzn2.x86_64    docker://20.10.7
```

For more information about what you see in the output, see View Kubernetes resources (p. 508).

2. View the workloads running on your cluster.

```
kubectl get pods -A -o wide
```

The example output is as follows.

Fargate – Linux

```
NAMESPACE     NAME                        READY   STATUS    RESTARTS   AGE   IP
           NODE                                                      NOMINATED NODE
   READINESS GATES
kube-system   coredns-69dfb8f894-9z95l    1/1     Running   0          18m
  192.168.164.53     fargate-ip-192-168-164-53.region-code.compute.internal
 <none>          <none>
kube-system   coredns-69dfb8f894-c8v66    1/1     Running   0          18m
  192.168.141.147    fargate-ip-192-168-141-147.region-code.compute.internal
 <none>          <none>
```

Managed nodes – Linux

```
NAMESPACE     NAME                        READY   STATUS    RESTARTS   AGE     IP
               NODE                                              NOMINATED NODE
 READINESS GATES
kube-system   aws-node-6ctpm              1/1     Running   0          7m43s
  192.168.72.129    ip-192-168-72-129.region-code.compute.internal    <none>
  <none>
kube-system   aws-node-cbntg              1/1     Running   0          7m46s
  192.168.12.49     ip-192-168-12-49.region-code.compute.internal     <none>
  <none>
kube-system   coredns-559b5db75d-26t47    1/1     Running   0          14m
  192.168.78.81     ip-192-168-72-129.region-code.compute.internal    <none>
  <none>
kube-system   coredns-559b5db75d-9rvnk    1/1     Running   0          14m
  192.168.29.248    ip-192-168-12-49.region-code.compute.internal     <none>
  <none>
kube-system   kube-proxy-l8pbd            1/1     Running   0          7m46s
  192.168.12.49     ip-192-168-12-49.region-code.compute.internal     <none>
  <none>
kube-system   kube-proxy-zh85h            1/1     Running   0          7m43s
  192.168.72.129    ip-192-168-72-129.region-code.compute.internal    <none>
  <none>
```

For more information about what you see in the output, see View Kubernetes resources (p. 508).

Step 3: Delete your cluster and nodes

After you've finished with the cluster and nodes that you created for this tutorial, you should clean up by deleting the cluster and nodes with the following command. If you want to do more with this cluster before you clean up, see Next steps (p. 15).

```
eksctl delete cluster --name my-cluster --region region-code
```

Next steps

The following documentation topics help you to extend the functionality of your cluster.

- Deploy a sample application (p. 360) to your cluster.
- The IAM entity (user or role) that created the cluster is the only IAM entity that can make calls to the Kubernetes API server with `kubectl` or the AWS Management Console. If you want other IAM users or roles to have access to your cluster, then you need to add them. For more information, see Enabling IAM user and role access to your cluster (p. 404) and Required permissions (p. 508).
- Before deploying a cluster for production use, we recommend familiarizing yourself with all of the settings for clusters (p. 23) and nodes (p. 101). Some settings (such as enabling SSH access to Amazon EC2 nodes) must be made when the cluster is created.
- To increase security for your cluster, configure the Amazon VPC Container Networking Interface plugin to use IAM roles for service accounts (p. 280).

Getting started with Amazon EKS – AWS Management Console and AWS CLI

This guide helps you to create all of the required resources to get started with Amazon Elastic Kubernetes Service (Amazon EKS) using the AWS Management Console and the AWS CLI. In this guide, you manually create each resource. At the end of this tutorial, you will have a running Amazon EKS cluster that you can deploy applications to.

The procedures in this guide give you complete visibility into how each resource is created and how the resources interact with each other. If you'd rather have most of the resources created for you automatically, use the `eksctl` CLI to create your cluster and nodes. For more information, see Getting started with Amazon EKS – `eksctl` (p. 12).

Prerequisites

Before starting this tutorial, you must install and configure the following tools and resources that you need to create and manage an Amazon EKS cluster.

- **AWS CLI** – A command line tool for working with AWS services, including Amazon EKS. This guide requires that you use version `2.6.3` or later or `1.23.11` or later. For more information, see Installing, updating, and uninstalling the AWS CLI in the AWS Command Line Interface User Guide. After installing the AWS CLI, we recommend that you also configure it. For more information, see Quick configuration with `aws configure` in the AWS Command Line Interface User Guide.
- **`kubectl`** – A command line tool for working with Kubernetes clusters. This guide requires that you use version `1.22` or later. For more information, see Installing `kubectl` (p. 4).
- **Required IAM permissions** – The IAM security principal that you're using must have permissions to work with Amazon EKS IAM roles and service linked roles, AWS CloudFormation, and a VPC and related resources. For more information, see Actions, resources, and condition keys for Amazon Elastic Kubernetes Service and Using service-linked roles in the IAM User Guide. You must complete all steps in this guide as the same user.

Step 1: Create your Amazon EKS cluster

Important
To get started as simply and quickly as possible, this topic includes steps to create a cluster with default settings. Before creating a cluster for production use, we recommend that you familiarize yourself with all settings and deploy a cluster with the settings that meet your requirements. For more information, see Creating an Amazon EKS cluster (p. 23). Some settings can only be enabled when creating your cluster.

To create your cluster

1. Create an Amazon VPC with public and private subnets that meets Amazon EKS requirements. Replace *region-code* with any AWS Region that is supported by Amazon EKS. For a list of AWS Regions, see Amazon EKS endpoints and quotas in the AWS General Reference guide. You can replace *my-eks-vpc-stack* with any name you choose.

```
aws cloudformation create-stack \
  --region region-code \
  --stack-name my-eks-vpc-stack \
  --template-url https://s3.us-west-2.amazonaws.com/amazon-eks/
cloudformation/2020-10-29/amazon-eks-vpc-private-subnets.yaml
```

   Tip
   For a list of all the resources the previous command creates, open the AWS CloudFormation console at https://console.aws.amazon.com/cloudformation. Choose the *my-eks-vpc-stack* stack and then choose the **Resources** tab.

2. Create a cluster IAM role and attach the required Amazon EKS IAM managed policy to it. 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.

   a. Copy the following contents to a file named *cluster-role-trust-policy.json*.

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

   b. Create the role.

```
aws iam create-role \
  --role-name myAmazonEKSClusterRole \
  --assume-role-policy-document file://"cluster-role-trust-policy.json"
```

   c. Attach the required Amazon EKS managed IAM policy to the role.

```
aws iam attach-role-policy \
  --policy-arn arn:aws:iam::aws:policy/AmazonEKSClusterPolicy \
  --role-name myAmazonEKSClusterRole
```

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

Make sure that the AWS Region shown in the upper right of your console is the AWS Region that you want to create your cluster in. If it's not, choose the dropdown next to the AWS Region name and choose the AWS Region that you want to use.

4. Choose **Add cluster**, and then choose **Create**. If you don't see this option, then choose **Clusters** in the left navigation pane first.
5. On the **Configure cluster** page, do the following:

   a. Enter a **Name** for your cluster, such as **`my-cluster`**.
   b. For **Cluster Service Role**, choose *`myAmazonEKSClusterRole`*.
   c. Leave the remaining settings at their default values and choose **Next**.
6. On the **Specify networking** page, do the following:

   a. Choose the ID of the VPC that you created in a previous step from the **VPC** dropdown list. It is something like *`vpc-00x0000x000x0x000`* | *`my-eks-vpc-stack-VPC`*.
   b. Leave the remaining settings at their default values and choose **Next**.
7. On the **Configure logging** page, choose **Next**.
8. On the **Review and create** page, choose **Create**.

   To the right of the cluster's name, the cluster status is **Creating** for several minutes until the cluster provisioning process completes. Don't continue to the next step until the status is **Active**.

   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. 529).

Step 2: Configure your computer to communicate with your cluster

In this section, you create a `kubeconfig` file for your cluster. The settings in this file enable the `kubectl` CLI to communicate with your cluster.

To configure your computer to communicate with your cluster

1. Create or update a `kubeconfig` file for your cluster. Replace *`region-code`* with the AWS Region that you created your cluster in. Replace *`my-cluster`* with the name of your cluster.

```
aws eks update-kubeconfig --region region-code --name my-cluster
```

   By default, the `config` file is created in `~/.kube` or the new cluster's configuration is added to an existing `config` file in `~/.kube`.
2. Test your configuration.

```
kubectl get svc
```

   Note
   If you receive any authorization or resource type errors, see Unauthorized or access denied (`kubectl`) (p. 530) in the troubleshooting section.

   The example output is as follows.

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

Step 3: Create nodes

Important

To get started as simply and quickly as possible, this topic includes steps to create nodes with default settings. Before creating nodes for production use, we recommend that you familiarize yourself with all settings and deploy nodes with the settings that meet your requirements. For more information, see Amazon EKS nodes (p. 101). Some settings can only be enabled when creating your nodes.

You can create a cluster with one ofthe following node types. To learn more about each type, see Amazon EKS nodes (p. 101). After your cluster is deployed, you can add other node types.

- **Fargate – Linux** – Choose this type of node if you want to run Linux applications on AWS Fargate. Fargate is a serverless compute engine that lets you deploy Kubernetes pods without managing Amazon EC2 instances.
- **Managed nodes – Linux** – Choose this type of node if you want to run Amazon Linux applications on Amazon EC2 instances. Though not covered in this guide, you can also add Windows self-managed (p. 136) and Bottlerocket (p. 134) nodes to your cluster.

Fargate – Linux

Create a Fargate profile. When Kubernetes pods are deployed with criteria that matches the criteria defined in the profile, the pods are deployed to Fargate.

To create a Fargate profile

1. Create an IAM role and attach the required Amazon EKS IAM managed policy to it. When your cluster creates pods on Fargate infrastructure, the components running on the Fargate infrastructure must make calls to AWS APIs on your behalf. This is so that they can do actions such as pull container images from Amazon ECR or route logs to other AWS services. The Amazon EKS pod execution role provides the IAM permissions to do this.

   a. Copy the following contents to a file named *pod-execution-role-trust-policy*.json. Replace *region-code* with the AWS Region that your cluster is in. If you want to use the same role in all AWS Regions in your account, replace *region-code* with *. Replace *111122223333* with your account ID and *my-cluster* with the name of your cluster. If you want to use the same role for all clusters in your account, replace *my-cluster* with *.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Condition": {
         "ArnLike": {
            "aws:SourceArn": "arn:aws:eks:region-
code:111122223333:fargateprofile/my-cluster/*"
         }
      },
      "Principal": {
        "Service": "eks-fargate-pods.amazonaws.com"
      },
```

```
      "Action": "sts:AssumeRole"
    }
  ]
}
```

b. Create a pod execution IAM role.

```
aws iam create-role \
  --role-name AmazonEKSFargatePodExecutionRole \
  --assume-role-policy-document file://"pod-execution-role-trust-policy.json"
```

c. Attach the required Amazon EKS managed IAM policy to the role.

```
aws iam attach-role-policy \
  --policy-arn arn:aws:iam::aws:policy/AmazonEKSFargatePodExecutionRolePolicy \
  --role-name AmazonEKSFargatePodExecutionRole
```

2. Open the Amazon EKS console at https://console.aws.amazon.com/eks/home#/clusters.
3. On the **Clusters** page, choose the *my-cluster* cluster.
4. On the ***my-cluster*** page, do the following:
   a. Choose the **Compute** tab.
   b. Under **Fargate Profiles**, choose **Add Fargate Profile**.
5. On the **Configure Fargate Profile** page, do the following:
   a. For **Name**, enter a unique name for your Fargate profile, such as ***my-profile***.
   b. For **Pod execution role**, choose the **AmazonEKSFargatePodExecutionRole** that you created in a previous step.
   c. Choose the **Subnets** dropdown and deselect any subnet with `Public` in its name. Only private subnets are supported for pods that are running on Fargate.
   d. Choose **Next**.
6. On the **Configure pod selection** page, do the following:
   a. For **Namespace**, enter **`default`**.
   b. Choose **Next**.
7. On the **Review and create** page, review the information for your Fargate profile and choose **Create**.
8. After a few minutes, the **Status** in the **Fargate Profile configuration** section will change from **Creating** to **Active**. Don't continue to the next step until the status is **Active**.
9. If you plan to deploy all pods to Fargate (none to Amazon EC2 nodes), do the following to create another Fargate profile and run the default name resolver (CoreDNS) on Fargate.

   Note
   If you don't do this, you won't have any nodes at this time.

   a. On the **Fargate Profile** page, choose *my-profile*.
   b. Under **Fargate profiles**, choose **Add Fargate Profile**.
   c. For **Name**, enter ***CoreDNS***.
   d. For **Pod execution role**, choose the **AmazonEKSFargatePodExecutionRole** that you created in a previous step.
   e. Choose the **Subnets** dropdown and deselect any subnet with `Public` in its name. Only private subnets are supported for pods running on Fargate.
   f. Choose **Next**.
   g. For **Namespace**, enter **`kube-system`**.

h. Choose **Match labels**, and then choose **Add label**.

i. Enter **`k8s-app`** for **Key** and **`kube-dns`** for value. This is necessary for the default name resolver (CoreDNS) to deploy to Fargate.

j. Choose **Next**.

k. On the **Review and create** page, review the information for your Fargate profile and choose **Create**.

l. Run the following command to remove the default `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"}]'
```

Note

The system creates and deploys two nodes based on the Fargate profile label you added. You won't see anything listed in **Node Groups** because they aren't applicable for Fargate nodes, but you will see the new nodes listed in the **Overview** tab.

Managed nodes – Linux

Create a managed node group, specifying the subnets and node IAM role that you created in previous steps.

To create your Amazon EC2 Linux managed node group

1. Create a node IAM role and attach the required Amazon EKS IAM managed policy to it. The Amazon EKS node `kubelet` daemon makes calls to AWS APIs on your behalf. Nodes receive permissions for these API calls through an IAM instance profile and associated policies.

   a. Copy the following contents to a file named *`node-role-trust-policy.json`*.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "ec2.amazonaws.com"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
```

   b. Create the node IAM role.

```
aws iam create-role \
  --role-name myAmazonEKSNodeRole \
  --assume-role-policy-document file://"node-role-trust-policy.json"
```

   c. Attach the required managed IAM policies to the role.

```
aws iam attach-role-policy \
  --policy-arn arn:aws:iam::aws:policy/AmazonEKSWorkerNodePolicy \
  --role-name myAmazonEKSNodeRole
```

```
aws iam attach-role-policy \
  --policy-arn arn:aws:iam::aws:policy/AmazonEC2ContainerRegistryReadOnly \
  --role-name myAmazonEKSNodeRole
aws iam attach-role-policy \
  --policy-arn arn:aws:iam::aws:policy/AmazonEKS_CNI_Policy \
  --role-name myAmazonEKSNodeRole
```

2. Open the Amazon EKS console at https://console.aws.amazon.com/eks/home#/clusters.
3. Choose the name of the cluster that you created in Step 1: Create your Amazon EKS cluster (p. 16), such as *my-cluster*.
4. On the ***my-cluster*** page, do the following:

   a. Choose the **Compute** tab.
   b. Choose **Add Node Group**.
5. On the **Configure Node Group** page, do the following:

   a. For **Name**, enter a unique name for your managed node group, such as ***my-nodegroup***.
   b. For **Node IAM role name**, choose *myAmazonEKSNodeRole* role that you created in a previous step. We recommend that each node group use its own unique IAM role.
   c. Choose **Next**.
6. On the **Set compute and scaling configuration** page, accept the default values and choose **Next**.
7. On the **Specify networking** page, accept the default values and choose **Next**.
8. On the **Review and create** page, review your managed node group configuration and choose **Create**.
9. After several minutes, the **Status** in the **Node Group configuration** section will change from **Creating** to **Active**. Don't continue to the next step until the status is **Active**.

Step 4: View resources

You can view your nodes and Kubernetes workloads.

To view your nodes and workloads

1. In the left navigation pane, choose **Clusters**. In the list of **Clusters**, choose the name of the cluster that you created, such as *my-cluster*.
2. On the ***my-cluster*** page, choose the following:

   a. **Compute** tab – You see the list of **Nodes** that were deployed for the cluster. You can choose the name of a node to see more information about it.
   b. **Resources tab** – You see all of the Kubernetes resources that are deployed by default to an Amazon EKS cluster. Select any resource type in the console to learn more about it.

Step 5: Delete resources

After you've finished with the cluster and nodes that you created for this tutorial, you should delete the resources that you created. If you want to do more with this cluster before you delete the resources, see Next steps (p. 22).

To delete the resources that you created in this guide

1. Delete any node groups or Fargate profiles that you created.

a. Open the Amazon EKS console at https://console.aws.amazon.com/eks/home#/clusters.
b. In the left navigation pane, choose **Clusters**. In the list of clusters, choose `my-cluster`.
c. Choose the **Compute** tab.
d. If you created a node group, choose the `my-nodegroup` node group and then choose **Delete**. Enter **`my-nodegroup`**, and then choose **Delete**.
e. For each Fargate profile that you created, choose it and then choose **Delete**. Enter the name of the profile, and then choose **Delete**.

   Note
   When deleting a second Fargate profile, you may need to wait for the first one to finish deleting.
f. Don't continue until the node group or Fargate profiles are deleted.

2. Delete the cluster.

   a. In the left navigation pane, choose **Clusters**. In the list of clusters, choose `my-cluster`.
   b. Choose **Delete cluster**.
   c. Enter **`my-cluster`** and then choose **Delete**. Don't continue until the cluster is deleted.
3. Delete the VPC AWS CloudFormation stack that you created.

   a. Open the AWS CloudFormation console at https://console.aws.amazon.com/cloudformation.
   b. Choose the `my-eks-vpc-stack` stack, and then choose **Delete**.
   c. In the **Delete `my-eks-vpc-stack`** confirmation dialog box, choose **Delete stack**.
4. Delete the IAM roles that you created.

   a. Open the IAM console at https://console.aws.amazon.com/iam/.
   b. In the left navigation pane, choose **Roles**.
   c. Select each role you created from the list (**`myAmazonEKSClusterRole`**, as well as **AmazonEKSFargatePodExecutionRole** or `myAmazonEKSNodeRole`). Choose **Delete**, enter the requested confirmation text, then choose **Delete**.

Next steps

The following documentation topics help you to extend the functionality of your cluster.

- The IAM entity (user or role) that created the cluster is the only IAM entity that can make calls to the Kubernetes API server with `kubectl` or the AWS Management Console. If you want other IAM users or roles to have access to your cluster, then you need to add them. For more information, see Enabling IAM user and role access to your cluster (p. 404) and Required permissions (p. 508).
- Deploy a sample application (p. 360) to your cluster.
- Before deploying a cluster for production use, we recommend familiarizing yourself with all of the settings for clusters (p. 23) and nodes (p. 101). Some settings (such as enabling SSH access to Amazon EC2 nodes) must be made when the cluster is created.
- To increase security for your cluster, configure the Amazon VPC Container Networking Interface plugin to use IAM roles for service accounts (p. 280).

Amazon EKS clusters

An Amazon EKS cluster consists of two primary components:

- The Amazon EKS control plane
- Amazon EKS 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 using AWS KMS. 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 nodes (for example, to support `kubectl exec` `logs` `proxy` data flows).

Important
In the Amazon EKS environment, `etcd` storage is limited to 8GB as per upstream guidance. You can monitor the `etcd_db_total_size_in_bytes` metric for the current database size.

Amazon EKS 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.

Note

- You can find out how the different components of Amazon EKS work in Amazon EKS networking (p. 260).
- For connected clusters, see Amazon EKS Connector (p. 545).

Creating an Amazon EKS cluster

This topic provides an overview of the available options and describes what to consider when you create 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. 4) guides. These guides help you to create a simple, default cluster without expanding into all of the available options.

Prerequisites

- An existing VPC and subnets that meet Amazon EKS requirements (p. 260). Before you deploy a cluster for production use, we recommend that you have a thorough understanding of the VPC and subnet requirements. If you don't have a VPC and subnets, you can create them using an Amazon EKS provided AWS CloudFormation template (p. 263).
- The `kubectl` command line tool is installed on your computer or AWS CloudShell. The version can be the same as or up to one minor version earlier or later than the Kubernetes version of your cluster. For example, if your cluster version is `1.21`, you can use `kubectl` version `1.20`,`1.21`, or `1.22` with it. To install or upgrade `kubectl`, see Installing `kubectl` (p. 4).
- Version `2.6.3` or later or `1.23.11` or later of the AWS CLI installed and configured on your computer or AWS CloudShell. For more information, see Installing, updating, and uninstalling the AWS CLI and Quick configuration with `aws configure` in the AWS Command Line Interface User Guide.
- An IAM user or role with permissions to `create` and `describe` an Amazon EKS cluster. For more information, see Actions, resources, and condition keys for Amazon EKS.

When an Amazon EKS cluster is created, the IAM entity (user or role) that creates the cluster is permanently added to the Kubernetes RBAC authorization table as the administrator. This entity has `system:masters` permissions. The identity of this entity isn't visible in your cluster configuration. So, it's important to note the entity that created the cluster and make sure that you never delete it. Initially, only the IAM entity that created the server can make calls to the Kubernetes API server using `kubectl`. If you use the console to create the cluster, you must ensure that the same IAM credentials are in the AWS SDK credential chain when you run `kubectl` commands on your cluster. After your cluster is created, you can grant other IAM entities access to your cluster.

To create an Amazon EKS cluster

1. If you already have a cluster IAM role, or you're going to create your cluster with `eksctl`, then you can skip this step. By default, `eksctl` creates a role for you.

   #### To create an Amazon EKS cluster IAM role

   1. Run the following command to create an IAM trust policy JSON file.

   ```
   cat >eks-cluster-role-trust-policy.json <<EOF
   {
     "Version": "2012-10-17",
     "Statement": [
       {
         "Effect": "Allow",
         "Principal": {
           "Service": "eks.amazonaws.com"
         },
         "Action": "sts:AssumeRole"
       }
     ]
   }
   EOF
   ```

   2. Create the Amazon EKS cluster IAM role. If necessary, preface *eks-cluster-role-trust-policy.json* with the path on your computer that you wrote the file to in the previous step. The command associates the trust policy that you created in the previous step to the role. To create an IAM role, the IAM entity (user or role) that is creating the role must be assigned the following IAM action (permission): `iam:CreateRole`.

   ```
   aws iam create-role --role-name AmazonEKSClusterRole --assume-role-policy-document
    file://"eks-cluster-role-trust-policy.json"
   ```

   3. Attach the Amazon EKS managed policy named `AmazonEKSClusterPolicy` to the role. To attach an IAM policy to an IAM entity (user or role), the IAM entity that is attaching the policy must be assigned one of the following IAM actions (permissions): `iam:AttachUserPolicy` or `iam:AttachRolePolicy`.

   ```
   aws iam attach-role-policy --policy-arn arn:aws:iam::aws:policy/
   AmazonEKSClusterPolicy --role-name AmazonEKSClusterRole
   ```

2. Create an Amazon EKS cluster.

   You can create a cluster by using `eksctl`, the AWS Management Console, or the AWS CLI.

   eksctl

   Prerequisite

   Version `0.104.0` or later of the `eksctl` command line tool installed on your computer or AWS CloudShell. To install or update `eksctl`, see Installing `eksctl` (p. 10).

To create your cluster

Create an Amazon EKS `IPv4` cluster with the Amazon EKS latest Kubernetes version in your default AWS Region. Before running command, make the following replacements:

- Replace *`region-code`* with the AWS Region that you want to create your cluster in.
- Replace *`my-cluster`* with a name for your cluster.
- Replace *`1.22`* with any Amazon EKS supported version (p. 65).
- Change the values for `vpc-private-subnets` to meet your requirements. You can also add additional IDs. You must specify at least two subnet IDs. If you'd rather specify public subnets, you can change `--vpc-private-subnets` to `--vpc-public-subnets`. Public subnets have an associated route table with a route to an internet gateway, but private subnets don't have an associated route table. We recommend using private subnets whenever possible.

  The subnets that you choose must meet the Amazon EKS subnet requirements (p. 261). Before selecting subnets, we recommend that you're familiar with all of the Amazon EKS VPC and subnet requirements and considerations (p. 260). You can't change which subnets you want to use after cluster creation.

```
eksctl create cluster --name my-cluster --region region-code --version 1.22 --vpc-private-subnets subnet-ExampleID1,subnet-ExampleID2 --without-nodegroup
```

Cluster provisioning takes several minutes. While the cluster is being created, several lines of output appear. The last line of output is similar to the following example line.

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

> **Tip**
> To see the most options that you can specify when creating a cluster with `eksctl`, use the **`eksctl create cluster --help`** command. To see all the available options, you can use a `config` file. For more information, see Using config files and the config file schema in the `eksctl` documentation. You can find config file examples on GitHub.

Optional settings

The following are optional settings that, if required, must be added to the previous command. You can only enable these options when you create the cluster, not after. If you need to specify these options, you must create the cluster with an `eksctl` config file and specify the settings, rather than using the previous command.

- If you want to specify one or more security groups that Amazon EKS assigns to the network interfaces that it creates, specify the `securityGroup` option.

  Whether you choose any security groups or not, Amazon EKS creates a security group that enables communication between your cluster and your VPC. Amazon EKS associates this security group, and any that you choose, to the network interfaces that it creates. For more information about the cluster security group that Amazon EKS creates, see the section called "Security group requirements" (p. 267). You can modify the rules in the cluster security group that Amazon EKS creates. If you choose to add your own security groups, you can't change the ones that you choose after cluster creation.
- If you want to specify which `IPv4` Classless Inter-domain Routing (CIDR) block Kubernetes assigns service IP addresses from, specify the `serviceIPv4CIDR` option.

Specifying your own range can help prevent conflicts between Kubernetes services and other networks peered or connected to your VPC. Enter a range in CIDR notation. For example: `10.2.0.0/16`.

The CIDR block must meet the following requirements:

- Be within one of the following ranges: `10.0.0.0/8`, `172.16.0.0/12`, or `192.168.0.0/16`.
- Have a minimum size of `/24` and a maximum size of `/12`.
- Not overlap with the range of the VPC for your Amazon EKS resources.

You can only specify this option when using the `IPv4` address family and only at cluster creation. If you don't specify this, then Kubernetes assigns service IP addresses from either the `10.100.0.0/16` or `172.20.0.0/16` CIDR blocks.

- If you're creating cluster that's version `1.21` or later and want the cluster to assign `IPv6` addresses to pods and services instead of `IPv4` addresses, specify the `ipFamily` option.

  Kubernetes assigns `IPv4` addresses to pods and services, by default. Before deciding to use the `IPv6` family, make sure that you're familiar with all of the considerations and requirements in the the section called "VPC requirements and considerations" (p. 260), the section called "Subnet requirements and considerations" (p. 261), the section called "Security group requirements" (p. 267), and the section called "IPv6" (p. 286) topics. If you choose the `IPv6` family, you can't specify an address range for Kubernetes to assign `IPv6` service addresses from like you can for the `IPv4` family. Kubernetes assigns service addresses from the unique local address range (`fc00::/7`).

AWS Management Console

To create your cluster

1. Open the Amazon EKS console at https://console.aws.amazon.com/eks/home#/clusters.
2. Choose **Add cluster** and then choose **Create**.
3. On the **Configure cluster** page, enter the following fields:
   - **Name** – A name for your cluster. It must be unique.
   - **Kubernetes version** – The version of Kubernetes to use for your cluster.
   - **Cluster service role** – Choose the Amazon EKS cluster IAM role that you created to allow the Kubernetes control plane to manage AWS resources on your behalf.
   - **Secrets encryption** – (Optional) Choose to enable secrets encryption of Kubernetes secrets using a KMS key. You can also enable this after you create your cluster. Before you enable this capability, make sure that you're familiar with the information in the section called "Enabling secret encryption" (p. 47).
   - **Tags** – (Optional) Add any tags to your cluster. For more information, see Tagging your Amazon EKS resources (p. 433).
4. Select **Next**.
5. On the **Specify networking** page, select values for the following fields:
   - **VPC** – Choose an existing VPC that meets Amazon EKS VPC requirements (p. 260) to create your cluster in. Before choosing a VPC, we recommend that you're familiar with all of the requirements and considerations in the section called "VPC and subnet requirements" (p. 260). You can't change which VPC you want to use after cluster creation. If no VPCs are listed, then you need to create one first. For more information, see the section called "Creating a VPC" (p. 263).
   - **Subnets** – By default, all available subnets in the VPC specified in the previous field are preselected. You must select at least two.

The subnets that you choose must meet the Amazon EKS subnet requirements (p. 261). Before selecting subnets, we recommend that you're familiar with all of the Amazon EKS VPC and subnet requirements and considerations (p. 260). You can't change which subnets you want to use after cluster creation.

Security groups – (Optional) Specify one or more security groups that you want Amazon EKS to associate to the network interfaces that it creates.

Whether you choose any security groups or not, Amazon EKS creates a security group that enables communication between your cluster and your VPC. Amazon EKS associates this security group, and any that you choose, to the network interfaces that it creates. For more information about the cluster security group that Amazon EKS creates, see the section called "Security group requirements" (p. 267). You can modify the rules in the cluster security group that Amazon EKS creates. If you choose to add your own security groups, you can't change the ones that you choose after cluster creation.

- **Choose cluster IP address family** – If the version you chose for your cluster is `1.20` or earlier, only the **IPv4** option is available. If you chose `1.21` or later, then **IPv4** and **IPv6** are available.

  Kubernetes assigns `IPv4` addresses to pods and services, by default. Before deciding to use the `IPv6` family, make sure that you're familiar with all of the considerations and requirements in the the section called "VPC requirements and considerations" (p. 260), the section called "Subnet requirements and considerations" (p. 261), the section called "Security group requirements" (p. 267), and the section called "IPv6" (p. 286) topics. If you choose the `IPv6` family, you can't specify an address range for Kubernetes to assign `IPv6` service addresses from like you can for the `IPv4` family. Kubernetes assigns service addresses from the unique local address range (`fc00::/7`).

- (Optional) Choose **Configure Kubernetes Service IP address range** and specify a **Service `IPv4` range**.

  Specifying your own range can help prevent conflicts between Kubernetes services and other networks peered or connected to your VPC. Enter a range in CIDR notation. For example: `10.2.0.0/16`.

  The CIDR block must meet the following requirements:

  - Be within one of the following ranges: `10.0.0.0/8`, `172.16.0.0/12`, or `192.168.0.0/16`.
  - Have a minimum size of `/24` and a maximum size of `/12`.
  - Not overlap with the range of the VPC for your Amazon EKS resources.

  You can only specify this option when using the `IPv4` address family and only at cluster creation. If you don't specify this, then Kubernetes assigns service IP addresses from either the `10.100.0.0/16` or `172.20.0.0/16` CIDR blocks.

- For **Cluster endpoint access**, select an option. After your cluster is created, you can change this option. Before selecting a non-default option, make sure to familiarize yourself with the options and their implications. For more information, see the section called "Configuring endpoint access" (p. 42).

6. You can accept the defaults in the **Networking add-ons** section to install the default version of the Amazon VPC CNI plugin for Kubernetes (p. 269), CoreDNS (p. 338), and `kube-proxy` (p. 344) Amazon EKS add-ons. Or, alternatively, you can select a different version. If you don't require the functionality of any of the add-ons, you can remove them once your cluster is created. If you need to manage Amazon EKS managed settings for any of these add-ons yourself, remove Amazon EKS management of the add-on after your cluster is created. For more information, see Amazon EKS add-ons (p. 389).
7. Select **Next**.

8. On the **Configure logging** page, you can optionally choose which log types that you want to enable. By default, each log type is **Disabled**. Before selecting a different option, familiarize yourself with the information in Amazon EKS control plane logging (p. 50). After you create the cluster, you can change this option.
9. Select **Next**.
10. On the **Review and create** page, review the information that you entered or selected on the previous pages. If you need to make changes, choose **Edit**. When you're satisfied, choose **Create**. The **Status** field shows **CREATING** while the cluster is provisioned.

    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. 529).

    Cluster provisioning takes several minutes.

AWS CLI

To create your cluster

1. Create your cluster with the command that follows. Before running the command, make the following replacements:
   - Replace *region-code* with the AWS Region that you want to create your cluster in.
   - Replace *my-cluster* with a name for your cluster.
   - Replace *1.22* with any Amazon EKS supported version (p. 65).
   - Replace *111122223333* with your account ID and *AmazonEKSClusterRole* with the name of your cluster IAM role.
   - Replace the values for `subnetIds` with your own. You can also add additional IDs. You must specify at least two subnet IDs.

     The subnets that you choose must meet the Amazon EKS subnet requirements (p. 261). Before selecting subnets, we recommend that you're familiar with all of the Amazon EKS VPC and subnet requirements and considerations (p. 260). You can't change which subnets you want to use after cluster creation.
   - If you don't want to specify a security group ID, remove `,securityGroupIds=sg-ExampleID1` from the command. If you want to specify one or more security group IDs, replace the values for `securityGroupIds` with your own. You can also add additional IDs.

     Whether you choose any security groups or not, Amazon EKS creates a security group that enables communication between your cluster and your VPC. Amazon EKS associates this security group, and any that you choose, to the network interfaces that it creates. For more information about the cluster security group that Amazon EKS creates, see the section called "Security group requirements" (p. 267). You can modify the rules in the cluster security group that Amazon EKS creates. If you choose to add your own security groups, you can't change the ones that you choose after cluster creation.

```
aws eks create-cluster --region region-code --name my-cluster --kubernetes-version 1.22 \
   --role-arn arn:aws:iam::111122223333:role/AmazonEKSClusterRole \
   --resources-vpc-config subnetIds=subnet-ExampleID1,subnet-ExampleID2,securityGroupIds=sg-ExampleID1
```

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. 529).

Optional settings

The following are optional settings that, if required, must be added to the previous command. You can only enable these options when you create the cluster, not after.

- If you want to specify which `IPv4` Classless Inter-domain Routing (CIDR) block Kubernetes assigns service IP addresses from, you must specify it by adding the **`--kubernetes-network-config serviceIpv4Cidr=`*`CIDR block`*** to the following command.

  Specifying your own range can help prevent conflicts between Kubernetes services and other networks peered or connected to your VPC. Enter a range in CIDR notation. For example: `10.2.0.0/16`.

  The CIDR block must meet the following requirements:

  - Be within one of the following ranges: `10.0.0.0/8`, `172.16.0.0/12`, or `192.168.0.0/16`.
  - Have a minimum size of `/24` and a maximum size of `/12`.
  - Not overlap with the range of the VPC for your Amazon EKS resources.

  You can only specify this option when using the `IPv4` address family and only at cluster creation. If you don't specify this, then Kubernetes assigns service IP addresses from either the `10.100.0.0/16` or `172.20.0.0/16` CIDR blocks.
- If you're creating a cluster of version `1.21` or later and want the cluster to assign `IPv6` addresses to pods and services instead of `IPv4` addresses, add **`--kubernetes-network-config ipFamily=ipv6`** to the following command.

  Kubernetes assigns `IPv4` addresses to pods and services, by default. Before deciding to use the `IPv6` family, make sure that you're familiar with all of the considerations and requirements in the the section called "VPC requirements and considerations" (p. 260), the section called "Subnet requirements and considerations" (p. 261), the section called "Security group requirements" (p. 267), and the section called "IPv6" (p. 286) topics. If you choose the `IPv6` family, you can't specify an address range for Kubernetes to assign `IPv6` service addresses from like you can for the `IPv4` family. Kubernetes assigns service addresses from the unique local address range (`fc00::/7`).

2. It takes several minutes to provision the cluster. You can query the status of your cluster with the following command.

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

   Don't proceed to the next step until the output returned is `ACTIVE`.

3. If you created your cluster using `eksctl`, then you can skip this step. This is because `eksctl` already completed this step for you. Enable `kubectl` to communicate with your cluster by adding a new context to the `kubectl config` file. For more information about how to create and update the file, see the section called "Create a `kubeconfig` for Amazon EKS" (p. 415).

```
aws eks update-kubeconfig --region region-code --name my-cluster
```

The example output is as follows.

```
Added new context arn:aws:eks:region-code:111122223333:cluster/my-cluster to /home/
username/.kube/config
```

4. Confirm communication with your cluster by running the following command.

```
kubectl get svc
```

The example output is as follows.

```
NAME         TYPE        CLUSTER-IP   EXTERNAL-IP   PORT(S)   AGE
kubernetes   ClusterIP   10.100.0.1   <none>        443/TCP   28h
```

5. (Recommended) To use some Amazon EKS add-ons, or to enable individual Kubernetes workloads to have specific AWS Identity and Access Management (IAM) permissions, create an IAM OpenID Connect (OIDC) provider for your cluster. For instructions on how to create an IAM OIDC provider for your cluster, see Create an IAM OIDC provider for your cluster (p. 448). You only need to create an IAM OIDC provider for your cluster once. To learn more about Amazon EKS add-ons, see Amazon EKS add-ons (p. 389). To learn more about assigning specific IAM permissions to your workloads, see IAM roles for service accounts technical overview (p. 444).
6. (Recommended) Configure your cluster for the Amazon VPC CNI plugin for Kubernetes plugin before deploying Amazon EC2 nodes to your cluster. By default, the plugin was installed with your cluster. When you add Amazon EC2 nodes to your cluster, the plugin is automatically deployed to each Amazon EC2 node that you add. The plugin requires you to attach one of the following IAM policies to an IAM role:

   - **`AmazonEKS_CNI_Policy` managed IAM policy** – If your cluster uses the `IPv4` family
   - **An IAM policy that you create (p. 284)** – If your cluster uses the `IPv6` family

   The IAM role that you attach the policy to can be the node IAM role, or a dedicated role used only for the plugin. We recommend attaching the policy to this role. For more information about creating the role, see the section called “Configure plugin for IAM account” (p. 280) or the section called “Node IAM role” (p. 476).
7. If you deployed your cluster using the AWS Management Console, you can skip this step. The AWS Management Console deploys the Amazon VPC CNI plugin for Kubernetes, CoreDNS, and `kube-proxy` Amazon EKS add-ons, by default.

   (Optional) If you deploy your cluster using either `eksctl` or the AWS CLI, then the Amazon VPC CNI plugin for Kubernetes, CoreDNS, and `kube-proxy` self-managed add-ons are deployed. You can migrate the Amazon VPC CNI plugin for Kubernetes, CoreDNS, and `kube-proxy` self-managed add-ons that are deployed with your cluster to Amazon EKS add-ons. For more information, see Amazon EKS add-ons (p. 389).

Recommended next steps:

- Grant the IAM entity that created the cluster the required permissions to view Kubernetes resources in the AWS Management Console (p. 508)
- Grant IAM entities access to your cluster (p. 404). If you want the entities to view Kubernetes resources in the Amazon EKS console, grant the the section called “Required permissions” (p. 508) to the entities.

- Enable the private endpoint for your cluster (p. 42) if you want nodes and users to access your cluster from within your VPC.
- Enable secrets encryption for your cluster (p. 47)
- Configure logging for your cluster (p. 50)
- Add nodes to your cluster (p. 101)

Updating an Amazon EKS cluster Kubernetes version

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

> **Important**
> We recommend that, before you update to a new Kubernetes version, you review the information in Amazon EKS Kubernetes versions (p. 65) and also review in the update steps in this topic. If you're updating to version `1.22`, you must make the changes listed in the section called "Kubernetes version `1.22` prerequisites" (p. 35) to your cluster before updating it.

New Kubernetes versions sometimes introduce significant changes. Therefore, we recommend that you test the behavior of your applications against a new Kubernetes version before you update your production clusters. You can do this by building a continuous integration workflow to test your application behavior 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're 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 aren't 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're constantly evaluating and improving our Kubernetes infrastructure management processes.

To update the cluster, Amazon EKS requires up to five free IP addresses from the subnets that you specified when you created your cluster. Amazon EKS creates new cluster elastic network interfaces (network interfaces) in any of the subnets that you specified. The network interfaces may be created in different subnets than your existing network interfaces are in, so make sure that your security group rules allow required cluster communication (p. 267) for any of the subnets that you specified when you created your cluster. If any of the subnets that you specified when you created the cluster don't exist, don't have enough free IP addresses, or don't have security group rules that allows necessary cluster communication, then the update can fail.

> **Note**
> Even though Amazon EKS runs a highly available control plane, you might experience minor service interruptions during an update. For example, assume that you attempt to connect to an API server around when it's terminated and replaced by a new API server that's 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.

Update the Kubernetes version for your Amazon EKS cluster

To update the Kubernetes version for your cluster

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

   - Get the Kubernetes version of your cluster control plane with the **`kubectl version --short`** command.

     ```
     kubectl version --short
     ```

   - Get the Kubernetes version of your nodes with the **`kubectl get nodes`** command. This command returns all self-managed and managed Amazon EC2 and Fargate nodes. Each Fargate pod is listed as its own node.

     ```
     kubectl get nodes
     ```

   Before updating your control plane to a new Kubernetes version, make sure that the Kubernetes minor version of both the managed nodes and Fargate nodes in your cluster are the same as your control plane's version. For example, if your control plane is running version `1.21` and one of your nodes is running version `1.20`, then you must update your nodes to version `1.21`. We also recommend that you update your self-managed nodes to the same version as your control plane before updating the control plane. For more information, see Updating a managed node group (p. 115) and Self-managed node updates (p. 141). To update the version of a Fargate node, first delete the pod that's represented by the node. Then update your control plane. Any remaining pods will update to the new version after you redeploy them.
2. By default, the pod security policy admission controller is enabled on Amazon EKS clusters. Before updating your cluster, ensure that the proper pod security policies are in place. This is to avoid potential security issues. You can check for the default policy with the **`kubectl get psp eks.privileged`** command.

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

   If you receive the following error, see default pod security policy (p. 503) before proceeding.

   ```
   Error from server (NotFound): podsecuritypolicies.extensions "eks.privileged" not found
   ```
3. If the Kubernetes version that you originally deployed your cluster with was Kubernetes `1.18` or later, skip this step.

   You might need to remove a discontinued term from your CoreDNS manifest.

   a. Check to see if your CoreDNS manifest has a line that only has the word `upstream`.

      ```
      kubectl get configmap coredns -n kube-system -o jsonpath='{$.data.Corefile}' | grep
       upstream
      ```

      If no output is returned, this means that your manifest doesn't have the line. If this is the case, skip to the next step. If the word `upstream` is returned, remove the line.

   b. Remove the line near the top of the file that only has the word `upstream` in the configmap file. Don't change anything else in the file. After the line is removed, save the changes.

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

4. Update your cluster using `eksctl`, the AWS Management Console, or the AWS CLI.

   Important

   - If you're updating to version `1.22`, you must make the changes listed in the section called "Kubernetes version `1.22` prerequisites" (p. 35) to your cluster before updating it.
   - Because Amazon EKS runs a highly available control plane, you can update only one minor version at a time. For more information about this requirement, see Kubernetes Version and Version Skew Support Policy. Assume that your current version is `1.20` and you want to update to 1.22. Then, you must first update your cluster to `1.21` and then later update it from `1.21` to `1.22`.
   - Make sure that the `kubelet` on your managed and Fargate nodes are at the same Kubernetes version as your control plane before you update. We recommend that your self-managed nodes are at the same version as the control plane. They can be only up to one version behind the current version of the control plane.
   - If your cluster is configured with a version of the Amazon VPC CNI plugin that is earlier than `1.8.0`, then we recommend that you update the plugin to version `1.11.2` before updating your cluster to version `1.21` or later. For more information, see Updating the Amazon VPC CNI plugin for Kubernetes add-on (p. 272) or Updating the Amazon VPC CNI plugin for Kubernetes self-managed add-on (p. 275).

eksctl

This procedure requires `eksctl` version `0.104.0` or later. You can check your version with the following command:

```
eksctl version
```

For instructions on how to install and update `eksctl`, see Installing or upgrading `eksctl` (p. 10).

Update the Kubernetes version of your Amazon EKS control plane to one minor version later than its current version with the following command. Replace *`my-cluster`* with your cluster name.

```
eksctl upgrade cluster --name my-cluster --approve
```

The update takes several minutes to complete.

AWS Management Console

a. Open the Amazon EKS console at https://console.aws.amazon.com/eks/home#/clusters.
b. Choose the name of the Amazon EKS cluster to update and choose **Update cluster version**.
c. For **Kubernetes version**, select the version to update your cluster to and choose **Update**.
d. For **Cluster name**, enter the name of your cluster and choose **Confirm**.

   The update takes several minutes to complete.

AWS CLI

a. Update your Amazon EKS cluster with the following AWS CLI command. Replace the *`example-values`* with your own.

```
aws eks update-cluster-version \
 --region region-code \
 --name my-cluster \
 --kubernetes-version 1.22
```

The example output is as follows.

```
{
    "update": {
        "id": "b5f0ba18-9a87-4450-b5a0-825e6e84496f",
        "status": "InProgress",
        "type": "VersionUpdate",
        "params": [
            {
                "type": "Version",
                "value": "1.22"
            },
            {
                "type": "PlatformVersion",
                "value": "eks.1"
            }
        ],
...
        "errors": []
    }
}
```

b. Monitor the status of your cluster update with the following command. Use the cluster name and update ID that the previous command returned. When a `Successful` status is displayed, the update is complete. The update takes several minutes to complete.

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

The example output is as follows.

```
{
    "update": {
        "id": "b5f0ba18-9a87-4450-b5a0-825e6e84496f",
        "status": "Successful",
        "type": "VersionUpdate",
        "params": [
            {
                "type": "Version",
                "value": "1.22"
            },
            {
                "type": "PlatformVersion",
                "value": "eks.1"
            }
        ],
...
        "errors": []
    }
}
```

5. After your cluster update is complete, update your nodes to the same Kubernetes minor version as your updated cluster. For more information, see Self-managed node updates (p. 141) and Updating

a managed node group (p. 115). Any new pods that are launched on Fargate have a `kubelet` version that matches your cluster version. Existing Fargate pods aren't changed.

6. (Optional) If you deployed the Kubernetes Cluster Autoscaler to your cluster before updating the cluster, update the Cluster Autoscaler to the latest version that matches the Kubernetes major and minor version that you updated 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 `1.22` find the latest Cluster Autoscaler release that begins with `1.22`. Record the semantic version number (`<1.22.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. If necessary, replace *1.22.n* with your own value.

```
kubectl -n kube-system set image deployment.apps/cluster-autoscaler cluster-autoscaler=k8s.gcr.io/autoscaling/cluster-autoscaler:v1.22.n
```

7. (Clusters with GPU nodes only) If your cluster has 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/v0.9.0/nvidia-device-plugin.yml
```

8. Update the Amazon VPC CNI plugin for Kubernetes, CoreDNS, and `kube-proxy` add-ons. If you updated your cluster to version `1.21` or later, than we recommend updating the add-ons to the minimum versions listed in Service account tokens (p. 443).

   - If you updated your cluster to version `1.18`, you can add Amazon EKS add-ons. For instructions, see Adding the Amazon VPC CNI Amazon EKS add-on (p. 270), Adding the CoreDNS Amazon EKS add-on (p. 338), and Adding the `kube-proxy` Amazon EKS add-on (p. 345). To learn more about Amazon EKS add-ons, see Amazon EKS add-ons (p. 389).
   - If you updated to version `1.19` or later and are using Amazon EKS add-ons, in the Amazon EKS console, select **Clusters**, then select the name of the cluster that you updated in the left navigation pane. Notifications appear in the console. They inform you that a new version is available for each addon that has an available update. To update an add-on, select the **Add-ons** tab. In one of the boxes for an add-on that has an update available, select **Update now**, select an available version, and then select **Update**.
   - Alternately, you can use the AWS CLI or `eksctl` to update the Amazon VPC CNI plugin for Kubernetes (p. 275), CoreDNS (p. 340), and `kube-proxy` (p. 346) Amazon EKS add-ons.

Kubernetes version `1.22` prerequisites

A number of deprecated beta APIs (`v1beta1`) have been removed in version `1.22` in favor of the GA (`v1`) version of those same APIs. As noted in the Kubernetes version `1.22` API and Feature removal blog and deprecated API migration guide, API changes are required for the following deployed resources before updating a cluster to version `1.22`.

Before updating your cluster to Kubernetes version `1.22`, make sure to do the following:

- Change your YAML manifest files and clients to reference the new APIs.
- Update custom integrations and controllers to call the new APIs.
- Make sure that you use an updated version of any third-party tools. These tools include ingress controllers, service mesh controllers, continuous delivery systems, and other tools that call the new

APIs. To check for discontinued API usage in your cluster, enable audit control plane logging and specify `v1beta` as an event filter. Replacement APIs are available in Kubernetes for several versions.

- If you currently have the AWS Load Balancer Controller deployed to your cluster, you must update it to version `2.4.1` before updating your cluster to Kubernetes version `1.22`.

Important

When you update clusters to version `1.22`, existing persisted objects can be accessed using the new APIs. However, you must migrate manifests and update clients to use these new APIs. Updating the clusters prevents potential workload failures.

Kubernetes version `1.22` removes support from the following beta APIs. Migrate your manifests and API clients based on the following information:

Resource	Beta version	GA version	Notes
`ValidatingWebhookConfiguration` `MutatingWebhookConfiguration`	`admissionregistration.k8s.io/v1beta1`	`admissionregistration.k8s.io/v1`	• `webhooks[*].failurePolicy` default changed from `Ignore` to `Fail` for `v1`. • `webhooks[*].matchPolicy` default changed from `Exact` to `Equivalent` for `v1`. • `webhooks[*].timeoutSeconds` default changed from `30s` to `10s` for `v1`. • `webhooks[*].sideEffects` default value is removed, and the field made required, and only `None` and `NoneOnDryRun` are permitted for `v1`. • `webhooks[*].admissionReviewVersions` default value is removed and the field made required for `v1` (supported versions for `AdmissionReview` are `v1` and `v1beta1`). • `webhooks[*].name` must be unique in the list for objects created via `admissionregistration.k8s.io/v1`.
`CustomResourceDefinition`	`apiextensions.k8s.io/v1beta1`	`apiextensions.k8s.io/v1`	• `spec.scope` is no longer defaulted to `Namespaced` and must be explicitly specified. • `spec.version` is removed in `v1`; use `spec.versions` instead • `spec.validation` is removed in `v1`; use `spec.versions[*].schema` instead. • `spec.subresources` is removed in `v1`; use `spec.versions[*].subresources` instead.

Resource	Beta version	GA version	Notes
			• `spec.additionalPrinterColumns` is removed in v1; use spec.versions[*].additionalPrinterColumns instead. • `spec.conversion.webhookClientConfig` is moved to `spec.conversion.webhook.clientConfig` in `v1`. • `spec.conversion.conversionReviewVersion` is moved to `spec.conversion.webhook.conversionRevie` in `v1`. • `spec.versions[*].schema.openAPIV3Schema` is now required when creating `v1` `CustomResourceDefinition` objects, and must be a structural schema. • `spec.preserveUnknownFields: true` is disallowed when creating `v1` `CustomResourceDefinition` objects; it must be specified within schema definitions as `x-kubernetes-preserve-unknown-fields: true`. • In `additionalPrinterColumns` items, the `JSONPath` field was renamed to `jsonPath` in `v1` (fixes #66531).
`APIService`	`apiregistration.k8s.io/v1beta1`	`apiregistration.k8s.io/v1`	None
`TokenReview`	`authentication.k8s.io/v1beta1`	`authentication.k8s.io/v1`	None
`SubjectAccessReview` `LocalSubjectAccessReview` `SelfSubjectAccessReview`	`authorization.k8s.io/v1beta1`	`authorization.k8s.io/v1`	`spec.group` is renamed to `spec.groups`

Resource	Beta version	GA version	Notes

`CertificateSigningRequest`	`certificates.k8s.io/v1beta1`	`certificates.k8s.io/v1`	• For API clients requesting certificates:
			  • spec.signerName is now required (see known Kubernetes signers), and requests for `kubernetes.io/legacy-unknown` are not allowed to be created via the certificates.k8s.io/v1 API
			  • `spec.usages` is now required, may not contain duplicate values, and must only contain known usages
			• For API clients approving or signing certificates:
			  • `status.conditions` may not contain duplicate types
			  • `status.conditions[*].status` is now required
			  • `status.certificate` must be PEM-encoded, and contain only `CERTIFICATE` blocks

`Lease`	`coordination.k8s.io/v1beta1`	`coordination.k8s.io/v1`	None
`Ingress`	• `extensions/v1beta1` • `networking.k8s.io/v1beta1`	`networking.k8s.io/v1`	• `spec.backend` is renamed to `spec.defaultBackend` • The backend `serviceName` field is renamed to service.name • Numeric backend `servicePort` fields are renamed to `service.port.number` • String backend `servicePort` fields are renamed to `service.port.name` • `pathType` is now required for each specified path. Options are `Prefix`, `Exact`, and `ImplementationSpecific`. To match the undefined `v1beta1` behavior, use `ImplementationSpecific`
`IngressClass`	`networking.k8s.io/v1beta1`	`networking.k8s.io/v1`	None
`RBAC`	`rbac.authorization.k8s.io/v1beta1`	`rbac.authorization.k8s.io/v1`	None
`PriorityClass`	`scheduling.k8s.io/v1beta1`	`scheduling.k8s.io/v1`	None

Resource	Beta version	GA version	Notes
CSIDriver CSINode StorageClass VolumeAttachment	storage.k8s.io/ v1beta1	storage.k8s.io/ v1	None

To learn more about the API removal, see the Deprecated API migration guide.

Deleting an Amazon EKS 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.

To remove a connected cluster, see Deregistering a cluster (p. 550)

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.
- If you receive an error because the cluster creator has been removed, see this article to resolve.

You can delete a cluster with `eksctl`, the AWS Management Console, or the AWS CLI. Select the tab with the name of the tool that you'd like to use to delete your cluster.

eksctl

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

This procedure requires `eksctl` version `0.104.0` or later. You can check your version with the following command:

```
eksctl version
```

For instructions on how to install or upgrade `eksctl`, see Installing or upgrading `eksctl` (p. 10).

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 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-nodes"
[#]  waiting for stack "eksctl-prod-nodegroup-standard-nodes" 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 all node groups and Fargate profiles.

   a. Open the Amazon EKS console at https://console.aws.amazon.com/eks/home#/clusters.
   b. In the left navigation pane, choose Amazon EKS **Clusters**, and then in the tabbed list of clusters, choose the name of the cluster that you want to delete.
   c. Choose the **Compute** tab and choose a node group to delete. Choose **Delete**, enter the name of the node group, and then choose **Delete**. Delete all node groups in the cluster.

      Note
      The node groups listed are managed node groups (p. 105) only.
   d. Choose a **Fargate Profile** to delete, select **Delete**, enter the name of the profile, and then choose **Delete**. Delete all Fargate profiles in the cluster.
4. Delete all self-managed node AWS CloudFormation stacks.

   a. Open the AWS CloudFormation console at https://console.aws.amazon.com/cloudformation.
   b. Choose the node stack to delete, and then choose **Delete**.
   c. In the **Delete stack** confirmation dialog box, choose **Delete stack**. Delete all self-managed node stacks in the cluster.
5. Delete the cluster.

   a. Open the Amazon EKS console at https://console.aws.amazon.com/eks/home#/clusters.
   b. choose the cluster to delete and choose **Delete**.
   c. On the delete cluster confirmation screen, choose **Delete**.
6. (Optional) Delete the VPC AWS CloudFormation stack.

   a. Open the AWS CloudFormation console at https://console.aws.amazon.com/cloudformation.
   b. Select the VPC stack to delete, and then choose **Delete**.
   c. In the **Delete stack** confirmation dialog box, choose **Delete stack**.

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 all node groups and Fargate profiles.

   a. List the node groups in your cluster with the following command.

   ```
   aws eks list-nodegroups --cluster-name <my-cluster>
   ```

   Note
   The node groups listed are managed node groups (p. 105) only.

   b. Delete each node group with the following command. Delete all node groups in the cluster.

   ```
   aws eks delete-nodegroup --nodegroup-name <my-nodegroup> --cluster-name <my-
   cluster>
   ```

   c. List the Fargate profiles in your cluster with the following command.

   ```
   aws eks list-fargate-profiles --cluster-name <my-cluster>
   ```

   d. Delete each Fargate profile with the following command. Delete all Fargate profiles in the cluster.

   ```
   aws eks delete-fargate-profile --fargate-profile-name <my-fargate-profile> --
   cluster-name <my-cluster>
   ```

4. Delete all self-managed node AWS CloudFormation stacks.

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

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

   b. Delete each node stack with the following command, replacing *<node-stack>* with your node stack name. Delete all self-managed node stacks in the cluster.

   ```
   aws cloudformation delete-stack --stack-name <node-stack>
   ```

5. Delete the cluster with the following command, replacing <my-cluster> with your cluster name.

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

6. (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 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 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*.

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 your cluster's VPC (such as node to control plane communication) leave the VPC but not Amazon's network.

Endpoint public access	Endpoint private access	Behavior
		• 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 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 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. 46). • 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. 31) your cluster.

You can modify your cluster API server endpoint access using the AWS Management Console or AWS CLI. Select the tab with the name of the tool that you'd like to use to modify your endpoint access with.

AWS Management Console

To modify your cluster API server endpoint access using the 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. Choose the **Networking** tab and 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. 437). 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 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 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 an allowed CIDR block on your public endpoint.
7. Choose **Update** to finish.

AWS CLI

To modify your cluster API server endpoint access using the AWS CLI

Complete the following steps using the AWS CLI version `1.23.11` 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. 437). If you restrict access to your public endpoint using CIDR blocks, it is recommended that you also enable private endpoint access so that 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 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 an allowed 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 my-cluster \
    --resources-vpc-config
 endpointPublicAccess=<true>,publicAccessCidrs="203.0.113.5/32",endpointPrivateAccess=<true>
```

   The example output is as follows.

```
{
    "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 my-cluster \
    --update-id e6f0905f-a5d4-4a2a-8c49-EXAMPLE00000
```

The example output is as follows.

```
{
    "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 requirements and considerations (p. 267).

  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 Enabling IAM user and role access to your cluster (p. 404) and Unauthorized or access denied (`kubectl`) (p. 530).
- **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 requirements and considerations (p. 267).

  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 Enabling IAM user and role access to your cluster (p. 404) and Unauthorized or access denied (`kubectl`) (p. 530).

Enabling secret encryption on an existing cluster

If you enable secrets encryption, the Kubernetes secrets are encrypted using the AWS KMS key that you select. The KMS key must meet the following conditions:

- Symmetric
- Can encrypt and decrypt data
- Created in the same AWS Region as the cluster
- If the KMS key was created in a different account, the user must have access to the KMS key.

For more information, see Allowing users in other accounts to use a KMS key in the *AWS Key Management Service Developer Guide*.

Warning
You can't disable secrets encryption after enabling it. This action is irreversible.

eksctl

You can enable encryption in two ways:

- Add encryption to your cluster with a single command.

  To automatically re-encrypt your secrets, run the following command.

```
eksctl utils enable-secrets-encryption \
```

```
    --cluster <my-cluster> \
    --key-arn arn:aws:kms:<Region-code>:<account>:key/<key>
```

To opt-out of automatically re-encrypting your secrets, run the following command.

```
eksctl utils enable-secrets-encryption
    --cluster my-cluster \
    --key-arn arn:aws:kms:region-code:account:key/key \
    --encrypt-existing-secrets=false
```

- Add encryption to your cluster with a `.yaml` file.

```
# cluster.yaml

apiVersion: eksctl.io/v1alpha5
kind: ClusterConfig

metadata:
  name: my-cluster
  region: region-code

secretsEncryption:
  keyARN: arn:aws:kms:<Region-code>:<account>:key/<key>
```

To have your secrets re-encrypt automatically, run the following command.

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

To opt out of automatically re-encrypting your secrets, run the following command.

```
eksctl utils enable-secrets-encryption -f kms-cluster.yaml --encrypt-existing-
secrets=false
```

AWS Management Console

1. Open the Amazon EKS console at https://console.aws.amazon.com/eks/home#/clusters.
2. Choose the cluster that you want to add KMS encryption to.
3. Choose the **Overview** tab (this is selected by default).
4. Scroll down to the **Secrets encryption** section and choose **Enable**.
5. Select a key from the dropdown list and choose the **Enable** button. If no keys are listed, you must create one first. For more information, see Creating keys
6. Choose the **Confirm** button to use the chosen key.

AWS CLI

1. Associate the secrets encryption configuration with your cluster using the following AWS CLI command. Replace the *example-values* with your own.

```
aws eks associate-encryption-config \
    --cluster-name <my-cluster> \
    --encryption-config '[{"resources":["secrets"],"provider":
{"keyArn":"arn:aws:kms:<Region-code>:<account>:key/<key>"}}]'
```

The output is as follows.

```
{
  "update": {
    "id": "3141b835-8103-423a-8e68-12c2521ffa4d",
    "status": "InProgress",
    "type": "AssociateEncryptionConfig",
    "params": [
      {
        "type": "EncryptionConfig",
        "value": "[{\"resources\":[\"secrets\"],\"provider\":{\"keyArn\":
\"arn:aws:kms:region-code:account:key/key\"}}]"
      }
    ],
    "createdAt": 1613754188.734,
    "errors": []
  }
}
```

2. You can monitor the status of your encryption update with the following command. Use the specific `cluster name` and `update ID` that was returned in the previous output. When a `Successful` status is displayed, the update is complete.

```
aws eks describe-update \
    --region <Region-code> \
    --name <my-cluster> \
    --update-id <3141b835-8103-423a-8e68-12c2521ffa4d>
```

The output is as follows.

```
{
  "update": {
    "id": "3141b835-8103-423a-8e68-12c2521ffa4d",
    "status": "Successful",
    "type": "AssociateEncryptionConfig",
    "params": [
      {
        "type": "EncryptionConfig",
        "value": "[{\"resources\":[\"secrets\"],\"provider\":{\"keyArn\":
\"arn:aws:kms:region-code:account:key/key\"}}]"
      }
    ],
    "createdAt": 1613754188.734>,
    "errors": []
  }
}
```

3. To verify that encryption is enabled in your cluster, run the `describe-cluster` command. The response contains an `EncryptionConfig` string.

```
aws eks describe-cluster --region <Region-code> --name <my-cluster>
```

After you enabled encryption on your cluster, you must encrypt all existing secrets with the new key:

Note
If you use `eksctl`, running the following command is necessary only if you opt out of re-encrypting your secrets automatically.

```
kubectl get secrets --all-namespaces -o json | kubectl annotate --overwrite -f - kms-
encryption-timestamp="time value"
```

Warning
If you enable secrets encryption for an existing cluster and the KMS key that you use is ever deleted, then there's no way to recover the cluster. If you delete the KMS key, you permanently put the cluster in a degraded state. For more information, see Deleting AWS KMS keys.

Note
By default, the `create-key` command creates a symmetric encryption KMS key with a key policy that gives the account root admin access on AWS KMS actions and resources. If you want to scope down the permissions, make sure that the `kms:DescribeKey` and `kms:CreateGrant` actions are permitted on the policy for the principal that calls the `create-cluster` API.
For clusters using KMS Envelope Encryption, `kms:CreateGrant` permissions are required. The condition `kms:GrantIsForAWSResource` is not supported for the CreateCluster action, and should not be used in KMS policies to control `kms:CreateGrant` permissions for users performing CreateCluster.

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 and the audit policy 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 Cluster management (p. 424).
- **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.
2. Choose the name of the cluster to display your cluster information.
3. Choose the **Logging** tab and choose **Manage logging**.
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 **Save changes** 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. Replace *my-cluster* with your cluster name and specify your desired endpoint access values.

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

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

   The example output is as follows.

```
{
    "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 describe-update \
    --region region-code\
    --name my-cluster \
    --update-id 883405c8-65c6-4758-8cee-2a7c1340a6d9
```

The example output is as follows.

```
{
    "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. The link opens the console and 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...>`

Viewing API server flags in the Amazon CloudWatch console

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. 50). 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. On the **Logging** tab, choose **Manage logging**.
   d. For **API server**, make sure that the log type is **Enabled**.
   e. Choose **Save changes** to finish.
2. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/
3. Choose **Logs**, then **Log groups** in the side menu. Choose the cluster of which you want to see the logs, then choose the **Log streams** tab.
4. 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.
5. 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.

Enabling Windows support for your Amazon EKS cluster

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

Considerations

- Amazon EC2 instance types `C3`, `C4`, `D2`, `I2`, `M4` (excluding `m4.16xlarge`), `M6a.x`, and `R3` instances aren't supported for Windows workloads.

- Host networking mode is not supported for Windows workloads.
- Amazon EKS clusters must contain one or more Linux or Fargate nodes to run core system pods that only run on Linux, such as CoreDNS.
- The `kubelet` and `kube-proxy` event logs are redirected to the `EKS` Windows Event Log and are set to a 200 MB limit.
- You can't use Security groups for pods (p. 314) with pods running on Windows nodes.
- You can't use custom networking (p. 298) with Windows nodes.
- You can't use IP prefixes (p. 311) with Windows nodes. This is a requirement for using IPv6 (p. 286), so you can't use `IPv6` with Windows nodes either.
- Windows nodes support one elastic network interface per node. The number of pods that you can run per Windows 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 Windows Instances*.
- In an Amazon EKS cluster, a single service with a load balancer can support up to 64 back-end pods. Each pod has its own unique IP address. This is a limitation of the Windows operating system on the Amazon EC2 nodes.
- You can't deploy Windows managed or Fargate nodes. You can only create self-managed Windows nodes. For more information, see Launching self-managed Windows nodes (p. 136).
- You can't retrieve logs from the `vpc-resource-controller` Pod. You previously could when you deployed the controller to the data plane.
- There is a cool down period before an `IPv4` address is assigned to a new Pod. This prevents traffic from flowing to an older Pod with the same `IPv4` address due to stale `kube-proxy` rules.
- The source for the controller is managed on GitHub. To contribute to, or file issues against the controller, visit the project on GitHub.

Prerequisites

- An existing cluster. The cluster must be running one of the Kubernetes versions and platform versions listed in the following table. Any Kubernetes and platform versions later than those listed are also supported. If your cluster or platform version is earlier than one of the following versions, you need to enable legacy Windows support (p. 57) on your cluster's data plane. Once your cluster is at one of the following Kubernetes and platform versions, or later, you can remove legacy Windows support (p. 56) and enable Windows support (p. 55) on your control plane.

Kubernetes version	Platform version
1.22	eks.1
1.21	eks.3
1.20	eks.3
1.19	eks.7
1.18	eks.9

  Your cluster must have at least one (we recommend at least two) Linux node or Fargate pod to run CoreDNS. If you enable legacy Windows support, you must use a Linux node (you can't use a Fargate pod) to run CoreDNS.
- An existing Amazon EKS cluster IAM role (p. 474).

Enabling Windows support

If your cluster is not at, or later, than one of the Kubernetes and platform versions listed in the Prerequisites (p. 54), you must enable legacy Windows support instead. For more information, see Enabling legacy Windows support (p. 57).

If you've never enabled Windows support on your cluster, skip to the next step.

If you enabled Windows support on a cluster that is earlier than a Kubernetes or platform version listed in the Prerequisites (p. 54), then you must first remove the `vpc-resource-controller` and `vpc-admission-webhook` from your data plane (p. 56). They're deprecated and no longer needed.

To enable Windows support for your cluster

1. If you don't have Amazon Linux nodes in your cluster and use security groups for pods, skip to the next step. Otherwise, confirm that the `AmazonEKSVPCResourceController` managed policy is attached to your cluster role (p. 474). Replace *`eksClusterRole`* with your cluster role name.

```
aws iam list-attached-role-policies --role-name eksClusterRole
```

   The example output is as follows.

```
{
    "AttachedPolicies": [
        {
            "PolicyName": "AmazonEKSClusterPolicy",
            "PolicyArn": "arn:aws:iam::aws:policy/AmazonEKSClusterPolicy"
        },
        {
            "PolicyName": "AmazonEKSVPCResourceController",
            "PolicyArn": "arn:aws:iam::aws:policy/AmazonEKSVPCResourceController"
        }
    ]
}
```

   If the policy is attached, as it is in the previous output, skip the next step.

2. Attach the **AmazonEKSVPCResourceController** managed policy to your Amazon EKS cluster IAM role (p. 474). Replace *`eksClusterRole`* with your cluster role name.

```
aws iam attach-role-policy \
  --role-name eksClusterRole \
  --policy-arn arn:aws:iam::aws:policy/AmazonEKSVPCResourceController
```

3. Create a file named *`vpc-resource-controller-configmap.yaml`* with the following contents.

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: amazon-vpc-cni
  namespace: kube-system
data:
  enable-windows-ipam: "true"
```

4. Apply the ConfigMap to your cluster.

```
kubectl apply -f vpc-resource-controller-configmap.yaml
```

Removing legacy Windows support from your data plane

If you enabled Windows support on a cluster that is earlier than a Kubernetes or platform version listed in the Prerequisites (p. 54), then you must first remove the `vpc-resource-controller` and `vpc-admission-webhook` from your data plane. They're deprecated and no longer needed because the functionality that they provided is now enabled on the control plane.

1. Uninstall the `vpc-resource-controller` with the following command. Use this command regardless of which tool you originally installed it with. Replace *region-code* (only the instance of that text after `/manifests/`) with the AWS Region that your cluster is in.

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

2. Uninstall the `vpc-admission-webhook` using the instructions for the tool that you installed it with.

   eksctl

   Run the following commands.

```
kubectl delete deployment -n kube-system vpc-admission-webhook
kubectl delete service -n kube-system vpc-admission-webhook
kubectl delete mutatingwebhookconfigurations.admissionregistration.k8s.io vpc-
admission-webhook-cfg
```

   kubectl on macOS or Windows

   Run the following command. Replace *region-code* (only the instance of that text after `/manifests/`) with the AWS Region that your cluster is in.

```
kubectl delete -f https://s3.us-west-2.amazonaws.com/amazon-eks//manifests/region-
code/vpc-admission-webhook/latest/vpc-admission-webhook-deployment.yaml
```

3. Enable Windows support (p. 55) for your cluster on the control plane.

Disabling Windows support

To disable Windows support on your cluster

1. If your cluster contains Amazon Linux nodes and you use security groups for pods (p. 314) with them, then skip this step.

   Remove the `AmazonVPCResourceController` managed IAM policy from your cluster role (p. 474). Replace *eksClusterRole* with the name of your cluster role and *111122223333* with your account ID.

```
aws iam detach-role-policy \
    --role-name eksClusterRole \
    --policy-arn arn:aws:iam::aws:policy/AmazonEKSVPCResourceController
```

2. Disable Windows IPAM in the `amazon-vpc-cni` ConfigMap.

```
kubectl patch configmap/amazon-vpc-cni \-n kube-system \--type merge \-p '{"data":
{"enable-windows-ipam":"false"}}'
```

Deploying Pods

When you deploy Pods to your cluster, you need to specify the operating system that they use if you're running a mixture of node types.

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
```

You can deploy a sample application (p. 360) to see the node selectors in use.

Enabling legacy Windows support

If your cluster is at, or later, than one of the Kubernetes and platform versions listed in the Prerequisites (p. 54), then we recommend that you enable Windows support on your control plane instead. For more information, see Enabling Windows support (p. 55).

The following steps help you to enable legacy Windows support for your Amazon EKS cluster's data plane if your cluster or platform version are earlier than the versions listed in the Prerequisites (p. 54). Once your cluster and platform version are at, or later than a version listed in the Prerequisites (p. 54), we recommend that you remove legacy Windows support (p. 56) and enable it for your control plane (p. 55).

You can use `eksctl`, a Windows client, or a macOS or Linux client to enable legacy Windows support for your cluster.

eksctl

To enable legacy Windows support for your cluster with `eksctl`

Prerequisite

This procedure requires `eksctl` version `0.104.0` 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. 10).

1. Enable Windows support for your Amazon EKS cluster with the following `eksctl` command. Replace *`my-cluster`* with the name of your cluster. 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 my-cluster --approve
```

> **Important**
> The VPC admission controller webhook is signed with a certificate that expires one year after the date of issue. To avoid down time, make sure to renew the certificate before it expires. For more information, see Renewing the VPC admission webhook certificate (p. 61).

2. After you have enabled Windows support, you can launch a Windows node group into your cluster. For more information, see Launching self-managed Windows nodes (p. 136).

Windows

To enable legacy Windows support for your cluster with a Windows client

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

1. Deploy the VPC resource controller to your cluster.

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

> **Important**
> The VPC admission controller webhook is signed with a certificate that expires one year after the date of issue. To avoid down time, make sure to renew the certificate before it expires. For more information, see Renewing the VPC admission webhook certificate (p. 61).

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 Launching self-managed Windows nodes (p. 136).

macOS and Linux

To enable legacy 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 AWS Region that your cluster resides in.

1. Deploy the VPC resource controller to your cluster.

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

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

```
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
```

Important
The VPC admission controller webhook is signed with a certificate that expires one year after the date of issue. To avoid down time, make sure to renew the certificate before it expires. For more information, see Renewing the VPC admission webhook certificate (p. 61).

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                      ROLE                                  AGE
eks:kube-proxy-windows    ClusterRole/system:node-proxier       19h
```

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 self-managed Windows nodes (p. 136).

Renewing the VPC admission webhook certificate

The certificate used by the VPC admission webhook expires one year after issue. To avoid down time, it's important that you renew the certificate before it expires. You can check the expiration date of your current certificate with the following command.

```
kubectl get secret \
    -n kube-system \
    vpc-admission-webhook-certs -o json | \
    jq -r '.data."cert.pem"' | \
    base64 -decode | \
    openssl x509 \
    -noout \
    -enddate | \
    cut -d= -f2
```

The example output is as follows.

```
May 28 14:23:00 2022 GMT
```

You can renew the certificate using `eksctl` or a Windows or Linux/macOS computer. Follow the instructions for the tool you originally used to install the VPC admission webhook. For example, if you originally installed the VPC admission webhook using `eksctl`, then you should renew the certificate using the instructions on the `eksctl` tab.

eksctl

1. Reinstall the certificate. Replace *<cluster-name>* (including *<>*) with the name of your cluster.

   ```
   eksctl utils install-vpc-controllers -cluster <cluster-name> -approve
   ```

2. Verify that you receive the following output.

   ```
   2021/05/28 05:24:59 [INFO] generate received request
   2021/05/28 05:24:59 [INFO] received CSR
   2021/05/28 05:24:59 [INFO] generating key: rsa-2048
   2021/05/28 05:24:59 [INFO] encoded CSR
   ```

3. Restart the webhook deployment.

   ```
   kubectl rollout restart deployment -n kube-system vpc-admission-webhook
   ```

4. If the certificate that you renewed was expired, and you have Windows pods stuck in the `Container creating` state, then you must delete and redeploy those pods.

Windows

1. Get the script to generate new certificate.

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

2. Prepare parameter for the script.

```
./webhook-create-signed-cert.ps1 -ServiceName vpc-admission-webhook-svc -SecretName
 vpc-admission-webhook-certs -Namespace kube-system
```

3. Restart the webhook deployment.

```
kubectl rollout restart deployment -n kube-system vpc-admission-webhook-deployment
```

4. If the certificate that you renewed was expired, and you have Windows pods stuck in the `Container creating` state, then you must delete and redeploy those pods.

Linux and macOS

Prerequisite

You must have OpenSSL and jq installed on your computer.

1. Get the script to generate new certificate.

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

2. Change the permissions.

```
chmod +x webhook-create-signed-cert.sh
```

3. Run the script.

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

4. Restart the webhook.

```
kubectl rollout restart deployment -n kube-system vpc-admission-webhook-deployment
```

5. If the certificate that you renewed was expired, and you have Windows pods stuck in the `Container creating` state, then you must delete and redeploy those pods.

Private cluster requirements

This topic describes how to deploy an Amazon EKS private cluster without outbound internet access. If you're not familiar with Amazon EKS networking, see De-mystifying cluster networking for Amazon EKS worker nodes.

Requirements

The following requirements must be met to run Amazon EKS in a private cluster without outbound internet access.

- A container image must be in or copied to Amazon Elastic Container Registry (Amazon ECR) or to a registry inside the VPC to be pulled. For more information, see Creating local copies of container images (p. 64).
- Endpoint private access is required for nodes to register with the cluster endpoint. Endpoint public access is optional. For more information, see Amazon EKS cluster endpoint access control (p. 42).
- For Linux and Windows nodes, you must include bootstrap arguments when launching self-managed nodes. This text bypasses the Amazon EKS introspection and doesn't require access to the Amazon EKS API from within the VPC. Replace *api-server-endpoint* and *certificate-authority* with the values from your Amazon EKS cluster.
  - For Linux nodes:

    ```
    --apiserver-endpoint api-server-endpoint --b64-cluster-ca certificate-authority
    ```

    For additional arguments, see the bootstrap script on GitHub.
  - For Windows nodes:

    ```
    -APIServerEndpoint api-server-endpoint -Base64ClusterCA certificate-authority
    ```

    For additional arguments, see Amazon EKS optimized Windows AMI (p. 221).
- The `aws-auth` ConfigMap must be created from within the VPC. For more information about create the `aws-auth` ConfigMap, see Enabling IAM user and role access to your cluster (p. 404).

Considerations

Here are some things to consider when running Amazon EKS in a private cluster without outbound internet access.

- Many AWS services support private clusters, but you must use a VPC endpoint. For more information, see VPC endpoints. Some commonly-used services and endpoints include:

Service	Endpoint
Amazon Elastic Container Registry	`com.amazonaws.region-code.ecr.api` and `com.amazonaws.region-code.ecr.dkr` and the Amazon S3 gateway endpoint
Application Load Balancers and Network Load Balancers	`com.amazonaws.region-code.elasticloadbalancing`
AWS X-Ray	`com.amazonaws.region-code.xray`
Amazon CloudWatch Logs	`com.amazonaws.region-code.logs`
IAM roles for service accounts (p. 444)	`com.amazonaws.region-code.sts`
App Mesh • The App Mesh sidecar injector for Kubernetes is supported. For more information, see App Mesh sidecar injector on GitHub. • The App Mesh controller for Kubernetes isn't supported. For more information, see App Mesh controller on GitHub.	`com.amazonaws.region-code.appmesh-envoy-management`

Service	Endpoint
• Cluster Autoscaler (p. 89) is supported. When deploying Cluster Autoscaler pods, make sure that the command line includes `--aws-use-static-instance-list=true`. For more information, see Use Static Instance List on GitHub. The worker node VPC must also include the STS VPC endpoint and autoscaling VPC endpoint.	

- Before deploying the Amazon EFS CSI driver (p. 242) , the kustomization.yaml file must be changed to set the container images to use the same AWS Region as the Amazon EKS cluster.
- Self-managed and managed nodes (p. 128) are supported. The instances for nodes must have access to the VPC endpoints. If you create a managed node group, the VPC endpoint security group must allow the CIDR for the subnets, or you must add the created node security group to the VPC endpoint security group.
- The Amazon FSx for Lustre CSI driver (p. 254) isn't supported.
- AWS Fargate (p. 149) is supported with private clusters. You can use the AWS Load Balancer Controller (p. 330) to deploy AWS Application Load Balancers (ALBs) and Network Load Balancers with. The controller supports network load balancers with IP targets, which are required for use with Fargate. For more information, see Application load balancing on Amazon EKS (p. 379) and Create a network load balancer (p. 375).
- Installing the AWS Load Balancer Controller add-on (p. 330) is supported. However, while installing, you should use command line flags to set `enable-shield`, `enable-waf`, and `enable-wafv2` to false. In addition, certificate discovery with hostnames from the Ingress objects isn't supported. This is because the controller needs to reach ACM, which doesn't have a VPC endpoint.
- Some container software products use API calls that access the AWS Marketplace Metering service to monitor usage. Private clusters do not allow these calls, so these container types cannot be used for private clusters.

Creating local copies of container images

Because a private cluster has no outbound internet access, container images can't be pulled from external sources such as Docker Hub. Instead, container images must be copied locally to Amazon ECR or to an alternative registry accessible in the VPC. A container image can be copied to Amazon ECR from outside the private VPC. The private cluster accesses the Amazon ECR repository using the Amazon ECR VPC endpoints. You must have Docker and the AWS CLI installed on the workstation that you use to create the local copy.

To create a local copy of a container image

1. Create an Amazon ECR repository. For more information, see Creating a repository.
2. Pull the container image from the external registry using `docker pull`.
3. Tag your image with the Amazon ECR registry, repository, and the optional image tag name combination using `docker tag`.
4. Authenticate to the registry. For more information, see Registry authentication.
5. Push the image to Amazon ECR using `docker push`.

   Note
   Make sure to update your resource configuration to use the new image location.

   The following example pulls the amazon/aws-node-termination-handler image, using tag `v1.3.1-linux-amd64`, from Docker Hub and creates a local copy in Amazon ECR.

```
aws ecr create-repository --repository-name amazon/aws-node-termination-handler
docker pull amazon/aws-node-termination-handler:v1.3.1-linux-amd64
docker tag amazon/aws-node-termination-handler 111122223333.dkr.ecr.region-
code.amazonaws.com/amazon/aws-node-termination-handler:v1.3.1-linux-amd64
aws ecr get-login-password --region region-code | docker login --username AWS --
password-stdin 111122223333.dkr.ecr.region-code.amazonaws.com
docker push 111122223333.dkr.ecr.region-code.amazonaws.com/amazon/aws-node-termination-
handler:v1.3.1-linux-amd64
```

AWS STS endpoints for IAM roles for service accounts

Pods configured with IAM roles for service accounts (p. 444) acquire credentials from an AWS Security Token Service (AWS STS) API call. If there is no outbound internet access, you must create and use an AWS STS VPC endpoint in your VPC. Most AWS `v1` SDKs use the global AWS STS endpoint by default (`sts.amazonaws.com`), which doesn't use the AWS STS VPC endpoint. To use the AWS STS VPC endpoint, you may need to configure the SDK to use the regional AWS STS endpoint (`sts.region-code.amazonaws.com`). You can do this by setting the `AWS_STS_REGIONAL_ENDPOINTS` environment variable with a value of `regional`, along with the AWS Region.

For example, in a pod spec:

```
...
containers:
- env:
- name: AWS_REGION
value: region-code
    - name: AWS_STS_REGIONAL_ENDPOINTS
    value: regional
    ...
    ```
```

Replace *region-code* with the AWS Region that your cluster is in.

# Amazon EKS Kubernetes versions

The Kubernetes project is continually integrating new features, design updates, and bug fixes. The community releases new Kubernetes minor versions, such as `1.22`. New version updates are available on average every three months. Each minor version is supported for approximately twelve months after it's first released.

## Available Amazon EKS Kubernetes versions

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

- `1.22.9`
- `1.21.12`
- `1.20.15`
- `1.19.16`

If your application doesn't require a specific version of Kubernetes, we recommend that you use the latest available Kubernetes version that's 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 instructions on how to update your cluster, see Updating an Amazon
```

EKS cluster Kubernetes version (p. 31). For more information about Kubernetes releases, see Amazon EKS Kubernetes release calendar (p. 72) and Amazon EKS version support and FAQ (p. 72).

> **Note**
> Starting with the Kubernetes version `1.24` launch, officially published Amazon EKS AMIs will include `containerd` as the only runtime. Kubernetes version `1.18–1.23` use Docker as the default runtime. However, these versions have a bootstrap flag option that you can use test out your workloads on any supported cluster with `containerd`. For more information, see Amazon EKS is ending support for `Dockershim` (p. 170).

Kubernetes 1.22

Kubernetes `1.22` is now available in Amazon EKS. For more information about Kubernetes `1.22`, see the official release announcement.

- Kubernetes `1.22` removes a number of APIs that are no longer available. You might need to make changes to your application before you upgrade to Amazon EKS version `1.22`. Follow the Kubernetes version `1.22` prerequisites (p. 35) carefully before updating your cluster.
- **Important**
  `BoundServiceAccountTokenVolume` graduated to stable and enabled by default in Kubernetes version `1.22`. This feature improves security of service account tokens. It allows workloads that are running on Kubernetes to request JSON web tokens that are audience, time, and key bound. Service account tokens now have an expiration of one hour. In previous Kubernetes versions, they didn't have an expiration. This means that clients that rely on these tokens must refresh the tokens within an hour. The following Kubernetes client SDKs refresh tokens automatically within the required time frame:
  - Go version `0.15.7` and later
  - Python version `12.0.0` and later
  - Java version `9.0.0` and later
  - JavaScript version `0.10.3` and later
  - Ruby `master` branch
  - Haskell version `0.3.0.0`
  - C# version `7.0.5` and later

  If your workload is using an older client version, then you must update it. To enable a smooth migration of clients to the newer time-bound service account tokens, Kubernetes version `1.22` adds an extended expiry period to the service account token over the default one hour. For Amazon EKS clusters, the extended expiry period is 90 days. Your Amazon EKS cluster's Kubernetes API server rejects requests with tokens older than 90 days. We recommend that you check your applications and their dependencies to make sure that the Kubernetes client SDKs are the same or later than the versions listed above. For instructions about how to identify pods that are using stale tokens, see Kubernetes service accounts (p. 443).
- The Ingress API versions `extensions/v1beta1` and `networking.k8s.io/v1beta1` have been removed in Kubernetes `1.22`. If you're using the AWS Load Balancer Controller, you must upgrade to at least version `2.4.1` before you upgrade your Amazon EKS clusters to version `1.22`. Additionally, you must modify Ingress manifests to use apiVersion `networking.k8s.io/v1`. This has been available since Kubernetes version `1.19`. For more information about changes between Ingress `v1beta1` and `v1`, see the Kubernetes documentation. The AWS Load Balancer Controller controller sample manifest uses the `v1` spec.
- The Amazon EKS legacy Windows support controllers use the `admissionregistration.k8s.io/v1beta1` API that was removed in Kubernetes `1.22`. If you're running Windows workloads, you must remove legacy Windows support and enable Windows support before upgrading to Amazon EKS version `1.22`.
- The CertificateSigningRequest (CSR) API version `certificates.k8s.io/v1beta1` was removed in Kubernetes version `1.22`. You must migrate manifests and API clients to use the

certificates.k8s.io/v1 CSR API. This API has been available since version 1.19. For instructions on how to use CSR in Amazon EKS, see Certificate signing (p. 441).

- The CustomResourceDefinition API version apiextensions.k8s.io/v1beta1 was removed in Kubernetes 1.22. Make sure that all custom resource definitions in your cluster are updated to v1. API version v1 custom resource definitions are required to have Open API v3 schema validation defined. For more information, see the Kubernetes documentation.
- If you're using App Mesh, you must upgrade to at least App Mesh controller v1.4.3 or later before you upgrade to Amazon EKS version 1.22. Older versions of the App Mesh controller use v1beta1 CustomResourceDefinition API version and aren't compatible with Kubernetes version 1.22 and later.
- Amazon EKS version 1.22 enables the EndpointSliceTerminatingCondition feature by default, which will include pods in terminating state within EndpointSlices. If you set enableEndpointSlices to True (the default is disabled) in the AWS Load Balancer Controller, you must upgraded to at least AWS Load Balancer Controller version 2.4.1+ before upgrading to Amazon EKS versionn 1.22.
- Starting with Amazon EKS version 1.22, kube-proxy is configured by default to expose Prometheus metrics outside the pod. This behavior change addresses the request made in containers roadmap issue #657 .
- The initial launch of Amazon EKS version 1.22 uses etcd version 3.4 as a backend, and is not affected by the possibility of data corruption present in etcd version 3.5.
- Starting with Amazon EKS 1.22, Amazon EKS is decoupling AWS cloud-specific control logic from core control plane code to the out-of-tree AWS Kubernetes Cloud Controller Manager. This is in line with the upstream Kubernetes recommendation. By decoupling the interoperability logic between Kubernetes and the underlying cloud infrastructure, the cloud-controller-manager component enables cloud providers to release features at a different pace compared to the main Kubernetes project. This change is transparent and requires no action. However, a new log stream named cloud-controller-manager now appears under the ControllerManager log type when enabled. For more information, see Amazon EKS control plane logging.
- Starting with Amazon EKS 1.22, Amazon EKS is changing the default AWS Security Token Service endpoint used by IAM roles for service accounts (IRSA) to be the regional endpoint instead of the global endpoint to reduce latency and improve reliability. You can optionally configure IRSA to use the global endpoint in Associate an IAM role to a service account (p. 452).

The following Kubernetes features are now supported in Kubernetes 1.22 Amazon EKS clusters:

- **Server-side Apply graduates to GA** - Server-side Apply helps users and controllers manage their resources through declarative configurations. It allows them to create or modify objects declaratively by sending their fully specified intent. After being in beta for a couple releases, Server-side Apply is now generally available.
- Warning mechanism for deprecated API user - Use of deprecated APIs produces warnings visible to API consumers, and metrics visible to cluster administrators.

For the complete Kubernetes 1.22 changelog, see https://github.com/kubernetes/kubernetes/blob/master/CHANGELOG/CHANGELOG-1.22.md#changelog-since-v1210.

Kubernetes 1.21

Kubernetes 1.21 is now available in Amazon EKS. For more information about Kubernetes 1.21, see the official release announcement.

- **Important**
  BoundServiceAccountTokenVolume graduated to beta and is enabled by default in Kubernetes version 1.21. This feature improves security of service account tokens by allowing workloads running on Kubernetes to request JSON web tokens that are audience, time,

and key bound. Service account tokens now have an expiration of one hour. In previous Kubernetes versions, they didn't have an expiration. This means that clients that rely on these tokens must refresh the tokens within an hour. The following Kubernetes client SDKs refresh tokens automatically within the required time frame:

  - Go version `0.15.7` and later
  - Python version `12.0.0` and later
  - Java version `9.0.0` and later
  - JavaScript version `0.10.3` and later
  - Ruby `master` branch
  - Haskell version `0.3.0.0`
  - C# version `7.0.5` and later

  If your workload is using an older client version, then you must update it. To enable a smooth migration of clients to the newer time-bound service account tokens, Kubernetes version `1.21` adds an extended expiry period to the service account token over the default one hour. For Amazon EKS clusters, the extended expiry period is 90 days. Your Amazon EKS cluster's Kubernetes API server rejects requests with tokens older than 90 days. We recommend that you check your applications and their dependencies to make sure that the Kubernetes client SDKs are the same or later than the versions listed above. For instructions about how to identify pods that are using stale tokens, see Kubernetes service accounts (p. 443).

- Dual-stack networking support (`IPv4` and `IPv6` addresses) on pods, services, and nodes reached beta status. However, Amazon EKS and the Amazon VPC CNI plugin for Kubernetes don't currently support dual stack networking.
- The Amazon EKS Optimized Amazon Linux 2 AMI now contains a bootstrap flag to enable the `containerd` runtime as a Docker alternative. This flag allows preparation for the removal of Docker as a supported runtime in the next Kubernetes release. For more information, see Enable the `containerd` runtime bootstrap flag (p. 179). This can be tracked through the container roadmap on Github.
- Managed node groups support for Cluster Autoscaler priority expander.

  Newly created managed node groups on Amazon EKS version `1.21` clusters use the following format for the underlying Auto Scaling group name:

  ```
  eks-<managed-node-group-name>-<uuid>
  ```

  This enables using the priority expander feature of Cluster Autoscaler to scale node groups based on user defined priorities. A common use case is to prefer scaling spot node groups over on-demand groups. This behavior change solves the containers roadmap issue #1304.

The following Kubernetes features are now supported in Amazon EKS `1.21` clusters:

- CronJobs (previously ScheduledJobs) have now graduated to stable status. With this change, users perform regularly scheduled actions such as backups and report generation.
- Immutable Secrets and ConfigMaps have now graduated to stable status. A new, immutable field was added to these objects to reject changes. This rejection protects the cluster from updates that can unintentionally break the applications. Because these resources are immutable, `kubelet` doesn't watch or poll for changes. This reduces `kube-apiserver` load and improving scalability and performance.
- Graceful Node Shutdown has now graduated to beta status. With this update, the `kubelet` is aware of node shutdown and can gracefully terminate that node's pods. Before this update, when a node shutdown, its pods didn't follow the expected termination lifecycle. This caused workload problems. Now, the `kubelet` can detect imminent system shutdown through `systemd`, and inform running pods so they terminate gracefully.
- Pods with multiple containers can now use the `kubectl.kubernetes.io/default-container` annotation to have a container preselected for `kubectl` commands.

- PodSecurityPolicy is being phased out. PodSecurityPolicy will still be functional for several more releases according to Kubernetes deprecation guidelines. For more information, see PodSecurityPolicy Deprecation: Past, Present, and Future and the AWS blog.

For the complete Kubernetes `1.21` changelog, see https://github.com/kubernetes/kubernetes/blob/master/CHANGELOG/CHANGELOG-1.21.md.

Kubernetes 1.20

For more information about Kubernetes `1.20`, see the official release announcement.

- `1.20` brings new default roles and users. You can find more information in Default EKS Kubernetes roles and users. Ensure that you are using a supported cert-manager version.

The following Kubernetes features are now supported in Kubernetes `1.20` Amazon EKS clusters:

- API Priority and Fairness has reached beta status and is enabled by default. This allows `kube-apiserver` to categorize incoming requests by priority levels.
- RuntimeClass has reached stable status. The `RuntimeClass` resource provides a mechanism for supporting multiple runtimes in a cluster and surfaces information about that container runtime to the control plane.
- Process ID Limits has now graduated to general availability.
- kubectl debug has reached beta status. `kubectl debug` provides support for common debugging workflows directly from `kubectl`.
- The Docker container runtime has been phased out. The Kubernetes community has written a blog post about this in detail with a dedicated FAQ page. Docker-produced images can continue to be used and will work as they always have. You can safely ignore the `Dockershim` deprecation warning message printed in `kubelet` startup logs. Amazon EKS will eventually move to `containerd` as the runtime for the Amazon EKS optimized Amazon Linux 2 AMI. You can follow the containers roadmap issue for more details.
- Pod Hostname as FQDN has graduated to beta status. This feature allows setting a pod's hostname to its Fully Qualified Domain Name (FQDN), giving the ability to set the hostname field of the kernel to the FQDN of a pod.
- The client-go credential plugins can now be passed in the current cluster information via the `KUBERNETES_EXEC_INFO` environment variable. This enhancement allows Go clients to authenticate using external credential providers, such as a key management system (KMS).

For the complete Kubernetes `1.20` changelog, see https://github.com/kubernetes/kubernetes/blob/master/CHANGELOG/CHANGELOG-1.20.md.

Kubernetes 1.19

For more information about Kubernetes `1.19`, see the official release announcement.

- Starting with `1.19`, Amazon EKS no longer adds the `kubernetes.io/cluster/`*`cluster-name`* tag to subnets passed in when clusters are created. This subnet tag is only required if you want to influence where the Kubernetes service controller or AWS Load Balancer Controller places Elastic Load Balancers. For more information about the requirements of subnets passed to Amazon EKS during cluster creation, see updates to the section called "VPC and subnet requirements" (p. 260).
  - Subnet tags aren't modified on existing clusters updated to `1.19`.
  - The AWS Load Balancer Controller version `2.1.1` and earlier required the *`cluster-name`* subnet tag. In version `2.1.2` and later, you can specify the tag to refine subnet discovery, but it's not

required. For more information about the AWS Load Balancer Controller, see Installing the AWS Load Balancer Controller add-on (p. 330). For more information about subnet tagging when using a load balancer, see Application load balancing on Amazon EKS (p. 379) and Network load balancing on Amazon EKS (p. 373).

- You're no longer required to provide a security context for non-root containers that must access the web identity token file for use with IAM roles for service accounts. For more information, see IAM roles for service accounts (p. 444) andproposal for file permission handling in projected service account volume on GitHub.
- The pod identity webhook has been updated to address the missing startup probes GitHub issue. The webhook also now supports an annotation to control token expiration. For more information, see the GitHub pull request.
- CoreDNS version `1.8.0` is the recommended version for Amazon EKS `1.19` clusters. This version is installed by default in new Amazon EKS `1.19` clusters. For more information, see Managing the CoreDNS add-on (p. 338).
- Amazon EKS optimized Amazon Linux 2 AMIs include the Linux kernel version `5.4` for Kubernetes version `1.19`. For more information, see Amazon EKS optimized Amazon Linux AMI (p. 183).
- The `CertificateSigningRequest API` has been promoted to stable `certificates.k8s.io/v1` with the following changes:
  - `spec.signerName` is now required. You can't create requests for `kubernetes.io/legacy-unknown` with the `certificates.k8s.io/v1` API.
  - You can continue to create CSRs with the `kubernetes.io/legacy-unknown` signer name with the `certificates.k8s.io/v1beta1` API.
  - You can continue to request that a CSR to is signed for a non-node server cert, webhooks (for example, with the `certificates.k8s.io/v1beta1` API). These CSRs aren't auto-approved.
  - To approve certificates, a privileged user requires `kubectl` `1.18.8` or later.

  For more information about the certificate `v1` API, see Certificate Signing Requests in the Kubernetes documentation.

The following Amazon EKS Kubernetes resources are critical for the Kubernetes control plane to work. We recommend that you don't delete or edit them.

Permission	Kind	Namespace	Reason
`eks:certificate-controller`	`Rolebinding`	`kube-system`	Impacts signer and approver functionality in the control plane.
`eks:certificate-controller`	`Role`	`kube-system`	Impacts signer and approver functionality in the control plane.
`eks:certificate-controller`	`ClusterRolebinding`	All	Impacts `kubelet`'s ability to request server certificates, which affects certain cluster functionality like `kubectl exec` and `kubectl logs`.

The following Kubernetes features are now supported in Kubernetes `1.19` Amazon EKS clusters:

- The `ExtendedResourceToleration` admission controller is enabled. This admission controller automatically adds tolerations for taints to pods requesting extended resources, such as

GPUs. This way, you don't have to manually add the tolerations. For more information, see ExtendedResourceToleration in the Kubernetes documentation.

- Elastic Load Balancers (CLB and NLB) provisioned by the in-tree Kubernetes service controller support filtering the nodes included as instance targets. This can help prevent reaching target group limits in large clusters. For more information, see the related GitHub issue and the `service.beta.kubernetes.io/aws-load-balancer-target-node-labels` annotation under Other ELB annotations in the Kubernetes documentation.
- Pod Topology Spread has reached stable status. You can use topology spread constraints to control how pods are spread across your cluster among failure-domains such as regions, zones, nodes, and other user-defined topology domains. This can help to achieve high availability, as well as efficient resource utilization. For more information, see Pod Topology Spread Constraints in the Kubernetes documentation.
- The Ingress API has reached general availability. For more information, see Ingress in the Kubernetes documentation.
- EndpointSlices are enabled by default. EndpointSlices are a new API that provides a more scalable and extensible alternative to the Endpoints API for tracking IP addresses, ports, readiness, and topology information for Pods backing a Service. For more information, see Scaling Kubernetes Networking With EndpointSlices in the Kubernetes blog.
- Secret and ConfigMap volumes can now be marked as immutable. This significantly reduces load on the API server if there are many Secret and ConfigMap volumes in the cluster. For more information, see ConfigMap and Secret in the Kubernetes documentation.

For the complete Kubernetes `1.19` changelog, see https://github.com/kubernetes/kubernetes/blob/master/CHANGELOG/CHANGELOG-1.19.md.

Kubernetes 1.18

For more information about Kubernetes `1.18`, see the official release announcement.

The following Kubernetes features are now supported in Kubernetes `1.18` Amazon EKS clusters:

- Topology Manager has reached beta status. This feature allows the CPU and Device Manager to coordinate resource allocation decisions, optimizing for low latency with machine learning and analytics workloads. For more information, see Control Topology Management Policies on a node in the Kubernetes documentation.
- Server-side Apply is updated with a new beta version. This feature tracks and manages changes to fields of all new Kubernetes objects. This helps you to know what changed your resources and when. For more information, see What is Server-side Apply? in the Kubernetes documentation.
- A new `pathType` field and a new `IngressClass` resource has been added to the Ingress specification. These features make it simpler to customize Ingress configuration, and are supported by the AWS Load Balancer Controller (p. 379) (formerly called the ALB Ingress Controller). For more information, see Improvements to the Ingress API in Kubernetes`1.18` in the Kubernetes documentation.
- Configurable horizontal pod autoscaling behavior. For more information, see Support for configurable scaling behavior in the Kubernetes documentation.
- In `1.18` clusters, you no longer need to include the `AWS_DEFAULT_REGION=`*`region-code`* environment variable to pods when using IAM roles for service accounts in China Regions, whether you use the mutating web hook or configure the environment variables manually. You still need to include the variable for all pods in earlier versions.
- New clusters contain updated default values in `externalTrafficPolicy`. `HealthyThresholdCount` and `UnhealthyThresholdCount` are 2 each, and `HealthCheckIntervalSeconds` is reduced to 10 seconds. Clusters created in older versions and upgraded retain the old values.

For the complete Kubernetes `1.18` changelog, see https://github.com/kubernetes/kubernetes/blob/master/CHANGELOG/CHANGELOG-1.18.md.

Amazon EKS Kubernetes release calendar

Note
Dates with only a month and a year are approximate and are updated with an exact date when it's known.

Kubernetes version	Upstream release	Amazon EKS release	Amazon EKS end of support
`1.18`	March 23, 2020	October 13, 2020	March 31, 2022
`1.19`	August 26, 2020	February 16, 2021	August 1, 2022
`1.20`	December 8, 2020	May 18, 2021	October 3, 2022
`1.21`	April 8, 2021	July 19, 2021	February 2023
`1.22`	August 4, 2021	April 4, 2022	May 2023
`1.23`	December 7, 2021	August 2022	October 2023

Amazon EKS version support and FAQ

In line with the Kubernetes community support for Kubernetes versions, Amazon EKS is committed to supporting at least four production-ready versions of Kubernetes at any given time. We will announce the end of support date 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 end of support date of a Kubernetes version on Amazon EKS will be on or after the date that the Kubernetes project stops supporting the version upstream.

Frequently asked questions

Q: How long is a Kubernetes version supported by Amazon EKS?

A: A Kubernetes version is supported for 14 months after first being available on Amazon EKS. This is true even if upstream Kubernetes no longer support a version that's available on Amazon EKS. We backport security patches that are applicable to the Kubernetes versions that are supported on Amazon EKS.

Q: Am I notified when support is ending for a Kubernetes version on Amazon EKS?

A: Yes, if any clusters in your account are running the version nearing the end of support, Amazon EKS sends out a notice through the AWS Health Dashboard approximately 12 months after the Kubernetes version was released on Amazon EKS. The notice includes the end of support date. This is at least 60 days from the date of the notice.

Q: What happens on the end of support date?

A: On the end of support date, you can no longer create new Amazon EKS clusters with the unsupported version. Existing control planes are automatically updated by Amazon EKS to the earliest supported version through a gradual deployment process after the end of support date. After the automatic control plane update, make sure to manually update cluster add-ons and Amazon EC2 nodes. For more information, see the section called “Update the Kubernetes version for your Amazon EKS cluster ” (p. 32).

Q: When exactly is my control plane automatically updated after the end of support date?

A: Amazon EKS can't provide specific timeframes. Automatic updates can happen at any time after the end of support date. We recommend that you proactively update your control plane without relying on the Amazon EKS automatic update process. For more information, see the section called “Updating Kubernetes version” (p. 31).

Q: Can I leave my control plane on a Kubernetes version indefinitely?

A: No, cloud security at AWS is the highest priority. Past a certain point (usually one year), the Kubernetes community stops releasing CVE patches and discourages CVE submission for deprecated versions. This means that vulnerabilities specific to an older version of Kubernetes might not even be reported. This leaves clusters exposed with no notice and no remediation options in the event of a vulnerability. Given this, Amazon EKS doesn't allow control planes to stay on a version that reached end of support.

Q: Which Kubernetes features are supported by Amazon EKS?

A: Amazon EKS supports all general availability features of the Kubernetes API. It also supports all beta features, which are enabled by default. Alpha features aren't supported.

Q: Are Amazon EKS managed node groups automatically updated along with the cluster control plane version?

A: No, a managed node group creates Amazon EC2 instances in your account. These instances aren't automatically upgraded when you or Amazon EKS update your control plane. Assume that Amazon EKS automatically updates your control plane. The Kubernetes version that's on your managed node group might be more than one version earlier than your control plane. Then, assume that a managed node group contains instances that are running a version of Kubernetes that's more than one version earlier than the control plane. The node group has a health issue in the **Node Groups** section of the **Compute** tab of your cluster in the console. Last, if a node group has an available version update, **Update now** appears next to the node group in the console. For more information, see the section called “Updating a managed node group” (p. 115). We recommend maintaining the same Kubernetes version on your control plane and nodes.

Q: Are self-managed node groups automatically updated along with the cluster control plane version?

A: No, a self-managed node group includes Amazon EC2 instances in your account. These instances aren't automatically upgraded when you or Amazon EKS update the control plane version on your behalf. A self-managed node group doesn't have any indication in the console that it needs updating. You can view the `kubelet` version installed on a node by selecting the node in the **Nodes** list on the **Overview** tab of your cluster to determine which nodes need updating. You must manually update the nodes. For more information, see the section called “Updates” (p. 141).

The Kubernetes project tests compatibility between the control plane and nodes for up to two minor versions. For example, `1.20` nodes continue to operate when orchestrated by a `1.22` control plane. However, running a cluster with nodes that are persistently two minor versions behind the control plane isn't recommended. For more information, see Kubernetes version and version skew support policy in the Kubernetes documentation. We recommend maintaining the same Kubernetes version on your control plane and nodes.

Q: Are pods running on Fargate automatically upgraded with an automatic cluster control plane version upgrade?

Yes, Fargate pods run on infrastructure in AWS owned accounts on the Amazon EKS side of the shared responsibility model (p. 440). Amazon EKS uses the Kubernetes eviction API to attempt to gracefully drain pods that are running on Fargate. For more information, see The Eviction API in the Kubernetes documentation. If a pod can’t be evicted, Amazon EKS issues a Kubernetes `delete pod` command. We strongly recommend running Fargate pods as part of a replication controller such as a Kubernetes deployment. This is so a pod is automatically rescheduled after deletion. For more information, see

Deployments in the Kubernetes documentation. The new version of the Fargate pod is deployed with a `kubelet` version that's the same version as your updated cluster control plane version.

> **Important**
> If you update the control plane, you still need to update the Fargate nodes yourself. To update Fargate nodes, delete the Fargate pod represented by the node and redeploy the pod. The new pod is deployed with a `kubelet` version that's the same version as your cluster.

Amazon EKS platform versions

Amazon EKS platform versions represent the capabilities of the Amazon EKS 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 1.22, 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. 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.
- Amazon EKS might publish a new node AMI with a corresponding patch version. However, all patch versions are compatible between the EKS control plane and node AMIs for a given Kubernetes minor version.

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

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 `1.22`

Kubernetes version	Amazon EKS platform version	Enabled admission controllers	Release notes	Release date
`1.22.9`	`eks.2`	`NamespaceLifecycle`, `LimitRanger`, `ServiceAccount`, `DefaultStorageClass`, `ResourceQuota`, `DefaultTolerationSeconds`, `NodeRestriction`, `MutatingAdmissionWebhook`, `ValidatingAdmissionWebhook`, `PodSecurityPolicy`,	New platform version with security fixes and enhancements.	May 31, 2022

Kubernetes version	Amazon EKS platform version	Enabled admission controllers	Release notes	Release date
		TaintNodesByCondition, StorageObjectInUseProtection, PersistentVolumeClaimResize, ExtendedResourceToleration, CertificateApproval, PodPriority, CertificateSigning, CertificateSubjectRestriction, RuntimeClass, DefaultIngressClass		
1.22.6	eks.1	NamespaceLifecycle, LimitRanger, ServiceAccount, DefaultStorageClass, ResourceQuota, DefaultTolerationSeconds, NodeRestriction, MutatingAdmissionWebhook, ValidatingAdmissionWebhook, PodSecurityPolicy, TaintNodesByCondition, StorageObjectInUseProtection, PersistentVolumeClaimResize, ExtendedResourceToleration, CertificateApproval, PodPriority, CertificateSigning, CertificateSubjectRestriction, RuntimeClass, DefaultIngressClass	Initial release of Kubernetes version 1.22 for Amazon EKS. For more information, see Kubernetes 1.22 (p. 66).	April 4, 2022

Kubernetes version 1.21

Kubernetes version	Amazon EKS platform version	Enabled admission controllers	Release notes	Release date
1.21.12	eks.7	NamespaceLifecycle, LimitRanger, ServiceAccount, DefaultStorageClass, ResourceQuota, DefaultTolerationSeconds, NodeRestriction, MutatingAdmissionWebhook, ValidatingAdmissionWebhook, PodSecurityPolicy, TaintNodesByCondition, StorageObjectInUseProtection, PersistentVolumeClaimResize,	New platform version with security fixes and enhancements.	May 31, 2022

Kubernetes version	Amazon EKS platform version	Enabled admission controllers	Release notes	Release date
		ExtendedResourceToleration, CertificateApproval, PodPriority, CertificateSigning, CertificateSubjectRestriction, RuntimeClass, DefaultIngressClass		
1.21.9	eks.6	NamespaceLifecycle, LimitRanger, ServiceAccount, DefaultStorageClass, ResourceQuota, DefaultTolerationSeconds, NodeRestriction, MutatingAdmissionWebhook, ValidatingAdmissionWebhook, PodSecurityPolicy, TaintNodesByCondition, StorageObjectInUseProtection, PersistentVolumeClaimResize, ExtendedResourceToleration, CertificateApproval, PodPriority, CertificateSigning, CertificateSubjectRestriction, RuntimeClass, DefaultIngressClass	The AWS Security Token Service endpoint is reverted back to the global endpoint from the previous platform version. If you want to use the Regional endpoint when using IAM roles for service accounts, then you have to enable it. For instructions on how to enable the regional endpoint, see Configure the AWS Security Token Service endpoint for a service account (p. 454).	April 8, 2022
1.21.5	eks.5	NamespaceLifecycle, LimitRanger, ServiceAccount, DefaultStorageClass, ResourceQuota, DefaultTolerationSeconds, NodeRestriction, MutatingAdmissionWebhook, ValidatingAdmissionWebhook, PodSecurityPolicy, TaintNodesByCondition, StorageObjectInUseProtection, PersistentVolumeClaimResize, ExtendedResourceToleration, CertificateApproval, PodPriority, CertificateSigning, CertificateSubjectRestriction, RuntimeClass, DefaultIngressClass	When using IAM roles for service accounts (p. 444), the AWS Security Token Service Regional endpoint is now used by default instead of the global endpoint. This change is reverted back to the global endpoint in eks.6 however. An updated Fargate scheduler provisions nodes at a significantly higher rate during large deployments.	March 10, 2022

Kubernetes version	Amazon EKS platform version	Enabled admission controllers	Release notes	Release date
`1.21.5`	`eks.4`	`NamespaceLifecycle`, `LimitRanger`, `ServiceAccount`, `DefaultStorageClass`, `ResourceQuota`, `DefaultTolerationSeconds`, `NodeRestriction`, `MutatingAdmissionWebhook`, `ValidatingAdmissionWebhook`, `PodSecurityPolicy`, `TaintNodesByCondition`, `StorageObjectInUseProtection`, `PersistentVolumeClaimResize`, `ExtendedResourceToleration`, `CertificateApproval`, `PodPriority`, `CertificateSigning`, `CertificateSubjectRestriction`, `RuntimeClass`, `DefaultIngressClass`	Version `1.10.1-eksbuild.1` of the Amazon VPC CNI self-managed and Amazon EKS add-on is now the default version deployed.	
`1.21.2`	`eks.3`	`NamespaceLifecycle`, `LimitRanger`, `ServiceAccount`, `DefaultStorageClass`, `ResourceQuota`, `DefaultTolerationSeconds`, `NodeRestriction`, `MutatingAdmissionWebhook`, `ValidatingAdmissionWebhook`, `PodSecurityPolicy`, `TaintNodesByCondition`, `StorageObjectInUseProtection`, `PersistentVolumeClaimResize`, `ExtendedResourceToleration`, `CertificateApproval`, `PodPriority`, `CertificateSigning`, `CertificateSubjectRestriction`, `RuntimeClass`, `DefaultIngressClass`	New platform version with support for Windows `IPv4` address management on the VPC Resource Controller running on the Kubernetes control plane. Added the Kubernetes filter directive for Fargate Fluent Bit logging.	
`1.21.2`	`eks.2`	`NamespaceLifecycle`, `LimitRanger`, `ServiceAccount`, `DefaultStorageClass`, `ResourceQuota`, `DefaultTolerationSeconds`, `NodeRestriction`, `MutatingAdmissionWebhook`, `ValidatingAdmissionWebhook`, `PodSecurityPolicy`, `TaintNodesByCondition`, `Priority`, `StorageObjectInUseProtection`, `PersistentVolumeClaimResize`, `ExtendedResourceToleration`	New platform version with security fixes and enhancements.	

Kubernetes version	Amazon EKS platform version	Enabled admission controllers	Release notes	Release date
1.21.2	eks.1	NamespaceLifecycle, LimitRanger, ServiceAccount, DefaultStorageClass, ResourceQuota, DefaultTolerationSeconds, NodeRestriction, MutatingAdmissionWebhook, ValidatingAdmissionWebhook, PodSecurityPolicy, TaintNodesByCondition, Priority, StorageObjectInUseProtection, PersistentVolumeClaimResize, ExtendedResourceToleration	Initial release of Kubernetes version 1.21 for Amazon EKS. For more information, see Kubernetes 1.21 (p. 67).	

Kubernetes version 1.20

Kubernetes version	Amazon EKS platform version	Enabled admission controllers	Release notes	Release date
1.20.15	eks.6	NamespaceLifecycle, LimitRanger, ServiceAccount, DefaultStorageClass, ResourceQuota, DefaultTolerationSeconds, NodeRestriction, MutatingAdmissionWebhook, ValidatingAdmissionWebhook, PodSecurityPolicy, TaintNodesByCondition, StorageObjectInUseProtection, PersistentVolumeClaimResize, ExtendedResourceToleration, CertificateApproval, PodPriority, CertificateSigning, CertificateSubjectRestriction, RuntimeClass, DefaultIngressClass	New platform version with security fixes and enhancements.	May 31, 2022
1.20.15	eks.5	NamespaceLifecycle, LimitRanger, ServiceAccount, DefaultStorageClass, ResourceQuota, DefaultTolerationSeconds, NodeRestriction, MutatingAdmissionWebhook,	The AWS Security Token Service endpoint is reverted back to the global endpoint from the previous platform version. If you want to use the Regional endpoint when using IAM roles for service accounts, then you have to enable it. For	April 8, 2022

Kubernetes version	Amazon EKS platform version	Enabled admission controllers	Release notes	Release date
		ValidatingAdmissionWebhook, PodSecurityPolicy, TaintNodesByCondition, StorageObjectInUseProtection, PersistentVolumeClaimResize, ExtendedResourceToleration, CertificateApproval, PodPriority, CertificateSigning, CertificateSubjectRestriction, RuntimeClass, DefaultIngressClass	instructions on how to enable the regional endpoint, see Configure the AWS Security Token Service endpoint for a service account (p. 454).	
1.20.11	eks.4	NamespaceLifecycle, LimitRanger, ServiceAccount, DefaultStorageClass, ResourceQuota, DefaultTolerationSeconds, NodeRestriction, MutatingAdmissionWebhook, ValidatingAdmissionWebhook, PodSecurityPolicy, TaintNodesByCondition, StorageObjectInUseProtection, PersistentVolumeClaimResize, ExtendedResourceToleration, CertificateApproval, PodPriority, CertificateSigning, CertificateSubjectRestriction, RuntimeClass, DefaultIngressClass	When using IAM roles for service accounts (p. 444), the AWS Security Token Service Regional endpoint is now used by default instead of the global endpoint. This change is reverted back to the global endpoint in eks.5 however. An updated Fargate scheduler provisions nodes at a significantly higher rate during large deployments.	March 10, 2022

Kubernetes version	Amazon EKS platform version	Enabled admission controllers	Release notes	Release date
1.20.11	eks.3	NamespaceLifecycle, LimitRanger, ServiceAccount, DefaultStorageClass, ResourceQuota, DefaultTolerationSeconds, NodeRestriction, MutatingAdmissionWebhook, ValidatingAdmissionWebhook, PodSecurityPolicy, TaintNodesByCondition, StorageObjectInUseProtection, PersistentVolumeClaimResize, ExtendedResourceToleration, CertificateApproval, PodPriority, CertificateSigning, CertificateSubjectRestriction, RuntimeClass, DefaultIngressClass	New platform version with support for Windows IPv4 address management on the VPC Resource Controller running on the Kubernetes control plane. Added the Kubernetes filter directive for Fargate Fluent Bit logging.	
1.20.7	eks.2	NamespaceLifecycle, LimitRanger, ServiceAccount, DefaultStorageClass, ResourceQuota, DefaultTolerationSeconds, NodeRestriction, MutatingAdmissionWebhook, ValidatingAdmissionWebhook, PodSecurityPolicy, TaintNodesByCondition, Priority, StorageObjectInUseProtection, PersistentVolumeClaimResize, ExtendedResourceToleration	New platform version with security fixes and enhancements.	
1.20.4	eks.1	NamespaceLifecycle, LimitRanger, ServiceAccount, DefaultStorageClass, ResourceQuota, DefaultTolerationSeconds, NodeRestriction, MutatingAdmissionWebhook, ValidatingAdmissionWebhook, PodSecurityPolicy, TaintNodesByCondition, Priority, StorageObjectInUseProtection, PersistentVolumeClaimResize, ExtendedResourceToleration	Initial release of Kubernetes version 1.20 for Amazon EKS. For more information, see Kubernetes 1.20 (p. 69).	

Kubernetes version 1.19

Kubernetes version	Amazon EKS platform version	Enabled admission controllers	Release notes	Release date
1.19.16	eks.10	NamespaceLifecycle, LimitRanger, ServiceAccount, DefaultStorageClass, ResourceQuota, DefaultTolerationSeconds, NodeRestriction, MutatingAdmissionWebhook, ValidatingAdmissionWebhook, PodSecurityPolicy, TaintNodesByCondition, StorageObjectInUseProtection, PersistentVolumeClaimResize, ExtendedResourceToleration, CertificateApproval, PodPriority, CertificateSigning, CertificateSubjectRestriction, RuntimeClass, DefaultIngressClass	New platform version with security fixes and enhancements.	May 31, 2022
1.19.16	eks.9	NamespaceLifecycle, LimitRanger, ServiceAccount, DefaultStorageClass, ResourceQuota, DefaultTolerationSeconds, NodeRestriction, MutatingAdmissionWebhook, ValidatingAdmissionWebhook, PodSecurityPolicy, TaintNodesByCondition, StorageObjectInUseProtection, PersistentVolumeClaimResize, ExtendedResourceToleration, CertificateApproval, PodPriority, CertificateSigning, CertificateSubjectRestriction, RuntimeClass, DefaultIngressClass	The AWS Security Token Service endpoint is reverted back to the global endpoint from the previous platform version. If you want to use the Regional endpoint when using IAM roles for service accounts, then you have to enable it. For instructions on how to enable the regional endpoint, see Configure the AWS Security Token Service endpoint for a service account (p. 454).	April 8, 2022
1.19.15	eks.8	NamespaceLifecycle, LimitRanger, ServiceAccount, DefaultStorageClass, ResourceQuota, DefaultTolerationSeconds, NodeRestriction,	When using IAM roles for service accounts (p. 444), the AWS Security Token Service Regional endpoint is now used by default instead of the global endpoint. This change is reverted back to	March 10, 2022

Kubernetes version	Amazon EKS platform version	Enabled admission controllers	Release notes	Release date
		MutatingAdmissionWebhook, ValidatingAdmissionWebhook, PodSecurityPolicy, TaintNodesByCondition, StorageObjectInUseProtection, PersistentVolumeClaimResize, ExtendedResourceToleration, CertificateApproval, PodPriority, CertificateSigning, CertificateSubjectRestriction, RuntimeClass, DefaultIngressClass	the global endpoint in eks.9 however. An updated Fargate scheduler provisions nodes at a significantly higher rate during large deployments.	
1.19.15	eks.7	NamespaceLifecycle, LimitRanger, ServiceAccount, DefaultStorageClass, ResourceQuota, DefaultTolerationSeconds, NodeRestriction, MutatingAdmissionWebhook, ValidatingAdmissionWebhook, PodSecurityPolicy, TaintNodesByCondition, StorageObjectInUseProtection, PersistentVolumeClaimResize, ExtendedResourceToleration, CertificateApproval, PodPriority, CertificateSigning, CertificateSubjectRestriction, RuntimeClass, DefaultIngressClass	New platform version with support for Windows IPv4 address management on the VPC Resource Controller running on the Kubernetes control plane. Added the Kubernetes filter directive for Fargate Fluent Bit logging.	
1.19.8	eks.6	NamespaceLifecycle, LimitRanger, ServiceAccount, DefaultStorageClass, ResourceQuota, DefaultTolerationSeconds, NodeRestriction, MutatingAdmissionWebhook, ValidatingAdmissionWebhook, PodSecurityPolicy, TaintNodesByCondition, Priority, StorageObjectInUseProtection, PersistentVolumeClaimResize, ExtendedResourceToleration	New platform version with security fixes and enhancements.	

Kubernetes version	Amazon EKS platform version	Enabled admission controllers	Release notes	Release date
1.19.8	eks.5	NamespaceLifecycle, LimitRanger, ServiceAccount, DefaultStorageClass, ResourceQuota, DefaultTolerationSeconds, NodeRestriction, MutatingAdmissionWebhook, ValidatingAdmissionWebhook, PodSecurityPolicy, TaintNodesByCondition, Priority, StorageObjectInUseProtection, PersistentVolumeClaimResize, ExtendedResourceToleration	New platform version that supports custom security groups with Fargate.	
1.19.8	eks.4	NamespaceLifecycle, LimitRanger, ServiceAccount, DefaultStorageClass, ResourceQuota, DefaultTolerationSeconds, NodeRestriction, MutatingAdmissionWebhook, ValidatingAdmissionWebhook, PodSecurityPolicy, TaintNodesByCondition, Priority, StorageObjectInUseProtection, PersistentVolumeClaimResize, ExtendedResourceToleration	New platform version with security fixes and enhancements.	
1.19.8	eks.3	NamespaceLifecycle, LimitRanger, ServiceAccount, DefaultStorageClass, ResourceQuota, DefaultTolerationSeconds, NodeRestriction, MutatingAdmissionWebhook, ValidatingAdmissionWebhook, PodSecurityPolicy, TaintNodesByCondition, Priority, StorageObjectInUseProtection, PersistentVolumeClaimResize, ExtendedResourceToleration	New platform version with security fixes and enhancements.	

Kubernetes version	Amazon EKS platform version	Enabled admission controllers	Release notes	Release date
1.19.6	eks.2	NamespaceLifecycle, LimitRanger, ServiceAccount, DefaultStorageClass, ResourceQuota, DefaultTolerationSeconds, NodeRestriction, MutatingAdmissionWebhook, ValidatingAdmissionWebhook, PodSecurityPolicy, TaintNodesByCondition, Priority, StorageObjectInUseProtection, PersistentVolumeClaimResize, ExtendedResourceToleration	New platform version with security fixes and enhancements.	
1.19.6	eks.1	NamespaceLifecycle, LimitRanger, ServiceAccount, DefaultStorageClass, ResourceQuota, DefaultTolerationSeconds, NodeRestriction, MutatingAdmissionWebhook, ValidatingAdmissionWebhook, PodSecurityPolicy, TaintNodesByCondition, Priority, StorageObjectInUseProtection, PersistentVolumeClaimResize, ExtendedResourceToleration	Initial release of Kubernetes version 1.19 for Amazon EKS. For more information, see Kubernetes 1.19 (p. 69).	

Kubernetes version 1.18

Kubernetes version	Amazon EKS platform version	Enabled admission controllers	Release notes	Release date
1.18.20	eks.12	NamespaceLifecycle, LimitRanger, ServiceAccount, DefaultStorageClass, ResourceQuota, DefaultTolerationSeconds, NodeRestriction, MutatingAdmissionWebhook, ValidatingAdmissionWebhook, PodSecurityPolicy, TaintNodesByCondition,	New platform version with security fixes and enhancements.	May 31, 2022

Kubernetes version	Amazon EKS platform version	Enabled admission controllers	Release notes	Release date
		StorageObjectInUseProtection, PersistentVolumeClaimResize, ExtendedResourceToleration, CertificateApproval, PodPriority, CertificateSigning, CertificateSubjectRestriction, RuntimeClass, DefaultIngressClass		
1.18.20	eks.11	NamespaceLifecycle, LimitRanger, ServiceAccount, DefaultStorageClass, ResourceQuota, DefaultTolerationSeconds, NodeRestriction, MutatingAdmissionWebhook, ValidatingAdmissionWebhook, PodSecurityPolicy, TaintNodesByCondition, StorageObjectInUseProtection, PersistentVolumeClaimResize, ExtendedResourceToleration, CertificateApproval, PodPriority, CertificateSigning, CertificateSubjectRestriction, RuntimeClass, DefaultIngressClass	The AWS Security Token Service endpoint is reverted back to the global endpoint from the previous platform version. If you want to use the Regional endpoint when using IAM roles for service accounts, then you have to enable it. For instructions on how to enable the regional endpoint, see Configure the AWS Security Token Service endpoint for a service account (p. 454).	April 8, 2022
1.18.20	eks.10	NamespaceLifecycle, LimitRanger, ServiceAccount, DefaultStorageClass, ResourceQuota, DefaultTolerationSeconds, NodeRestriction, MutatingAdmissionWebhook, ValidatingAdmissionWebhook, PodSecurityPolicy, TaintNodesByCondition, StorageObjectInUseProtection, PersistentVolumeClaimResize, CertificateApproval, PodPriority, CertificateSigning, CertificateSubjectRestriction, RuntimeClass, DefaultIngressClass	When using IAM roles for service accounts (p. 444), the AWS Security Token Service Regional endpoint is now used by default instead of the global endpoint. This change is reverted back to the global endpoint in eks.11 however. An updated Fargate scheduler provisions nodes at a significantly higher rate during large deployments.	March 10, 2022

Kubernetes version	Amazon EKS platform version	Enabled admission controllers	Release notes	Release date
1.18.20	eks.9	NamespaceLifecycle, LimitRanger, ServiceAccount, DefaultStorageClass, ResourceQuota, DefaultTolerationSeconds, NodeRestriction, MutatingAdmissionWebhook, ValidatingAdmissionWebhook, PodSecurityPolicy, TaintNodesByCondition, StorageObjectInUseProtection, PersistentVolumeClaimResize, CertificateApproval, PodPriority, CertificateSigning, CertificateSubjectRestriction, RuntimeClass, DefaultIngressClass	New platform version with support for Windows IPv4 address management on the VPC Resource Controller running on the Kubernetes control plane. Added the Kubernetes filter directive for Fargate Fluent Bit logging.	
1.18.20	eks.8	NamespaceLifecycle, LimitRanger, ServiceAccount, DefaultStorageClass, ResourceQuota, DefaultTolerationSeconds, NodeRestriction, MutatingAdmissionWebhook, ValidatingAdmissionWebhook, PodSecurityPolicy, TaintNodesByCondition, Priority, StorageObjectInUseProtection, PersistentVolumeClaimResize	New platform version with security fixes and enhancements.	
1.18.16	eks.7	NamespaceLifecycle, LimitRanger, ServiceAccount, DefaultStorageClass, ResourceQuota, DefaultTolerationSeconds, NodeRestriction, MutatingAdmissionWebhook, ValidatingAdmissionWebhook, PodSecurityPolicy, TaintNodesByCondition, Priority, StorageObjectInUseProtection, PersistentVolumeClaimResize	New platform version that supports custom security groups with Fargate.	

Kubernete version	Amazon EKS platform version	Enabled admission controllers	Release notes	Release date
1.18.16	eks.6	NamespaceLifecycle, LimitRanger, ServiceAccount, DefaultStorageClass, ResourceQuota, DefaultTolerationSeconds, NodeRestriction, MutatingAdmissionWebhook, ValidatingAdmissionWebhook, PodSecurityPolicy, TaintNodesByCondition, Priority, StorageObjectInUseProtection, PersistentVolumeClaimResize	New platform version with security fixes and enhancements.	
1.18.16	eks.5	NamespaceLifecycle, LimitRanger, ServiceAccount, DefaultStorageClass, ResourceQuota, DefaultTolerationSeconds, NodeRestriction, MutatingAdmissionWebhook, ValidatingAdmissionWebhook, PodSecurityPolicy, TaintNodesByCondition, Priority, StorageObjectInUseProtection, PersistentVolumeClaimResize	New platform version with security fixes and enhancements.	
1.18.9	eks.4	NamespaceLifecycle, LimitRanger, ServiceAccount, DefaultStorageClass, ResourceQuota, DefaultTolerationSeconds, NodeRestriction, MutatingAdmissionWebhook, ValidatingAdmissionWebhook, PodSecurityPolicy, TaintNodesByCondition, Priority, StorageObjectInUseProtection, PersistentVolumeClaimResize	New platform version with security fixes and enhancements.	

Kubernetes version	Amazon EKS platform version	Enabled admission controllers	Release notes	Release date
1.18.9	eks.3	NamespaceLifecycle, LimitRanger, ServiceAccount, DefaultStorageClass, ResourceQuota, DefaultTolerationSeconds, NodeRestriction, MutatingAdmissionWebhook, ValidatingAdmissionWebhook, PodSecurityPolicy, TaintNodesByCondition, Priority, StorageObjectInUseProtection, PersistentVolumeClaimResize	Includes support for Amazon EKS add-ons (p. 389) and Fargate logging (p. 161).	
1.18.9	eks.2	NamespaceLifecycle, LimitRanger, ServiceAccount, DefaultStorageClass, ResourceQuota, DefaultTolerationSeconds, NodeRestriction, MutatingAdmissionWebhook, ValidatingAdmissionWebhook, PodSecurityPolicy, TaintNodesByCondition, Priority, StorageObjectInUseProtection, PersistentVolumeClaimResize	New platform version with security fixes and enhancements.	
1.18.8	eks.1	NamespaceLifecycle, LimitRanger, ServiceAccount, DefaultStorageClass, ResourceQuota, DefaultTolerationSeconds, NodeRestriction, MutatingAdmissionWebhook, ValidatingAdmissionWebhook, PodSecurityPolicy, TaintNodesByCondition, Priority, StorageObjectInUseProtection, PersistentVolumeClaimResize	Initial release of Kubernetes version 1.18 for Amazon EKS. For more information, see Kubernetes 1.18 (p. 71).	

Autoscaling

Autoscaling is a function that automatically scales your resources up or down to meet changing demands. This is a major Kubernetes function that would otherwise require extensive human resources to perform manually.

Amazon EKS supports two autoscaling products. The Kubernetes Cluster Autoscaler and the Karpenter open source autoscaling project. The cluster autoscaler uses AWS scaling groups, while Karpenter works directly with the Amazon EC2 fleet.

Cluster Autoscaler

The Kubernetes Cluster Autoscaler automatically adjusts the number of nodes in your cluster when pods fail or are rescheduled onto other nodes. The Cluster Autoscaler is typically installed as a Deployment in your cluster. It uses leader election to ensure high availability, but scaling is done by only one replica at a time.

Before you deploy the Cluster Autoscaler, make sure that you're familiar with how Kubernetes concepts interface with AWS features. The following terms are used throughout this topic:

- **Kubernetes Cluster Autoscaler** – A core component of the Kubernetes control plane that makes scheduling and scaling decisions. For more information, see Kubernetes Control Plane FAQ on GitHub.
- **AWS Cloud provider implementation** – An extension of the Kubernetes Cluster Autoscaler that implements the decisions of the Kubernetes Cluster Autoscaler by communicating with AWS products and services such as Amazon EC2. For more information, see Cluster Autoscaler on AWS on GitHub.
- **Node groups** – A Kubernetes abstraction for a group of nodes within a cluster. Node groups aren't a true Kubernetes resource, but they're found as an abstraction in the Cluster Autoscaler, Cluster API, and other components. Nodes that are found within a single node group might share several common properties such as labels and taints. However, they can still consist of more than one Availability Zone or instance type.
- **Amazon EC2 Auto Scaling groups** – A feature of AWS that's used by the Cluster Autoscaler. Auto Scaling groups are suitable for a large number of use cases. Amazon EC2 Auto Scaling groups are configured to launch instances that automatically join their Kubernetes cluster. They also apply labels and taints to their corresponding node resource in the Kubernetes API.

For reference, Managed node groups (p. 105) are managed using Amazon EC2 Auto Scaling groups, and are compatible with the Cluster Autoscaler.

This topic describes how you can deploy the Cluster Autoscaler to your Amazon EKS cluster and configure it to modify your Amazon EC2 Auto Scaling groups.

Prerequisites

Before deploying the Cluster Autoscaler, you must meet the following prerequisites:

- An existing Amazon EKS cluster – If you don’t have a cluster, see Creating an Amazon EKS cluster (p. 23).
- An existing IAM OIDC provider for your cluster. To determine whether you have one or need to create one, see Create an IAM OIDC provider for your cluster (p. 448).
- Node groups with Auto Scaling groups tags. The Cluster Autoscaler requires the following tags on your Auto Scaling groups so that they can be auto-discovered.
  - If you used `eksctl` to create your node groups, these tags are automatically applied.
  - If you didn't use `eksctl`, 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/<my-cluster>`	`owned`

Key	Value
k8s.io/cluster-autoscaler/enabled	true

Create an IAM policy and role

Create an IAM policy that grants the permissions that the Cluster Autoscaler requires to use an IAM role. Replace all of the *<example-values>* (including <>) with your own values throughout the procedures.

1. Create an IAM policy.

   a. Save the following contents to a file that's named `cluster-autoscaler-policy.json`. If your existing node groups were created with `eksctl` and you used the `--asg-access` option, then this policy already exists and you can skip to step 2.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "VisualEditor0",
            "Effect": "Allow",
            "Action": [
                "autoscaling:SetDesiredCapacity",
                "autoscaling:TerminateInstanceInAutoScalingGroup"
            ],
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "aws:ResourceTag/k8s.io/cluster-autoscaler/<my-cluster>": "owned"
                }
            }
        },
        {
            "Sid": "VisualEditor1",
            "Effect": "Allow",
            "Action": [
                "autoscaling:DescribeAutoScalingInstances",
                "autoscaling:DescribeAutoScalingGroups",
                "ec2:DescribeLaunchTemplateVersions",
                "autoscaling:DescribeTags",
                "autoscaling:DescribeLaunchConfigurations"
            ],
            "Resource": "*"
        }
    ]
}
```

   b. Create the policy with the following command. You can change the value for `policy-name`.

```
aws iam create-policy \
    --policy-name AmazonEKSClusterAutoscalerPolicy \
    --policy-document file://cluster-autoscaler-policy.json
```

   Take note of the Amazon Resource Name (ARN) that's returned in the output. You need to use it in a later step.

2. You can create an IAM role and attach an IAM policy to it using `eksctl` or the AWS Management Console. Select the desired tab for the following instructions.

eksctl

1. Run the following command if you created your Amazon EKS cluster with `eksctl`. If you created your node groups using the `--asg-access` option, then replace *<AmazonEKSClusterAutoscalerPolicy>* with the name of the IAM policy that `eksctl` created for you. The policy name is similar to `eksctl-<my-cluster>-nodegroup-ng-<xxxxxxxx>-PolicyAutoScaling`.

```
eksctl create iamserviceaccount \
  --cluster=<my-cluster> \
  --namespace=kube-system \
  --name=cluster-autoscaler \
  --attach-policy-
arn=arn:aws:iam::<111122223333>:policy/<AmazonEKSClusterAutoscalerPolicy> \
  --override-existing-serviceaccounts \
  --approve
```

2. We recommend that, if you created your node groups using the `--asg-access` option, you detach the IAM policy that `eksctl` created and attached to the Amazon EKS node IAM role (p. 476) that `eksctl` created for your node groups. You detach the policy from the node IAM role for Cluster Autoscaler to function properly. Detaching the policy doesn't give other pods on your nodes the permissions in the policy. For more information, see Removing IAM identity permissions in the Amazon EC2 User Guide for Linux Instances.

AWS Management Console

a. Open the IAM console at https://console.aws.amazon.com/iam/.

b. In the left navigation pane, choose **Roles**. Then choose **Create role**.

c. In the **Trusted entity type** section, choose **Web identity**.

d. In the **Web identity** section:

   i. For **Identity provider**, choose the **OpenID Connect provider URL** for your cluster (as shown in the cluster **Overview** tab in Amazon EKS).

   ii. For **Audience**, choose `sts.amazonaws.com`.

e. Choose **Next**.

f. In the **Filter policies** box, enter **`AmazonEKSClusterAutoscalerPolicy`**. Then select the check box to the left of the policy name returned in the search.

g. Choose **Next**.

h. For **Role name**, enter a unique name for your role, such as **`AmazonEKSClusterAutoscalerRole`**.

i. For **Description**, enter descriptive text such as **`Amazon EKS - Cluster autoscaler role`**.

j. Choose **Create role**.

k. After the role is created, choose the role in the console to open it for editing.

l. Choose the **Trust relationships** tab, and then choose **Edit trust policy**.

m. Find the line that looks similar to the following:

```
"oidc.eks.region-code.amazonaws.com/id/EXAMPLED539D4633E53DE1B71EXAMPLE:aud":
 "sts.amazonaws.com"
```

Change the line to look like the following line. Replace *`EXAMPLED539D4633E53DE1B71EXAMPLE`* with your cluster's OIDC provider ID. Replace *`region-code`* with the AWS Region that your cluster is in.

```
"oidc.eks.region-code.amazonaws.com/id/EXAMPLED539D4633E53DE1B71EXAMPLE:sub":
 "system:serviceaccount:kube-system:cluster-autoscaler"
```

n. Choose **Update policy** to finish.

Deploy the Cluster Autoscaler

Complete the following steps to deploy the Cluster Autoscaler. We recommend that you review Deployment considerations (p. 93) and optimize the Cluster Autoscaler deployment before you deploy it to a production cluster.

To deploy the Cluster Autoscaler

1. Download the Cluster Autoscaler YAML file.

```
curl -o cluster-autoscaler-autodiscover.yaml https://raw.githubusercontent.com/
kubernetes/autoscaler/master/cluster-autoscaler/cloudprovider/aws/examples/cluster-
autoscaler-autodiscover.yaml
```

2. Modify the YAML file and replace *<YOUR CLUSTER NAME>* with your cluster name. Also consider replacing the `cpu` and `memory` values as determined by your environment.
3. Apply the YAML file to your cluster.

```
kubectl apply -f cluster-autoscaler-autodiscover.yaml
```

4. Annotate the `cluster-autoscaler` service account with the ARN of the IAM role that you created previously. Replace the *<example values>* with your own values.

```
kubectl annotate serviceaccount cluster-autoscaler \
  -n kube-system \
  eks.amazonaws.com/role-
arn=arn:aws:iam::<ACCOUNT_ID>:role/<AmazonEKSClusterAutoscalerRole>
```

5. Patch the deployment to add the `cluster-autoscaler.kubernetes.io/safe-to-evict` annotation to the Cluster Autoscaler pods with the following command.

```
kubectl patch deployment cluster-autoscaler \
  -n kube-system \
  -p '{"spec":{"template":{"metadata":{"annotations":{"cluster-
autoscaler.kubernetes.io/safe-to-evict": "false"}}}}}'
```

6. 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 add the following options. `--balance-similar-node-groups` ensures that there is enough available compute across all availability zones. `--skip-nodes-with-system-pods=false` ensures that there are no problems with scaling to zero.

- `--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.

7. Open the Cluster Autoscaler releases page from GitHub in a web browser and find the latest Cluster Autoscaler version that matches the Kubernetes major and minor version of your cluster. For example, if the Kubernetes version of your cluster is `1.22`, find the latest Cluster Autoscaler release that begins with `1.22`. Record the semantic version number (`1.22.n`) for that release to use in the next step.
8. Set the Cluster Autoscaler image tag to the version that you recorded in the previous step with the following command. Replace *`1.22.n`* with your own value.

```
kubectl set image deployment cluster-autoscaler \
  -n kube-system \
  cluster-autoscaler=k8s.gcr.io/autoscaling/cluster-autoscaler:v<1.22.n>
```

View your Cluster Autoscaler logs

After you have deployed the Cluster Autoscaler, you can view the logs and verify that it's monitoring your cluster load.

View your Cluster Autoscaler logs with the following command.

```
kubectl -n kube-system logs -f deployment.apps/cluster-autoscaler
```

The example output is as follows.

```
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
```

Deployment considerations

Review the following considerations to optimize your Cluster Autoscaler deployment.

Scaling considerations

The Cluster Autoscaler can be configured to include any additional features of your nodes. These features can include Amazon EBS volumes attached to nodes, Amazon EC2 instance types of nodes, or GPU accelerators.

Scope node groups across more than one Availability Zone

We recommend that you configure multiple node groups, scope each group to a single Availability Zone, and enable the `--balance-similar-node-groups` feature. If you only create one node group, scope that node group to span over more than one Availability Zone.

When setting `--balance-similar-node-groups` to true, make sure that the node groups you want the Cluster Autoscaler to balance have matching labels (except for automatically added zone labels). You can pass a `--balancing-ignore-label` flag to nodes with different labels to balance them regardless, but this should only be done as needed.

Optimize your node groups

The Cluster Autoscaler makes assumptions about how you're using node groups. This includes which instance types that you use within a group. To align with these assumptions, configure your node group based on these considerations and recommendations:

- Each node in a node group must have identical scheduling properties. This includes labels, taints, and resources.
  - For `MixedInstancePolicies`, the instance types must have compatible CPU, memory, and GPU specifications.
  - The first instance type that's specified in the policy simulates scheduling.
  - If your policy has additional instance types with more resources, resources might be wasted after scale out.
  - If your policy has additional instance types with fewer resources than the original instance types, pods might fail to schedule on the instances.
- Configure a smaller number of node groups with a larger number of nodes because the opposite configuration can negatively affect scalability.
- Use Amazon EC2 features whenever both systems provide support them (for example, use Regions and `MixedInstancePolicy`.)

If possible, we recommend that you use Managed node groups (p. 105). Managed node groups come with powerful management features. These include features for Cluster Autoscaler such as automatic Amazon EC2 Auto Scaling group discovery and graceful node termination.

Use EBS volumes as persistent storage

Persistent storage is critical for building stateful applications, such as databases and distributed caches. With Amazon EBS volumes, you can build stateful applications on Kubernetes. However, you're limited to only building them within a single Availability Zone. For more information, see How do I use persistent storage in Amazon EKS?. For a better solution, consider building stateful applications that are sharded across more than one Availability Zone using a separate Amazon EBS volume for each Availability Zone. Doing so means that your application can be highly available. Moreover, the Cluster Autoscaler can balance the scaling of the Amazon EC2 Auto Scaling groups. To do this, make sure that the following conditions are met:

- Node group balancing is enabled by setting `balance-similar-node-groups=true`.
- Your node groups are configured with identical settings (outside of being in more than one Availability Zone and using different Amazon EBS volumes).

Co-scheduling

Machine learning distributed training jobs benefit significantly from the minimized latency of same-zone node configurations. These workloads deploy multiple pods to a specific zone.You can achieve this by setting pod affinity for all co-scheduled pods or node affinity using `topologyKey: topology.kubernetes.io/zone`. Using this configuration, the Cluster Autoscaler scales out a specific zone to match demands. Allocate multiple Amazon EC2 Auto Scaling groups, with one for each Availability Zone, to enable failover for the entire co-scheduled workload. Make sure that the following conditions are met:

- Node group balancing is enabled by setting `balance-similar-node-groups=true`.
- Node affinity, pod preemption, or both, are used when clusters include both Regional and Zonal node groups.
  - Use Node Affinity to force or encourage regional pods and avoid zonal node groups.
  - Don't schedule zonal pods onto Regional node groups. Doing so can result in imbalanced capacity for your Regional pods.
  - Configure pod preemption if your zonal workloads can tolerate disruption and relocation. Doing so enforces preemption and rescheduling on a less contested zone for your Regionally scaled pods.

Accelerators and GPUs

Some clusters use specialized hardware accelerators such as a dedicated GPU. When scaling out, the accelerator can take several minutes to advertise the resource to the cluster. During this time, the Cluster Autoscaler simulates that this node has the accelerator. However, until the accelerator becomes ready and updates the available resources of the node, pending pods can't be scheduled on the node. This can result in repeated unnecessary scale out.

Nodes with accelerators and high CPU or memory utilization aren't considered for scale down even if the accelerator is unused. However, this can be result in unncessary costs. To avoid these costs, the Cluster Autoscaler can apply special rules to consider nodes for scale down if they have unoccupied accelerators.

To ensure the correct behavior for these cases, configure the `kubelet` on your accelerator nodes to label the node before it joins the cluster. The Cluster Autoscaler uses this label selector to invoke the accelerator optimized behavior. Make sure that the following conditions are met:

- The `kubelet` for GPU nodes is configured with `--node-labels k8s.amazonaws.com/accelerator=$ACCELERATOR_TYPE`.
- Nodes with accelerators adhere to the identical scheduling properties rule.

Scaling from zero

Cluster Autoscaler can scale node groups to and from zero. This might result in a significant cost savings. The Cluster Autoscaler detects the CPU, memory, and GPU resources of an Auto Scaling group by inspecting the `InstanceType` that is specified in its `LaunchConfiguration` or `LaunchTemplate`. Some pods require additional resources such as `WindowsENI` or `PrivateIPv4Address`. Or they might require specific `NodeSelectors` or `Taints`. These latter two can't be discovered from the `LaunchConfiguration`. However, the Cluster Autoscaler can account for these factors by discovering them from the following tags on the Auto Scaling group.

```
Key: k8s.io/cluster-autoscaler/node-template/resources/$RESOURCE_NAME
Value: 5
Key: k8s.io/cluster-autoscaler/node-template/label/$LABEL_KEY
Value: $LABEL_VALUE
Key: k8s.io/cluster-autoscaler/node-template/taint/$TAINT_KEY
Value: NoSchedule
```

Note

- When scaling to zero, your capacity is returned to Amazon EC2 and might become unavailable in the future.
- You can use describeNodegroup to diagnose issues with managed node groups when scaling to and from zero.

Additional configuration parameters

There are many configuration options that can be used to tune the behavior and performance of the Cluster Autoscaler. For a complete list of parameters, see What are the parameters to CA? on GitHub.

Performance considerations

There are a few key items that you can change to tune the performance and scalability of the Cluster Autoscaler. The primary ones are any resources that are provided to the process, the scan interval of the algorithm, and the number of node groups in the cluster. However, there are also several other factors that are involved in the true runtime complexity of this algorithm. These include the scheduling plug-in complexity and the number of pods. These are considered to be unconfigurable parameters because they're integral to the workload of the cluster and can't easily be tuned.

Scalability refers to how well the Cluster Autoscaler performs as the number of pods and nodes in your Kubernetes cluster increases. If its scalability quotas are reached, the performance and functionality of the Cluster Autoscaler degrades. Additionally, when it exceeds its scalability quotas, the Cluster Autoscaler can no longer add or remove nodes in your cluster.

Performance refers to how quickly the Cluster Autoscaler can make and implement scaling decisions. A perfectly performing Cluster Autoscaler instantly make decisions and invoke scaling actions in response to specific conditions, such as a pod becoming unschedulable.

Be familiar with the runtime complexity of the autoscaling algorithm. Doing so makes it easier to tune the Cluster Autoscaler to operate well in large clusters (with more than 1,000 nodes).

The Cluster Autoscaler loads the state of the entire cluster into memory. This includes the pods, nodes, and node groups. On each scan interval, the algorithm identifies unschedulable pods and simulates scheduling for each node group. Know that tuning these factors in different ways comes with different tradeoffs.

Vertical autoscaling

You can scale the Cluster Autoscaler to larger clusters by increasing the resource requests for its deployment. This is one of the simpler methods to do this. Increase both the memory and CPU for the large clusters. Know that how much you should increase the memory and CPU depends greatly on the specific cluster size. The autoscaling algorithm stores all pods and nodes in memory. This can result in a memory footprint larger than a gigabyte in some cases. You usually need to increase resources manually. If you find that you often need to manually increase resources, consider using the Addon Resizer or Vertical Pod Autoscaler to automate the process.

Reducing the number of node groups

You can lower the number of node groups to improve the performance of the Cluster Autoscaler in large clusters. If you structured your node groups on an individual team or application basis, this might be challenging. Even though this is fully supported by the Kubernetes API, this is considered to be a Cluster Autoscaler anti-pattern with repercussions for scalability. There are many advantages to using multiple node groups that, for example, use both Spot or GPU instances. In many cases, there are alternative designs that achieve the same effect while using a small number of groups. Make sure that the following conditions are met:

- Isolate pods by using namespaces rather than node groups.
  - You might not be able to do this in low-trust multi-tenant clusters.
  - Set pod `ResourceRequests` and `ResourceLimits` properly to avoid resource contention.
  - Use larger instance types for more optimal bin packing and reduced system pod overhead.
- Avoid using `NodeTaints` or `NodeSelectors` to schedule pods. Only use them on a limited basis.
- Define Regional resources as a single Amazon EC2 Auto Scaling group with more than one Availability Zone.

Reducing the scan interval

Using a low scan interval, such as the default setting of ten seconds, ensures that the Cluster Autoscaler responds as quickly as possible when pods become unschedulable. However, each scan results in many API calls to the Kubernetes API and Amazon EC2 Auto Scaling group or the Amazon EKS managed node group APIs. These API calls can result in rate limiting or even service unavailability for your Kubernetes control plane.

The default scan interval is ten seconds, but on AWS, launching a node takes significantly longer to launch a new instance. This means that it’s possible to increase the interval without significantly increasing overall scale up time. For example, if it takes two minutes to launch a node, don't change the interval to one minute because this might result in a trade-off of 6x reduced API calls for 38% slower scale ups.

Sharing across node groups

You can configure the Cluster Autoscaler to operate on a specific set of node groups. By using this functionality, you can deploy multiple instances of the Cluster Autoscaler. Configure each instance to operate on a different set of node groups. By doing this, you can use arbitrarily large numbers of node groups, trading cost for scalability. However, we only recommend that you do this as last resort for improving the performance of Cluster Autoscaler.

This configuration has its drawbacks. It can result in unnecessary scale out of multiple node groups. The extra nodes scale back in after the `scale-down-delay`.

```
metadata:
  name: cluster-autoscaler
  namespace: cluster-autoscaler-1

...

--nodes=1:10:k8s-worker-asg-1
--nodes=1:10:k8s-worker-asg-2

---

metadata:
  name: cluster-autoscaler
  namespace: cluster-autoscaler-2

...

--nodes=1:10:k8s-worker-asg-3
--nodes=1:10:k8s-worker-asg-4
```

Make sure that the following conditions are true.

- Each shard is configured to point to a unique set of Amazon EC2 Auto Scaling groups.
- Each shard is deployed to a separate namespace to avoid leader election conflicts.

Cost efficiency and availability

The primary options for tuning the cost efficiency of the Cluster Autoscaler are related to provisioning Amazon EC2 instances. Additionally, cost efficiency must be balanced with availability. This section describes strategies such as using Spot instances to reduce costs and overprovisioning to reduce latency when creating new nodes.

- **Availability** – Pods can be scheduled quickly and without disruption. This is true even for when newly created pods need to be scheduled and for when a scaled down node terminates any remaining pods scheduled to it.
- **Cost** – Determined by the decision behind scale-out and scale-in events. Resources are wasted if an existing node is underutilized or if a new node is added that is too large for incoming pods. Depending on the specific use case, there can be costs that are associated with prematurely terminating pods due to an aggressive scale down decision.

Spot instances

You can use Spot Instances in your node groups to save up to 90% off the on-demand price. This has the trade-off of Spot Instances possibly being interrupted at any time when Amazon EC2 needs the capacity back. `Insufficient Capacity Errors` occur whenever your Amazon EC2 Auto Scaling group can't scale up due to a lack of available capacity. Selecting many different instance families has two main benefits. First, it can increase your chance of achieving your desired scale by tapping into many Spot capacity pools. Second, it also can decrease the impact of Spot Instance interruptions on cluster availability. Mixed Instance Policies with Spot Instances are a great way to increase diversity without increasing the number of node groups. However, know that, if you need guaranteed resources, use On-Demand Instances instead of Spot Instances.

Spot instances might be terminated when demand for instances rises. For more information, see the Spot Instance Interruptions section of the Amazon EC2 User Guide for Linux Instances. The AWS Node Termination Handler project automatically alerts the Kubernetes control plane when a node is going down. The project uses the Kubernetes API to cordon the node to ensure that no new work is scheduled there, then drains it and removes any existing work.

It’s critical that all instance types have similar resource capacity when configuring Mixed instance policies. The scheduling simulator of the autoscaler uses the first instance type in the Mixed Instance Policy. If subsequent instance types are larger, resources might be wasted after a scale up. If the instances are smaller, your pods may fail to schedule on the new instances due to insufficient capacity. For example, `M4`, `M5`, `M5a,` and `M5n` instances all have similar amounts of CPU and memory and are great candidates for a Mixed Instance Policy. The Amazon EC2 Instance Selector tool can help you identify similar instance types or additional critical criteria, such as size. For more information, see Amazon EC2 Instance Selector on GitHub.

We recommend that you isolate your On-Demand and Spot instances capacity into separate Amazon EC2 Auto Scaling groups. We recommend this over using a base capacity strategy because the scheduling properties of On-Demand and Spot instances are different. Spot Instances can be interrupted at any time. When Amazon EC2 needs the capacity back, preemptive nodes are often tainted, thus requiring an explicit pod toleration to the preemption behavior. This results in different scheduling properties for the nodes, so they should be separated into multiple Amazon EC2 Auto Scaling groups.

The Cluster Autoscaler involves the concept of Expanders. They collectively provide different strategies for selecting which node group to scale. The strategy `--expander=least-waste` is a good general purpose default, and if you're going to use multiple node groups for Spot Instance diversification, as described previously, it could help further cost-optimize the node groups by scaling the group that would be best utilized after the scaling activity.

Prioritizing a node group or Auto Scaling group

You might also configure priority-based autoscaling by using the `Priority` expander. `--expander=priority` enables your cluster to prioritize a node group or Auto Scaling group, and if it is unable to scale for any reason, it will choose the next node group in the prioritized list. This is useful in situations where, for example, you want to use `P3` instance types because their GPU provides optimal performance for your workload, but as a second option you can also use `P2` instance types. For example:

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: cluster-autoscaler-priority-expander
  namespace: kube-system
data:
  priorities: |-
    10:
      - .*p2-node-group.*
    50:
      - .*p3-node-group.*
```

Cluster Autoscaler attempts to scale up the Amazon EC2 Auto Scaling group matching the name `p3-node-group`. If this operation doesn't succeed within `--max-node-provision-time`, it then attempts to scale an Amazon EC2 Auto Scaling group matching the name `p2-node-group`. This value defaults to 15 minutes and can be reduced for more responsive node group selection. However, if the value is too low, unnecessary scaleout might occur.

Overprovisioning

The Cluster Autoscaler helps to minimize costs by ensuring that nodes are only added to the cluster when they're needed and are removed when they're unused. This significantly impacts deployment latency because many pods must wait for a node to scale up before they can be scheduled. Nodes can take multiple minutes to become available, which can increase pod scheduling latency by an order of magnitude.

This can be mitigated using overprovisioning, which trades cost for scheduling latency. Overprovisioning is implemented using temporary pods with negative priority. These pods occupy space in the cluster. When newly created pods are unschedulable and have a higher priority, the temporary pods are preempted to make room. Then, the temporary pods become unschedulable, causing the Cluster Autoscaler to scale out new overprovisioned nodes.

There are other benefits to overprovisioning. Without overprovisioning, pods in a highly utilized cluster make less optimal scheduling decisions using the `preferredDuringSchedulingIgnoredDuringExecution` rule. A common use case for this is to separate pods for a highly available application across Availability Zones using `AntiAffinity`. Overprovisioning can significantly increase the chance that a node of the desired zone is available.

It's important to choose an appropriate amount of overprovisioned capacity. One way that you can make sure that you choose an appropriate amount is by taking your average scaleup frequency and dividing it by the duration of time it takes to scale up a new node. For example, if, on average, you require a new node every 30 seconds and Amazon EC2 takes 30 seconds to provision a new node, a single node of overprovisioning ensures that there’s always an extra node available. Doing this can reduce scheduling latency by 30 seconds at the cost of a single additional Amazon EC2 instance. To make better zonal scheduling decisions, you can also overprovision the number of nodes to be the same as the number of Availability Zones in your Amazon EC2 Auto Scaling group. Doing this ensures that the scheduler can select the best zone for incoming pods.

Prevent scale down eviction

Some workloads are expensive to evict. Big data analysis, machine learning tasks, and test runners can take a long time to complete and must be restarted if they're interrupted. The Cluster Autoscaler helps to scale down any node under the `scale-down-utilization-threshold`. This interrupts any remaining pods on the node. However, you can prevent this from happening by ensuring that pods that are expensive to evict are protected by a label recognized by the Cluster Autoscaler. To do this, ensure that pods that are expensive to evict have the label `cluster-autoscaler.kubernetes.io/safe-to-evict=false`.

Karpenter

Amazon EKS supports the Karpenter open-source autoscaling project. See the Karpenter documentation to deploy it.

About Karpenter

Karpenter is a flexible, high-performance Kubernetes cluster autoscaler that helps improve application availability and cluster efficiency. Karpenter launches right-sized compute resources, (for example, Amazon EC2 instances), in response to changing application load in under a minute. Through integrating Kubernetes with AWS, Karpenter can provision just-in-time compute resources that precisely meet the requirements of your workload. Karpenter automatically provisions new compute resources based on the specific requirements of cluster workloads. These include compute, storage, acceleration, and scheduling requirements. Amazon EKS supports clusters using Karpenter, although Karpenter works with any conformant Kubernetes cluster.

How Karpenter works

Karpenter works in tandem with the Kubernetes scheduler by observing incoming pods over the lifetime of the cluster. It launches or terminates nodes to maximize application availability and cluster utilization. When there is enough capacity in the cluster, the Kubernetes scheduler will place incoming pods as usual. When pods are launched that cannot be scheduled using the existing capacity of the cluster, Karpenter bypasses the Kubernetes scheduler and works directly with your provider's compute service, (for example, Amazon EC2), to launch the minimal compute resources needed to fit those pods and binds the pods to the nodes provisioned. As pods are removed or rescheduled to other nodes, Karpenter looks for opportunities to terminate under-utilized nodes. Running fewer, larger nodes in your cluster reduces overhead from daemonsets and Kubernetes system components and provides more opportunities for efficient bin-packing.

Prerequisites

Before deploying Karpenter, you must meet the following prerequisites:

- An existing Amazon EKS cluster – If you don't have a cluster, see Creating an Amazon EKS cluster (p. 23).
- An existing IAM OIDC provider for your cluster. To determine whether you have one or need to create one, see Create an IAM OIDC provider for your cluster (p. 448).
- A user or role with permission to create a cluster.
- AWS CLI
- Installing `kubectl` (p. 4)
- Using Helm with Amazon EKS (p. 432)

You can deploy Karpenter using `eksctl` if you prefer. See Installing `eksctl` (p. 10).

Amazon EKS nodes

Your Amazon EKS cluster can schedule pods on any combination of Self-managed nodes (p. 128), Amazon EKS Managed node groups (p. 105), and AWS Fargate (p. 149). To learn more about nodes deployed in your cluster, see View Kubernetes resources (p. 508).

Note
Nodes must be in the same VPC as the subnets you selected when you created the cluster. However, the nodes don't have to be in the same subnets.

The following table provides several criteria to evaluate when deciding which options best meet your requirements. This table doesn't include connected nodes (p. 545) that were created outside of Amazon EKS, which can only be viewed.

Note
Bottlerocket has some specific differences from the general information in this table. For more information, see the Bottlerocket documentation on GitHub.

Criteria	EKS managed node groups	Self managed nodes	AWS Fargate
Can be deployed to AWS Outposts	No	Yes – For more information, see Amazon EKS nodes on AWS Outposts (p. 222).	No
Can be deployed to an AWS Local Zone	No	Yes – For more information, see Amazon EKS and AWS Local Zones (p. 528).	No
Can run containers that require Windows	No	Yes (p. 53) – Your cluster still requires at least one (two recommended for availability) Linux node though.	No
Can run containers that require Linux	Yes	Yes	Yes
Can run workloads that require the Inferentia chip	Yes (p. 399) – Amazon Linux nodes only	Yes (p. 399) – Amazon Linux only	No
Can run workloads that require a GPU	Yes (p. 180) – Amazon Linux nodes only	Yes (p. 180) – Amazon Linux only	No
Can run workloads that require Arm processors	Yes (p. 182)	Yes (p. 182)	No
Can run AWS Bottlerocket	Yes	Yes (p. 134)	No

Criteria	EKS managed node groups	Self managed nodes	AWS Fargate
Pods share a kernel runtime environment with other pods	Yes – All of your pods on each of your nodes	Yes – All of your pods on each of your nodes	No – Each pod has a dedicated kernel
Pods share CPU, memory, storage, and network resources with other pods.	Yes – Can result in unused resources on each node	Yes – Can result in unused resources on each node	No – Each pod has dedicated resources and can be sized independently to maximize resource utilization.
Pods can use more hardware and memory than requested in pod specs	Yes – If the pod requires more resources than requested, and resources are available on the node, the pod can use additional resources.	Yes – If the pod requires more resources than requested, and resources are available on the node, the pod can use additional resources.	No – The pod can be re-deployed using a larger vCPU and memory configuration though.
Must deploy and manage Amazon EC2 instances	Yes (p. 108) – automated through Amazon EKS if you deployed an Amazon EKS optimized AMI. If you deployed a custom AMI, then you must update the instance manually.	Yes – Manual configuration or using Amazon EKS provided AWS CloudFormation templates to deploy Linux (x86) (p. 129), Linux (Arm) (p. 182), or Windows (p. 53) nodes.	No
Must secure, maintain, and patch the operating system of Amazon EC2 instances	Yes	Yes	No
Can provide bootstrap arguments at deployment of a node, such as extra kubelet arguments.	Yes – Using a launch template (p. 120) with a custom AMI	Yes – For more information, view the bootstrap script usage information on GitHub.	No
Can assign IP addresses to pods from a different CIDR block than the IP address assigned to the node.	Yes – Using a launch template (p. 120) with a custom AMI	Yes, using Tutorial: Custom networking (p. 298).	No
Can SSH into node	Yes	Yes	No – There's no node host operating system to SSH to.

Criteria	EKS managed node groups	Self managed nodes	AWS Fargate
Can deploy your own custom AMI to nodes	Yes – Using a launch template (p. 120)	Yes	No
Can deploy your own custom CNI to nodes	Yes – Using a launch template (p. 120) with a custom AMI	Yes	No
Must update node AMI on your own	Yes (p. 115) – If you deployed an Amazon EKS optimized AMI, you're notified in the Amazon EKS console when updates are available. You can perform the update with one-click in the console. If you deployed a custom AMI, you're not notified in the Amazon EKS console when updates are available. You must perform the update on your own.	Yes (p. 147) – Using tools other than the Amazon EKS console. This is because self managed nodes can't be managed with the Amazon EKS console.	No

Criteria	EKS managed node groups	Self managed nodes	AWS Fargate
Must update node Kubernetes version on your own	Yes (p. 115) – If you deployed an Amazon EKS optimized AMI, you're notified in the Amazon EKS console when updates are available. You can perform the update with one-click in the console. If you deployed a custom AMI, you're not notified in the Amazon EKS console when updates are available. You must perform the update on your own.	Yes (p. 147) – Using tools other than the Amazon EKS console. This is because self managed nodes can't be managed with the Amazon EKS console.	No – You don't manage nodes.
Can use Amazon EBS storage with pods	Yes (p. 229)	Yes (p. 229)	No
Can use Amazon EFS storage with pods	Yes (p. 242)	Yes (p. 242)	Yes (p. 242)
Can use Amazon FSx for Lustre storage with pods	Yes (p. 254)	Yes (p. 254)	No
Can use Network Load Balancer for services	Yes (p. 373)	Yes (p. 373)	Yes, when using the Create a network load balancer (p. 375)
Pods can run in a public subnet	Yes	Yes	No
Can assign different VPC security groups to individual pods	Yes (p. 314) – Linux nodes only	Yes (p. 314) – Linux nodes only	Yes, in version `1.18` or later clusters
Can run Kubernetes DaemonSets	Yes	Yes	No
Support `HostPort` and `HostNetwork` in the pod manifest	Yes	Yes	No
AWS Region availability	All Amazon EKS supported regions	All Amazon EKS supported regions	Some Amazon EKS supported regions (p. 149)
Can run containers on EC2 dedicated hosts	No	Yes	No

Criteria	EKS managed node groups	Self managed nodes	AWS Fargate
Pricing	Cost of Amazon EC2 instance that runs multiple pods. For more information, see Amazon EC2 pricing.	Cost of Amazon EC2 instance that runs multiple pods. For more information, see Amazon EC2 pricing.	Cost of an individual Fargate memory and CPU configuration. Each pod has its own cost. For more information, see AWS Fargate pricing.

Managed node groups

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

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, automatically update, or terminate nodes for your cluster with a single operation. Node updates and terminations automatically drain nodes to ensure that your applications stay available.

Every managed node is provisioned as part of an Amazon EC2 Auto Scaling group that's managed for you by Amazon EKS. Every resource including the instances and Auto Scaling groups runs within your AWS account. Each node group runs 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. 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 Amazon EKS – AWS Management Console and AWS CLI (p. 15).

To add a managed node group to an existing cluster, see Creating a managed node group (p. 108).

Managed node groups concepts

- Amazon EKS managed node groups create and manage Amazon EC2 instances for you.
- Every managed node is provisioned as part of an Amazon EC2 Auto Scaling group that's managed for you by Amazon EKS. Moreover, every resource including Amazon EC2 instances and Auto Scaling groups run within your AWS account.
- The Auto Scaling group of a managed node group spans every subnet 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. 89).

  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. 89), you should

Check Part 2 for further chapter