
Amazon EKS

User Guide



Amazon EKS: User Guide

Copyright © Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

Amazon's trademarks and trade dress may not be used in connection with any product or service that is not Amazon's, in any manner that is likely to cause confusion among customers, or in any manner that disparages or discredits Amazon. All other trademarks not owned by Amazon are the property of their respective owners, who may or may not be affiliated with, connected to, or sponsored by Amazon.

Table of Contents

What is Amazon EKS?	1
Amazon EKS control plane architecture	1
How does Amazon EKS work?	2
Pricing	2
Deployment options	2
Getting started with Amazon EKS	4
Installing <code>kubectl</code>	4
Installing <code>eksctl</code>	10
Installing or upgrading <code>eksctl</code>	11
Using <code>eksctl</code>	12
Prerequisites	13
Step 1: Create cluster and nodes	13
Step 2: View Kubernetes resources	14
Step 4: Delete cluster and nodes	15
Next steps	15
Using the console and AWS CLI	16
Prerequisites	16
Step 1: Create cluster	16
Step 2: Configure cluster communication	18
Step 3: Create nodes	18
Step 4: View resources	22
Step 5: Delete resources	22
Next steps	23
Clusters	24
Creating a cluster	24
(Optional) Create an IAM role to own your Amazon EKS cluster	25
Updating a cluster	34
Updating Kubernetes version	34
Kubernetes version 1.22 prerequisites	34
To update the Kubernetes version for your Amazon EKS cluster	38
Enabling secret encryption on an existing cluster	42
Deleting an Amazon EKS cluster	44
Cluster endpoint access	47
Modifying cluster endpoint access	48
Accessing a private only API server	52
Autoscaling	52
Cluster Autoscaler	52
Karpenter	63
Control plane logging	64
Enabling and disabling control plane logs	65
Viewing cluster control plane logs	66
Kubernetes versions	67
Available Amazon EKS Kubernetes versions	67
Kubernetes 1.22	67
Kubernetes 1.21	69
Kubernetes 1.20	70
Kubernetes 1.19	71
Kubernetes 1.18	73
Kubernetes 1.17	73
Amazon EKS Kubernetes release calendar	74
Amazon EKS version support and FAQ	74
Platform versions	76
Kubernetes version 1.22	77
Kubernetes version 1.21	77

Kubernetes version 1.20	80
Kubernetes version 1.19	82
Kubernetes version 1.18	85
Kubernetes version 1.17	89
Windows support	92
Enabling Windows support	93
Removing legacy Windows support	94
Disabling Windows support	95
Deploying Pods	95
Enabling legacy Windows support	96
Viewing API server flags	101
Private clusters	102
Requirements	102
Considerations	103
Creating local copies of container images	104
AWS STS endpoints for IAM roles for service accounts	104
Nodes	105
Managed node groups	109
Managed node groups concepts	109
Managed node group capacity types	110
Creating a managed node group	112
Updating a managed node group	119
Node taints on managed node groups	123
Launch template support	124
Deleting a managed node group	131
Self-managed nodes	132
Amazon Linux	133
Bottlerocket	138
Windows	140
Updates	146
AWS Fargate	153
Fargate considerations	154
Getting started with Fargate	155
Fargate profile	158
Fargate pod configuration	161
Fargate pod patching	163
Fargate metrics	164
Fargate logging	165
Instance types	172
Maximum pods	173
Amazon EKS optimized AMIs	174
Dockershim deprecation	175
Amazon Linux	175
Ubuntu Linux	205
Bottlerocket	205
Windows	211
Storage	229
Storage classes	229
Amazon EBS CSI driver	231
Create an IAM policy and role	231
Manage the Amazon EKS add-on	237
Deploy a sample application	242
Amazon EFS CSI driver	244
Create an IAM policy and role	244
Install the Amazon EFS driver	247
Create an Amazon EFS file system	249
Deploy a sample application	251

Amazon FSx for Lustre CSI driver	256
Amazon FSx for NetApp ONTAP CSI driver	262
Networking	263
Creating a VPC for Amazon EKS	265
Creating a VPC for Amazon EKS	266
VPC considerations	269
VPC IP addressing	270
Subnet tagging	272
Increase available IP addresses	272
Amazon EKS security group considerations	272
Cluster security group	272
Control plane and node security groups	273
Pod networking (CNI)	276
Configure plugin for IAM account	277
Managing VPC CNI add-on	281
Use cases	293
CNI metrics helper	324
Alternate compatible CNI plugins	329
Installing the AWS Load Balancer Controller add-on	329
Managing CoreDNS add-on	337
Adding the Amazon EKS add-on	338
Updating the Amazon EKS add-on	339
Removing the Amazon EKS add-on	341
Updating the self-managed add-on	342
Managing kube-proxy add-on	343
Adding the Amazon EKS add-on	344
Updating the Amazon EKS add-on	345
Removing the Amazon EKS add-on	347
Updating the self-managed add-on	348
Installing the Calico add-on	350
Install Calico	350
Stars policy demo	352
Remove Calico	358
Workloads	359
Sample application deployment	359
Vertical Pod Autoscaler	366
Deploy the Vertical Pod Autoscaler	366
Test your Vertical Pod Autoscaler installation	367
Horizontal Pod Autoscaler	370
Run a Horizontal Pod Autoscaler test application	370
Network load balancing	372
Create a network load balancer	374
(Optional) Deploy a sample application	376
Application load balancing	378
(Optional) Deploy a sample application	380
Restrict service external IP address assignment	382
Copy an image to a repository	384
Amazon container image registries	386
Amazon EKS add-ons	388
Add-on configuration	389
Machine learning training	391
Create node group	392
(Optional) Deploy a sample EFA compatible application	396
Machine learning inference	397
Prerequisites	397
Create a cluster	397
(Optional) Deploy a TensorFlow Serving application image	398

(Optional) Make predictions against your TensorFlow Serving service	400
Cluster authentication	402
Enabling user and role access	402
Add users or roles	403
Apply the <code>aws-authConfigMap</code> to your cluster	408
OIDC identity provider authentication	409
Associate an OIDC identity provider	410
Disassociate an OIDC identity provider from your cluster	412
Example IAM policy	412
Create a <code>kubeconfig</code> for Amazon EKS	413
Create <code>kubeconfig</code> file automatically	413
Create <code>kubeconfig</code> manually	414
Installing <code>aws-iam-authenticator</code>	417
Default Amazon EKS roles and users	420
Cluster management	422
Tutorial: Deploy Kubernetes Dashboard	422
Prerequisites	423
Step 1: Deploy the Kubernetes dashboard	423
Step 2: Create an <code>eks-admin</code> service account and cluster role binding	424
Step 3: Connect to the dashboard	425
Step 4: Next steps	426
Metrics server	426
Prometheus metrics	427
Viewing the raw metrics	427
Deploying Prometheus	428
Store your Prometheus metrics in Amazon Managed Service for Prometheus	430
Using Helm	430
Tagging your resources	431
Tag basics	432
Tagging your resources	432
Tag restrictions	433
Working with tags using the console	433
Working with tags using the CLI, API, or <code>eksctl</code>	434
Service quotas	435
Service quotas	436
Security	438
Certificate signing	439
CSR example	439
Kubernetes service accounts	440
Service account tokens	440
Cluster add-ons	441
IAM roles for service accounts	442
Identity and access management	455
Audience	455
Authenticating with identities	455
Managing access using policies	457
How Amazon EKS works with IAM	459
Identity-based policy examples	462
Using service-linked roles	465
Cluster IAM role	472
Node IAM role	474
Pod execution IAM role	477
Connector IAM role	480
AWS managed policies	483
Troubleshooting	498
Compliance validation	498
Resilience	499

Infrastructure security	500
Configuration and vulnerability analysis	500
Security best practices	501
Pod security policy	501
Amazon EKS default pod security policy	501
Delete default policy	502
Install or restore default policy	503
Managing Kubernetes secrets	504
Amazon EKS Connector considerations	504
AWS responsibilities	505
Customer responsibilities	505
View Kubernetes resources	506
Required permissions	506
Observability	511
Logging and monitoring	511
Amazon EKS monitoring and logging tools	512
Logging Amazon EKS API calls with AWS CloudTrail	514
Amazon EKS information in CloudTrail	514
Understanding Amazon EKS log file entries	514
Amazon EKS add-on support for ADOT Operator	516
ADOT considerations	517
AWS Distro for OpenTelemetry (ADOT) prerequisites	517
Create an IAM role	518
Manage the ADOT Operator	518
Deploy the ADOT Collector	521
Deploy a sample application	535
Working with other services	538
Creating Amazon EKS resources with AWS CloudFormation	538
Amazon EKS and AWS CloudFormation templates	538
Learn more about AWS CloudFormation	538
Use AWS App Mesh with Kubernetes	539
Amazon EKS on AWS Outposts	539
Prerequisites	539
Outpost considerations	539
.....	541
Deploy an Amazon EKS cluster with worker nodes on AWS Outposts	541
Amazon EKS on AWS Local Zones	545
Deep Learning Containers	545
Troubleshooting	546
IAM	554
Amazon EKS Connector Troubleshooting	556
Common issues	556
Frequently asked questions	559
Basic troubleshooting	559
ADOT Amazon EKS add-on Troubleshooting	561
Common issues	561
Amazon EKS Connector	563
Considerations	563
Required IAM permissions	563
Connecting a cluster	564
Step 1: Registering the cluster	564
Step 2: Applying the manifest file	566
Granting access to a user to view Kubernetes resources on a cluster	567
Prerequisites	567
Deregister a cluster	568
Related projects	570
Management tools	570

eksctl	570
AWS controllers for Kubernetes	570
Flux CD	570
CDK for Kubernetes	570
Networking	571
Amazon VPC CNI plugin for Kubernetes	571
AWS Load Balancer Controller for Kubernetes	571
ExternalDNS	571
App Mesh Controller	571
Security	571
AWS IAM authenticator	571
Machine learning	572
Kubeflow	572
Auto Scaling	572
Cluster autoscaler	572
Escalator	572
Monitoring	572
Prometheus	572
Continuous integration / continuous deployment	573
Jenkins X	573
Amazon EKS new features and roadmap	574
Document history	575

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.

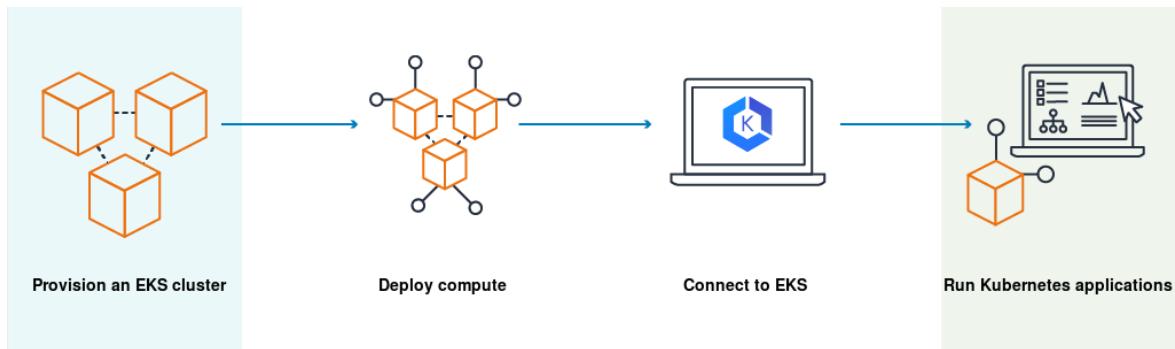
Amazon EKS control plane architecture

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 on AWS Outposts \(p. 539\)](#).

- **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 \(p. 16\)](#) – 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
```

- **Kubernetes 1.17**

```
curl -o kubectl https://s3.us-west-2.amazonaws.com/amazon-eks/1.17.12/2020-11-02/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
```

- **Kubernetes 1.17**

```
curl -o kubectl.sha256 https://s3.us-west-2.amazonaws.com/amazon-eks/1.17.12/2020-11-02/bin/darwin/amd64/kubectl.sha256
```

- a. Check the SHA-256 sum for your downloaded binary.

```
openssl sha1 -sha256 kubectl
```

- b. 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
```

- **Kubernetes 1.17**

```
curl -o kubectl https://s3.us-west-2.amazonaws.com/amazon-eks/1.17.12/2020-11-02/bin/linux/amd64/kubectl
```

```
curl -o kubectl https://s3.us-west-2.amazonaws.com/amazon-eks/1.17.12/2020-11-02/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 Linux using 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
```

- **Kubernetes 1.17**

```
curl -o kubectl.sha256 https://s3.us-west-2.amazonaws.com/amazon-eks/1.17.12/2020-11-02/bin/linux/amd64/kubectl.sha256
```

```
curl -o kubectl.sha256 https://s3.us-west-2.amazonaws.com/amazon-eks/1.17.12/2020-11-02/bin/linux/arm64/kubectl.sha256
```

- Check the SHA-256 sum for your downloaded binary.

```
openssl sha1 -sha256 kubectl
```

- Compare the generated SHA-256 sum in the command output against your downloaded SHA-256 file. The two should match.

- Apply execute permissions to the binary.

```
chmod +x ./kubectl
```

- 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
```

- (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
```

- After you install kubectl, you can verify its version with the following command:

```
kubectl version --short --client
```

Windows

To install kubectl on Windows

- Open a PowerShell terminal.
- 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
```

- **Kubernetes 1.17**

```
curl -o kubectl.exe https://s3.us-west-2.amazonaws.com/amazon-eks/1.17.12/2020-11-02/bin/windows/amd64/kubectl.exe
```

- (Optional) Verify the downloaded binary with the SHA-256 sum for your binary.

- 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
```

- **Kubernetes 1.17**

```
curl -o kubectl.exe.sha256 https://s3.us-west-2.amazonaws.com/amazon-eks/1.17.12/2020-11-02/bin/windows/amd64/kubectl.exe.sha256
```

- a. 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 installed on your computer or AWS CloudShell. The version must be the same, or up to two versions later than your cluster version. 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 \(p. 417\)](#), 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 0.97.0. 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/v0.97.0/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 0.97.0. 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/v0.97.0/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 0.97.0. 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/v0.97.0/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 \(p. 16\)](#).

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 0.97.0 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 \(p. 24\)](#) and [Amazon EKS nodes \(p. 105\)](#). 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 \(p. 105\)](#). 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 \(p. 140\)](#) and [Bottlerocket \(p. 138\)](#) 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
```

Output

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
```

Example output:

Fargate – Linux

NAME	VERSION	INTERNAL-IP	EXTERNAL-IP	OS-IMAGE	STATUS	ROLES	AGE
VERSION		CONTAINER-RUNTIME					KERNEL-
fargate-ip-192-168-141-147. <i>region-code</i> .compute.internal	v1.22.6-eks-7c9bda	192.168.141.147	<none>	containerd://1.3.2	Ready	<none>	Amazon Linux 2
5.4.156-83.273.amzn2.x86_64							
fargate-ip-192-168-164-53. <i>region-code</i> .compute.internal	v1.22.6-eks-7c9bda	192.168.164.53	<none>	containerd://1.3.2	Ready	<none>	Amazon Linux 2
7m30s	5.4.156-83.273.amzn2.x86_64						

Managed nodes – Linux

NAME	INTERNAL-IP	EXTERNAL-IP	OS-IMAGE	STATUS	ROLES	AGE	VERSION
CONTAINER-RUNTIME							KERNEL-VERSION
ip-192-168-12-49. <i>region-code</i> .compute.internal	v1.22.6-eks-d1db3c	192.168.12.49	52.35.116.65	Ready	<none>	6m7s	Amazon Linux 2
5.4.156-83.273.amzn2.x86_64							
ip-192-168-72-129. <i>region-code</i> .compute.internal	v1.22.6-eks-d1db3c	192.168.72.129	44.242.140.21	Ready	<none>	6m4s	Amazon Linux 2
5.4.156-83.273.amzn2.x86_64							

For more information about what you see in the output, see [View Kubernetes resources \(p. 506\)](#).

2. View the workloads running on your cluster.

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

Example output:

Fargate – Linux

NAMESPACE	NAME	READY	STATUS	RESTARTS	AGE	IP NOMINATED	NODE
	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 NOMINATED	NODE
	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>	<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>	<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>	<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>	<none>						
kube-system	kube-proxy-l8pb6	1/1	Running	0	7m46s		
	192.168.12.49		ip-192-168-12-49.region-code.compute.internal				<none>
<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>	<none>						

For more information about what you see in the output, see [View Kubernetes resources \(p. 506\)](#).

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. 359) 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 \(p. 402\)](#) and [Required permissions \(p. 506\)](#).

- Before deploying a cluster for production use, we recommend familiarizing yourself with all of the settings for [clusters \(p. 24\)](#) and [nodes \(p. 105\)](#). 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 \(p. 277\)](#).

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 \(p. 24\)](#). 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 **Amazon EKS 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-00x0000x000x000** | **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 \(p. 546\)](#).

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 \(kubectl\) \(p. 547\)](#) in the troubleshooting section.

Output

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

Step 3: 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 \(p. 105\)](#). Some settings can only be enabled when creating your nodes.

You can create a cluster with one of the following node types. To learn more about each type, see [Amazon EKS nodes \(p. 105\)](#). 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 \(p. 140\)](#) and [Bottlerocket \(p. 138\)](#) 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-arm 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 **Configuration** tab.
 - b. Choose the **Compute** tab.
 - c. 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 **Configuration** tab.

- b. Choose the **Compute** tab.
- c. 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 Amazon EKS **Clusters**. Then 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. **Overview** 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 \(p. 23\)](#).

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 Amazon EKS **Clusters**. In the list of clusters, choose *my-cluster*.
 - c. Choose the **Configuration** tab, and then 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 Amazon EKS **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 \(p. 402\)](#) and [Required permissions \(p. 506\)](#).
- Deploy a [sample application \(p. 359\)](#) to your cluster.
- Before deploying a cluster for production use, we recommend familiarizing yourself with all of the settings for [clusters \(p. 24\)](#) and [nodes \(p. 105\)](#). 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 \(p. 277\)](#).

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`, and `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 \(p. 263\)](#).
- For connected clusters, see [Amazon EKS Connector \(p. 563\)](#).

Creating an Amazon EKS cluster

This topic walks you through creating an Amazon EKS cluster. If this is your first time creating an Amazon EKS cluster, then we recommend that you follow one of our [Getting started with Amazon EKS \(p. 4\)](#) guides instead. They provide complete end-to-end walkthroughs for creating an Amazon EKS cluster with nodes.

To connect an external Kubernetes cluster to view in Amazon EKS, see [Amazon EKS Connector \(p. 563\)](#).

Important

- 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 (with `system:masters` permissions). Initially, only that IAM user can make calls to the Kubernetes API server using `kubectl`. For more information, see [Enabling IAM user and role access to your cluster \(p. 402\)](#). If you use the console to create the cluster, you must ensure that the same IAM user credentials are in the AWS SDK credential chain when you are running `kubectl` commands on your cluster.
- As a best practice, ensure that an IAM role is added to the `aws-auth` ConfigMap. This ensures that a cluster can be deleted after the creating user has been deleted. If you are in this situation and cannot delete a cluster, see [this article](#) to resolve the problem.

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

To deploy a new cluster on AWS Outposts, see [Deploy an Amazon EKS cluster with worker nodes on AWS Outposts \(p. 541\)](#).

(Optional) Create an IAM role to own your Amazon EKS cluster

The IAM security principal that creates your Amazon EKS cluster has permanent, full access to the Kubernetes API. We recommend creating a dedicated IAM role associated with the cluster to contain these permissions and to mitigate any authentication failures by allowing the role to be applied to another user.

Cloudformation

This procedure creates a role with sufficient permissions to create a cluster and defines a trust relationship to assume the role.

1. Copy the following contents to a file named `ClusterCreateRoleStack.yaml`:

```
AWSTemplateFormatVersion: "2010-09-09"
Resources:
  ClusterCreateRole:
    Type: 'AWS::IAM::Role'
    Properties:
      RoleName: "ClusterCreate"
      AssumeRolePolicyDocument:
        Version: "2012-10-17"
        Statement:
          - Effect: Allow
            Principal:
              AWS:
                - arn:aws:iam:: <your-account-id>:root
            Action:
              - 'sts:AssumeRole'
      Path: /
      ManagedPolicyArns:
        - arn:aws:iam::aws:policy/AdministratorAccess
```

2. Create the AWS CloudFormation stack for the role with the following command:

```
aws cloudformation create-stack \
  --stack-name ClusterCreateRoleStack \
  --template-body file://$(pwd)/ClusterCreateRoleStack.yaml \
  --capabilities CAPABILITY_NAMED_IAM
```

AWS Management Console

1. Open the IAM page in the [AWS Management Console](#).
2. Choose **Roles** under **Access Management** in the left sidebar, and then choose **Create Role**.
3. On the **Select trusted entity** page, do the following:
 - a. Select **AWS Account** as the Trusted Entity.
 - b. For the AWS account, choose **This account**.
4. On the **Add permissions** page, do the following:

- a. Select the **AdministratorAccess** policy.
- b. Leave the remaining settings at their default values and choose **Next**.
5. On the **Name, review, and create** page, add a name (for example, "ClusterCreate"), and choose **Create Role**.

After creating this role, you can now apply it to individual users who can all create and manage the same clusters.

eksctl

Prerequisite

Version 0.97.0 or later of the eksctl command line tool installed on your computer or AWS CloudShell. To install or update eksctl, see [Installing eksctl \(p. 10\)](#).

1. (Optional) Assume an IAM Role to own the cluster

The IAM entity (user or role) that creates a cluster is permanently added to the Kubernetes RBAC authorization table as the administrator with `system:masters` permissions. If you [created a dedicated IAM role \(p. 25\)](#), you can use that role in this step. These steps create an AWS CLI profile to automate assuming the role. For more information, see [.](#)

1. Insert a new profile for the role in the `~/.aws/config` file. This example command uses the `ClusterCreate` role created above:

```
[profile ClusterCreate]
role_arn = arn:aws:iam::<your-account-id>:role/ClusterCreate
source_profile = <default-profile>
```

2. Configure the AWS CLI to use the new profile. This dedicated IAM role will be used to create the cluster, and any user with this role can manage it:

```
export AWS_PROFILE=ClusterCreate
```

2. Create an Amazon EKS IPv4 cluster with the Amazon EKS latest Kubernetes version in your default Region. If you want to create an IPv6 cluster, you must deploy your cluster using a config file. For an example, see [Deploy an IPv6 cluster and nodes \(p. 295\)](#). Replace the `example-values` with your own values. You can replace `1.22` with any [supported version \(p. 67\)](#). 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.

```
eksctl create cluster \
--name my-cluster \
--version 1.22 \
--without-nodegroup
```

3. (Recommended) Add the `--with-oidc` flag to the previous command to automatically create an [AWS Identity and Access Management \(IAM\) OIDC provider \(p. 446\)](#) for your cluster. Creating the OIDC provider allows some Amazon EKS add-ons or your own individual Kubernetes workloads to have specific AWS Identity and Access Management (IAM) permissions. 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 \(p. 388\)](#). To learn more about assigning specific IAM permissions to your workloads, see [Technical overview \(p. 442\)](#).

Tip

To see most options that can be specified when creating a cluster with eksctl, use the `eksctl create cluster --help` command. To see all 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.

Important

If you plan to deploy self-managed nodes in AWS Outposts, AWS Wavelength, or AWS Local Zones after your cluster is deployed, you must have an existing VPC that meets Amazon EKS requirements and use the `--vpc-private-subnets` option with the previous command. The subnet IDs that you specify can't be the AWS Outposts, AWS Wavelength, or AWS Local Zones subnets. For more information about using an existing VPC, see [Use existing VPC: other custom configuration](#) in the `eksctl` documentation.

Warning

There is a `secretsEncryption` option that requires an existing AWS KMS key in AWS Key Management Service (AWS KMS). If you create a cluster using a config file with the `secretsEncryption` option and the KMS key that you use is ever deleted, then there is no path to recovery for the cluster. If you enable `secrets encryption`, the Kubernetes secrets are encrypted using the KMS key that you select. Create the symmetric encryption KMS key in the same Region as the cluster, and if the KMS key was created in a different account, the user must have access to the KMS key. Assymmetric KMS keys are not supported. For more information, see [Allowing users in other accounts to use a KMS key](#) in the [AWS Key Management Service Developer Guide](#).

By default, the `create-key` command creates a `symmetric key` with a key policy that gives the account root admin access on AWS KMS actions and resources. For more information, see [Creating keys](#). 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 will be calling the `create-cluster` API. Amazon EKS does not support the policy condition `kms:GrantIsForAWSResource`. Creating a cluster will not work if this action is in the KMS key policy statement.

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

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

After your cluster is created, you can migrate the Amazon VPC CNI, CoreDNS, and `kube-proxy` self-managed add-ons that were deployed with your cluster to Amazon EKS add-ons. For more information, see [Amazon EKS add-ons \(p. 388\)](#).

AWS Management Console

Prerequisites

- An existing VPC and a dedicated security group that meet the requirements for an Amazon EKS cluster. For more information, see [Cluster VPC and subnet considerations \(p. 269\)](#) and [Amazon EKS security group considerations \(p. 272\)](#). If you don't have a VPC, you can follow [Creating a VPC for your Amazon EKS cluster \(p. 265\)](#) to create one. If you want to assign IPv6 IP addresses to Pods and Services, then ensure that your VPC, subnets, and security groups meet the requirements and considerations listed in [Assigning IPv6 addresses to pods and services \(p. 294\)](#) or use the Amazon EKS public and private subnet AWS CloudFormation IPv6 VPC template to deploy an IPv6 VPC.
- An existing Amazon EKS cluster service IAM role. If you don't have the role, you can follow [Amazon EKS IAM roles \(p. 461\)](#) to create one.

To create your cluster with the console

1. (Optional) [Assume an appropriate role for creating the cluster](#). The IAM entity (user or role) that creates a cluster is permanently added to the Kubernetes RBAC authorization table as the

administrator with `system:masters` permissions. If you created a dedicated IAM role, you can use that role in this step. These steps create an AWS CLI profile to automate assuming the role. For more information, see <https://docs.aws.amazon.com/cli/latest/userguide/cli-configure-role.html>.

2. Open the Amazon EKS console at <https://console.aws.amazon.com/eks/home#/clusters>.
3. Choose **Create cluster**.
4. On the **Configure cluster** page, fill in the following fields:
 - **Name** – A unique name for your cluster.
 - **Kubernetes version** – The version of Kubernetes to use for your cluster.
 - **Cluster Service Role** – Choose the Amazon EKS cluster role to allow the Kubernetes control plane to manage AWS resources on your behalf. For more information, see the [Prerequisites \(p. 27\)](#).
 - **Secrets encryption** – (Optional) Choose to enable [secrets encryption](#) of Kubernetes secrets using a KMS key. The KMS key must be symmetric, created in the same region as the cluster, and 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](#).

If no keys are listed, you must create one first. For more information, see [Creating keys](#).

Note

By default, the `create-key` command creates a [symmetric key](#) with a key policy that gives the account root admin access on AWS KMS actions and resources. For more information, see [Creating keys](#). 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 will be calling the `create-cluster` API.

Amazon EKS does not support the policy condition `kms:GrantIsForAWSResource`. Creating a cluster will not work if this action is in the KMS key policy statement.

Warning

Deletion of the KMS key will permanently put the cluster in a degraded state. If any KMS keys used for cluster creation are scheduled for deletion, verify that this is the intended action before deletion. Once the KMS key is deleted, there is no path to recovery for the cluster. For more information, see [Deleting AWS KMS keys](#).

- **Tags** – (Optional) Add any tags to your cluster. For more information, see [Tagging your Amazon EKS resources \(p. 431\)](#).
5. Select **Next**.
6. On the **Specify networking** page, select values for the following fields:
 - **VPC** – Select an existing VPC to use for your cluster. If none are listed, then you need to create one first. For more information, see the [Prerequisites \(p. 27\)](#).
 - **Subnets** – By default, the available subnets in the VPC specified in the previous field are preselected. Unselect any subnet that you don't want to host cluster resources, such as worker nodes or load balancers. The subnets must meet the requirements for an Amazon EKS cluster. For more information, see [Cluster VPC and subnet considerations \(p. 269\)](#).

Important

- If you select subnets that were created before March 26, 2020 using one of the Amazon EKS AWS CloudFormation VPC templates, be aware of a default setting change that was introduced on March 26, 2020. For more information, see [Creating a VPC for your Amazon EKS cluster \(p. 265\)](#).
- Don't select subnets in AWS Outposts, AWS Wavelength or AWS Local Zones. If you plan to deploy self-managed nodes in AWS Outposts, AWS Wavelength or AWS Local Zones subnets after you deploy your cluster, then make sure that you have, or can create, Outposts subnets in the VPC that you select.

Security groups – The **SecurityGroups** value from the AWS CloudFormation output that you generated when you created your [VPC \(p. 265\)](#). This security group has **ControlPlaneSecurityGroup** in the dropdown name.

Important

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

- **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 version 1.21 or later for your cluster version, then you can choose whether Kubernetes will assign **IPv4** or **IPv6** addresses to Pods and Services. You can't change this option after cluster creation. If you choose IPv6, you can't choose the **Configure Kubernetes Service IP address range** option. Kubernetes assigns Service addresses from the unique local address range (`fc00 :: /7`). You can't specify a custom address range.
- (Optional) Choose **Configure Kubernetes Service IP address range** and specify a **Service IPv4 range** if you want to specify which CIDR block Kubernetes assigns service IP addresses from. The CIDR block must meet the following requirements:
 - Within one of the following ranges: `10.0.0.0/8`, `172.16.0.0/12`, or `192.168.0.0/16`.
 - Between `/24` and `/12`.
 - Doesn't overlap with any CIDR block specified in your VPC.

We recommend specifying a CIDR block that doesn't overlap with any other networks that are peered or connected to your VPC. If you don't enable this, Kubernetes assigns service IP addresses from either the `10.100.0.0/16` or `172.20.0.0/16` CIDR blocks.

Important

You can only specify a custom CIDR block when you create a cluster and can't change this value once the cluster is created.

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

Important

If you created a VPC without outbound internet access, then you must enable private access.

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

For more information about the previous options, see [Modifying cluster endpoint access \(p. 48\)](#).

7. You can accept the defaults in the **Networking add-ons** section to install the default version of the [AWS VPC CNI \(p. 276\)](#), [CoreDNS \(p. 337\)](#), and [kube-proxy \(p. 343\)](#) Amazon EKS add-ons, or 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, then you can remove Amazon EKS management of the add-on once your cluster is created. For more information, see [Amazon EKS add-ons \(p. 388\)](#).

Important

The AWS VPC CNI add-on is configured to use the IAM permissions assigned to the [Amazon EKS node IAM role \(p. 474\)](#). After the cluster is created, but before you deploy any Amazon EC2 nodes to your cluster, you must ensure that the `AmazonEKS_CNI_Policy` managed IAM policy (if using IPv4 for your cluster) or the `AmazonEKS_CNI_IPv6_Policy` IAM policy (that you [create yourself \(p. 281\)](#) if you're using IPv6 for your cluster) is attached to either the node IAM role, or to a different role associated to the Kubernetes service account that the add-on runs as. We recommend that you assign the policy to a different IAM role than the node IAM role by completing the instructions in [Configuring the Amazon VPC CNI plugin to use IAM roles for service accounts \(p. 277\)](#). Once your cluster and IAM role are created, you can [update the add-on \(p. 284\)](#) to use the IAM role that you create.

8. Select **Next**.
9. On the **Configure logging** page, you can optionally choose which log types that you want to enable. By default, each log type is **Disabled**. For more information, see [Amazon EKS control plane logging \(p. 64\)](#).
10. Select **Next**.
11. On the **Review and create** page, review the information that you entered or selected on the previous pages. Select **Edit** if you need to make changes to any of your selections. Once you're satisfied with your settings, select **Create**. The **Status** field shows **CREATING** until the cluster provisioning process completes.

Note

You might receive an error that one of the Availability Zones in your request doesn't have sufficient capacity to create an Amazon EKS cluster. If this happens, the error output contains the Availability Zones that can support a new cluster. Retry creating your cluster with at least two subnets that are located in the supported Availability Zones for your account. For more information, see [Insufficient capacity \(p. 546\)](#).

Cluster provisioning takes several minutes.

12. Follow the procedures in [Create a kubeconfig for Amazon EKS \(p. 413\)](#) to enable communication with your new cluster.
13. (Optional) To use some Amazon EKS add-ons, or to enable individual Kubernetes workloads to have specific AWS Identity and Access Management (IAM) permissions, you need to create an IAM OpenID Connect (OIDC) provider for your cluster. To create an IAM OIDC provider for your cluster, see [Create an IAM OIDC provider for your cluster \(p. 446\)](#). 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 \(p. 388\)](#). To learn more about assigning specific IAM permissions to your workloads, see [Technical overview \(p. 442\)](#).
14. If you're going to deploy Amazon EC2 nodes to your cluster, then you must attach one of the following policies to either your cluster IAM role, or to an IAM role that you create specifically for the Amazon VPC CNI add-on (this option requires the previous step). For more information about creating the role and configuring the add-on to use it, see [Configuring the Amazon VPC CNI plugin to use IAM roles for service accounts \(p. 277\)](#).
 - The `AmazonEKS_CNI_Policy` managed policy, if you created a 1.20 or earlier cluster or a 1.21 or later cluster with the IPv4 family.
 - [An IAM policy that you create \(p. 281\)](#), if you created a 1.21 or later cluster with the IPv6 family.
15. (Optional) Configure the VPC CNI add-on to use its own IAM role. This option requires the IAM OIDC provider created in a previous step and that you created an IAM role specifically for the add-on in the previous step. For more information, see [Updating the Amazon VPC CNI Amazon EKS add-on \(p. 284\)](#).

AWS CLI

Prerequisites

- 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 existing VPC and a dedicated security group that meet the requirements for an Amazon EKS cluster. For more information, see [Cluster VPC and subnet considerations \(p. 269\)](#) and [Amazon EKS security group considerations \(p. 272\)](#). If you don't have a VPC, you can follow [Creating a VPC for your Amazon EKS cluster \(p. 265\)](#) to create one. If you want to assign IPv6 IP addresses to Pods and Services, then ensure that your VPC, subnets, and security group meets the requirements listed in the considerations in [Assigning IPv6 addresses to pods and services \(p. 294\)](#).
- An existing Amazon EKS cluster IAM role. If you don't have the role, you can follow [Amazon EKS IAM roles \(p. 461\)](#) to create one.

To create your cluster with the AWS CLI

1. (Optional) [Assume an appropriate role for creating the cluster](#). The IAM entity (user or role) that creates a cluster is permanently added to the Kubernetes RBAC authorization table as the administrator with `system:masters` permissions. If you created a dedicated IAM role, you can use that role in this step. These steps create an AWS CLI profile to automate assuming the role. For more information, see <https://docs.aws.amazon.com/cli/latest/userguide/cli-configure-role.html>.
 1. Insert a new profile for the role in the `~/.aws/config` file. This example command uses the `ClusterCreate` role created above:

```
[profile ClusterCreate]
role_arn = arn:aws:iam::<your-account-id>:role/ClusterCreate
source_profile = <default-profile>
```

2. Configure the AWS CLI to use the new profile. This dedicated IAM role will be used to create the cluster, and any user with this role can manage it:

```
export AWS_PROFILE=ClusterCreate
```

2. Create your cluster with the following command. Replace the Amazon Resource Name (ARN) of your Amazon EKS cluster IAM role that you created in [Amazon EKS cluster IAM role \(p. 472\)](#) and the subnet and security group IDs for the VPC that you created in [Creating a VPC for your Amazon EKS cluster \(p. 265\)](#). Replace `my-cluster` with your cluster name. Replace `region-code` with the AWS Region that your cluster is in. You can replace `1.22` with any [supported version \(p. 67\)](#).

For `subnetIds`, don't specify subnets in AWS Outposts, AWS Wavelength or AWS Local Zones. If you plan to deploy self-managed nodes in AWS Outposts, AWS Wavelength or AWS Local Zones subnets after you deploy your cluster, then make sure that you have, or can create, Outposts subnets in the VPC that you specify.

If you 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 and specify 1.21 or later for `--kubernetes-version`.

```
aws eks create-cluster \
--region region-code \
--name my-cluster \
```

```
--kubernetes-version 1.22 \
--role-arn arn:aws:iam::111122223333:role/eksClusterRole \
--resources-vpc-config subnetIds=subnet-
a9189fe2,subnet-50432629,securityGroupIds=sg-f5c54184
```

Note

If your IAM user doesn't have administrative privileges, you must explicitly add permissions for that user to call the Amazon EKS API operations. For more information, see [Amazon EKS identity-based policy examples \(p. 462\)](#).

Output:

```
{
  "cluster": {
    "name": "my-cluster",
    "arn": "arn:aws:eks:region-code:111122223333:cluster/my-cluster",
    "createdAt": 1527785885.159,
    "version": "1.22",
    "roleArn": "arn:aws:iam::111122223333:role/eksClusterRole",
    "resourcesVpcConfig": {
      "subnetIds": [
        "subnet-a9189fe2",
        "subnet-50432629"
      ],
      "securityGroupIds": [
        "sg-f5c54184"
      ],
      "vpcId": "vpc-a54041dc",
      "endpointPublicAccess": true,
      "endpointPrivateAccess": false
    },
    "status": "CREATING",
    "certificateAuthority": {}
  }
}
```

Note

You might receive an error that one of the Availability Zones in your request doesn't have sufficient capacity to create an Amazon EKS cluster. If this happens, the error output contains the Availability Zones that can support a new cluster. Retry creating your cluster with at least two subnets that are located in the supported Availability Zones for your account. For more information, see [Insufficient capacity \(p. 546\)](#).

To encrypt the Kubernetes secrets, first create a KMS key using the [create-key](#) operation.

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

Note

By default, the [create-key](#) command creates a [symmetric encryption KMS key](#) that encrypts and decrypts data. This key has a key policy that gives the account's root user admin access on AWS KMS actions and resources. For more information, see [Creating keys](#). 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 will be calling the `create-cluster` API. Amazon EKS does not support the policy condition `kms:GrantIsForAWSResource`. Creating a cluster will not work if this action is in the KMS key policy statement.

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

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

The keyArn member can contain either the alias or ARN of your KMS key. The KMS key must be

- Symmetric
- Able to encrypt and decrypt data
- Created in the same 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

Deletion of the KMS key permanently puts the cluster in a degraded state. If any KMS keys used for cluster creation are scheduled for deletion, verify that this is the intended action before deletion. Once the KMS key is deleted, there is no path to recovery for the cluster. For more information, see [Deleting AWS KMS keys](#).

3. Cluster provisioning takes several minutes. You can query the status of your cluster with the following command. When your cluster status is ACTIVE, you can proceed.

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

4. Follow the procedures in [Create a kubeconfig for Amazon EKS \(p. 413\)](#) to enable communication with your new cluster.
5. (Optional) To use some Amazon EKS add-ons, or to enable individual Kubernetes workloads to have specific AWS Identity and Access Management (IAM) permissions, you need to create an IAM OpenID Connect (OIDC) provider for your cluster. To create an IAM OIDC provider for your cluster, see [Create an IAM OIDC provider for your cluster \(p. 446\)](#). 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 \(p. 388\)](#). To learn more about assigning specific IAM permissions to your workloads, see [Technical overview \(p. 442\)](#).
6. If you're going to deploy Amazon EC2 nodes to your cluster, then you must attach one of the following policies to either your cluster IAM role, or to an IAM role that you create specifically for the Amazon VPC CNI add-on (this option requires the previous step). For more information about creating the role and configuring the add-on to use it, see [Configuring the Amazon VPC CNI plugin to use IAM roles for service accounts \(p. 277\)](#).:
 - The `AmazonEKS_CNI_Policy` IAM managed policy, if you created a 1.20 or earlier cluster or a 1.21 or later cluster with the IPv4 family.
 - [An IAM policy that you create \(p. 281\)](#), if you created a 1.21 or later cluster with the IPv6 family.
7. (Optional) Migrate the Amazon VPC CNI, CoreDNS, and `kube-proxy` self-managed add-ons that were deployed with your cluster to Amazon EKS add-ons. For more information, see [Amazon EKS add-ons \(p. 388\)](#). Configure the Amazon EKS VPC CNI add-on to use its own IAM role. This option requires the IAM OIDC provider created in a previous step and that you created an IAM role specifically for the add-on in the previous step. For more information, see [Updating the Amazon VPC CNI Amazon EKS add-on \(p. 284\)](#).

Recommended next steps:

- Create a kubeconfig to access the API server ([p. 413](#))
- Add IAM users or roles to your Amazon EKS cluster ([p. 402](#))

Updating a cluster

You can update an existing Amazon EKS cluster to a new Kubernetes version or [secrets encryption](#) for your cluster.

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 \(p. 67\)](#) and also review in the update steps in this topic.

New Kubernetes versions have introduced 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 were provided when you created the cluster. If these subnets don't have available IP addresses, then the update can fail. Additionally, if any of the subnets or security groups that were provided when the cluster was created are deleted, the cluster update process might 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.

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 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 v1.22.

Before updating to 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 applicable, update load balance controller to v2.4.1 before updating to v1.22.

Important

If you update the 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.

The v1.22 release removes support from the following Beta APIs. Migrate your manifests and API clients with regard to the following information.

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

Resource	Beta version	GA version	Notes
			<p>spec.versions[*].additionalPrinterColumns instead.</p> <ul style="list-style-type: none"> • spec.conversion.webhookClientConfig is moved to spec.conversion.webhook.clientConfig in v1. • spec.conversion.conversionReviewVersion is moved to spec.conversion.webhook.conversionReviewVersion 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	alpha
TokenReview	authentication.k8s.io/v1beta1	authentication.k8s.io/v1	alpha
SubjectAccessReview ¹ LocalSubjectAccessReview ¹ 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	<ul style="list-style-type: none"> For API clients requesting certificates: <ul style="list-style-type: none"> <code>spec.signerName</code> is now required (see known Kubernetes signers), and requests for <code>kubernetes.io/legacy-unknown</code> are not allowed to be created via the certificates.k8s.io/v1 API <code>spec.usages</code> is now required, may not contain duplicate values, and must only contain known usages For API clients approving or signing certificates: <ul style="list-style-type: none"> <code>status.conditions</code> may not contain duplicate types <code>status.conditions[*].status</code> is now required <code>status.certificate</code> must be PEM-encoded, and contain only CERTIFICATE blocks
Lease	coordination.k8s.io/v1beta1	coordination.k8s.io/v1	None
Ingress	<ul style="list-style-type: none"> <code>extensions/v1beta1</code> <code>networking.k8s.io/v1beta1</code> 	networking.k8s.io/v1	<ul style="list-style-type: none"> <code>spec.backend</code> is renamed to <code>spec.defaultBackend</code> The backend <code>serviceName</code> field is renamed to <code>service.name</code> Numeric backend <code>servicePort</code> fields are renamed to <code>service.port.number</code> String backend <code>servicePort</code> fields are renamed to <code>service.port.name</code> <code>pathType</code> is now required for each specified path. Options are <code>Prefix</code>, <code>Exact</code>, and <code>ImplementationSpecific</code>. To match the undefined <code>v1beta1</code> behavior, use <code>ImplementationSpecific</code>
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](#).

To update the Kubernetes version for your Amazon EKS cluster

Update the Kubernetes version for your 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, 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 \(p. 119\)](#) and [Self-managed node updates \(p. 146\)](#). To update the version of a Fargate node, delete the pod that's represented by the node. Then, redeploy the pod after you update your control plane.

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 \(p. 501\)](#) before proceeding.

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

3. If you originally deployed your cluster on Kubernetes 1.17 or earlier, 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

- 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 1.11.0 before updating your cluster to version 1.21 or later. For more information, see [Updating the Amazon VPC CNI Amazon EKS add-on \(p. 284\)](#) or [Updating the Amazon VPC CNI self-managed add-on \(p. 287\)](#).

eksctl

This procedure requires `eksctl` version 0.97.0 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 \(p. 11\)](#).

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>` (including `<>`) 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 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 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"  
            }  
        ]  
    }  
}
```

```
    ],
...
}
} ]
```

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 \(p. 146\)](#) and [Updating a managed node group \(p. 119\)](#). 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.yaml
```

8. Update the VPC CNI, 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 \(p. 441\)](#).
 - If you updated your cluster to version 1.17 or earlier, refer to the instructions in [Updating the Amazon VPC CNI self-managed add-on \(p. 287\)](#), [Updating the CoreDNS self-managed add-on \(p. 342\)](#), and [Updating the kube-proxy self-managed add-on \(p. 348\)](#) to update your Amazon VPC CNI, CoreDNS, and kube-proxy add-ons.
 - 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 \(p. 282\)](#), [Adding the CoreDNS Amazon EKS add-on \(p. 338\)](#), and [Adding the kube-proxy Amazon EKS add-on \(p. 344\)](#). To learn more about Amazon EKS add-ons, see [Amazon EKS add-ons \(p. 388\)](#).
 - 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 **Configuration** tab, and then 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 \(p. 287\)](#), [CoreDNS \(p. 339\)](#), and [kube-proxy \(p. 345\)](#) Amazon EKS add-ons.

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 **Configuration** tab.
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. Amazon EKS doesn't support the policy condition `kms:GrantIsForAWSResource`. If this action is in the KMS key policy statement, creating a cluster doesn't work.

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 \(p. 568\)](#)

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 0.97.0 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 \(p. 11\)](#).

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, select the name of the cluster that you want to delete.
 - c. Select the **Configuration** tab. On the **Compute** tab, select a node group to delete, select **Delete**, enter the name of the node group, and then select **Delete**. Delete all node groups in the cluster.
- Note**
The node groups listed are [managed node groups \(p. 109\)](#) only.

 - d. Select a **Fargate Profile** to delete, select **Delete**, enter the name of the profile, and then select **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. Select 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. Select 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 \(p. 109\)](#) 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	<ul style="list-style-type: none"> 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. 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	<ul style="list-style-type: none"> Kubernetes API requests within your cluster's VPC (such as node to control plane communication) use the private VPC endpoint.

Endpoint public access	Endpoint private access	Behavior
		<ul style="list-style-type: none"> Your cluster API server is accessible from the internet. You can, optionally, limit the CIDR blocks that can access the public endpoint.
Disabled	Enabled	<ul style="list-style-type: none"> 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 <code>kubectl</code> commands must come from within the VPC or a connected network. For connectivity options, see Accessing a private only API server (p. 52). 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. <p>If your endpoint does not resolve to a private IP address within the VPC for an existing cluster, you can:</p> <ul style="list-style-type: none"> 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. 34) 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 **Configuration** tab. On the **Networking** tab, 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 \(p. 435\)](#). 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 \(p. 435\)](#). 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>
```

Output:

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

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

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

Output:

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

Accessing a private only API server

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

- **Connected network** – Connect your network to the VPC with an [AWS transit gateway](#) or other [connectivity](#) option and then use a computer in the connected network. You must ensure that your Amazon EKS control plane security group contains rules to allow ingress traffic on port 443 from your connected network.
- **Amazon EC2 bastion host** – You can launch an Amazon EC2 instance into a public subnet in your cluster's VPC and then log in via SSH into that instance to run `kubectl` commands. For more information, see [Linux bastion hosts on AWS](#). You must ensure that your Amazon EKS control plane security group contains rules to allow ingress traffic on port 443 from your bastion host. For more information, see [Amazon EKS security group considerations \(p. 272\)](#).

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 \(p. 402\)](#) and [Unauthorized or access denied \(`kubectl`\) \(p. 547\)](#).

- **AWS Cloud9 IDE** – AWS Cloud9 is a cloud-based integrated development environment (IDE) that lets you write, run, and debug your code with just a browser. You can create an AWS Cloud9 IDE in your cluster's VPC and use the IDE to communicate with your cluster. For more information, see [Creating an environment in AWS Cloud9](#). You must ensure that your Amazon EKS control plane security group contains rules to allow ingress traffic on port 443 from your IDE security group. For more information, see [Amazon EKS security group considerations \(p. 272\)](#).

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 \(p. 402\)](#) and [Unauthorized or access denied \(`kubectl`\) \(p. 547\)](#).

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 \(p. 109\)](#) 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 \(p. 24\)](#).
- 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 \(p. 446\)](#).
- 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
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 \(p. 474\)](#) 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 under **Configuration Details** 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 \(p. 57\)](#) 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
```

Example output:

```
I0926 23:15:55.165842      1 static_autoscaler.go:138] Starting main loop
I0926 23:15:55.166279      1 utils.go:595] No pod using affinity / antiaffinity found in
cluster, disabling affinity predicate for this loop
I0926 23:15:55.166293      1 static_autoscaler.go:294] Filtering out schedulables
I0926 23:15:55.166330      1 static_autoscaler.go:311] No schedulable pods
I0926 23:15:55.166338      1 static_autoscaler.go:319] No unschedulable pods
I0926 23:15:55.166345      1 static_autoscaler.go:366] Calculating unneeded nodes
I0926 23:15:55.166357      1 utils.go:552] Skipping ip-192-168-3-111.<region-
code>.compute.internal - node group min size reached
I0926 23:15:55.166365      1 utils.go:552] Skipping ip-192-168-71-83.<region-
code>.compute.internal - node group min size reached
I0926 23:15:55.166373      1 utils.go:552] Skipping ip-192-168-60-191.<region-
code>.compute.internal - node group min size reached
I0926 23:15:55.166435      1 static_autoscaler.go:393] Scale down status:
unneededOnly=false lastScaleUpTime=2019-09-26 21:42:40.908059094 ...
I0926 23:15:55.166458      1 static_autoscaler.go:403] Starting scale down
I0926 23:15:55.166488      1 scale_down.go:706] No candidates for scale down
```

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 \(p. 109\)](#). 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 unnecessary 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: $LABEL_VALUE
Key: k8s.io/cluster-autoscaler/node-template/taint/$TINT_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 \(p. 24\)](#).
- 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 \(p. 446\)](#).
- A user or role with permission to create a cluster.
- AWS CLI
- [Installing kubectl \(p. 4\)](#)
- [Using Helm with Amazon EKS \(p. 430\)](#)

You can deploy Karpenter using eksctl if you prefer. See [Installing eksctl \(p. 10\)](#).

Amazon EKS control plane logging

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

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

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

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

- **Kubernetes API server component logs (api)** – Your cluster's API server is the control plane component that exposes the Kubernetes API. For more information, see [kube-apiserver](#) in the Kubernetes documentation.

- **Audit (audit)** – Kubernetes audit logs provide a record of the individual users, administrators, or system components that have affected your cluster. For more information, see [Auditing](#) in the Kubernetes documentation.
- **Authenticator (authenticator)** – Authenticator logs are unique to Amazon EKS. These logs represent the control plane component that Amazon EKS uses for Kubernetes [Role Based Access Control](#) (RBAC) authentication using IAM credentials. For more information, see [Cluster management \(p. 422\)](#).
- **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. Select the **Configuration** tab.
4. Under **Logging**, choose **Manage logging**.
5. For each individual log type, choose whether the log type should be **Enabled** or **Disabled**. By default, each log type is **Disabled**.
6. 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}]}'
```

Output:

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

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

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

Output:

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

Viewing cluster control plane logs

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

To learn more about viewing, analyzing, and managing logs in CloudWatch, see the [Amazon CloudWatch Logs User Guide](#).

To view your cluster control plane logs on the CloudWatch console

1. Open the [CloudWatch console](#). 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...>

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:

- 1.22.6
- 1.21.9
- 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 a cluster \(p. 34\)](#). For more information about Kubernetes releases, see [Amazon EKS Kubernetes release calendar \(p. 74\)](#) and [Amazon EKS version support and FAQ \(p. 74\)](#).

Note

Starting with the Kubernetes version 1.23 launch, officially published Amazon EKS AMIs will include containerd as the only runtime. Kubernetes version 1.18–1.21 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 \(p. 175\)](#).

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 v1.22. Follow the [Kubernetes version 1.22 prerequisites \(p. 34\)](#) 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 v0.15.7 and later
- Python v12.0.0 and later
- Java v9.0.0 and later
- JavaScript v0.10.3 and later
- Ruby master branch
- Haskell v0.3.0.0
- C# v7.0.5 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 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 \(p. 441\)](#).

- 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 v1.19](#). For more information about changes between Ingress v1beta1 and v1, see the [Kubernetes documentation](#). The AWS Load Balancer [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 v1.22.
- The [CertificateSigningRequest \(CSR\)](#) API version certificates.k8s.io/v1beta1 was removed in Kubernetes 1.22. You must migrate manifests and API clients to use the certificates.k8s.io/v1 CSR API. This API has been available since v1.19. For instructions on how to use CSR in Amazon EKS, see [Certificate signing \(p. 439\)](#).
- 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 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 v1.22 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 v1.22.
- 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 v1.22 uses etcd v3.4 as a backend, and is not affected by the [possibility of data corruption](#) present in etcd v3.5.

- 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 STS endpoint used by IAM Role 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 \(p. 450\)](#).

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 v0.15.7 and later
 - Python v12.0.0 and later
 - Java v9.0.0 and later
 - JavaScript v0.10.3 and later
 - Ruby master branch
 - Haskell v0.3.0.0
 - C# v7.0.5 and laterIf 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 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 \(p. 441\)](#).

- [Dual-stack networking](#) support (IPv4 and IPv6 addresses) on pods, services, and nodes reached beta status. However, Amazon EKS and the Amazon VPC CNI 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 \(p. 184\)](#). 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 v1.21 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 Kubernetes 1.21 Amazon EKS 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. EKS will eventually move to `containerd` as the runtime for the 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 considerations" \(p. 269\)](#).
 - Subnet tags aren't modified on existing clusters updated to 1.19.
 - The AWS Load Balancer Controller version v2.1.1 and earlier required the `<cluster-name>` subnet tag. In version v2.1.2 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 \(p. 329\)](#). For more information about subnet tagging when using a load balancer, see [Application load balancing on Amazon EKS \(p. 378\)](#) and [Network load balancing on Amazon EKS \(p. 372\)](#).
- 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 \(p. 442\)](#) and [proposal 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 \(p. 337\)](#).
- 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 \(p. 187\)](#).
- 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 be 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
<code>eks:certificate-controller</code>	Rolebinding	<code>kube-system</code>	Impacts signer and approver functionality in the control plane.
<code>eks:certificate-controller</code>	Role	<code>kube-system</code>	Impacts signer and approver functionality in the control plane.
<code>eks:certificate-controller</code>	ClusterRolebinding	All	Impacts kubelet's ability to request server certificates, which affects certain cluster functionality like <code>kubectl exec</code> and <code>kubectl logs</code> .

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 \(p. 378\)](#) (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>.

Kubernetes 1.17

For more information about Kubernetes 1.17, see the [official release announcement](#).

Important

- The `CSIMigrationAWS` feature flag isn't enabled on EKS. It will be enabled in a future release. Detailed migration instructions will also be provided at that time. For more information about CSI migration, see the [Kubernetes blog](#).

The following Kubernetes features are now supported in Kubernetes 1.17 Amazon EKS clusters:

- [Cloud Provider Labels](#) have reached general availability. If you're using the beta labels in your pod specs for features such as node affinity, or in any custom controllers, then we recommend that you start migrating them to the new GA labels. For information about the new labels, see the following Kubernetes documentation:
 - `node.kubernetes.io/instance-type`
 - `topology.kubernetes.io/region`
 - `topology.kubernetes.io/zone`

- The [ResourceQuotaScopeSelectors](#) feature has graduated to generally available. You can use this feature to limit the number of resources a quota supports to only those that pertain to the scope.
- The [TaintNodesByCondition](#) feature has graduated to generally available. This feature allows you to taint nodes that have conditions such as high disk or memory pressure.
- The [CSI Topology](#) feature has graduated to generally available, and is fully supported by the [EBS CSI driver](#). You can use topology to restrict the Availability Zone where a volume is provisioned.
- [Finalizer protection](#) for services of type `LoadBalancer` has graduated to generally available. This feature ensures that a service resource isn't fully deleted until the correlating load balancer is also deleted.
- Custom resources now support [default values](#). You specify values in an [OpenAPI v3 validation schema](#).
- The [Windows containers RunAsUsername](#) feature is now in beta. You can use it to run Windows applications in a container as a different user name than the default.

For the complete Kubernetes 1.17 changelog, see <https://github.com/kubernetes/kubernetes/blob/master/CHANGELOG/CHANGELOG-1.17.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
1.17	December 9, 2019	July 10, 2020	November 2, 2021
1.18	March 23, 2020	October 13, 2020	March 31, 2022
1.19	August 26, 2020	February 16, 2021	June 30, 2022
1.20	December 8, 2020	May 18, 2021	September 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 "To update the Kubernetes version for your Amazon EKS cluster" \(p. 38\)](#).

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 a cluster" \(p. 34\)](#).

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 on the **Configuration** 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" \(p. 119\)](#). 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" \(p. 146\)](#).

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 \(p. 438\)](#). 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
1.22.6	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 1.22 for Amazon EKS. For more information, see Kubernetes 1.22 (p. 67) .	April 4, 2022

Kubernetes version 1.21

Kubernetes version	Amazon EKS platform version	Enabled admission controllers	Release notes	Release date
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	April 8, 2022

Kubernetes version	Amazon EKS platform version	Enabled admission controllers	Release notes	Release date
			Security Token Service endpoint for a service account (p. 452) .	
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. 442) , 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
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.	

Kubernetes version	Amazon EKS platform version	Enabled admission controllers	Release notes	Release date
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.	
1.21.2	eks.1	NamespaceLifecycle, LimitRanger, ServiceAccount, DefaultStorageClass, ResourceQuota, DefaultTolerationSeconds, NodeRestriction, MutatingAdmissionWebhook, ValidatingAdmissionWebhook, PodSecurityPolicy, TaintNodesByCondition, Priority, StorageObjectInUseProtection, PersistentVolumeClaimResize, ExtendedResourceToleration	Initial release of Kubernetes 1.21 for Amazon EKS. For more information, see Kubernetes 1.21 (p. 69) .	

Kubernetes version 1.20

Kubernetes version	Amazon EKS platform version	Enabled admission controllers	Release notes	Release date
1.20.15	eks.5	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. 452) .	April 8, 2022
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. 442) , 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
1.20.11	eks.3	NamespaceLifecycle, LimitRanger, ServiceAccount, DefaultStorageClass, ResourceQuota, DefaultTolerationSeconds, NodeRestriction,	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	

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	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 1.20 for Amazon EKS. For more information, see Kubernetes 1.20 (p. 70) .	

Kubernetes version 1.19

Kubernetes version	Amazon EKS platform version	Enabled admission controllers	Release notes	Release date
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. 452) .	April 8, 2022
1.19.15	eks.8	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. 442) , 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.9 however. An updated Fargate scheduler provisions nodes at a significantly higher rate during large deployments.	March 10, 2022
1.19.15	eks.7	NamespaceLifecycle, LimitRanger, ServiceAccount, DefaultStorageClass, ResourceQuota, DefaultTolerationSeconds, NodeRestriction,	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	

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	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.	
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.	

Kubernetes version	Amazon EKS platform version	Enabled admission controllers	Release notes	Release date
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.	
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.	

Kubernetes version	Amazon EKS platform version	Enabled admission controllers	Release notes	Release date
1.19.6	eks.1	NamespaceLifecycle, LimitRanger, ServiceAccount, DefaultStorageClass, ResourceQuota, DefaultTolerationSeconds, NodeRestriction, MutatingAdmissionWebhook, ValidatingAdmissionWebhook, PodSecurityPolicy, TaintNodesByCondition, Priority, StorageObjectInUseProtection, PersistentVolumeClaimResize, ExtendedResourceToleration	Initial release of Kubernetes 1.19 for Amazon EKS. For more information, see Kubernetes 1.19 (p. 71) .	

Kubernetes version 1.18

Kubernetes version	Amazon EKS platform version	Enabled admission controllers	Release notes	Release date
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. 452) .	April 8, 2022
1.18.20	eks.10	NamespaceLifecycle, LimitRanger, ServiceAccount, DefaultStorageClass, ResourceQuota, DefaultTolerationSeconds,	When using IAM roles for service accounts (p. 442) , 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
		NodeRestriction, MutatingAdmissionWebhook, ValidatingAdmissionWebhook, PodSecurityPolicy, TaintNodesByCondition, StorageObjectInUseProtection, PersistentVolumeClaimResize, CertificateApproval, PodPriority, CertificateSigning, CertificateSubjectRestriction, RuntimeClass, DefaultIngressClass	the global endpoint in eks.11 however. An updated Fargate scheduler provisions nodes at a significantly higher rate during large deployments.	
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.	

Kubernetes version	Amazon EKS platform version	Enabled admission controllers	Release notes	Release date
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.	
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.	

Kubernetes version	Amazon EKS platform version	Enabled admission controllers	Release notes	Release date
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.	
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. 388) and Fargate logging (p. 165) .	
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.	

Kubernetes version	Amazon EKS platform version	Enabled admission controllers	Release notes	Release date
1.18.8	eks.1	NamespaceLifecycle, LimitRanger, ServiceAccount, DefaultStorageClass, ResourceQuota, DefaultTolerationSeconds, NodeRestriction, MutatingAdmissionWebhook, ValidatingAdmissionWebhook, PodSecurityPolicy, TaintNodesByCondition, Priority, StorageObjectInUseProtection, PersistentVolumeClaimResize	Initial release of Kubernetes 1.18 for Amazon EKS. For more information, see Kubernetes 1.18 (p. 73) .	

Kubernetes version 1.17

Kubernetes version	Amazon EKS platform version	Enabled admission controllers	Release notes
1.17.17	eks.10	NamespaceLifecycle, LimitRanger, ServiceAccount, DefaultStorageClass, ResourceQuota, DefaultTolerationSeconds, NodeRestriction, MutatingAdmissionWebhook, ValidatingAdmissionWebhook, PodSecurityPolicy, TaintNodesByCondition, StorageObjectInUseProtection, PersistentVolumeClaimResize, PodPriority, RuntimeClass	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.
1.17.17	eks.9	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
1.17.17	eks.8	NamespaceLifecycle , LimitRanger , ServiceAccount , DefaultStorageClass , ResourceQuota , DefaultTolerationSeconds , NodeRestriction , MutatingAdmissionWebhook , ValidatingAdmissionWebhook , PodSecurityPolicy , TaintNodesByCondition , Priority , StorageObjectInUseProtection , PersistentVolumeClaimResize	New platform version with security fixes and enhancements.
1.17.17	eks.7	NamespaceLifecycle , LimitRanger , ServiceAccount , DefaultStorageClass , ResourceQuota , DefaultTolerationSeconds , NodeRestriction , MutatingAdmissionWebhook , ValidatingAdmissionWebhook , PodSecurityPolicy , TaintNodesByCondition , Priority , StorageObjectInUseProtection , PersistentVolumeClaimResize	New platform version with security fixes and enhancements.
1.17.12	eks.6	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
1.17.12	eks.5	NamespaceLifecycle, LimitRanger, ServiceAccount, DefaultStorageClass, ResourceQuota, DefaultTolerationSeconds, NodeRestriction, MutatingAdmissionWebhook, ValidatingAdmissionWebhook, PodSecurityPolicy, TaintNodesByCondition, Priority, StorageObjectInUseProtection, PersistentVolumeClaimResize	Includes support for Fargate logging (p. 165) .
1.17.12	eks.4	NamespaceLifecycle, LimitRanger, ServiceAccount, DefaultStorageClass, ResourceQuota, DefaultTolerationSeconds, NodeRestriction, MutatingAdmissionWebhook, ValidatingAdmissionWebhook, PodSecurityPolicy, TaintNodesByCondition, Priority, StorageObjectInUseProtection, PersistentVolumeClaimResize	New platform version with security fixes and enhancements.
1.17.9	eks.3	NamespaceLifecycle, LimitRanger, ServiceAccount, DefaultStorageClass, ResourceQuota, DefaultTolerationSeconds, NodeRestriction, MutatingAdmissionWebhook, ValidatingAdmissionWebhook, PodSecurityPolicy, TaintNodesByCondition, Priority, StorageObjectInUseProtection, PersistentVolumeClaimResize	New platform version with support for Security groups for pods (p. 313) . This release creates a vpc-resource-controller service account that is required for the VPC resource controller.

Kubernetes version	Amazon EKS platform version	Enabled admission controllers	Release notes
1.17.9	eks.2	NamespaceLifecycle, LimitRanger, ServiceAccount, DefaultStorageClass, ResourceQuota, DefaultTolerationSeconds, NodeRestriction, MutatingAdmissionWebhook, ValidatingAdmissionWebhook, PodSecurityPolicy, TaintNodesByCondition, Priority, StorageObjectInUseProtection, PersistentVolumeClaimResize	New platform version with security fixes and enhancements, including UDP support for services of type LoadBalancer when using NLB and support for using Amazon EFS volumes with Fargate pods. For more information, see the Allow UDP for AWS NLB issue on GitHub.
1.17.6	eks.1	NamespaceLifecycle, LimitRanger, ServiceAccount, DefaultStorageClass, ResourceQuota, DefaultTolerationSeconds, NodeRestriction, MutatingAdmissionWebhook, ValidatingAdmissionWebhook, PodSecurityPolicy, TaintNodesByCondition, Priority, StorageObjectInUseProtection, PersistentVolumeClaimResize	Initial release of Kubernetes 1.17 for Amazon EKS. For more information, see Kubernetes 1.17 (p. 73) .

Windows support

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 are not 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 \(p. 313\)](#) with pods running on Windows nodes.
- You can't use [custom networking \(p. 306\)](#) with Windows nodes.
- You can't use [IP prefixes \(p. 310\)](#) with Windows nodes. This is a requirement for using [IPv6 \(p. 294\)](#), 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 \(p. 140\)](#).
- 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 \(p. 96\)](#) 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 \(p. 94\)](#) and [enable Windows support \(p. 93\)](#) 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
1.17	eks.10

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 \(p. 472\)](#).

Enabling Windows support

If your cluster is not at, or later, than one of the Kubernetes and platform versions listed in the [Prerequisites \(p. 93\)](#), you must enable legacy Windows support instead. For more information, see [Enabling legacy Windows support \(p. 96\)](#).

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 \(p. 93\)](#), then you must first [remove the vpc-resource-controller and vpc-admission-webhook from your data plane \(p. 94\)](#). 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 \(p. 472\)](#). Replace `eksClusterRole` with your cluster role name.

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

Example output:

```
{  
    "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 \(p. 472\)](#). 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 \(p. 93\)](#), 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 \(p. 93\)](#) 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 \(p. 313\)](#) with them, then skip this step.

Remove the `AmazonVPCResourceController` managed IAM policy from your [cluster role \(p. 472\)](#). 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 \(p. 359\)](#) 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 \(p. 93\)](#), then we recommend that you enable Windows support on your control plane instead. For more information, see [Enabling Windows support \(p. 93\)](#).

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 \(p. 93\)](#). Once your cluster and platform version are at, or later than a version listed in the [Prerequisites \(p. 93\)](#), we recommend that you [remove legacy Windows support \(p. 94\)](#) and [enable it for your control plane \(p. 93\)](#).

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 0.97.0 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 \(p. 11\)](#).

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 \(p. 99\)](#).

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 \(p. 140\)](#).

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 \(p. 99\)](#).

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 \(p. 140\)](#).

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 \(p. 99\)](#).

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 \(p. 140\)](#).

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
```

Example output:

```
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.

Viewing API server flags

You can use the control plane logging feature for Amazon EKS clusters to view the API server flags that were enabled when a cluster was created. For more information, see [Amazon EKS control plane logging \(p. 64\)](#). 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. Select the **Configuration** tab. 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.

Private clusters

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 \(p. 104\)](#).
- 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 \(p. 47\)](#).
- 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 \(p. 227\)](#).

- 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 \(p. 402\)](#).

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. <i>region-code</i> .ecr.api and com.amazonaws. <i>region-code</i> .ecr.dkr
Application Load Balancers and Network Load Balancers	com.amazonaws. <i>region-code</i> .elasticloadbalancing
AWS X-Ray	com.amazonaws. <i>region-code</i> .xray
Amazon CloudWatch Logs	com.amazonaws. <i>region-code</i> .logs
IAM roles for service accounts (p. 442)	com.amazonaws. <i>region-code</i> .sts
App Mesh	com.amazonaws. <i>region-code</i> .appmesh-envoy-management
<ul style="list-style-type: none"> 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. Cluster Autoscaler (p. 52) is supported. When deploying Cluster Autoscaler pods, make sure that the command line includes <code>--aws-use-static-instance-list=true</code>. 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 \(p. 244\)](#), 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 \(p. 132\)](#) 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 \(p. 256\)](#) isn't supported.
- [AWS Fargate \(p. 153\)](#) is supported with private clusters. You can use the [AWS Load Balancer Controller \(p. 329\)](#) 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 \(p. 378\)](#) and [Create a network load balancer \(p. 374\)](#).
- [Installing the AWS Load Balancer Controller add-on \(p. 329\)](#) 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.

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 \(p. 442\)](#) 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: ''
value: region-code
- name: AWS_STS_REGIONAL_ENDPOINTS
value: regional
...  
...
```

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

Amazon EKS nodes

Your Amazon EKS cluster can schedule pods on any combination of [Self-managed nodes \(p. 132\)](#), [Amazon EKS Managed node groups \(p. 109\)](#), and [AWS Fargate \(p. 153\)](#). To learn more about nodes deployed in your cluster, see [View Kubernetes resources \(p. 506\)](#).

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 \(p. 563\)](#) 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 on AWS Outposts (p. 539) .	No
Can be deployed to AWS Local Zones	No	Yes – For more information, see Amazon EKS on AWS Local Zones (p. 545) .	No
Can run containers that require Windows	No	Yes (p. 92) – 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. 397) – Amazon Linux nodes only	Yes (p. 397) – Amazon Linux only	No
Can run workloads that require a GPU	Yes (p. 185) – Amazon Linux nodes only	Yes (p. 185) – Amazon Linux only	No
Can run workloads that require Arm processors	Yes (p. 186)	Yes (p. 186)	No
Can run AWS Bottlerocket	Yes	Yes (p. 138)	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. 112) – 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. 133) , Linux (Arm) (p. 186) , or Windows (p. 92) 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. 124) 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. 124) with a custom AMI	Yes, using CNI custom networking (p. 306) .	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. 124)	Yes	No
Can deploy your own custom CNI to nodes	Yes – Using a launch template (p. 124) with a custom AMI	Yes	No
Must update node AMI on your own	Yes (p. 119) – 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. 151) – 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. 119) – 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. 151) – 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. 231)	Yes (p. 231)	No
Can use Amazon EFS storage with pods	Yes (p. 244)	Yes (p. 244)	Yes (p. 244)
Can use Amazon FSx for Lustre storage with pods	Yes (p. 256)	Yes (p. 256)	No
Can use Network Load Balancer for services	Yes (p. 372)	Yes (p. 372)	Yes, when using the Create a network load balancer (p. 374)
Pods can run in a public subnet	Yes	Yes	No
Can assign different VPC security groups to individual pods	Yes (p. 313) – Linux nodes only	Yes (p. 313) – Linux nodes only	Yes, in 1.18 or later clusters
Can run Kubernetes DaemonSets	Yes	Yes	No
Support <code>HostPort</code> and <code>HostNetwork</code> 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. 153)
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 \(p. 16\)](#).

To add a managed node group to an existing cluster, see [Creating a managed node group \(p. 112\)](#).

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 \(p. 52\)](#).

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 \(p. 52\)](#), you should

configure multiple node groups, each scoped to a single Availability Zone. In addition, you should enable the `--balance-similar-node-groups` feature.

- You can use a custom launch template for a greater level of flexibility and customization when deploying managed nodes. If you deploy using a launch template, you can also use a custom AMI. For more information, see [Launch template support \(p. 124\)](#). If you don't use a custom launch template when first creating a managed node group, there is an auto-generated launch template. Don't manually modify this auto-generated template or errors occur.
- Amazon EKS follows the shared responsibility model for CVEs and security patches on managed node groups. When managed nodes run an Amazon EKS optimized AMI, Amazon EKS is responsible for building patched versions of the AMI when bugs or issues are reported. We can publish a fix. However, you're responsible for deploying these patched AMI versions to your managed node groups. When managed nodes run a custom AMI, you're responsible for building patched versions of the AMI when bugs or issues are reported and then deploying the AMI. For more information, see [Updating a managed node group \(p. 119\)](#).
- Amazon EKS managed node groups can be launched in both public and private subnets. If you launch a managed node group in a public subnet on or after April 22, 2020, the subnet must have `MapPublicIpOnLaunch` set to true for the instances to successfully join a cluster. If the public subnet was created using `eksctl` or the [Amazon EKS vended AWS CloudFormation templates \(p. 265\)](#) on or after March 26, 2020, then this setting is already set to true. If the public subnets were created before March 26, 2020, you must change the setting manually. For more information, see [Modifying the public IPv4 addressing attribute for your subnet](#).
- When using VPC endpoints in private subnets, you must create endpoints for `com.amazonaws.region.ecr.api`, `com.amazonaws.region.ecr.dkr`, and a gateway endpoint for Amazon S3. For more information, see [Amazon ECR interface VPC endpoints \(AWS PrivateLink\)](#).
- Managed node groups can't be deployed on [AWS Outposts \(p. 539\)](#) or in AWS Wavelength or AWS Local Zones.
- You can create multiple managed node groups within a single cluster. For example, you can create one node group with the standard Amazon EKS optimized Amazon Linux 2 AMI for some workloads and another with the GPU variant for workloads that require GPU support.
- If your managed node group encounters an [Amazon EC2 instance status check](#) failure, Amazon EKS returns an error message to help you to diagnose the issue. For more information, see [Managed node group errors \(p. 548\)](#).
- Amazon EKS adds Kubernetes labels to managed node group instances. These Amazon EKS provided labels are prefixed with `eks.amazonaws.com`.
- Amazon EKS automatically drains nodes using the Kubernetes API during terminations or updates. Updates respect the pod disruption budgets that you set for your pods.
- There are no additional costs to use Amazon EKS managed node groups. You only pay for the AWS resources that you provision.
- If you want to encrypt Amazon EBS volumes for your nodes, you can deploy the nodes using a launch template. To deploy managed nodes with encrypted Amazon EBS volumes without using a launch template, encrypt all new Amazon EBS volumes created in your account. For more information, see [Encryption by default](#) in the [Amazon EC2 User Guide for Linux Instances](#).

Managed node group capacity types

When creating a managed node group, you can choose either the On-Demand or Spot capacity type. Amazon EKS deploys a managed node group with an Amazon EC2 Auto Scaling Group that either contains only On-Demand or only Amazon EC2 Spot Instances. You can schedule pods for fault tolerant applications to Spot managed node groups, and fault intolerant applications to On-Demand node groups within a single Kubernetes cluster. By default, a managed node group deploys On-Demand Amazon EC2 instances.

On-Demand

With On-Demand Instances, you pay for compute capacity by the second, with no long-term commitments.

How it works

By default, if you don't specify a **Capacity Type**, the managed node group is provisioned with On-Demand Instances. A managed node group configures an Amazon EC2 Auto Scaling group on your behalf with the following settings applied:

- The allocation strategy to provision On-Demand capacity is set to `prioritized`. Managed node groups use the order of instance types passed in the API to determine which instance type to use first when fulfilling On-Demand capacity. For example, you might specify three instance types in the following order: `c5.large`, `c4.large`, and `c3.large`. When your On-Demand Instances are launched, the managed node group fulfills On-Demand capacity by starting with `c5.large`, then `c4.large`, and then `c3.large`. For more information, see [Amazon EC2 Auto Scaling group](#) in the [Amazon EC2 Auto Scaling User Guide](#).
- Amazon EKS adds the following Kubernetes label to all nodes in your managed node group that specifies the capacity type: `eks.amazonaws.com/capacityType: ON_DEMAND`. You can use this label to schedule stateful or fault intolerant applications on On-Demand nodes.

Spot

Amazon EC2 Spot Instances are spare Amazon EC2 capacity that offers steep discounts off of On-Demand prices. Amazon EC2 Spot Instances can be interrupted with a two-minute interruption notice when EC2 needs the capacity back. For more information, see [Spot Instances](#) in the [Amazon EC2 User Guide for Linux Instances](#). You can configure a managed node group with Amazon EC2 Spot Instances to optimize costs for the compute nodes running in your Amazon EKS cluster.

How it works

To use Spot Instances inside a managed node group, create a managed node group by setting the capacity type as `spot`. A managed node group configures an Amazon EC2 Auto Scaling group on your behalf with the following Spot best practices applied:

- The allocation strategy to provision Spot capacity is set to `capacity-optimized` to ensure that your Spot nodes are provisioned in the optimal Spot capacity pools. To increase the number of Spot capacity pools available for allocating capacity from, configure a managed node group to use multiple instance types.
- Amazon EC2 Spot Capacity Rebalancing is enabled so that Amazon EKS can gracefully drain and rebalance your Spot nodes to minimize application disruption when a Spot node is at elevated risk of interruption. For more information, see [Amazon EC2 Auto Scaling Capacity Rebalancing](#) in the [Amazon EC2 Auto Scaling User Guide](#).
- When a Spot node receives a rebalance recommendation, Amazon EKS automatically attempts to launch a new replacement Spot node and waits until it successfully joins the cluster.
- When a replacement Spot node is bootstrapped and in the `Ready` state on Kubernetes, Amazon EKS cordons and drains the Spot node that received the rebalance recommendation. Cordon the Spot node ensures that the service controller doesn't send any new requests to this Spot node. It also removes it from its list of healthy, active Spot nodes. Draining the Spot node ensures that running pods are evicted gracefully.
- If a Spot two-minute interruption notice arrives before the replacement Spot node is in a `Ready` state, Amazon EKS starts draining the Spot node that received the rebalance recommendation.
- Amazon EKS adds the following Kubernetes label to all nodes in your managed node group that specifies the capacity type: `eks.amazonaws.com/capacityType: SPOT`. You can use this label to schedule fault tolerant applications on Spot nodes.

Considerations for selecting a capacity type

When deciding whether to deploy a node group with On-Demand or Spot capacity, you should consider the following conditions:

- Spot Instances are a good fit for stateless, fault-tolerant, flexible applications. These include batch and machine learning training workloads, big data ETLs such as Apache Spark, queue processing applications, and stateless API endpoints. Because Spot is spare Amazon EC2 capacity, which can change over time, we recommend that you use Spot capacity for interruption-tolerant workloads. More specifically, Spot capacity is suitable for workloads that can tolerate periods where the required capacity isn't available.
- We recommend that you use On-Demand for applications that are fault intolerant. This includes cluster management tools such as monitoring and operational tools, deployments that require StatefulSets, and stateful applications, such as databases.
- To maximize the availability of your applications while using Spot Instances, we recommend that you configure a Spot managed node group to use multiple instance types. We recommend applying the following rules when using multiple instance types:
 - Within a managed node group, if you're using the [Cluster Autoscaler \(p. 52\)](#), we recommend using a flexible set of instance types with the same amount of vCPU and memory resources. This is to ensure that the nodes in your cluster scale as expected. For example, if you need four vCPUs and eight GiB memory, use c3.xlarge, c4.xlarge, c5.xlarge, c5d.xlarge, c5a.xlarge, c5n.xlarge, or other similar instance types.
 - To enhance application availability, we recommend deploying multiple Spot managed node groups. For this, each group should use a flexible set of instance types that have the same vCPU and memory resources. For example, if you need 4 vCPUs and 8 GiB memory, we recommend that you create one managed node group with c3.xlarge, c4.xlarge, c5.xlarge, c5d.xlarge, c5a.xlarge, c5n.xlarge, or other similar instance types, and a second managed node group with m3.xlarge, m4.xlarge, m5.xlarge, m5d.xlarge, m5a.xlarge, m5n.xlarge or other similar instance types.
 - When deploying your node group with the Spot capacity type that's using a custom launch template, use the API to pass multiple instance types. Don't pass a single instance type through the launch template. For more information about deploying a node group using a launch template, see [Launch template support \(p. 124\)](#).

Creating a managed node group

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

If this is your first time launching an Amazon EKS managed node group, we recommend that you follow one of our [Getting started with Amazon EKS \(p. 4\)](#) guides instead. The guides provide walkthroughs for creating an Amazon EKS cluster with nodes.

Important

- Amazon EKS nodes are standard Amazon EC2 instances. You're billed based on the normal Amazon EC2 prices. For more information, see [Amazon EC2 Pricing](#).
- You can't create managed nodes in an AWS Region where you have AWS Outposts, AWS Wavelength, or AWS Local Zones enabled. You can create self-managed nodes in an AWS Region where you have AWS Outposts, AWS Wavelength, or AWS Local Zones enabled. For more information, see [Launching self-managed Amazon Linux nodes \(p. 133\)](#), [Launching self-managed Windows nodes \(p. 140\)](#), and [Launching self-managed Bottlerocket nodes \(p. 138\)](#).

Prerequisites

- An existing Amazon EKS cluster. To deploy one, see [Creating an Amazon EKS cluster \(p. 24\)](#).
- (Optional, but recommended) The Amazon VPC CNI add-on configured with its own IAM role that has the necessary IAM policy attached to it. For more information, see [Configuring the Amazon VPC CNI plugin to use IAM roles for service accounts \(p. 277\)](#).
- Familiarity with the considerations listed in [Choosing an Amazon EC2 instance type \(p. 172\)](#). Depending on the instance type you choose, there may be additional prerequisites for your cluster and VPC.

You can create a managed node group with `eksctl` or the AWS Management Console.

`eksctl`

To create a managed node group with `eksctl`

This procedure requires `eksctl` version 0.97.0 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 \(p. 11\)](#).

1. (Optional) If the [AmazonEKS_CNI_Policy](#) managed IAM policy is attached to your [the section called "Node IAM role" \(p. 474\)](#), we recommend assigning it to an IAM role that you associate to the Kubernetes `aws-node` service account instead. For more information, see [Configuring the Amazon VPC CNI plugin to use IAM roles for service accounts \(p. 277\)](#).
2. Create a managed node group with or without using a custom launch template. Manually specifying a launch template allows for greater customization of a node group. For example, it can allow deploying a custom AMI or providing arguments to the `bootstrap.sh` script in an Amazon EKS optimized AMI. For a complete list of every available option and default, enter the following command.

```
eksctl create nodegroup --help
```

Replace every `example-value` with your own values.

Important

If you don't use a custom launch template when first creating a managed node group, don't use one at a later time for the node group. If you didn't specify a custom launch template, the system auto-generates a launch template that we don't recommend that you modify manually. Manually modifying this auto-generated launch template might cause errors.

- **Without a launch template** – `eksctl` creates a default Amazon EC2 launch template in your account and deploys the node group using a launch template that it creates based on options that you specify. Before specifying a value for `--node-type`, see [Choosing an Amazon EC2 instance type \(p. 172\)](#).

Replace `my-key` with the name of your Amazon EC2 key pair or public key. This key is used to SSH into your nodes after they launch. If you don't already have an Amazon EC2 key pair, you can create one in the AWS Management Console. For more information, see [Amazon EC2 key pairs](#) in the [Amazon EC2 User Guide for Linux Instances](#).

We recommend blocking pod access to IMDS if the following conditions are true:

- You plan to assign IAM roles to all of your Kubernetes service accounts so that pods only have the minimum permissions that they need.
- No pods in the cluster require access to the Amazon EC2 instance metadata service (IMDS) for other reasons, such as retrieving the current AWS Region.

For more information, see [Restrict access to the instance profile assigned to the worker node](#).

If you want to block pod access to IMDS, then add the `--disable-pod-imds` option to the following command.

```
eksctl create nodegroup \
--cluster my-cluster \
--region region-code \
--name my-mng \
--node-type m5.large \
--nodes 3 \
--nodes-min 2 \
--nodes-max 4 \
--ssh-access \
--ssh-public-key my-key
```

Your instances can optionally assign a significantly higher number of IP addresses to pods, assign IP addresses to pods from a different CIDR block than the instance's, and be deployed to a cluster without internet access. For more information, see [Increase the amount of available IP addresses for your Amazon EC2 nodes \(p. 310\)](#), [CNI custom networking \(p. 306\)](#), and [Private clusters \(p. 102\)](#) for additional options to add to the previous command.

Managed node groups calculates and applies a single value for the maximum number of pods that can run on each node of your node group, based on instance type. If you create a node group with different instance types, the smallest value calculated across all instance types is applied as the maximum number of pods that can run on every instance type in the node group. Managed node groups calculates the value using the script referenced in [Amazon EKS recommended maximum pods for each Amazon EC2 instance type \(p. 173\)](#).

- **With a launch template** – The launch template must already exist and must meet the requirements specified in [Launch template configuration basics \(p. 124\)](#).

We recommend blocking pod access to IMDS if the following conditions are true:

- You plan to assign IAM roles to all of your Kubernetes service accounts so that pods only have the minimum permissions that they need.
- No pods in the cluster require access to the Amazon EC2 instance metadata service (IMDS) for other reasons, such as retrieving the current AWS Region.

For more information, see [Restrict access to the instance profile assigned to the worker node](#).

If you want to block pod access to IMDS, then specify the necessary settings in the launch template.

- a. Create a file named `eks-nodegroup.yaml` with the following contents. Several settings that you specify when deploying without a launch template are moved into the launch template. If you don't specify a version, the template's default version is used.

```
apiVersion: eksctl.io/v1alpha5
kind: ClusterConfig
metadata:
  name: my-cluster
  region: region-code
managedNodeGroups:
  - name: node-group-lt
```

```
launchTemplate:  
  id: lt-id  
  version: "1"
```

For a complete list of `eksctl` config file settings, see [Config file schema](#) in the `eksctl` documentation. Your instances can optionally assign a significantly higher number of IP addresses to pods, assign IP addresses to pods from a different CIDR block than the instance's, use the `containerd` runtime, and be deployed to a cluster without outbound internet access. For more information, see [Increase the amount of available IP addresses for your Amazon EC2 nodes \(p. 310\)](#), [CNI custom networking \(p. 306\)](#), [Enable the containerd runtime bootstrap flag \(p. 184\)](#), and [Private clusters \(p. 102\)](#) for additional options to add to the config file.

If you didn't specify an AMI ID in your launch template, managed node groups calculates and applies a single value for the maximum number of pods that can run on each node of your node group, based on instance type. If you create a node group with different instance types, the smallest value calculated across all instance types is applied as the maximum number of pods that can run on every instance type in the node group. Managed node groups calculates the value using the script referenced in [Amazon EKS recommended maximum pods for each Amazon EC2 instance type \(p. 173\)](#).

If you specified an AMI ID in your launch template, specify the maximum number of pods that can run on each node of your node group if you're using [custom networking \(p. 306\)](#) or want to [increase the number of IP addresses assigned to your instance \(p. 310\)](#). For more information, see [Amazon EKS recommended maximum pods for each Amazon EC2 instance type \(p. 173\)](#).

- b. Deploy the nodegroup with the following command.

```
eksctl create nodegroup --config-file eks-nodegroup.yaml
```

AWS Management Console

To create a managed node group using the AWS Management Console

1. Wait for your cluster status to show as ACTIVE. You can't create a managed node group for a cluster that isn't already ACTIVE.
2. Open the Amazon EKS console at <https://console.aws.amazon.com/eks/home#/clusters>.
3. Choose the name of the cluster that you want to create a managed node group in.
4. Select the **Configuration** tab.
5. On the **Configuration** tab, select the **Compute** tab, and then choose **Add Node Group**.
6. On the **Configure Node Group** page, fill out the parameters accordingly, and then choose **Next**.
 - **Name** – Enter a unique name for your managed node group.
 - **Node IAM Role** – Choose the node instance role to use with your node group. For more information, see [Amazon EKS node IAM role \(p. 474\)](#).

Important

We recommend using a role that's not currently in use by any self-managed node group. Otherwise, you plan to use with a new self-managed node group. For more information, see [Deleting a managed node group \(p. 131\)](#).

- **Use launch template** – (Optional) Choose if you want to use an existing launch template. Select a **Launch Template Name**. Then, select a **Launch template version**. If you don't select a version, then Amazon EKS uses the template's default version. Launch templates allow for more customization of your node group, such as allowing you to deploy a custom AMI, assign a significantly higher number of IP addresses to pods, assign IP addresses to pods from a

different CIDR block than the instance's, enable the containerd runtime for your instances, and deploying nodes to a cluster without outbound internet access. For more information, see [Increase the amount of available IP addresses for your Amazon EC2 nodes \(p. 310\)](#), [CNI custom networking \(p. 306\)](#), [Enable the containerd runtime bootstrap flag \(p. 184\)](#), and [Private clusters \(p. 102\)](#).

The launch template must meet the requirements in [Launch template support \(p. 124\)](#). If you don't use your own launch template, the Amazon EKS API creates a default Amazon EC2 launch template in your account and deploys the node group using the default launch template.

If you implement [IAM roles for service accounts \(p. 442\)](#), assign necessary permissions directly to every pod that requires access to AWS services, and no pods in your cluster require access to IMDS for other reasons, such as retrieving the current AWS Region, then you can also disable access to IMDS for pods that don't use host networking in a launch template. For more information, see [Restrict access to the instance profile assigned to the worker node](#).

- **Kubernetes labels** – (Optional) You can choose to apply Kubernetes labels to the nodes in your managed node group.
 - **Kubernetes taints** – (Optional) You can choose to apply Kubernetes taints to the nodes in your managed node group. The available options in the **Effect** menu are **NoSchedule**, **NoExecute**, and **PreferNoSchedule**.
 - **Tags** – (Optional) You can choose to tag your Amazon EKS managed node group. These tags don't propagate to other resources in the node group, such as Auto Scaling groups or instances. For more information, see [Tagging your Amazon EKS resources \(p. 431\)](#).
 - **Node group update configuration** – (Optional) You can select the number or percentage of nodes to be updated in parallel. Select either **Number** or **Percentage** to enter a value. These nodes won't be available during the update.
7. On the **Set compute and scaling configuration** page, fill out the parameters accordingly, and then choose **Next**.

- **AMI type** – Choose **Amazon Linux 2 (AL2_x86_64)** for Linux non-GPU instances, **Amazon Linux 2 GPU Enabled (AL2_x86_64_GPU)** for Linux GPU instances, **Amazon Linux 2 Arm (AL2_ARM_64)** for Linux Arm instances, **Bottlerocket (BOTTLEROCKET_x86_64)** for Bottlerocket x86_64 instances, or **Bottlerocket Arm (BOTTLEROCKET_ARM_64)** for Bottlerocket Arm instances.

If you are deploying Arm instances, be sure to review the considerations in [Amazon EKS optimized Arm Amazon Linux AMIs \(p. 186\)](#) before deploying.

If you specified a launch template on the previous page, and specified an AMI in the launch template, then you can't select a value. The value from the template is displayed. The AMI specified in the template must meet the requirements in [Specifying an AMI \(p. 128\)](#).

- **Capacity type** – Select a capacity type. For more information about choosing a capacity type, see [Managed node group capacity types \(p. 110\)](#). You can't mix different capacity types within the same node group. If you want to use both capacity types, create separate node groups, each with their own capacity and instance types.
- **Instance types** – By default, one or more instance type is specified. To remove a default instance type, select the X on the right side of the instance type. Choose the instance types to use in your managed node group. For more information, see [Choosing an Amazon EC2 instance type \(p. 172\)](#).

The console displays a set of commonly used instance types. If you need to create a managed node group with an instance type that's not displayed, then use eksctl, the AWS CLI, AWS CloudFormation, or an SDK to create the node group. If you specified a launch template on the previous page, then you can't select a value because the instance type must be specified in the launch template. The value from the launch template is displayed. If you selected

Spot for Capacity type, then we recommend specifying multiple instance types to enhance availability.

- **Disk size** – Enter the disk size (in GiB) to use for your node's root volume.

If you specified a launch template on the previous page, then you can't select a value because it must be specified in the launch template.

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

Note

Amazon EKS doesn't automatically scale your node group in or out. However, you can configure the Kubernetes [Cluster Autoscaler \(p. 52\)](#) to do this for you.

- For **Maximum unavailable**, select one of the following options and specify a **Value**:
 - **Number** – Select and specify the number of nodes in your node group that can be updated in parallel. These nodes will be unavailable during update.
 - **Percentage** – Select and specify the percentage of nodes in your node group that can be updated in parallel. These nodes will be unavailable during update. This is useful if you have a large number of nodes in your node group.
8. On the **Specify networking** page, fill out the parameters accordingly, and then choose **Next**.
- **Subnets** – Choose the subnets to launch your managed nodes into.

Important

If you are running a stateful application across multiple Availability Zones that is backed by Amazon EBS volumes and using the Kubernetes [Cluster Autoscaler \(p. 52\)](#), you should configure multiple node groups, each scoped to a single Availability Zone. In addition, you should enable the `--balance-similar-node-groups` feature.

Important

- If you choose a public subnet, and your cluster has only the public API server endpoint enabled, then the subnet must have `MapPublicIpOnLaunch` set to `true` for the instances to successfully join a cluster. If the subnet was created using `eksctl` or the [Amazon EKS vended AWS CloudFormation templates \(p. 265\)](#) on or after March 26, 2020, then this setting is already set to `true`. If the subnets were created with `eksctl` or the AWS CloudFormation templates before March 26, 2020, then you need to change the setting manually. For more information, see [Modifying the public IPv4 addressing attribute for your subnet](#).
- If you use a launch template and specify multiple network interfaces, Amazon EC2 won't auto-assign a public IPv4 address, even if `MapPublicIpOnLaunch` is set to `true`. For nodes to join the cluster in this scenario, you must either enable the cluster's private API server endpoint, or launch nodes in a private subnet with outbound internet access provided through an alternative method, such as a NAT Gateway. For more information, see [Amazon EC2 instance IP addressing](#) in the [Amazon EC2 User Guide for Linux Instances](#).
- **Configure SSH access to nodes (Optional).** Enabling SSH allows you to connect to your instances and gather diagnostic information if there are issues. Complete the following steps to enable remote access. We highly recommend enabling remote access when you create a node group. You can't enable remote access after the node group is created.

If you chose to use a launch template, then this option isn't shown. To enable remote access to your nodes, specify a key pair in the launch template and ensure that the proper port is

open to the nodes in the security groups that you specify in the launch template. For more information, see [Using custom security groups \(p. 126\)](#).

- For **SSH key pair** (Optional), choose an Amazon EC2 SSH key to use. For more information, see [Amazon EC2 key pairs](#) in the *Amazon EC2 User Guide for Linux Instances*. If you chose to use a launch template, then you can't select one. When an Amazon EC2 SSH key is provided for node groups using Bottlerocket AMIs, the administrative container is also enabled. For more information, see [Admin container](#) on GitHub.
 - For **Allow SSH remote access from**, if you want to limit access to specific instances, then select the security groups that are associated to those instances. If you don't select specific security groups, then SSH access is allowed from anywhere on the internet (0.0.0.0/0).
9. On the **Review and create** page, review your managed node group configuration and choose **Create**.

If nodes fail to join the cluster, then see [Nodes fail to join cluster \(p. 546\)](#) in the Troubleshooting guide.

10. Watch the status of your nodes and wait for them to reach the **Ready** status.

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

11. (GPU nodes only) If you chose a GPU instance type and the Amazon EKS optimized accelerated AMI, then you must apply the [NVIDIA device plugin for Kubernetes](#) as a DaemonSet on your cluster with the following command.

```
kubectl apply -f https://raw.githubusercontent.com/NVIDIA/k8s-device-plugin/v0.9.0/nvidia-device-plugin.yaml
```

12. (Optional) After you add Linux worker nodes to your cluster, follow the procedures in [Windows support \(p. 92\)](#) to add Windows support to your cluster and to add Windows worker nodes. Every Amazon EKS cluster must contain at least one Linux worker node, even if you only want to run Windows workloads in your cluster.

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

- 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 \(p. 402\)](#) and [Required permissions \(p. 506\)](#).
- We recommend blocking pod access to IMDS if the following conditions are true:
 - You plan to assign IAM roles to all of your Kubernetes service accounts so that pods only have the minimum permissions that they need.
 - No pods in the cluster require access to the Amazon EC2 instance metadata service (IMDS) for other reasons, such as retrieving the current AWS Region.

For more information, see [Restrict access to the instance profile assigned to the worker node](#).

- [Cluster Autoscaler \(p. 52\)](#) – Configure the Kubernetes Cluster Autoscaler to automatically adjust the number of nodes in your node groups.
- [Deploy a sample application \(p. 359\)](#) to your cluster.
- [Cluster management \(p. 422\)](#) – Learn how to use important tools for managing your cluster.

Updating a managed node group

When you initiate a managed node group update, Amazon EKS automatically updates your nodes for you, completing the steps listed in [Managed node update behavior \(p. 121\)](#). If you're using an Amazon EKS optimized AMI, Amazon EKS automatically applies the latest security patches and operating system updates to your nodes as part of the latest AMI release version.

There are several scenarios where it's useful to update your Amazon EKS managed node group's version or configuration:

- You have updated the Kubernetes version for your Amazon EKS cluster and want to update your nodes to use the same Kubernetes version.
- A new AMI release version is available for your managed node group. For more information about AMI versions, see these sections:
 - [Amazon EKS optimized Amazon Linux AMI versions \(p. 187\)](#)
 - [Amazon EKS optimized Bottlerocket AMIs \(p. 205\)](#)
 - [Amazon EKS optimized Windows AMI versions \(p. 219\)](#)
- You want to adjust the minimum, maximum, or desired count of the instances in your managed node group.
- You want to add or remove Kubernetes labels from the instances in your managed node group.
- You want to add or remove AWS tags from your managed node group.
- You need to deploy a new version of a launch template with configuration changes, such as an updated custom AMI.
- You have deployed version 1.9.0 or later of the Amazon VPC CNI add-on, enabled the add-on for prefix delegation, and want new AWS Nitro System instances in a node group to support a significantly increased number of pods. For more information, see [Increase the amount of available IP addresses for your Amazon EC2 nodes \(p. 310\)](#).

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

When a node in a managed node group is terminated due to a scaling operation or update, the pods in that node are drained first. For more information, see [Managed node update behavior \(p. 121\)](#).

Update a node group version

You can update a node group version with `eksctl` or the AWS Management Console. Select the tab with the name of the tool that you want to use to update your node group. The version that you update to can't be greater than the control plane's version.

`eksctl`

To update a node group version with `eksctl`

- Update a managed node group to the latest AMI release of the same Kubernetes version that's currently deployed on the nodes with the following command. Replace every `example-value` with your own values.

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

Note

If you're upgrading a node group that's deployed with a launch template to a new launch template version, add `--launch-template-version` *version-number* to the preceding command. The launch template must meet the requirements described in [Launch template support \(p. 124\)](#). If the launch template includes a custom AMI, the AMI must meet the requirements in [Specifying an AMI \(p. 128\)](#). When you upgrade your node group to a newer version of your launch template, every node is recycled to match the new configuration of the launch template version that's specified.

You can't directly upgrade a node group that's deployed without a launch template to a new launch template version. Instead, you must deploy a new node group using the launch template to update the node group to a new launch template version.

You can upgrade a node group to the same version as the control plane's Kubernetes version. For example, if you have a cluster running Kubernetes 1.22, you can upgrade nodes currently running Kubernetes 1.21 to version 1.22 with the following command.

```
eksctl upgrade nodegroup \  
  --name=node-group-name \  
  --cluster=cluster-name \  
  --kubernetes-version=1.22
```

AWS Management Console

To update a node group version with the AWS Management Console

1. Open the Amazon EKS console at <https://console.aws.amazon.com/eks/home#/clusters>.
2. Choose the cluster that contains the node group to update.
3. If at least one node group has an available update, a box appears at the top of the page notifying you of the available update. If you select the **Configuration** tab and then the **Compute** tab, you'll see **Update now** in the **AMI release version** column in the **Node Groups** table for the node group that has an available update. To update the node group, select **Update now**. You won't see a notification for node groups that were deployed with a custom AMI. If your nodes are deployed with a custom AMI, complete the following steps to deploy a new updated custom AMI.
 - a. Create a new version of your AMI.
 - b. Create a new launch template version with the new AMI ID.
 - c. Upgrade the nodes to the new version of the launch template.
4. On the **Update Node Group version** page, select:
 - **Update Node Group version** – This option is unavailable if you deployed a custom AMI or your Amazon EKS optimized AMI is currently on the latest version for your cluster.
 - **Launch template version** – This option is unavailable if the node group is deployed without a custom launch template. You can only update the launch template version for a node group that has been deployed with a custom launch template. Select the version that you want to update the node group to. If your node group is configured with a custom AMI, then the version that you select must also specify an AMI. When you upgrade to a newer version of your launch template, every node is recycled to match the new configuration of the launch template version specified.
5. For **Update strategy**, select one of the following options and then choose **Update**.

- **Rolling update** – This option respects the pod disruption budgets for your cluster. Updates fail if there's a pod disruption budget issue that causes Amazon EKS to be unable to gracefully drain the pods that are running on this node group.
- **Force update** – This option doesn't respect pod disruption budgets. Updates occur regardless of pod disruption budget issues by forcing node restarts to occur.

Edit a node group configuration

You can modify some of the configurations of a managed node group.

To edit a node group configuration

1. Open the Amazon EKS console at <https://console.aws.amazon.com/eks/home#/clusters>.
2. Choose the cluster that contains the node group to edit.
3. Select the **Configuration** tab. On the **Compute** tab, select the node group to edit, and choose **Edit**.
4. (Optional) On the **Edit node group** page, edit the **Group configuration**.
 - **Tags** – Add tags to or remove tags from your node group resource. These tags are only applied to the Amazon EKS node group. They don't propagate to other resources, such as subnets or Amazon EC2 instances in the node group.
 - **Kubernetes labels** – Add or remove Kubernetes labels to the nodes in your node group. The labels shown here are only the labels that you have applied with Amazon EKS. Other labels may exist on your nodes that aren't shown here.
 - **Kubernetes taints** – Add or remove Kubernetes taints to the nodes in your node group. Added taints can have the effect of either `No_Schedule`, `Prefer_No_Schedule`, or `No_Execute`.
5. (Optional) On the **Edit node group** page, edit the **Group size**.
 - **Minimum size** – Specify the minimum number of nodes that the managed node group can scale in to.
 - **Maximum size** – Specify the maximum number of nodes that the managed node group can scale out to. For the maximum number of nodes supported in a node group, see [Amazon EKS service quotas \(p. 435\)](#).
 - **Desired size** – Specify the current number of nodes that the managed node group should maintain.
6. (Optional) Edit the **Node Group update configuration**. Select either **Number** or **Percentage**.
 - **Number** – Select and specify the number of nodes in your node group that can be updated in parallel. These nodes will be unavailable during update.
 - **Percentage** – Select and specify the percentage of nodes in your node group that can be updated in parallel. These nodes will be unavailable during update. This is useful if you have many nodes in your node group.
7. When you're finished editing, choose **Save changes**.

Managed node update behavior

The Amazon EKS managed worker node upgrade strategy has four different phases described in the following sections.

Setup phase

The setup phase has these steps:

1. Creates a new Amazon EC2 launch template version for the Auto Scaling group that's associated with your node group. The new launch template version uses the target AMI or customer-provided launch template version for the update.
2. Updates the Auto Scaling group to use the latest launch template version.
3. Determines the maximum quantity of nodes to upgrade in parallel using the `updateConfig` property for the node group. The maximum unavailable has a quota of 100 nodes. The default value is one node. For more information, see the `updateConfig` property in the *Amazon EKS API Reference*.

Scale up phase

When upgrading the nodes in a managed node group, the upgraded nodes are launched in the same Availability Zone as those that are being upgraded. To guarantee this placement, we use Amazon EC2's Availability Zone Rebalancing. For more information, see [Availability Zone Rebalancing](#) in the *Amazon EC2 Auto Scaling User Guide*. To meet this requirement, it's possible that we'd launch up to two instances per Availability Zone in your Managed Node Group.

The scale up phase has these steps:

1. Increments the Auto Scaling Group's maximum size and desired size by the larger of either:
 - Up to twice the number of Availability Zones that the Auto Scaling group is deployed in.
 - The maximum unavailable of upgrade.

For example, if your node group has five Availability Zones and `maxUnavailable` as one, the upgrade process can launch a maximum of 10 nodes. However when `maxUnavailable` is 20 (or anything greater than 10, the process would launch 20 new nodes).

2. After scaling the Auto Scaling Group, it checks if the nodes using the latest configuration are present in the node group. This step succeeds only when it meets these criteria:
 - At least one new node is launched in every Availability Zone where the node exists.
 - Every new node should be in Ready state.
 - New nodes should have Amazon EKS applied labels.

These are the Amazon EKS applied labels on the worker nodes in a regular node group:

- `eks.amazonaws.com/nodegroup-image=<$amiName>`
- `eks.amazonaws.com/nodegroup=<$nodeGroupName>`

These are the Amazon EKS applied labels on the worker nodes in a custom launch template or AMI node group:

- `eks.amazonaws.com/nodegroup-image=<$amiName>`
- `eks.amazonaws.com/nodegroup=<$nodeGroupName>`
- `eks.amazonaws.com/sourceLaunchTemplateId=<$launchTemplateId>`
- `eks.amazonaws.com/sourceLaunchTemplateVersion=<$launchTemplateVersion>`

3. Applies an `eks.amazonaws.com/nodegroup=unschedulable:NoSchedule` taint on every node in the node group without the latest labels. This prevents nodes that have already been updated from a previous failed update from being tainted.

The following are known reasons which lead to a `NodeCreationFailure` error in this phase:

- **Insufficient capacity in the Availability Zone** – There is a possibility that the Availability Zone might not have capacity of requested instance types. It's recommended to configure multiple instance types while creating a managed node group.
- **Customers hitting EC2 instance limits in their account** – You may need to increase the number of Amazon EC2 instances your account can run simultaneously using Service Quotas. For more

information, see [EC2 Service Quotas](#) in the *Amazon Elastic Compute Cloud User Guide for Linux Instances*.

- **Custom user data** – Custom user data can sometimes break the bootstrap process. This scenario can lead to the kubelet not starting on the node or nodes not getting expected Amazon EKS labels on them. For more information on handling custom LT/AMI, see [Specifying an AMI \(p. 128\)](#).
- **Any changes which make a node unhealthy or not ready** – Node disk pressure, memory pressure, and similar conditions can lead to a node not going to Ready state.

Upgrade phase

The upgrade phase has these steps:

1. Randomly selects a node, up to the maximum unavailable configured for the node group.
2. Drains the pods from the node. If the pods don't leave the node within 15 minutes and there's no force flag, the upgrade phase fails with a `PodEvictionFailure` error. For this scenario, you can apply the force flag with the `update-nodegroup-version` request to delete the pods.
3. Cordons the node after every pod is evicted and waits for 60 seconds. This is done so that the service controller doesn't send any new requests to this node and removes this node from its list of active nodes.
4. Sends a termination request to the Auto Scaling Group for the cordoned node.
5. Repeats the previous upgrade steps until there are no nodes in the node group that are deployed with the earlier version of the launch template.

The following are known reasons which lead to a `PodEvictionFailure` error in this phase:

- **Aggressive PDB** – Aggressive PDB is defined on the pod or there are multiple PDBs pointing to the same pod.
- **Deployment tolerating all the taints** – Once every pod is evicted, it's expected for the node to be empty because the node is tainted in the earlier steps. However, if the deployment tolerates every taint, then the node is more likely to be non-empty, leading to pod eviction failure.

Scale down phase

The scale down phase decrements the Auto Scaling group maximum size and desired size by one to return to values before the update started.

If the Upgrade workflow determines that the Cluster Autoscaler is scaling up the node group during the scale down phase of the workflow, it exits immediately without bringing the node group back to its original size.

Node taints on managed node groups

Amazon EKS supports configuring Kubernetes taints through managed node groups. Taints and tolerations work together to ensure that pods aren't scheduled onto inappropriate nodes.

One or more taints can be applied to a node. This marks that the node shouldn't accept any pods that don't tolerate the taints. Tolerations are applied to pods and allow, but don't require, the pods to schedule onto nodes with matching taints.

Kubernetes node taints can be applied to new and existing managed node groups using the AWS Management Console or through the Amazon EKS API.

The following is an example of creating a node group with a taint using the AWS CLI:

```
aws eks create-nodegroup \
```

```
--cli-input-json '
{
  "clusterName": "my-cluster",
  ...
  "taints": [
    {
      "key": "dedicated",
      "value": "gpuGroup",
      "effect": "NO_SCHEDULE"
    }
  ],
}
```

For more information on taints and tolerations, see the [Kubernetes documentation](#). For more information and examples of usage, see the [Kubernetes reference documentation](#).

Note

- Maximum of 50 taints are allowed for one node group.
- Taints can be updated after you create the node group using the `UpdateNodegroupConfig` API.
- The taint key must begin with a letter or number. It can contain letters, numbers, hyphens (-), periods (.), and underscores (_). It can be up to 63 characters long.
- Optionally, the taint key can begin with a DNS subdomain prefix and a single /. If it begins with a DNS subdomain prefix, it can be 253 characters long.
- The value is optional and must begin with a letter or number. It can contain letters, numbers, hyphens (-), periods (.), and underscores (_). It can be up to 63 characters long.
- The effect must be one of `No_Schedule`, `Prefer_No_Schedule`, or `No_Execute`.

Launch template support

Managed node groups are always deployed with a launch template to be used with the Amazon EC2 Auto Scaling Group. The Amazon EKS API creates this launch template either by copying one you provide or by creating one automatically with default values in your account. For the highest level of customization, you can deploy managed nodes using your own launch template and a custom AMI. We don't recommend that you modify auto-generated launch templates. So, make sure to specify a custom one when first creating a managed node group if you want greater flexibility.

After you deployed a managed node group with your own launch template, update it with a different version of the same launch template. When you update your node group to a different version of your launch template, all nodes in the group are recycled to match the new configuration of the specified launch template version. Existing node groups that don't use a custom launch template can't be updated directly. Rather, you must create a new node group with a custom launch template to do so.

Launch template configuration basics

You can create an Amazon EC2 Auto Scaling launch template with the AWS Management Console, AWS CLI, or an AWS SDK. For more information, see [Creating a Launch Template for an Auto Scaling Group](#) in the *Amazon EC2 Auto Scaling User Guide*. Some of the settings in a launch template are similar to the settings used for managed node configuration. When deploying or updating a node group with a launch template, some settings must be specified in either the node group configuration or the launch template. Don't specify both places. If a setting exists where it shouldn't, then operations such as creating or updating a node group fail.

The following table lists the settings that are prohibited in a launch template. It also lists similar settings, if any are available, that are required in the managed node group configuration. The listed settings are the settings that appear in the console. They might have similar but different names in the AWS CLI and SDK.

Launch template – Prohibited	Amazon EKS node group configuration
IAM instance profile under Advanced details	Node IAM Role under Node Group configuration on the Configure Node Group page
Subnet under Network interfaces (Add network interface)	Subnets under Node Group network configuration on the Specify networking page
Shutdown behavior and Stop - Hibernate behavior under Advanced details. Retain default Don't include in launch template setting in launch template for both settings.	No equivalent. Amazon EKS must control the instance lifecycle, not the Auto Scaling group.

The following table lists the prohibited settings in a managed node group configuration. It also lists similar settings, if any are available, which are required in a launch template. The listed settings are the settings that appear in the console. They might have similar names in the AWS CLI and SDK.

Amazon EKS node group configuration – Prohibited	Launch template
(Only if you specified a custom AMI in a launch template) AMI type under Node Group compute configuration on Set compute and scaling configuration page – Console displays Specified in launch template and the AMI ID that was specified. If an AMI type wasn't specified in the launch template, you can select an AMI in the node group configuration.	AMI under Launch template contents – You must specify an ID if you have either of the following requirements: <ul style="list-style-type: none"> Using a custom AMI. If you specify an AMI that doesn't meet the requirements listed in Specifying an AMI (p. 128), the node group deployment will fail. Want to provide user data to provide arguments to the <code>bootstrap.sh</code> file included with an Amazon EKS optimized AMI. You can enable your instances to assign a significantly higher number of IP addresses to pods, assign IP addresses to pods from a different CIDR block than the instance's, enable the <code>containerd</code> runtime, or deploy a private cluster without outbound internet access. For more information, see Increase the amount of available IP addresses for your Amazon EC2 nodes (p. 310), CNI custom networking (p. 306), Enable the containerd runtime bootstrap flag (p. 184), Private clusters (p. 102), and Specifying an AMI (p. 128).
Disk size under Node Group compute configuration on Set compute and scaling configuration page – Console displays Specified in launch template .	Size under Storage (Volumes) (Add new volume) . You must specify this in the launch template.
SSH key pair under Node Group configuration on the Specify Networking page – The console displays the key that was specified in the launch template or displays Not specified in launch template .	Key pair name under Key pair (login) .

Amazon EKS node group configuration – Prohibited	Launch template
You can't specify source security groups that are allowed remote access when using a launch template.	Security groups under Network settings for the instance or Security groups under Network interfaces (Add network interface) , but not both. For more information, see Using custom security groups (p. 126) .

Note

- If you deploy a node group using a launch template, specify zero or one **Instance type** under **Launch template contents** in a launch template. Alternatively, you can specify 0–20 instance types for **Instance types** on the **Set compute and scaling configuration** page in the console. Or, you can do so using other tools that use the Amazon EKS API. If you specify an instance type in a launch template, and use that launch template to deploy your node group, then you can't specify any instance types in the console or using other tools that use the Amazon EKS API. If you don't specify an instance type in a launch template, in the console, or using other tools that use the Amazon EKS API, the `t3.medium` instance type is used. If your node group is using the Spot capacity type, then we recommend specifying multiple instance types using the console. For more information, see [Managed node group capacity types \(p. 110\)](#).
- If any containers that you deploy to the node group use the Instance Metadata Service Version 2, make sure to set the **Metadata response hop limit** to 2 in your launch template. For more information, see [Instance metadata and user data](#) in the *Amazon EC2 User Guide for Linux Instances*. If you deploy a managed node group without using a custom launch template, this value is automatically set for the node group in the default launch template.

Tagging Amazon EC2 instances

You can use the `TagSpecification` parameter of a launch template to specify which tags to apply to Amazon EC2 instances in your node group. The IAM entity calling the `CreateNodegroup` or `UpdateNodegroupVersion` APIs must have permissions for `ec2:RunInstances` and `ec2:CreateTags`, and the tags must be added to the launch template.

Using custom security groups

You can use a launch template to specify custom Amazon EC2 [security groups](#) to apply to instances in your node group. This can be either in the instance level security groups parameter or as part of the network interface configuration parameters. However, you can't create a launch template that specifies both instance level and network interface security groups. Consider the following conditions that apply to using custom security groups with managed node groups:

- Amazon EKS only allows launch templates with a single network interface specification.
- By default, Amazon EKS applies the [cluster security group \(p. 272\)](#) to the instances in your node group to facilitate communication between nodes and the control plane. If you specify custom security groups in the launch template using either option mentioned earlier, Amazon EKS doesn't add the cluster security group. So, you must ensure that the inbound and outbound rules of your security groups enable communication with the endpoint of your cluster. If your security group rules are incorrect, the worker nodes can't join the cluster. For more information about security group rules, see [Amazon EKS security group considerations \(p. 272\)](#).
- If you need SSH access to the instances in your node group, include a security group that allows that access.

Amazon EC2 user data

The launch template includes a section for custom user data. You can specify configuration settings for your node group in this section without manually creating individual custom AMIs. For more information about the settings available for Bottlerocket, see [Using user data](#) on GitHub.

You can supply Amazon EC2 user data in your launch template using `cloud-init` when launching your instances. For more information, see the [cloud-init](#) documentation. Your user data can be used to perform common configuration operations. This includes the following operations:

- [Including users or groups](#)
- [Installing packages](#)

Amazon EC2 user data in launch templates that are used with managed node groups must be in the [MIME multi-part archive](#) format for Amazon Linux AMIs and TOML format for Bottlerocket AMIs. This is because your user data is merged with Amazon EKS user data required for nodes to join the cluster. Don't specify any commands in your user data that starts or modifies `kubelet`. This is performed as part of the user data merged by Amazon EKS. Certain `kubelet` parameters, such as setting labels on nodes, can be configured directly through the managed node groups API.

Note

For more information about advanced `kubelet` customization, including manually starting it or passing in custom configuration parameters, see [Specifying an AMI \(p. 128\)](#). If a custom AMI ID is specified in a launch template, Amazon EKS doesn't merge user data.

The following details provide more information about the user data section for Amazon Linux or Bottlerocket.

Amazon Linux user data

You can combine multiple user data blocks together into a single MIME multi-part file. For example, you can combine a cloud booothook that configures the Docker daemon with a user data shell script that installs a custom package. A MIME multi-part file consists of the following components:

- The content type and part boundary declaration – `Content-Type: multipart/mixed; boundary="==MYBOUNDARY=="`
- The MIME version declaration – `MIME-Version: 1.0`
- One or more user data blocks, which contain the following components:
 - The opening boundary, which signals the beginning of a user data block – `====MYBOUNDARY==`
 - The content type declaration for the block: `Content-Type: text/cloud-config; charset="us-ascii"`. For more information about content types, see the [cloud-init](#) documentation.
 - The content of the user data (for example, a list of shell commands or `cloud-init` directives).
 - The closing boundary, which signals the end of the MIME multi-part file: `-- ==MYBOUNDARY==`

The following is an example of a MIME multi-part file that you can use to create your own.

```
MIME-Version: 1.0
Content-Type: multipart/mixed; boundary=="==MYBOUNDARY=="

====MYBOUNDARY==
Content-Type: text/x-shellscript; charset="us-ascii"

#!/bin/bash
echo "Running custom user data script"
```

```
--=MYBOUNDARY=--
```

Bottlerocket user data

Bottlerocket structures user data in the TOML format. You can provide user data to be merged with the user data provided by Amazon EKS. For example, you can provide additional kubelet settings.

```
[settings.kubernetes.system-reserved]
cpu = "10m"
memory = "100Mi"
ephemeral-storage= "1Gi"
```

For more information about the supported settings, see [Bottlerocket documentation](#). You can configure node labels and taints in your user data. However, we recommend that you configure these within your node group instead. Amazon EKS applies these configurations when you do so.

When user data is merged, formatting isn't preserved, but the content remains the same. The configuration that you provide in your user data overrides any settings that are configured by Amazon EKS. So, if you set `settings.kubernetes.max-pods` or `settings.kubernetes.cluster-dns-ip`, values in your user data are applied to the nodes.

Amazon EKS doesn't support all valid TOML. The following is a list of known unsupported formats:

- Quotes within quoted keys: '`quoted "value"`' = "value"
- Escaped quotes in values: `str = "I'm a string. \"You can quote me\""`
- Mixed floats and integers: `numbers = [0.1, 0.2, 0.5, 1, 2, 5]`
- Mixed types in arrays: `contributors = ["foo@example.com", { name = "Baz", email = "baz@example.com" }]`
- Bracketed headers with quoted keys: `[foo . "bar.baz"]`

Specifying an AMI

If you have either of the following requirements, then specify an AMI ID in the `imageId` field of your launch template. Select the requirement you have for additional information.

Provide user data to pass arguments to the `bootstrap.sh` file included with an Amazon EKS optimized AMI

Bootstrapping is a term used to describe adding commands that can be run when an instance starts. You can pass arguments to the `bootstrap.sh` script by using `eksctl` without specifying a launch template. Or you can do so by specifying the information in the user data section of a launch template.

eksctl without specifying a launch template

Create a file named `my-nodegroup.yaml` with the following contents. This example creates a node group using `containerd` as the runtime, but you can modify it as needed. It also provides an additional `kubelet` argument to set a custom `max-pods` value using the `bootstrap.sh` script included with the Amazon EKS optimized AMI. For more information, see the `bootstrap.sh` file on GitHub.

Replace every `example-value` with your own values:

- To retrieve your desired value for `ami-1234567890abcdef0`, you can use the tables in the following sections:
 - [Amazon EKS optimized Amazon Linux AMIs \(p. 175\)](#)
 - [Amazon EKS optimized Bottlerocket AMIs \(p. 205\)](#)

- [Amazon EKS optimized Windows AMIs \(p. 211\)](#)
- For help with selecting `my-max-pods-value`, see [Amazon EKS recommended maximum pods for each Amazon EC2 instance type \(p. 173\)](#).
- To retrieve the `certificate-authority` for your cluster, run the following command.

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

- To retrieve the `api-server-endpoint` for your cluster, run the following command.

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

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

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

managedNodeGroups:
  - name: my-nodegroup
    ami: ami-1234567890abcdef0
    instanceType: m5.large
    privateNetworking: true
    disableIMDSv1: true
    labels: { x86-al2-specified-mng }
    overrideBootstrapCommand: |
      #!/bin/bash
      /etc/eks/bootstrap.sh my-cluster \
        --kubelet-extra-args '--max-pods=my-max-pods-value' \
        --b64-cluster-ca certificate-authority \
        --apiserver-endpoint api-server-endpoint \
        --dns-cluster-ip service-ipv4-cidr.10 \
        --use-max-pods false \
        --container-runtime containerd
```

The only required argument in the previous example is the cluster name (`my-cluster`). However, by setting the values for `--apiserver-endpoint`, `--b64-cluster-ca`, and `--dns-cluster-ip`, there's no need for the bootstrap script to make a `describeCluster` call. This is useful in private cluster setups or clusters where you're scaling in and out nodes frequently.

You can find the values for your cluster to specify the values for the optional arguments with the following command.

```
aws eks describe-cluster --name my-cluster
```

The example values for the optional arguments are the name of the properties returned in the output from the command. The value for `--dns-cluster-ip` is your service CIDR with .10 at the end. For example, if the returned value for `serviceIpv4Cidr` is `10.100.0.0/16`, then your value is `10.100.0.10`.

For every available `eksctl` config file option, see [Config file schema](#) in the `eksctl` documentation. The `eksctl` utility still creates a launch template for you and populates its user data with the data that you provide in the config file.

Create a node group with the following command.

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

User data in a launch template

Specify the following information in the user data section of your launch template. This example creates a node group that uses `containerd` as the runtime, but you can modify it as needed. It also provides an additional `kubelet` argument to set a custom `max-pods` value using the `bootstrap.sh` script included with the Amazon EKS optimized AMI. For more information, see the `bootstrap.sh` file on GitHub.

Replace every *example-value* with your own values:

- For help with selecting *my-max-pods-value*, see [Amazon EKS recommended maximum pods for each Amazon EC2 instance type \(p. 173\)](#).
- To retrieve the *certificate-authority* for your cluster, run the following command.

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

- To retrieve the *api-server-endpoint* for your cluster, run the following command.

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

```
MIME-Version: 1.0  
Content-Type: multipart/mixed; boundary="==MYBOUNDARY=="  
  
==MYBOUNDARY==  
Content-Type: text/x-shellscrip; charset="us-ascii"  
  
#!/bin/bash  
set -ex  
/etc/eks/bootstrap.sh my-cluster \  
--kubelet-extra-args '--max-pods=my-max-pods-value' \  
--b64-cluster-ca certificate-authority \  
--apiserver-endpoint api-server-endpoint \  
--dns-cluster-ip service-ipv4-cidr.10 \  
--use-max-pods false \  
--container-runtime containerd  
  
==MYBOUNDARY==
```

The only required argument in the previous example is the cluster name (*my-cluster*). However, by setting the values for `--apiserver-endpoint`, `--b64-cluster-ca`, and `--dns-cluster-ip`, there's no need for the `bootstrap` script to make a `describeCluster` call. This is useful in private cluster setups or clusters where you're scaling in and out nodes frequently.

You can find the values for your cluster to specify the values for the optional arguments with the following command.

```
aws eks describe-cluster --name my-cluster
```

The example values for the optional arguments are the name of the properties returned in the output from the command. The value for `--dns-cluster-ip` is your service CIDR with .10 at the end. For example, if the returned value for `serviceIpv4Cidr` is 10.100.0.0/16, then your value is 10.100.0.10.

Run a custom AMI due to specific security, compliance, or internal policy requirements

For more information, see [Amazon Machine Images \(AMI\)](#) in the *Amazon EC2 User Guide for Linux Instances*. The Amazon EKS AMI build specification contains resources and configuration scripts for building a custom Amazon EKS AMI based on Amazon Linux 2. For more information, see [Amazon EKS AMI Build Specification](#) on GitHub. To build custom AMIs installed with other operating systems, see [Amazon EKS Sample Custom AMIs](#) on GitHub.

Important

When specifying an AMI, Amazon EKS doesn't merge any user data. Rather, you're responsible for supplying the required bootstrap commands for nodes to join the cluster. If your nodes fail to join the cluster, the Amazon EKS `CreateNodegroup` and `UpdateNodegroupVersion` actions also fail.

The following are the limits and conditions involved with specifying an AMI ID with managed node groups:

- You must create a new node group to switch between specifying an AMI ID in a launch template and not specifying an AMI ID.
- You aren't notified in the console when a newer AMI version is available. To update your node group to a newer AMI version, you need to create a new version of your launch template with an updated AMI ID. Then, you need to update the node group with the new launch template version.
- The following fields can't be set in the API if you specify an AMI ID:
 - `amiType`
 - `releaseVersion`
 - `version`
- You can't specify a Windows AMI ID because Windows can't be used in managed node groups.

Deleting a managed node group

This topic describes how you can delete an Amazon EKS managed node group.

When you delete a managed node group, Amazon EKS first sets the minimum, maximum, and desired size of your Auto Scaling group to zero. This then causes your node group to scale down. Before each instance is terminated, Amazon EKS sends a signal to drain the pods from that node and then waits a few minutes. If the pods haven't drained after a few minutes, Amazon EKS lets Auto Scaling continue the termination of the instance. After every instance is terminated, the Auto Scaling group is deleted.

Important

If you delete a managed node group that uses a node IAM role that isn't used by any other managed node group in the cluster, the role is removed from the [aws-auth ConfigMap \(p. 402\)](#). If any of the self-managed node groups in the cluster are using the same node IAM role, the self-managed nodes move to the `NotReady` status. Additionally, the cluster operation are also disrupted. You can add the mapping back to the ConfigMap to minimize disruption.

You can delete a managed node group with `eksctl` or the AWS Management Console.

`eksctl`

To delete a managed node group with `eksctl`

- Enter the following command. Replace every *example-value* with your own values.

```
eksctl delete nodegroup \
--cluster my-cluster \
```

```
--region region-code \  
--name my-mng
```

AWS Management Console

To delete your managed node group using the AWS Management Console

1. Open the Amazon EKS console at <https://console.aws.amazon.com/eks/home#/clusters>.
2. Choose the cluster that contains the node group to delete.
3. Select the **Configuration** tab. On the **Compute** tab, select the node group to delete, and choose **Delete**.
4. On the **Delete Node group: <node group name>** page, enter the name of the node group in the text field and choose **Delete**.

Self-managed nodes

A cluster contains one or more Amazon EC2 nodes that pods are scheduled on. Amazon EKS nodes run in your AWS account and connect to the control plane of your cluster through the cluster API server endpoint. You deploy one or more nodes into a node group. A node group is one or more Amazon EC2 instances that are deployed in an [Amazon EC2 Auto Scaling group](#). All instances in a node group must have the following characteristics:

- Be the same instance type
- Be running the same Amazon Machine Image (AMI)
- Use the same [Amazon EKS node IAM role \(p. 474\)](#)

A cluster can contain several node groups. If each node group meets the previous requirements, the cluster can contain node groups that contain different instance types and host operating systems. Each node group can contain several nodes.

Amazon EKS nodes are standard Amazon EC2 instances. You're billed for them based on EC2 prices. For more information, see [Amazon EC2 pricing](#).

Amazon EKS provides specialized Amazon Machine Images (AMI) that are called Amazon EKS optimized AMIs. The AMIs are configured to work with Amazon EKS and include Docker, `kubelet`, and the AWS IAM Authenticator. The AMIs also contain a specialized [bootstrap script](#) that allows it to discover and connect to your cluster's control plane automatically.

If you restrict access to the public endpoint of your cluster using CIDR blocks, we recommend that you also enable private endpoint access. This is so that nodes can communicate with the cluster. Without the private endpoint enabled, the CIDR blocks that you specify for public access must include the egress sources from your VPC. For more information, see [Amazon EKS cluster endpoint access control \(p. 47\)](#).

To add self-managed nodes to your Amazon EKS cluster, see the topics that follow. If you launch self-managed nodes manually, add the following tag to each node. For more information, see [Adding and deleting tags on an individual resource](#). If you follow the steps in the guides that follow, the required tag is automatically added to nodes for you.

Key	Value
<code>kubernetes.io/cluster/<i>cluster-name</i></code>	owned

For more information about nodes from a general Kubernetes perspective, see [Nodes](#) in the Kubernetes documentation.

Topics

- [Launching self-managed Amazon Linux nodes \(p. 133\)](#)
- [Launching self-managed Bottlerocket nodes \(p. 138\)](#)
- [Launching self-managed Windows nodes \(p. 140\)](#)
- [Self-managed node updates \(p. 146\)](#)

Launching self-managed Amazon Linux nodes

This topic describes how you can launch an Auto Scaling group of Linux nodes that register with your Amazon EKS cluster. After the nodes join the cluster, you can deploy Kubernetes applications to them. You can also launch self-managed Amazon Linux 2 nodes with `eksctl` or the AWS Management Console.

Prerequisites

- An existing Amazon EKS cluster. To deploy one, see [Creating an Amazon EKS cluster \(p. 24\)](#). If you have subnets in the AWS Region where you have AWS Outposts, AWS Wavelength, or AWS Local Zones enabled, those subnets must not have been passed in when you created your cluster.
- (Optional, but recommended) The Amazon VPC CNI add-on configured with its own IAM role that has the necessary IAM policy attached to it. For more information, see [Configuring the Amazon VPC CNI plugin to use IAM roles for service accounts \(p. 277\)](#).
- Familiarity with the considerations listed in [Choosing an Amazon EC2 instance type \(p. 172\)](#). Depending on the instance type you choose, there may be additional prerequisites for your cluster and VPC.

`eksctl`

Prerequisite

Version 0.97.0 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 launch self-managed Linux nodes using `eksctl`

1. (Optional) If the **AmazonEKS_CNI_Policy** managed IAM policy is attached to your [the section called "Node IAM role" \(p. 474\)](#), we recommend assigning it to an IAM role that you associate to the Kubernetes `aws-node` service account instead. For more information, see [Configuring the Amazon VPC CNI plugin to use IAM roles for service accounts \(p. 277\)](#).
2. The following command creates a node group in an existing cluster. Replace every **example-value** with your own values. The nodes are created with the same Kubernetes version as the control plane, by default.

For a complete list of supported values for `--node-type`, see [amazon-eks-nodegroup.yaml](#) on GitHub. Before choosing a value for `--node-type`, review [Choosing an Amazon EC2 instance type \(p. 172\)](#).

Replace **my-key** with the name of your Amazon EC2 key pair or public key. This key is used to SSH into your nodes after they launch. If you don't already have an Amazon EC2 key pair, you can create one in the AWS Management Console. For more information, see [Amazon EC2 key pairs](#) in the *Amazon EC2 User Guide for Linux Instances*.

Create your node group with the following command.

Important

If you want to deploy a node group to AWS Outposts, AWS Wavelength, or AWS Local Zones subnets, there are additional considerations:

- The subnets must not have been passed in when you created the cluster.
- You must create the node group with a config file that specifies the subnets and `volumeType: gp2`. For more information, see [Create a nodegroup from a config file](#) and [Config file schema](#) in the `eksctl` documentation.

```
eksctl create nodegroup \
--cluster my-cluster \
--name al-nodes \
--node-type t3.medium \
--nodes 3 \
--nodes-min 1 \
--nodes-max 4 \
--ssh-access \
--managed=false \
--ssh-public-key my-key
```

To deploy a node group that allows your instance to assign a significantly higher number of IP addresses to pods, assign IP addresses to pods from a different CIDR block than that of the instance. Then, enable the `containerd` runtime you must deploy the node group using a config file. For more information, see [Increase the amount of available IP addresses for your Amazon EC2 nodes \(p. 310\)](#), [CNI custom networking \(p. 306\)](#), and [Enable the containerd runtime bootstrap flag \(p. 184\)](#). For instructions on how to deploy a private node group without outbound internet access, see [Private clusters \(p. 102\)](#). For a complete list of all available options and defaults, enter the following command.

```
eksctl create nodegroup --help
```

If nodes fail to join the cluster, then see [Nodes fail to join cluster \(p. 546\)](#) in the Troubleshooting guide.

The output is as follows. Several lines are output while the nodes are created. One of the last lines of output is the following example line.

```
[#]  created 1 nodegroup(s) in cluster "my-cluster"
```

3. We recommend blocking pod access to IMDS if the following conditions are true:
 - You plan to assign IAM roles to all of your Kubernetes service accounts so that pods only have the minimum permissions that they need.
 - No pods in the cluster require access to the Amazon EC2 instance metadata service (IMDS) for other reasons, such as retrieving the current AWS Region.

For more information, see [Restrict access to the instance profile assigned to the worker node](#).

AWS Management Console

Step 1: To launch self-managed Linux nodes using the AWS Management Console

1. Wait for your cluster status to show as ACTIVE. If you launch your nodes before the cluster is active, the nodes fail to register with the cluster and you will have to relaunch them.

2. Download the latest version of the AWS CloudFormation template.

```
curl -o amazon-eks-nodegroup.yaml https://s3.us-west-2.amazonaws.com/amazon-eks/cloudformation/2020-10-29/amazon-eks-nodegroup.yaml
```

3. Open the AWS CloudFormation console at <https://console.aws.amazon.com/cloudformation>.
4. Choose **Create stack** and then select **With new resources (standard)**.
5. For **Specify template**, select **Upload a template file** and then select **Choose file**. Select the `amazon-eks-nodegroup.yaml` file that you downloaded in step 2 and then select **Next**.
6. On the **Specify stack details** page, enter the following parameters accordingly:
 - **Stack name:** Choose a stack name for your AWS CloudFormation stack. For example, you can call it `cluster-name-nodes`. The 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.
 - **ClusterName:** Enter the name that you used when you created your Amazon EKS cluster. This name must equal the cluster name or your nodes can't join the cluster.
 - **ClusterControlPlaneSecurityGroup:** Choose the **SecurityGroups** value from the AWS CloudFormation output that you generated when you created your [VPC \(p. 265\)](#).

The following steps show one operation to retrieve the applicable group.

1. Open the Amazon EKS console at <https://console.aws.amazon.com/eks/home#/clusters>.
2. Choose the name of the cluster.
3. Choose the **Configuration** tab.
4. Choose the **Networking** tab.
5. Use the **Additional Security Group** value as a reference when selecting from the **ClusterControlPlaneSecurityGroup** dropdown list.
- **NodeGroupName:** Enter a name for your node group. This name can be used later to identify the Auto Scaling node group that's created for your nodes.
- **NodeAutoScalingGroupMinSize:** Enter the minimum number of nodes that your node Auto Scaling group can scale in to.
- **NodeAutoScalingGroupDesiredCapacity:** Enter the desired number of nodes to scale to when your stack is created.
- **NodeAutoScalingGroupMaxSize:** Enter the maximum number of nodes that your node Auto Scaling group can scale out to.
- **NodeInstanceType:** Choose an instance type for your nodes. For more information, see [Choosing an Amazon EC2 instance type \(p. 172\)](#).
- **NodeImageIdSSMParam:** Pre-populated with the Amazon EC2 Systems Manager parameter of a recent Amazon EKS optimized Amazon Linux AMI ID for a Kubernetes version. To use a different Kubernetes minor version supported with Amazon EKS, replace `1.x` with a different [supported version \(p. 67\)](#). We recommend specifying the same Kubernetes version as your cluster.

To use the Amazon EKS optimized accelerated AMI, replace `amazon-linux-2` with `amazon-linux-2-gpu`. To use the Amazon EKS optimized Arm AMI, replace `amazon-linux-2` with `amazon-linux-2-arm64`.

Note

The Amazon EKS node AMI is based on Amazon Linux 2. You can track security or privacy events for Amazon Linux 2 at the [Amazon Linux Security Center](#) or subscribe to the associated [RSS feed](#). Security and privacy events include an overview of the issue, what packages are affected, and how to update your instances to correct the issue.

- **NodeImageId:** (Optional) If you're using your own custom AMI (instead of the Amazon EKS optimized AMI), enter a node AMI ID for your AWS Region. If you specify a value here, it overrides any values in the **NodeImageIdSSMParam** field.
- **NodeVolumeSize:** Specify a root volume size for your nodes, in GiB.
- **KeyName:** Enter the name of an Amazon EC2 SSH key pair that you can use to connect using SSH into your nodes with after they launch. If you don't already have an Amazon EC2 key pair, you can create one in the AWS Management Console. For more information, see [Amazon EC2 key pairs](#) in the *Amazon EC2 User Guide for Linux Instances*.

Note

If you don't provide a key pair here, the AWS CloudFormation stack creation fails.

- **BootstrapArguments:** Specify any optional arguments to pass to the node bootstrap script, such as extra kubelet arguments. For more information, view the [bootstrap script usage information](#) on GitHub. To deploy a node group that allows your instance to assign a significantly higher number of IP addresses to pods, assign IP addresses to pods from a different CIDR block than that of the instance. Then, enable the containerd runtime or deploy a private cluster without outbound internet access. For information about arguments to add, see [Increase the amount of available IP addresses for your Amazon EC2 nodes \(p. 310\)](#), [CNI custom networking \(p. 306\)](#), [Enable the containerd runtime bootstrap flag \(p. 184\)](#), and [Private clusters \(p. 102\)](#).
- **DisableIMDSv1:** By default, each node supports the Instance Metadata Service Version 1 (IMDSv1) and IMDSv2. You can disable IMDSv1. To prevent future nodes and pods in the node group from using IMDSv1, set **DisableIMDSv1** to **true**. For more information about IMDS, see [Configuring the instance metadata service](#). For more information about restricting access to it on your nodes, see [Restrict access to the instance profile assigned to the worker node](#).
- **VpcId:** Enter the ID for the [VPC \(p. 265\)](#) that you created.
- **Subnets:** Choose the subnets that you created for your VPC. If you created your VPC using the steps that are described in [Creating a VPC for your Amazon EKS cluster \(p. 265\)](#), specify only the private subnets within the VPC for your nodes to launch into.

Important

- If any of the subnets are public subnets, then they must have the automatic public IP address assignment setting enabled. If the setting isn't enabled for the public subnet, then any nodes that you deploy to that public subnet won't be assigned a public IP address and won't be able to communicate with the cluster or other AWS services. If the subnet was deployed before March 26, 2020 using either of the [Amazon EKS AWS CloudFormation VPC templates \(p. 265\)](#), or by using `eksctl`, then automatic public IP address assignment is disabled for public subnets. For information about how to enable public IP address assignment for a subnet, see [Modifying the Public IPv4 Addressing Attribute for Your Subnet](#). If the node is deployed to a private subnet, then it's able to communicate with the cluster and other AWS services through a NAT gateway.
- If the subnets don't have internet access, make sure that you're aware of the considerations and extra steps in [Private clusters \(p. 102\)](#).
- If you're deploying the nodes in a 1.18 or earlier cluster, make sure that the subnets you select are tagged with the cluster name. Replace `my-cluster` with the name of your cluster. Then, run the following command to see a list of the subnets currently tagged with your cluster name.

```
aws ec2 describe-subnets --filters Name=tag:kubernetes.io/cluster/my-cluster,Values=shared | grep SubnetId
```

If the subnet you want to select isn't returned in the output from the previous command, manually add the tag to the subnet. For more information, see [Subnet tagging \(p. 272\)](#).

- If you select AWS Outposts, AWS Wavelength, or AWS Local Zones subnets, the subnets must not have been passed in when you created the cluster.
7. Acknowledge that the stack might create IAM resources, and then choose **Create stack**.
 8. When your stack has finished creating, select it in the console and choose **Outputs**.
 9. Record the **NodeInstanceRole** for the node group that was created. You need this when you configure your Amazon EKS nodes.

Step 2: To enable nodes to join your cluster

Note

If you launched nodes inside a private VPC without outbound internet access, make sure to enable nodes to join your cluster from within the VPC.

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

- a. Download the configuration map using the following command.

```
curl -o aws-auth-cm.yaml https://s3.us-west-2.amazonaws.com/amazon-eks/cloudformation/2020-10-29/aws-auth-cm.yaml
```

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

Important

Don't modify any other lines in this file.

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

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

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

Note

If you receive any authorization or resource type errors, see [Unauthorized or access denied \(kubectl\) \(p. 547\)](#) in the troubleshooting section.

If nodes fail to join the cluster, then see [Nodes fail to join cluster \(p. 546\)](#) in the Troubleshooting guide.

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

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

3. (GPU nodes only) If you chose a GPU instance type and the Amazon EKS optimized accelerated AMI, you must apply the [NVIDIA device plugin for Kubernetes](#) as a DaemonSet on your cluster with the following command.

```
kubectl apply -f https://raw.githubusercontent.com/NVIDIA/k8s-device-plugin/v0.9.0/nvidia-device-plugin.yaml
```

4. (Optional) Deploy a [sample application \(p. 359\)](#) to test your cluster and Linux nodes.
5. (Optional) If the [AmazonEKS_CNI_Policy](#) managed IAM policy (if you have an IPv4 cluster) or the [AmazonEKS_CNI_IPv6_Policy](#) (that you [created yourself \(p. 281\)](#) if you have an IPv6 cluster) is attached to your [the section called "Node IAM role" \(p. 474\)](#), we recommend assigning it to an IAM role that you associate to the Kubernetes aws-node service account instead. For more information, see [Configuring the Amazon VPC CNI plugin to use IAM roles for service accounts \(p. 277\)](#).
6. We recommend blocking pod access to IMDS if the following conditions are true:
 - You plan to assign IAM roles to all of your Kubernetes service accounts so that pods only have the minimum permissions that they need.
 - No pods in the cluster require access to the Amazon EC2 instance metadata service (IMDS) for other reasons, such as retrieving the current AWS Region.

For more information, see [Restrict access to the instance profile assigned to the worker node](#).

Launching self-managed Bottlerocket nodes

Note

Managed node groups might offer some advantages for your use case. For more information, see [Managed node groups \(p. 109\)](#).

This topic describes how to launch an Auto Scaling group of [Bottlerocket](#) nodes that register with your Amazon EKS cluster. Bottlerocket is a Linux-based open-source operating system from AWS that you can use for running containers on virtual machines or bare metal hosts. After the nodes join the cluster, you can deploy Kubernetes applications to them. For more information about Bottlerocket, see [Using a Bottlerocket AMI with Amazon EKS](#) on GitHub and [Custom AMI support](#) in the eksctl documentation.

For information about in-place upgrades, see [Bottlerocket Update Operator](#) on GitHub.

Important

Amazon EKS nodes are standard Amazon EC2 instances, and you are billed for them based on normal Amazon EC2 instance prices. For more information, see [Amazon EC2 pricing](#).

Important

- You can deploy to Amazon EC2 instances with x86 or Arm processors. However, you can't deploy to instances that have Inferentia chips.
- You can't deploy to the following regions: China (Beijing) (`cn-north-1`) or China (Ningxia) (`cn-northwest-1`).
- There is no AWS CloudFormation template to deploy nodes with.
- Bottlerocket images don't come with an SSH server or a shell. You can use out-of-band access methods to allow SSH enabling the admin container and to pass some bootstrapping configuration steps with user data. For more information, see these sections in the [bottlerocket README.md](#) on GitHub:
 - [Exploration](#)
 - [Admin container](#)
 - [Kubernetes settings](#)

To launch Bottlerocket nodes using eksctl

This procedure requires eksctl version 0.97.0 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 \(p. 11\)](#).

Note

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

1. Create a file named `bottlerocket.yaml` with the following contents. Replace every `example-value` with your own values. Replace `my-cluster` with the name of your cluster. To deploy on Arm instances, replace `m5.large` with an Arm instance type. Replace `my-ec2-keypair-name` with the name of an Amazon EC2 SSH key pair that you can use to connect using SSH into your nodes after they launch. If you don't already have an Amazon EC2 key pair, you can create one in the AWS Management Console. For more information, see [Amazon EC2 key pairs](#) in the [Amazon EC2 User Guide for Linux Instances](#). If your cluster is in the AWS GovCloud (US-East) or AWS GovCloud (US-East) AWS Regions, then replace `arn:aws:` with `arn:aws-us-gov:`.

If specifying an Arm Amazon EC2 instance type, then review the considerations in [Amazon EKS optimized Arm Amazon Linux AMIs \(p. 186\)](#) before deploying. For instructions on how to deploy using a custom AMI, see [Building Bottlerocket](#) on GitHub and [Custom AMI support](#) in the eksctl documentation. To deploy a managed node group, deploy a custom AMI using a launch template. For more information, see [Launch template support \(p. 124\)](#).

Important

To deploy a node group to AWS Outposts, AWS Wavelength, or AWS Local Zones subnets, don't pass AWS Outposts, AWS Wavelength, or AWS Local Zones subnets when you create the cluster. You must specify the subnets in the following example. For more information see [Create a nodegroup from a config file](#) and [Config file schema](#) in the eksctl documentation. Replace `region-code` with the AWS Region that your cluster is in.

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

metadata:
  name: my-cluster
  region: region-code
  version: '1.22'

iam:
  withOIDC: true

nodeGroups:
  - name: ng-bottlerocket
    instanceType: m5.large
    desiredCapacity: 3
    amiFamily: Bottlerocket
    ami: auto-ssm
    iam:
      attachPolicyARNs:
        - arn:aws:iam::aws:policy/AmazonEKSWorkerNodePolicy
        - arn:aws:iam::aws:policy/AmazonEC2ContainerRegistryReadOnly
        - arn:aws:iam::aws:policy/AmazonSSMManagedInstanceCore
        - arn:aws:iam::aws:policy/AmazonEKS_CNI_Policy
    ssh:
      allow: true
      publicKeyName: my-ec2-keypair-name
```

2. Deploy your nodes with the following command.

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

Example output:

Several lines are output while the nodes are created. One of the last lines of output is the following example line.

```
[#] created 1 nodegroup(s) in cluster "my-cluster"
```

3. (Optional) Create a Kubernetes [persistent volume](#) on a Bottlerocket node using the [Amazon EBS CSI Plugin](#). The default Amazon EBS driver relies on file system tools that aren't included with Bottlerocket. For more information about creating a storage class using the driver, see [Amazon EBS CSI driver \(p. 231\)](#).
4. (Optional) By default, `kube-proxy` sets the `nf_conntrack_max` kernel parameter to a default value that may differ from what Bottlerocket originally sets at boot. To keep Bottlerocket's [default setting](#), edit the `kube-proxy` configuration with the following command.

```
kubectl edit -n kube-system daemonset kube-proxy
```

Add `--conntrack-max-per-core` and `--conntrack-min` to the `kube-proxy` arguments that are in the following example. A setting of 0 implies no change.

```
containers:
- command:
  - kube-proxy
  - --v=2
  - --config=/var/lib/kube-proxy-config/config
  - --conntrack-max-per-core=0
  - --conntrack-min=0
```

5. (Optional) Deploy a [sample application \(p. 359\)](#) to test your Bottlerocket nodes.
6. We recommend blocking pod access to IMDS if the following conditions are true:
 - You plan to assign IAM roles to all of your Kubernetes service accounts so that pods only have the minimum permissions that they need.
 - No pods in the cluster require access to the Amazon EC2 instance metadata service (IMDS) for other reasons, such as retrieving the current AWS Region.

For more information, see [Restrict access to the instance profile assigned to the worker node](#).

Launching self-managed Windows nodes

This topic describes how to launch an Auto Scaling group of Windows nodes that register with your Amazon EKS cluster. After the nodes join the cluster, you can deploy Kubernetes applications to them.

Important

Amazon EKS nodes are standard Amazon EC2 instances, and you are billed for them based on normal Amazon EC2 instance prices. For more information, see [Amazon EC2 pricing](#).

Enable Windows support for your cluster. We recommend that you review important considerations before you launch a Windows node group. For more information, see [Enabling Windows support \(p. 93\)](#).

You can launch self-managed Windows nodes with `eksctl` or the AWS Management Console.

eksctl

To launch self-managed Windows nodes using eksctl

This procedure requires that you have installed eksctl, and that your eksctl version is at least 0.97.0. 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 \(p. 11\)](#).

Note

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

1. (Optional) If the [AmazonEKS_CNI_Policy](#) managed IAM policy (if you have an IPv4 cluster) or the [AmazonEKS_IPV6_Policy](#) (that you [created yourself \(p. 281\)](#) if you have an IPv6 cluster) is attached to your [the section called "Node IAM role" \(p. 474\)](#), we recommend assigning it to an IAM role that you associate to the Kubernetes aws-node service account instead. For more information, see [Configuring the Amazon VPC CNI plugin to use IAM roles for service accounts \(p. 277\)](#).
2. This procedure assumes that you have an existing cluster. If you don't already have an Amazon EKS cluster and an Amazon Linux 2 node group to add a Windows node group to, we recommend that you follow the [Getting started with Amazon EKS – eksctl \(p. 12\)](#) guide. The guide provides a complete walkthrough for how to create an Amazon EKS cluster with Amazon Linux nodes.

Create your node group with the following command. Replace every *example-value* with your own values.

Important

To deploy a node group to AWS Outposts, AWS Wavelength, or AWS Local Zones subnets, don't pass the AWS Outposts, AWS Wavelength, or AWS Local Zones subnets when you create the cluster. Create the node group with a config file, specifying the AWS Outposts, AWS Wavelength, or AWS Local Zones subnets. For more information, see [Create a nodegroup from a config file](#) and [Config file schema](#) in the eksctl documentation.

Note

Amazon EKS optimized Windows AMIs can be configured to use containerd as a runtime. When using eksctl for launching Windows nodes, specify containerRuntime as containerd in the node group configuration. For more information, see [Enable the containerd runtime bootstrap flag \(p. 218\)](#) in this user guide or [Define container runtime](#) in the eksctl documentation. Replace *region-code* with the AWS Region that your cluster is in.

```
eksctl create nodegroup \
--region region-code \
--cluster my-cluster \
--name ng-windows \
--node-type t2.large \
--nodes 3 \
--nodes-min 1 \
--nodes-max 4 \
--node-ami-family WindowsServer2019FullContainer
```

Note

- If nodes fail to join the cluster, see [Nodes fail to join cluster \(p. 546\)](#) in the Troubleshooting guide.

- To see the available options for `eksctl` commands, enter the following command.

```
eksctl command -help
```

The output is as follows. Several lines are output while the nodes are created. One of the last lines of output is the following example line.

```
[#] created 1 nodegroup(s) in cluster "my-cluster"
```

3. (Optional) Deploy a [sample application \(p. 359\)](#) to test your cluster and Windows nodes.
4. We recommend blocking pod access to IMDS if the following conditions are true:
 - You plan to assign IAM roles to all of your Kubernetes service accounts so that pods only have the minimum permissions that they need.
 - No pods in the cluster require access to the Amazon EC2 instance metadata service (IMDS) for other reasons, such as retrieving the current AWS Region.

For more information, see [Restrict access to the instance profile assigned to the worker node](#).

AWS Management Console

Prerequisites

- An existing Amazon EKS cluster and a Linux node group. If you don't have these resources, we recommend that you follow one of our [Getting started with Amazon EKS \(p. 4\)](#) guides to create them. The guides describe how to create an Amazon EKS cluster with Linux nodes.
- An existing VPC and security group that meet the requirements for an Amazon EKS cluster. For more information, see [Cluster VPC and subnet considerations \(p. 269\)](#) and [Amazon EKS security group considerations \(p. 272\)](#). The [Getting started with Amazon EKS \(p. 4\)](#) guide creates a VPC that meets the requirements. Alternatively, you can also follow [Creating a VPC for your Amazon EKS cluster \(p. 265\)](#) to create one manually.
- An existing Amazon EKS cluster that uses a VPC and security group that meets the requirements of an Amazon EKS cluster. For more information, see [Creating an Amazon EKS cluster \(p. 24\)](#). If you have subnets in the AWS Region where you have AWS Outposts, AWS Wavelength, or AWS Local Zones enabled, those subnets must not have been passed in when you created the cluster.

Step 1: To launch self-managed Windows nodes using the AWS Management Console

1. Wait for your cluster status to show as ACTIVE. If you launch your nodes before the cluster is active, the nodes fail to register with the cluster and you need to relaunch them.
2. Open the AWS CloudFormation console at <https://console.aws.amazon.com/cloudformation>
3. Choose **Create stack**.
4. For **Specify template**, select **Amazon S3 URL**, copy the following URL, paste it into **Amazon S3 URL**, and select **Next** twice.

```
https://s3.us-west-2.amazonaws.com/amazon-eks/cloudformation/2020-10-29/amazon-eks-windows-nodegroup.yaml
```

5. On the **Quick create stack** page, enter the following parameters accordingly:

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

- **ClusterName:** Enter the name that you used when you created your Amazon EKS cluster.

Important

This name must exactly match the name that you used in [Step 1: Create your Amazon EKS cluster \(p. 16\)](#). Otherwise, your nodes can't join the cluster.

- **ClusterControlPlaneSecurityGroup:** Choose the security group from the AWS CloudFormation output that you generated when you created your [VPC \(p. 265\)](#).

The following steps show one method to retrieve the applicable group.

1. Open the Amazon EKS console at <https://console.aws.amazon.com/eks/home#/clusters>.
2. Choose the name of the cluster.
3. Choose the **Configuration** tab.
4. Choose the **Networking** tab.
5. Use the **Additional Security Group** value as a reference when selecting from the **ClusterControlPlaneSecurityGroup** dropdown list.

- **NodeGroupName:** Enter a name for your node group. This name can be used later to identify the Auto Scaling node group that's created for your nodes.
- **NodeAutoScalingGroupMinSize:** Enter the minimum number of nodes that your node Auto Scaling group can scale in to.
- **NodeAutoScalingGroupDesiredCapacity:** Enter the desired number of nodes to scale to when your stack is created.
- **NodeAutoScalingGroupMaxSize:** Enter the maximum number of nodes that your node Auto Scaling group can scale out to.
- **NodeInstanceType:** Choose an instance type for your nodes. For more information, see [Choosing an Amazon EC2 instance type \(p. 172\)](#).

Note

The supported instance types for the latest version of the [Amazon VPC CNI plugin for Kubernetes](#) are listed in [vpc_ip_resource_limit.go](#) on GitHub. You might need to update your CNI version to use the latest supported instance types. For more information, see [Updating the Amazon VPC CNI self-managed add-on \(p. 287\)](#).

- **NodeImageIdSSMParam:** Pre-populated with the Amazon EC2 Systems Manager parameter of the current recommended Amazon EKS optimized Windows Core AMI ID. To use the full version of Windows, replace **Core** with **Full**.
- **NodeImageId:** (Optional) If you're using your own custom AMI (instead of the Amazon EKS optimized AMI), enter a node AMI ID for your AWS Region. If you specify a value for this field, it overrides any values in the **NodeImageIdSSMParam** field.
- **NodeVolumeSize:** Specify a root volume size for your nodes, in GiB.
- **KeyName:** Enter the name of an Amazon EC2 SSH key pair that you can use to connect using SSH into your nodes after they launch. If you don't already have an Amazon EC2 key pair, you can create one in the AWS Management Console. For more information, see [Amazon EC2 key pairs](#) in the *Amazon EC2 User Guide for Windows Instances*.

Note

If you don't provide a key pair here, the AWS CloudFormation stack fails to be created.

- **BootstrapArguments:** Specify any optional arguments to pass to the node bootstrap script, such as extra kubelet arguments using `-KubeletExtraArgs`.

Note

You can configure Amazon EKS optimized Windows AMIs to use `containerd` as a runtime. When using an AWS CloudFormation template to create Windows nodes, specify `-ContainerRuntime containerd` in a bootstrap argument to enable the `containerd` runtime. For more information, see [Enable the containerd runtime bootstrap flag \(p. 218\)](#).

- **DisableIMDSv1:** By default, each node supports the Instance Metadata Service Version 1 (IMDSv1) and IMDSv2. You can disable IMDSv1. To prevent future nodes and pods in the node group from using MDSv1, set **DisableIMDSv1** to **true**. For more information about IMDS, see [Configuring the instance metadata service](#).
- **VpcId:** Select the ID for the [VPC \(p. 265\)](#) that you created.
- **NodeSecurityGroups:** Select the security group that was created for your Linux node group when you created your [VPC \(p. 265\)](#). If your Linux nodes have more than one security group attached to them, specify all of them. This for, for example, if the Linux node group was created with eksctl.
- **Subnets:** Choose the subnets that you created. If you created your VPC using the steps in [Creating a VPC for your Amazon EKS cluster \(p. 265\)](#), then specify only the private subnets within the VPC for your nodes to launch into.

Important

- If any of the subnets are public subnets, then they must have the automatic public IP address assignment setting enabled. If the setting isn't enabled for the public subnet, then any nodes that you deploy to that public subnet won't be assigned a public IP address and won't be able to communicate with the cluster or other AWS services. If the subnet was deployed before March 26, 2020 using either of the [Amazon EKS AWS CloudFormation VPC templates \(p. 265\)](#), or by using eksctl, then automatic public IP address assignment is disabled for public subnets. For information about how to enable public IP address assignment for a subnet, see [Modifying the Public IPv4 Addressing Attribute for Your Subnet](#). If the node is deployed to a private subnet, then it's able to communicate with the cluster and other AWS services through a NAT gateway.
- If the subnets don't have internet access, then make sure that you're aware of the considerations and extra steps in [Private clusters \(p. 102\)](#).
- If you're deploying the nodes in a 1.18 or earlier cluster, make sure that the subnets you select are tagged with the cluster name. Replace *my-cluster* with the name of your cluster. Then, run the following command to see a list of the subnets currently tagged with your cluster name.

```
aws ec2 describe-subnets --filters Name=tag:kubernetes.io/cluster/my-cluster,Values=shared | grep SubnetId
```

If the subnet you want to select isn't returned in the output from the previous command, manually add the tag to the subnet. For more information, see [Subnet tagging \(p. 272\)](#).

- If you select AWS Outposts, AWS Wavelength, or AWS Local Zones subnets, then the subnets must not have been passed in when you created the cluster.
6. Acknowledge that the stack might create IAM resources, and then choose **Create stack**.
 7. When your stack has finished creating, select it in the console and choose **Outputs**.
 8. Record the **NodeInstanceRole** for the node group that was created. You need this when you configure your Amazon EKS Windows nodes.

Step 2: To enable nodes to join your cluster

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

```
curl -o aws-auth-cm-windows.yaml https://s3.us-west-2.amazonaws.com/amazon-eks/cloudformation/2020-10-29/aws-auth-cm-windows.yaml
```

- b. Open the file using your preferred text editor. Replace the *ARN of instance role (not instance profile) of **Linux** node* and *ARN of instance role (not instance profile) of **Windows** node* snippets with the **NodeInstanceRole** values that you recorded for your Linux and Windows nodes, and save the file.

Important

- Don't modify any other lines in this file.
- Don't use the same IAM role for both Windows and Linux nodes.

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

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

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

Note

If you receive any authorization or resource type errors, see [Unauthorized or access denied \(kubectl\) \(p. 547\)](#) in the troubleshooting section.

If nodes fail to join the cluster, then see [Nodes fail to join cluster \(p. 546\)](#) in the Troubleshooting guide.

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

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

3. (Optional) Deploy a [sample application \(p. 359\)](#) to test your cluster and Windows nodes.
4. (Optional) If the **AmazonEKS_CNI_Policy** managed IAM policy (if you have an IPv4 cluster) or the **AmazonEKS_CNI_IPv6_Policy** (that you [created yourself \(p. 281\)](#) if you have an IPv6 cluster) is attached to your [the section called "Node IAM role" \(p. 474\)](#), we recommend assigning it to an IAM role that you associate to the Kubernetes aws-node service account instead. For more information, see [Configuring the Amazon VPC CNI plugin to use IAM roles for service accounts \(p. 277\)](#).
5. We recommend blocking pod access to IMDS if the following conditions are true:
 - You plan to assign IAM roles to all of your Kubernetes service accounts so that pods only have the minimum permissions that they need.
 - No pods in the cluster require access to the Amazon EC2 instance metadata service (IMDS) for other reasons, such as retrieving the current AWS Region.

For more information, see [Restrict access to the instance profile assigned to the worker node](#).

Self-managed node updates

When a new Amazon EKS optimized AMI is released, consider replacing the nodes in your self-managed node group with the new AMI. Likewise, if you have updated the Kubernetes version for your Amazon EKS cluster, update the nodes to use nodes with the same Kubernetes version.

Important

This topic covers node updates for self-managed nodes. If you are using [Managed node groups \(p. 109\)](#), see [Updating a managed node group \(p. 119\)](#).

There are two basic ways to update self-managed node groups in your clusters to use a new AMI:

- [Migrating to a new node group \(p. 146\)](#) – Create a new node group and migrate your pods to that group. Migrating to a new node group is more graceful than simply updating the AMI ID in an existing AWS CloudFormation stack. This is because the migration process taints the old node group as `NoSchedule` and drains the nodes after a new stack is ready to accept the existing pod workload.
- [Updating an existing self-managed node group \(p. 151\)](#) – Update the AWS CloudFormation stack for an existing node group to use the new AMI. This method isn't supported for node groups that were created with `eksctl`.

Migrating to a new node group

This topic describes how you can create a new node group, gracefully migrate your existing applications to the new group, and remove the old node group from your cluster. You can migrate to a new node group using `eksctl` or the AWS Management Console.

To migrate your applications to a new node group with `eksctl`

This procedure requires `eksctl` version 0.97.0 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 \(p. 11\)](#).

Note

This procedure only works for clusters and node groups that were created with `eksctl`.

1. Retrieve the name of your existing node groups, replacing `my-cluster` with your cluster name.

```
eksctl get nodegroups --cluster=my-cluster
```

Example output:

CLUSTER	NODEGROUP	CREATED	MIN SIZE	MAX SIZE
DESIRED CAPACITY	INSTANCE TYPE	IMAGE ID		
default	standard-nodes	2019-05-01T22:26:58Z	1	4
	t3.medium	ami-05a71d034119fffc12		3

2. Launch a new node group with `eksctl` with the following command. In the command, replace every `example-value` with your own values. The version number can't be later than the Kubernetes version for your control plane. Also, it can't be more than two minor versions earlier than the

Kubernetes version for your control plane. We recommend that you use the same version as your control plane.

We recommend blocking pod access to IMDS if the following conditions are true:

- You plan to assign IAM roles to all of your Kubernetes service accounts so that pods only have the minimum permissions that they need.
- No pods in the cluster require access to the Amazon EC2 instance metadata service (IMDS) for other reasons, such as retrieving the current AWS Region.

For more information, see [Restrict access to the instance profile assigned to the worker node](#).

To block pod access to IMDS, add the `--disable-pod-imds` option to the following command.

Note

For more available flags and their descriptions, see <https://eksctl.io/>.

```
eksctl create nodegroup \
--cluster my-cluster \
--version 1.22 \
--name standard-nodes-new \
--node-type t3.medium \
--nodes 3 \
--nodes-min 1 \
--nodes-max 4 \
--node-ami auto
```

3. When the previous command completes, verify that all of your nodes have reached the Ready state with the following command:

```
kubectl get nodes
```

4. Delete the original node group with the following command. In the command, replace every `example-value` with your cluster and node group names:

```
eksctl delete nodegroup --cluster my-cluster --name standard-nodes
```

To migrate your applications to a new node group with the AWS Management Console and AWS CLI

1. Launch a new node group by following the steps that are outlined in [Launching self-managed Amazon Linux nodes \(p. 133\)](#).
2. When your stack has finished creating, select it in the console and choose **Outputs**.
3. Record the **NodeInstanceRole** for the node group that was created. You need this to add the new Amazon EKS nodes to your cluster.

Note

If you attached any additional IAM policies to your old node group IAM role, attach those same policies to your new node group IAM role to maintain that functionality on the new group. This applies to you if you added permissions for the Kubernetes [Cluster Autoscaler](#), for example.

4. Update the security groups for both node groups so that they can communicate with each other. For more information, see [Amazon EKS security group considerations \(p. 272\)](#).
 - a. Record the security group IDs for both node groups. This is shown as the **NodeSecurityGroup** value in the AWS CloudFormation stack outputs.

You can use the following AWS CLI commands to get the security group IDs from the stack names. In these commands, `oldNodes` is the AWS CloudFormation stack name for your older node stack, and `newNodes` is the name of the stack that you are migrating to. Replace every `example-value` with your own values.

```
oldNodes="old_node_CFN_stack_name"  
newNodes="new_node_CFN_stack_name"  
  
oldSecGroup=$(aws cloudformation describe-stack-resources --stack-name $oldNodes \  
--query 'StackResources[?  
ResourceType==`AWS::EC2::SecurityGroup`].PhysicalResourceId' \  
--output text)  
newSecGroup=$(aws cloudformation describe-stack-resources --stack-name $newNodes \  
--query 'StackResources[?  
ResourceType==`AWS::EC2::SecurityGroup`].PhysicalResourceId' \  
--output text)
```

- b. Add ingress rules to each node security group so that they accept traffic from each other.

The following AWS CLI commands add inbound rules to each security group that allow all traffic on all protocols from the other security group. This configuration allows pods in each node group to communicate with each other while you're migrating your workload to the new group.

```
aws ec2 authorize-security-group-ingress --group-id $oldSecGroup \  
--source-group $newSecGroup --protocol -1  
aws ec2 authorize-security-group-ingress --group-id $newSecGroup \  
--source-group $oldSecGroup --protocol -1
```

5. Edit the `aws-auth` configmap to map the new node instance role in RBAC.

```
kubectl edit configmap -n kube-system aws-auth
```

Add a new `mapRoles` entry for the new node group. If your cluster is in the AWS GovCloud (US-East) or AWS GovCloud (US-East) AWS Regions, then replace `arn:aws:` with `arn:aws-us-gov:`

```
apiVersion: v1  
data:  
mapRoles: |  
- rolearn: ARN of instance role (not instance profile)  
username: system:node:{${EC2PrivateDNSName}}  
groups:  
- system:bootstrappers  
- system:nodes  
- rolearn: arn:aws:iam::111122223333:role/nodes-1-16-NodeInstanceRole-U11V27W93CX5  
username: system:node:{${EC2PrivateDNSName}}  
groups:  
- system:bootstrappers  
- system:nodes
```

Replace the `ARN of instance role (not instance profile)` snippet with the `NodeInstanceRole` value that you recorded in a previous step (p. 147). Then, save and close the file to apply the updated configmap.

6. Watch the status of your nodes and wait for your new nodes to join your cluster and reach the Ready status.

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

7. (Optional) If you're using the Kubernetes [Cluster Autoscaler](#), scale the deployment down to zero (0) replicas to avoid conflicting scaling actions.

```
kubectl scale deployments/cluster-autoscaler --replicas=0 -n kube-system
```

8. Use the following command to taint each of the nodes that you want to remove with `NoSchedule`. This is so that new pods aren't scheduled or rescheduled on the nodes that you're replacing.

```
kubectl taint nodes node_name key=value:NoSchedule
```

If you're upgrading your nodes to a new Kubernetes version, you can identify and taint all of the nodes of a particular Kubernetes version (in this case, 1.20) with the following code snippet. The version number can't be later than the Kubernetes version of your control plane. It also can't be more than two minor versions earlier than the Kubernetes version of your control plane. We recommend that you use the same version as your control plane.

```
K8S_VERSION=1.20
nodes=$(kubectl get nodes -o jsonpath=".items[?(@.status.nodeInfo.kubeletVersion==\"v$K8S_VERSION\")]".metadata.name")
for node in ${nodes[@]}
do
    echo "Tainting $node"
    kubectl taint nodes $node key=value:NoSchedule
done
```

9. Determine your cluster's DNS provider.

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

The following is the output. This cluster is using `coredns` for DNS resolution, but your cluster can return `kube-dns` instead:

NAME	DESIRED	CURRENT	UP-TO-DATE	AVAILABLE	AGE
coredns	1	1	1	1	31m

10. If your current deployment is running fewer than two replicas, scale out the deployment to two replicas. Replace `coredns` with `kubedns` if your previous command output returned that instead.

```
kubectl scale deployments/coredns --replicas=2 -n kube-system
```

11. Drain each of the nodes that you want to remove from your cluster with the following command:

```
kubectl drain node_name --ignore-daemonsets --delete-local-data
```

If you're upgrading your nodes to a new Kubernetes version, identify and drain all of the nodes of a particular Kubernetes version (in this case, 1.20) with the following code snippet.

```
K8S_VERSION=1.20
nodes=$(kubectl get nodes -o jsonpath=".items[?(@.status.nodeInfo.kubeletVersion==\"v$K8S_VERSION\")]".metadata.name")
for node in ${nodes[@]}
do
    echo "Draining $node"
    kubectl drain $node --ignore-daemonsets --delete-local-data
done
```

12. After your old nodes finished draining, revoke the security group inbound rules you authorized earlier. Then, delete the AWS CloudFormation stack to terminate the instances.

Note

If you attached any additional IAM policies to your old node group IAM role, such as adding permissions for the Kubernetes [Cluster Autoscaler](#), detach those additional policies from the role before you can delete your AWS CloudFormation stack.

- a. Revoke the inbound rules that you created for your node security groups earlier. In these commands, `oldNodes` is the AWS CloudFormation stack name for your older node stack, and `newNodes` is the name of the stack that you are migrating to.

```
oldNodes="old_node_CFN_stack_name"  
newNodes="new_node_CFN_stack_name"  
  
oldSecGroup=$(aws cloudformation describe-stack-resources --stack-name $oldNodes \  
--query 'StackResources[?  
ResourceType==`AWS::EC2::SecurityGroup`].PhysicalResourceId' \  
--output text)  
newSecGroup=$(aws cloudformation describe-stack-resources --stack-name $newNodes \  
--query 'StackResources[?  
ResourceType==`AWS::EC2::SecurityGroup`].PhysicalResourceId' \  
--output text)  
aws ec2 revoke-security-group-ingress --group-id $oldSecGroup \  
--source-group $newSecGroup --protocol -1  
aws ec2 revoke-security-group-ingress --group-id $newSecGroup \  
--source-group $oldSecGroup --protocol -1
```

- b. Open the AWS CloudFormation console at <https://console.aws.amazon.com/cloudformation>.
- c. Select your old node stack.
- d. Choose **Delete**.
- e. In the **Delete stack** confirmation dialog box, choose **Delete stack**.

13. Edit the `aws-auth` configmap to remove the old node instance role from RBAC.

```
kubectl edit configmap -n kube-system aws-auth
```

Delete the `mapRoles` entry for the old node group. If your cluster is in the AWS GovCloud (US-East) or AWS GovCloud (US-East) AWS Regions, then replace `arn:aws:` with `arn:aws-us-gov:`

```
apiVersion: v1  
data:  
  mapRoles: |  
    - rolearn: arn:aws:iam::111122223333:role/nodes-1-16-NodeInstanceRole-W70725MZQFF8  
      username: system:node:{EC2PrivateDNSName}  
      groups:  
        - system:bootstrappers  
        - system:nodes  
    - rolearn: arn:aws:iam::111122223333:role/nodes-1-15-NodeInstanceRole-U11V27W93CX5  
      username: system:node:{EC2PrivateDNSName}  
      groups:  
        - system:bootstrappers  
        - system:nodes>
```

Save and close the file to apply the updated configmap.

14. (Optional) If you are using the Kubernetes [Cluster Autoscaler](#), scale the deployment back to one replica.

Note

You must also tag your new Auto Scaling group appropriately (for example, `k8s.io/cluster-autoscaler/enabled`, `k8s.io/cluster-autoscaler/my-cluster`) and update the command for your Cluster Autoscaler deployment to point to the newly tagged Auto Scaling group. For more information, see [Cluster Autoscaler on AWS](#).

```
kubectl scale deployments/cluster-autoscaler --replicas=1 -n kube-system
```

15. (Optional) Verify that you're using the latest version of the [Amazon VPC CNI plugin for Kubernetes](#). You might need to update your CNI version to use the latest supported instance types. For more information, see [Updating the Amazon VPC CNI self-managed add-on \(p. 287\)](#).
16. If your cluster is using `kube-dns` for DNS resolution (see [previous step \(p. 149\)](#)), scale in the `kube-dns` deployment to one replica.

```
kubectl scale deployments/kube-dns --replicas=1 -n kube-system
```

Updating an existing self-managed node group

This topic describes how you can update an existing AWS CloudFormation self-managed node stack with a new AMI. You can use this procedure to update your nodes to a new version of Kubernetes following a cluster update. Otherwise, you can update to the latest Amazon EKS optimized AMI for an existing Kubernetes version.

Important

This topic covers node updates for self-managed nodes. For information about using [Managed node groups \(p. 109\)](#), see [Updating a managed node group \(p. 119\)](#).

The latest default Amazon EKS node AWS CloudFormation template is configured to launch an instance with the new AMI into your cluster before removing an old one, one at a time. This configuration ensures that you always have your Auto Scaling group's desired count of active instances in your cluster during the rolling update.

Note

This method isn't supported for node groups that were created with `eksctl`. If you created your cluster or node group with `eksctl`, see [Migrating to a new node group \(p. 146\)](#).

To update an existing node group

1. Determine the DNS provider for your cluster.

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

The output is as follows. This cluster is using `coredns` for DNS resolution, but your cluster might return `kube-dns` instead.

NAME	DESIRED	CURRENT	UP-TO-DATE	AVAILABLE	AGE
<code>coredns</code>	1	1	1	1	31m

2. If your current deployment is running fewer than two replicas, scale out the deployment to two replicas. Replace `coredns` with `kube-dns` if your previous command output returned that instead.

```
kubectl scale deployments/coredns --replicas=2 -n kube-system
```

3. (Optional) If you're using the Kubernetes [Cluster Autoscaler](#), scale the deployment down to zero (0) replicas to avoid conflicting scaling actions.

```
kubectl scale deployments/cluster-autoscaler --replicas=0 -n kube-system
```

4. Determine the instance type and desired instance count of your current node group. You enter these values later when you update the AWS CloudFormation template for the group.
 - a. Open the Amazon EC2 console at <https://console.aws.amazon.com/ec2/>.
 - b. In the left navigation pane, choose **Launch Configurations**, and note the instance type for your existing node launch configuration.
 - c. In the left navigation pane, choose **Auto Scaling Groups**, and note the **Desired** instance count for your existing node Auto Scaling group.
5. Open the AWS CloudFormation console at <https://console.aws.amazon.com/cloudformation>.
6. Select your node group stack, and then choose **Update**.
7. Select **Replace current template** and select **Amazon S3 URL**.
8. For **Amazon S3 URL**, paste the following URL into the text area to ensure that you're using the latest version of the node AWS CloudFormation template. Then, choose **Next**:

```
https://s3.us-west-2.amazonaws.com/amazon-eks/cloudformation/2020-10-29/amazon-eks-nodegroup.yaml
```

9. On the **Specify stack details** page, fill out the following parameters, and choose **Next**:
 - **NodeAutoScalingGroupDesiredCapacity** – Enter the desired instance count that you recorded in a [previous step \(p. 152\)](#). Or, enter your new desired number of nodes to scale to when your stack is updated.
 - **NodeAutoScalingGroupMaxSize** – Enter the maximum number of nodes to which your node Auto Scaling group can scale out. This value must be at least one node more than your desired capacity. This is so that you can perform a rolling update of your nodes without reducing your node count during the update.
 - **NodeInstanceType** – Choose the instance type you recorded in a [previous step \(p. 152\)](#). Alternatively, choose a different instance type for your nodes. Before choosing a different instance type, review [Choosing an Amazon EC2 instance type \(p. 172\)](#). Each Amazon EC2 instance type supports a maximum number of elastic network interfaces (ENIs) and each ENI supports a maximum number of IP addresses. Because each worker node and pod is assigned its own IP address, it's important to choose an instance type that will support the maximum number of pods that you want to run on each worker node. For a list of the number of ENIs and IP addresses supported by instance types, see [IP addresses per network interface per instance type](#). For example, the m5.large instance type supports a maximum of 30 IP addresses for the worker node and pods.

Note

The supported instance types for the latest version of the [Amazon VPC CNI plugin for Kubernetes](#) are shown in [vpc_ip_resource_limit.go](#) on GitHub. You might need to update your CNI version to use the latest supported instance types. For more information, see [Updating the Amazon VPC CNI self-managed add-on \(p. 287\)](#).

Important

Some instance types might not be available in all AWS Regions.

- **NodeImageIdSSMParam** – The Amazon EC2 Systems Manager parameter of the AMI ID that you want to update to. The following value uses the latest Amazon EKS optimized AMI for Kubernetes version 1.22.

```
/aws/service/eks/optimized-ami/1.22/amazon-linux-2/recommended/image_id
```

You can replace [1.22](#) with a [supported Kubernetes version \(p. 76\)](#) that's the same. Or, it should be up to one version earlier than the Kubernetes version running on your control plane. We recommend that you keep your nodes at the same version as your control plane. If you want to use the Amazon EKS optimized accelerated AMI, then replace [amazon-linux-2](#) with [amazon-linux-2-gpu](#).

Note

Using the Amazon EC2 Systems Manager parameter enables you to update your nodes in the future without having to look up and specify an AMI ID. If your AWS CloudFormation stack is using this value, any stack update always launches the latest recommended Amazon EKS optimized AMI for your specified Kubernetes version. This is even the case even if you don't change any values in the template.

- **NodeImageId** – To use your own custom AMI, enter the ID for the AMI to use.

Important

This value overrides any value specified for **NodeImageIdSSMParam**. If you want to use the **NodeImageIdSSMParam** value, ensure that the value for **NodeImageId** is blank.

- **DisableIMDSv1** – By default, each node supports the Instance Metadata Service Version 1 (IMDSv1) and IMDSv2. However, you can disable IMDSv1. Select **true** if you don't want any nodes or any pods scheduled in the node group to use IMDSv1. For more information about IMDS, see [Configuring the instance metadata service](#). If you've implemented IAM roles for service accounts, assign necessary permissions directly to all pods that require access to AWS services. This way, no pods in your cluster require access to IMDS for other reasons, such as retrieving the current AWS Region. Then, you can also disable access to IMDSv2 for pods that don't use host networking. For more information, see [Restrict access to the instance profile assigned to the worker node](#).

10. (Optional) On the **Options** page, tag your stack resources. Choose **Next**.
11. On the **Review** page, review your information, acknowledge that the stack might create IAM resources, and then choose **Update stack**.

Note

The update of each node in the cluster takes several minutes. Wait for the update of all nodes to complete before performing the next steps.

12. If your cluster's DNS provider is `kube-dns`, scale in the `kube-dns` deployment to one replica.

```
kubectl scale deployments/kube-dns --replicas=1 -n kube-system
```

13. (Optional) If you are using the Kubernetes [Cluster Autoscaler](#), scale the deployment back to your desired amount of replicas.

```
kubectl scale deployments/cluster-autoscaler --replicas=1 -n kube-system
```

14. (Optional) Verify that you're using the latest version of the [Amazon VPC CNI plugin for Kubernetes](#). You might need to update your CNI version to use the latest supported instance types. For more information, see [Updating the Amazon VPC CNI self-managed add-on \(p. 287\)](#).

AWS Fargate

This topic discusses using Amazon EKS to run Kubernetes pods on AWS Fargate.

AWS Fargate is a technology that provides on-demand, right-sized compute capacity for [containers](#). With AWS Fargate, you don't have to provision, configure, or scale groups of virtual machines on your own to run containers. You also don't need to choose server types, decide when to scale your node groups, or optimize cluster packing. You can control which pods start on Fargate and how they run with [Fargate profiles \(p. 158\)](#). Fargate profiles are defined as part of your Amazon EKS cluster.

Amazon EKS integrates Kubernetes with AWS Fargate by using controllers that are built by AWS using the upstream, extensible model provided by Kubernetes. These controllers run as part of the Amazon EKS managed Kubernetes control plane and are responsible for scheduling native Kubernetes pods onto Fargate. The Fargate controllers include a new scheduler that runs alongside the default Kubernetes scheduler in addition to several mutating and validating admission controllers. When you start a pod that meets the criteria for running on Fargate, the Fargate controllers that are running in the cluster recognize, update, and schedule the pod onto Fargate.

This topic describes the different components of pods that run on Fargate, and calls out special considerations for using Fargate with Amazon EKS.

AWS Fargate considerations

Here are some things to consider about using Fargate on Amazon EKS.

- AWS Fargate with Amazon EKS is available in all Amazon EKS Regions except China (Beijing), China (Ningxia), AWS GovCloud (US-East), and AWS GovCloud (US-West).
- Each pod that runs on Fargate has its own isolation boundary. They don't share the underlying kernel, CPU resources, memory resources, or elastic network interface with another pod.
- Network Load Balancers and Application Load Balancers (ALBs) can be used with Fargate with IP targets only. For more information, see [Create a network load balancer \(p. 374\)](#) and [Application load balancing on Amazon EKS \(p. 378\)](#).
- Fargate exposed services only run on target type IP mode, and not on node IP mode. The recommended way to check the connectivity from a service running on a managed node and a service running on Fargate is to connect via service name.
- Pods must match a Fargate profile at the time that they're scheduled to run on Fargate. Pods that don't match a Fargate profile might be stuck as Pending. If a matching Fargate profile exists, you can delete pending pods that you have created to reschedule them onto Fargate.
- You can only use [Security groups for pods \(p. 313\)](#) with pods that run on Fargate that are part of a 1.18 or later cluster.
- Daemonsets aren't supported on Fargate. If your application requires a daemon, reconfigure that daemon to run as a sidecar container in your pods.
- Privileged containers aren't supported on Fargate.
- Pods running on Fargate can't specify `HostPort` or `HostNetwork` in the pod manifest.
- The default `nofile` and `nproc` soft limit is 1024 and the hard limit is 65535 for Fargate pods.
- GPUs aren't currently available on Fargate.
- Pods that run on Fargate are only supported on private subnets (with NAT gateway access to AWS services, but not a direct route to an Internet Gateway), so your cluster's VPC must have private subnets available. For clusters without outbound internet access, see [Private clusters \(p. 102\)](#).
- You can use the [Vertical Pod Autoscaler \(p. 366\)](#) to initially right size the CPU and memory for your Fargate pods, and then use the [Horizontal Pod Autoscaler \(p. 370\)](#) to scale those pods. If you want the Vertical Pod Autoscaler to automatically re-deploy pods to Fargate with larger CPU and memory combinations, set the mode for the Vertical Pod Autoscaler to either Auto or Recreate to ensure correct functionality. For more information, see the [Vertical Pod Autoscaler](#) documentation on GitHub.
- DNS resolution and DNS hostnames must be enabled for your VPC. For more information, see [Viewing and updating DNS support for your VPC](#).
- Amazon EKS Fargate adds defense-in-depth for Kubernetes applications by isolating each Pod within a Virtual Machine (VM). This VM boundary prevents access to host-based resources used by other Pods in the event of a container escape, which is a common method of attacking containerized applications and gain access to resources outside of the container.

Using Amazon EKS doesn't change your responsibilities under the [shared responsibility model \(p. 438\)](#). You should carefully consider the configuration of cluster security and governance controls. The safest way to isolate an application is always to run it in a separate cluster.

- Fargate profiles support specifying subnets from VPC secondary CIDR blocks. You might want to specify a secondary CIDR block. This is because there's a limited number of IP addresses available in a subnet. As a result, there's also a limited number of pods that can be created in the cluster. By using different subnets for pods, you can increase the number of available IP addresses. For more information, see [Adding IPv4 CIDR blocks to a VPC](#).
- The Amazon EC2 instance metadata service (IMDS) isn't available to pods that are deployed to Fargate nodes. If you have pods that are deployed to Fargate that need IAM credentials, assign them to your pods using [IAM roles for service accounts \(p. 442\)](#). If your pods need access to other information available through IMDS, then you must hard code this information into your pod spec. This includes the AWS Region or Availability Zone that a pod is deployed to.
- You can't deploy Fargate pods to AWS Outposts, AWS Wavelength or AWS Local Zones.
- Amazon EKS must periodically patch AWS Fargate pods to keep them secure. We attempt the updates in a way that reduces impact, but there are times when pods must be deleted if they aren't successfully evicted. There are some actions you can take to minimize disruption. For more information, see [Fargate pod patching \(p. 163\)](#).

Getting started with AWS Fargate using Amazon EKS

This topic helps you to get started running pods on AWS Fargate with your Amazon EKS cluster.

If you restrict access to the public endpoint of your cluster using CIDR blocks, we recommend that you also enable private endpoint access so that Fargate pods can communicate with the cluster. Without the private endpoint enabled, the CIDR blocks that you specify for public access must include the egress sources from your VPC. For more information, see [Amazon EKS cluster endpoint access control \(p. 47\)](#).

Prerequisite

An existing cluster. AWS Fargate with Amazon EKS is available in all Amazon EKS Regions except China (Beijing), China (Ningxia), AWS GovCloud (US-East), and AWS GovCloud (US-West). If you don't already have an Amazon EKS cluster, see [Getting started with Amazon EKS \(p. 4\)](#).

Ensure that existing nodes can communicate with Fargate pods

If you're working with a new cluster with no nodes, or a cluster with only [managed node groups \(p. 109\)](#), you can skip to [Create a Fargate pod execution role \(p. 156\)](#).

Assume that you're working with an existing cluster that already has nodes that are associated with it. You should make sure that pods on these nodes can communicate freely with pods running on Fargate. Pods running on Fargate are automatically configured to use the cluster security group for the cluster that they're associated with. Ensure that any existing nodes in your cluster can send and receive traffic to and from the cluster security group. [Managed node groups \(p. 109\)](#) are automatically configured to use the cluster security group as well, so you don't need to modify or check them for this compatibility.

For existing node groups that were created with eksctl or the Amazon EKS managed AWS CloudFormation templates, you can add the cluster security group to the nodes manually. Or, alternatively, you can modify the Auto Scaling group launch template for the node group to attach the cluster security group to the instances. For more information, see [Changing an instance's security groups in the Amazon VPC User Guide](#).

You can check for a security group for your cluster in the AWS Management Console under the **Networking** section for the cluster. Or, you can do this using the following AWS CLI command. When using this command, replace `my-cluster` with the name of your cluster.

```
aws eks describe-cluster --name my-cluster --query
cluster.resourcesVpcConfig.clusterSecurityGroupId
```

Create a Fargate pod execution role

When your cluster creates pods on AWS Fargate, the components that run on the Fargate infrastructure must make calls to AWS APIs on your behalf. The Amazon EKS pod execution role provides the IAM permissions to do this. To create an AWS Fargate pod execution role, see [Amazon EKS pod execution IAM role \(p. 477\)](#).

Note

If you created your cluster with `eksctl` using the `--fargate` option, then your cluster already has a pod execution role. Similarly, if you use `eksctl` to create your Fargate profiles, `eksctl` creates your pod execution role if one isn't already created.

Create a Fargate profile for your cluster

Before you can schedule pods running on Fargate in your cluster, you must define a Fargate profile that specifies which pods should use Fargate when they are launched. For more information, see [AWS Fargate profile \(p. 158\)](#).

Note

If you created your cluster with `eksctl` using the `--fargate` option, then a Fargate profile has already been created for your cluster with selectors for all pods in the `kube-system` and `default` namespaces. Use the following procedure to create Fargate profiles for any other namespaces you would like to use with Fargate.

You can create a Fargate profile using `eksctl` or the AWS Management Console.

This procedure requires `eksctl` version 0.97.0 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 \(p. 11\)](#).

`eksctl`

To create a Fargate profile with `eksctl`

Create your Fargate profile with the following `eksctl` command, replacing every *example-value* with your own values. You're required to specify a namespace. However, the `--labels` option isn't required.

```
eksctl create fargateprofile \
    --cluster my-cluster \
    --name my-fargate-profile \
    --namespace my-kubernetes-namespace \
    --labels key=value
```

AWS Management Console

To create a Fargate profile for a cluster with the AWS Management Console

1. Open the Amazon EKS console at <https://console.aws.amazon.com/eks/home#/clusters>.
2. Choose the cluster to create a Fargate profile for.
3. Under **Fargate profiles**, choose **Add Fargate profile**.
4. On the **Configure Fargate Profile** page, do the following:

- a. For **Name**, enter a unique name for your Fargate profile.
 - b. For **Pod execution role**, choose the pod execution role to use with your Fargate profile. Only the IAM roles with the `eks-fargate-pods.amazonaws.com` service principal are shown. If you don't see any roles listed, you must create one. For more information, see [Amazon EKS pod execution IAM role \(p. 477\)](#).
 - c. Choose the **Subnets** dropdown and deselect any subnet with `Public` in its name. Only private subnets are supported for pods running on Fargate.
 - d. For **Tags**, you can optionally tag your Fargate profile. These tags don't propagate to other resources associated with the profile such as pods.
 - e. Choose **Next**.
5. On the **Configure pod selection** page, do the following:
 - a. For **Namespace**, enter a namespace to match for pods, such as `kube-system` or `default`.
 - b. (Optional) Add Kubernetes labels to the selector that pods in the specified namespace must have to match the selector. For example, you can add the label `infrastructure: fargate` to the selector. This is so that only the pods in the specified namespace that also have the `infrastructure: fargate` Kubernetes label match the selector.
 - c. Choose **Next**.
 6. On the **Review and create** page, review the information for your Fargate profile and choose **Create**.

Update CoreDNS

By default, CoreDNS is configured to run on Amazon EC2 infrastructure on Amazon EKS clusters. If you want to *only* run your pods on Fargate in your cluster, complete the following steps.

Note

If you created your cluster with `eksctl` using the `--fargate` option, then you can skip to [Next steps \(p. 158\)](#).

1. Create a Fargate profile for CoreDNS. Replace `my-cluster` with your cluster name, `111122223333` with your account ID, `AmazonEKSFargatePodExecutionRole` with the name of your pod execution role, and `0000000000000001`, `0000000000000002`, and `0000000000000003` with the IDs of your private subnets. If you don't have a pod execution role, you must [create one \(p. 156\)](#) first. If your cluster is in the AWS GovCloud (US-East) or AWS GovCloud (US-East) AWS Regions, then replace `arn:aws:` with `arn:aws-us-gov:` before running the following command.

Important

The role ARN can't include a path. The format of the role ARN must be `arn:aws:iam::111122223333:role/role-name`. For more information, see [aws-auth ConfigMap does not grant access to the cluster \(p. 554\)](#).

```
aws eks create-fargate-profile \
--fargate-profile-name coredns \
--cluster-name my-cluster \
--pod-execution-role-arn
arn:aws:iam::111122223333:role/AmazonEKSFargatePodExecutionRole \
--selectors namespace=kube-system,labels={k8s-app=kube-dns} \
--subnets subnet-0000000000000001 subnet-0000000000000002 subnet-0000000000000003
```

2. Run the following command to remove the `eks.amazonaws.com/compute-type : ec2` annotation from the CoreDNS pods.

```
kubectl patch deployment coredns \
-n kube-system \
```

```
--type json \
-p='[{"op": "remove", "path": "/spec/template/metadata/annotations/
eks.amazonaws.com-1compute-type"}]'
```

Next steps

- You can start migrating your existing applications to run on Fargate with the following workflow.
- 1. [Create a Fargate profile \(p. 159\)](#) that matches your application's Kubernetes namespace and Kubernetes labels.
- 2. Delete and re-create any existing pods so that they are scheduled on Fargate. For example, the following command triggers a rollout of the coredns deployment. You can modify the namespace and deployment type to update your specific pods.

```
kubectl rollout restart -n kube-system deployment coredns
```

- Deploy the [Application load balancing on Amazon EKS \(p. 378\)](#) to allow Ingress objects for your pods running on Fargate.
- You can use the [Vertical Pod Autoscaler \(p. 366\)](#) to initially right size the CPU and memory for your Fargate pods, and then use the [Horizontal Pod Autoscaler \(p. 370\)](#) to scale those pods. If you want the Vertical Pod Autoscaler to automatically re-deploy pods to Fargate with larger CPU and memory combinations, set the Vertical Pod Autoscaler's mode to either Auto or Recreate. This is to ensure correct functionality. For more information, see the [Vertical Pod Autoscaler](#) documentation on GitHub.
- You can set up the [AWS Distro for OpenTelemetry \(ADOT\)](#) collector for application monitoring by following [these instructions](#).

AWS Fargate profile

Before you can schedule pods on Fargate in your cluster, you must define at least one Fargate profile that specifies which pods use Fargate when launched.

The Fargate profile allows an administrator to declare which pods run on Fargate. This declaration is done through the profile's selectors. Each profile can have up to five selectors that contain a namespace and optional labels. You must define a namespace for every selector. The label field consists of multiple optional key-value pairs. Pods that match a selector (by matching a namespace for the selector and all of the labels specified in the selector) are scheduled on Fargate. If a namespace selector is defined without any labels, Amazon EKS attempts to schedule all pods that run in that namespace onto Fargate using the profile. If a to-be-scheduled pod matches any of the selectors in the Fargate profile, then that pod is scheduled on Fargate.

If a pod matches multiple Fargate profiles, Amazon EKS picks one of the matches at random. In this case, you can specify which profile a pod should use by adding the following Kubernetes label to the pod specification: `eks.amazonaws.com/fargate-profile: my-fargate-profile`. However, the pod must still match a selector in that profile in order to be scheduled onto Fargate. Kubernetes affinity/anti-affinity rules aren't taken into consideration and are unnecessary with Amazon EKS Fargate pods.

When you create a Fargate profile, you must specify a pod execution role for the Amazon EKS components that run on the Fargate infrastructure using the profile. This role is added to the cluster's Kubernetes [Role Based Access Control](#) (RBAC) for authorization so that the kubelet that's running on the Fargate infrastructure can register with your Amazon EKS cluster and appear in your cluster as a node. The pod execution role also provides IAM permissions to the Fargate infrastructure to allow read access to Amazon ECR image repositories. For more information, see [Amazon EKS pod execution IAM role \(p. 477\)](#).

Fargate profiles are immutable. However, you can create a new updated profile to replace an existing profile and then delete the original after the updated profile has finished creating.

Note

Any pods that are running using a Fargate profile will be stopped and put into pending when the profile is deleted.

If any Fargate profiles in a cluster are in the `DELETING` status, you must wait for that Fargate profile to finish deleting before you can create any other profiles in that cluster.

Amazon EKS and Fargate try to spread pods across each of the subnets defined in the Fargate profile, but you may end up with an uneven spread. If you must have an even spread (such as when deploying two replicas without any downtime), then you need to use two Fargate profiles. Each profile should have only one subnet.

Fargate profile components

The following components are contained in a Fargate profile.

- **Pod execution role** – When your cluster creates pods on AWS Fargate, the `kubelet` that's running on the Fargate infrastructure must make calls to AWS APIs on your behalf. This is, for example, to pull container images from Amazon ECR. The Amazon EKS pod execution role provides the IAM permissions to do this.

When you create a Fargate profile, you must specify a pod execution role to use with your pods. This role is added to the cluster's Kubernetes [Role Based Access Control](#) (RBAC) for authorization. This is so that the `kubelet` that's running on the Fargate infrastructure can register with your Amazon EKS cluster and appear in your cluster as a node. For more information, see [Amazon EKS pod execution IAM role \(p. 477\)](#).

- **Subnets** – The IDs of subnets to launch pods into that use this profile. At this time, pods that are running on Fargate aren't assigned public IP addresses. Therefore, only private subnets (with no direct route to an Internet Gateway) are accepted for this parameter.
- **Selectors** – The selectors to match for pods to use this Fargate profile. Each selector must have an associated namespace. Optionally, you can also specify labels for a namespace. You may specify up to five selectors in a Fargate profile. A pod only must match one selector to run using the Fargate profile.
- **Namespace** – You must specify a namespace for a selector. The selector only matches pods that are created in this namespace, but you can create multiple selectors to target multiple namespaces.
- **Labels** – You can optionally specify Kubernetes labels to match for the selector. The selector only matches pods that have all of the labels that are specified in the selector.

Creating a Fargate profile

This topic helps you to create a Fargate profile. AWS Fargate with Amazon EKS is available in all Amazon EKS Regions except China (Beijing), China (Ningxia), AWS GovCloud (US-East), and AWS GovCloud (US-West).. You also must have created a pod execution role to use for your Fargate profile. For more information, see [Amazon EKS pod execution IAM role \(p. 477\)](#). Pods that are running on Fargate are only supported on private subnets (with [NAT gateway](#) access to AWS services, but not a direct route to an Internet Gateway), so your cluster's VPC must have private subnets available. You can create a profile with `eksctl` or the AWS Management Console. Select the tab with the name of the tool that you want to create your Fargate profile with.

This procedure requires `eksctl` version 0.97.0 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 \(p. 11\)](#).

eksctl

To create a Fargate profile with eksctl

Create your Fargate profile with the following eksctl command, replacing every *example-value* with your own values. You're required to specify a namespace. However, the --labels option isn't required.

```
eksctl create fargateprofile \
    --cluster my-cluster \
    --name my-fargate-profile \
    --namespace my-kubernetes-namespace \
    --labels key=value
```

AWS Management Console

To create a Fargate profile for a cluster with the AWS Management Console

1. Open the Amazon EKS console at <https://console.aws.amazon.com/eks/home#/clusters>.
2. Choose the cluster to create a Fargate profile for.
3. Choose the **Configuration** tab.
4. Choose the **Compute** tab.
5. Under **Fargate Profiles**, choose **Add Fargate Profile**.
6. 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 pod execution role to use with your Fargate profile. Only the IAM roles with the `eks-fargate-pods.amazonaws.com` service principal are shown. If you don't see any roles listed, you must create one. For more information, see [Amazon EKS pod execution IAM role \(p. 477\)](#).
 - 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. For **Tags**, you can optionally tag your Fargate profile. These tags don't propagate to other resources associated with the profile, such as pods.
 - e. Choose **Next**.
7. On the **Configure pod selection** page, do the following:
 - a. For **Namespace**, enter a namespace to match for pods, such as `kube-system` or `default`.
 - b. (Optional) Add Kubernetes labels to the selector. Specifically add them to the one that the pods in the specified namespace need to match. For example, you could add the label `infrastructure: fargate` to the selector so that only pods in the specified namespace that also have the `infrastructure: fargate` Kubernetes label match the selector.
 - c. Choose **Next**.
8. On the **Review and create** page, review the information for your Fargate profile and choose **Create**.

Deleting a Fargate profile

This topic helps you to delete a Fargate profile.

When you delete a Fargate profile, any pods that were scheduled onto Fargate with the profile are deleted. If those pods match another Fargate profile, then they are scheduled on Fargate with that profile. If they no longer match any Fargate profiles, then they aren't scheduled onto Fargate and may remain as pending.

Only one Fargate profile in a cluster can be in the `DELETING` status at a time. Wait for a Fargate profile to finish deleting before you can delete any other profiles in that cluster.

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

`eksctl`

To delete a Fargate profile with `eksctl`

Use the following command to delete a profile from a cluster. Replace every *example-value* with your own values.

```
eksctl delete fargateprofile --name my-profile --cluster my-cluster
```

AWS Management Console

To delete a Fargate profile from a cluster with the AWS Management Console

1. Open the Amazon EKS console at <https://console.aws.amazon.com/eks/home#/clusters>.
2. In the left navigation pane, choose Amazon EKS **Clusters**. In the list of clusters, choose the cluster that you want to delete the Fargate profile from.
3. Choose the **Configuration** tab, and then choose the **Compute** tab.
4. Choose the Fargate profile to delete, and then choose **Delete**.
5. On the **Delete Fargate Profile** page, type the name of the profile and then choose **Delete**.

AWS CLI

To delete a Fargate profile with AWS CLI

Use the following command to delete a profile from a cluster. Replace every *example-value* with your own values.

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

Fargate pod configuration

This section describes some of the unique pod configuration details for running Kubernetes pods on AWS Fargate.

Pod CPU and memory

With Kubernetes, you can define requests, a minimum vCPU amount, and memory resources that are allocated to each container in a pod. Pods are scheduled by Kubernetes to ensure that at least the requested resources for each pod are available on the compute resource. For more information, see [Managing compute resources for containers](#) in the Kubernetes documentation.

Note

Since Amazon EKS Fargate runs only one pod per node, the scenario of evicting pods in case of fewer resources doesn't occur. All Amazon EKS Fargate pods run with guaranteed priority, so the requested CPU and memory must be equal to the limit for all of the containers. For more information, see [Configure Quality of Service for Pods](#) in the Kubernetes documentation.

When pods are scheduled on Fargate, the vCPU and memory reservations within the pod specification determine how much CPU and memory to provision for the pod.

- The maximum request out of any Init containers is used to determine the Init request vCPU and memory requirements.
- Requests for all long-running containers are added up to determine the long-running request vCPU and memory requirements.
- The larger of the above two values is chosen for the vCPU and memory request to use for your pod.
- Fargate adds 256 MB to each pod's memory reservation for the required Kubernetes components (`kubelet`, `kube-proxy`, and `containerd`).

Fargate rounds up to the compute configuration shown below that most closely matches the sum of vCPU and memory requests in order to ensure pods always have the resources that they need to run.

If you don't specify a vCPU and memory combination, then the smallest available combination is used (.25 vCPU and 0.5 GB memory).

The following table shows the vCPU and memory combinations that are available for pods running on Fargate.

vCPU value	Memory value
.25 vCPU	0.5 GB, 1 GB, 2 GB
.5 vCPU	1 GB, 2 GB, 3 GB, 4 GB
1 vCPU	2 GB, 3 GB, 4 GB, 5 GB, 6 GB, 7 GB, 8 GB
2 vCPU	Between 4 GB and 16 GB in 1-GB increments
4 vCPU	Between 8 GB and 30 GB in 1-GB increments

The additional memory reserved for the Kubernetes components can cause a Fargate task with more vCPUs than requested to be provisioned. For example, a request for 1 vCPU and 8 GB memory will have 256 MB added to its memory request, and will provision a Fargate task with 2 vCPUs and 9 GB memory, since no task with 1 vCPU and 9 GB memory is available.

There is no correlation between the size of the pod running on Fargate and the node size reported by Kubernetes with `kubectl get nodes`. The reported node size is often larger than the pod's capacity. You can verify pod capacity with the following command. Replace `pod-name` with the name of your pod.

```
kubectl describe pod pod-name
```

Example output:

```
...
annotations:
  CapacityProvisioned: 0.25vCPU 0.5GB
...
```

The `CapacityProvisioned` annotation represents the enforced pod capacity and it determines the cost of your pod running on Fargate. For pricing information for the compute configurations, see [AWS Fargate Pricing](#).

Fargate storage

When provisioned, each pod running on Fargate receives 20 GB of container image layer storage. Pod storage is ephemeral. After a pod stops, the storage is deleted. New pods launched onto Fargate on

or after May 28, 2020, have encryption of the ephemeral storage volume enabled by default. The ephemeral pod storage is encrypted with an AES-256 encryption algorithm using AWS Fargate managed keys.

Note

The usable storage for Amazon EKS pods that run on Fargate is less than 20 GB because some space is used by the `kubelet` and other Kubernetes modules that are loaded inside the pod.

Fargate pod patching

Amazon EKS must periodically patch AWS Fargate pods to keep them secure. Updates are attempted in a way that creates the least impact on your services. However, if pods aren't successfully evicted, there are times when they must be deleted. The following are actions that you can take to minimize potential disruptions:

- Set appropriate pod disruption budgets (PDBs) to control the number of pods that are down simultaneously.
- Create event rules to react to failed evictions before the pods are deleted.

Amazon EKS works closely with the Kubernetes community to make bug fixes and security patches available as quickly as possible. New patch versions of Kubernetes are made available as part of Fargate platform versions. All Fargate pods start on the most recent Kubernetes patch version, which is available from Amazon EKS for the Kubernetes version of your cluster. If you have a pod with an older patch version, Amazon EKS might restart it to update it to the latest version. This ensures that your pods are equipped with the latest security updates. That way, if there's a critical [Common Vulnerabilities and Exposures \(CVE\)](#) issue, you're kept up to date to reduce security risks.

To limit the number of pods that are down at one time when pods are patched, you can set pod disruption budgets (PDBs). You can use PDBs to define minimum availability based on the requirements of each of your applications while still allowing updates to occur. For more information, see [Specifying a Disruption Budget for your Application](#) in the *Kubernetes Documentation*.

Amazon EKS uses the [Eviction API](#) to safely drain the pod while respecting the PDBs that you set for the application. Pods are evicted by Availability Zone to minimize impact. If the eviction succeeds, the new pod gets the latest patch and no further action is required.

When the eviction for a pod fails, Amazon EKS sends an event to your account with details about the pods that failed eviction. You can act on the message before the scheduled termination time. The specific time varies based on the urgency of the patch. When it's time, Amazon EKS attempts to evict the pods again. However, this time a new event isn't sent if the eviction fails. If the eviction fails again, your existing pods are deleted periodically so that the new pods can have the latest patch.

The following is a sample event received when the pod eviction fails. It contains details about the cluster, pod name, pod namespace, Fargate profile, and the scheduled termination time.

```
{  
    "version": "0",  
    "id": "12345678-90ab-cdef-0123-4567890abcde",  
    "detail-type": "EKS Fargate Pod Scheduled Termination",  
    "source": "aws.eks",  
    "account": "111122223333",  
    "time": "2021-06-27T12:52:44Z",  
    "region": "region-code",  
    "resources": [  
        "default/my-database-deployment"  
    ],  
    "detail": {  
        "clusterName": "my-cluster",  
        "fargateProfileName": "my-fargate-profile",  
    }  
}
```

```

    "podName": "my-pod-name",
    "podNamespace": "default",
    "evictErrorMessage": "Cannot evict pod as it would violate the pod's disruption
budget",
    "scheduledTerminationTime": "2021-06-30T12:52:44.832Z[UTC]"
}
}

```

In addition, having multiple PDBs associated with a pod can cause an eviction failure event. This event returns the following error message.

```
"evictErrorMessage": "This pod has multiple PodDisruptionBudget, which the eviction
subresource does not support",
```

You can create a desired action based on this event. For example, you can adjust your pod disruption budget (PDB) to control how the pods are evicted. More specifically, suppose that you start with a PDB that specifies the target percentage of pods that are available. Before your pods are force terminated during an upgrade, you can adjust the PDB to a different percentage of pods. To receive this event, you must create an Amazon EventBridge rule in the AWS account and Region that the cluster belongs to. The rule must use the following **Custom pattern**. For more information, see [Creating Amazon EventBridge rules that react to events](#) in the *Amazon EventBridge User Guide*.

```
{
  "source": ["aws.eks"],
  "detail-type": ["EKS Fargate Pod Scheduled Termination"]
}
```

A suitable target can be set for the event to capture it. For a complete list of available targets, see [Amazon EventBridge targets](#) in the *Amazon EventBridge User Guide*.

Fargate metrics

You can collect system metrics and CloudWatch usage metrics for AWS Fargate.

Application metrics

For applications running on Amazon EKS and AWS Fargate, you can use the AWS Distro for OpenTelemetry (ADOT). ADOT allows you to collect system metrics and send them to CloudWatch Container Insights dashboards. To get started with ADOT for applications running on Fargate, see [Using CloudWatch Container Insights with AWS Distro for OpenTelemetry](#) in the ADOT documentation.

Usage metrics

You can use CloudWatch usage metrics to provide visibility into your account's usage of resources. Use these metrics to visualize your current service usage on CloudWatch graphs and dashboards.

AWS Fargate usage metrics correspond to AWS service quotas. You can configure alarms that alert you when your usage approaches a service quota. For more information about Fargate service quotas, see [Amazon EKS service quotas \(p. 435\)](#).

AWS Fargate publishes the following metrics in the AWS/Usage namespace.

Metric	Description
ResourceCount	The total number of the specified resource running on your account. The resource is defined by the dimensions associated with the metric.

The following dimensions are used to refine the usage metrics that are published by AWS Fargate.

Dimension	Description
Service	The name of the AWS service containing the resource. For AWS Fargate usage metrics, the value for this dimension is <code>Fargate</code> .
Type	The type of entity that's being reported. Currently, the only valid value for AWS Fargate usage metrics is <code>Resource</code> .
Resource	<p>The type of resource that's running.</p> <p>Currently, AWS Fargate returns information on your Fargate On-Demand usage. The resource value for Fargate On-Demand usage is <code>OnDemand</code>.</p> <p>Note Fargate On-Demand usage combines Amazon EKS pods using Fargate, Amazon ECS tasks using the Fargate launch type and Amazon ECS tasks using the <code>FARGATE</code> capacity provider.</p>
Class	The class of resource being tracked. Currently, AWS Fargate doesn't use the class dimension.

Creating a CloudWatch alarm to monitor Fargate resource usage metrics

AWS Fargate provides CloudWatch usage metrics that correspond to the AWS service quotas for Fargate On-Demand resource usage. In the Service Quotas console, you can visualize your usage on a graph. You can also configure alarms that alert you when your usage approaches a service quota. For more information, see [Fargate metrics \(p. 164\)](#).

Use the following steps to create a CloudWatch alarm based on the Fargate resource usage metrics.

To create an alarm based on your Fargate usage quotas (AWS Management Console)

1. Open the Service Quotas console at <https://console.aws.amazon.com/servicequotas/>.
2. In the left navigation pane, choose **AWS services**.
3. From the **AWS services** list, search for and select **AWS Fargate**.
4. In the **Service quotas** list, choose the Fargate usage quota you want to create an alarm for.
5. In the Amazon CloudWatch alarms section, choose **Create**.
6. For **Alarm threshold**, choose the percentage of your applied quota value that you want to set as the alarm value.
7. For **Alarm name**, enter a name for the alarm and then choose **Create**.

Fargate logging

Amazon EKS on Fargate offers a built-in log router based on Fluent Bit. This means that you don't explicitly run a Fluent Bit container as a sidecar, but Amazon runs it for you. All that you have to do is configure the log router. The configuration happens through a dedicated `ConfigMap` that must meet the following criteria:

- Named `aws-logging`
- Created in a dedicated namespace called `aws-observability`

Once you've created the ConfigMap, Amazon EKS on Fargate automatically detects it and configures the log router with it. Fargate uses a version of AWS for Fluent Bit, an upstream compliant distribution of Fluent Bit managed by AWS. For more information, see [AWS for Fluent Bit](#) on GitHub.

The log router allows you to use the breadth of services at AWS for log analytics and storage. You can stream logs from Fargate directly to Amazon CloudWatch, Amazon OpenSearch Service. You can also stream logs to destinations such as Amazon S3, Amazon Kinesis Data Streams, and partner tools through Amazon Kinesis Data Firehose.

Prerequisites

- An existing Fargate profile that specifies an existing Kubernetes namespace that you deploy Fargate pods to. For more information, see [Create a Fargate profile for your cluster \(p. 156\)](#).
- An existing Fargate pod execution role. For more information, see [Create a Fargate pod execution role \(p. 156\)](#).

Log router configuration

To configure the log router

In the following steps, replace every *example-value* with your own values.

1. Create a dedicated Kubernetes namespace named aws-observability.

- a. Save the following contents to a file named *aws-observability-namespace.yaml* on your computer. The value for name must be *aws-observability* and the *aws-observability: enabled* label is required.

```
kind: Namespace
apiVersion: v1
metadata:
  name: aws-observability
  labels:
    aws-observability: enabled
```

- b. Create the namespace.

```
kubectl apply -f aws-observability-namespace.yaml
```

2. Create a ConfigMap with a `Fluent Conf` data value to ship container logs to a destination. `Fluent Conf` is Fluent Bit, which is a fast and lightweight log processor configuration language that's used to route container logs to a log destination of your choice. For more information, see [Configuration File](#) in the Fluent Bit documentation.

Important

The main sections included in a typical `Fluent Conf` are `Service`, `Input`, `Filter`, and `Output`. The Fargate log router however, only accepts:

- The `Filter` and `Output` sections and manages the `Service` and `Input` sections itself.
- A `Parser` section.

If you provide any sections other than `Filter`, `Output`, and `Parser`, the sections are rejected.

When creating the ConfigMap, take into account the following rules that Fargate uses to validate fields:

- [FILTER], [OUTPUT], and [PARSER] are supposed to be specified under each corresponding key. For example, [FILTER] must be under `filters.conf`. You can have one or more [FILTER]s under `filters.conf`. The [OUTPUT] and [PARSER] sections should also be under their corresponding keys. By specifying multiple [OUTPUT] sections, you can route your logs to different destinations at the same time.
- Fargate validates the required keys for each section. Name and match are required for each [FILTER] and [OUTPUT]. Name and format are required for each [PARSER]. The keys are case-insensitive.
- Environment variables such as `#{ENV_VAR}` aren't allowed in the ConfigMap.
- The indentation has to be the same for either directive or key-value pair within each `filters.conf`, `output.conf`, and `parsers.conf`. Key-value pairs have to be indented more than directives.
- Fargate validates against the following supported filters: `grep`, `parser`, `record_modifier`, `rewrite_tag`, `throttle`, `nest`, `modify`, and `kubernetes`.
- Fargate validates against the following supported output: `es`, `firehose`, `kinesis_firehose`, `cloudwatch`, `cloudwatch_logs`, `kinesis`, and `kubernetes`.
- At least one supported Output plugin has to be provided in the ConfigMap to enable logging. Filter and Parser aren't required to enable logging.

You can also run Fluent Bit on Amazon EC2 using the desired configuration to troubleshoot any issues that arise from validation. Create your ConfigMap using one of the following examples.

Important

Amazon EKS Fargate logging doesn't support dynamic configuration of ConfigMaps. Any changes to ConfigMaps are applied to new pods only. Changes aren't applied to existing pods.

Create a ConfigMap using the example for your desired log destination.

CloudWatch

To create a ConfigMap for CloudWatch

You have two output options when using CloudWatch:

- [An output plugin written in C](#)
- [An output plugin written in Golang](#)

The following example shows you how to use the `cloudwatch_logs` plugin to send logs to CloudWatch.

1. Save the following contents to a file named `aws-logging-cloudwatch-configmap.yaml`. Replace `region-code` with the AWS Region that your cluster is in. The parameters under [OUTPUT] are required.

```
kind: ConfigMap
apiVersion: v1
metadata:
  name: aws-logging
  namespace: aws-observability
data:
  output.conf: |
    [OUTPUT]
      Name cloudwatch_logs
      Match *
      region region-code
```

```
log_group_name fluent-bit-cloudwatch
log_stream_prefix from-fluent-bit-
auto_create_group true
log_key log

parsers.conf: |
  [PARSER]
    Name crio
    Format Regex
    Regex ^(?<time>[^ ]+) (?<stream>stdout|stderr) (?<logtag>P|F) (?<log>.*$)
    Time_Key time
    Time_Format %Y-%m-%dT%H:%M:%S.%L%z

filters.conf: |
  [FILTER]
    Name parser
    Match *
    Key_name log
    Parser crio
```

2. Apply the manifest to your cluster.

```
kubectl apply -f aws-logging-cloudwatch-configmap.yaml
```

3. Download the CloudWatch IAM policy to your computer. You can also [view the policy](#) on GitHub.

```
curl -o permissions.json https://raw.githubusercontent.com/aws-samples/
amazon-eks-fluent-logging-examples/mainline/examples/fargate/cloudwatchlogs/
permissions.json
```

Amazon OpenSearch Service

To create a ConfigMap for Amazon OpenSearch Service

If you want to send logs to Amazon OpenSearch Service, you can use `es` output, which is a plugin written in C. The following example shows you how to use the plugin to send logs to OpenSearch.

1. Save the following contents to a file named `aws-logging-opensearch-configmap.yaml`. Replace every `example-value` with your own values.

```
kind: ConfigMap
apiVersion: v1
metadata:
  name: aws-logging
  namespace: aws-observability
data:
  output.conf: |
    [OUTPUT]
      Name es
      Match *
      Host search-example-gjxdclagiprbglqn42jsty66y.region-
code.amazonaws.com
      Port 443
      Index example
      Type example_type
      AWS_Auth On
      AWS_Region region-code
      tls On
```

2. Apply the manifest to your cluster.

```
kubectl apply -f aws-logging-opensearch-configmap.yaml
```

3. Download the OpenSearch IAM policy to your computer. You can also [view the policy](#) on GitHub.

```
curl -o permissions.json https://raw.githubusercontent.com/aws-samples/amazon-eks-fluent-logging-examples/mainline/examples/fargate/amazon-elasticsearch/permissions.json
```

Make sure that OpenSearch Dashboards' access control is configured properly. The `all_access` role in OpenSearch Dashboards needs to have the Fargate pod execution role and the IAM role mapped. The same mapping must be done for the `security_manager` role. You can add the previous mappings by selecting `Menu`, then `Security`, then `Roles`, and then select the respective roles. For more information, see [How do I troubleshoot CloudWatch Logs so that it streams to my Amazon ES domain?](#).

Kinesis Data Firehose

To create a ConfigMap for Kinesis Data Firehose

You have two output options when sending logs to Kinesis Data Firehose:

- `kinesis_firehose` – An output plugin written in C.
- `firehose` – An output plugin written in Golang.

The following example shows you how to use the `kinesis_firehose` plugin to send logs to Kinesis Data Firehose.

1. Save the following contents to a file named `aws-logging-firehose-configmap.yaml`. Replace `region-code` with the AWS Region that your cluster is in.

```
kind: ConfigMap
apiVersion: v1
metadata:
  name: aws-logging
  namespace: aws-observability
data:
  output.conf: |
    [OUTPUT]
    Name kinesis_firehose
    Match *
    region region-code
    delivery_stream my-stream-firehose
```

2. Apply the manifest to your cluster.

```
kubectl apply -f aws-logging-firehose-configmap.yaml
```

3. Download the Kinesis Data Firehose IAM policy to your computer. You can also [view the policy](#) on GitHub.

```
curl -o permissions.json https://raw.githubusercontent.com/aws-samples/amazon-eks-fluent-logging-examples/mainline/examples/fargate/kinesis-firehose/permissions.json
```

3. Create an IAM policy from the policy file you downloaded in a previous step.

```
aws iam create-policy --policy-name eks-fargate-logging-policy --policy-document file://permissions.json
```

4. Attach the IAM policy to the pod execution role specified for your Fargate profile. Replace **111122223333** with your account ID. If your cluster is in the AWS GovCloud (US-East) or AWS GovCloud (US-East) AWS Regions, then replace `arn:aws:` with `arn:aws-us-gov:` before running the following command.

```
aws iam attach-role-policy \
--policy-arn arn:aws:iam::111122223333:policy/eks-fargate-logging-policy \
--role-name AmazonEKSFargatePodExecutionRole
```

Kubernetes filter support

This feature requires the following minimum Kubernetes version and platform level, or later. Any Kubernetes and platform versions later than those listed in the table are supported.

Kubernetes version	Platform level
1.21	eks.3
1.20	eks.3
1.19	eks.7
1.18	eks.9
1.17	eks.10

The Fluent Bit Kubernetes filter allows you to add Kubernetes metadata to your log files. For more information about the filter, see [Kubernetes](#) in the Fluent Bit documentation. You can apply a filter using the API server endpoint.

```
filters.conf: |
[FILTER]
  Name      kubernetes
  Match    kube./*
  Merge_Log On
  Buffer_Size 0
  Kube_Meta_Cache_TTL 300s
```

Important

- `Kube_URL`, `Kube_CA_File`, `Kube_Token_Command`, and `Kube_Token_File` are service owned configuration parameters and must not be specified. Amazon EKS Fargate populates these values.
- `Kube_Meta_Cache_TTL` is the time Fluent Bit waits until it communicates with the API server for the latest metadata. If `Kube_Meta_Cache_TTL` isn't specified then Amazon EKS Fargate appends a default value of 30 minutes to lessen the load on the API server, since Fluent Bit communicates with the API server to get the latest metadata.

To ship Fluent Bit process logs to your account

You can ship Fluent Bit process logs to Amazon CloudWatch using the following ConfigMap. Replace `region-code` with the AWS Region that your cluster is in.

```
kind: ConfigMap
apiVersion: v1
metadata:
  name: aws-logging
  namespace: aws-observability
  labels:
data:
  # Configuration files: server, input, filters and output
  # =====
  flb_log_cw: "true" #ships fluent-bit process logs to CloudWatch

  output.conf: |
    [OUTPUT]
      Name cloudwatch
      Match kube.*
      region region-code
      log_group_name fluent-bit-cloudwatch
      log_stream_prefix from-fluent-bit-
      auto_create_group true
```

The logs are in the AWS Region that the cluster resides in under CloudWatch. The log group name is `my-cluster-fluent-bit-logs` and the Fluent Bit logstream name is `fluent-bit-podname-pod-namespace`.

Note

- The process logs are shipped only when the Fluent Bit process successfully starts. If there is a failure while starting Fluent Bit, the process logs are missed. You can only ship process logs to CloudWatch.
- To debug shipping process logs to your account, you can apply the previous ConfigMap to get the process logs. Fluent Bit failing to start is usually due to your ConfigMap not being parsed or accepted by Fluent Bit while starting.

Test application

1. Deploy a sample pod.
 - a. Save the following contents to a file named `sample-app.yaml` on your computer.

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: sample-app
  namespace: same-namespace-as-your-fargate-profile
spec:
  replicas: 3
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
        - name: nginx
          image: nginx:latest
      ports:
```

```
- name: http
  containerPort: 80
```

- b. Apply the manifest to the cluster.

```
kubectl apply -f sample-app.yaml
```

2. View the NGINX logs using the destination(s) that you configured in the ConfigMap.

Size considerations

We suggest that you plan for up to 50 MB of memory for the log router. If you expect your application to generate logs at very high throughput then you should plan for up to 100 MB.

Troubleshooting

To confirm whether the logging feature is enabled or disabled for some reason, such as an invalid ConfigMap, and why it's invalid, check your pod events with `kubectl describe pod pod-name`. The output might include pod events that clarify whether logging is enabled or not, such as the following example output.

```
...
Annotations:           CapacityProvisioned: 0.25vCPU 0.5GB
                      Logging: LoggingDisabled: LOGGING_CONFIGMAP_NOT_FOUND
                      kubernetes.io/psp: eks.privileged
...
Events:
Type    Reason        Age            From
Message
----   -----
-----  

Warning  LoggingDisabled <unknown>  fargate-scheduler
        Disabled logging because aws-logging configmap was not found. configmap "aws-logging" not found
```

The pod events are ephemeral with a time period depending on the settings. You can also view a pod's annotations using `kubectl describe pod pod-name`. In the pod annotation, there is information about whether the logging feature is enabled or disabled and the reason.

Choosing an Amazon EC2 instance type

Amazon EC2 provides a wide selection of instance types for worker nodes. Each instance type offers different compute, memory, storage, and network capabilities. Each instance is also grouped in an instance family based on these capabilities. For a list, see [Available instance types](#) in the *Amazon EC2 User Guide for Linux Instances* and [Available instance types](#) in the *Amazon EC2 User Guide for Windows Instances*. To make sure that the instance type you select is compatible with Amazon EKS, consider the following criteria. Amazon EKS releases several variations of Amazon EC2 AMIs to enable support.

- The `mac` family isn't currently supported by any Amazon EKS AMI.
- The `g3`, `g4`, `inf`, and `p` families are supported by x86-accelerated Amazon EKS optimized AMIs. Drivers for the `g5g` instances which are Arm based accelerated instances aren't currently supported.
- The `a`, `c`, `hpc`, `m`, and `t` families are only supported by the [x86 and Arm AMIs \(p. 175\)](#).

When choosing between instance types that are supported by Amazon EKS, consider the following capabilities of each type.

- **Number of instances in a node group** – In general, fewer, larger instances are better, especially if you have a lot of Daemonsets. Each instance requires API calls to the API server, so the more instances you have, the more load on the API server.
- **Operating system** – Review the supported instance types for [Linux](#), [Windows](#), and [Bottlerocket](#). Before creating Windows instances, review [the section called "Windows support" \(p. 92\)](#).
- **Hardware architecture** – Do you need x86 or Arm? You can only deploy Linux on Arm. Before deploying Arm instances, review [the section called "Arm" \(p. 186\)](#). Do you need instances built on the Nitro System ([Linux](#) or [Windows](#)) or that have [Accelerated](#) capabilities? If you need accelerated capabilities, you can only use Linux with Amazon EKS.
- **Maximum number of pods** – Since each pod is assigned its own IP address, the number of IP addresses supported by an instance type is a factor in determining the number of pods that can run on the instance. To manually determine how many pods an instance type supports, see [Amazon EKS recommended maximum pods for each Amazon EC2 instance type \(p. 173\)](#).

Note

If you're using an Amazon EKS optimized Amazon Linux 2 AMI that's v20220406 or newer, you can use a new instance type without upgrading to the latest AMI. For these AMIs, the AMI auto-calculates the necessary `max-pods` value if it isn't listed in the `eni-max-pods.txt` file. Instance types that are currently in preview may not be supported by Amazon EKS by default. Values for `max-pods` for such types still need to be added to `eni-max-pods.txt` in our AMI.

[AWS Nitro System](#) instance types optionally support significantly more IP addresses than non-Nitro System instance types. However, not all IP addresses assigned for an instance are available to pods. To assign a significantly larger number of IP addresses to your instances, you must have version 1.9.0 or later of the Amazon VPC CNI add-on installed in your cluster and configured appropriately. For more information, see [Increase the amount of available IP addresses for your Amazon EC2 nodes \(p. 310\)](#). To assign the largest number of IP addresses to your instances, you must have version 1.10.1 or later of the Amazon VPC CNI add-on installed in your 1.21 or later cluster and deploy the cluster with the IPv6 family.

- **IP family** – You can use any supported instance type when using the IPv4 family for a cluster, which allows your cluster to assign private IPv4 addresses to your pods and Services. But if you want to use the IPv6 family for your cluster, then you must use [AWS Nitro System](#) instance types or bare metal instance types. Only IPv4 is supported for Windows instances. Your cluster must be a new 1.21 or later cluster running version 1.10.1 or later of the Amazon VPC CNI add-on. For more information about using IPv6, see [Assigning IPv6 addresses to pods and services \(p. 294\)](#).
- **Version of the Amazon VPC CNI add-on that you're running** – The latest version of the [Amazon VPC CNI plugin for Kubernetes](#) supports [these instance types](#). You may need to update your Amazon VPC CNI add-on version to take advantage of the latest supported instance types. For more information, see [the section called "Updating the self-managed add-on" \(p. 287\)](#). The latest version supports the latest features for use with Amazon EKS. Earlier versions don't support all features. You can view features supported by different versions in the [Changelog](#) on GitHub.
- **AWS Region that you're creating your nodes in** – Not all instance types are available in all AWS Regions.
- **Whether you're using security groups for pods** – If you're using security groups for pods, only specific instance types are supported. For more information, see [Security groups for pods \(p. 313\)](#).

Amazon EKS recommended maximum pods for each Amazon EC2 instance type

Since each pod is assigned its own IP address, the number of IP addresses supported by an instance type is a factor in determining the number of pods that can run on the instance. Amazon EKS provides a script that you can download and run to determine the Amazon EKS recommended maximum number of pods to run on each instance type. The script uses hardware attributes of each instance, and

configuration options, to determine the maximum pods number. You can use the number returned in these steps to enable capabilities such as [assigning IP addresses to pods from a different subnet than the instance's \(p. 306\)](#) and [significantly increasing the number of IP addresses for your instance \(p. 310\)](#). If you're using a managed node group with multiple instance types, use a value that would work for all instance types.

1. Download a script that you can use to calculate the maximum number of pods for each instance type.

```
curl -o max-pods-calculator.sh https://raw.githubusercontent.com/awslabs/amazon-eks-ami/master/files/max-pods-calculator.sh
```

2. Mark the script as executable on your computer.

```
chmod +x max-pods-calculator.sh
```

3. Run the script, replacing `m5.large` with the instance type that you plan to deploy and `1.9.0-eksbuild.1` with your Amazon VPC CNI add-on version. To determine your add-on version, see the update procedures in [Managing the Amazon VPC CNI add-on \(p. 281\)](#).

```
./max-pods-calculator.sh --instance-type m5.large --cni-version 1.9.0-eksbuild.1
```

Example output:

29

You can add the following options to the script to see the maximum pods supported when using optional capabilities.

- `--cni-custom-networking-enabled` – Use this option when you want to assign IP addresses from a different subnet than your instance's. For more information, see [the section called "CNI custom networking" \(p. 306\)](#). Adding this option to the previous script with the same example values yields 20.
- `--cni-prefix-delegation-enabled` – Use this option when you want to assign significantly more IP addresses to each elastic network interface. This capability requires an Amazon Linux instance that run on the Nitro System and version 1.9.0 or later of the Amazon VPC CNI add-on. For more information, see [the section called "Increase available IP addresses" \(p. 310\)](#). Adding this option to the previous script with the same example values yields 110.

You can also run the script with the `--help` option to see all available options.

Amazon EKS optimized AMIs

You can deploy nodes with pre-built Amazon EKS optimized [Amazon Machine Images \(AMIs\)](#), or your own custom AMIs. For more information about each type of Amazon EKS optimized AMI, see one of the following topics. For instructions on how to create your own custom AMI, see [Amazon EKS optimized Amazon Linux AMI build script \(p. 204\)](#).

Topics

- [Amazon EKS is ending support for DockerShim \(p. 175\)](#)
- [Amazon EKS optimized Amazon Linux AMIs \(p. 175\)](#)
- [Amazon EKS optimized Ubuntu Linux AMIs \(p. 205\)](#)
- [Amazon EKS optimized Bottlerocket AMIs \(p. 205\)](#)

- [Amazon EKS optimized Windows AMIs \(p. 211\)](#)

Amazon EKS is ending support for Dockershim

Kubernetes no longer supports Dockershim. Additionally, the Kubernetes team plans to completely remove the runtime in Kubernetes version 1.24. Amazon EKS is ending support for Dockershim starting with the Kubernetes version 1.23 launch. Amazon EKS AMIs that are officially published will have containerd as the only runtime starting with version 1.23. This is targeted for end of the second quarter of 2022.

There's a `kubectl` plugin you can use to see which of your Kubernetes workloads are mounting the Docker socket volume. For more information, see [Detector for Docker Socket \(DDS\) on GitHub](#). Amazon EKS AMIs that run Kubernetes version 1.17 through 1.21 use Docker as the default runtime, but have a bootstrap flag option that you can use to test out your workloads on any supported cluster today with containerd. For more information, see [Enable the containerd runtime bootstrap flag \(p. 184\)](#).

We will continue to publish AMIs for existing Kubernetes versions until the end of their support date. For more information, see [Amazon EKS Kubernetes release calendar \(p. 74\)](#). If you require more time to test your workloads on containerd, you can stay on a supported version before 1.23 and still be fully supported. But, when version 1.23 is made available and you want to upgrade official Amazon EKS AMIs to that version or higher, make sure to validate that your workloads run on containerd.

If you will be upgrading to version 1.23 or higher but you still want to use Dockershim, build your own AMI with Docker installed. However, when Kubernetes removes Dockershim support, make sure that you migrate to containerd to remain on a supported version. For more information, see [Kubernetes is Moving on From Dockershim: Commitments and Next Steps](#) on the *Kubernetes Blog*.

The containerd runtime provides more reliable performance and security, and it's the runtime we're standardizing on across Amazon EKS (Fargate and Bottlerocket already use containerd only). containerd helps to minimize the number of Amazon EKS AMI releases that are required to address Dockershim [Common Vulnerabilities and Exposures \(CVEs\)](#). As Dockershim already uses containerd internally, you might not need to make any changes. However, there are some situations where changes might or must be required:

- You must make changes for any application that mounts the Docker socket. For example, building container images using a container are impacted. Many monitoring tools also mount the Docker socket. You may have to wait for updates or re-deploy workloads for runtime monitoring.
- You might need to make changes for any application that's reliant on specific Docker settings. For example, using `HTTPS_PROXY` can require changes. For more information, see `dockerd` in the *Docker Docs*.
- If you're using the Amazon ECR credential helper to pull images, you must switch to the `kubelet` image credential provider. For more information, see [Configure a kubelet image credential provider](#) in the Kubernetes documentation.

Amazon EKS optimized Amazon Linux AMIs

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

Note

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

- Before deploying an accelerated or Arm AMI, review the information in [Amazon EKS optimized accelerated Amazon Linux AMIs \(p. 185\)](#) and [Amazon EKS optimized Arm Amazon Linux AMIs \(p. 186\)](#).
- Amazon EKS optimized Amazon Linux 2 contains an optional bootstrap flag to enable the `containerd` runtime. This feature provides a clear path to migrate to `containerd`. Amazon EKS is ending support for Docker starting with the Kubernetes version 1.23 launch. The `containerd` runtime is widely adopted in the Kubernetes community and is a graduated project with the CNCF. You can test it by adding a node group to a new or existing cluster. For more information, see [Enable the containerd runtime bootstrap flag \(p. 184\)](#). When bootstrapped in Amazon EKS optimized accelerated Amazon Linux AMIs for v1.21, [AWS Inferentia](#) workloads aren't supported.

In the following tables, choose **View AMI ID** for the Kubernetes version, AWS Region, and processor type that are specific to your Amazon Linux instance. You can also retrieve the IDs with an AWS Systems Manager parameter. For more information, see [Retrieving Amazon EKS optimized Amazon Linux AMI IDs \(p. 203\)](#).

1.22

Kubernetes version 1.22

AWS Region	x86	x86 accelerated	Arm
US East (Ohio) (<code>us-east-2</code>)	View AMI ID	View AMI ID	View AMI ID
US East (N. Virginia) (<code>us-east-1</code>)	View AMI ID	View AMI ID	View AMI ID
US West (Oregon) (<code>us-west-2</code>)	View AMI ID	View AMI ID	View AMI ID
US West (N. California) (<code>us-west-1</code>)	View AMI ID	View AMI ID	View AMI ID
Africa (Cape Town) (<code>af-south-1</code>)	View AMI ID	View AMI ID	View AMI ID
Asia Pacific (Hong Kong) (<code>ap-east-1</code>)	View AMI ID	View AMI ID	View AMI ID
Asia Pacific (Mumbai) (<code>ap-south-1</code>)	View AMI ID	View AMI ID	View AMI ID
Asia Pacific (Tokyo) (<code>ap-northeast-1</code>)	View AMI ID	View AMI ID	View AMI ID
Asia Pacific (Seoul) (<code>ap-northeast-2</code>)	View AMI ID	View AMI ID	View AMI ID
Asia Pacific (Osaka) (<code>ap-northeast-3</code>)	View AMI ID	View AMI ID	View AMI ID
Asia Pacific (Singapore) (<code>ap-southeast-1</code>)	View AMI ID	View AMI ID	View AMI ID
Asia Pacific (Sydney) (<code>ap-southeast-2</code>)	View AMI ID	View AMI ID	View AMI ID

AWS Region	x86	x86 accelerated	Arm
Asia Pacific (Jakarta) (ap-southeast-3)	View AMI ID	View AMI ID	View AMI ID
Canada (Central) (ca-central-1)	View AMI ID	View AMI ID	View AMI ID
China (Beijing) (cn-north-1)	View AMI ID	View AMI ID	View AMI ID
China (Ningxia) (cn-northwest-1)	View AMI ID	View AMI ID	View AMI ID
Europe (Frankfurt) (eu-central-1)	View AMI ID	View AMI ID	View AMI ID
Europe (Ireland) (eu-west-1)	View AMI ID	View AMI ID	View AMI ID
Europe (London) (eu-west-2)	View AMI ID	View AMI ID	View AMI ID
Europe (Milan) (eu-south-1)	View AMI ID	View AMI ID	View AMI ID
Europe (Paris) (eu-west-3)	View AMI ID	View AMI ID	View AMI ID
Europe (Stockholm) (eu-north-1)	View AMI ID	View AMI ID	View AMI ID
Middle East (Bahrain) (me-south-1)	View AMI ID	View AMI ID	View AMI ID
South America (São Paulo) (sa-east-1)	View AMI ID	View AMI ID	View AMI ID
AWS GovCloud (US-East) (us-gov-east-1)	View AMI ID	View AMI ID	View AMI ID
AWS GovCloud (US-West) (us-gov-west-1)	View AMI ID	View AMI ID	View AMI ID

1.21

Kubernetes version 1.21

AWS Region	x86	x86 accelerated	Arm
US East (Ohio) (us-east-2)	View AMI ID	View AMI ID	View AMI ID
US East (N. Virginia) (us-east-1)	View AMI ID	View AMI ID	View AMI ID

AWS Region	x86	x86 accelerated	Arm
US West (Oregon) (us-west-2)	View AMI ID	View AMI ID	View AMI ID
US West (N. California) (us-west-1)	View AMI ID	View AMI ID	View AMI ID
Africa (Cape Town) (af-south-1)	View AMI ID	View AMI ID	View AMI ID
Asia Pacific (Hong Kong) (ap-east-1)	View AMI ID	View AMI ID	View AMI ID
Asia Pacific (Mumbai) (ap-south-1)	View AMI ID	View AMI ID	View AMI ID
Asia Pacific (Tokyo) (ap-northeast-1)	View AMI ID	View AMI ID	View AMI ID
Asia Pacific (Seoul) (ap-northeast-2)	View AMI ID	View AMI ID	View AMI ID
Asia Pacific (Osaka) (ap-northeast-3)	View AMI ID	View AMI ID	View AMI ID
Asia Pacific (Singapore) (ap-southeast-1)	View AMI ID	View AMI ID	View AMI ID
Asia Pacific (Sydney) (ap-southeast-2)	View AMI ID	View AMI ID	View AMI ID
Asia Pacific (Jakarta) (ap-southeast-3)	View AMI ID	View AMI ID	View AMI ID
Canada (Central) (ca-central-1)	View AMI ID	View AMI ID	View AMI ID
China (Beijing) (cn-north-1)	View AMI ID	View AMI ID	View AMI ID
China (Ningxia) (cn-northwest-1)	View AMI ID	View AMI ID	View AMI ID
Europe (Frankfurt) (eu-central-1)	View AMI ID	View AMI ID	View AMI ID
Europe (Ireland) (eu-west-1)	View AMI ID	View AMI ID	View AMI ID
Europe (London) (eu-west-2)	View AMI ID	View AMI ID	View AMI ID
Europe (Milan) (eu-south-1)	View AMI ID	View AMI ID	View AMI ID
Europe (Paris) (eu-west-3)	View AMI ID	View AMI ID	View AMI ID

AWS Region	x86	x86 accelerated	Arm
Europe (Stockholm) (eu-north-1)	View AMI ID	View AMI ID	View AMI ID
Middle East (Bahrain) (me-south-1)	View AMI ID	View AMI ID	View AMI ID
South America (São Paulo) (sa-east-1)	View AMI ID	View AMI ID	View AMI ID
AWS GovCloud (US-East) (us-gov-east-1)	View AMI ID	View AMI ID	View AMI ID
AWS GovCloud (US-West) (us-gov-west-1)	View AMI ID	View AMI ID	View AMI ID

1.20

Kubernetes version 1.20

AWS Region	x86	x86 accelerated	Arm
US East (Ohio) (us-east-2)	View AMI ID	View AMI ID	View AMI ID
US East (N. Virginia) (us-east-1)	View AMI ID	View AMI ID	View AMI ID
US West (Oregon) (us-west-2)	View AMI ID	View AMI ID	View AMI ID
US West (N. California) (us-west-1)	View AMI ID	View AMI ID	View AMI ID
Africa (Cape Town) (af-south-1)	View AMI ID	View AMI ID	View AMI ID
Asia Pacific (Hong Kong) (ap-east-1)	View AMI ID	View AMI ID	View AMI ID
Asia Pacific (Mumbai) (ap-south-1)	View AMI ID	View AMI ID	View AMI ID
Asia Pacific (Tokyo) (ap-northeast-1)	View AMI ID	View AMI ID	View AMI ID
Asia Pacific (Seoul) (ap-northeast-2)	View AMI ID	View AMI ID	View AMI ID
Asia Pacific (Osaka) (ap-northeast-3)	View AMI ID	View AMI ID	View AMI ID
Asia Pacific (Singapore) (ap-southeast-1)	View AMI ID	View AMI ID	View AMI ID

AWS Region	x86	x86 accelerated	Arm
Asia Pacific (Sydney) (ap-southeast-2)	View AMI ID	View AMI ID	View AMI ID
Asia Pacific (Jakarta) (ap-southeast-3)	View AMI ID	View AMI ID	View AMI ID
Canada (Central) (ca-central-1)	View AMI ID	View AMI ID	View AMI ID
China (Beijing) (cn-north-1)	View AMI ID	View AMI ID	View AMI ID
China (Ningxia) (cn-northwest-1)	View AMI ID	View AMI ID	View AMI ID
Europe (Frankfurt) (eu-central-1)	View AMI ID	View AMI ID	View AMI ID
Europe (Ireland) (eu-west-1)	View AMI ID	View AMI ID	View AMI ID
Europe (London) (eu-west-2)	View AMI ID	View AMI ID	View AMI ID
Europe (Milan) (eu-south-1)	View AMI ID	View AMI ID	View AMI ID
Europe (Paris) (eu-west-3)	View AMI ID	View AMI ID	View AMI ID
Europe (Stockholm) (eu-north-1)	View AMI ID	View AMI ID	View AMI ID
Middle East (Bahrain) (me-south-1)	View AMI ID	View AMI ID	View AMI ID
South America (São Paulo) (sa-east-1)	View AMI ID	View AMI ID	View AMI ID
AWS GovCloud (US-East) (us-gov-east-1)	View AMI ID	View AMI ID	View AMI ID
AWS GovCloud (US-West) (us-gov-west-1)	View AMI ID	View AMI ID	View AMI ID

1.19

Kubernetes version 1.19

AWS Region	x86	x86 accelerated	Arm
US East (Ohio) (us-east-2)	View AMI ID	View AMI ID	View AMI ID

AWS Region	x86	x86 accelerated	Arm
US East (N. Virginia) (us-east-1)	View AMI ID	View AMI ID	View AMI ID
US West (Oregon) (us-west-2)	View AMI ID	View AMI ID	View AMI ID
US West (N. California) (us-west-1)	View AMI ID	View AMI ID	View AMI ID
Africa (Cape Town) (af-south-1)	View AMI ID	View AMI ID	View AMI ID
Asia Pacific (Hong Kong) (ap-east-1)	View AMI ID	View AMI ID	View AMI ID
Asia Pacific (Mumbai) (ap-south-1)	View AMI ID	View AMI ID	View AMI ID
Asia Pacific (Tokyo) (ap-northeast-1)	View AMI ID	View AMI ID	View AMI ID
Asia Pacific (Seoul) (ap-northeast-2)	View AMI ID	View AMI ID	View AMI ID
Asia Pacific (Osaka) (ap-northeast-3)	View AMI ID	View AMI ID	View AMI ID
Asia Pacific (Singapore) (ap-southeast-1)	View AMI ID	View AMI ID	View AMI ID
Asia Pacific (Sydney) (ap-southeast-2)	View AMI ID	View AMI ID	View AMI ID
Asia Pacific (Jakarta) (ap-southeast-3)	View AMI ID	View AMI ID	View AMI ID
Canada (Central) (ca-central-1)	View AMI ID	View AMI ID	View AMI ID
China (Beijing) (cn-north-1)	View AMI ID	View AMI ID	View AMI ID
China (Ningxia) (cn-northwest-1)	View AMI ID	View AMI ID	View AMI ID
Europe (Frankfurt) (eu-central-1)	View AMI ID	View AMI ID	View AMI ID
Europe (Ireland) (eu-west-1)	View AMI ID	View AMI ID	View AMI ID
Europe (London) (eu-west-2)	View AMI ID	View AMI ID	View AMI ID
Europe (Milan) (eu-south-1)	View AMI ID	View AMI ID	View AMI ID

AWS Region	x86	x86 accelerated	Arm
Europe (Paris) (eu-west-3)	View AMI ID	View AMI ID	View AMI ID
Europe (Stockholm) (eu-north-1)	View AMI ID	View AMI ID	View AMI ID
Middle East (Bahrain) (me-south-1)	View AMI ID	View AMI ID	View AMI ID
South America (São Paulo) (sa-east-1)	View AMI ID	View AMI ID	View AMI ID
AWS GovCloud (US-East) (us-gov-east-1)	View AMI ID	View AMI ID	View AMI ID
AWS GovCloud (US-West) (us-gov-west-1)	View AMI ID	View AMI ID	View AMI ID

1.18

Kubernetes version 1.18

AWS Region	x86	x86 accelerated	Arm
US East (Ohio) (us-east-2)	View AMI ID	View AMI ID	View AMI ID
US East (N. Virginia) (us-east-1)	View AMI ID	View AMI ID	View AMI ID
US West (Oregon) (us-west-2)	View AMI ID	View AMI ID	View AMI ID
US West (N. California) (us-west-1)	View AMI ID	View AMI ID	View AMI ID
Africa (Cape Town) (af-south-1)	View AMI ID	View AMI ID	View AMI ID
Asia Pacific (Hong Kong) (ap-east-1)	View AMI ID	View AMI ID	View AMI ID
Asia Pacific (Mumbai) (ap-south-1)	View AMI ID	View AMI ID	View AMI ID
Asia Pacific (Tokyo) (ap-northeast-1)	View AMI ID	View AMI ID	View AMI ID
Asia Pacific (Seoul) (ap-northeast-2)	View AMI ID	View AMI ID	View AMI ID
Asia Pacific (Osaka) (ap-northeast-3)	View AMI ID	View AMI ID	View AMI ID

AWS Region	x86	x86 accelerated	Arm
Asia Pacific (Singapore) (ap-southeast-1)	View AMI ID	View AMI ID	View AMI ID
Asia Pacific (Sydney) (ap-southeast-2)	View AMI ID	View AMI ID	View AMI ID
Asia Pacific (Jakarta) (ap-southeast-3)	View AMI ID	View AMI ID	View AMI ID
Canada (Central) (ca-central-1)	View AMI ID	View AMI ID	View AMI ID
China (Beijing) (cn-north-1)	View AMI ID	View AMI ID	View AMI ID
China (Ningxia) (cn-northwest-1)	View AMI ID	View AMI ID	View AMI ID
Europe (Frankfurt) (eu-central-1)	View AMI ID	View AMI ID	View AMI ID
Europe (Ireland) (eu-west-1)	View AMI ID	View AMI ID	View AMI ID
Europe (London) (eu-west-2)	View AMI ID	View AMI ID	View AMI ID
Europe (Milan) (eu-south-1)	View AMI ID	View AMI ID	View AMI ID
Europe (Paris) (eu-west-3)	View AMI ID	View AMI ID	View AMI ID
Europe (Stockholm) (eu-north-1)	View AMI ID	View AMI ID	View AMI ID
Middle East (Bahrain) (me-south-1)	View AMI ID	View AMI ID	View AMI ID
South America (São Paulo) (sa-east-1)	View AMI ID	View AMI ID	View AMI ID
AWS GovCloud (US-East) (us-gov-east-1)	View AMI ID	View AMI ID	View AMI ID
AWS GovCloud (US-West) (us-gov-west-1)	View AMI ID	View AMI ID	View AMI ID

Important

These AMIs require the latest AWS CloudFormation node template. You can't use these AMIs with a previous version of the node template. If you do, they fail to join your cluster. Make sure that you update any existing AWS CloudFormation node stacks with the latest template before you attempt to use these AMIs.

```
https://s3.us-west-2.amazonaws.com/amazon-eks/cloudformation/2020-10-29/amazon-eks-nodegroup.yaml
```

The AWS CloudFormation node template launches your nodes with Amazon EC2 user data that triggers a specialized [bootstrap script](#). This script allows your nodes to discover and connect to your cluster's control plane automatically. For more information, see [Launching self-managed Amazon Linux nodes \(p. 133\)](#).

Enable the containerd runtime bootstrap flag

The Amazon EKS optimized Amazon Linux 2 AMI contains an optional bootstrap flag to enable the containerd runtime. This feature provides a clear path to migrate to containerd. Amazon EKS is ending support for Docker starting with the Kubernetes version 1.23 launch. For more information, see [Amazon EKS is ending support for Dockershim \(p. 175\)](#).

You can enable the bootstrap flag by creating one of the following types of node groups.

- **Self-managed** – Create the node group using the instructions in [Launching self-managed Amazon Linux nodes \(p. 133\)](#). Specify an Amazon EKS optimized AMI and the following text for the `BootstrapArguments` parameter.

```
--container-runtime containerd
```

- **Managed** – If you use `eksctl`, create a file named `my-nodegroup.yaml` with the following contents. Replace every `example-value` with your own values. To retrieve your desired value for `ami-1234567890abcdef0`, you can use the previous AMI tables.

```
apiVersion: eksctl.io/v1alpha5
kind: ClusterConfig
metadata:
  name: my-cluster
  region: region-code
managedNodeGroups:
  - name: my-nodegroup
    ami: ami-1234567890abcdef0
    overrideBootstrapCommand: |
      #!/bin/bash
      /etc/eks/bootstrap.sh my-cluster --container-runtime containerd
```

Note

If you launch many nodes simultaneously, you may also want to specify values for the `--apiserver-endpoint`, `--b64-cluster-ca`, and `--dns-cluster-ip` bootstrap arguments to avoid errors. For more information, see [Specifying an AMI \(p. 128\)](#).

Run the following command to create the node group.

```
eksctl create nodegroup -f my-nodegroup.yaml --version 1.22
```

If you prefer to use a different tool to create your managed node group, you must deploy the node group using a launch template. In your launch template, specify an Amazon EKS optimized AMI ID, then [deploy the node group using a launch template \(p. 124\)](#) and provide the following user data. This user data passes arguments into the `bootstrap.sh` file. For more information about the `bootstrap.sh` file, see [bootstrap.sh](#) on GitHub.

```
/etc/eks/bootstrap.sh my-cluster \
--container-runtime containerd
```

Amazon EKS optimized accelerated Amazon Linux AMIs

The Amazon EKS optimized accelerated Amazon Linux AMI is built on top of the standard Amazon EKS optimized Amazon Linux AMI. It's configured to serve as an optional image for Amazon EKS nodes to support GPU and [Inferentia](#) based workloads.

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

- NVIDIA drivers
- The `nvidia-container-runtime` (as the default runtime)
- AWS Neuron container runtime

Note

- The Amazon EKS optimized accelerated AMI only supports GPU and Inferentia based instance types. Make sure to specify these instance types in your node AWS CloudFormation template. By using the Amazon EKS optimized accelerated AMI, you agree to [NVIDIA's user license agreement \(EULA\)](#).
- The Amazon EKS optimized accelerated AMI was previously referred to as the *Amazon EKS optimized AMI with GPU support*.
- Previous versions of the Amazon EKS optimized accelerated AMI installed the `nvidia-docker` repository. The repository is no longer included in Amazon EKS AMI version v20200529 and later.

To enable GPU based workloads

The following procedure describes how to run a workload on a GPU based instance with the Amazon EKS optimized accelerated AMI. For more information about using Inferentia based workloads, see [Machine learning inference using AWS Inferentia \(p. 397\)](#).

1. After your GPU nodes join your cluster, you must apply the [NVIDIA device plugin for Kubernetes](#) as a DaemonSet on your cluster with the following command.

```
kubectl apply -f https://raw.githubusercontent.com/NVIDIA/k8s-device-plugin/v0.9.0/nvidia-device-plugin.yaml
```

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

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

To deploy a pod to test that your GPU nodes are configured properly

1. Create a file named `nvidia-smi.yaml` with the following contents. This manifest launches a Cuda container that runs `nvidia-smi` on a node.

```
apiVersion: v1
kind: Pod
metadata:
  name: nvidia-smi
spec:
  restartPolicy: OnFailure
  containers:
  - name: nvidia-smi
```

```
image: nvidia/cuda:9.2-devel
args:
- "nvidia-smi"
resources:
limits:
nvidia.com/gpu: 1
```

2. Apply the manifest with the following command.

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

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

```
kubectl logs nvidia-smi
```

The output as follows.

```
Mon Aug  6 20:23:31 2018
+-----+
| NVIDIA-SMI 396.26                 Driver Version: 396.26 |
|-----+-----+-----+
| GPU  Name      Persistence-M| Bus-Id     Disp.A  | Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap| Memory-Usage | GPU-Util  Compute M. |
|-----+-----+-----+
| 0  Tesla V100-SXM2... On   | 00000000:00:1C.0 Off |          0 |
| N/A   46C    P0    47W / 300W |        0MiB / 16160MiB |     0%      Default |
+-----+-----+-----+
+-----+
| Processes:                               GPU Memory |
| GPU  PID  Type  Process name             Usage      |
|-----+-----+-----+
| No running processes found
+-----+
```

Amazon EKS optimized Arm Amazon Linux AMIs

Arm instances deliver significant cost savings for scale-out and Arm-based applications such as web servers, containerized microservices, caching fleets, and distributed data stores. When adding Arm nodes to your cluster, review the following considerations.

Considerations

- If your cluster was deployed before August 17, 2020, you must do a one-time upgrade of critical cluster add-on manifests. This is so that Kubernetes can pull the correct image for each hardware architecture in use in your cluster. For more information about updating cluster add-ons, see [To update the Kubernetes version for your Amazon EKS cluster \(p. 38\)](#). If you deployed your cluster on or after August 17, 2020, then your coredns, kube-proxy, and Amazon VPC CNI Plugin for Kubernetes add-ons are already multi-architecture capable.
- Applications deployed to Arm nodes must be compiled for Arm.
- You can't use the [Amazon FSx for Lustre CSI driver \(p. 256\)](#) with Arm.
- If you have DaemonSets that are deployed in an existing cluster, or you want to deploy them to a new cluster that you also want to deploy Arm nodes in, then verify that your DaemonSet can run on all hardware architectures in your cluster.
- You can run Arm node groups and x86 node groups in the same cluster. If you do, consider deploying multi-architecture container images to a container repository such as Amazon Elastic Container Registry and then adding node selectors to your manifests so that Kubernetes knows what hardware

architecture a pod can be deployed to. For more information, see [Pushing a multi-architecture image](#) in the *Amazon ECR User Guide* and the [Introducing multi-architecture container images for Amazon ECR](#) blog post.

Amazon EKS optimized Amazon Linux AMI versions

This topic lists versions of the Amazon EKS optimized Amazon Linux AMIs and their corresponding versions of `kubelet`, Docker, the Linux kernel, and the [Packer build script \(p. 204\)](#) configuration.

The Amazon EKS optimized AMI metadata, including the AMI ID, for each variant can be retrieved programmatically. For more information, see [Retrieving Amazon EKS optimized Amazon Linux AMI IDs \(p. 203\)](#).

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

`k8s_major_version.k8s_minor_version.k8s_patch_version-release_date`

For a list of notable changes in each version, see [Changelog](#) on GitHub.

Amazon EKS optimized Amazon Linux AMI

The following tables list the current and previous versions of the Amazon EKS optimized Amazon Linux AMI.

Kubernetes version 1.22

Kubernetes version 1.22

AMI version	kubelet version	Docker version	Kernel version	Packer version	Containerd version
1.22.6-20220429	1.22.6	20.10.13-2.amzn2	5.4.188-104.520220409	1.4.13-2.amzn2.0.1	
1.22.6-20220421	1.22.6	20.10.13-2.amzn2	5.4.188-104.520220402	1.4.13-2.amzn2.0.1	
1.22.6-20220420	1.22.6	20.10.13-2.amzn2	5.4.188-104.520220400	1.4.13-2.amzn2.0.1	
1.22.6-20220406	1.22.6	20.10.13-2.amzn2	5.4.181-99.3520220406	1.4.13-2.amzn2.0.1	
1.22.6-20220317	1.22.6	20.10.7-5.amzn2	5.4.181-99.3520220327	1.4.6-8.amzn2	

Kubernetes version 1.21

Kubernetes version 1.21

AMI version	kubelet version	Docker version	Kernel version	Packer version	Containerd version
1.21.5-20220429	1.21.5	20.10.13-2.amzn2	5.4.188-104.520220409	1.4.13-2.amzn2.0.1	
1.21.5-20220421	1.21.5	20.10.13-2.amzn2	5.4.188-104.520220402	1.4.13-2.amzn2.0.1	
1.21.5-20220420	1.21.5	20.10.13-2.amzn2	5.4.188-104.520220400	1.4.13-2.amzn2.0.1	
1.21.5-20220406	1.21.5	20.10.13-2.amzn2	5.4.181-99.3520220406	1.4.13-2.amzn2.0.1	
1.21.5-20220317	1.21.5	20.10.7-5.amzn2	5.4.181-99.3520220327	1.4.6-8.amzn2	

AMI version	kubelet version	Docker version	Kernel version	Packer version	Containerd version
1.21.5-20220309	1.21.5	20.10.7-5.amzn2	5.4.181-99.3520200309	1.4.6-8.amzn2	
1.21.5-20220303	1.21.5	20.10.7-5.amzn2	5.4.176-91.3520200303	1.4.6-8.amzn2	
1.21.5-20220226	1.21.5	20.10.7-5.amzn2	5.4.176-91.3520200226	1.4.6-7.amzn2	
1.21.5-20220216	1.21.5	20.10.7-5.amzn2	5.4.176-91.3520200216	1.4.6-7.amzn2	
1.21.5-20220210	1.21.5	20.10.7-5.amzn2	5.4.176-91.352020020	1.4.6-7.amzn2	
1.21.5-20220123	1.21.5	20.10.7-5.amzn2	5.4.172-90.3520200123	1.4.6-7.amzn2	
1.21.5-20220112	1.21.5	20.10.7-5.amzn2	5.4.162-86.2750200112	1.4.6-7.amzn2	
1.21.5-20211206	1.21.5	20.10.7-5.amzn2	5.4.156	v20211206	1.4.6-7.amzn2
1.21.5-20211117	1.21.5	20.10.7-5.amzn2	5.4.156	v20211117	1.4.6-7.amzn2
1.21.5-20211109	1.21.5	20.10.7-3.amzn2	5.4.149	v20211109	1.4.6-3.amzn2
1.21.4-20211013	1.21.4	20.10.7-3.amzn2	5.4.149	v20211013	1.4.6-3.amzn2
1.21.4-20211008	1.21.4	20.10.7-3.amzn2	5.4.149	v20211008	1.4.6-3.amzn2
1.21.2-20211004	1.21.2	20.10.7-3.amzn2	5.4.144	v20211004	1.4.6-3.amzn2
1.21.2-20211003	1.21.2	20.10.7-3.amzn2	5.4.144	v20211003	1.4.6-3.amzn2
1.21.2-20210914	1.21.2	19.03.13-ce-1	5.4.141	v20210914	1.4.6-2.amzn2
1.21.2-20210830	1.21.2	19.03.13-ce-1	5.4.141	v20210830	1.4.6-2.amzn2
1.21.2-20210826	1.21.2	19.03.13-ce-1	5.4.129	v20210826	1.4.6-2.amzn2
1.21.2-20210813	1.21.2	19.03.13-ce-1	5.4.129	v20210813	1.4.6-2.amzn2
1.21.2-20210722	1.21.2	19.03.13-ce-1	5.4.129	v20210722	1.4.6-2.amzn2
1.21.2-20210720	1.21.2	19.03.13-ce-1	5.4.129	v20210720	1.4.6-2.amzn2
1.21.2-20210716	1.21.2	19.03.13-ce-1	5.4.129	v20210716	1.4.6-2.amzn2

Kubernetes version 1.20

Kubernetes version 1.20

AMI version	kubelet version	Docker version	Kernel version	Packer version	Containerd version
1.20.11-20220429	1.20.11	20.10.13-2.amzn2	5.4.188-104.352020409	1.4.13-2.amzn2.0.1	
1.20.11-20220421	1.20.11	20.10.13-2.amzn2	5.4.188-104.352020401	1.4.13-2.amzn2.0.1	
1.20.11-20220420	1.20.11	20.10.13-2.amzn2	5.4.188-104.352020400	1.4.13-2.amzn2.0.1	
1.20.11-20220406	1.20.11	20.10.13-2.amzn2	5.4.181-99.352020406	1.4.13-2.amzn2.0.1	
1.20.11-20220317	1.20.11	20.10.7-5.amzn2	5.4.181-99.352020327	1.4.6-8.amzn2	

AMI version	kubelet version	Docker version	Kernel version	Packer version	Containerd version
1.20.11-20220309	1.20.11	20.10.7-5.amzn2	5.4.181-99.3520220309	3520220309	1.4.6-8.amzn2
1.20.11-20220303	1.20.11	20.10.7-5.amzn2	5.4.176-91.3520220303	3520220303	1.4.6-8.amzn2
1.20.11-20220226	1.20.11	20.10.7-5.amzn2	5.4.176-91.3520220226	3520220226	1.4.6-7.amzn2
1.20.11-20220216	1.20.11	20.10.7-5.amzn2	5.4.176-91.3520220216	3520220216	1.4.6-7.amzn2
1.20.11-20220210	1.20.11	20.10.7-5.amzn2	5.4.176-91.3520220210	3520220210	1.4.6-7.amzn2
1.20.11-20220123	1.20.11	20.10.7-5.amzn2	5.4.172-90.3520220123	3520220123	1.4.6-7.amzn2
1.20.11-20220112	1.20.11	20.10.7-5.amzn2	5.4.162-86.2750220112	2750220112	1.4.6-7.amzn2
1.20.11-20211206	1.20.11	20.10.7-5.amzn2	5.4.156	v20211206	1.4.6-7.amzn2
1.20.11-20211117	1.20.11	20.10.7-5.amzn2	5.4.156	v20211117	1.4.6-7.amzn2
1.20.11-20211109	1.20.11	20.10.7-3.amzn2	5.4.149	v20211109	1.4.6-3.amzn2
1.20.10-20211013	1.20.10	20.10.7-3.amzn2	5.4.149	v20211013	1.4.6-3.amzn2
1.20.10-20211008	1.20.10	20.10.7-3.amzn2	5.4.149	v20211008	1.4.6-3.amzn2
1.20.7-20211004	1.20.7	20.10.7-3.amzn2	5.4.144	v20211004	1.4.6-3.amzn2
1.20.7-20211003	1.20.7	20.10.7-3.amzn2	5.4.144	v20211003	1.4.6-3.amzn2
1.20.7-20210914	1.20.7	19.03.13-ce-1	5.4.141	v20210914	1.4.6-2.amzn2
1.20.7-20210830	1.20.7	19.03.13-ce-1	5.4.141	v20210830	1.4.6-2.amzn2
1.20.4-20210826	1.20.4	19.03.13-ce-1	5.4.129	v20210826	1.4.6-2.amzn2
1.20.4-20210813	1.20.4	19.03.13-ce-1	5.4.129	v20210813	1.4.6-2.amzn2
1.20.4-20210722	1.20.4	19.03.13-ce-1	5.4.129	v20210722	1.4.6-2.amzn2
1.20.4-20210720	1.20.4	19.03.13-ce-1	5.4.129	v20210720	1.4.6-2.amzn2
1.20.4-20210716	1.20.4	19.03.13-ce-1	5.4.129	v20210716	1.4.6-2.amzn2
1.20.4-20210628	1.20.4	19.03.13-ce-1	5.4.129	v20210628	1.4.6-2.amzn2
1.20.4-20210621	1.20.4	19.03.13-ce-1	5.4.129	v20210621	1.4.6-2.amzn2
1.20.4-20210526	1.20.4	19.03.13-ce-1	5.4.129	v20210526	1.4.6-2.amzn2
1.20.4-20210519	1.20.4	19.03.13-ce-1	5.4.129	v20210519	1.4.6-2.amzn2
1.20.4-20210518	1.20.4	19.03.13-ce-1	5.4.129	v20210518	1.4.6-2.amzn2
1.20.4-20210512	1.20.4	19.03.13-ce-1	5.4.129	v20210512	1.4.6-2.amzn2

Kubernetes version 1.19

Kubernetes version 1.19

AMI version	kubelet version	Docker version	Kernel version	Packer version	Containerd version
1.19.15-20220429	1.19.15	20.10.13-2.amzn2	5.4.188-104.159220409	v20220409	1.4.13-2.amzn2.0.1
1.19.15-20220421	1.19.15	20.10.13-2.amzn2	5.4.188-104.159220402	v20220402	1.4.13-2.amzn2.0.1
1.19.15-20220420	1.19.15	20.10.13-2.amzn2	5.4.188-104.159220400	v20220400	1.4.13-2.amzn2.0.1
1.19.15-20220406	1.19.15	20.10.13-2.amzn2	5.4.181-99.35202020406	v202020406	1.4.13-2.amzn2.0.1
1.19.15-20220317	1.19.15	20.10.7-5.amzn2	5.4.181-99.35202020327	v202020327	1.4.6-8.amzn2
1.19.15-20220309	1.19.15	20.10.7-5.amzn2	5.4.181-99.35202020309	v202020309	1.4.6-8.amzn2
1.19.15-20220303	1.19.15	20.10.7-5.amzn2	5.4.176-91.35202020303	v202020303	1.4.6-8.amzn2
1.19.15-20220226	1.19.15	20.10.7-5.amzn2	5.4.176-91.35202020226	v202020226	1.4.6-7.amzn2
1.19.15-20220216	1.19.15	20.10.7-5.amzn2	5.4.176-91.35202020216	v202020216	1.4.6-7.amzn2
1.19.15-20220210	1.19.15	20.10.7-5.amzn2	5.4.176-91.35202020220	v202020220	1.4.6-7.amzn2
1.19.15-20220123	1.19.15	20.10.7-5.amzn2	5.4.172-90.35202020123	v202020123	1.4.6-7.amzn2
1.19.15-20220112	1.19.15	20.10.7-5.amzn2	5.4.162-86.25202020122	v202020122	1.4.6-7.amzn2
1.19.15-20211206	1.19.15	20.10.7-5.amzn2	5.4.156	v20211206	1.4.6-7.amzn2
1.19.15-20211117	1.19.15	20.10.7-5.amzn2	5.4.156	v20211117	1.4.6-7.amzn2
1.19.15-20211109	1.19.15	20.10.7-3.amzn2	5.4.149	v20211109	1.4.6-3.amzn2
1.19.14-20211013	1.19.14	20.10.7-3.amzn2	5.4.149	v20211013	1.4.6-3.amzn2
1.19.14-20211008	1.19.14	20.10.7-3.amzn2	5.4.149	v20211008	1.4.6-3.amzn2
1.19.13-20211004	1.19.13	20.10.7-3.amzn2	5.4.144	v20211004	1.4.6-3.amzn2
1.19.13-20211003	1.19.13	20.10.7-3.amzn2	5.4.144	v20211003	1.4.6-3.amzn2
1.19.13-20210914	1.19.13	19.03.13-ce-1	5.4.141	v20210914	1.4.6-2.amzn2
1.19.13-20210830	1.19.13	19.03.13-ce-1	5.4.141	v20210830	1.4.6-2.amzn2
1.19.13-20210826	1.19.13	19.03.13-ce-1	5.4.129	v20210826	1.4.6-2.amzn2
1.19.13-20210813	1.19.13	19.03.13-ce-1	5.4.129	v20210813	1.4.6-2.amzn2
1.19.6-20210722	1.19.6	19.03.13-ce-1	5.4.129	v20210722	1.4.6-2.amzn2
1.19.6-20210720	1.19.6	19.03.13-ce-1	5.4.129	v20210720	1.4.6-2.amzn2
1.19.6-20210716	1.19.6	19.03.13-ce-1	5.4.129	v20210716	1.4.6-2.amzn2
1.19.6-20210628	1.19.6	19.03.13-ce-1	5.4.129	v20210628	1.4.6-2.amzn2
1.19.6-20210621	1.19.6	19.03.13-ce-1	5.4.129	v20210621	1.4.6-2.amzn2
1.19.6-20210526	1.19.6	19.03.13-ce-1	5.4.129	v20210526	1.4.6-2.amzn2

AMI version	kubelet version	Docker version	Kernel version	Packer version	Containerd version
1.19.6-20210519	1.19.6	19.03.13-ce-1	5.4.129	v20210519	1.4.6-2.amzn2
1.19.6-20210518	1.19.6	19.03.13-ce-1	5.4.129	v20210518	1.4.6-2.amzn2
1.19.6-20210512	1.19.6	19.03.13-ce-1	5.4.129	v20210512	1.4.6-2.amzn2
1.19.6-20210504	1.19.6	19.03.13-ce-1	5.4.129	v20210504	1.4.6-2.amzn2
1.19.6-20210501	1.19.6	19.03.13-ce-1	5.4.129	v20210501	1.4.6-2.amzn2
1.19.6-20210414	1.19.6	19.03.13-ce-1	5.4.129	v20210414	1.4.6-2.amzn2
1.19.6-20210329	1.19.6	19.03.13-ce-1	5.4.129	v20210329	1.4.6-2.amzn2
1.19.6-20210322	1.19.6	19.03.13-ce-1	5.4.129	v20210322	1.4.6-2.amzn2
1.19.6-20210310	1.19.6	19.03.13-ce-1	5.4.129	v20210310	1.4.6-2.amzn2
1.19.6-20210302	1.19.6	19.03.13-ce-1	5.4.129	v20210302	1.4.6-2.amzn2
1.19.6-20210208	1.19.6	19.03.6-ce-4	5.4.129	v20210208	1.4.6-2.amzn2

Kubernetes version 1.18

Kubernetes version 1.18

AMI version	kubelet version	Docker version	Kernel version	Packer version	Containerd version
1.18.20-20220406	1.18.20	20.10.13-2.amzn2	4.14.268-205202204062	v202204062	1.4.13-2.amzn2.0.1
1.18.20-20220317	1.18.20	20.10.7-5.amzn2	4.14.268-205202203172	v202203172	1.4.6-8.amzn2
1.18.20-20220309	1.18.20	20.10.7-5.amzn2	4.14.268-205202203092	v202203092	1.4.6-8.amzn2
1.18.20-20220303	1.18.20	20.10.7-5.amzn2	4.14.262-200202203032	v202203032	1.4.6-8.amzn2
1.18.20-20220226	1.18.20	20.10.7-5.amzn2	4.14.262-200202202262	v202202262	1.4.6-7.amzn2
1.18.20-20220216	1.18.20	20.10.7-5.amzn2	4.14.262-200202202162	v202202162	1.4.6-7.amzn2
1.18.20-20220210	1.18.20	20.10.7-5.amzn2	4.14.262-200202202102	v202202102	1.4.6-7.amzn2
1.18.20-20220123	1.18.20	20.10.7-5.amzn2	4.14.256-197202201232	v202201232	1.4.6-7.amzn2
1.18.20-20220112	1.18.20	20.10.7-5.amzn2	4.14.256-197202201122	v202201122	1.4.6-7.amzn2
1.18.20-20211206	1.18.20	20.10.7-5.amzn2	4.14.252	v20211206	1.4.6-7.amzn2
1.18.20-20211117	1.18.20	20.10.7-5.amzn2	4.14.252	v20211117	1.4.6-7.amzn2
1.18.20-20211109	1.18.20	20.10.7-3.amzn2	4.14.252	v20211109	1.4.6-3.amzn2
1.18.20-20211013	1.18.20	20.10.7-3.amzn2	4.14.248	v20211013	1.4.6-3.amzn2
1.18.20-20211008	1.18.20	20.10.7-3.amzn2	4.14.248	v20211008	1.4.6-3.amzn2
1.18.20-20211004	1.18.20	20.10.7-3.amzn2	4.14.246	v20211004	1.4.6-3.amzn2

AMI version	kubelet version	Docker version	Kernel version	Packer version	Containerd version
1.18.20-20211003	1.18.20	20.10.7-3.amzn2	4.14.246	v20211003	1.4.6-3.amzn2
1.18.20-20210914	1.18.20	19.03.13-ce-1	4.14.243	v20210914	1.4.6-2.amzn2
1.18.20-20210830	1.18.20	19.03.13-ce-1	4.14.243	v20210830	1.4.6-2.amzn2
1.18.20-20210826	1.18.20	19.03.13-ce-1	4.14.238	v20210826	1.4.6-2.amzn2
1.18.20-20210813	1.18.20	19.03.13-ce-1	4.14.238	v20210813	1.4.6-2.amzn2
1.18.9-20210722	1.18.9	19.03.13-ce-1	4.14.238	v20210722	1.4.6-2.amzn2
1.18.9-20210720	1.18.9	19.03.13-ce-1	4.14.238	v20210720	1.4.6-2.amzn2
1.18.9-20210716	1.18.9	19.03.13-ce-1	4.14.238	v20210716	1.4.6-2.amzn2
1.18.9-20210628	1.18.9	19.03.13-ce-1	4.14.238	v20210628	1.4.6-2.amzn2
1.18.9-20210621	1.18.9	19.03.13-ce-1	4.14.238	v20210621	1.4.6-2.amzn2
1.18.9-20210526	1.18.9	19.03.13-ce-1	4.14.238	v20210526	1.4.6-2.amzn2
1.18.9-20210519	1.18.9	19.03.13-ce-1	4.14.238	v20210519	1.4.6-2.amzn2
1.18.9-20210518	1.18.9	19.03.13-ce-1	4.14.238	v20210518	1.4.6-2.amzn2
1.18.9-20210512	1.18.9	19.03.13-ce-1	4.14.238	v20210512	1.4.6-2.amzn2
1.18.9-20210504	1.18.9	19.03.13-ce-1	4.14.238	v20210504	1.4.6-2.amzn2
1.18.9-20210501	1.18.9	19.03.13-ce-1	4.14.238	v20210501	1.4.6-2.amzn2
1.18.9-20210414	1.18.9	19.03.13-ce-1	4.14.238	v20210414	1.4.6-2.amzn2
1.18.9-20210329	1.18.9	19.03.13-ce-1	4.14.238	v20210329	1.4.6-2.amzn2
1.18.9-20210322	1.18.9	19.03.13-ce-1	4.14.238	v20210322	1.4.6-2.amzn2
1.18.9-20210310	1.18.9	19.03.13-ce-1	4.14.238	v20210310	1.4.6-2.amzn2
1.18.9-20210302	1.18.9	19.03.13-ce-1	4.14.238	v20210302	1.4.6-2.amzn2
1.18.9-20210208	1.18.9	19.03.6-ce-4	4.14.238	v20210208	1.4.6-2.amzn2
1.18.9-20210125	1.18.9	19.03.6-ce-4	4.14.238	v20210125	1.4.6-2.amzn2
1.18.9-20210112	1.18.9	19.03.6-ce-4	4.14.238	v20210112	1.4.6-2.amzn2
1.18.9-20201211	1.18.9	19.03.6-ce-4	4.14.238	v20201211	1.4.6-2.amzn2
1.18.9-20201126	1.18.9	19.03.6-ce-4	4.14.238	v20201126	1.4.6-2.amzn2
1.18.9-20201117	1.18.9	19.03.6-ce-4	4.14.238	v20201117	1.4.6-2.amzn2
1.18.9-20201112	1.18.9	19.03.6-ce-4	4.14.238	v20201112	1.4.6-2.amzn2
1.18.8-20201007	1.18.8	19.03.6-ce-4	4.14.238	v20201007	1.4.6-2.amzn2

Kubernetes version 1.17

Kubernetes version 1.17

AMI version	kubelet version	Docker version	Kernel version	Packer version	Containerd version
1.17.17-20211117	1.17.17	20.10.7-5.amzn2	4.14.252	v20211117	1.4.6-7.amzn2
1.17.17-20211109	1.17.17	20.10.7-3.amzn2	4.14.252	v20211109	1.4.6-3.amzn2
1.17.17-20211013	1.17.17	20.10.7-3.amzn2	4.14.248	v20211013	1.4.6-3.amzn2
1.17.17-20211008	1.17.17	20.10.7-3.amzn2	4.14.248	v20211008	1.4.6-3.amzn2
1.17.17-20211004	1.17.17	20.10.7-3.amzn2	4.14.246	v20211004	1.4.6-3.amzn2
1.17.17-20211003	1.17.17	20.10.7-3.amzn2	4.14.246	v20211003	1.4.6-3.amzn2
1.17.17-20210914	1.17.17	19.03.13-ce-1	4.14.243	v20210914	1.4.6-2.amzn2
1.17.17-20210830	1.17.17	19.03.13-ce-1	4.14.243	v20210830	1.4.6-2.amzn2
1.17.12-20210826	1.17.12	19.03.13-ce-1	4.14.238	v20210826	1.4.6-2.amzn2
1.17.12-20210813	1.17.12	19.03.13-ce-1	4.14.238	v20210813	1.4.6-2.amzn2
1.17.12-20210722	1.17.12	19.03.13-ce-1	4.14.238	v20210722	1.4.6-2.amzn2
1.17.12-20210720	1.17.12	19.03.13-ce-1	4.14.238	v20210720	1.4.6-2.amzn2
1.17.12-20210716	1.17.12	19.03.13-ce-1	4.14.238	v20210716	1.4.6-2.amzn2
1.17.12-20210628	1.17.12	19.03.13-ce-1	4.14.238	v20210628	1.4.6-2.amzn2
1.17.12-20210621	1.17.12	19.03.13-ce-1	4.14.238	v20210621	1.4.6-2.amzn2
1.17.12-20210526	1.17.12	19.03.13-ce-1	4.14.238	v20210526	1.4.6-2.amzn2
1.17.12-20210519	1.17.12	19.03.13-ce-1	4.14.238	v20210519	1.4.6-2.amzn2
1.17.12-20210518	1.17.12	19.03.13-ce-1	4.14.238	v20210518	1.4.6-2.amzn2
1.17.12-20210512	1.17.12	19.03.13-ce-1	4.14.238	v20210512	1.4.6-2.amzn2
1.17.12-20210504	1.17.12	19.03.13-ce-1	4.14.238	v20210504	1.4.6-2.amzn2
1.17.12-20210501	1.17.12	19.03.13-ce-1	4.14.238	v20210501	1.4.6-2.amzn2
1.17.12-20210414	1.17.12	19.03.13-ce-1	4.14.238	v20210414	1.4.6-2.amzn2
1.17.12-20210329	1.17.12	19.03.13-ce-1	4.14.238	v20210329	1.4.6-2.amzn2
1.17.12-20210322	1.17.12	19.03.13-ce-1	4.14.238	v20210322	1.4.6-2.amzn2
1.17.12-20210310	1.17.12	19.03.13-ce-1	4.14.238	v20210310	1.4.6-2.amzn2
1.17.12-20210302	1.17.12	19.03.13-ce-1	4.14.238	v20210302	1.4.6-2.amzn2
1.17.12-20210208	1.17.12	19.03.6-ce-4	4.14.238	v20210208	1.4.6-2.amzn2
1.17.12-20210125	1.17.12	19.03.6-ce-4	4.14.238	v20210125	1.4.6-2.amzn2
1.17.12-20210112	1.17.12	19.03.6-ce-4	4.14.238	v20210112	1.4.6-2.amzn2

AMI version	kubelet version	Docker version	Kernel version	Packer version	Containerd version
1.17.12-20201211	1.17.12	19.03.6-ce-4	4.14.238	v20201211	1.4.6-2.amzn2
1.17.12-20201126	1.17.12	19.03.6-ce-4	4.14.238	v20201126	1.4.6-2.amzn2
1.17.12-20201117	1.17.12	19.03.6-ce-4	4.14.238	v20201117	1.4.6-2.amzn2
1.17.12-20201112	1.17.12	19.03.6-ce-4	4.14.238	v20201112	1.4.6-2.amzn2
1.17.11-20201007	1.17.11	19.03.6-ce-4	4.14.238	v20201007	1.4.6-2.amzn2
1.17.11-20201002	1.17.11	19.03.6-ce-4	4.14.238	v20201002	1.4.6-2.amzn2
1.17.11-20200921	1.17.11	19.03.6-ce-4	4.14.238	v20200921	1.4.6-2.amzn2
1.17.9-20200904	1.17.9	19.03.6-ce-4	4.14.238	v20200904	1.4.6-2.amzn2
1.17.9-20200814	1.17.9	19.03.6-ce-4	4.14.238	v20200814	1.4.6-2.amzn2
1.17.9-20200723	1.17.9	19.03.6-ce	4.14.238	v20200723	1.4.6-2.amzn2
1.17.7-20200710	1.17.7	19.03.6-ce	4.14.238	v20200710	1.4.6-2.amzn2
1.17.7-20200709	1.17.7	19.03.6-ce	4.14.238	v20200709	1.4.6-2.amzn2

Amazon EKS optimized accelerated Amazon Linux AMI

The tables below list the current and previous versions of the Amazon EKS optimized accelerated Amazon Linux AMI.

Kubernetes version 1.22

Kubernetes version 1.22

AMI version	kubelet version	Docker version	Kernel version	Packer version	Containerd version	NVIDIA driver version
1.22.6-20220429	1.22.6	20.10.13-2.arm64h288-1042329.0422h2	4.14.238	v20220429	1.4.13-2.amzn2	470.57.02-1
1.22.6-20220421	1.22.6	20.10.13-2.arm64h288-1042329.0422h2	4.14.238	v20220421	1.4.13-2.amzn2	470.57.02-1
1.22.6-20220420	1.22.6	20.10.13-2.arm64h288-1042329.0422h2	4.14.238	v20220420	1.4.13-2.amzn2	470.57.02-1
1.22.6-20220406	1.22.6	20.10.13-2.arm64h281-99v30220406	4.14.238	v20220406	1.4.13-2.amzn2	470.57.02-1
1.22.6-20220317	1.22.6	20.10.7-5.arm64h2181-99v30220317	4.14.238	v20220317	1.4.6-8.amzn2	470.57.02-1

Kubernetes version 1.21

Kubernetes version 1.21

AMI version	kubelet version	Docker version	Kernel version	Packer version	Containerd version	NVIDIA driver version
1.21.5-20220429	1.21.5	20.10.13-2.arm64h288-1042329.0422h2	4.14.238	v20220429	1.4.13-2.amzn2	470.57.02-1

AMI version	kubelet version	Docker version	Kernel version	Packer version	Containerd version	NVIDIA driver version
1.21.5-20220421	1.21.5	20.10.13-2.amzn2-19.12.88-1042029.0.4212	5.15.1-1.21.5-20220421	1.4.13-2.amzn2-1.4.13-20220421	1.4.13-2.amzn2-1.4.13-20220421	470.57.02-1
1.21.5-20220420	1.21.5	20.10.13-2.amzn2-19.12.88-1042029.0.4202	5.15.1-1.21.5-20220420	1.4.13-2.amzn2-1.4.13-20220420	1.4.13-2.amzn2-1.4.13-20220420	470.57.02-1
1.21.5-20220406	1.21.5	20.10.13-2.amzn2-19.12.81-99.352.204.062	5.15.1-1.21.5-20220406	1.4.13-2.amzn2-1.4.13-20220406	1.4.13-2.amzn2-1.4.13-20220406	470.57.02-1
1.21.5-20220317	1.21.5	20.10.7-5.amzn2-181-99.352.203.172	5.15.1-1.21.5-20220317	1.4.6-8.amzn2-1.4.6-8.20220317	1.4.6-8.amzn2-1.4.6-8.20220317	470.57.02-1
1.21.5-20220309	1.21.5	20.10.7-5.amzn2-181-99.352.203.092	5.15.1-1.21.5-20220309	1.4.6-8.amzn2-1.4.6-8.20220309	1.4.6-8.amzn2-1.4.6-8.20220309	470.57.02-1
1.21.5-20220303	1.21.5	20.10.7-5.amzn2-176-91.352.203.032	5.15.1-1.21.5-20220303	1.4.6-8.amzn2-1.4.6-8.20220303	1.4.6-8.amzn2-1.4.6-8.20220303	470.57.02-1
1.21.5-20220226	1.21.5	20.10.7-5.amzn2-176-91.352.202.2262	5.15.1-1.21.5-20220226	1.4.6-7.amzn2-1.4.6-7.20220226	1.4.6-7.amzn2-1.4.6-7.20220226	470.57.02-1
1.21.5-20220216	1.21.5	20.10.7-5.amzn2-176-91.352.202.2162	5.15.1-1.21.5-20220216	1.4.6-7.amzn2-1.4.6-7.20220216	1.4.6-7.amzn2-1.4.6-7.20220216	470.57.02-1
1.21.5-20220210	1.21.5	20.10.7-5.amzn2-176-91.352.202.2102	5.15.1-1.21.5-20220210	1.4.6-7.amzn2-1.4.6-7.20220210	1.4.6-7.amzn2-1.4.6-7.20220210	470.57.02-1
1.21.5-20220123	1.21.5	20.10.7-5.amzn2-172-90.352.201.232	5.15.1-1.21.5-20220123	1.4.6-7.amzn2-1.4.6-7.20220123	1.4.6-7.amzn2-1.4.6-7.20220123	470.57.02
1.21.5-20220112	1.21.5	20.10.7-5.amzn2-162-86.352.201.122	5.15.1-1.21.5-20220112	1.4.6-7.amzn2-1.4.6-7.20220112	1.4.6-7.amzn2-1.4.6-7.20220112	470.57.02
1.21.5-20211206	1.21.5	20.10.7-5.amzn2-156	v20211206	1.4.6-7.amzn2-1.4.6-7.20211206	1.4.6-7.amzn2-1.4.6-7.20211206	470.57.02
1.21.5-20211117	1.21.5	20.10.7-5.amzn2-156	v20211117	1.4.6-7.amzn2-1.4.6-7.20211117	1.4.6-7.amzn2-1.4.6-7.20211117	470.57.02
1.21.5-20211109	1.21.5	20.10.7-3.amzn2-149	v20211109	1.4.6-3.amzn2-1.4.6-3.20211109	1.4.6-3.amzn2-1.4.6-3.20211109	470.57.02
1.21.4-20211013	1.21.4	20.10.7-3.amzn2-149	v20211013	1.4.6-3.amzn2-1.4.6-3.20211013	1.4.6-3.amzn2-1.4.6-3.20211013	460.73.01
1.21.4-20211008	1.21.4	20.10.7-3.amzn2-149	v20211008	1.4.6-3.amzn2-1.4.6-3.20211008	1.4.6-3.amzn2-1.4.6-3.20211008	460.73.01
1.21.2-20211004	1.21.2	20.10.7-3.amzn2-144	v20211004	1.4.6-3.amzn2-1.4.6-3.20211004	1.4.6-3.amzn2-1.4.6-3.20211004	460.73.01
1.21.2-20211003	1.21.2	20.10.7-3.amzn2-144	v20211003	1.4.6-3.amzn2-1.4.6-3.20211003	1.4.6-3.amzn2-1.4.6-3.20211003	460.73.01
1.21.2-20210914	1.21.2	19.03.13-ce-1	5.4.141	v20210914	1.4.6-2.amzn2-1.4.6-2.20210914	460.73.01
1.21.2-20210830	1.21.2	19.03.13-ce-1	5.4.141	v20210830	1.4.6-2.amzn2-1.4.6-2.20210830	460.73.01
1.21.2-20210826	1.21.2	19.03.13-ce-1	5.4.129	v20210826	1.4.6-2.amzn2-1.4.6-2.20210826	460.73.01
1.21.2-20210813	1.21.2	19.03.13-ce-1	5.4.129	v20210813	1.4.6-2.amzn2-1.4.6-2.20210813	460.73.01
1.21.2-20210722	1.21.2	19.03.13-ce-1	5.4.129	v20210722	1.4.6-2.amzn2-1.4.6-2.20210722	460.73.01
1.21.2-20210720	1.21.2	19.03.13-ce-1	5.4.129	v20210720	1.4.6-2.amzn2-1.4.6-2.20210720	460.73.01
1.21.2-20210716	1.21.2	19.03.13-ce-1	5.4.129	v20210716	1.4.6-2.amzn2-1.4.6-2.20210716	460.73.01

Kubernetes version 1.20

Kubernetes version 1.20

AMI version	kubelet version	Docker version	Kernel version	Packer version	Containerd version	NVIDIA driver version
1.20.11-20220429	1.20.11	20.10.13-2.amzn2-k8s1.288-10420599.04.20220429	5.15.4-1.20220429.20220429.2	v20220429	1.4.13-2.amzn2	470.57.02-1
1.20.11-20220421	1.20.11	20.10.13-2.amzn2-k8s1.288-10420599.04.20220421	5.15.4-1.20220421.20220421.2	v20220421	1.4.13-2.amzn2	470.57.02-1
1.20.11-20220420	1.20.11	20.10.13-2.amzn2-k8s1.288-10420599.04.20220420	5.15.4-1.20220420.20220420.2	v20220420	1.4.13-2.amzn2	470.57.02-1
1.20.11-20220406	1.20.11	20.10.13-2.amzn2-k8s1.281-99.v20220406	5.15.4-1.20220406.20220406.2	v20220406	1.4.13-2.amzn2	470.57.02-1
1.20.11-20220317	1.20.11	20.10.7-5.amzn2-k8s1.2181-99.v20220317	5.15.4-1.20220317.20220317.2	v20220317	1.4.6-8.amzn2	470.57.02-1
1.20.11-20220309	1.20.11	20.10.7-5.amzn2-k8s1.2181-99.v20220309	5.15.4-1.20220309.20220309.2	v20220309	1.4.6-8.amzn2	470.57.02-1
1.20.11-20220303	1.20.11	20.10.7-5.amzn2-k8s1.2176-91.v20220303	5.15.4-1.20220303.20220303.2	v20220303	1.4.6-8.amzn2	470.57.02-1
1.20.11-20220226	1.20.11	20.10.7-5.amzn2-k8s1.2176-91.v20220226	5.15.4-1.20220226.20220226.2	v20220226	1.4.6-7.amzn2	470.57.02-1
1.20.11-20220216	1.20.11	20.10.7-5.amzn2-k8s1.2176-91.v20220216	5.15.4-1.20220216.20220216.2	v20220216	1.4.6-7.amzn2	470.57.02-1
1.20.11-20220210	1.20.11	20.10.7-5.amzn2-k8s1.2176-91.v20220210	5.15.4-1.20220210.20220210.2	v20220210	1.4.6-7.amzn2	470.57.02-1
1.20.11-20220123	1.20.11	20.10.7-5.amzn2-k8s1.2172-90.v20220123	5.15.4-1.20220123.20220123.2	v20220123	1.4.6-7.amzn2	470.57.02
1.20.11-20220112	1.20.11	20.10.7-5.amzn2-k8s1.2162-86.v20220112	5.15.4-1.20220112.20220112.2	v20220112	1.4.6-7.amzn2	470.57.02
1.20.11-20211206	1.20.11	20.10.7-5.amzn2-k8s1.2156	5.15.4-1.20211206.20211206.2	v20211206	1.4.6-7.amzn2	470.57.02
1.20.11-20211117	1.20.11	20.10.7-5.amzn2-k8s1.2156	5.15.4-1.20211117.20211117.2	v20211117	1.4.6-7.amzn2	470.57.02
1.20.11-20211109	1.20.11	20.10.7-3.amzn2-k8s1.2149	5.15.4-1.20211109.20211109.2	v20211109	1.4.6-3.amzn2	470.57.02
1.20.10-20211013	1.20.10	20.10.7-3.amzn2-k8s1.2149	5.15.4-1.20211013.20211013.2	v20211013	1.4.6-3.amzn2	460.73.01
1.20.10-20211008	1.20.10	20.10.7-3.amzn2-k8s1.2149	5.15.4-1.20211008.20211008.2	v20211008	1.4.6-3.amzn2	460.73.01
1.20.7-20211004	1.20.7	20.10.7-3.amzn2-k8s1.2144	5.15.4-1.20211004.20211004.2	v20211004	1.4.6-3.amzn2	460.73.01
1.20.7-20211003	1.20.7	20.10.7-3.amzn2-k8s1.2144	5.15.4-1.20211003.20211003.2	v20211003	1.4.6-3.amzn2	460.73.01
1.20.7-20210914	1.20.7	19.03.13-ce-1	5.4.141	v20210914	1.4.6-2.amzn2	460.73.01
1.20.7-20210830	1.20.7	19.03.13-ce-1	5.4.141	v20210830	1.4.6-2.amzn2	460.73.01
1.20.4-20210826	1.20.4	19.03.13-ce-1	5.4.129	v20210826	1.4.6-2.amzn2	460.73.01
1.20.4-20210813	1.20.4	19.03.13-ce-1	5.4.129	v20210813	1.4.6-2.amzn2	460.73.01
1.20.4-20210722	1.20.4	19.03.13-ce-1	5.4.129	v20210722	1.4.6-2.amzn2	460.73.01
1.20.4-20210720	1.20.4	19.03.13-ce-1	5.4.129	v20210720	1.4.6-2.amzn2	460.73.01

AMI version	kubelet version	Docker version	Kernel version	Packer version	Containerd version	NVIDIA driver version
1.20.4-20210716	1.20.4	19.03.13-ce-1	5.4.129	v20210716	1.4.6-2.amzn2	460.73.01
1.20.4-20210628	1.20.4	19.03.13-ce-1	5.4.129	v20210628	1.4.6-2.amzn2	460.73.01
1.20.4-20210621	1.20.4	19.03.13-ce-1	5.4.129	v20210621	1.4.6-2.amzn2	460.73.01
1.20.4-20210526	1.20.4	19.03.13-ce-1	5.4.129	v20210526	1.4.6-2.amzn2	460.73.01
1.20.4-20210519	1.20.4	19.03.13-ce-1	5.4.129	v20210519	1.4.6-2.amzn2	460.73.01
1.20.4-20210518	1.20.4	19.03.13-ce-1	5.4.129	v20210518	1.4.6-2.amzn2	460.73.01
1.20.4-20210512	1.20.4	19.03.13-ce-1	5.4.129	v20210512	1.4.6-2.amzn2	460.73.01

Kubernetes version 1.19

Kubernetes version 1.19

AMI version	kubelet version	Docker version	Kernel version	Packer version	Containerd version	NVIDIA driver version
1.19.15-20220429	1.19.15	20.10.13-2.amzn2-19.03.13-1042022042912	5.4.128-1042022042912	1.4.13-2.amzn2	470.57.02-1	470.57.02-1
1.19.15-20220421	1.19.15	20.10.13-2.amzn2-19.03.13-1042022042112	5.4.128-1042022042112	1.4.13-2.amzn2	470.57.02-1	470.57.02-1
1.19.15-20220420	1.19.15	20.10.13-2.amzn2-19.03.13-1042022042012	5.4.128-1042022042012	1.4.13-2.amzn2	470.57.02-1	470.57.02-1
1.19.15-20220406	1.19.15	20.10.13-2.amzn2-19.03.13-1042022040612	5.4.128-1042022040612	1.4.13-2.amzn2	470.57.02-1	470.57.02-1
1.19.15-20220317	1.19.15	20.10.7-5.amzn2-19.03.13-99v2022031712	5.4.128-99v2022031712	1.4.6-8.amzn2	470.57.02-1	470.57.02-1
1.19.15-20220309	1.19.15	20.10.7-5.amzn2-19.03.13-99v2022030912	5.4.128-99v2022030912	1.4.6-8.amzn2	470.57.02-1	470.57.02-1
1.19.15-20220303	1.19.15	20.10.7-5.amzn2-19.03.13-91v2022030312	5.4.128-91v2022030312	1.4.6-8.amzn2	470.57.02-1	470.57.02-1
1.19.15-20220226	1.19.15	20.10.7-5.amzn2-19.03.13-91v2022022612	5.4.128-91v2022022612	1.4.6-7.amzn2	470.57.02-1	470.57.02-1
1.19.15-20220216	1.19.15	20.10.7-5.amzn2-19.03.13-91v2022021612	5.4.128-91v2022021612	1.4.6-7.amzn2	470.57.02-1	470.57.02-1
1.19.15-20220210	1.19.15	20.10.7-5.amzn2-19.03.13-91v2022021012	5.4.128-91v2022021012	1.4.6-7.amzn2	470.57.02-1	470.57.02-1
1.19.15-20220123	1.19.15	20.10.7-5.amzn2-19.03.13-90v2022012312	5.4.128-90v2022012312	1.4.6-7.amzn2	470.57.02	
1.19.15-20220112	1.19.15	20.10.7-5.amzn2-19.03.13-86v2022011212	5.4.128-86v2022011212	1.4.6-7.amzn2	470.57.02	
1.19.15-20211206	1.19.15	20.10.7-5.amzn2-19.03.13-156v2021120612	5.4.128-156v2021120612	v20211206	1.4.6-7.amzn2	470.57.02
1.19.15-20211117	1.19.15	20.10.7-5.amzn2-19.03.13-156v2021111712	5.4.128-156v2021111712	v20211117	1.4.6-7.amzn2	470.57.02

AMI version	kubebundle version	Docker version	Kernel version	Packer version	Containerd version	NVIDIA driver version
1.19.15-20211109	1.19.15	20.10.7-3.amzn2-2149	v20211109	1.4.6-3.amzn2	470.57.02	
1.19.14-20211013	1.19.14	20.10.7-3.amzn2-2149	v20211013	1.4.6-3.amzn2	460.73.01	
1.19.14-20211008	1.19.14	20.10.7-3.amzn2-2149	v20211008	1.4.6-3.amzn2	460.73.01	
1.19.13-20211004	1.19.13	20.10.7-3.amzn2-2144	v20211004	1.4.6-3.amzn2	460.73.01	
1.19.13-20211003	1.19.13	20.10.7-3.amzn2-2144	v20211003	1.4.6-3.amzn2	460.73.01	
1.19.13-20210914	1.19.13	19.03.13-ce-1	5.4.141	v20210914	1.4.6-2.amzn2	460.73.01
1.19.13-20210830	1.19.13	19.03.13-ce-1	5.4.141	v20210830	1.4.6-2.amzn2	460.73.01
1.19.13-20210826	1.19.13	19.03.13-ce-1	5.4.129	v20210826	1.4.6-2.amzn2	460.73.01
1.19.13-20210813	1.19.13	19.03.13-ce-1	5.4.129	v20210813	1.4.6-2.amzn2	460.73.01
1.19.6-20210722	1.19.6	19.03.13-ce-1	5.4.129	v20210722	1.4.6-2.amzn2	460.73.01
1.19.6-20210720	1.19.6	19.03.13-ce-1	5.4.129	v20210720	1.4.6-2.amzn2	460.73.01
1.19.6-20210716	1.19.6	19.03.13-ce-1	5.4.129	v20210716	1.4.6-2.amzn2	460.73.01
1.19.6-20210628	1.19.6	19.03.13-ce-1	5.4.129	v20210628	1.4.6-2.amzn2	460.73.01
1.19.6-20210621	1.19.6	19.03.13-ce-1	5.4.129	v20210621	1.4.6-2.amzn2	460.73.01
1.19.6-20210526	1.19.6	19.03.13-ce-1	5.4.129	v20210526	1.4.6-2.amzn2	460.73.01
1.19.6-20210519	1.19.6	19.03.13-ce-1	5.4.129	v20210519	1.4.6-2.amzn2	460.73.01
1.19.6-20210518	1.19.6	19.03.13-ce-1	5.4.129	v20210518	1.4.6-2.amzn2	460.73.01
1.19.6-20210512	1.19.6	19.03.13-ce-1	5.4.129	v20210512	1.4.6-2.amzn2	460.73.01
1.19.6-20210504	1.19.6	19.03.13-ce-1	5.4.129	v20210504	1.4.6-2.amzn2	460.73.01
1.19.6-20210501	1.19.6	19.03.13-ce-1	5.4.129	v20210501	1.4.6-2.amzn2	460.32.03
1.19.6-20210414	1.19.6	19.03.13-ce-1	5.4.129	v20210414	1.4.6-2.amzn2	460.32.03

AMI version	kubelet version	Docker version	Kernel version	Packer version	Containerd version	NVIDIA driver version
1.19.6-20210329	1.19.6	19.03.13-ce-1	5.4.129	v20210329	1.4.6-2.amzn2	460.32.03
1.19.6-20210322	1.19.6	19.03.13-ce-1	5.4.129	v20210322	1.4.6-2.amzn2	460.27.04
1.19.6-20210310	1.19.6	19.03.13-ce-1	5.4.129	v20210310	1.4.6-2.amzn2	460.27.04
1.19.6-20210302	1.19.6	19.03.13-ce-1	5.4.129	v20210302	1.4.6-2.amzn2	450.51.06
1.19.6-20210208	1.19.6	19.03.6-ce-4	5.4.129	v20210208	1.4.6-2.amzn2	450.51.06

Kubernetes version 1.18

Kubernetes version 1.18

AMI version	kubelet version	Docker version	Kernel version	Packer version	Containerd version	NVIDIA driver version
1.18.20-20220406	1.18.20	20.10.13-2.amzn2	4.14.2268-202020406zn2	1.4.13-2.amzn2	470.57.02-1	
1.18.20-20220317	1.18.20	20.10.7-5.amzn2	4.14.268-202020317zn2	1.4.6-8.amzn2	470.57.02-1	
1.18.20-20220309	1.18.20	20.10.7-5.amzn2	4.14.268-202020309zn2	1.4.6-8.amzn2	470.57.02-1	
1.18.20-20220303	1.18.20	20.10.7-5.amzn2	4.14.262-202020303zn2	1.4.6-8.amzn2	470.57.02-1	
1.18.20-20220226	1.18.20	20.10.7-5.amzn2	4.14.262-202020326zn2	1.4.6-7.amzn2	470.57.02-1	
1.18.20-20220216	1.18.20	20.10.7-5.amzn2	4.14.262-202020316zn2	1.4.6-7.amzn2	470.57.02-1	
1.18.20-20220210	1.18.20	20.10.7-5.amzn2	4.14.262-202020310zn2	1.4.6-7.amzn2	470.57.02-1	
1.18.20-20220123	1.18.20	20.10.7-5.amzn2	4.14.256-1920200323zn2	1.4.6-7.amzn2	470.57.02	
1.18.20-20220112	1.18.20	20.10.7-5.amzn2	4.14.256-1920200312zn2	1.4.6-7.amzn2	470.57.02	
1.18.20-20211206	1.18.20	20.10.7-5.amzn2	4.14.252	v20211206	1.4.6-7.amzn2	470.57.02
1.18.20-20211117	1.18.20	20.10.7-5.amzn2	4.14.252	v20211117	1.4.6-7.amzn2	470.57.02
1.18.20-20211109	1.18.20	20.10.7-3.amzn2	4.14.252	v20211109	1.4.6-3.amzn2	470.57.02
1.18.20-20211103	1.18.20	20.10.7-3.amzn2	4.14.248	v20211103	1.4.6-3.amzn2	460.73.01
1.18.20-20211008	1.18.20	20.10.7-3.amzn2	4.14.248	v20211008	1.4.6-3.amzn2	460.73.01
1.18.20-20211004	1.18.20	20.10.7-3.amzn2	4.14.246	v20211004	1.4.6-3.amzn2	460.73.01
1.18.20-20211003	1.18.20	20.10.7-3.amzn2	4.14.246	v20211003	1.4.6-3.amzn2	460.73.01
1.18.20-20210914	1.18.20	19.03.13-ce-1	4.14.243	v20210914	1.4.6-2.amzn2	460.73.01

AMI version	kubebundle version	Docker version	Kernel version	Packer version	Containerd version	NVIDIA driver version
1.18.20-20210830	1.18.20	19.03.13-ce-1	4.14.243	v20210830	1.4.6-2.amzn2	460.73.01
1.18.20-20210826	1.18.20	19.03.13-ce-1	4.14.238	v20210826	1.4.6-2.amzn2	460.73.01
1.18.20-20210813	1.18.20	19.03.13-ce-1	4.14.238	v20210813	1.4.6-2.amzn2	460.73.01
1.18.9-20210722	1.18.9	19.03.13-ce-1	4.14.238	v20210722	1.4.6-2.amzn2	460.73.01
1.18.9-20210720	1.18.9	19.03.13-ce-1	4.14.238	v20210720	1.4.6-2.amzn2	460.73.01
1.18.9-20210716	1.18.9	19.03.13-ce-1	4.14.238	v20210716	1.4.6-2.amzn2	460.73.01
1.18.9-20210628	1.18.9	19.03.13-ce-1	4.14.238	v20210628	1.4.6-2.amzn2	460.73.01
1.18.9-20210621	1.18.9	19.03.13-ce-1	4.14.238	v20210621	1.4.6-2.amzn2	460.73.01
1.18.9-20210526	1.18.9	19.03.13-ce-1	4.14.238	v20210526	1.4.6-2.amzn2	460.73.01
1.18.9-20210519	1.18.9	19.03.13-ce-1	4.14.238	v20210519	1.4.6-2.amzn2	460.73.01
1.18.9-20210518	1.18.9	19.03.13-ce-1	4.14.238	v20210518	1.4.6-2.amzn2	460.73.01
1.18.9-20210512	1.18.9	19.03.13-ce-1	4.14.238	v20210512	1.4.6-2.amzn2	460.73.01
1.18.9-20210504	1.18.9	19.03.13-ce-1	4.14.238	v20210504	1.4.6-2.amzn2	460.73.01
1.18.9-20210501	1.18.9	19.03.13-ce-1	4.14.238	v20210501	1.4.6-2.amzn2	460.32.03
1.18.9-20210414	1.18.9	19.03.13-ce-1	4.14.238	v20210414	1.4.6-2.amzn2	460.32.03
1.18.9-20210329	1.18.9	19.03.13-ce-1	4.14.238	v20210329	1.4.6-2.amzn2	460.32.03
1.18.9-20210322	1.18.9	19.03.13-ce-1	4.14.238	v20210322	1.4.6-2.amzn2	460.32.03
1.18.9-20210310	1.18.9	19.03.13-ce-1	4.14.238	v20210310	1.4.6-2.amzn2	460.27.04
1.18.9-20210302	1.18.9	19.03.13-ce-1	4.14.238	v20210302	1.4.6-2.amzn2	450.51.06

AMI version	kubelet version	Docker version	Kernel version	Packer version	Containerd version	NVIDIA driver version
1.18.9-20210208	1.18.9	19.03.6-ce-4	4.14.238	v20210208	1.4.6-2.amzn2	450.51.06
1.18.9-20210125	1.18.9	19.03.6-ce-4	4.14.238	v20210125	1.4.6-2.amzn2	450.51.06
1.18.9-20210112	1.18.9	19.03.6-ce-4	4.14.238	v20210112	1.4.6-2.amzn2	450.51.06
1.18.9-20201211	1.18.9	19.03.6-ce-4	4.14.238	v20201211	1.4.6-2.amzn2	450.51.06
1.18.9-20201126	1.18.9	19.03.6-ce-4	4.14.238	v20201126	1.4.6-2.amzn2	450.51.06
1.18.9-20201117	1.18.9	19.03.6-ce-4	4.14.238	v20201117	1.4.6-2.amzn2	450.51.06
1.18.9-20201112	1.18.9	19.03.6-ce-4	4.14.238	v20201112	1.4.6-2.amzn2	450.51.06
1.18.8-20201007	1.18.8	19.03.6-ce-4	4.14.238	v20201007	1.4.6-2.amzn2	418.87.00

Kubernetes version 1.17

Kubernetes version 1.17

AMI version	kubelet version	Docker version	Kernel version	Packer version	Containerd version	NVIDIA driver version
1.17.17-20211117	1.17.17	20.10.7-5.amzn2	4.14.252	v20211117	1.4.6-7.amzn2	470.57.02
1.17.17-20211109	1.17.17	20.10.7-3.amzn2	4.14.252	v20211109	1.4.6-3.amzn2	470.57.02
1.17.17-20211013	1.17.17	20.10.7-3.amzn2	4.14.248	v20211013	1.4.6-3.amzn2	460.73.01
1.17.17-20211008	1.17.17	20.10.7-3.amzn2	4.14.248	v20211008	1.4.6-3.amzn2	460.73.01
1.17.17-20211004	1.17.17	20.10.7-3.amzn2	4.14.246	v20211004	1.4.6-3.amzn2	460.73.01
1.17.17-20211003	1.17.17	20.10.7-3.amzn2	4.14.246	v20211003	1.4.6-3.amzn2	460.73.01
1.17.17-20210914	1.17.17	19.03.13-ce-1	4.14.243	v20210914	1.4.6-2.amzn2	460.73.01
1.17.17-20210830	1.17.17	19.03.13-ce-1	4.14.243	v20210830	1.4.6-2.amzn2	460.73.01
1.17.12-20210826	1.17.12	19.03.13-ce-1	4.14.238	v20210826	1.4.6-2.amzn2	460.73.01
1.17.12-20210813	1.17.12	19.03.13-ce-1	4.14.238	v20210813	1.4.6-2.amzn2	460.73.01

AMI version	kubebundle version	Docker version	Kernel version	Packer version	Containerd version	NVIDIA driver version
1.17.12-20210722	1.17.12	19.03.13-ce-1	4.14.238	v20210722	1.4.6-2.amzn2	460.73.01
1.17.12-20210720	1.17.12	19.03.13-ce-1	4.14.238	v20210720	1.4.6-2.amzn2	460.73.01
1.17.12-20210716	1.17.12	19.03.13-ce-1	4.14.238	v20210716	1.4.6-2.amzn2	460.73.01
1.17.12-20210628	1.17.12	19.03.13-ce-1	4.14.238	v20210628	1.4.6-2.amzn2	460.73.01
1.17.12-20210621	1.17.12	19.03.13-ce-1	4.14.238	v20210621	1.4.6-2.amzn2	460.73.01
1.17.12-20210526	1.17.12	19.03.13-ce-1	4.14.238	v20210526	1.4.6-2.amzn2	460.73.01
1.17.12-20210519	1.17.12	19.03.13-ce-1	4.14.238	v20210519	1.4.6-2.amzn2	460.73.01
1.17.12-20210518	1.17.12	19.03.13-ce-1	4.14.238	v20210518	1.4.6-2.amzn2	460.73.01
1.17.12-20210512	1.17.12	19.03.13-ce-1	4.14.238	v20210512	1.4.6-2.amzn2	460.73.01
1.17.12-20210504	1.17.12	19.03.13-ce-1	4.14.238	v20210504	1.4.6-2.amzn2	460.73.01
1.17.12-20210501	1.17.12	19.03.13-ce-1	4.14.238	v20210501	1.4.6-2.amzn2	460.32.03
1.17.12-20210414	1.17.12	19.03.13-ce-1	4.14.238	v20210414	1.4.6-2.amzn2	460.32.03
1.17.12-20210329	1.17.12	19.03.13-ce-1	4.14.238	v20210329	1.4.6-2.amzn2	460.32.03
1.17.12-20210322	1.17.12	19.03.13-ce-1	4.14.238	v20210322	1.4.6-2.amzn2	460.27.04
1.17.12-20210310	1.17.12	19.03.13-ce-1	4.14.238	v20210310	1.4.6-2.amzn2	460.27.04
1.17.12-20210302	1.17.12	19.03.13-ce-1	4.14.238	v20210302	1.4.6-2.amzn2	450.51.06
1.17.12-20210208	1.17.12	19.03.6-ce-4	4.14.238	v20210208	1.4.6-2.amzn2	450.51.06
1.17.12-20210125	1.17.12	19.03.6-ce-4	4.14.238	v20210125	1.4.6-2.amzn2	450.51.06
1.17.12-20210112	1.17.12	19.03.6-ce-4	4.14.238	v20210112	1.4.6-2.amzn2	450.51.06

AMI version	kubelet version	Docker version	Kernel version	Packer version	Containerd version	NVIDIA driver version
1.17.12-20201211	1.17.12	19.03.6-ce-4	4.14.238	v20201211	1.4.6-2.amzn2	450.51.06
1.17.12-20201126	1.17.12	19.03.6-ce-4	4.14.238	v20201126	1.4.6-2.amzn2	450.51.06
1.17.12-20201117	1.17.12	19.03.6-ce-4	4.14.238	v20201117	1.4.6-2.amzn2	450.51.06
1.17.12-20201112	1.17.12	19.03.6-ce-4	4.14.238	v20201112	1.4.6-2.amzn2	450.51.06
1.17.11-20201007	1.17.11	19.03.6-ce-4	4.14.238	v20201007	1.4.6-2.amzn2	418.87.00
1.17.11-20201002	1.17.11	19.03.6-ce-4	4.14.238	v20201002	1.4.6-2.amzn2	418.87.00
1.17.11-20200921	1.17.11	19.03.6-ce-4	4.14.238	v20200921	1.4.6-2.amzn2	418.87.00
1.17.9-20200904	1.17.9	19.03.6-ce-4	4.14.238	v20200904	1.4.6-2.amzn2	418.87.00
1.17.9-20200814	1.17.9	19.03.6-ce-4	4.14.238	v20200814	1.4.6-2.amzn2	418.87.00
1.17.9-20200723	1.17.9	19.03.6-ce	4.14.238	v20200723	1.4.6-2.amzn2	418.87.00
1.17.7-20200710	1.17.7	19.03.6-ce	4.14.238	v20200710	1.4.6-2.amzn2	418.87.00
1.17.7-20200709	1.17.7	19.03.6-ce	4.14.238	v20200709	1.4.6-2.amzn2	418.87.00

Retrieving Amazon EKS optimized Amazon Linux AMI IDs

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

You can retrieve the AMI ID with the AWS CLI or the AWS Management Console.

- **AWS CLI** – You can retrieve the image ID of the latest recommended Amazon EKS optimized Amazon Linux AMI with the following command by using the sub-parameter `image_id`. Replace `1.22` with a [supported version \(p. 76\)](#). Replace `region-code` with the AWS Region that your cluster is in.. Replace `amazon-linux-2` with `amazon-linux-2-gpu` to see the accelerated AMI ID and `amazon-linux-2-arm64` to see the Arm ID.

```
aws ssm get-parameter --name /aws/service/eks/optimized-ami/1.22/amazon-linux-2/recommended/image_id --region region-code --query "Parameter.Value" --output text
```

Example output:

```
ami-1234567890abcdef0
```

- **AWS Management Console** – You can query for the recommended Amazon EKS optimized AMI ID using a URL. The URL opens the Amazon EC2 Systems Manager console with the value of the ID for the parameter. In the following URL, replace `1.22` with a [supported version \(p. 76\)](#) and `region-code` with an [Amazon EKS supported Region](#) for which you want the AMI ID. Replace `amazon-linux-2` with `amazon-linux-2-gpu` to see the accelerated AMI ID and `amazon-linux-2-arm64` to see the Arm ID.

```
https://console.aws.amazon.com/systems-manager/parameters/aws/service/eks/optimized-ami/1.22/amazon-linux-2/recommended/image_id/description?region=region-code
```

Amazon EKS optimized Amazon Linux AMI build script

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

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

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

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

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

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

Amazon EKS optimized Amazon Linux 2 contains an optional bootstrap flag to enable the `containerd` runtime. When bootstrapped in Amazon EKS optimized accelerated Amazon Linux AMIs for v1.21, [AWSInferentia](#) workloads are not supported.

Configuring VT1 for your custom Amazon Linux AMI

Custom Amazon Linux AMIs in Amazon EKS can support the VT1 video transcoding instance family for Amazon Linux 2, Ubuntu 18, and Ubuntu 20. VT1 supports the Xilinx U30 media transcoding cards with accelerated H.264/AVC and H.265/HEVC codecs. To get the benefit of these accelerated instances, you must follow these steps:

1. Create and launch a base AMI from Amazon Linux 2, Ubuntu 18, or Ubuntu 20.
2. After the based AMI is launched, Install the [XRT driver](#) and runtime on the node.
3. [Create a cluster](#).
4. Install the Kubernetes [FPGA plugin](#) on your cluster.

```
kubectl apply -f fpga-device-plugin.yml
```

The plugin will now advertise Xilinx U30 devices per node on your Amazon EKS cluster. You can use the FFMPEG docker image to run example video transcoding workloads on your Amazon EKS cluster.

Configuring DL1 for your custom Amazon Linux 2 AMI

Custom Amazon Linux 2 AMIs in Amazon EKS can support deep learning workloads at scale through additional configuration and Kubernetes add-ons. This document describes the components required to set up a generic Kubernetes solution for an on-premise setup or as a baseline in a larger cloud configuration. To support this function, you will have to perform the following steps in your custom environment:

- SynapseAI® Software drivers loaded on the system – These are included in the [AMIs available on Github](#).

The Habana device plugin -- A Daemonset that allows you to automatically enable the registration of Habana devices in your Kubernetes cluster and track device health.

- Kubernetes version between 1.19 to 1.21 (confirmed through testing).
- Helm 3.x
- [Helm chart to install MPI Operator](#).
- MPI Operator

1. Create and launch a base AMI from Amazon Linux 2, Ubuntu 18, or Ubuntu 20.
2. Follow [these instructions](#) to set up the environment for DL1.

Amazon EKS optimized Ubuntu Linux AMIs

Canonical has partnered with Amazon EKS to create node AMIs that you can use in your clusters.

[Canonical](#) delivers a built-for-purpose Kubernetes Node OS image. This minimized Ubuntu image is optimized for Amazon EKS and includes the custom AWS kernel that is jointly developed with AWS. For more information, see [Ubuntu and Amazon Elastic Kubernetes Service](#) and [Optimized support for Amazon EKS on Ubuntu 18.04](#).

Amazon EKS optimized Bottlerocket AMIs

The Amazon EKS optimized Bottlerocket AMI is built on top of [Bottlerocket](#). Bottlerocket is a Linux based open-source operating system that is purpose built by AWS for running containers on virtual machines or bare metal hosts. It's secure and only includes the bare minimum packages required to run containers. This reduces the attack surface and impact of vulnerabilities. It also requires less effort to meet node compliance requirements. For more information about Bottlerocket, see the [documentation](#) and [releases on GitHub](#).

The AMI is configured to work with Amazon EKS and it includes containerd and kubelet .

The following points are some considerations when using Bottlerocket for your AMI type:

- You can deploy to Amazon EC2 instances with x86 or Arm processors, but not to instances that have Inferentia chips.
- You can't deploy to the following regions: China (Beijing) (cn-north-1) or China (Ningxia) (cn-northwest-1).
- There is no AWS CloudFormation template to deploy nodes with.
- Bottlerocket images don't come with an SSH server or a shell. You can use out-of-band access methods to allow SSH enabling the admin container and to pass some bootstrapping configuration steps with user data. For more information, see these sections in the [bottlerocket README.md](#) on GitHub:
 - [Exploration](#)

- Admin container
- Kubernetes settings
- By default, Bottlerocket has a [control container](#) that's enabled. This container runs the [AWS Systems Manager agent](#) that you can use to run commands or start shell sessions on Amazon EC2 Bottlerocket instances. For more information, see [Setting up Session Manager](#) in the [AWS Systems Manager User Guide](#).
- Bottlerocket also has an [admin container](#) that's enabled if an SSH key is given when creating the node group. We recommend using the admin container only for development and testing scenarios. We don't recommend using it for production environments.

Select a link in one of the following tables to view the latest Amazon EKS optimized Bottlerocket AMI ID for a region and Kubernetes version. You can also retrieve the IDs with an AWS Systems Manager parameter using different tools. For more information, see [Retrieving Amazon EKS optimized Bottlerocket AMI IDs \(p. 210\)](#).

1.22

Kubernetes version 1.22

AWS Region	x86 (Standard)	x86 (NVIDIA)	Arm (Standard)	Arm (NVIDIA)
US East (Ohio) (us-east-2)	View AMI ID			
US East (N. Virginia) (us-east-1)	View AMI ID			
US West (Oregon) (us-west-2)	View AMI ID			
US West (N. California) (us-west-1)	View AMI ID			
Africa (Cape Town) (af-south-1)	View AMI ID			
Asia Pacific (Hong Kong) (ap-east-1)	View AMI ID			
Asia Pacific (Mumbai) (ap-south-1)	View AMI ID			
Asia Pacific (Tokyo) (ap-northeast-1)	View AMI ID			
Asia Pacific (Seoul) (ap-northeast-2)	View AMI ID			
Asia Pacific (Singapore) (ap-southeast-1)	View AMI ID			
Asia Pacific (Sydney) (ap-southeast-2)	View AMI ID			
Asia Pacific (Jakarta) (ap-southeast-3)	View AMI ID			
Canada (Central) (ca-central-1)	View AMI ID			
Europe (Frankfurt) (eu-central-1)	View AMI ID			
Europe (Ireland) (eu-west-1)	View AMI ID			
Europe (London) (eu-west-2)	View AMI ID			

AWS Region	x86 (Standard)	x86 (NVIDIA)	Arm (Standard)	Arm (NVIDIA)
Europe (Milan) (eu-south-1)	View AMI ID			
Europe (Paris) (eu-west-3)	View AMI ID			
Europe (Stockholm) (eu-north-1)	View AMI ID			
Middle East (Bahrain) (me-south-1)	View AMI ID			
South America (São Paulo) (sa-east-1)	View AMI ID			
AWS GovCloud (US-East) (us-gov-east-1)	View AMI ID			
AWS GovCloud (US-West) (us-gov-west-1)	View AMI ID			

1.21

Kubernetes version 1.21

AWS Region	x86	Arm
US East (Ohio) (us-east-2)	View AMI ID	View AMI ID
US East (N. Virginia) (us-east-1)	View AMI ID	View AMI ID
US West (Oregon) (us-west-2)	View AMI ID	View AMI ID
US West (N. California) (us-west-1)	View AMI ID	View AMI ID
Africa (Cape Town) (af-south-1)	View AMI ID	View AMI ID
Asia Pacific (Hong Kong) (ap-east-1)	View AMI ID	View AMI ID
Asia Pacific (Mumbai) (ap-south-1)	View AMI ID	View AMI ID
Asia Pacific (Tokyo) (ap-northeast-1)	View AMI ID	View AMI ID
Asia Pacific (Seoul) (ap-northeast-2)	View AMI ID	View AMI ID
Asia Pacific (Singapore) (ap-southeast-1)	View AMI ID	View AMI ID
Asia Pacific (Sydney) (ap-southeast-2)	View AMI ID	View AMI ID
Asia Pacific (Jakarta) (ap-southeast-3)	View AMI ID	View AMI ID
Canada (Central) (ca-central-1)	View AMI ID	View AMI ID
Europe (Frankfurt) (eu-central-1)	View AMI ID	View AMI ID
Europe (Ireland) (eu-west-1)	View AMI ID	View AMI ID
Europe (London) (eu-west-2)	View AMI ID	View AMI ID
Europe (Milan) (eu-south-1)	View AMI ID	View AMI ID
Europe (Paris) (eu-west-3)	View AMI ID	View AMI ID

AWS Region	x86	Arm
Europe (Stockholm) (eu-north-1)	View AMI ID	View AMI ID
Middle East (Bahrain) (me-south-1)	View AMI ID	View AMI ID
South America (São Paulo) (sa-east-1)	View AMI ID	View AMI ID
AWS GovCloud (US-East) (us-gov-east-1)	View AMI ID	View AMI ID
AWS GovCloud (US-West) (us-gov-west-1)	View AMI ID	View AMI ID

1.20

Kubernetes version 1.20

AWS Region	x86	Arm
US East (Ohio) (us-east-2)	View AMI ID	View AMI ID
US East (N. Virginia) (us-east-1)	View AMI ID	View AMI ID
US West (Oregon) (us-west-2)	View AMI ID	View AMI ID
US West (N. California) (us-west-1)	View AMI ID	View AMI ID
Africa (Cape Town) (af-south-1)	View AMI ID	View AMI ID
Asia Pacific (Hong Kong) (ap-east-1)	View AMI ID	View AMI ID
Asia Pacific (Mumbai) (ap-south-1)	View AMI ID	View AMI ID
Asia Pacific (Tokyo) (ap-northeast-1)	View AMI ID	View AMI ID
Asia Pacific (Seoul) (ap-northeast-2)	View AMI ID	View AMI ID
Asia Pacific (Singapore) (ap-southeast-1)	View AMI ID	View AMI ID
Asia Pacific (Sydney) (ap-southeast-2)	View AMI ID	View AMI ID
Asia Pacific (Jakarta) (ap-southeast-3)	View AMI ID	View AMI ID
Canada (Central) (ca-central-1)	View AMI ID	View AMI ID
Europe (Frankfurt) (eu-central-1)	View AMI ID	View AMI ID
Europe (Ireland) (eu-west-1)	View AMI ID	View AMI ID
Europe (London) (eu-west-2)	View AMI ID	View AMI ID
Europe (Milan) (eu-south-1)	View AMI ID	View AMI ID
Europe (Paris) (eu-west-3)	View AMI ID	View AMI ID
Europe (Stockholm) (eu-north-1)	View AMI ID	View AMI ID
Middle East (Bahrain) (me-south-1)	View AMI ID	View AMI ID
South America (São Paulo) (sa-east-1)	View AMI ID	View AMI ID
AWS GovCloud (US-East) (us-gov-east-1)	View AMI ID	View AMI ID

AWS Region	x86	Arm
AWS GovCloud (US-West) (us-gov-west-1)	View AMI ID	View AMI ID

1.19

Kubernetes version 1.19

AWS Region	x86	Arm
US East (Ohio) (us-east-2)	View AMI ID	View AMI ID
US East (N. Virginia) (us-east-1)	View AMI ID	View AMI ID
US West (Oregon) (us-west-2)	View AMI ID	View AMI ID
US West (N. California) (us-west-1)	View AMI ID	View AMI ID
Africa (Cape Town) (af-south-1)	View AMI ID	View AMI ID
Asia Pacific (Hong Kong) (ap-east-1)	View AMI ID	View AMI ID
Asia Pacific (Mumbai) (ap-south-1)	View AMI ID	View AMI ID
Asia Pacific (Tokyo) (ap-northeast-1)	View AMI ID	View AMI ID
Asia Pacific (Seoul) (ap-northeast-2)	View AMI ID	View AMI ID
Asia Pacific (Singapore) (ap-southeast-1)	View AMI ID	View AMI ID
Asia Pacific (Sydney) (ap-southeast-2)	View AMI ID	View AMI ID
Asia Pacific (Jakarta) (ap-southeast-3)	View AMI ID	View AMI ID
Canada (Central) (ca-central-1)	View AMI ID	View AMI ID
Europe (Frankfurt) (eu-central-1)	View AMI ID	View AMI ID
Europe (Ireland) (eu-west-1)	View AMI ID	View AMI ID
Europe (London) (eu-west-2)	View AMI ID	View AMI ID
Europe (Milan) (eu-south-1)	View AMI ID	View AMI ID
Europe (Paris) (eu-west-3)	View AMI ID	View AMI ID
Europe (Stockholm) (eu-north-1)	View AMI ID	View AMI ID
Middle East (Bahrain) (me-south-1)	View AMI ID	View AMI ID
South America (São Paulo) (sa-east-1)	View AMI ID	View AMI ID
AWS GovCloud (US-East) (us-gov-east-1)	View AMI ID	View AMI ID
AWS GovCloud (US-West) (us-gov-west-1)	View AMI ID	View AMI ID

1.18

Kubernetes version 1.18

AWS Region	x86	Arm
US East (Ohio) (us-east-2)	View AMI ID	View AMI ID
US East (N. Virginia) (us-east-1)	View AMI ID	View AMI ID
US West (Oregon) (us-west-2)	View AMI ID	View AMI ID
US West (N. California) (us-west-1)	View AMI ID	View AMI ID
Africa (Cape Town) (af-south-1)	View AMI ID	View AMI ID
Asia Pacific (Hong Kong) (ap-east-1)	View AMI ID	View AMI ID
Asia Pacific (Mumbai) (ap-south-1)	View AMI ID	View AMI ID
Asia Pacific (Tokyo) (ap-northeast-1)	View AMI ID	View AMI ID
Asia Pacific (Seoul) (ap-northeast-2)	View AMI ID	View AMI ID
Asia Pacific (Singapore) (ap-southeast-1)	View AMI ID	View AMI ID
Asia Pacific (Sydney) (ap-southeast-2)	View AMI ID	View AMI ID
Asia Pacific (Jakarta) (ap-southeast-3)	View AMI ID	View AMI ID
Canada (Central) (ca-central-1)	View AMI ID	View AMI ID
Europe (Frankfurt) (eu-central-1)	View AMI ID	View AMI ID
Europe (Ireland) (eu-west-1)	View AMI ID	View AMI ID
Europe (London) (eu-west-2)	View AMI ID	View AMI ID
Europe (Milan) (eu-south-1)	View AMI ID	View AMI ID
Europe (Paris) (eu-west-3)	View AMI ID	View AMI ID
Europe (Stockholm) (eu-north-1)	View AMI ID	View AMI ID
Middle East (Bahrain) (me-south-1)	View AMI ID	View AMI ID
South America (São Paulo) (sa-east-1)	View AMI ID	View AMI ID
AWS GovCloud (US-East) (us-gov-east-1)	View AMI ID	View AMI ID
AWS GovCloud (US-West) (us-gov-west-1)	View AMI ID	View AMI ID

Retrieving Amazon EKS optimized Bottlerocket AMI IDs

You can retrieve the Amazon Machine Image (AMI) ID for Amazon EKS optimized AMIs by querying the AWS Systems Manager Parameter Store API. Using this parameter, you don't need to manually look up Amazon EKS optimized AMI IDs. For more information about the Systems Manager Parameter Store API, see [GetParameter](#). Your user account must have the `ssm:GetParameter` IAM permission to retrieve the Amazon EKS optimized AMI metadata.

You can retrieve the AMI ID with the AWS CLI or the AWS Management Console.

- **AWS CLI** – You can retrieve the image ID of the latest recommended Amazon EKS optimized Bottlerocket AMI with the following AWS CLI command by using the sub-parameter `image_id`. Replace `1.22` with a [supported version \(p. 76\)](#) and `region-code` with an [Amazon EKS supported Region](#) for which you want the AMI ID.

```
aws ssm get-parameter --name /aws/service/bottlerocket/aws-k8s-1.22/x86_64/latest/  
image_id --region region-code --query "Parameter.Value" --output text
```

Example output:

```
ami-1234567890abcdef0
```

- **AWS Management Console** – You can query for the recommended Amazon EKS optimized AMI ID using a URL in the AWS Management Console. The URL opens the Amazon EC2 Systems Manager console with the value of the ID for the parameter. In the following URL, replace `1.22` with a [supported version \(p. 76\)](#) and `region-code` with an [Amazon EKS supported Region](#) for which you want the AMI ID.

```
https://console.aws.amazon.com/systems-manager/parameters/aws/service/bottlerocket/aws-k8s-1.22/x86\_64/latest/image\_id/description?region=region-code
```

Amazon EKS optimized Windows AMIs

Windows Amazon EKS optimized AMIs are built on top of Windows Server 2019 and Windows Server 20H2. They are configured to serve as the base image for Amazon EKS nodes. By default, the AMIs include the following components:

- kubelet
- kube-proxy
- AWS IAM Authenticator
- CSI proxy
- Docker
- Containerd (Amazon EKS v1.21 or greater)

Note

You can track security or privacy events for Windows Server with the [Microsoft security update guide](#).

The AMI IDs for the latest Amazon EKS optimized AMI are in the following tables. You can also retrieve the IDs with an AWS Systems Manager parameter using different tools. For more information, see [Retrieving Amazon EKS optimized Windows AMI IDs \(p. 226\)](#).

Windows Server 2019 is a Long-Term Servicing Channel (LTSC) release, whereas Versions 20H2 is a Semi-Annual Channel (SAC) release. For more information, see [Windows Server servicing channels](#) in the Microsoft documentation. Windows Server 20H2 support was added to Kubernetes in version 1.21. For more information about Windows OS version support, see [Intro to Windows support in Kubernetes](#).

1.22

Kubernetes version 1.22

AWS Region	Windows Server 2019 Core	Windows Server 2019 Full	Windows Server 20H2 Core
US East (Ohio) (us-east-2)	View AMI ID	View AMI ID	View AMI ID
US East (N. Virginia) (us-east-1)	View AMI ID	View AMI ID	View AMI ID
US West (Oregon) (us-west-2)	View AMI ID	View AMI ID	View AMI ID
US West (N. California) (us-west-1)	View AMI ID	View AMI ID	View AMI ID
Africa (Cape Town) (af-south-1)	View AMI ID	View AMI ID	View AMI ID
Asia Pacific (Hong Kong) (ap-east-1)	View AMI ID	View AMI ID	View AMI ID
Asia Pacific (Mumbai) (ap-south-1)	View AMI ID	View AMI ID	View AMI ID
Asia Pacific (Tokyo) (ap-northeast-1)	View AMI ID	View AMI ID	View AMI ID
Asia Pacific (Seoul) (ap-northeast-2)	View AMI ID	View AMI ID	View AMI ID
Asia Pacific (Osaka) (ap-northeast-3)	View AMI ID	View AMI ID	View AMI ID
Asia Pacific (Singapore) (ap-southeast-1)	View AMI ID	View AMI ID	View AMI ID
Asia Pacific (Sydney) (ap-southeast-2)	View AMI ID	View AMI ID	View AMI ID
Asia Pacific (Jakarta) (ap-southeast-3)	View AMI ID	View AMI ID	View AMI ID
Canada (Central) (ca-central-1)	View AMI ID	View AMI ID	View AMI ID
China (Beijing) (cn-north-1)	View AMI ID	View AMI ID	View AMI ID
China (Ningxia) (cn-northwest-1)	View AMI ID	View AMI ID	View AMI ID
Europe (Frankfurt) (eu-central-1)	View AMI ID	View AMI ID	View AMI ID
Europe (Ireland) (eu-west-1)	View AMI ID	View AMI ID	View AMI ID

AWS Region	Windows Server 2019 Core	Windows Server 2019 Full	Windows Server 20H2 Core
Europe (London) (eu-west-2)	View AMI ID	View AMI ID	View AMI ID
Europe (Milan) (eu-south-1)	View AMI ID	View AMI ID	View AMI ID
Europe (Paris) (eu-west-3)	View AMI ID	View AMI ID	View AMI ID
Europe (Stockholm) (eu-north-1)	View AMI ID	View AMI ID	View AMI ID
Middle East (Bahrain) (me-south-1)	View AMI ID	View AMI ID	View AMI ID
South America (São Paulo) (sa-east-1)	View AMI ID	View AMI ID	View AMI ID
AWS GovCloud (US-East) (us-gov-east-1)	View AMI ID	View AMI ID	View AMI ID
AWS GovCloud (US-West) (us-gov-west-1)	View AMI ID	View AMI ID	View AMI ID

1.21

Kubernetes version 1.21

AWS Region	Windows Server 2019 Core	Windows Server 2019 Full	Windows Server 20H2 Core
US East (Ohio) (us-east-2)	View AMI ID	View AMI ID	View AMI ID
US East (N. Virginia) (us-east-1)	View AMI ID	View AMI ID	View AMI ID
US West (Oregon) (us-west-2)	View AMI ID	View AMI ID	View AMI ID
US West (N. California) (us-west-1)	View AMI ID	View AMI ID	View AMI ID
Africa (Cape Town) (af-south-1)	View AMI ID	View AMI ID	View AMI ID
Asia Pacific (Hong Kong) (ap-east-1)	View AMI ID	View AMI ID	View AMI ID
Asia Pacific (Mumbai) (ap-south-1)	View AMI ID	View AMI ID	View AMI ID
Asia Pacific (Tokyo) (ap-northeast-1)	View AMI ID	View AMI ID	View AMI ID

AWS Region	Windows Server 2019 Core	Windows Server 2019 Full	Windows Server 20H2 Core
Asia Pacific (Seoul) (ap-northeast-2)	View AMI ID	View AMI ID	View AMI ID
Asia Pacific (Osaka) (ap-northeast-3)	View AMI ID	View AMI ID	View AMI ID
Asia Pacific (Singapore) (ap-southeast-1)	View AMI ID	View AMI ID	View AMI ID
Asia Pacific (Sydney) (ap-southeast-2)	View AMI ID	View AMI ID	View AMI ID
Asia Pacific (Jakarta) (ap-southeast-3)	View AMI ID	View AMI ID	View AMI ID
Canada (Central) (ca-central-1)	View AMI ID	View AMI ID	View AMI ID
China (Beijing) (cn-north-1)	View AMI ID	View AMI ID	View AMI ID
China (Ningxia) (cn-northwest-1)	View AMI ID	View AMI ID	View AMI ID
Europe (Frankfurt) (eu-central-1)	View AMI ID	View AMI ID	View AMI ID
Europe (Ireland) (eu-west-1)	View AMI ID	View AMI ID	View AMI ID
Europe (London) (eu-west-2)	View AMI ID	View AMI ID	View AMI ID
Europe (Milan) (eu-south-1)	View AMI ID	View AMI ID	View AMI ID
Europe (Paris) (eu-west-3)	View AMI ID	View AMI ID	View AMI ID
Europe (Stockholm) (eu-north-1)	View AMI ID	View AMI ID	View AMI ID
Middle East (Bahrain) (me-south-1)	View AMI ID	View AMI ID	View AMI ID
South America (São Paulo) (sa-east-1)	View AMI ID	View AMI ID	View AMI ID
AWS GovCloud (US-East) (us-gov-east-1)	View AMI ID	View AMI ID	View AMI ID
AWS GovCloud (US-West) (us-gov-west-1)	View AMI ID	View AMI ID	View AMI ID

1.20

Kubernetes version 1.20

AWS Region	Windows Server 2019 Core	Windows Server 2019 Full
US East (Ohio) (us-east-2)	View AMI ID	View AMI ID
US East (N. Virginia) (us-east-1)	View AMI ID	View AMI ID
US West (Oregon) (us-west-2)	View AMI ID	View AMI ID
US West (N. California) (us-west-1)	View AMI ID	View AMI ID
Africa (Cape Town) (af-south-1)	View AMI ID	View AMI ID
Asia Pacific (Hong Kong) (ap-east-1)	View AMI ID	View AMI ID
Asia Pacific (Mumbai) (ap-south-1)	View AMI ID	View AMI ID
Asia Pacific (Tokyo) (ap-northeast-1)	View AMI ID	View AMI ID
Asia Pacific (Seoul) (ap-northeast-2)	View AMI ID	View AMI ID
Asia Pacific (Osaka) (ap-northeast-3)	View AMI ID	View AMI ID
Asia Pacific (Singapore) (ap-southeast-1)	View AMI ID	View AMI ID
Asia Pacific (Sydney) (ap-southeast-2)	View AMI ID	View AMI ID
Asia Pacific (Jakarta) (ap-southeast-3)	View AMI ID	View AMI ID
Canada (Central) (ca-central-1)	View AMI ID	View AMI ID
China (Beijing) (cn-north-1)	View AMI ID	View AMI ID
China (Ningxia) (cn-northwest-1)	View AMI ID	View AMI ID
Europe (Frankfurt) (eu-central-1)	View AMI ID	View AMI ID
Europe (Ireland) (eu-west-1)	View AMI ID	View AMI ID
Europe (London) (eu-west-2)	View AMI ID	View AMI ID
Europe (Milan) (eu-south-1)	View AMI ID	View AMI ID
Europe (Paris) (eu-west-3)	View AMI ID	View AMI ID

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

1.19

Kubernetes version 1.19

AWS Region	Windows Server 2019 Core	Windows Server 2019 Full
US East (Ohio) (us-east-2)	View AMI ID	View AMI ID
US East (N. Virginia) (us-east-1)	View AMI ID	View AMI ID
US West (Oregon) (us-west-2)	View AMI ID	View AMI ID
US West (N. California) (us-west-1)	View AMI ID	View AMI ID
Africa (Cape Town) (af-south-1)	View AMI ID	View AMI ID
Asia Pacific (Hong Kong) (ap-east-1)	View AMI ID	View AMI ID
Asia Pacific (Mumbai) (ap-south-1)	View AMI ID	View AMI ID
Asia Pacific (Tokyo) (ap-northeast-1)	View AMI ID	View AMI ID
Asia Pacific (Seoul) (ap-northeast-2)	View AMI ID	View AMI ID
Asia Pacific (Osaka) (ap-northeast-3)	View AMI ID	View AMI ID
Asia Pacific (Singapore) (ap-southeast-1)	View AMI ID	View AMI ID
Asia Pacific (Sydney) (ap-southeast-2)	View AMI ID	View AMI ID
Asia Pacific (Jakarta) (ap-southeast-3)	View AMI ID	View AMI ID

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

Amazon EKS Windows AMI release calendar

The following table lists the release and end of support dates for Windows versions on Amazon EKS. If an end date is blank, it's because the version is still supported.

Windows version	Amazon EKS release	Amazon EKS end of support
Windows Server 20H2 Core	8/12/2021	8/9/2022
Windows Server 2004 Core	8/19/2020	12/14/2021
Windows Server 2019 Core	10/7/2019	
Windows Server 2019 Full	10/7/2019	
Windows Server 1909 Core	10/7/2019	12/8/2020

Bootstrap script configuration parameters

When you create a Windows node, there's a script on the node that allows for configuring different parameters. Depending on your setup, this script can be found on the node at a location similar to:

C:\Program Files\Amazon\EKS\Start-EKSBootstrap.ps1. The script includes the following parameters:

- `-EKSClusterName` – Specifies the Amazon EKS cluster name for this worker node to join.
- `-KubeletExtraArgs` – Specifies extra arguments for `kubelet` (optional).
- `-KubeProxyExtraArgs` – Specifies extra arguments for `kube-proxy` (optional).
- `-APIServerEndpoint` – Specifies the Amazon EKS cluster API server endpoint (optional). Only valid when used with `-Base64ClusterCA`. Bypasses calling `Get-EKSCluster`.
- `-Base64ClusterCA` – Specifies the base64 encoded cluster CA content (optional). Only valid when used with `-APIServerEndpoint`. Bypasses calling `Get-EKSCluster`.
- `-DNSClusterIP` – Overrides the IP address to use for DNS queries within the cluster (optional). Defaults to 10.100.0.10 or 172.20.0.10 based on the IP address of the primary interface.
- `-ContainerRuntime` – Specifies the container runtime to be used on the node.

Enable the `containerd` runtime bootstrap flag

The Amazon EKS optimized Windows AMI contains an optional bootstrap flag to enable the `containerd` runtime. This feature gives you a clear path to migrate to `containerd`. Amazon EKS is ending support for Docker starting with the Kubernetes version 1.23 launch. For more information, see [Amazon EKS is ending support for Dockershim \(p. 175\)](#).

Until Kubernetes version 1.23, the supported values for the container runtime are `docker` and `containerd`, specified when launching the Windows nodes using either `eksctl` or the AWS Management Console.

- If the specified value is `docker`, then Docker is used as the runtime on the node.
- If the specified value is `containerd` and the Amazon EKS version is greater than 1.20, then `containerd` is selected as the runtime. If the Amazon EKS version is less than 1.21, then the bootstrap fails and nodes are unable to join the cluster.
- If any other value is specified, then the bootstrap fails and the node isn't able to join the cluster.
- If this flag itself isn't specified, then the default value of the container runtime is selected. For Amazon EKS version 1.21 and lower, this would be Docker.

When launching Windows nodes in your Amazon EKS cluster, follow the steps in [Launching self-managed Windows nodes \(p. 140\)](#). Windows self-managed nodes with the `containerd` runtime can be launched using `eksctl` or the AWS Management Console.

`eksctl`

To enable the `containerd` runtime with `eksctl`

For Windows self-managed nodes, the container runtime can be specified in the configuration while creating new node groups. You can use the following `test-windows-with-containerd.yaml` as reference.

Note

You must use `eksctl` version [0.93.0-rc.0](#) to use the `containerRuntime` setting in the configuration file.

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

metadata:
  name: windows-containerd-cluster
  region: us-west-2
```

```
version: '1.21'

nodeGroups:
  - name: windows-ng
    instanceType: m5.2xlarge
    amiFamily: WindowsServer2019FullContainer
    volumeSize: 100
    minSize: 2
    maxSize: 3
    containerRuntime: containerd
  - name: linux-ng
    amiFamily: AmazonLinux2
    minSize: 2
    maxSize: 3
```

The node groups can then be created using the following command.

```
eksctl create cluster -f test-windows-with-containerd.yaml
```

Alternatively, you can also specify the `EKS_CONTAINER_RUNTIME` environment variable as a pre-bootstrap command in the `eksctl` configuration file.

```
preBootstrapCommands:
  - Invoke-Expression -Command
    '[Environment]::SetEnvironmentVariable("EKS_CONTAINER_RUNTIME", "containerd",
[System.EnvironmentVariableTarget]::Machine)'
```

For more information, see [Creating a nodegroup from a config file, defining containerd runtime, and Config file schema](#) in the `eksctl` documentation.

AWS Management Console

To enable the `containerd` runtime with the AWS Management Console

In the AWS CloudFormation template, there's a parameter named `BootstrapArguments` which can be used to pass in additional arguments to the bootstrap script. A parameter named `ContainerRuntime` can be used to select a particular runtime on the node.

Specify the following in `BootstrapArguments` to enable the `containerd` runtime:

```
-ContainerRuntime containerd
```

Amazon EKS optimized Windows AMI versions

This topic lists versions of the Amazon EKS optimized Windows AMIs and their corresponding versions of `kubelet`, `Docker`, `containerd`, and `csi-proxy`.

The Amazon EKS optimized AMI metadata, including the AMI ID, for each variant can be retrieved programmatically. For more information, see [Retrieving Amazon EKS optimized Windows AMI IDs \(p. 226\)](#).

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

```
k8s_major_version.k8s_minor_version-release_date
```

Amazon EKS optimized Windows Server 20H2 Core AMI

The tables below list the current and previous versions of the Amazon EKS optimized Windows AMI.

Kubernetes version 1.22

Kubernetes version 1.22

AMI version	kubelet version	Docker version	containerd version	csi-proxy version
1.22-2022.04.14	1.22.6	20.10.9	1.6.2	1.1.1

Kubernetes version 1.21

Kubernetes version 1.21

AMI version	kubelet version	Docker version	containerd version	csi-proxy version
1.21.2022.04.14	1.21.5	20.10.9	1.62	1.1.1
1.21.2022.03.10	1.21.5	20.10.9	1.61	N/A
1.21.2022.02.23	1.21.5	20.10.9	N/A	N/A
1.21.2022.01.18	1.21.5	20.10.9	N/A	N/A
1.21.2021.12.21	1.21.5	20.10.8	N/A	N/A
1.21-2021.11.10	1.21.4	20.10.7	N/A	N/A
1.21-2021.10.14	1.21.4	20.10.7	N/A	N/A
1.21-2021.09.16	1.21.2	20.10.7	N/A	N/A
1.21-2021.08.12	1.21.2	20.10.6	N/A	N/A

Amazon EKS optimized Windows Server 2019 Core AMI

The tables below list the current and previous versions of the Amazon EKS optimized Windows AMI.

Kubernetes version 1.22

Kubernetes version 1.22

AMI version	kubelet version	Docker version	containerd version	csi-proxy version
1.22-2022.04.14	1.22.6	20.10.9	1.6.2	1.1.1

Kubernetes version 1.21

Kubernetes version 1.21

AMI version	kubelet version	Docker version	containerd version	csi-proxy version
1.21-2022.04.14	1.21.5	20.10.9	1.62	1.1.1
1.21.2021.03.10	1.21.5	20.10.9	1.61	N/A

AMI version	kubelet version	Docker version	containerd version	csi-proxy version
1.21.2022.02.23	1.21.5	20.10.9	N/A	N/A
1.21.2022.01.18	1.21.5	20.10.9	N/A	N/A
1.21.2021.12.21	1.21.5	20.10.8	N/A	N/A
1.21-2021.10.14	1.21.4	20.10.7	N/A	N/A
1.21-2021.11.10	1.21.4	20.10.7	N/A	N/A
1.21-2021.09.16	1.21.2	20.10.7	N/A	N/A
1.21-2021.08.12	1.21.2	20.10.6	N/A	N/A

Kubernetes version 1.20

Kubernetes version 1.20

AMI version	kubelet version	Docker version	csi-proxy version
1.20-2022.04.14	1.20.11	20.10.9	1.1.1
1.20-2022.03.10	1.20.11	20.10.9	N/A
1.20-2022.02.23	1.20.11	20.10.9	N/A
1.20-2022.01.8	1.20.11	20.10.9	N/A
1.20-2021.12.21	1.20.11	20.10.8	N/A
1.20-2021.11.10	1.20.10	20.10.7	N/A
1.20-2021.10.14	1.20.10	20.10.7	N/A
1.20-2021.09.16	1.20.7	20.10.7	N/A
1.20-2021.08.12	1.20.4	20.10.6	N/A
1.20-2021.07.14	1.20.4	20.10.6	N/A
1.20-2021.06.16	1.20.4	20.10.5	N/A
1.20-2021.05.18	1.20.4	20.10.4	N/A

Kubernetes version 1.19

Kubernetes version 1.19

AMI version	kubelet version	Docker version	csi-proxy version
1.19-2022.04.14	1.19.15	20.10.9	1.1.1
1.19-2022.03.10	1.19.15	20.10.9	N/A
1.19-2022.02.23	1.19.15	20.10.9	N/A
1.19-2022.01.18	1.19.15	20.10.9	N/A

AMI version	kubelet version	Docker version	csi-proxy version
1.19-2021.12.21	1.19.15	20.10.8	N/A
1.19-2021.11.10	1.19.14	20.10.7	N/A
1.19-2021.10.14	1.19.14	20.10.7	N/A
1.19-2021.09.16	1.19.13	20.10.7	N/A
1.19-2021.08.12	1.19.8	20.10.6	N/A
1.19-2021.07.14	1.19.8	20.10.6	N/A
1.19-2021.06.16	1.19.8	20.10.5	N/A
1.19-2021.05.18	1.19.8	20.10.4	N/A
1.19-2021.04.14	1.19.6	20.10.0	N/A
1.19-2021.03.10	1.19.6	19.03.14	N/A
1.19-2021.02.18	1.19.6	19.03.14	N/A

Kubernetes version 1.18

Kubernetes version 1.18

AMI version	kubelet version	Docker version	csi-proxy version
1.18-2022.03.10	1.18.20	20.10.9	1.1.1
1.18-2022.02.23	1.18.20	20.10.9	N/A
1.18-2022.01.18	1.18.20	20.10.9	N/A
1.18-2021.12.21	1.18.20	20.10.8	N/A
1.18-2021.11.10	1.18.20	20.10.7	N/A
1.18-2021.10.14	1.18.20	20.10.7	N/A
1.18-2021.09.16	1.18.20	20.10.7	N/A
1.18-2021.08.12	1.18.16	20.10.6	N/A
1.18-2021.07.14	1.18.16	20.10.6	N/A
1.18-2021.06.16	1.18.16	20.10.5	N/A
1.18-2021.05.18	1.18.16	20.10.4	N/A
1.18-2021.04.14	1.18.9	20.10.0	N/A
1.18-2021.03.10	1.18.9	19.03.14	N/A
1.18-2021.02.10	1.18.9	19.03.14	N/A
1.18-2021.01.13	1.18.9	19.03.14	N/A
1.18-2020.12.11	1.18.9	19.03.13	N/A

AMI version	kubelet version	Docker version	csi-proxy version
1.18-2020.11.12	1.18.9	18.09.7	N/A
1.18-2020.10.29	1.18.8	18.09.7	N/A
1.18-2020.10.08	1.18.8	18.09.7	N/A

Kubernetes version 1.17

The most recent version is the last version we're releasing for Amazon EKS 1.17 clusters. It will be available until the 1.17 end of support date. For more information, see [Amazon EKS Kubernetes release calendar \(p. 74\)](#). For newer AMI releases, update your cluster to a later Kubneretes version. For more information, see [Updating a cluster \(p. 34\)](#).

Kubernetes version 1.17

AMI version	kubelet version	Docker version
1.17-2021.08.12	1.17.17	20.10.6
1.17-2021.06.16	1.17.17	20.10.5
1.17-2021.05.18	1.17.17	20.10.4
1.17-2021.04.14	1.17.12	20.10.0
1.17-2021.03.10	1.17.12	19.03.14
1.17-2021.02.10	1.17.12	19.03.14
1.17-2021.01.13	1.17.12	19.03.14
1.17-2020.12.11	1.17.12	19.03.13
1.17-2020.11.12	1.17.12	18.09.7
1.17-2020.10.29	1.17.11	18.09.7
1.17-2020.09.09	1.17.11	18.09.7
1.17-2020.08.13	1.17.9	18.09.7

Amazon EKS optimized Windows Server 2019 Full AMI

The tables below list the current and previous versions of the Amazon EKS optimized Windows AMI.

Kubernetes version 1.22

Kubernetes version 1.22

AMI version	kubelet version	Docker version	containerd version
1.22-2022.04.14	1.22.6	20.10.9	1.6.2

Kubernetes version 1.21

Kubernetes version 1.21

AMI version	kubelet version	Docker version	containerd version	csi-proxy version
1.21.2022.04.14	1.21.5	20.10.9	1.6.2	1.1.1
1.21.2022.03.10	1.21.5	20.10.9	1.6.1	N/A
1.21.2022.02.23	1.21.5	20.10.9	N/A	N/A
1.21.2022.01.18	1.21.5	20.10.9	N/A	N/A
1.21.2021.12.21	1.21.5	20.10.8	N/A	N/A
1.21-2021.11.10	1.21.4	20.10.7	N/A	N/A
1.21-2021.10.14	1.21.4	20.10.7	N/A	N/A
1.21-2021.09.16	1.21.2	20.10.7	N/A	N/A
1.21-2021.08.12	1.21.2	20.10.6	N/A	N/A

Kubernetes version 1.20

Kubernetes version 1.20

AMI version	kubelet version	Docker version	csi-proxy version
1.20-2022.04.14	1.20.11	20.10.9	1.1.1
1.20-2022.03.10	1.20.11	20.10.9	N/A
1.20-2022.02.23	1.20.11	20.10.9	N/A
1.20-2022.01.8	1.20.11	20.10.9	N/A
1.20-2021.12.21	1.20.11	20.10.8	N/A
1.20-2021.11.10	1.20.10	20.10.7	N/A
1.20-2021.10.14	1.20.10	20.10.7	N/A
1.20-2021.09.16	1.20.7	20.10.7	N/A
1.20-2021.08.12	1.20.4	20.10.6	N/A
1.20-2021.07.14	1.20.4	20.10.6	N/A
1.20-2021.06.16	1.20.4	20.10.5	N/A
1.20-2021.05.18	1.20.4	20.10.4	N/A

Kubernetes version 1.19

Kubernetes version 1.19

AMI version	kubelet version	Docker version	csi-proxy version
1.19-2022.04.14	1.19.15	20.10.9	1.1.1
1.19-2022.03.10	1.19.15	20.10.9	N/A
1.19-2022.02.23	1.19.15	20.10.9	N/A
1.19-2022.01.18	1.19.15	20.10.9	N/A
1.19-2021.12.21	1.19.15	20.10.8	N/A
1.19-2021.11.10	1.19.14	20.10.7	N/A
1.19-2021.10.14	1.19.14	20.10.7	N/A
1.19-2021.09.16	1.19.13	20.10.7	N/A
1.19-2021.08.12	1.19.8	20.10.6	N/A
1.19-2021.07.14	1.19.8	20.10.6	N/A
1.19-2021.06.16	1.19.8	20.10.5	N/A
1.19-2021.05.18	1.19.8	20.10.4	N/A
1.19-2021.04.14	1.19.6	20.10.0	N/A
1.19-2021.03.10	1.19.6	19.03.14	N/A
1.19-2021.02.18	1.19.6	19.03.14	N/A

Kubernetes version 1.18

Kubernetes version 1.18

AMI version	kubelet version	Docker version	csi-proxy version
1.18-2022.03.10	1.18.20	20.10.9	1.1.1
1.18-2022.02.23	1.18.20	20.10.9	N/A
1.18-2022.01.18	1.18.20	20.10.9	N/A
1.18-2021.12.21	1.18.20	20.10.8	N/A
1.18-2021.11.10	1.18.20	20.10.7	N/A
1.18-2021.10.14	1.18.20	20.10.7	N/A
1.18-2021.09.16	1.18.20	20.10.7	N/A
1.18-2021.08.12	1.18.16	20.10.6	N/A
1.18-2021.07.14	1.18.16	20.10.6	N/A
1.18-2021.06.16	1.18.16	20.10.5	N/A

AMI version	kubelet version	Docker version	csi-proxy version
1.18-2021.05.18	1.18.16	20.10.4	N/A
1.18-2021.04.14	1.18.9	20.10.0	N/A
1.18-2021.03.10	1.18.9	19.03.14	N/A
1.18-2021.02.10	1.18.9	19.03.14	N/A
1.18-2021.01.13	1.18.9	19.03.14	N/A
1.18-2020.12.11	1.18.9	19.03.13	N/A
1.18-2020.11.12	1.18.9	18.09.7	N/A
1.18-2020.10.29	1.18.8	18.09.7	N/A
1.18-2020.10.08	1.18.8	18.09.7	N/A

Kubernetes version 1.17

The most recent version is the last version we're releasing for Amazon EKS 1.17 clusters. It will be available until the 1.17 end of support date. For more information, see [Amazon EKS Kubernetes release calendar \(p. 74\)](#). For newer AMI releases, update your cluster to a later Kubneretes version. For more information, see [Updating a cluster \(p. 34\)](#).

Kubernetes version 1.17

AMI version	kubelet version	Docker version
1.17-2021.08.12	1.17.17	20.10.6
1.17-2021.06.16	1.17.17	20.10.5
1.17-2021.05.18	1.17.17	20.10.4
1.17-2021.04.14	1.17.12	20.10.0
1.17-2021.03.10	1.17.12	19.03.14
1.17-2021.02.10	1.17.12	19.03.14
1.17-2021.01.13	1.17.12	19.03.14
1.17-2020.12.11	1.17.12	19.03.13
1.17-2020.11.12	1.17.12	18.09.7
1.17-2020.10.29	1.17.11	18.09.7
1.17-2020.09.09	1.17.11	18.09.7
1.17-2020.08.13	1.17.9	18.09.7

Retrieving Amazon EKS optimized Windows AMI IDs

You can programmatically retrieve the Amazon Machine Image (AMI) ID for Amazon EKS optimized AMIs by querying the AWS Systems Manager Parameter Store API. This parameter eliminates the need for you to manually look up Amazon EKS optimized AMI IDs. For more information about the Systems Manager

Parameter Store API, see [GetParameter](#). Your user account must have the `ssm:GetParameter` IAM permission to retrieve the Amazon EKS optimized AMI metadata.

You can retrieve the AMI ID with the AWS CLI or the AWS Management Console.

- **AWS CLI** – You can retrieve the image ID of the latest recommended Amazon EKS optimized Windows AMI with the following command by using the sub-parameter `image_id`. You can replace `1.22` with any supported Amazon EKS version and can replace `region-code` with an [Amazon EKS supported Region](#) for which you want the AMI ID. Replace `Core` with `Full` to see the Windows Server full AMI ID. You can also replace `2019` with `20H2` for the Core version only for version 1.21 and later.

```
aws ssm get-parameter --name /aws/service/ami-windows-latest/Windows_Server-2019-English-Core-EKS_Optimized-1.22/image_id --region region-code --query "Parameter.Value" --output text
```

Example output:

```
ami-1234567890abcdef0
```

- **AWS Management Console** – You can query for the recommended Amazon EKS optimized AMI ID using a URL. The URL opens the Amazon EC2 Systems Manager console with the value of the ID for the parameter. In the following URL, you can replace `1.21` with any supported Amazon EKS version and can replace `region-code` with an [Amazon EKS supported Region](#) for which you want the AMI ID. Replace `Core` with `Full` to see the Windows Server full AMI ID. You can also replace `2019` with `20H2` for the Core version only for version 1.21 and later.

```
https://console.aws.amazon.com/systems-manager/parameters/aws/service/ami-windows-latest/Windows_Server-2019-English-Core-EKS_Optimized-1.22/image_id/description?region=region-code
```

Amazon EKS optimized Windows AMI

You can use Amazon EC2 Image Builder to create custom Amazon EKS optimized Windows AMIs. You must create your own Image Builder recipe. For more information, see [Create image recipes and versions](#) in the *EC2 Image Builder User Guide*. When creating a recipe and selecting a **Source image**, you have the following options:

- **Select managed images** – If you select this option, you can choose one of the following options for **Image origin**.
 - **Quick start (Amazon-managed)** – In the Image name dropdown, select an [Amazon EKS supported Windows Server version \(p. 211\)](#).
 - **Images owned by me** – For **Image name**, select the ARN of your own image with your own license. The image that you provide can't already have Amazon EKS components installed.
 - **Enter custom AMI ID** – For **AMI ID**, enter the ID for your AMI with your own license. The image that you provide can't already have Amazon EKS components installed.

In the search box under **Build components - Windows**, select **Amazon-managed** in the dropdown list and then search on `eks`. Select the `eks-optimized-ami-windows` search result, even though the result returned may not be the version that you want. Under **Selected components**, select **Versioning options**, then select **Specify component version**. Enter `version.x`, replacing `version` with a [supported Kubernetes version \(p. 67\)](#). If you enter `1.22.x` as the component version, your Image Builder pipeline builds an AMI with the latest `1.22.x` `kubelet` version.

To determine which `kubelet` and Docker versions are installed with the component, select **Components** in the left navigation. Under **Components**, change **Owned by me** to **Quick start (Amazon-managed)**.

In the **Find components by name** box, enter eks. The search results show the kubelet and Docker version in the component returned for each supported Kubernetes version. The components go through functional testing on the Amazon EKS supported Windows versions. Any other Windows versions are not supported and might not be compatible with the component.

Also include the update-windows component for the latest Windows patches for the operating system.

Storage

This chapter covers storage options for Amazon EKS clusters.

The [Storage classes \(p. 229\)](#) topic uses the in-tree Amazon EBS storage provisioner.

Note

The existing [in-tree Amazon EBS plugin](#) is still supported, but by using a CSI driver, you benefit from the decoupling of Kubernetes upstream release cycle and CSI driver release cycle. Eventually, the in-tree plugin will be discontinued in favor of the CSI driver. However, the CSI driver isn't supported on Fargate.

Topics

- [Storage classes \(p. 229\)](#)
- [Amazon EBS CSI driver \(p. 231\)](#)
- [Amazon EFS CSI driver \(p. 244\)](#)
- [Amazon FSx for Lustre CSI driver \(p. 256\)](#)
- [Amazon FSx for NetApp ONTAP CSI driver \(p. 262\)](#)

Storage classes

Amazon EKS clusters that were created prior to Kubernetes version 1.11 weren't created with any storage classes. You must define storage classes for your cluster to use and you should define a default storage class for your persistent volume claims. For more information, see [Storage classes](#) in the Kubernetes documentation.

Note

This topic uses the [in-tree Amazon EBS storage provisioner](#). The existing [in-tree Amazon EBS plugin](#) is still supported, but by using a CSI driver, you benefit from the decoupling of Kubernetes upstream release cycle and CSI driver release cycle. Eventually, the in-tree plugin will be discontinued in favor of the CSI driver. However, the CSI driver isn't supported on Fargate.

To create an AWS storage class for your Amazon EKS cluster

1. Determine which storage classes your cluster already has.

```
kubectl get storageclass
```

Example output:

NAME	PROVISIONER	RECLAIMPOLICY	VOLUMEBINDINGMODE
ALLOWVOLUMEEXPANSION	AGE		
gp2 (default)	kubernetes.io/aws-ebs	Delete	WaitForFirstConsumer false
	34m		

If your cluster returns the previous output, then it already has the storage class defined in the remaining steps. You can define other storage classes using the steps for deploying any of the CSI drivers in the [Storage \(p. 229\)](#) chapter. Once deployed, you can set one of the storage classes as your [default \(p. 230\)](#) storage class.

2. Create an AWS storage class manifest file for your storage class. The `gp2-storage-class.yaml` example below defines a storage class called `gp2` that uses the Amazon EBS `gp2` volume type.

For more information about the options available for AWS storage classes, see [AWS EBS](#) in the Kubernetes documentation.

```
kind: StorageClass
apiVersion: storage.k8s.io/v1
metadata:
  name: gp2
  annotations:
    storageclass.kubernetes.io/is-default-class: "true"
provisioner: kubernetes.io/aws-ebs
parameters:
  type: gp2
  fsType: ext4
```

3. Use `kubectl` to create the storage class from the manifest file.

```
kubectl create -f gp2-storage-class.yaml
```

Output:

```
storageclass "gp2" created
```

To define a default storage class

1. List the existing storage classes for your cluster. A storage class must be defined before you can set it as a default.

```
kubectl get storageclass
```

Output:

NAME	PROVISIONER	AGE
gp2	kubernetes.io/aws-ebs	8m

2. Choose a storage class and set it as your default by setting the `storageclass.kubernetes.io/is-default-class=true` annotation.

```
kubectl annotate storageclass gp2 storageclass.kubernetes.io/is-default-class=true
```

Output:

```
storageclass "gp2" patched
```

3. Verify that the storage class is now set as default.

```
kubectl get storageclass
```

Output:

```
gp2 (default)  kubernetes.io/aws-ebs  12m
```

Amazon EBS CSI driver

The Amazon Elastic Block Store (Amazon EBS) Container Storage Interface (CSI) driver allows Amazon Elastic Kubernetes Service (Amazon EKS) clusters to manage the lifecycle of Amazon EBS volumes for persistent volumes.

Here are some things to consider about using the Amazon EBS CSI driver.

- The Amazon EBS CSI plugin requires IAM permissions to make calls to AWS APIs on your behalf. For more information, see [Creating the Amazon EBS CSI driver IAM role for service accounts \(p. 231\)](#).
- You can run the Amazon EBS CSI controller on Fargate, but you can't mount volumes to Fargate pods.
- Alpha features of the Amazon EBS CSI driver aren't supported on Amazon EKS clusters.

The Amazon EBS CSI driver isn't installed when you first create a cluster. To use the driver, you must add it as an Amazon EKS add-on or as a self-managed add-on.

- For instructions on how to add it as an Amazon EKS add-on, see [Managing the Amazon EBS CSI driver as an Amazon EKS add-on \(p. 237\)](#).
- For instructions on how to add it as a self-managed add-on, see the [Amazon EBS Container Storage Interface \(CSI\) driver](#) project on GitHub.

After you installed the CSI driver with either method, you can test the functionality with a sample application. For more information, see [Deploy a sample application and verify that the CSI driver is working \(p. 242\)](#).

Creating the Amazon EBS CSI driver IAM role for service accounts

The Amazon EBS CSI plugin requires IAM permissions to make calls to AWS APIs on your behalf. For more information, see [Set up driver permission](#) on GitHub.

When the plugin is deployed, it creates and is configured to use a service account that's named `ebs-csi-controller-sa`. The service account is bound to a Kubernetes `clusterrole` that's assigned the required Kubernetes permissions.

Note

No matter if you configure the Amazon EBS CSI plugin to use IAM roles for service accounts, the pods have access to the permissions that are assigned to the IAM role. This is the case except when you block access to IMDS. For more information, see [Security best practices for Amazon EKS \(p. 501\)](#).

Prerequisites

- An existing cluster that's version 1.18 or later.
 - 1.18 requires `eks.9` or later.
 - 1.19 requires `eks.7` or later.
 - 1.20 requires `eks.3` or later.
 - 1.21 requires `eks.3` or later.
- An existing AWS Identity and Access Management (IAM) OpenID Connect (OIDC) provider for your cluster. To determine whether you already have one, or to create one, see [Create an IAM OIDC provider for your cluster \(p. 446\)](#).

Create an IAM role and attach the required AWS managed policy to it. You can use `eksctl`, the AWS Management Console, or the AWS CLI.

`eksctl`

To create your Amazon EBS CSI plugin IAM role with `eksctl`

1. Create an IAM role and attach the required AWS managed policy with the following command. Replace `my-cluster` with the name of your cluster. The command deploys an AWS CloudFormation stack that creates an IAM role, attaches the IAM policy to it, and annotates the existing `ebs-csi-controller-sa` service account with the Amazon Resource Name (ARN) of the IAM role. If your cluster is in the AWS GovCloud (US-East) or AWS GovCloud (US-East) AWS Regions, then replace `arn:aws:` with `arn:aws-us-gov:` before running the following command.

```
eksctl create iamserviceaccount \
--name ebs-csi-controller-sa \
--namespace kube-system \
--cluster my-cluster \
--attach-policy-arn arn:aws:iam::aws:policy/service-role/AmazonEBSCSIDriverPolicy \
\
--approve \
--role-only \
--role-name AmazonEKS_EBS_CSI_DriverRole
```

2. If you use a custom KMS key for encryption on your Amazon EBS volumes, customize the IAM role as needed. For example, do the following:
 - a. Copy and paste the following code into a new `kms-key-for-encryption-on-ebs.json` file. Replace `custom-key-id` with the custom KMS key ID.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "kms:CreateGrant",
        "kms>ListGrants",
        "kms:RevokeGrant"
      ],
      "Resource": ["custom-key-id"],
      "Condition": {
        "Bool": {
          "kms:GrantIsForAWSResource": "true"
        }
      }
    },
    {
      "Effect": "Allow",
      "Action": [
        "kms:Encrypt",
        "kms:Decrypt",
        "kms:ReEncrypt*",
        "kms:GenerateDataKey*",
        "kms:DescribeKey"
      ],
      "Resource": ["custom-key-id"]
    }
  ]
}
```

- b. Create the policy. You can change `KMS_Key_For_Encryption_On_EBS_Policy` to a different name. However, if you do, make sure to change it in later steps, too.

```
aws iam create-policy \
--policy-name KMS_Key_For_Encryption_On_EBS_Policy \
--policy-document file://kms-key-for-encryption-on-ebs.json
```

- c. Attach the IAM policy to the role. Replace `111122223333` with your account ID. If your cluster is in the AWS GovCloud (US-East) or AWS GovCloud (US-East) AWS Regions, then replace `arn:aws:` with `arn:aws-us-gov:` before running the following command.

```
aws iam attach-role-policy \
--policy-arn
arn:aws:iam::111122223333:policy/KMS_Key_For_Encryption_On_EBS_Policy \
--role-name AmazonEKS_EBS_CSI_DriverRole
```

AWS Management Console

To create your Amazon EBS CSI plugin IAM role with the AWS Management Console

1. Open the IAM console at <https://console.aws.amazon.com/iam/>.
2. In the left navigation pane, choose **Roles**.
3. On the **Roles** page, choose **Create role**.
4. On the **Select trusted entity** page, do the following:
 - a. In the **Trusted entity type** section, choose **Web identity**.
 - b. For **Identity provider**, choose the **OpenID Connect provider URL** for your cluster (as shown under **Configuration Details** in Amazon EKS).
 - c. For **Audience**, choose `sts.amazonaws.com`.
 - d. Choose **Next**.
5. On the **Add permissions** page, do the following:
 - a. In the **Filter policies** box, enter `AmazonEBSCSIDriverPolicy`.
 - b. Select the check box to the left of the `AmazonEBSCSIDriverPolicy` returned in the search.
 - c. Choose **Next**.
6. On the **Name, review, and create** page, do the following:
 - a. For **Role name**, enter a unique name for your role, such as `AmazonEKS_EBS_CSI_DriverRole`.
 - b. Under **Add tags (Optional)**, add metadata to the role by attaching tags as key–value pairs. For more information about using tags in IAM, see [Tagging IAM Entities](#) in the *IAM User Guide*
 - c. Choose **Create role**.
7. After the role is created, choose the role in the console to open it for editing.
8. Choose the **Trust relationships** tab, and then choose **Edit trust policy**.
9. Find the line that looks similar to the following line:

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

Add a comma to the end of the previous line, and then add the following line after the previous line. Replace `region-code` with the AWS Region that your cluster is in. Replace `EXAMPLED539D4633E53DE1B71EXAMPLE` with your cluster's OIDC provider ID.

```
"oidc.eks.region-code.amazonaws.com/id/EXAMPLED539D4633E53DE1B71EXAMPLE:sub":  
"system:serviceaccount:kube-system:ebs-csi-controller-sa"
```

10. Choose **Update policy** to finish.
11. If you use a custom KMS key for encryption on your Amazon EBS volumes, customize the IAM role as needed. For example, do the following:
 - a. In the left navigation pane, choose **Policies**.
 - b. On the **Policies** page, choose **Create Policy**.
 - c. On the **Create policy** page, choose the **JSON** tab.
 - d. Copy and paste the following code into the editor, replacing `custom-key-id` with the custom KMS key ID:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "kms:CreateGrant",
        "kms>ListGrants",
        "kms:RevokeGrant"
      ],
      "Resource": [ "custom-key-id" ],
      "Condition": {
        "Bool": {
          "kms:GrantIsForAWSResource": "true"
        }
      }
    },
    {
      "Effect": "Allow",
      "Action": [
        "kms:Encrypt",
        "kms:Decrypt",
        "kms:ReEncrypt*",
        "kms:GenerateDataKey*",
        "kms:DescribeKey"
      ],
      "Resource": [ "custom-key-id" ]
    }
  ]
}
```

- e. Choose **Next: Tags**.
- f. On the **Add tags (Optional)** page, choose **Next: Review**.
- g. For **Name**, enter a unique name for your policy (for example, `KMS_Key_For_Encryption_On_EBS_Policy`).
- h. Choose **Create policy**.
- i. In the left navigation pane, choose **Roles**.
- j. Choose the `AmazonEKS_EBS_CSI_DriverRole` in the console to open it for editing.
- k. From the **Add permissions** drop-down list, choose **Attach policies**.
- l. In the **Filter policies** box, enter `KMS_Key_For_Encryption_On_EBS_Policy`.

- m. Select the check box to the left of the *KMS_Key_For_Encryption_On_EBS_Policy* that was returned in the search.
- n. Choose **Attach policies**.

AWS CLI

To create your Amazon EBS CSI plugin IAM role with the AWS CLI

1. View your cluster's OIDC provider URL. Replace *my-cluster* with your cluster name. If the output from the command is None, review the **Prerequisites**.

```
aws eks describe-cluster \
--name my-cluster \
--query "cluster.identity.oidc.issuer" \
--output text
```

Example output:

```
https://oidc.eks.region-code.amazonaws.com/id/EXAMPLED539D4633E53DE1B71EXAMPLE
```

2. Create the IAM role.

- a. Copy the following contents to a file that's named *aws-ebs-csi-driver-trust-policy.json*. Replace *111122223333* with your account ID, *region-code* with your AWS Region, and *EXAMPLED539D4633E53DE1B71EXAMPLE* with the value that was returned in the previous step. If your cluster is in the AWS GovCloud (US-East) or AWS GovCloud (US-East) AWS Regions, then replace *arn:aws:* with *arn:aws-us-gov:*.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Federated": "arn:aws:iam::111122223333:oidc-provider/oidc.eks.region-code.amazonaws.com/id/EXAMPLED539D4633E53DE1B71EXAMPLE"
      },
      "Action": "sts:AssumeRoleWithWebIdentity",
      "Condition": {
        "StringEquals": {
          "oidc.eks.region-code.amazonaws.com/id/EXAMPLED539D4633E53DE1B71EXAMPLE:aud": "sts.amazonaws.com",
          "oidc.eks.region-code.amazonaws.com/id/EXAMPLED539D4633E53DE1B71EXAMPLE:sub": "system:serviceaccount:kube-system:ebs-csi-controller-sa"
        }
      }
    }
  ]
}
```

- b. Create the role. You can change *AmazonEKS_EBS_CSI_DriverRole* to a different name. If you change it, make sure to change it in later steps.

```
aws iam create-role \
--role-name AmazonEKS_EBS_CSI_DriverRole \
--assume-role-policy-document file://"aws-ebs-csi-driver-trust-policy.json"
```

3. Attach the required AWS managed policy to the role. If your cluster is in the AWS GovCloud (US-East) or AWS GovCloud (US-East) AWS Regions, then replace `arn:aws:` with `arn:aws-us-gov:` before running the following command.

```
aws iam attach-role-policy \
--policy-arn arn:aws:iam::aws:policy/service-role/AmazonEBSCSIDriverPolicy \
--role-name AmazonEKS_EBS_CSI_DriverRole
```

4. If you use a custom KMS key for encryption on your Amazon EBS volumes, customize the IAM role as needed. For example, do the following:
 - a. Copy and paste the following code into a new `kms-key-for-encryption-on-ebs.json` file. Replace `custom-key-id` with the custom KMS key ID.

```
{ "Version": "2012-10-17", "Statement": [ { "Effect": "Allow", "Action": [ "kms:CreateGrant", "kms>ListGrants", "kms:RevokeGrant" ], "Resource": [ "custom-key-id" ], "Condition": { "Bool": { "kms:GrantIsForAWSResource": "true" } } }, { "Effect": "Allow", "Action": [ "kms:Encrypt", "kms:Decrypt", "kms:ReEncrypt*", "kms:GenerateDataKey*", "kms:DescribeKey" ], "Resource": [ "custom-key-id" ] } ] }
```

- b. Create the policy. You can change `KMS_Key_For_Encryption_On_EBS_Policy` to a different name. However, if you do, make sure to change it in later steps, too.

```
aws iam create-policy \
--policy-name KMS_Key_For_Encryption_On_EBS_Policy \
--policy-document file:///kms-key-for-encryption-on-ebs.json
```

- c. Attach the IAM policy to the role. Replace `111122223333` with your account ID. If your cluster is in the AWS GovCloud (US-East) or AWS GovCloud (US-East) AWS Regions, then replace `arn:aws:` with `arn:aws-us-gov:` before running the following command.

```
aws iam attach-role-policy \
--policy-arn
arn:aws:iam::111122223333:policy/KMS_Key_For_Encryption_On_EBS_Policy \
--role-name AmazonEKS_EBS_CSI_DriverRole
```

Managing the Amazon EBS CSI driver as an Amazon EKS add-on

To improve security and reduce the amount of work, you can manage the Amazon EBS CSI driver as an Amazon EKS add-on. For information about Amazon EKS add-ons, see [Amazon EKS add-ons \(p. 388\)](#). You can add the Amazon EBS CSI add-on by following the steps in [Adding the Amazon EBS CSI add-on \(p. 237\)](#).

If you added the Amazon EBS CSI add-on, you can manage it by following the steps in the [Updating the Amazon EBS CSI driver as an Amazon EKS add-on \(p. 238\)](#) and [Removing the Amazon EBS CSI add-on \(p. 241\)](#) sections.

Prerequisites

- An existing cluster that's version 1.18 or later. To see the required platform version, run the following command.

```
aws eks describe-addon-versions --addon-name aws-ebs-csi-driver
```

- An existing AWS Identity and Access Management (IAM) OpenID Connect (OIDC) provider for your cluster. To determine whether you already have one, or to create one, see [Create an IAM OIDC provider for your cluster \(p. 446\)](#).
- An Amazon EBS CSI driver IAM role. For more information, see [Creating the Amazon EBS CSI driver IAM role for service accounts \(p. 231\)](#).

Important

To use the snapshot functionality of the Amazon EBS CSI driver, you must install the external snapshotter before the installation of the add-on. The external snapshotter components must be installed in the following order:

- [CustomResourceDefinition](#) (CRD) for `volumesnapshotclasses`, `volumesnapshots`, and `volumesnapshotcontents`
- [RBAC](#) (`ClusterRole`, `ClusterRoleBinding`, and so on)
- [controller deployment](#)

For more information, see [CSI Snapshotter](#) on GitHub.

Adding the Amazon EBS CSI add-on

Important

Before adding the Amazon EBS CSI add-on, confirm that you don't self-manage any settings that Amazon EKS will start managing. To determine which settings Amazon EKS manages, see [Amazon EKS add-on configuration \(p. 389\)](#).

You can use `eksctl`, the AWS Management Console, or the AWS CLI to add the Amazon EBS CSI add-on to your cluster .

`eksctl`

To add the Amazon EBS CSI add-on using `eksctl`

Replace `my-cluster` with the name of your cluster, `111122223333` with your account ID, and `AmazonEKS_EBS_CSI_DriverRole` with the name of the role created earlier. If your cluster is in the AWS GovCloud (US-East) or AWS GovCloud (US-East) AWS Regions, then replace `arn:aws:` with `arn:aws-us-gov:` before running the following command.

```
eksctl create addon --name aws-ebs-csi-driver --cluster my-cluster --service-account-role-arn arn:aws:iam::111122223333:role/AmazonEKS_EBS_CSI_DriverRole --force
```

If you remove the `--force` option and there's a conflict with your existing settings, the command fails. You can use the resulting error message to troubleshoot the conflict. Before specifying this option, make sure that the Amazon EKS add-on doesn't manage settings that you need to self-manage. This is because those settings are overwritten with this option. For more information about managing Amazon EKS add-ons, see [Amazon EKS add-on configuration \(p. 389\)](#).

AWS Management Console

To add the Amazon EBS CSI add-on using the AWS Management Console

1. Open the Amazon EKS console at <https://console.aws.amazon.com/eks/home#/clusters>.
2. In the left navigation pane, choose Amazon EKS **Clusters**.
3. Choose the name of the cluster that you want to configure the Amazon EBS CSI add-on for.
4. Choose the **Configuration** tab.
5. Choose the **Add-ons** tab.
6. Choose **Add new**.
 - a. Select **Amazon EBS CSI Driver** for **Name**.
 - b. Select the **Version** you'd like to use.
 - c. For **Service account role**, select the name of an IAM role that you attached the IAM policy to.
 - d. If you select **Override existing configuration for this add-on on the cluster**, one or more of the settings for the existing add-on can be overwritten with the Amazon EKS add-on settings. If you don't enable this option and there's a conflict with your existing settings, the operation fails. You can use the resulting error message to troubleshoot the conflict. Before selecting this option, make sure that the Amazon EKS add-on doesn't manage settings that you need to self-manage. For more information about managing Amazon EKS add-ons, see [Amazon EKS add-on configuration \(p. 389\)](#).
 - e. Choose **Add**.

AWS CLI

To add the Amazon EBS CSI add-on using the AWS CLI

Replace `my-cluster` with the name of your cluster, `111122223333` with your account ID, and `AmazonEKS_EBS_CSI_DriverRole` with the name of the role that was created earlier. If your cluster is in the AWS GovCloud (US-East) or AWS GovCloud (US-East) AWS Regions, then replace `arn:aws:` with `arn:aws-us-gov:` before running the following command.

```
aws eks create-addon \
  --cluster-name my-cluster \
  --addon-name aws-ebs-csi-driver \
  --service-account-role-arn
  arn:aws:iam::111122223333:role/AmazonEKS_EBS_CSI_DriverRole
```

Updating the Amazon EBS CSI driver as an Amazon EKS add-on

Amazon EKS doesn't automatically update Amazon EBS CSI for your cluster when new versions are released or after you [update your cluster \(p. 34\)](#) to a new Kubernetes minor version. To update Amazon EBS CSI on an existing cluster, you must initiate the update and then Amazon EKS updates the add-on for you.

Important

Update your cluster and nodes to a new Kubernetes minor version before you update Amazon EBS CSI to the same minor version.

eksctl

To update the Amazon EBS CSI add-on using eksctl

1. Check the current version of your Amazon EBS CSI add-on. Replace *my-cluster* with your cluster name.

```
eksctl get addon --name aws-ebs-csi-driver --cluster my-cluster
```

Example output:

NAME	VERSION	STATUS	ISSUES	IAMROLE	UPDATE
aws-ebs-csi-driver	v1.4.0-eksbuild.preview	ACTIVE	0		
	v1.5.1-eksbuild.1				

2. Update the add-on to the version returned under UPDATE AVAILABLE in the output of the previous step.

```
eksctl update addon \
--name aws-ebs-csi-driver \
--version v1.5.1-eksbuild.1 \
--cluster my-cluster \
--force
```

If you remove the `--force` option and there's a conflict with your existing settings, the command fails. You can use the resulting error message to troubleshoot the conflict. Before specifying this option, make sure that the Amazon EKS add-on doesn't manage settings that you need to self-manage. This is because those settings are overwritten with this option. For more information about managing Amazon EKS add-ons, see [Amazon EKS add-on configuration \(p. 389\)](#).

AWS Management Console

To update the Amazon EBS CSI add-on using the AWS Management Console

1. Open the Amazon EKS console at <https://console.aws.amazon.com/eks/home#/clusters>.
2. In the left navigation pane, choose Amazon EKS **Clusters**.
3. Choose the name of the cluster that you want to update the Amazon EBS CSI add-on for.
4. Choose the **Configuration** tab.
5. Choose the **Add-ons** tab.
6. Select the radio button in the upper right of the **aws-ebs-csi-driver** box.
7. Choose **Edit**.
 - a. Select the **Version** of the Amazon EKS add-on that you want to use.
 - b. For **Service account role**, select the name of the IAM role that you've attached the Amazon EBS CSI driver IAM policy to.
 - c. If you select **Override existing configuration for this add-on on the cluster**, then one or more of the settings for the existing add-on can be overwritten with the Amazon EKS add-on settings. If you don't enable this option and there's a conflict with your existing settings,

the operation fails with an error message to help you resolve the conflict. Before selecting this option, make sure that the Amazon EKS add-on doesn't manage settings that you need to self-manage. For more information about managing Amazon EKS add-ons, see [Amazon EKS add-on configuration \(p. 389\)](#).

- d. Select **Update**.

AWS CLI

To update the Amazon EBS CSI add-on using the AWS CLI

1. Check the current version of your Amazon EBS CSI add-on. Replace *my-cluster* with your cluster name.

```
aws eks describe-addon \
--cluster-name my-cluster \
--addon-name aws-ebs-csi-driver \
--query "addon.addonVersion" \
--output text
```

Example output:

```
v1.4.0-eksbuild.preview
```

2. Determine which versions of the Amazon EBS CSI add-on are available for your cluster version.

```
aws eks describe-addon-versions \
--addon-name aws-ebs-csi-driver \
--kubernetes-version 1.20 \
--query "addons[].addonVersions[].[addonVersion,
compatibilities[].[defaultVersion]]" \
--output text
```

Example output:

```
v1.5.1-eksbuild.1
False
v1.4.0-eksbuild.preview
True
```

The version with **True** underneath is the default version deployed with new clusters with the version that you specified.

3. Update the add-on to the version with **True** that was returned in the output of the previous step. If it was returned in the output, you can also update to a later version.

```
aws eks update-addon \
--cluster-name my-cluster \
--addon-name aws-ebs-csi-driver \
--addon-version v1.4.0-eksbuild.preview \
--resolve-conflicts OVERWRITE
```

If you remove the `--resolve-conflicts OVERWRITE` option and there's a conflict with your existing settings, then the command fails with an error message to help you resolve the conflict. Before specifying this option, make sure that the Amazon EKS add-on doesn't manage settings that you need to self-manage. This is because those settings are overwritten with this option. For more information about managing Amazon EKS add-ons, see [Amazon EKS add-on configuration \(p. 389\)](#).

Removing the Amazon EBS CSI add-on

You have two options for removing an Amazon EKS add-on.

- **Preserve add-on software on your cluster** – This option removes Amazon EKS management of any settings. It also removes the ability for Amazon EKS to notify you of updates and automatically update the Amazon EKS add-on after you initiate an update. However, it preserves the add-on software on your cluster. This option makes the add-on a self-managed add-on, rather than an Amazon EKS add-on. With this option, there's no downtime for the add-on. The commands in this procedure use this option.
- **Remove add-on software entirely from your cluster** – We recommend that you remove the Amazon EKS add-on from your cluster only if there are no resources on your cluster that are dependent on it. To do this option, delete `--preserve` from the command you use in this procedure.

If the add-on has an IAM account associated with it, the IAM account isn't removed.

You can use `eksctl`, the AWS Management Console, or the AWS CLI to remove the Amazon EBS CSI add-on.

`eksctl`

To remove the Amazon EBS CSI add-on using `eksctl`

Replace `my-cluster` with the name of your cluster, and then run the following command.

```
eksctl delete addon --cluster my-cluster --name aws-ebs-csi-driver --preserve
```

AWS Management Console

To remove the Amazon EBS CSI add-on using the AWS Management Console

1. Open the Amazon EKS console at <https://console.aws.amazon.com/eks/home#/clusters>.
2. In the left navigation pane, choose Amazon EKS **Clusters**.
3. Choose the name of the cluster that you want to remove the Amazon EBS CSI add-on for.
4. Choose the **Configuration** tab.
5. Choose the **Add-ons** tab.
6. Select the radio button in the upper right of the **aws-ebs-csi-driver** box.
7. Choose **Remove**.
8. Select **Preserve on cluster** if you want Amazon EKS to stop managing settings for the add-on. Do this if you want to retain the add-on software on your cluster. This is so that you can manage all of the settings of the add-on on your own.
9. Enter **aws-ebs-csi-driver**.
10. Select **Remove**.

AWS CLI

To remove the Amazon EBS CSI add-on using the AWS CLI

Replace `my-cluster` with the name of your cluster, and then run the following command.

```
aws eks delete-addon --cluster-name my-cluster --addon-name aws-ebs-csi-driver --  
preserve
```

Deploy a sample application and verify that the CSI driver is working

You can test the CSI driver functionality with a sample application. This topic shows one example, but you can also do the following:

- Deploy a sample application that uses the external snapshotter to create volume snapshots. For more information, see [Volume Snapshots](#) on GitHub.
- Deploy a sample application that uses volume resizing. For more information, see [Volume Resizing](#) on GitHub.

This procedure uses the [Dynamic volume provisioning](#) example from the [Amazon EBS Container Storage Interface \(CSI\) driver](#) GitHub repository to consume a dynamically provisioned Amazon EBS volume.

1. Clone the [Amazon EBS Container Storage Interface \(CSI\) driver](#) GitHub repository to your local system.

```
git clone https://github.com/kubernetes-sigs/aws-ebs-csi-driver.git
```

2. Navigate to the dynamic-provisioning example directory.

```
cd aws-ebs-csi-driver/examples/kubernetes/dynamic-provisioning/
```

3. Deploy the ebs-sc storage class, ebs-claim persistent volume claim, and app sample application from the manifests directory.

```
kubectl apply -f manifests/
```

4. Describe the ebs-sc storage class.

```
kubectl describe storageclass ebs-sc
```

Example output:

```
Name:           ebs-sc
IsDefaultClass: No
Annotations:    kubectl.kubernetes.io/last-applied-
                configuration={"apiVersion":"storage.k8s.io/v1","kind":"StorageClass","metadata":{ "annotations":{}, "name":"ebs-
                sc"}, "provisioner":"ebs.csi.aws.com", "volumeBindingMode":"WaitForFirstConsumer"}
Provisioner:     ebs.csi.aws.com
Parameters:      <none>
AllowVolumeExpansion: <unset>
MountOptions:    <none>
ReclaimPolicy:   Delete
VolumeBindingMode: WaitForFirstConsumer
Events:          <none>
```

Note

The storage class uses the `WaitForFirstConsumer` volume binding mode. This means that volumes aren't dynamically provisioned until a pod makes a persistent volume claim. For more information, see [Volume Binding Mode](#) in the Kubernetes documentation.

5. Watch the pods in the default namespace. After a few minutes, the app pod's status changes to `Running`.

```
kubectl get pods --watch
```

Enter Ctrl+C to return to a shell prompt.

6. List the persistent volumes in the default namespace. Look for a persistent volume with the default/ebs-claim claim.

```
kubectl get pv
```

Example output:

NAME	STATUS	CLAIM	STORAGECLASS	CAPACITY	ACCESS MODES	RECLAIM POLICY
pvc-37717cd6-d0dc-11e9-b17f-06fad4858a5a	Bound	default/ebs-claim	ebs-sc	4Gi	RWO	Delete

7. Describe the persistent volume. Replace pvc-37717cd6-d0dc-11e9-b17f-06fad4858a5a with the value from the output in the previous step.

```
kubectl describe pv pvc-37717cd6-d0dc-11e9-b17f-06fad4858a5a
```

Example output:

```
Name:           pvc-37717cd6-d0dc-11e9-b17f-06fad4858a5a
Labels:         <none>
Annotations:   pv.kubernetes.io/provisioned-by: ebs.csi.aws.com
Finalizers:    [kubernetes.io/pv-protection external-attacher/ebs-csi-aws-com]
StorageClass:  ebs-sc
Status:        Bound
Claim:         default/ebs-claim
Reclaim Policy: Delete
Access Modes:  RWO
VolumeMode:   Filesystem
Capacity:     4Gi
Node Affinity:
  Required Terms:
    Term 0: topology.ebs.csi.aws.com/zone in [region-code]
Message:
Source:
  Type:          CSI (a Container Storage Interface (CSI) volume source)
  Driver:        ebs.csi.aws.com
  VolumeHandle: vol-0d651e157c6d93445
  ReadOnly:      false
  VolumeAttributes: storage.kubernetes.io/
csiProvisionerIdentity=1567792483192-8081-ebs.csi.aws.com
Events:        <none>
```

The Amazon EBS volume ID is the value for VolumeHandle in the previous output.

8. Verify that the pod is writing data to the volume.

```
kubectl exec -it app -- cat /data/out.txt
```

Example output:

```
Wed May 5 16:17:03 UTC 2021
Wed May 5 16:17:08 UTC 2021
Wed May 5 16:17:13 UTC 2021
```

```
Wed May 5 16:17:18 UTC 2021
...
```

9. After you're done, delete the resources for this sample application.

```
kubectl delete -f manifests/
```

Amazon EFS CSI driver

The [Amazon EFS Container Storage Interface \(CSI\) driver](#) provides a CSI interface that allows Kubernetes clusters running on AWS to manage the lifecycle of Amazon EFS file systems.

This topic shows you how to deploy the Amazon EFS CSI Driver to your Amazon EKS cluster and verify that it works.

Note

Alpha features of the Amazon EFS CSI Driver aren't supported on Amazon EKS clusters.

For detailed descriptions of the available parameters and complete examples that demonstrate the driver's features, see the [Amazon EFS Container Storage Interface \(CSI\) driver](#) project on GitHub.

Considerations

- The Amazon EFS CSI Driver isn't compatible with Windows-based container images.
- You can't use dynamic persistent volume provisioning with Fargate nodes, but you can use static provisioning.
- Dynamic provisioning requires 1.2 or later of the driver, which requires a 1.17 or later cluster. You can statically provision persistent volumes using 1.1 of the driver on any [supported Amazon EKS cluster version \(p. 67\)](#).
- Version 1.3.2 or later of this driver supports the Arm64 architecture, including Amazon EC2 Graviton-based instances.

Prerequisites

- An existing AWS Identity and Access Management (IAM) OpenID Connect (OIDC) provider for your cluster. To determine whether you already have one, or to create one, see [Create an IAM OIDC provider for your cluster \(p. 446\)](#).
- 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.
- The `kubectl` command line tool installed on your computer or AWS CloudShell. The version must be the same, or up to two versions later than your cluster version. To install or upgrade `kubectl`, see [Installing kubectl \(p. 4\)](#).

Note

A pod running on AWS Fargate automatically mounts an Amazon EFS file system, without needing the manual driver installation steps described on this page.

Create an IAM policy and role

Create an IAM policy and assign it to an IAM role. The policy will allow the Amazon EFS driver to interact with your file system.

To deploy the Amazon EFS CSI driver to an Amazon EKS cluster

1. Create an IAM policy that allows the CSI driver's service account to make calls to AWS APIs on your behalf.
 - a. Download the IAM policy document from GitHub. You can also view the [policy document](#).

```
curl -o iam-policy-example.json https://raw.githubusercontent.com/kubernetes-sigs/aws-efs-csi-driver/v1.3.7/docs/iam-policy-example.json
```

- b. Create the policy. You can change `AmazonEKS_EFS_CSI_Driver_Policy` to a different name, but if you do, make sure to change it in later steps too.

```
aws iam create-policy \
--policy-name AmazonEKS_EFS_CSI_Driver_Policy \
--policy-document file://iam-policy-example.json
```

2. Create an IAM role and attach the IAM policy to it. Annotate the Kubernetes service account with the IAM role ARN and the IAM role with the Kubernetes service account name. You can create the role using eksctl or the AWS CLI.

eksctl

Run the following command to create the IAM role and Kubernetes service account. It also attaches the policy to the role, annotates the Kubernetes service account with the IAM role ARN, and adds the Kubernetes service account name to the trust policy for the IAM role. Replace `my-cluster` with your cluster name and `111122223333` with your account ID. Replace `region-code` with the AWS Region that your cluster is in. If your cluster is in the AWS GovCloud (US-East) or AWS GovCloud (US-East) AWS Regions, then replace `arn:aws:` with `arn:aws-us-gov::`.

```
eksctl create iamserviceaccount \
--cluster my-cluster \
--namespace kube-system \
--name efs-csi-controller-sa \
--attach-policy-arn
arn:aws:iam::111122223333:policy/AmazonEKS_EFS_CSI_Driver_Policy \
--approve \
--region region-code
```

AWS CLI

1. Determine your cluster's OIDC provider URL. Replace `my-cluster` with your cluster name. If the output from the command is None, review the [Prerequisites](#).

```
aws eks describe-cluster --name my-cluster --query "cluster.identity.oidc.issuer" \
--output text
```

Example output:

```
https://oidc.eks.region-code.amazonaws.com/id/EXAMPLED539D4633E53DE1B71EXAMPLE
```

2. Create the IAM role, granting the Kubernetes service account the `AssumeRoleWithWebIdentity` action.
 - a. Copy the following contents to a file named `trust-policy.json`. Replace `111122223333` with your account ID. Replace `EXAMPLED539D4633E53DE1B71EXAMPLE` and `region-code` with the values returned in the previous step. If your cluster is in

the AWS GovCloud (US-East) or AWS GovCloud (US-East) AWS Regions, then replace `arn:aws:` with `arn:aws-us-gov:`.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "Federated": "arn:aws:iam::111122223333:oidc-provider/oidc.eks.region-
                code.amazonaws.com/id/EXAMPLED539D4633E53DE1B71EXAMPLE"
            },
            "Action": "sts:AssumeRoleWithWebIdentity",
            "Condition": {
                "StringEquals": {
                    "oidc.eks.region-code.amazonaws.com/
                    id/EXAMPLED539D4633E53DE1B71EXAMPLE:sub": "system:serviceaccount:kube-
                    system:efs-csi-controller-sa"
                }
            }
        }
    ]
}
```

- b. Create the role. You can change `AmazonEKS_EFS_CSI_DriverRole` to a different name, but if you do, make sure to change it in later steps too.

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

3. Attach the IAM policy to the role. Replace `111122223333` with your account ID. If your cluster is in the AWS GovCloud (US-East) or AWS GovCloud (US-East) AWS Regions, then replace `arn:aws:` with `arn:aws-us-gov:` before running the following command.

```
aws iam attach-role-policy \
--policy-arn arn:aws:iam::111122223333:policy/AmazonEKS_EFS_CSI_Driver_Policy \
--role-name AmazonEKS_EFS_CSI_DriverRole
```

4. Create a Kubernetes service account that's annotated with the ARN of the IAM role that you created.
- a. Save the following contents to a file named `efs-service-account.yaml`. Replace `111122223333` with your account ID. If your cluster is in the AWS GovCloud (US-East) or AWS GovCloud (US-East) AWS Regions, then replace `arn:aws:` with `arn:aws-us-gov:`.

```
---
apiVersion: v1
kind: ServiceAccount
metadata:
  labels:
    app.kubernetes.io/name: aws-efs-csi-driver
    name: efs-csi-controller-sa
    namespace: kube-system
  annotations:
    eks.amazonaws.com/role-arn:
      arn:aws:iam::111122223333:role/AmazonEKS_EFS_CSI_DriverRole
```

5. Create the Kubernetes service account on your cluster. The Kubernetes service account named `efs-csi-controller-sa` is annotated with the IAM role that you created named `AmazonEKS_EFS_CSI_DriverRole`.

```
kubectl apply -f efs-service-account.yaml
```

Install the Amazon EFS driver

Install the Amazon EFS CSI driver using Helm or a manifest.

Important

- The following steps install the 1.3.7 version of the driver, which requires a 1.17 or later cluster. If you're installing the driver on a cluster that's earlier than version 1.17, you need to install version 1.1 of the driver. For more information, see [Amazon EFS CSI driver](#) on GitHub.
- Encryption of data in transit using TLS is enabled by default. Using [encryption in transit](#), data is encrypted during its transition over the network to the Amazon EFS service. To disable it and mount volumes using NFSv4, set the volumeAttributes field encryptInTransit to "false" in your persistent volume manifest. For an example manifest, see [Encryption in Transit example](#) on GitHub.

Helm

This procedure requires Helm V3 or later. To install or upgrade Helm, see [Using Helm with Amazon EKS \(p. 430\)](#).

To install the driver using Helm

1. Add the Helm repo.

```
helm repo add aws-efs-csi-driver https://kubernetes-sigs.github.io/aws-efs-csi-driver/
```

2. Update the repo.

```
helm repo update
```

3. Install a release of the driver using the Helm chart. Replace the repository address with the cluster's [container image address \(p. 386\)](#).

```
helm upgrade -i aws-efs-csi-driver aws-efs-csi-driver/aws-efs-csi-driver \
--namespace kube-system \
--set image.repository=602401143452.dkr.ecr.region-code.amazonaws.com/eks/aws-efs-csi-driver \
--set controller.serviceAccount.create=false \
--set controller.serviceAccount.name=efs-csi-controller-sa
```

Manifest (private registry)

If you want to download the image with a manifest, we recommend first trying these steps to pull secured images from the private Amazon ECR registry.

To install the driver using images stored in the private Amazon ECR registry

1. Download the manifest.

```
kubectl kustomize \
```

```
"github.com/kubernetes-sigs/aws-efs-csi-driver/deploy/kubernetes/overlays/stable/ecr?ref=release-1.3" > private-ecr-driver.yaml
```

Note

If you encounter an issue that you aren't able to resolve by adding IAM permissions, try the "Manifest (public registry)" steps instead.

2. In the following command, replace `region-code` with the AWS Region that your cluster is in and then run the modified command to replace us-west-2 in the file with your AWS Region.

```
sed -i.bak -e 's|us-west-2|region-code|' private-ecr-driver.yaml
```

3. Replace `account` in the following command with the account from [Amazon container image registries \(p. 386\)](#) for the AWS Region that your cluster is in and then run the modified command to replace 602401143452 in the file.

```
sed -i.bak -e 's|602401143452|account|' private-ecr-driver.yaml
```

4. Edit the `private-ecr-driver.yaml` file and remove the following lines that create a Kubernetes service account. These lines aren't needed because the service account was created in a previous step.

```
apiVersion: v1
kind: ServiceAccount
metadata:
  labels:
    app.kubernetes.io/name: aws-efs-csi-driver
  name: efs-csi-controller-sa
  namespace: kube-system
---
```

5. Apply the manifest.

```
kubectl apply -f private-ecr-driver.yaml
```

Manifest (public registry)

For some situations, you may not be able to add the necessary IAM permissions to pull from the private Amazon ECR registry. One example of this scenario is if your IAM user or role isn't allowed to authenticate with someone else's account. When this is true, you can use the public Amazon ECR registry.

To install the driver using images stored in the public Amazon ECR registry

1. Download the manifest.

```
kubectl kustomize \
  "github.com/kubernetes-sigs/aws-efs-csi-driver/deploy/kubernetes/overlays/
stable/?ref=release-1.3" > public-ecr-driver.yaml
```

2. Edit the file and remove the following lines that create a Kubernetes service account. This isn't necessary because the service account was created in a previous step.

```
apiVersion: v1
kind: ServiceAccount
metadata:
  labels:
    app.kubernetes.io/name: aws-efs-csi-driver
```

```
name: efs-csi-controller-sa
namespace: kube-system
---
```

3. Apply the manifest.

```
kubectl apply -f public-ecr-driver.yaml
```

Create an Amazon EFS file system

The Amazon EFS CSI driver supports [Amazon EFS access points](#), which are application-specific entry points into an Amazon EFS file system that make it easier to share a file system between multiple pods. Access points can enforce a user identity for all file system requests that are made through the access point, and enforce a root directory for each pod. For more information, see [Amazon EFS access points](#) on GitHub.

Important

You must complete the following steps in the same terminal because variables are set and used across the steps.

To create an Amazon EFS file system for your Amazon EKS cluster

1. Retrieve the VPC ID that your cluster is in and store it in a variable for use in a later step. Replace *my-cluster* with your cluster name.

```
vpc_id=$(aws eks describe-cluster \
--name my-cluster \
--query "cluster.resourcesVpcConfig.vpcId" \
--output text)
```

2. Retrieve the CIDR range for your cluster's VPC and store it in a variable for use in a later step.

```
cidr_range=$(aws ec2 describe-vpcs \
--vpc-ids $vpc_id \
--query "Vpcs[].CidrBlock" \
--output text)
```

3. Create a security group with an inbound rule that allows inbound NFS traffic for your Amazon EFS mount points.

- a. Create a security group. Replace the *example values* with your own.

```
security_group_id=$(aws ec2 create-security-group \
--group-name MyEfsSecurityGroup \
--description "My EFS security group" \
--vpc-id $vpc_id \
--output text)
```

- b. Create an inbound rule that allows inbound NFS traffic from the CIDR for your cluster's VPC.

```
aws ec2 authorize-security-group-ingress \
--group-id $security_group_id \
--protocol tcp \
--port 2049 \
--cidr $cidr_range
```

Important

To further restrict access to your file system, you can use the CIDR for your subnet instead of the VPC.

4. Create an Amazon EFS file system for your Amazon EKS cluster.

- Create a file system. Replace *region-code* with the AWS Region that your cluster is in.

```
file_system_id=$(aws efs create-file-system \
--region region-code \
--performance-mode generalPurpose \
--query 'FileSystemId' \
--output text)
```

- Create mount targets.

- Determine the IP address of your cluster nodes.

```
kubectl get nodes
```

Example output:

NAME	STATUS	ROLES	AGE	VERSION
ip-192-168-56-0. <i>region-code</i> .compute.internal	Ready	<none>	19m	v1.19.6-
eks-49a6c0				

- Determine the IDs of the subnets in your VPC and which Availability Zone the subnet is in.

```
aws ec2 describe-subnets \
--filters "Name=vpc-id,Values=$vpc_id" \
--query 'Subnets[*].{SubnetId: SubnetId,AvailabilityZone: AvailabilityZone,CidrBlock: CidrBlock}' \
--output table
```

Example output:

DescribeSubnets		
AvailabilityZone	CidrBlock	SubnetId
<i>region-codec</i>	192.168.128.0/19	subnet- <i>EXAMPLE6e421a0e97</i>
<i>region-codeb</i>	192.168.96.0/19	subnet- <i>EXAMPLEd0503db0ec</i>
<i>region-codec</i>	192.168.32.0/19	subnet- <i>EXAMPLEe2ba886490</i>
<i>region-codeb</i>	192.168.0.0/19	subnet- <i>EXAMPLE123C7c5182</i>
<i>region-codea</i>	192.168.160.0/19	subnet- <i>EXAMPLE0416ce588p</i>

- Add mount targets for the subnets that your nodes are in. From the output in the previous two steps, the cluster has one node with an IP address of 192.168.56.0. That IP address is within the CidrBlock of the subnet with the ID subnet-*EXAMPLEe2ba886490*. As a result, the following command creates a mount target for the subnet the node is in. If there were more nodes in the cluster, you'd run the command once for a subnet in each AZ that you had a node in, replacing subnet-*EXAMPLEe2ba886490* with the appropriate subnet ID.

```
aws efs create-mount-target \
--file-system-id $file_system_id \
--subnet-id subnet-EXAMPLEe2ba886490 \
--security-groups $security_group_id
```

Deploy a sample application

You can deploy a sample app that dynamically creates a persistent volume, or you can manually create a persistent volume. You can replace the examples given in this section with a different application.

Dynamic

Important

You can't use dynamic provisioning with Fargate nodes.

Prerequisite

You must use version 1.2x or later of the Amazon EFS CSI driver, which requires a 1.17 or later cluster. To update your cluster, see [the section called "Updating a cluster" \(p. 34\)](#).

To deploy a sample application that uses a persistent volume that the controller creates

This procedure uses the [Dynamic Provisioning](#) example from the [Amazon EFS Container Storage Interface \(CSI\) driver](#) GitHub repository. It dynamically creates a persistent volume through [Amazon EFS access points](#) and a Persistent Volume Claim (PVC) that's consumed by a pod.

1. Create a storage class for EFS. For all parameters and configuration options, see [Amazon EFS CSI Driver](#) on GitHub.
 - a. Retrieve your Amazon EFS file system ID. You can find this in the Amazon EFS console, or use the following AWS CLI command.

```
aws efs describe-file-systems --query "FileSystems[*].FileSystemId" --output text
```

Output:

```
fs-582a03f3
```

- b. Download a StorageClass manifest for Amazon EFS.

```
curl -o storageclass.yaml https://raw.githubusercontent.com/kubernetes-sigs/aws-efs-csi-driver/master/examples/kubernetes/dynamic_provisioning/specs/storageclass.yaml
```

- c. Edit the file. Find the following line, and replace the value for `fileSystemId` with your file system ID.

```
fileSystemId: fs-582a03f3
```

- d. Deploy the storage class.

```
kubectl apply -f storageclass.yaml
```

2. Test automatic provisioning by deploying a Pod that makes use of the `PersistentVolumeClaim`:

- a. Download a manifest that deploys a pod and a PersistentVolumeClaim.

```
curl -o pod.yaml https://raw.githubusercontent.com/kubernetes-sigs/aws-efs-csi-driver/master/examples/kubernetes/dynamic_provisioning/specs/pod.yaml
```

- b. Deploy the pod with a sample app and the `PersistentVolumeClaim` used by the pod.

```
kubectl apply -f pod.yaml
```

- Determine the names of the pods running the controller.

```
kubectl get pods -n kube-system | grep efs-csi-controller
```

Example output:

efs-csi-controller-74ccf9f566-q5989	3/3	Running	0	40m
efs-csi-controller-74ccf9f566-wswg9	3/3	Running	0	40m

- After few seconds, you can observe the controller picking up the change (edited for readability). Replace **74ccf9f566-q5989** with a value from one of the pods in your output from the previous command.

```
kubectl logs efs-csi-controller-74ccf9f566-q5989 \
-n kube-system \
-c csi-provisioner \
--tail 10
```

Example output:

```
...
1 controller.go:737] successfully created PV pvc-5983ffec-96cf-40c1-9cd6-e5686ca84eca
for PVC efs-claim and csi volume name fs-95bcec92::fsap-02a88145b865d3a87
```

If you don't see the previous output, run the previous command using one of the other controller pods.

- Confirm that a persistent volume was created with a status of Bound to a PersistentVolumeClaim:

```
kubectl get pv
```

Example output:

NAME	STATUS	CLAIM	STORAGECLASS	CAPACITY	ACCESS MODES	RECLAIM POLICY
pvc-5983ffec-96cf-40c1-9cd6-e5686ca84eca	Bound	default/efs-claim	efs-sc	20Gi	RWX	Delete
					7m57s	

- View details about the PersistentVolumeClaim that was created.

```
kubectl get pvc
```

Example output:

NAME	STATUS	VOLUME	CAPACITY	ACCESS
MODES	STORAGECLASS	AGE		
efs-claim	Bound	pvc-5983ffec-96cf-40c1-9cd6-e5686ca84eca	20Gi	RWX
efs-sc		9m7s		

- View the sample app pod's status until the STATUS becomes Running.

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

Example output:

NAME	READY	STATUS	RESTARTS	AGE	IP	NOMINATED NODE	READINESS	GATES	NODE
efs-example	1/1	Running	0	10m	192.168.78.156 <i>ip-192-168-73-191.region-code.compute.internal</i>	<none>	<none>	<none>	<none>

Note

If a pod doesn't have an IP address listed, make sure that you added a mount target for the subnet that your node is in (as described at the end of [Create an Amazon EFS file system \(p. 249\)](#)). Otherwise the pod won't leave ContainerCreating status. When an IP address is listed, it may take a few minutes for a pod to reach the Running status.

8. Confirm that the data is written to the volume.

```
kubectl exec efs-app -- bash -c "cat data/out"
```

Example output:

```
...
Tue Mar 23 14:29:16 UTC 2021
Tue Mar 23 14:29:21 UTC 2021
Tue Mar 23 14:29:26 UTC 2021
Tue Mar 23 14:29:31 UTC 2021
...
```

9. (Optional) Terminate the Amazon EKS node that your pod is running on and wait for the pod to be re-scheduled. Alternately, you can delete the pod and redeploy it. Complete the previous step again, confirming that the output includes the previous output.

Static

To deploy a sample application that uses a persistent volume that you create

This procedure uses the [Multiple Pods Read Write Many](#) example from the [Amazon EFS Container Storage Interface \(CSI\) driver](#) GitHub repository to consume a statically provisioned Amazon EFS persistent volume and access it from multiple pods with the `ReadWriteMany` access mode.

1. Clone the [Amazon EFS Container Storage Interface \(CSI\) driver](#) GitHub repository to your local system.

```
git clone https://github.com/kubernetes-sigs/aws-efs-csi-driver.git
```

2. Navigate to the `multiple_pods` example directory.

```
cd aws-efs-csi-driver/examples/kubernetes/multiple_pods/
```

3. Retrieve your Amazon EFS file system ID. You can find this in the Amazon EFS console, or use the following AWS CLI command.

```
aws efs describe-file-systems --query "FileSystems[*].FileSystemId" --output text
```

Output:

```
fs-582a03f3
```

4. Edit the `specs/pv.yaml` file and replace the `volumeHandle` value with your Amazon EFS file system ID.

```

apiVersion: v1
kind: PersistentVolume
metadata:
  name: efs-pv
spec:
  capacity:
    storage: 5Gi
  volumeMode: Filesystem
  accessModes:
    - ReadWriteMany
  persistentVolumeReclaimPolicy: Retain
  storageClassName: efs-sc
  csi:
    driver: efs.csi.aws.com
  volumeHandle: fs-582a03f3

```

Note

Because Amazon EFS is an elastic file system, it doesn't enforce any file system capacity limits. The actual storage capacity value in persistent volumes and persistent volume claims isn't used when creating the file system. However, because storage capacity is a required field in Kubernetes, you must specify a valid value, such as, 5Gi in this example. This value doesn't limit the size of your Amazon EFS file system.

5. Deploy the efs-sc storage class, efs-claim persistent volume claim, and efs-pv persistent volume from the specs directory.

```

kubectl apply -f specs/pv.yaml
kubectl apply -f specs/claim.yaml
kubectl apply -f specs/storageclass.yaml

```

6. List the persistent volumes in the default namespace. Look for a persistent volume with the default/efs-claim claim.

```
kubectl get pv -w
```

Output:

NAME	CAPACITY	ACCESS MODES	RECLAIM POLICY	STATUS	CLAIM
STORAGECLASS	REASON	AGE			
efs-pv	5Gi	RWX	Retain	Bound	default/efs-claim
efs-sc				2m50s	

Don't proceed to the next step until the STATUS is Bound.

7. Deploy the app1 and app2 sample applications from the specs directory.

```

kubectl apply -f specs/pod1.yaml
kubectl apply -f specs/pod2.yaml

```

8. Watch the pods in the default namespace and wait for the app1 and app2 pods' STATUS to become Running.

```
kubectl get pods --watch
```

Note

It may take a few minutes for the pods to reach the Running status.

9. Describe the persistent volume.

```
kubectl describe pv efs-pv
```

Output:

```
Name:          efs-pv
Labels:        none
Annotations:   kubectl.kubernetes.io/last-applied-configuration:
               {"apiVersion":"v1","kind":"PersistentVolume","metadata": {"annotations":{}, "name":"efs-pv"}, "spec":{"accessModes":["ReadWriteMany"]}, "capacity": {"pv.kubernetes.io/bound-by-controller": "yes"}, "finalizers": ["kubernetes.io/pv-protection"], "storageClass": "efs-sc", "status": "Bound", "claim": "default/efs-claim", "reclaimPolicy": "Retain", "accessModes": ["RWX"], "volumeMode": "Filesystem", "capacity": "5Gi", "nodeAffinity": "none", "message": "", "source": {"type": "CSI (a Container Storage Interface (CSI) volume source)", "driver": "efs.csi.aws.com", "volumeHandle": "fs-582a03f3", "readOnly": "false", "volumeAttributes": "none"}, "events": "none"}
```

The Amazon EFS file system ID is listed as the `VolumeHandle`.

10. Verify that the `app1` pod is successfully writing data to the volume.

```
kubectl exec -ti app1 -- tail /data/out1.txt
```

Output:

```
...
Mon Mar 22 18:18:22 UTC 2021
Mon Mar 22 18:18:27 UTC 2021
Mon Mar 22 18:18:32 UTC 2021
Mon Mar 22 18:18:37 UTC 2021
...
```

11. Verify that the `app2` pod shows the same data in the volume that `app1` wrote to the volume.

```
kubectl exec -ti app2 -- tail /data/out1.txt
```

Output:

```
...
Mon Mar 22 18:18:22 UTC 2021
Mon Mar 22 18:18:27 UTC 2021
Mon Mar 22 18:18:32 UTC 2021
Mon Mar 22 18:18:37 UTC 2021
...
```

12. When you finish experimenting, delete the resources for this sample application to clean up.

```
kubectl delete -f specs/
```

You can also manually delete the file system and security group that you created.

Amazon FSx for Lustre CSI driver

The [FSx for Lustre Container Storage Interface \(CSI\) driver](#) provides a CSI interface that allows Amazon EKS clusters to manage the lifecycle of FSx for Lustre file systems.

This topic shows you how to deploy the FSx for Lustre CSI Driver to your Amazon EKS cluster and verify that it works. We recommend using version 0.4.0 of the driver.

Note

This driver is supported on Kubernetes version 1.22 and later Amazon EKS clusters and nodes. The driver isn't supported on Fargate. Alpha features of the FSx for Lustre CSI Driver aren't supported on Amazon EKS clusters. The driver is in Beta release. It is well tested and supported by Amazon EKS for production use. Support for the driver will not be dropped, though details may change. If the schema or schematics of the driver changes, instructions for migrating to the next version will be provided.

For detailed descriptions of the available parameters and complete examples that demonstrate the driver's features, see the [FSx for Lustre Container Storage Interface \(CSI\) driver](#) project on GitHub.

Prerequisites

You must have:

- 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 existing Amazon EKS cluster. To deploy one, see [Getting started with Amazon EKS \(p. 4\)](#).
- An existing AWS Identity and Access Management (IAM) OpenID Connect (OIDC) provider for your cluster. To determine whether you already have one, or to create one, see [Create an IAM OIDC provider for your cluster \(p. 446\)](#).
- Version 0.97.0 or later of the eksctl command line tool installed on your computer or AWS CloudShell. To install or update eksctl, see [Installing eksctl \(p. 10\)](#).
- The kubectl command line tool installed on your computer or AWS CloudShell. The version must be the same, or up to two versions later than your cluster version. To install or upgrade kubectl, see [Installing kubectl \(p. 4\)](#).

To deploy the FSx for Lustre CSI driver to an Amazon EKS cluster

1. Create an Amazon S3 bucket that you will use to store data. Replace **fsx-csi-bucket** with a unique name.

```
aws s3 mb s3://fsx-csi-bucket
```

2. Create an IAM policy and service account that allows the driver to make calls to AWS APIs on your behalf.
 - a. Copy the following text and save it to a file named **fsx-csi-driver.json**. Replace **fsx-csi-bucket** with the bucket that you created earlier. If your cluster is in the AWS GovCloud (US-East) or AWS GovCloud (US-East) AWS Regions, then replace **arn:aws:** with **arn:aws-us-gov::**.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "iam:CreateServiceLinkedRole",
                "iam:AttachRolePolicy",
                "iam:PutRolePolicy"
            ],
            "Resource": "arn:aws:iam::*:role/aws-service-role/s3.data-source.lustre.fsx.amazonaws.com/*"
        },
        {
            "Action": "iam:CreateServiceLinkedRole",
            "Effect": "Allow",
            "Resource": "*",
            "Condition": {
                "StringLike": {
                    "iam:AWSServiceName": [
                        "fsx.amazonaws.com"
                    ]
                }
            }
        },
        {
            "Effect": "Allow",
            "Action": [
                "s3>ListBucket",
                "fsx>CreateFileSystem",
                "fsx>DeleteFileSystem",
                "fsx>DescribeFileSystems",
                "fsx>TagResource"
            ],
            "Resource": [
                "arn:aws:s3:::fsx-csi-bucket/*",
                "arn:aws:s3:::fsx-csi-bucket"
            ]
        }
    ]
}
```

- b. Create the policy. You can replace *Amazon_FSx_Lustre_CSI_Driver* with a different name.

```
aws iam create-policy \
--policy-name Amazon_FSx_Lustre_CSI_Driver \
--policy-document file://fsx-csi-driver.json
```

Take note of the policy Amazon Resource Name (ARN) that's returned. It looks similar to the following.

```
"Arn": "arn:aws:iam::111122223333:role/AmazonEKSFSxLustreCSIDriver",
```

3. Create a Kubernetes service account for the driver and attach the policy to the service account. Replacing the ARN of the policy with the ARN returned in the previous step. If your cluster is in the AWS GovCloud (US-East) or AWS GovCloud (US-East) AWS Regions, then replace `arn:aws:` with `arn:aws-us-gov:` before running the following command.

```
eksctl create iamserviceaccount \
--name fsx-csi-controller-sa \
```

```
--namespace kube-system \
--cluster my-cluster \
--attach-policy-arn arn:aws:iam::111122223333:policy/Amazon_FSx_Lustre_CSI_Driver \
--approve \
--role-name AmazonEKSFSxLustreCSIDriver \
--region region-code
```

Example output:

You'll see several lines of output as the service account is created. The last lines of output are similar to the following.

```
[#] 1 task: {
  2 sequential sub-tasks: {
    create IAM role for serviceaccount "kube-system/fsx-csi-controller-sa",
    create serviceaccount "kube-system/fsx-csi-controller-sa",
  }
[#] building iamserviceaccount stack "eksctl-my-cluster-addon-iamserviceaccount-kube-
system-fsx-csi-controller-sa"
[#] deploying stack "eksctl-my-cluster-addon-iamserviceaccount-kube-system-fsx-csi-
controller-sa"
[#] waiting for CloudFormation stack "eksctl-my-cluster-addon-iamserviceaccount-kube-
system-fsx-csi-controller-sa"
[#] created serviceaccount "kube-system/fsx-csi-controller-sa"
```

Note the name of the AWS CloudFormation stack that was deployed. In the example output above, the stack is named **eksctl-my-cluster-addon-iamserviceaccount-kube-system-fsx-csi-controller-sa**.

4. Deploy the driver with the following command.

Note

You can view the content being applied in [aws-fsx-csi-driver](#) on GitHub.

```
kubectl apply -k "github.com/kubernetes-sigs/aws-fsx-csi-driver/deploy/kubernetes/
overlays/stable/?ref=master"
```

Example output:

```
serviceaccount/fsx-csi-controller-sa created
serviceaccount/fsx-csi-node-sa created
clusterrole.rbac.authorization.k8s.io/fsx-csi-external-provisioner-role created
clusterrole.rbac.authorization.k8s.io/fsx-external-resizer-role created
clusterrolebinding.rbac.authorization.k8s.io/fsx-csi-external-provisioner-binding
created
clusterrolebinding.rbac.authorization.k8s.io/fsx-csi-resizer-binding created
deployment.apps/fsx-csi-controller created
daemonset.apps/fsx-csi-node created
csidriver.storage.k8s.io/fsx.csi.aws.com created
```

5. Note the ARN for the role that was created. If you didn't note it earlier and don't have it available anymore in the AWS CLI output, you can do the following to see it in the AWS Management Console.
 - a. Open the AWS CloudFormation console at <https://console.aws.amazon.com/cloudformation>.
 - b. Ensure that the console is set to the AWS Region that you created your IAM role in and then select **Stacks**.
 - c. Select the stack named **eksctl-my-cluster-addon-iamserviceaccount-kube-
system-fsx-csi-controller-sa**.
 - d. Select the **Outputs** tab. The **Role ARN** is listed on the **Outputs (1)** page.

6. Patch the driver deployment to add the service account that you created earlier, replacing the ARN with the ARN that you noted. If your cluster is in the AWS GovCloud (US-East) or AWS GovCloud (US-East) AWS Regions, then replace `arn:aws:` with `arn:aws-us-gov:` before running the following command.

```
kubectl annotate serviceaccount -n kube-system fsx-csi-controller-sa \
eks.amazonaws.com/role-arn=arn:aws:iam::111122223333:role/AmazonEKSFSxLustreCSIDriver
--overwrite=true
```

Example output:

```
serviceaccount/fsx-csi-controller-sa annotated
```

To deploy a Kubernetes storage class, persistent volume claim, and sample application to verify that the CSI driver is working

This procedure uses the [Dynamic volume provisioning for Amazon S3](#) from the [FSx for Lustre Container Storage Interface \(CSI\)](#) driver GitHub repository to consume a dynamically-provisioned FSx for Lustre volume.

1. Create a testfile in an export folder within a bucket. Replace `fsx-csi-bucket` with the bucket that you created earlier.

```
echo test-file >> testfile
aws s3 cp testfile s3://fsx-csi-bucket/export/testfile
```

Example output:

```
upload: ./testfile to s3://fsx-csi-bucket/export/testfile
```

2. Download the storageclass manifest with the following command.

```
curl -o storageclass.yaml https://raw.githubusercontent.com/kubernetes-sigs/aws-fsx-
csi-driver/master/examples/kubernetes/dynamic_provisioning_s3/specs/storageclass.yaml
```

3. Edit the parameters section of the `storageclass.yaml` file. Replace every `example-value` with your own values.

```
parameters:
  subnetId: subnet-0d7b5e117ad7b4961
  securityGroupIds: sg-05a37bfe01467059a
  s3ImportPath: s3://fsx-csi-bucket
  s3ExportPath: s3://fsx-csi-bucket/export
  deploymentType: SCRATCH_2
```

- **subnetId** – The subnet ID that the Amazon FSx for Lustre file system should be created in. Amazon FSx for Lustre isn't supported in all Availability Zones. Open the Amazon FSx for Lustre console at <https://console.aws.amazon.com/fsx/> to confirm that the subnet that you want to use is in a supported Availability Zone. The subnet can include your nodes, or can be a different subnet or VPC:
 - You can check for the subnets for your cluster in the AWS Management Console under the [Networking](#) section for the cluster.
 - If the subnet that you specify isn't the same subnet that you have nodes in, then your VPCs must be [connected](#), and you must ensure that you have the necessary ports open in your security groups.

- **securityGroupIds** – The security group ID for your nodes. You can check for a security group for your cluster in the AWS Management Console under the **Networking** section for the cluster. Or, you can do this using the following AWS CLI command. When using this command, replace *my-cluster* with the name of your cluster.

```
aws eks describe-cluster --name my-cluster --query
cluster.resourcesVpcConfig.clusterSecurityGroupId
```

Note

The security groups must allow inbound/outbound access to Lustre ports 988 and 1021–1023. For more information, see [Lustre Client VPC Security Group Rules](#) in the Amazon FSx for Lustre User Guide.

- **s3ImportPath** – The Amazon Simple Storage Service data repository that you want to copy data from to the persistent volume. Specify the *fsx-csi-bucket* that you created earlier.
- **s3ExportPath** – The Amazon S3 data repository that you want to export new or modified files to. Specify the *fsx-csi-bucket/export* folder that you created earlier.
- **deploymentType** – The file system deployment type. Valid values are SCRATCH_1, SCRATCH_2, PERSISTENT_1, and PERSISTENT_2. For more information about deployment types, see [Create your Amazon FSx for Lustre file system](#).

Note

The Amazon S3 bucket for `s3ImportPath` and `s3ExportPath` must be the same, otherwise the driver cannot create the FSx for Lustre file system. The `s3ImportPath` can stand alone. A random path will be created automatically like `s3://fsx-csi-bucket/FSxLustre20190308T012310Z`. The `s3ExportPath` cannot be used without specifying a value for `S3ImportPath`.

4. Create the storageclass.

```
kubectl apply -f storageclass.yaml
```

Example output:

```
storageclass.storage.k8s.io/fsx-sc created
```

5. Download the persistent volume claim manifest.

```
curl -o claim.yaml https://raw.githubusercontent.com/kubernetes-sigs/aws-fsx-csi-driver/master/examples/kubernetes/dynamic_provisioning_s3/specs/claim.yaml
```

6. (Optional) Edit the `claim.yaml` file. Change `1200Gi` to one of the increment values listed below, based on your storage requirements and the `deploymentType` that you selected in a previous step.

```
storage: 1200Gi
```

- SCRATCH_2 and PERSISTENT – 1.2 TiB, 2.4 TiB, or increments of 2.4 TiB over 2.4 TiB.
- SCRATCH_1 – 1.2 TiB, 2.4 TiB, 3.6 TiB, or increments of 3.6 TiB over 3.6 TiB.

7. Create the persistent volume claim.

```
kubectl apply -f claim.yaml
```

Example output:

```
persistentvolumeclaim/fsx-claim created
```

8. Confirm that the file system is provisioned.

```
kubectl get pvc
```

Example output:

NAME	STATUS	VOLUME	CAPACITY	ACCESS MODES
STORAGECLASS	AGE			
fsx-claim	Bound	pvc-15dad3c1-2365-11ea-a836-02468c18769e	1200Gi	RWX
fsx-sc	7m37s			

Note

The STATUS may show as Pending for 5-10 minutes, before changing to Bound. Don't continue with the next step until the STATUS is Bound.

9. Deploy the sample application.

```
kubectl apply -f https://raw.githubusercontent.com/kubernetes-sigs/aws-fsx-csi-driver/master/examples/kubernetes/dynamic_provisioning_s3/specs/pod.yaml
```

10. Verify that the sample application is running.

```
kubectl get pods
```

Example output:

NAME	READY	STATUS	RESTARTS	AGE
fsx-app	1/1	Running	0	8s

Access Amazon S3 files from the FSx for Lustre file system

If you only want to import data and read it without any modification and creation, then you don't need a value for `s3ExportPath` in your `storageclass.yaml` file. Verify that data was written to the FSx for Lustre file system by the sample app.

```
kubectl exec -it fsx-app ls /data
```

Example output:

```
export out.txt
```

The sample app wrote the `out.txt` file to the file system.

Archive files to the `s3ExportPath`

For new files and modified files, you can use the Lustre user space tool to archive the data back to Amazon S3 using the value that you specified for `s3ExportPath`.

1. Export the file back to Amazon S3.

```
kubectl exec -ti fsx-app -- lfs hsm_archive /data/out.txt
```

Note

- New files aren't synced back to Amazon S3 automatically. In order to sync files to the `s3ExportPath`, you need to [install the Lustre client](#) in your container image and manually run the `lfs hsm_archive` command. The container should run in privileged mode with the `CAP_SYS_ADMIN` capability.
 - This example uses a lifecycle hook to install the Lustre client for demonstration purpose. A normal approach is building a container image with the Lustre client.
2. Confirm that the `out.txt` file was written to the `s3ExportPath` folder in Amazon S3.

```
aws s3 ls fsx-csi-bucket/export/
```

Example output:

```
2019-12-23 12:11:35      4553 out.txt
2019-12-23 11:41:21      10 testfile
```

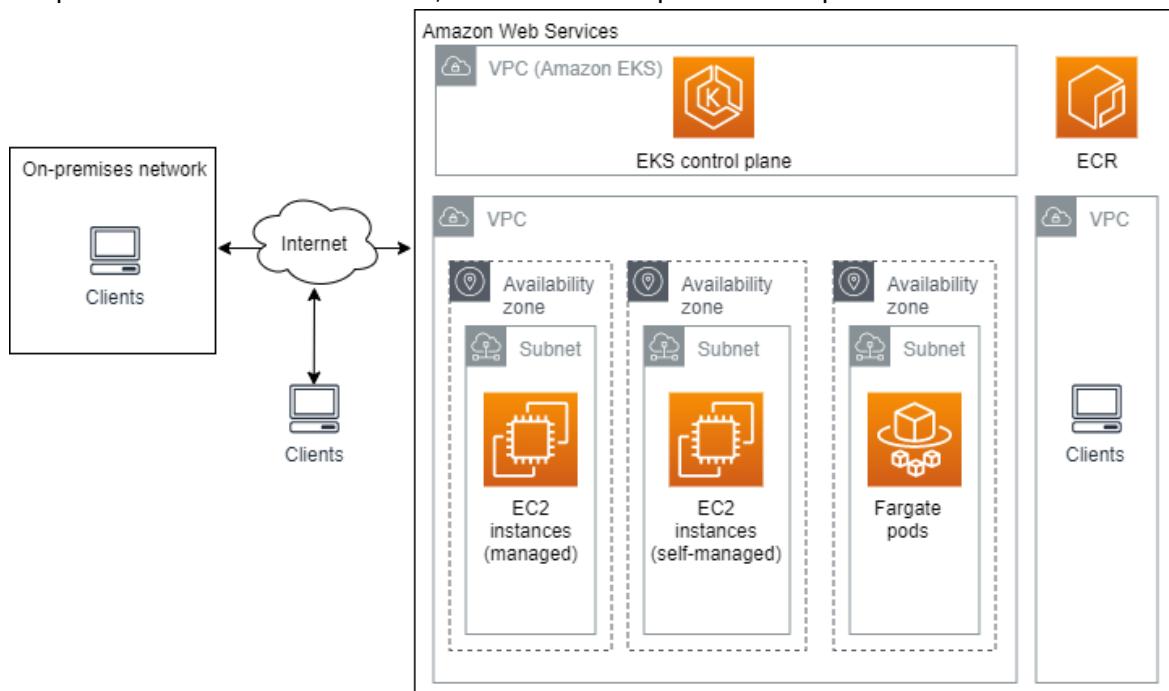
Amazon FSx for NetApp ONTAP CSI driver

NetApp's Astra Trident provides dynamic storage orchestration using a Container Storage Interface (CSI) compliant driver. This allows Amazon EKS clusters to manage the lifecycle of persistent volumes (PVs) backed by Amazon FSx for NetApp ONTAP file systems. To get started, see [Use Astra Trident with Amazon FSx for NetApp ONTAP](#) in the Astra Trident documentation.

Amazon FSx for NetApp ONTAP is a storage service that allows you to launch and run fully managed ONTAP file systems in the cloud. ONTAP is NetApp's file system technology that provides a widely adopted set of data access and data management capabilities. Amazon FSx for NetApp ONTAP provides the features, performance, and APIs of on-premises NetApp file systems with the agility, scalability, and simplicity of a fully managed AWS service. For more information, see the [FSx for ONTAP User Guide](#).

Amazon EKS networking

This chapter provides an overview of Amazon EKS networking. The following diagram shows key components of an Amazon EKS cluster, and the relationship of these components to a VPC.



The following explanations help you understand how components of the diagram relate to each other and which topics in this guide and other AWS guides that you can reference for more information.

- **Amazon VPC and subnets** – All Amazon EKS resources are deployed to one AWS Region in an existing subnet in an existing VPC. For more information, see [VPCs and subnets](#) in the Amazon VPC User Guide. Each subnet exists in one Availability Zone. The VPC and subnets must meet requirements such as the following:
 - VPCs and subnets must be tagged appropriately, so that Kubernetes knows that it can use them for deploying resources, such as load balancers. For more information, see [Subnet tagging \(p. 272\)](#). If you deploy the VPC using an Amazon EKS provided [AWS CloudFormation template \(p. 266\)](#) or using `eksctl`, then the VPC and subnets are tagged appropriately for you.
 - A subnet may or may not have internet access. If a subnet does not have internet access, the pods deployed within it must be able to access other AWS services, such as Amazon ECR, to pull container images. For more information about using subnets that don't have internet access, see [Private clusters \(p. 102\)](#).
 - Any public subnets that you use must be configured to auto-assign public IPv4 addresses or IPv6 addresses for Amazon EC2 instances launched within them. For more information, see [VPC IP addressing \(p. 270\)](#).
 - If using IPv6, each subnet must be configured to auto-assign IPv6 addresses. For more information, see [Modify the IPv6 addressing attribute for your subnet](#) in the Amazon VPC User Guide.
 - The nodes and control plane must be able to communicate over all ports through appropriately tagged [security groups](#). For more information, see [Amazon EKS security group considerations \(p. 272\)](#).

- You can implement a network segmentation and tenant isolation network policy. Network policies are similar to AWS security groups in that you can create network ingress and egress rules. Instead of assigning instances to a security group, you assign network policies to pods using pod selectors and labels. For more information, see [Installing the Calico add-on \(p. 350\)](#).

You can deploy a VPC and subnets that meet the Amazon EKS requirements through manual configuration, or by deploying the VPC and subnets using [eksctl \(p. 10\)](#), or an Amazon EKS provided AWS CloudFormation template. Both eksctl and the AWS CloudFormation template create the VPC and subnets with the required configuration. For more information, see [Creating a VPC for your Amazon EKS cluster \(p. 266\)](#).

- **Amazon EKS control plane** – Deployed and managed by Amazon EKS in an Amazon EKS managed VPC. When you create the cluster, Amazon EKS creates and manages network interfaces in your account that have `Amazon EKS <cluster name>` in their description. These network interfaces allow AWS Fargate and Amazon EC2 instances to communicate with the control plane.

By default, the control plane exposes a public endpoint so that clients and nodes can communicate with the cluster. You can limit the internet client source IP addresses that can communicate with the public endpoint. Alternatively, you can enable a private endpoint and disable the public endpoint or enable both the public and private endpoints. To learn more about cluster endpoints, see [Amazon EKS cluster endpoint access control \(p. 47\)](#).

Clients in your on-premises network or other VPCs can communicate with the public or private-only endpoint, if you've configured connectivity between the VPC that the cluster is deployed to and the other networks. For more information about connecting your VPC to other networks, see the [AWS Network-to-Amazon VPC connectivity options](#) and [Amazon VPC-to-Amazon VPC connectivity options](#) technical papers.

- **Amazon EC2 instances** – Each Amazon EC2 node is deployed to one subnet. Each node is assigned a [private IP address](#) from a CIDR block assigned to the subnet. If the subnets were created using one of the [Amazon EKS provided AWS CloudFormation templates \(p. 266\)](#), then nodes deployed to public subnets are automatically assigned a [public IPv4 address](#) by the subnet. Each node is deployed with the Amazon VPC CNI add-on by default. The add-on assigns each pod a private IP address from the CIDR block assigned to the subnet that the node is in and adds an IPv4 address as a secondary IP address to one of the [network interfaces](#) attached to the instance. This AWS resource is referred to as a *network interface* in the AWS Management Console and the Amazon EC2 API. Therefore, we use "network interface" in this documentation instead of "elastic network interface". The term "network interface" in this documentation always means "elastic network interface".

You can change this behavior by assigning additional IPv4 CIDR blocks to your VPC and enabling [CNI custom networking \(p. 306\)](#), which assigns IP addresses to pods from different subnets than the node is deployed to. To use custom networking, you must enable it when you launch your nodes. You can also associate unique security groups with some of the pods running on many Amazon EC2 instance types. For more information, see [Security groups for pods \(p. 313\)](#).

By default, the source IPv4 address of each pod that communicates with resources outside of the VPC is translated through network address translation (NAT) to the primary IP address of the primary network interface attached to the node. You can change this behavior to instead have a NAT device in a private subnet translate each pod's IPv4 address to the NAT device's IPv4 address. For more information, see [External source network address translation \(SNAT\) \(p. 305\)](#).

If your instance is deployed to a cluster that uses the IPv6 family, you must assign an IPv6 CIDR block to your VPC and subnets. Outbound IPv6 traffic is not network address translated. For more information about using IPv6 with your cluster, see [Assigning IPv6 addresses to pods and services \(p. 294\)](#).

- **Fargate pods** – Deployed to private subnets only. Each pod is assigned a private IPv4 (and optionally, an IPv6) address from the CIDR block assigned to the subnet. Fargate does not support all pod networking options. For more information, see [AWS Fargate considerations \(p. 154\)](#).

Creating a VPC for your Amazon EKS cluster

Amazon Virtual Private Cloud (Amazon VPC) enables you to launch AWS resources into a virtual network that you've defined. This virtual network closely resembles a traditional network that you'd operate in your own data center, with the benefits of using the scalable infrastructure of AWS. For more information, see the [Amazon VPC User Guide](#) and [De-mystifying cluster networking for Amazon EKS nodes](#).

If you want to use an existing VPC, then it must meet specific requirements for use with Amazon EKS. For more information, see [Cluster VPC and subnet considerations \(p. 269\)](#). This topic guides you through creating a VPC for your cluster using one of the following configurations:

- **Public and private subnets** – This VPC has two public and two private [subnets](#). One public and one private subnet are deployed to the same [Availability Zone](#). The other public and private subnets are deployed to a second Availability Zone in the same AWS Region. We recommend this option for all production deployments. This option allows you to deploy your nodes to private subnets and allows Kubernetes to deploy load balancers to the public subnets that can load balance traffic to pods running on nodes in the private subnets.

[Public IPv4 addresses](#) are automatically assigned to nodes deployed to public subnets, but public IPv4 addresses are not assigned to nodes deployed to private subnets. If you choose, you can also assign IPv6 addresses to nodes in public and private subnets. The nodes in private subnets can communicate with the cluster and other AWS services, and pods can communicate outbound to the internet through a [NAT gateway](#) (IPv4) or [egress-only Internet gateway](#) (IPv6) that is deployed in each Availability Zone. A [security group](#) is deployed that denies all inbound traffic and allows all outbound traffic. The subnets are tagged so that Kubernetes is able to deploy load balancers to them. For more information about subnet tagging, see [Subnet tagging \(p. 272\)](#).

For more information about this type of VPC, see [VPC with public and private subnets \(NAT\)](#).

- **Only public subnets** – This VPC has three public subnets that are deployed into different Availability Zones in the region. All nodes are automatically assigned public IPv4 addresses and can send and receive internet traffic through an internet gateway. A [security group](#) is deployed that denies all inbound traffic and allows all outbound traffic. The subnets are tagged so that Kubernetes can deploy load balancers to them. For more information about subnet tagging, see [Subnet tagging \(p. 272\)](#). For more information about this type of VPC, see [VPC with a single public subnet](#).
- **Only private subnets** – This VPC has three private subnets that are deployed into different Availability Zones in the AWS Region. All nodes can optionally send and receive internet traffic through a NAT instance or NAT gateway. A [security group](#) is deployed that denies all inbound traffic and allows all outbound traffic. The subnets are tagged so that Kubernetes can deploy internal load balancers to them. For more information about subnet tagging, see [Subnet tagging \(p. 272\)](#). For more information about this type of VPC, see [VPC with a private subnet only and AWS Site-to-Site VPN access](#).

Important

There are additional requirements if the VPC does not have outbound internet access, such as via a NAT Instance, NAT Gateway, Egress-only internet gateway, VPN, or Direct Connect. You must bypass the EKS cluster introspection by providing the cluster certificate authority and cluster API endpoint to the nodes. You also may need to configure VPC endpoints listed in [Modifying cluster endpoint access \(p. 48\)](#).

Important

If you deployed a VPC using `eksctl` or by using either of the Amazon EKS AWS CloudFormation VPC templates:

- On or after March 26, 2020 – Public IPv4 addresses are automatically assigned by public subnets to new nodes deployed to public subnets.

- Before March 26, 2020 – Public IPv4 addresses are not automatically assigned by public subnets to new nodes deployed to public subnets.

This change impacts new node groups deployed to public subnets in the following ways:

- [Managed node groups \(p. 112\)](#) – If the node group is deployed to a public subnet on or after April 22, 2020, the public subnet must have automatic assignment of public IP addresses enabled. For more information, see [Modifying the public IPv4 addressing attribute for your subnet](#).
- [Linux \(p. 133\)](#), [Windows \(p. 140\)](#), or [Arm \(p. 186\)](#) self-managed node groups – If the node group is deployed to a public subnet on or after March 26, 2020, the public subnet must have automatic assignment of public IP addresses enabled or the nodes must be launched with a public IP address. For more information, see [Modifying the public IPv4 addressing attribute for your subnet](#) or [Assigning a public IPv4 address during instance launch](#).

Prerequisite

You must have the necessary IAM permissions to create Amazon VPC resources such as VPCs, subnets, security groups, route tables and routes, and internet and NAT gateways. For more information, see [Create a VPC with a public subnet example policy](#) in the Amazon VPC User Guide and the full list of Actions, resources, and condition keys for Amazon EC2 in the [Service Authorization Reference](#).

Creating a VPC for your Amazon EKS cluster

You can create a VPC with public and private subnets, only public subnets, or only private subnets. Select the tab with the description of the type of VPC that you'd like to create.

Public and private subnets

To create your cluster VPC with public and private subnets

1. Open the AWS CloudFormation console at <https://console.aws.amazon.com/cloudformation>.
 2. From the navigation bar, select an AWS Region that supports Amazon EKS.
 3. Choose **Create stack, With new resources (standard)**.
 4. Under **Prerequisite - Prepare template**, make sure that **Template is ready** is selected and then under **Specify template**, select **Amazon S3 URL**.
 5. You can create an IPv4 network or an IPv6 network. Paste one of the following URLs into the text area under **Amazon S3 URL** and choose **Next**:
 - IPv4

`https://s3.us-west-2.amazonaws.com/amazon-eks/clouformation/2020-10-29/amazon-eks-vpc-private-subnets.yaml`

 - IPv6

`https://s3.us-west-2.amazonaws.com/amazon-eks/clouformation/2020-10-29/amazon-eks-ipv6-vpc-public-private-subnets.yaml`
6. On the **Specify stack details** page, fill out the parameters accordingly, and then choose **Next**.
 - **Stack name:** Choose a stack name for your AWS CloudFormation stack. For example, you can call it **eks-vpc**. The 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.

- **VpcBlock:** Choose an IPv4 CIDR range for your VPC. Each node, pod, and load balancer that you deploy is assigned an IPv4 address from this block. The default IPv4 values provide enough IP addresses for most implementations, but if it doesn't, then you can change it. For more information, see [VPC and subnet sizing](#) in the Amazon VPC User Guide. You can also add additional CIDR blocks to the VPC once it's created. If you're creating an IPv6 VPC, IPv6 CIDR ranges are automatically assigned for you from Amazon's Global Unicast Address space.
 - **PublicSubnet01Block:** Specify an IPv4 CIDR block for public subnet 1. The default value provides enough IP addresses for most implementations, but if it doesn't, then you can change it.. If you're creating an IPv6 VPC, this block is specified for you within the template.
 - **PublicSubnet02Block:** Specify an IPv4 CIDR block for public subnet 2. The default value provides enough IP addresses for most implementations, but if it doesn't, then you can change it. If you're creating an IPv6 VPC, this block is specified for you within the template.
 - **PrivateSubnet01Block:** Specify an IPv4 CIDR block for private subnet 1. The default value provides enough IP addresses for most implementations, but if it doesn't, then you can change it. If you're creating an IPv6 VPC, this block is specified for you within the template.
 - **PrivateSubnet02Block:** Specify an IPv4 CIDR block for private subnet 2. The default value provides enough IP addresses for most implementations, but if it doesn't, then you can change it. If you're creating an IPv6 VPC, this block is specified for you within the template.
7. (Optional) On the **Configure stack options** page, tag your stack resources and then choose **Next**.
 8. On the **Review** page, choose **Create stack**.
 9. When your stack is created, select it in the console and choose **Outputs**.
 10. Record the **SecurityGroups** value for the security group that was created. When you add nodes to your cluster, you must specify the ID of the security group. The security group is applied to the elastic network interfaces that are created by Amazon EKS in your subnets that allows the control plane to communicate with your nodes. These network interfaces have **Amazon EKS cluster name** in their description.
 11. Record the **VpcId** for the VPC that was created. You need this when you launch your node group template.
 12. Record the **SubnetIds** for the subnets that were created and whether you created them as public or private subnets. When you add nodes to your cluster, you must specify the IDs of the subnets that you want to launch the nodes into.
 13. If you created an IPv4 VPC, skip this step. If you created an IPv6 VPC, then you must enable the auto-assign IPv6 address option for the public subnets that were created by the template. That setting is already enabled for the private subnets. To enable the setting, complete the following steps.
 - a. Open the Amazon VPC console at <https://console.aws.amazon.com/vpc/>.
 - b. In the left navigation pane, choose **Subnets**
 - c. Select one of your public subnets (**stack-name/SubnetPublic01** or **stack-name/SubnetPublic02** contains the word **public**) and choose **Actions, Edit subnet settings**.
 - d. Choose the **Enable auto-assign IPv6 address** check box and then choose **Save**.
 - e. Complete the previous steps again for your other public subnet.

Only public subnets

To create your cluster VPC with only public subnets

1. Open the AWS CloudFormation console at <https://console.aws.amazon.com/cloudformation>.
2. From the navigation bar, select an AWS Region that supports Amazon EKS.
3. Choose **Create stack, With new resources (standard)**.

4. Under **Prepare template**, make sure that **Template is ready** is selected and then under **Template source**, select **Amazon S3 URL**.
5. Paste the following URL into the text area under **Amazon S3 URL** and choose **Next**:

```
https://s3.us-west-2.amazonaws.com/amazon-eks/cloudformation/2020-10-29/amazon-eks-vpc-sample.yaml
```

6. On the **Specify Details** page, fill out the parameters accordingly, and then choose **Next**.
 - **Stack name:** Choose a stack name for your AWS CloudFormation stack. For example, you can call it **eks-vpc**. The 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.
 - **VpcBlock:** Choose a CIDR block for your VPC. Each node, pod, and load balancer that you deploy is assigned an IPv4 address from this block. The default IPv4 values provide enough IP addresses for most implementations, but if it doesn't, then you can change it. For more information, see [VPC and subnet sizing](#) in the Amazon VPC User Guide. You can also add additional CIDR blocks to the VPC once it's created.
 - **Subnet01Block:** Specify a CIDR block for subnet 1. The default value provides enough IP addresses for most implementations, but if it doesn't, then you can change it.
 - **Subnet02Block:** Specify a CIDR block for subnet 2. The default value provides enough IP addresses for most implementations, but if it doesn't, then you can change it.
 - **Subnet03Block:** Specify a CIDR block for subnet 3. The default value provides enough IP addresses for most implementations, but if it doesn't, then you can change it.
7. (Optional) On the **Options** page, tag your stack resources. Choose **Next**.
8. On the **Review** page, choose **Create**.
9. When your stack is created, select it in the console and choose **Outputs**.
10. Record the **SecurityGroups** value for the security group that was created. When you add nodes to your cluster, you must specify the ID of the security group. The security group is applied to the elastic network interfaces that are created by Amazon EKS in your subnets that allows the control plane to communicate with your nodes. These network interfaces have **Amazon EKS cluster name** in their description.
11. Record the **VpcId** for the VPC that was created. You need this when you launch your node group template.
12. Record the **SubnetIds** for the subnets that were created. When you add nodes to your cluster, you must specify the IDs of the subnets that you want to launch the nodes into.
13. (Optional) Any cluster that you deploy to this VPC is able to assign private IPv4 addresses to your pods and services. If you want any clusters deployed to this VPC to assign private IPv6 addresses to your pods and services, then you must make updates to your VPC, subnet, route tables, and security groups. For more information, see [Migrate existing VPCs from IPv4 to IPv6](#) in the Amazon VPC User Guide. Amazon EKS requires that your subnets have the **Auto-assign IPv6 addresses** option enabled (it's disabled by default).

Only private subnets

To create your cluster VPC with only private subnets

1. Open the AWS CloudFormation console at <https://console.aws.amazon.com/cloudformation>.
2. From the navigation bar, select an AWS Region that supports Amazon EKS.
3. Choose **Create stack, With new resources (standard)**.
4. Under **Prepare template**, make sure that **Template is ready** is selected and then under **Template source**, select **Amazon S3 URL**.
5. Paste the following URL into the text area under **Amazon S3 URL** and choose **Next**:

```
https://s3.us-west-2.amazonaws.com/amazon-eks/cloudformation/2020-10-29/amazon-eks-fully-private-vpc.yaml
```

6. On the **Specify Details** page, fill out the parameters accordingly, and then choose **Next**.
 - **Stack name:** Choose a stack name for your AWS CloudFormation stack. For example, you can call it **eks-vpc**. The 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.
 - **VpcBlock:** Choose a CIDR block for your VPC. Each node, pod, and load balancer that you deploy is assigned an IPv4 address from this block. The default IPv4 values provide enough IP addresses for most implementations, but if it doesn't, then you can change it. For more information, see [VPC and subnet sizing](#) in the Amazon VPC User Guide. You can also add additional CIDR blocks to the VPC once it's created.
 - **PrivateSubnet01Block:** Specify a CIDR block for subnet 1. The default value provides enough IP addresses for most implementations, but if it doesn't, then you can change it.
 - **PrivateSubnet02Block:** Specify a CIDR block for subnet 2. The default value provides enough IP addresses for most implementations, but if it doesn't, then you can change it.
 - **PrivateSubnet03Block:** Specify a CIDR block for subnet 3. The default value provides enough IP addresses for most implementations, but if it doesn't, then you can change it.
7. (Optional) On the **Options** page, tag your stack resources. Choose **Next**.
8. On the **Review** page, choose **Create**.
9. When your stack is created, select it in the console and choose **Outputs**.
10. Record the **SecurityGroups** value for the security group that was created. When you add nodes to your cluster, you must specify the ID of the security group. The security group is applied to the elastic network interfaces that Amazon EKS creates in your subnets to allow the control plane to communicate with your nodes. These network interfaces have **Amazon EKS cluster name** in their description.
11. Record the **VpcId** for the VPC that was created. You need this when you launch your node group template.
12. Record the **SubnetIds** for the subnets that were created. When you add nodes to your cluster, you must specify the IDs of the subnets that you want to launch the nodes into.
13. (Optional) Any cluster that you deploy to this VPC is able to assign private IPv4 addresses to your pods and services. If you want any clusters deployed to this VPC to assign private IPv6 addresses to your pods and services, then you must make updates to your VPC, subnet, route tables, and security groups. For more information, see [Migrate existing VPCs from IPv4 to IPv6](#) in the Amazon VPC User Guide. Amazon EKS requires that your subnets have the Auto-assign IPv6 addresses option enabled (it's disabled by default).

Cluster VPC and subnet considerations

Amazon EKS recommends running a cluster in a VPC with public and private subnets so that Kubernetes can create public load balancers in the public subnets that load balance traffic to pods running on nodes that are in private subnets. This configuration is not required, however. You can run a cluster in a VPC with only private or only public subnets, depending on your networking and security requirements. For more information about clusters deployed to a VPC with only private subnets, see [Private clusters \(p. 102\)](#).

When you create an Amazon EKS cluster, you specify the VPC subnets where Amazon EKS can place Elastic network interfaces. Amazon EKS requires subnets in at least two Availability Zone, and creates up to four network interfaces across these subnets to facilitate control plane communication to your nodes. This communication channel supports Kubernetes functionality such as `kubectl exec` and `kubectl logs`. The Amazon EKS created [cluster security group \(p. 272\)](#) and any additional security groups that

you specify when you create your cluster are applied to these network interfaces. Each Amazon EKS created network interface has Amazon EKS *cluster name* in its description.

Make sure that the subnets that you specify during cluster creation have enough available IP addresses for the Amazon EKS created network interfaces. If you're going to deploy a cluster that uses the IPv4 family, we recommend creating small (/28), dedicated subnets for Amazon EKS created network interfaces, and only specifying these subnets as part of cluster creation. Other resources, such as nodes and load balancers, should be launched in separate subnets from the subnets specified during cluster creation.

Important

- Nodes and load balancers can be launched in any subnet in your cluster's VPC, including subnets not registered with Amazon EKS during cluster creation. Subnets do not require any tags for nodes. For Kubernetes load balancing auto discovery to work, subnets must be tagged as described in [Subnet tagging \(p. 272\)](#).
 - Subnets associated with your cluster cannot be changed after cluster creation. If you need to control exactly which subnets the Amazon EKS created network interfaces are placed in, then specify only two subnets during cluster creation, each in a different Availability Zone.
 - Do not select a subnet in AWS Outposts, AWS Wavelength, or an AWS Local Zone when creating your cluster.
 - Clusters created using v1.14 or earlier contain a `kubernetes.io/cluster/cluster-name` tag on your VPC. This tag was only used by Amazon EKS and can be safely removed.
 - It can take a cluster as long as five hours to recognize a CIDR block added to an existing VPC. Each of the subnets from the new CIDR block must have at least five free IP addresses for the control plane to reconcile.
 - Amazon EC2 [resource-based naming](#) is not supported with Amazon EKS. You must use IP address-based naming for your subnets and instances.
 - If you want Kubernetes to assign IPv6 addresses to pods and services, then you must have IPv4 and IPv6 CIDR blocks assigned to your VPC and subnets. For more information, see [Associate an IPv6 CIDR block with your VPC](#) in the Amazon VPC User Guide. Your route tables and security groups must also include IPv6 addresses. For more information, see [Migrate to IPv6](#) in the Amazon VPC User Guide.
 - Nodes must be able to communicate with the control plane and other AWS services. If your nodes are deployed in a private subnet and you want pods to have outbound access to the internet, then the private subnet must meet one of the following requirements:
 - Subnets with only IPv4 CIDR blocks must have a default route to a [NAT gateway](#). The NAT gateway must be assigned a public IPv4 address to provide internet access for the nodes.
- Subnets with IPv6 CIDR blocks must have a default route to an [egress-only internet gateway](#).
- Is configured with the necessary settings and requirements in [Private clusters \(p. 102\)](#).

Your VPC must have DNS hostname and DNS resolution support, or your nodes can't register with your cluster. For more information, see [Using DNS with Your VPC](#) in the Amazon VPC User Guide.

VPC IP addressing

If you want pods deployed to nodes in public subnets to have outbound internet access, then your public subnets must be configured to auto-assign public IPv4 addresses or IPv6 addresses. Determine whether your public subnets are configured to auto-assign public IPv4 addresses or IPv6 addresses with the following command. Replace the *example values* with your own values.

IPv4

```
aws ec2 describe-subnets \
--filters "Name=vpc-id,Values=VPC-ID" | grep 'SubnetId\|MapPublicIpOnLaunch'
```

IPv6

```
aws ec2 describe-subnets \
--filters "Name=vpc-id,Values=VPC-ID" | grep 'SubnetId\|\
AssignIpv6AddressOnCreation'
```

Example output:

IPv4

```
"MapPublicIpOnLaunch": true,
"SubnetId": "subnet-ID1",
"MapPublicIpOnLaunch": false,
"SubnetId": "subnet-ID2",
...
```

IPv6

```
"SubnetId": "subnet-ID1",
"AssignIpv6AddressOnCreation": true,
"SubnetId": "subnet-ID2",
"AssignIpv6AddressOnCreation": false,
...
```

For any subnets that have `MapPublicIpOnLaunch` or `AssignIpv6AddressOnCreation` set to `false`, change the setting to `true`.

IPv4

```
aws ec2 modify-subnet-attribute --subnet-id subnet-ID2 --map-public-ip-on-launch
```

IPv6

```
aws ec2 modify-subnet-attribute --subnet-id subnet-ID2 --assign-ipv6-address-on-
creation
```

Important

If you used an [Amazon EKS AWS CloudFormation template \(p. 265\)](#) to deploy your VPC before March 26, 2020, then you need to change the setting for your public subnets.

You can define both private (RFC 1918), and public (non-RFC 1918) classless inter-domain routing (CIDR) IPv4 ranges within the VPC used for your Amazon EKS cluster. For more information, see [Adding IPv4 CIDR blocks to a VPC](#) in the Amazon VPC User Guide. When choosing the CIDR blocks for your VPC and subnets, make sure that the blocks contain enough IPv4 addresses for all of the Amazon EC2 nodes and pods that you plan to deploy. There should be at least one IP address for each of your pods. You can conserve IP address use by implementing a transit gateway with a shared services VPC. For more information, see [Isolated VPCs with shared services](#) and [Amazon EKS VPC routable IP address conservation patterns in a hybrid network](#).

Subnet tagging

For 1.18 and earlier clusters, Amazon EKS adds the following tag to all subnets passed in during cluster creation. Amazon EKS does not add the tag to subnets passed in when creating 1.19 and later clusters. If the tag exists on subnets used by a cluster created on a version earlier than 1.19, and you update the cluster to 1.19 or later, the tag is not removed from the subnets.

- **Key** – `kubernetes.io/cluster/cluster-name`
- **Value** – `shared`

The tag might be required if you're using the AWS Load Balancer Controller. For more information about load balancer subnet tagging, see [Application load balancing on Amazon EKS \(p. 378\)](#) and [Network load balancing on Amazon EKS \(p. 372\)](#).

Increase available IPv4 addresses for your VPC

If your Amazon VPC is running out of IPv4 addresses, you can associate a secondary CIDR to an existing VPC. For more information, see [Associate a secondary IP address CIDR block with your VPC](#) in the Amazon VPC User Guide. With the new associated CIDR, you can create subnets using a subset of the newly associated CIDR. After creating the new subnet or subnets, you can create additional node groups, either managed or self-managed, that use the newly associated CIDR and subnets.

Amazon EKS security group considerations

The following sections describe the recommended or minimum required security group settings for the cluster, control plane, and node security groups of your cluster. These considerations are dependent on which Kubernetes version and Amazon EKS platform version you use.

Important

If you've configured your cluster to use [IPv6 \(p. 294\)](#), then your security groups must allow communication to and from all IPv4 and IPv6 addresses that pods communicate with or are communicated with from.

Cluster security group

Amazon EKS clusters, starting with Kubernetes version 1.14 and [platform version \(p. 76\)](#) `eks . 3`, create a cluster security group when they are created. This also happens when a cluster of an earlier version is upgraded to this Kubernetes version and platform version. A cluster security group is designed to allow all traffic from the control plane and [managed node groups \(p. 109\)](#) to flow freely between each other. By assigning the cluster security group to the elastic network interfaces created by Amazon EKS that allow the control plane to communicate with the managed node group instances, you don't need to configure complex security group rules to allow this communication. Any instance or network interface that is assigned this security group can freely communicate with other resources with this security group.

You can check for a cluster security group for your cluster in the AWS Management Console under the cluster's **Networking** section, or with the following AWS CLI command:

```
aws eks describe-cluster --name my-cluster --query
  cluster.resourcesVpcConfig.clusterSecurityGroupId
```

We recommend that you add the cluster security group to all existing and future node groups. For more information, see [Security Groups for Your VPC](#) in the *Amazon VPC User Guide*. Amazon EKS [managed node groups \(p. 109\)](#) are automatically configured to use the cluster security group.

	Protocol	Ports	Source	Destination
Recommended inbound traffic	All	All	Self	
Recommended outbound traffic	All	All		0.0.0.0/0 (IPv4) or ::/0 (IPv6)

Restricting cluster traffic

If you need to limit the open ports between the control plane and nodes, the default cluster security group can be modified to allow only the following required minimum ports. The required minimum ports are the same as they were in previous Amazon EKS versions.

	Protocol	Port	Source	Destination
Minimum inbound traffic	TCP	443	Cluster security group	
Minimum inbound traffic*	TCP	10250	Cluster security group	
CoreDNS	TCP and UDP	53	Cluster security group	
Minimum outbound traffic	TCP	443		Cluster security group
Minimum outbound traffic*	TCP	10250		Cluster security group
DNS	TCP and UDP	53		Cluster security group

*Any protocol and ports that you expect your nodes to use for inter-node communication should be included, if required. Nodes also require outbound internet access to the Amazon EKS APIs for cluster introspection and node registration at launch time, or that you've implemented the required necessary settings in [Private clusters \(p. 102\)](#). To pull container images, they require access to Amazon S3, Amazon ECR APIs, and any other container registries that they need to pull images from, such as DockerHub. For more information, see [AWS IP address ranges](#) in the AWS General Reference.

Control plane and node security groups

For Amazon EKS clusters created earlier than Kubernetes version 1.14 and [platform version \(p. 76\)](#) eks . 3, control plane to node communication was configured by manually creating a control plane security group and specifying that security group when you created the cluster. At cluster creation, this security group was then attached to the network interfaces created by Amazon EKS that allow communication between the control plane and the nodes. These network interfaces have `Amazon EKS cluster name` in their description.

Note

If you used the API directly, or a tool such as AWS CloudFormation to create your cluster and didn't specify a security group, then the default security group for the VPC was applied to the control plane cross-account network interfaces.

You can check the control plane security group for your cluster in the AWS Management Console under the cluster's **Networking** section (listed as **Additional security groups**), or with the following AWS CLI command:

```
aws eks describe-cluster --name my-cluster --query
cluster.resourcesVpcConfig.securityGroupIds
```

If you launch nodes with the AWS CloudFormation template in the [Getting started with Amazon EKS \(p. 4\)](#) walkthrough, AWS CloudFormation modifies the control plane security group to allow communication with the nodes. **Amazon EKS strongly recommends that you use a dedicated security group for each control plane (one for each cluster).** If you share a control plane security group with other Amazon EKS clusters or resources, you may block or disrupt connections to those resources.

The security group for the nodes and the security group for the control plane communication to the nodes have been set up to prevent communication to privileged ports in the nodes. If your applications require added inbound or outbound access from the control plane or nodes, you must add these rules to the security groups associated with your cluster. For more information, see [Security Groups for Your VPC](#) in the [Amazon VPC User Guide](#).

Note

To allow proxy functionality on privileged ports or to run the CNCF conformance tests yourself, you must edit the security groups for your control plane and the nodes. The security group on the nodes' side needs to allow inbound access for ports 0-65535 from the control plane, and the control plane side needs to allow outbound access to the nodes on ports 0-65535.

Control Plane Security Group

	Protocol	Port range	Source	Destination
Minimum inbound traffic	TCP	443	All node security groups When cluster endpoint private access (p. 47) is enabled: Any security groups that generate API server client traffic (such as <code>kubectl</code> commands on a bastion host within your cluster's VPC)	
Recommended inbound traffic	TCP	443	All node security groups When cluster endpoint private access (p. 47) is enabled: Any security groups that generate API server client traffic (such as <code>kubectl</code> commands on a bastion host)	

	Protocol	Port range	Source	Destination
			within your cluster's VPC)	
Minimum outbound traffic	TCP	10250		All node security groups
Recommended outbound traffic	TCP	1025-65535		All node security groups

Node security group

	Protocol	Port range	Source	Destination
Minimum inbound traffic (from other nodes)	Any protocol that you expect your nodes to use for inter-node communication	Any ports that you expect your nodes to use for inter-node communication	All node security groups	
CoreDNS	TCP and UDP	53	All node security groups	
Minimum inbound traffic (from control plane)	TCP	10250	Control plane security group	
Recommended inbound traffic	All TCP	All 443, 1025-65535	All node security groups Control plane security group	
Minimum outbound traffic*	TCP	443		Control plane security group
Recommended outbound traffic	All	All		0.0.0.0/0 (IPv4) or ::/0 (IPv6)

*Nodes also require access to the Amazon EKS APIs for cluster introspection and node registration at launch time either through the internet or VPC endpoints. To pull container images, they require access to the Amazon S3 and Amazon ECR APIs (and any other container registries, such as DockerHub). For more information, see [AWS IP address ranges](#) in the *AWS General Reference* and [Private clusters \(p. 102\)](#).

One, and only one, of the security groups associated to your nodes should have the following tag applied: For more information about tagging, see [Working with tags using the console \(p. 433\)](#).

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

Pod networking using the Amazon VPC Container network interface (CNI) plugin

Amazon EKS supports native VPC networking with the Amazon VPC Container Network Interface (CNI) plugin for Kubernetes. This plugin assigns a private IPv4 or IPv6 address from your VPC to each pod. The plugin is an open-source project that is maintained on GitHub. For more information, see [amazon-vpc-cni-k8s](#) and [Proposal: CNI plugin for Kubernetes networking over Amazon VPC](#) on GitHub. The Amazon VPC CNI plugin is fully supported for use on Amazon EKS and self-managed Kubernetes clusters on AWS.

Note

Kubernetes can use the [Container Networking Interface \(CNI\)](#) for configurable networking setups. The Amazon VPC CNI plugin might not meet requirements for all use cases. Amazon EKS maintains a network of partners that offer alternative CNI solutions with commercial support options. For more information, see [Alternate compatible CNI plugins \(p. 329\)](#).

When you create an Amazon EKS node, it has one network interface. All Amazon EC2 instance types support more than one network interface. The network interface attached to the instance when the instance is created is called the *primary network interface*. Any additional network interface attached to the instance is called a *secondary network interface*. Each network interface can be assigned multiple private IP addresses. One of the private IP addresses is the *primary IP address*, whereas all other addresses assigned to the network interface are *secondary IP addresses*. For more information about network interfaces, see [Elastic network interfaces](#) in the *Amazon EC2 User Guide for Linux Instances*. For more information about how many network interfaces and private IP addresses are supported for each network interface, see [IP addresses per network interface per instance type](#) in the *Amazon EC2 User Guide for Linux Instances*. For example, an `m5.1.large` instance type supports three network interfaces and ten private IP addresses for each network interface.

The Amazon VPC Container Network Interface (CNI) plugin for Kubernetes is deployed with each of your Amazon EC2 nodes in a Daemonset with the name `aws-node`. The plugin consists of two primary components:

- **L-IPAM daemon** – Responsible for creating network interfaces and attaching the network interfaces to Amazon EC2 instances, assigning secondary IP addresses to network interfaces, and maintaining a warm pool of IP addresses on each node for assignment to Kubernetes pods when they are scheduled. When the number of pods running on the node exceeds the number of addresses that can be assigned to a single network interface, the plugin starts allocating a new network interface, as long as the maximum number of network interfaces for the instance aren't already attached. There are configuration variables that allow you to change the default value for when the plugin creates new network interfaces. For more information, see [WARM_ENI_TARGET](#), [WARM_IP_TARGET](#) and [MINIMUM_IP_TARGET](#) on GitHub.

Each pod that you deploy is assigned one secondary private IP address from one of the network interfaces attached to the instance. Previously, it was mentioned that an `m5.1.large` instance supports three network interfaces and ten private IP addresses for each network interface. Even though an `m5.1.large` instance supports 30 private IP addresses, you can't deploy 30 pods to that node. To determine how many pods you can deploy to a node, use the following formula:

```
(Number of network interfaces for the instance type × (the number of IP addresses per network interface - 1)) + 2
```

Using this formula, an `m5.1.large` instance type can support a maximum of 29 pods. For a list of the maximum number of pods supported by each instance type, see [eni-max-pods.txt](#) on GitHub. Any pods that don't use host networking, such as CoreDNS, count towards the maximum pods. For the nodes of the cluster that CoreDNS runs on, you're only able to deploy 28 additional pods to an `m5.1.large` instance, not 29. Pods that use host networking, such as the Amazon VPC CNI plugin and `kube-proxy` don't count towards the maximum number.

By default, all pods deployed to a node are assigned the same security groups and are assigned private IP addresses from a CIDR block that is assigned to the subnet that one of the instance's network interfaces is connected to. You can assign IP addresses from a different CIDR block than the subnet that the primary network interface is connected to by configuring [CNI custom networking \(p. 306\)](#). You can also use CNI custom networking to assign all pods on a node the same security groups. The security groups assigned to all pods can be different than the security groups assigned to the primary network interface. You can assign unique security groups to pods deployed to many Amazon EC2 instance types using security groups for pods. For more information, see [Security groups for pods \(p. 313\)](#).

- **CNI plugin** – Responsible for wiring the host network (for example, configuring the network interfaces and virtual Ethernet pairs) and adding the correct network interface to the pod namespace.

If you use custom pod security policies with your cluster, be sure to review the [Pod security policy \(p. 502\)](#) topic for important information.

Configuring the Amazon VPC CNI plugin to use IAM roles for service accounts

The [Amazon VPC CNI plugin for Kubernetes](#) is the networking plugin for pod networking in Amazon EKS clusters. The plugin is responsible for allocating VPC IP addresses to Kubernetes nodes and configuring the necessary networking for pods on each node. The plugin:

- Requires AWS Identity and Access Management (IAM) permissions. If your cluster uses the IPv4 family, the permissions are specified in the [AmazonEKS_CNI_Policy](#) AWS managed policy. If your cluster uses the IPv6 family, then the permissions must be added to an [IAM policy that you create \(p. 281\)](#). You can attach the policy to the [Amazon EKS node IAM role \(p. 474\)](#), or to a separate IAM role. We recommend that you assign it to a separate role, as detailed in this topic.
- Creates and is configured to use a Kubernetes service account named `aws-node` when it's deployed. The service account is bound to a Kubernetes `clusterrole` named `aws-node`, which is assigned the required Kubernetes permissions.

Note

The pods for the Amazon VPC CNI plugin have access to the permissions assigned to the [Amazon EKS node IAM role \(p. 474\)](#), unless you block access to IMDS. For more information, see [Restrict access to the instance profile assigned to the worker node](#).

Prerequisites

- An existing Amazon EKS cluster. To deploy one, see [Getting started with Amazon EKS \(p. 4\)](#).
- An existing AWS Identity and Access Management (IAM) OpenID Connect (OIDC) provider for your cluster. To determine whether you already have one, or to create one, see [Create an IAM OIDC provider for your cluster \(p. 446\)](#).

Step 1: Create the Amazon VPC CNI plugin IAM role

To create the IAM role

1. Determine the IP family of your cluster.

```
aws eks describe-cluster --name my-cluster | grep ipFamily
```

Output

```
"ipFamily": "ipv4"
```

The output may return `ipv6` instead.

2. Create the IAM role. You can use `eksctl` or `kubectl` and the AWS CLI to create your CNI plugin IAM role.

`eksctl`

Create an IAM role and attach the IAM policy to the role with the command that matches the IP family of your cluster. The command creates and deploys an AWS CloudFormation stack that creates an IAM role, attaches the policy that you specify to it, and annotates the existing `aws-node` Kubernetes service account with the ARN of the IAM role that is created.

- IPv4

Replace `my-cluster` with your own value.

```
eksctl create iamserviceaccount \
--name aws-node \
--namespace kube-system \
--cluster my-cluster \
--role-name "AmazonEKSVPCCNIRole" \
--attach-policy-arn arn:aws:iam::aws:policy/AmazonEKS_CNI_Policy \
--override-existing-serviceaccounts \
--approve
```

- IPv6

Replace `my-cluster` with your own value. Replace `111122223333` with your account ID and replace `AmazonEKS_CNI_IPv6_Policy` with the name of your IPv6 policy. If you don't have an IPv6 policy, see [Create IAM policy for clusters that use the IPv6 family \(p. 281\)](#) to create one. To use IPv6 with your cluster, it must meet several requirements. For more information, see [Assigning IPv6 addresses to pods and services \(p. 294\)](#).

```
eksctl create iamserviceaccount \
--name aws-node \
--namespace kube-system \
--cluster my-cluster \
--role-name "AmazonEKSVPCCNIRole" \
--attach-policy-arn
arn:aws:iam::111122223333:policy/AmazonEKS_CNI_IPv6_Policy \
--override-existing-serviceaccounts \
--approve
```

kubectl and the AWS CLI

1. View your cluster's OIDC provider URL.

```
aws eks describe-cluster --name my-cluster --query "cluster.identity.oidc.issuer"
--output text
```

Example output:

```
https://oidc.eks.region-code.amazonaws.com/id/EXAMPLED539D4633E53DE1B71EXAMPLE
```

If no output is returned, then you must [create an IAM OIDC provider for your cluster \(p. 446\)](#).

2. Copy the following contents to a file named `vpc-cni-trust-policy.json`. Replace `111122223333` with your account ID and `EXAMPLED539D4633E53DE1B71EXAMPLE` from the output returned in the previous step. Replace `region-code` with the AWS Region that your cluster is in.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "Federated": "arn:aws:iam::111122223333:oidc-provider/
oidc.eks.region-code.amazonaws.com/id/EXAMPLED539D4633E53DE1B71EXAMPLE"
            },
            "Action": "sts:AssumeRoleWithWebIdentity",
            "Condition": {
                "StringEquals": {
                    "oidc.eks.region-code.amazonaws.com/
id/EXAMPLED539D4633E53DE1B71EXAMPLE:aud": "sts.amazonaws.com",
                    "oidc.eks.region-code.amazonaws.com/
id/EXAMPLED539D4633E53DE1B71EXAMPLE:sub": "system:serviceaccount:kube-system:aws-
node"
                }
            }
        }
    ]
}
```

3. Create the role. You can replace `AmazonEKSVPCCNIRole` with any name that you choose.

```
aws iam create-role \
--role-name AmazonEKSVPCCNIRole \
--assume-role-policy-document file://"vpc-cni-trust-policy.json"
```

4. Attach the required IAM policy to the role. Run the command that matches the IP family of your cluster.

- IPv4

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

- IPv6

Replace `111122223333` with your account ID and `AmazonEKS_CNI_IPv6_Policy` with the name of your IPv6 policy. If you don't have an IPv6 policy, see [Create IAM policy for clusters that use the IPv6 family \(p. 281\)](#) to create one. To use IPv6 with your cluster, it must meet several requirements. For more information, see [Assigning IPv6 addresses to pods and services \(p. 294\)](#).

```
aws iam attach-role-policy \
--policy-arn arn:aws:iam::111122223333:policy/AmazonEKS_CNI_IPv6_Policy \
--role-name AmazonEKSVPCCNIRole
```

5. Run the following command to annotate the `aws-node` service account with the ARN of the IAM role that you created previously. Replace the `example values` with your own values.

```
kubectl annotate serviceaccount \
-n kube-system aws-node \
eks.amazonaws.com/role-arn=arn:aws:iam::111122223333:role/AmazonEKSVPCCNIRole
```

3. (Optional) Configure the AWS Security Token Service endpoint type used by your Kubernetes service account. For more information, see [Configure the AWS Security Token Service endpoint for a service account \(p. 452\)](#).

Step 2: Re-deploy VPC CNI pods

1. Delete and re-create any existing pods that are associated with the service account to apply the credential environment variables. The annotation is not applied to pods that are currently running without the annotation. The following command deletes the existing aws-node DaemonSet pods and deploys them with the service account annotation.

```
kubectl delete pods -n kube-system -l k8s-app=aws-node
```

2. Confirm that the pods all restarted.

```
kubectl get pods -n kube-system -l k8s-app=aws-node
```

3. Describe one of the pods and verify that the `AWS_WEB_IDENTITY_TOKEN_FILE` and `AWS_ROLE_ARN` environment variables exist. Replace `9rgzw` with the name of one of your pods returned in the output of the previous step.

```
kubectl exec -n kube-system aws-node-9rgzw -c aws-node -- env | grep AWS
```

Output:

```
...
AWS_WEB_IDENTITY_TOKEN_FILE=/var/run/secrets/eks.amazonaws.com/serviceaccount/token
...
AWS_ROLE_ARN=arn:aws:iam::111122223333:role/AmazonEKSVPCCNIRole
...
```

If your pod is using the AWS Regional endpoint, then the following line is also returned in the previous output.

```
AWS_STS_REGIONAL_ENDPOINTS=regional
```

Step 3: Remove the CNI policy from the node IAM role

If your [Amazon EKS node IAM role \(p. 474\)](#) currently has the `AmazonEKS_CNI_Policy` IAM (IPv4) policy or an IPv6 policy (p. 281) attached to it, and you've created a separate IAM role, attached the policy to it instead, and assigned it to the `aws-node` Kubernetes service account, then we recommend that you remove the policy from your node role with the the AWS CLI command that matches the IP family of your cluster. Replace `AmazonEKSNodeRole` with the name of your node role.

- IPv4

```
aws iam detach-role-policy --role-name AmazonEKSNodeRole --policy-arn
arn:aws:iam::aws:policy/AmazonEKS_CNI_Policy
```

- IPv6

Replace `111122223333` with your account ID and `AmazonEKS_CNI_IPv6_Policy` with the name of your IPv6 policy.

```
aws iam detach-role-policy --role-name AmazonEKSNodeRole --policy-arn
arn:aws:iam::111122223333:policy/AmazonEKS_CNI_IPv6_Policy
```

Create IAM policy for clusters that use the IPv6 family

If you created a 1.21 or later cluster that uses the IPv6 family and the cluster has version 1.10.1 or later of the VPC CNI add-on configured, then you need to create an IAM policy that you can assign to an IAM role. If you have an existing 1.21 or later cluster that you didn't configure with the IPv6 family when you created it, then to use IPv6, you must create a new cluster. For more information about using IPv6 with your cluster, see [the section called "IPv6" \(p. 294\)](#).

1. Copy the following text and save it to a file named `vpc-cni-ipv6-policy.json`.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "ec2:AssignIpv6Addresses",
                "ec2:DescribeInstances",
                "ec2:DescribeTags",
                "ec2:DescribeNetworkInterfaces",
                "ec2:DescribeInstanceTypes"
            ],
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "ec2:CreateTags"
            ],
            "Resource": [
                "arn:aws:ec2:*:*:network-interface/*"
            ]
        }
    ]
}
```

2. Create the IAM policy.

```
aws iam create-policy \
--policy-name AmazonEKS_CNI_IPv6_Policy \
--policy-document file://vpc-cni-ipv6-policy.json
```

Managing the Amazon VPC CNI add-on

Amazon EKS supports native VPC networking with the Amazon VPC Container Network Interface (CNI) plugin for Kubernetes. Using this plugin allows Kubernetes pods to have the same IP address inside the pod as they do on the VPC network. For more information, see [Pod networking using the Amazon VPC Container network interface \(CNI\) plugin \(p. 276\)](#).

If you created a 1.18 or later cluster using the AWS Management Console, then Amazon EKS installed the Amazon EKS add-on for you. If you created a 1.18 or later cluster using any method other than the AWS Management Console, then Amazon EKS installed the self-managed add-on for you. You can migrate the self-managed add-on to the Amazon EKS add-on using the procedure in [Adding the Amazon VPC CNI Amazon EKS add-on \(p. 282\)](#). If you have a cluster that you've already added the Amazon VPC CNI Amazon EKS add-on to, you can manage it using the procedures in the [Updating the Amazon VPC CNI Amazon EKS add-on \(p. 284\)](#) and [Removing the Amazon VPC CNI Amazon EKS add-on \(p. 286\)](#) sections. For more information about Amazon EKS add-ons, see [Amazon EKS add-ons \(p. 388\)](#).

Recommended version of the Amazon VPC CNI add-on for each cluster version

	1.22	1.21	1.20	1.19	1.18	1.17
Add-on version	1.11.0-eksbuild.1	1.11.0-eksbuild.1	1.11.0-eksbuild.1	1.11.0-eksbuild.1	1.11.0-eksbuild.1	1.11.0-eksbuild.1

To update your Amazon EKS add-on version, see [Updating the Amazon VPC CNI Amazon EKS add-on \(p. 284\)](#). To update your self-managed add-on version using container images in the Amazon EKS Amazon Elastic Container Registry or your own repository, see [Updating the Amazon VPC CNI self-managed add-on \(p. 287\)](#).

Important

The version of the add-on that was deployed when you created your cluster may be earlier than the recommended version. If you've updated the self-managed add-on using a manifest, then the version doesn't include -eksbuild.1.

Prerequisites

- An existing Amazon EKS cluster. To deploy one, see [Getting started with Amazon EKS \(p. 4\)](#).
- If your cluster is 1.21 or later, make sure that your `kube-proxy` and `CoreDNS` add-ons are at the minimum versions listed in [Service account tokens \(p. 441\)](#).
- An existing AWS Identity and Access Management (IAM) OpenID Connect (OIDC) provider for your cluster. To determine whether you already have one, or to create one, see [Create an IAM OIDC provider for your cluster \(p. 446\)](#).
- An IAM role with the `AmazonEKS_CNI_Policy` IAM policy (if your cluster uses the IPv4 family) or an `IPv6 policy (p. 281)` (if your cluster uses the IPv6 family) attached to it. For more information, see [Configuring the Amazon VPC CNI plugin to use IAM roles for service accounts \(p. 277\)](#).
- If you are using version 1.7.0 or later of the CNI plugin and you use custom pod security policies, see the section called “Delete default policy” (p. 502)[Pod security policy \(p. 501\)](#).

Adding the Amazon VPC CNI Amazon EKS add-on

You can add the Amazon VPC CNI Amazon EKS add-on to your 1.18 or later cluster with `eksctl`, the AWS Management Console, or the AWS CLI.

Important

Before adding the Amazon VPC CNI Amazon EKS add-on, confirm that you do not self-manage any settings that Amazon EKS will start managing. To determine which settings Amazon EKS manages, see [Amazon EKS add-on configuration \(p. 389\)](#).

`eksctl`

To add the recommended version (p. 282) of the Amazon EKS add-on using `eksctl`

Replace `my-cluster` with the name of your cluster and `arn:aws:iam::111122223333:role/AmazonEKSVPCCNIRole` with your existing IAM role (see [Prerequisites \(p. 282\)](#)).

```
eksctl create addon \
--name vpc-cni \
--version v1.11.0-eksbuild.1 \
--cluster my-cluster \
--service-account-role-arn arn:aws:iam::111122223333:role/AmazonEKSVPCCNIRole \
--force
```

If any of the Amazon EKS add-on settings conflict with the existing settings for the self-managed add-on, then adding the Amazon EKS add-on fails, and you receive an error message to help you resolve the conflict.

If you want to add a different version of the add-on instead, then you can view all versions available for the add-on and your cluster's version with the following command. Replace **1.22** with your cluster's version.

```
eksctl utils describe-addon-versions --name vpc-cni --kubernetes-version 1.22 | grep
AddonVersion:
```

Replace v1.11.0-eksbuild.1 in the `create addon` command with the version returned in the output that you want to add and then run the `create addon` command.

AWS Management Console

To add the [recommended version \(p. 282\)](#) of the Amazon EKS add-on using the AWS Management Console

1. Open the Amazon EKS console at <https://console.aws.amazon.com/eks/home#/clusters>.
2. In the left navigation pane, select Amazon EKS **Clusters**, and then select the name of the cluster that you want to configure the Amazon VPC CNI Amazon EKS add-on for.
3. Choose the **Configuration** tab and then choose the **Add-ons** tab.
4. Select **Add new**.
 - Select **vpc-cni** for **Name**.
 - Select the **Version** you'd like to use. We recommend the **1.11.0-eksbuild.1** version, but you can select a different version if necessary.
 - For **Service account role**, select the name of an IAM role that you've attached the [AmazonEKS_CNI_Policy](#) IAM policy to (see [Prerequisites \(p. 282\)](#)).
 - Select **Override existing configuration for this add-on on the cluster**. If any of the Amazon EKS add-on settings conflict with the existing settings for the self-managed add-on, then adding the Amazon EKS add-on fails, and you receive an error message to help you resolve the conflict.
 - Select **Add**.

AWS CLI

To add the [recommended version \(p. 282\)](#) of the Amazon EKS add-on using the AWS CLI, replace *my-cluster* with the name of your cluster, `arn:aws:iam::111122223333:role/AmazonEKSCNIRole` with the ARN of an IAM role that you've attached the [AmazonEKS_CNI_Policy](#) IAM policy to (see [Prerequisites \(p. 282\)](#)), and then run the command.

```
aws eks create-addon \
--cluster-name my-cluster \
--addon-name vpc-cni \
--addon-version v1.11.0-eksbuild.1 \
--service-account-role-arn arn:aws:iam::111122223333:role/AmazonEKSVPCCNIRole \
```

```
--resolve-conflicts OVERWRITE
```

If any of the Amazon EKS add-on settings conflict with the existing settings for the self-managed add-on, then adding the Amazon EKS add-on fails, and you receive an error message to help you resolve the conflict.

If you want to add a different version of the add-on instead, then you can view all versions available for the add-on and your cluster's version with the following command. Replace [1.22](#) with your cluster's version.

```
aws eks describe-addon-versions \
--addon-name vpc-cni \
--kubernetes-version 1.22 \
--query "addons[].[addonVersions[].[addonVersion, compatibilities[].[version]]" \
--output text
```

Replace `v1.11.0-eksbuild.1` in the `create-addon` command with the version returned in the output that you want to add and then run the `create-addon` command.

Updating the Amazon VPC CNI Amazon EKS add-on

Important

Before updating the Amazon VPC CNI Amazon EKS add-on, confirm that you do not self-manage any settings that Amazon EKS manages. To determine which settings Amazon EKS manages, see [Amazon EKS add-on configuration \(p. 389\)](#).

This procedure is for updating the Amazon VPC CNI Amazon EKS add-on. If you haven't added the Amazon VPC CNI Amazon EKS add-on, complete the procedure in [Updating the Amazon VPC CNI self-managed add-on \(p. 287\)](#) instead. Amazon EKS does not automatically update the Amazon VPC CNI add-on when new versions are released or after you [update your cluster \(p. 34\)](#) to a new Kubernetes minor version. To update the Amazon VPC CNI add-on for an existing cluster, you must initiate the update and then Amazon EKS updates the add-on for you.

We recommend that you update one minor version at a time. For example, if your current minor version is `1.9` and you want to update to `1.11`, you should update to the latest patch version of `1.10` first, then update to the latest patch version of `1.11`.

You can update the Amazon VPC CNI Amazon EKS add-on on your `1.18` or later cluster using `eksctl`, the AWS Management Console, or the AWS CLI.

`eksctl`

To update the Amazon EKS add-on to the recommended version (p. 282) using `eksctl`

1. Check the current version of your `vpc-cni` Amazon EKS add-on. Replace `my-cluster` with your cluster name.

```
eksctl get addon --name vpc-cni --cluster my-cluster
```

Example output:

NAME	VERSION	STATUS	ISSUES	IAMROLE	UPDATE
AVAILABLE					
vpc-cni	v1.7.5-eksbuild.2	ACTIVE	0	arn:aws:iam::111122223333:role/ AmazonEKSVPCCNIRole	v1.11.0-eksbuild.1

2. Update the add-on to the [recommended version \(p. 282\)](#).

```
eksctl update addon \
--name vpc-cni \
--version 1.11.0-eksbuild.1 \
--cluster my-cluster \
--force
```

If you want to update to a different version of the add-on instead, then you can view all versions available for the add-on and your cluster's version with the following command. Replace [1.22](#) with your cluster's version.

```
eksctl utils describe-addon-versions --name vpc-cni --kubernetes-version 1.22 | grep AddonVersion:
```

Replace `v1.11.0-eksbuild.1` in the `update addon` command with the version returned in the output that you want to add and then run the `update addon` command.

AWS Management Console

To update the Amazon EKS add-on to the [recommended version \(p. 282\)](#) using the AWS Management Console

1. Open the Amazon EKS console at <https://console.aws.amazon.com/eks/home#/clusters>.
2. In the left navigation pane, select Amazon EKS **Clusters**, and then select the name of the cluster that you want to update the Amazon VPC CNI add-on for.
3. Choose the **Configuration** tab and then choose the **Add-ons** tab.
4. Select the box in the top right of the **vpc-cni** box and then choose **Edit**.
 - Select the **Version** of the Amazon EKS add-on that you want to use. We recommend the `1.11.0-eksbuild.1` version, but you can select a different version if necessary.
 - For **Service account role**, select the name of an IAM role that you've attached the `AmazonEKS_CNI_Policy` IAM policy to (see [Prerequisites \(p. 282\)](#)), if one isn't already selected.
 - Select **Override existing configuration for this add-on on the cluster**.
 - Select **Update**.

AWS CLI

To update the Amazon EKS add-on to the [recommended version \(p. 282\)](#) using the AWS CLI

1. Check the current version of your Amazon VPC CNI Amazon EKS add-on. Replace *my-cluster* with your cluster name.

```
aws eks describe-addon \
--cluster-name my-cluster \
--addon-name vpc-cni \
--query "addon.addonVersion" \
--output text
```

Example output:

```
v1.7.5-eksbuild.2
```

The version returned for you may be different.

2. Update the add-on to the [recommended version](#) (p. 282). Replace *my-cluster* with your cluster name.

```
aws eks update-addon \
--cluster-name my-cluster \
--addon-name vpc-cni \
--addon-version v1.11.0-eksbuild.1 \
--resolve-conflicts OVERWRITE
```

If you want to update to a different version of the add-on instead, then you can view all versions available for the add-on and your cluster's version with the following command. Replace *1.22* with your cluster's version.

```
aws eks describe-addon-versions \
--addon-name vpc-cni \
--kubernetes-version 1.22 \
--query "addons[].[addonVersions[].[addonVersion, compatibilities[].[version]]" \
--output text
```

Replace *v1.11.0-eksbuild.1* in the `update-addon` command with the version returned in the output that you want to add and then run the `update-addon` command.

Removing the Amazon VPC CNI Amazon EKS add-on

You have two options when removing an Amazon EKS add-on:

- **Preserve the add-on's software on your cluster** – This option removes Amazon EKS management of any settings and the ability for Amazon EKS to notify you of updates and automatically update the Amazon EKS add-on after you initiate an update, but preserves the add-on's software on your cluster. This option makes the add-on a self-managed add-on, rather than an Amazon EKS add-on. There is no downtime for the add-on.
- **Removing the add-on software entirely from your cluster** – You should only remove the Amazon EKS add-on from your cluster if there are no resources on your cluster are dependent on the functionality that the add-on provides. After removing the Amazon EKS add-on, you can add it again if you want to.

If the add-on has an IAM account associated with it, the IAM account is not removed.

You can remove the Amazon VPC CNI Amazon EKS add-on from your 1.18 or later cluster with `eksctl`, the AWS Management Console, or the AWS CLI.

`eksctl`

To remove the Amazon EKS add-on using `eksctl`

Replace *my-cluster* with the name of your cluster and then run the following command. Removing `--preserve` removes the add-on software from your cluster.

```
eksctl delete addon --cluster my-cluster --name vpc-cni --preserve
```

AWS Management Console

To remove the Amazon EKS add-on using the AWS Management Console

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

2. In the left navigation pane, select Amazon EKS **Clusters**, and then select the name of the cluster that you want to remove the Amazon VPC CNI Amazon EKS add-on for.
3. Choose the **Configuration** tab, and then choose the **Add-ons** tab.
4. Select the check box in the top right of the **vpc-cni** box and then choose **Remove**. Select **Preserve on cluster** if you want Amazon EKS to stop managing settings for the add-on, but want to retain the add-on software on your cluster so that you can self-manage all of the add-on's settings. Type **vpc-cni** and then select **Remove**.

AWS CLI

To remove the Amazon EKS add-on using the AWS CLI

Replace *my-cluster* with the name of your cluster and then run the following command. Removing **--preserve** removes the add-on software from your cluster.

```
aws eks delete-addon --cluster-name my-cluster --addon-name vpc-cni --preserve
```

Updating the Amazon VPC CNI self-managed add-on

If you have a 1.17 cluster, or a 1.18 or later cluster that you haven't added the Amazon VPC CNI Amazon EKS add-on to, or need to manage the add-on yourself, then complete the following steps to update the add-on. If you've added the Amazon VPC CNI Amazon EKS add-on, complete the procedure in [Updating the Amazon VPC CNI Amazon EKS add-on \(p. 284\)](#) instead.

Important

- Versions are specified as `major-version.minor-version.patch-version`
- You should only update one minor version at a time. For example, if your current minor version is 1.9 and you want to update to 1.11, then you should update to 1.10 first, then update to 1.11.
- All versions work with all Amazon EKS supported Kubernetes versions, though not all features of each release work with all Kubernetes versions. When using different Amazon EKS features, if a specific version of the CNI add-on is required, then it's noted in the feature documentation.
- We recommend that you update to version 1.11.0, though you can update to any [release version](#), if necessary.

To update the self-managed add-on

1. View [releases](#) on GitHub to see the available versions and familiarize yourself with the changes in the version that you want to update to.
2. Use the following command to determine your cluster's current Amazon VPC CNI add-on version:

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

Example output:

```
amazon-k8s-cni-init:v1.7.5-eksbuild.1
amazon-k8s-cni:v1.7.5-eksbuild.1
```

Your output might look different than the example output. In this example output, the Amazon VPC CNI add-on version is 1.7.5-eksbuild.1. The current recommended version is 1.11.0. The version

that Amazon EKS originally deployed with your cluster looks similar to the previous output. If you've already updated the add-on at least once using a manifest however, your output might not include `-eksbuild.1`.

3. Update the DaemonSet using [Helm V3 \(p. 430\)](#) or later, or by using a manifest.

Helm

- a. Add the `eks-charts` repository to Helm.

```
helm repo add eks https://aws.github.io/eks-charts
```

- b. Update your local repository to make sure that you have the most recent charts.

```
helm repo update
```

- c. Backup your current settings so that you can determine which settings you need to specify values for in a later step.

```
kubectl get daemonset aws-node -n kube-system -o yaml > aws-k8s-cni-old.yaml
```

- d. If you installed the existing AWS VPC CNI DaemonSet using Helm, then skip to the next step.

Complete one of the following options so that Helm can manage the DaemonSet resources:

- Add the Helm annotations and labels to your existing resources.
 - i. Save the following contents to a file named `helm-cni.sh`. Replace `aws-vpc-cni` if you want to use a different release name.

```
#!/usr/bin/env bash

set -euo pipefail

for kind in daemonSet clusterRole clusterRoleBinding serviceAccount; do
    echo "setting annotations and labels on $kind/aws-node"
    kubectl -n kube-system annotate --overwrite $kind aws-node meta.helm.sh/
release-name=aws-vpc-cni
    kubectl -n kube-system annotate --overwrite $kind aws-node meta.helm.sh/
release-namespace=kube-system
    kubectl -n kube-system label --overwrite $kind aws-node app.kubernetes.io/
managed-by=Helm
done
```

- ii. Make the script executable.

```
chmod +x helm-cni.sh
```

- iii. Run the script

```
./helm-cni.sh
```

- Remove the existing DaemonSet resources.

Important

Your cluster will experience downtime between completing this step and the next step.

```
kubectl delete serviceaccount aws-node -n kube-system
kubectl delete customresourcedefinition eniconfigs.crd.k8s.amazonaws.com
kubectl delete clusterrole aws-node
kubectl delete clusterrolebinding aws-node
```

```
kubectl delete daemonset aws-node -n kube-system
```

- e. Install the chart using one of the following options. Before running the installation, review the backup you made of the settings for your DaemonSet in a previous step and then review the [configuration settings](#) to determine if you need to set any of them.

If you have an existing IAM role to use with the DaemonSet, then add the following line at the end of the install options that follow. If you don't have an IAM role associated to the aws-node Kubernetes service account, then we recommend creating one. Replace `111122223333` with your account ID and `AmazonEKSVPCCNIRole` with the name of your role. To create a role, see [Configuring the Amazon VPC CNI plugin to use IAM roles for service accounts \(p. 277\)](#).

```
--set serviceAccount.annotations."eks\.amazonaws\.com/role-  
arn"\=arn:aws:iam::111122223333:role/AmazonEKSVPCCNIRole
```

If you added the Helm annotations and labels in the previous step, then add the following settings to any of the following options.

```
--set originalMatchLabels=true  
--set crd.create=false
```

- If your nodes have access to the Amazon EKS Amazon ECR repositories and are in the us-west-2 AWS Region, then install the chart with the release name `aws-vpc-cni` and default configuration.

```
helm upgrade -i aws-vpc-cni eks/aws-vpc-cni \  
--namespace kube-system \  
--set image.tag=v1.11.0 \  
--set init.image.tag=v1.11.0
```

- If your nodes have access to the Amazon EKS Amazon ECR repositories and are in an AWS Region other than us-west-2, then install the chart with the release name `aws-vpc-cni`. Replace `eks-ecr-account` with the value from [the section called "Amazon container image registries" \(p. 386\)](#) for the AWS Region that your cluster is in. Replace `region-code` with the AWS Region that your cluster is in.

```
helm upgrade -i aws-vpc-cni eks/aws-vpc-cni \  
--namespace kube-system \  
--set image.account=eks-ecr-account \  
--set image.region=region-code \  
--set image.tag=v1.11.0 \  
--set init.image.account=eks-ecr-account \  
--set init.image.region=region-code \  
--set init.image.tag=v1.11.0
```

- If your nodes don't have access to the Amazon EKS Amazon ECR repositories
 1. Pull the following container images and push them to a repository that your nodes have access to. For more information on how to pull, tag, and push an image to your own repository, see [Copy a container image from one repository to another repository \(p. 384\)](#). We recommend using the version in the following commands, but if necessary, you can replace it with any [release version](#). Replace `602401143452` and `region-code` with values from [the section called "Amazon container image registries" \(p. 386\)](#) for the AWS Region that your cluster is in.

```
602401143452.dkr.ecr.region-code.amazonaws.com/amazon-k8s-cni-init:v1.11.0  
602401143452.dkr.ecr.region-code.amazonaws.com/amazon-k8s-cni:v1.11.0
```

2. Install the chart with the release name `aws-vpc-cni` and default configuration.

Before running the installation, review the backup you made of the settings for your DaemonSet in a previous step and then review the [configuration settings](#) to determine if you need to set any of them. Replace `registry/repo:tag` with your registry, repository, and tag.

```
helm upgrade -i aws-vpc-cni eks/aws-vpc-cni \
--namespace kube-system \
--set image.override=registry/repo:tag \
--set init.image.override=registry/repo:tag
```

Manifest

- If you've changed any default settings for your current VPC CNI DaemonSet, or you need to pull the container images from your own repository to update the DaemonSet, or your cluster is in a region other than `us-west-2`, or you need to update to a specific patch version for version 1.7 or earlier, then skip to the next step.

Run the following command to update your Amazon VPC CNI add-on. You can change `1.11.0` to 1.7.0 or later. Regardless of the patch version that you specify for 1.7, such as 1.7.5, the latest patch version of the image (1.7.10) is pulled.

```
kubectl apply -f https://raw.githubusercontent.com/aws/amazon-vpc-cni-k8s/v1.11.0/config/master/aws-k8s-cni.yaml
```

If you need to update to a version earlier than 1.7.0, then pull the manifest with the following URL. You can change `1.6` to an earlier version, if necessary. The manifest pulls the latest patch version of the image for the version that you specify.

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

Skip to the [View the status of the DaemonSet \(p. 292\)](#) step.

- If your nodes have access to the Amazon EKS Amazon ECR image repositories, then skip to the next step.

Pull the following container images and push them to a repository that your nodes have access to. For more information on how to pull, tag, and push an image to your own repository, see [Copy a container image from one repository to another repository \(p. 384\)](#). We recommend using the version in the following commands, but if necessary, you can replace it with any [release version](#). Replace `602401143452` and `region-code` with values from [the section called "Amazon container image registries" \(p. 386\)](#) for the AWS Region that your cluster is in.

```
602401143452.dkr.ecr.region-code.amazonaws.com/amazon-k8s-cni-init:v1.11.0
602401143452.dkr.ecr.region-code.amazonaws.com/amazon-k8s-cni:v1.11.0
```

- If you haven't changed any of the default settings for the DaemonSet, skip to the next step.

Backup your current settings so that you can compare your settings to the default settings in the new manifest.

```
kubectl get daemonset aws-node -n kube-system -o yaml > aws-k8s-cni-old.yaml
```

- d. Download the manifest for the Amazon VPC CNI add-on. You can change **v1.11.0** to 1.7.0 or later. Regardless of the patch version that you specify for 1.7, such as 1.7.5, the latest patch version of the image (1.7.10) is pulled.

```
kubectl apply -f https://raw.githubusercontent.com/aws/amazon-vpc-cni-k8s/v1.11.0/config/master/aws-k8s-cni.yaml
```

If you need to update to a version earlier than 1.7.0, then pull the manifest with the following URL. You can change **v1.6** to an earlier version, if necessary. The manifest pulls the latest patch version of the image for the version that you specify.

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

If you need a specific patch version of 1.7 or earlier, open the file in a text editor and change **v1.11.0** in the following two lines to the specific patch version that you want. Depending on which version of the file that you downloaded, **v1.11.0** may be a different version number, or may be latest. After you've made the changes, save the file.

```
image: "602401143452.dkr.ecr.us-west-2.amazonaws.com/amazon-k8s-cni-init:v1.11.0"
image: "602401143452.dkr.ecr.us-west-2.amazonaws.com/amazon-k8s-cni:v1.11.0"
```

- e. If you didn't copy the container images to your own repository in a previous step, then skip to the next step.

Replace the registry, repository, and tag in the file with your own.

- i. Replace **your-registry** in the following command with your registry and then run the modified command to replace 602401143452.dkr.ecr.us-west-2.amazonaws.com in the file.

```
sed -i.bak -e 's|602401143452.dkr.ecr.us-west-2.amazonaws.com|your-registry|' aws-k8s-cni.yaml
```

- ii. Replace **your-repository** and **tag** in the following command with your repository and tag and then run the modified command to replace amazon-k8s-cni-init:v1.11.0 in the file. Replace **v1.11.0** with the version of the manifest that you downloaded.

```
sed -i.bak -e 's|amazon-k8s-cni-init:v1.11.0|your-repository:tag|' aws-k8s-cni.yaml
```

- iii. Replace **your-repository** and **tag** in the following command with your repository and tag and then run the modified command to replace amazon-k8s-cni:v1.11.0 in the file. Replace **v1.11.0** with the version of the manifest that you downloaded.

```
sed -i.bak -e 's|amazon-k8s-cni:v1.11.0|your-repository:tag|' aws-k8s-cni.yaml
```

- iv. Skip to the [Compare settings \(p. 292\)](#) step.

- f. Run the following command to replace information in the file with information for the AWS Region that your cluster is in.

- i. Replace us-west-2 in the file with the AWS Region that your cluster is in.

AWS GovCloud (US-East)

```
sed -i.bak -e 's|us-west-2|us-gov-east-1|' aws-k8s-cni.yaml
```

AWS GovCloud (US-West)

```
sed -i.bak -e 's|us-west-2|us-gov-west-1|' aws-k8s-cni.yaml
```

All other AWS Regions – Replace *region-code* with the AWS Region that your cluster is in.

```
sed -i.bak -e 's|us-west-2|region-code|' aws-k8s-cni.yaml
```

- ii. Replace 602401143452 in the file with the account for the AWS Region that your cluster is in.

AWS GovCloud (US-East)

```
sed -i.bak -e 's|602401143452|151742754352|' aws-k8s-cni.yaml
```

AWS GovCloud (US-West)

```
sed -i.bak -e 's|602401143452|013241004608|' aws-k8s-cni.yaml
```

All other AWS Regions – Replace *account* with the value from [Amazon container image registries \(p. 386\)](#) for the AWS Region that your cluster is in.

```
sed -i.bak -e 's|602401143452|account|' aws-k8s-cni.yaml
```

- g. If you've changed any default settings for your current VPC CNI DaemonSet then compare the settings in the new manifest to the backup file that you made in a previous step.

```
diff aws-k8s-cni.yaml aws-k8s-cni-old.yaml -u
```

Edit the new manifest file and make changes to any setting values so that they match the settings in your backup file.

- h. Apply the manifest file to your cluster.

```
kubectl apply -f aws-k8s-cni.yaml
```

4. View the status of the DaemonSet.

```
kubectl get daemonset aws-node -n kube-system
```

Example output:

NAME	DESIRED	CURRENT	READY	UP-TO-DATE	AVAILABLE	NODE SELECTOR	AGE
aws-node	2	2	2	2	2	<none>	4h39m

Once the numbers in the READY, UP-TO-DATE, and AVAILABLE columns are the same, then your update is complete. Your numbers may be different than those in the previous output.

5. View the DaemonSet to confirm the changes that you made.

```
kubectl get daemonset aws-node -n kube-system -o yaml
```

Choosing pod networking use cases

The Amazon VPC CNI plugin provides networking for pods. The following table helps you understand which networking use cases you can use together and the capabilities and Amazon VPC CNI plugin settings that you can use with different Amazon EKS node types. All information in the table applies to Linux IPv4 nodes only.

Amazon EKS node type (p. 105)	Amazon EC2			Fargate
CNI custom networking (p. 306) – Assign IP addresses from a different subnet than the node's subnet	Yes	Yes	Yes	Yes (subnets controlled through Fargate profile)
External source network address translation (SNAT) (p. 305)	Yes (default is false)	Yes (default is false)	Yes (true only)	Yes (true only)
Capabilities				
Security group scope	Node	Node	Pod	Pod
Amazon VPC subnet types	Private and public	Private and public	Private only	Private only
Network policy (p. 350) (Calico)	Compatible	Compatible	Compatible Only with version 1.11.0 or later of the Amazon VPC add-on configured with <code>POD_SECURITY_GROUP_ENFORCING_MODE=standard</code>	Not supported
Pod density per node	Medium	High	Low	One
Pod launch time	Better	Best	Good	Moderate
Amazon VPC CNI plugin settings (for more information about each setting, see amazon-vpc-cni-k8s on GitHub)				
WARM_ENI_TARGET	Yes	Not applicable	Not applicable	Not applicable
WARM_IP_TARGET	Yes	Yes	Not applicable	Not applicable
MINIMUM_IP_TARGET	Yes	Yes	Not applicable	Not applicable
WARM_PREFIX_TARGET	Not applicable	Yes	Not applicable	Not applicable

Note

- You can't use IPv6 with custom networking.
- IPv6 addresses are not translated, so SNAT doesn't apply.
- You can use Calico network policy with IPv6.
- Traffic flow to and from pods with associated security groups are not subjected to Calico network policy enforcement and are limited to Amazon VPC security group enforcement only.
- IP prefixes and IP addresses are associated with standard Amazon EC2 elastic network interfaces. Pods requiring specific security groups are assigned the primary IP address of a branch network interface. You can mix pods getting IP addresses, or IP addresses from IP prefixes with pods getting branch network interfaces on the same node.

Windows nodes

Each Windows node only supports one network interface and secondary IPv4 addresses for pods. As a result, you can't use IP address prefixes or IPv6 with Windows nodes. The maximum number of pods for each node is equal to the number of IP addresses that you can assign to each Elastic network interface, minus one. Calico network policies are supported on Windows. For more information, see [Open Source Calico for Windows Containers on Amazon EKS](#). You can't use [security groups for pods \(p. 313\)](#) on Windows.

Assigning IPv6 addresses to pods and services

By default, Kubernetes assigns IPv4 addresses to your pods and services. Instead of assigning IPv4 addresses to your pods and services, you can configure your cluster to assign IPv6 addresses to them. Amazon EKS doesn't support dual-stacked pods or services. As a result, you can't assign both IPv4 and IPv6 addresses to your pods and services.

You select which IP family you want to use for your cluster when you create it. You can't change the family after you create the cluster.

Considerations for using the IPv6 family for your cluster:

- You must create a new cluster that's version 1.21 or later and specify that you want to use the IPv6 family for that cluster. You can't enable the IPv6 family for a cluster that you updated from a previous version. For instructions on how to create a new cluster, see [Creating an Amazon EKS cluster \(p. 24\)](#).
- The version of the Amazon VPC CNI add-on that you deploy to your cluster must be version 1.10.1 or later. This version or later is deployed by default with a new 1.21 or later cluster. After you deploy the add-on, you can't downgrade your Amazon VPC CNI add-on to a version lower than 1.10.1 without first removing all nodes in all node groups in your cluster.
- Windows pods and services aren't supported.
- If you use Amazon EC2 nodes, you must configure the Amazon VPC CNI add-on with IP prefix delegation and IPv6. If you choose the IPv6 family when creating your cluster, the 1.10.1 version of the add-on defaults to this configuration. This is the case for both a self-managed or Amazon EKS add-on. For more information about IP prefix delegation, see [Increase the amount of available IP addresses for your Amazon EC2 nodes \(p. 310\)](#).
- When you create a cluster, the VPC and subnets that you specify must have an IPv6 CIDR block that's assigned to the VPC and subnets that you specify. They must also have an IPv4 CIDR block assigned to them. This is because, even if you only want to use IPv6, a VPC still requires an IPv4 CIDR block to function. For more information, see [Associate an IPv6 CIDR block with your VPC](#) in the Amazon VPC User Guide.
- When you create your cluster and nodes, you must specify subnets that are configured to auto-assign IPv6 addresses. Otherwise, you can't deploy your cluster and nodes. By default, this configuration

is disabled. For more information, see [Modify the IPv6 addressing attribute for your subnet](#) in the Amazon VPC User Guide.

- The route tables that are assigned to your subnets must have routes for IPv6 addresses. For more information, see [Migrate to IPv6](#) in the Amazon VPC User Guide.
- Your security groups must allow IPv6 addresses. For more information, see [Migrate to IPv6](#) in the Amazon VPC User Guide.
- You can only use IPv6 with AWS Nitro-based Amazon EC2 or Fargate nodes.
- You can't use IPv6 with [Security groups for pods \(p. 313\)](#) with Amazon EC2 nodes. However, you can use it with Fargate nodes. If you need separate security groups for individual pods, continue using the IPv4 family with Amazon EC2 nodes, or use Fargate nodes instead.
- If you previously used [custom networking \(p. 306\)](#) to help alleviate IP address exhaustion, you can use IPv6 instead. You can't use custom networking with IPv6. If you use custom networking for network isolation, then you might need to continue to use custom networking and the IPv4 family for your clusters.
- You can't use IPv6 with nodes that are deployed on AWS Outposts.
- Pods and services are only assigned an IPv6 address. They aren't assigned an IPv4 address. Because pods are able to communicate to IPv4 endpoints through NAT on the instance itself, [DNS64](#) and [NAT64](#) aren't needed. If the traffic needs a public IP address, the traffic is then source network address translated to a public IP.
- The source IPv6 address of a pod isn't source network address translated to the IPv6 address of the node when communicating outside of the VPC. It is routed using an internet gateway or egress-only internet gateway.
- All nodes are assigned an IPv4 and IPv6 address.
- The [Amazon FSx for Lustre CSI driver \(p. 256\)](#) is not supported.
- You can use version 2.3.1 or later of the AWS Load Balancer Controller to load balance [application \(p. 378\)](#) or [network \(p. 372\)](#) traffic to IPv6 pods in IP mode, but not instance mode. For more information, see [Installing the AWS Load Balancer Controller add-on \(p. 329\)](#).
- You must attach an IPv6 IAM policy to your node IAM or CNI IAM role. Between the two, we recommend that you attach it to a CNI IAM role. For more information, see [Create IAM policy for clusters that use the IPv6 family \(p. 281\)](#) and [Step 1: Create the Amazon VPC CNI plugin IAM role \(p. 277\)](#).
- Each Fargate pod receives an IPv6 address from the CIDR that's specified for the subnet that it's deployed in. The underlying hardware unit that runs Fargate pods gets a unique IPv4 and IPv6 address from the CIDRs that are assigned to the subnet that the hardware unit is deployed in.
- We recommend that you perform a thorough evaluation of your applications, Amazon EKS add-ons, and AWS services that you integrate with before deploying IPv6 clusters. This is to ensure that everything works as expected with IPv6.
- You can't use IPv6 with AWS App Mesh.
- Use of the Amazon EC2 [Instance Metadata Service IPv6 endpoint](#) is not supported with Amazon EKS.

Deploy an IPv6 cluster and nodes

In this topic, you deploy an IPv6 Amazon VPC, an Amazon EKS cluster with the IPv6 family, and a managed node group with Amazon EC2 Amazon Linux nodes. You can't deploy Amazon EC2 Windows nodes in an IPv6 cluster. You can also deploy Fargate nodes to your cluster, though those instructions aren't provided in this topic for simplicity.

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 \(p. 24\)](#), [Managed node groups \(p. 109\)](#) and the [considerations \(p. 294\)](#) for this topic. You can only enable some settings when creating your cluster.

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.

- The `kubectl` command line tool installed on your computer or AWS CloudShell. The version must be the same, or up to two versions later than your cluster version. To install or upgrade `kubectl`, see [Installing kubectl \(p. 4\)](#). For this tutorial, the version must be 1.21 or later.
- 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.

Procedures are provided to create the resources with either `eksctl` or the AWS CLI. You can also deploy the resources using the AWS Management Console, but those instructions aren't provided in this topic for simplicity.

`eksctl`

Prerequisite

`eksctl` version 0.97.0 or later installed on your computer. To install or update to it, see the section called ["Installing eksctl" \(p. 10\)](#).

To deploy an IPv6 cluster with `eksctl`

1. Copy the following contents to a file named `ipv6-cluster.yaml`. Replace the *example values* with your own. 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 `t3.medium` with any [AWS Nitro System instance type](#). The value for `version` must be `1.21` or a later [supported Amazon EKS Kubernetes version \(p. 67\)](#). 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.

```
---  
apiVersion: eksctl.io/v1alpha5  
kind: ClusterConfig  
  
metadata:  
  name: my-cluster  
  region: region-code  
  version: "1.21"  
  
  kubernetesNetworkConfig:  
    ipFamily: IPv6  
  
  addons:  
    - name: vpc-cni  
      version: latest  
    - name: coredns  
      version: latest  
    - name: kube-proxy  
      version: latest  
  
  iam:  
    withOIDC: true  
  
  managedNodeGroups:  
    - name: my-nodegroup  
      instanceType: t3.medium
```

2. Create your cluster.

```
eksctl create cluster -f ipv6-cluster.yaml
```

Cluster creation takes several minutes. Don't proceed until you see the last line of output, which looks similar to the following output.

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

3. Confirm that default pods are assigned IPv6 addresses.

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

Example output:

NAME	READY	STATUS	RESTARTS	AGE	IP	NOMINATED
NODE	READINESS	GATES				
aws-node-rs1ts	1/1	Running	1	5m36s		
	2600:1f13:b66:8200:11a5:ade0:c590:6ac8	ip-192-168-34-75.region-				
aws-node-t74jh	1/1	Running	0	5m32s		
	2600:1f13:b66:8203:4516:2080:8ced:1ca9	ip-192-168-253-70.region-				
coredns-85d5b4454c-cw7w2	1/1	Running	0	56m		
	2600:1f13:b66:8203:34e5::1	ip-192-168-253-70.region-				
coredns-85d5b4454c-tx6n8	1/1	Running	0	56m		
	2600:1f13:b66:8203:34e5::1	ip-192-168-253-70.region-				
kube-proxy-btpbk	1/1	Running	0	5m36s		
	2600:1f13:b66:8200:11a5:ade0:c590:6ac8	ip-192-168-34-75.region-				
kube-proxy-jjk2g	1/1	Running	0	5m33s		
	2600:1f13:b66:8203:4516:2080:8ced:1ca9	ip-192-168-253-70.region-				
app=kube-dns						

4. Confirm that default services are assigned IPv6 addresses.

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

Example output:

NAME	TYPE	CLUSTER-IP	EXTERNAL-IP	PORT(S)	AGE
SELECTOR					
kube-dns	ClusterIP	fd30:3087:b6c2::a	<none>	53/UDP, 53/TCP	57m k8s-
app=kube-dns					

5. (Optional) [Deploy a sample application](#) (p. 359) or deploy the [AWS Load Balancer Controller](#) (p. 329) and a sample application to load balance [application](#) (p. 378) or [network](#) (p. 372) traffic to IPv6 pods.

6. After you've finished with the cluster and nodes that you created for this tutorial, you should clean up the resources that you created with the following command.

```
eksctl delete cluster my-cluster
```

AWS CLI

Prerequisite

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. If you use the AWS CloudShell, you may need to [install version 2.6.3 or later or 1.23.11 or later of the AWS CLI](#), because the default AWS CLI version installed in the AWS CloudShell may be an earlier version.

Important

- You must complete all steps in this procedure as the same user.
- You must complete all steps in this procedure in the same shell. Several steps use variables set in previous steps. Steps that use variables won't function properly if the variable values are set in a different shell. If you use the [AWS CloudShell](#) to complete the following procedure, remember that if you don't interact with it using your keyboard or pointer for approximately 20–30 minutes, your shell session ends. Running processes do not count as interactions.
- The instructions are written for the Bash shell, and may need adjusting in other shells.

To create your cluster with the AWS CLI

Replace all *example values* in the steps of this procedure with your own values.

1. Run the following commands to set some variables used in later steps. Replace *region-code* with the AWS Region that you want to deploy your resources in. The value can be 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. Replace *my-cluster* and *my-nodegroup* with the names for your cluster and node group. Replace *111122223333* with your account ID.

```
export region_code=region-code
export cluster_name=my-cluster
export nodegroup_name=my-nodegroup
export account_id=111122223333
```

2. Create an Amazon VPC with public and private subnets that meets Amazon EKS and IPv6 requirements.
 - a. Run the following command to set a variable for your AWS CloudFormation stack name. You can replace *my-eks-ipv6-vpc* with any name you choose.

```
export vpc_stack_name=my-eks-ipv6-vpc
```

- b. Create an IPv6 VPC using an AWS CloudFormation template.

```
aws cloudformation create-stack \
--region $region_code \
--stack-name $vpc_stack_name \
--template-url https://s3.us-west-2.amazonaws.com/amazon-eks/
cloudformation/2020-10-29/amazon-eks-ipv6-vpc-public-private-subnets.yaml
```

The stack takes a few minutes to create. Run the following command. Don't continue to the next step until the output of the command is CREATE_COMPLETE.

```
aws cloudformation describe-stacks \
--region $region_code \
--stack-name $vpc_stack_name \
```

```
--query Stacks[].StackStatus \
--output text
```

- c. Retrieve the IDs of the public subnets that were created.

```
aws cloudformation describe-stacks \
--region $region_code \
--stack-name $vpc_stack_name \
--query='Stacks[ ].Outputs[?OutputKey==`SubnetsPublic`].OutputValue' \
--output text
```

Example output:

```
subnet-0a1a56c486EXAMPLE,subnet-099e6ca77aEXAMPLE
```

- d. Enable the auto-assign IPv6 address option for the public subnets that were created.

```
aws ec2 modify-subnet-attribute \
--region $region_code \
--subnet-id subnet-0a1a56c486EXAMPLE \
--assign-ipv6-address-on-creation
aws ec2 modify-subnet-attribute \
--region $region_code \
--subnet-id subnet-099e6ca77aEXAMPLE \
--assign-ipv6-address-on-creation
```

- e. Retrieve the names of the subnets and security groups created by the template from the deployed AWS CloudFormation stack and store them in variables for use in a later step.

```
security_groups=$(aws cloudformation describe-stacks \
--region $region_code \
--stack-name $vpc_stack_name \
--query='Stacks[ ].Outputs[?OutputKey==`SecurityGroups`].OutputValue' \
--output text)

public_subnets=$(aws cloudformation describe-stacks \
--region $region_code \
--stack-name $vpc_stack_name \
--query='Stacks[ ].Outputs[?OutputKey==`SubnetsPublic`].OutputValue' \
--output text)

private_subnets=$(aws cloudformation describe-stacks \
--region $region_code \
--stack-name $vpc_stack_name \
--query='Stacks[ ].Outputs[?OutputKey==`SubnetsPrivate`].OutputValue' \
--output text)

subnets=${public_subnets},${private_subnets}
```

3. 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 *eks-cluster-role-trust-policy.json*.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
```

```

        "Service": "eks.amazonaws.com"
    },
    "Action": "sts:AssumeRole"
]
}
}
```

- b. Run the following command to set a variable for your role name. You can replace `myAmazonEKSClusterRole` with any name you choose.

```
export cluster_role_name=myAmazonEKSClusterRole
```

- c. Create the role.

```

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

- d. Retrieve the ARN of the IAM role and store it in a variable for a later step.

```

cluster_iam_role=$(aws iam get-role \
--role-name $cluster_role_name \
--query="Role.Arn" \
--output text)
```

- e. 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 $cluster_role_name
```

4. Create your cluster.

- a. The version must be [1.21](#) or a later supported Amazon EKS Kubernetes version ([p. 67](#)).

```

aws eks create-cluster \
--region $region_code \
--name $cluster_name \
--kubernetes-version 1.21 \
--role-arn $cluster_iam_role \
--resources-vpc-config subnetIds=$subnets,securityGroupIds=$security_groups \
 \
--kubernetes-network-config ipFamily=ipv6
```

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 \(p. 546\)](#).

- b. The cluster takes several minutes to create. Run the following command. Don't continue to the next step until the output from the command is ACTIVE.

```

aws eks describe-cluster \
--region $region_code \
--name $cluster_name \
--query cluster.status
```

5. Create or update a `kubeconfig` file for your cluster so that you can communicate with your cluster.

```
aws eks update-kubeconfig \
--region $region_code \
--name $cluster_name
```

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

6. Create a node IAM role.

- a. Copy the following text and save it to a file named `vpc-cni-ipv6-policy.json`.

```
{ "Version": "2012-10-17", "Statement": [ { "Effect": "Allow", "Action": [ "ec2:AssignIpv6Addresses", "ec2:DescribeInstances", "ec2:DescribeTags", "ec2:DescribeNetworkInterfaces", "ec2:DescribeInstanceTypes" ], "Resource": "*" }, { "Effect": "Allow", "Action": [ "ec2:CreateTags" ], "Resource": [ "arn:aws:ec2:*.*:network-interface/*" ] } ] }
```

- b. Create the IAM policy.

```
aws iam create-policy \
--policy-name AmazonEKS_CNI_IPv6_Policy \
--policy-document file://vpc-cni-ipv6-policy.json
```

- c. Save the following contents to a file named `node-role-trust-relationship.json`.

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

- d. Run the following command to set a variable for your role name. You can replace `AmazonEKSNodeRole` with any name you choose.

```
export node_role_name=AmazonEKSNodeRole
```

- e. Create the IAM role.

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

- f. Attach the IAM policy to the IAM role.

```
aws iam attach-role-policy \
--policy-arn arn:aws:iam::$account_id:policy/AmazonEKS_CNI_IPv6_Policy \
--role-name $node_role_name
```

Important

For simplicity in this tutorial, the policy is attached to this IAM role. In a production cluster however, we recommend attaching the policy to a separate IAM role. For more information, see [Configuring the Amazon VPC CNI plugin to use IAM roles for service accounts \(p. 277\)](#).

- g. Attach two required IAM managed policies to the IAM role.

```
aws iam attach-role-policy \
--policy-arn arn:aws:iam::aws:policy/AmazonEKSWorkerNodePolicy \
--role-name $node_role_name
aws iam attach-role-policy \
--policy-arn arn:aws:iam::aws:policy/AmazonEC2ContainerRegistryReadOnly \
--role-name $node_role_name
```

- h. Retrieve the ARN of the IAM role and store it in a variable for a later step.

```
node_iam_role=$(aws iam get-role \
--role-name $node_role_name \
--query="Role.Arn" \
--output text)
```

7. Create a managed node group.

- a. View the IDs of the subnets that you created in a previous step.

```
echo $subnets
```

Example output:

```
subnet-0a1a56c486EXAMPLE,subnet-099e6ca77aEXAMPLE,subnet-0377963d69EXAMPLE,subnet-0c05f819d5EXAMPLE
```

- b. Create the node group. Replace `0a1a56c486EXAMPLE`, `099e6ca77aEXAMPLE`, `0377963d69EXAMPLE`, and `0c05f819d5EXAMPLE` with the values returned in the output of the previous step. Be sure to remove the commas between subnet IDs from the previous output in the following command. You can replace `t3.medium` with any [AWS Nitro System instance type](#).

```
aws eks create-nodegroup \
--region $region_code \
```

```
--cluster-name $cluster_name \
--nodegroup-name $nodegroup_name \
--subnets subnet-0a1a56c486EXAMPLE subnet-099e6ca77aEXAMPLE
subnet-0377963d69EXAMPLE subnet-0c05f819d5EXAMPLE \
--instance-types t3.medium \
--node-role $node_iam_role
```

The node group takes a few minutes to create. Run the following command. Don't proceed to the next step until the output returned is ACTIVE.

```
aws eks describe-nodegroup \
--region $region_code \
--cluster-name $cluster_name \
--nodegroup-name $nodegroup_name \
--query nodegroup.status \
--output text
```

8. Confirm that the default pods are assigned IPv6 addresses in the IP column.

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

Example output:

NAME	READY	STATUS	RESTARTS	AGE	IP	NOMINATED
NODE	READINESS	GATES				
aws-node- rs1ts	1/1	Running	1	5m36s		
2600:1f13:b66:8200:11a5:ade0:c590:6ac8		ip-192-168-34-75.region-				
aws-node- t74jh	1/1	Running	0	5m32s		
2600:1f13:b66:8203:4516:2080:8ced:1ca9		ip-192-168-253-70.region-				
coredns- 85d5b4454c-cw7w2	1/1	Running	0	56m		
2600:1f13:b66:8203:34e5::		ip-192-168-253-70.region-				
coredns- 85d5b4454c-tx6n8	1/1	Running	0	56m		
2600:1f13:b66:8203:34e5::1		ip-192-168-253-70.region-				
kube-proxy- btpbk	1/1	Running	0	5m36s		
2600:1f13:b66:8200:11a5:ade0:c590:6ac8		ip-192-168-34-75.region-				
kube-proxy- jjk2g	1/1	Running	0	5m33s		
2600:1f13:b66:8203:4516:2080:8ced:1ca9		ip-192-168-253-70.region-				
coredns- 85d5b4454c-tx6n8	1/1	Running	0	56m		

9. Confirm that the default services are assigned IPv6 addresses in the IP column.

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

Example output:

NAME	TYPE	CLUSTER-IP	EXTERNAL-IP	PORT(S)	AGE
SELECTOR					
kube-dns	ClusterIP	fd30:3087:b6c2::a	<none>	53/UDP, 53/TCP	57m k8s-
app=kube-dns					

10. (Optional) [Deploy a sample application \(p. 359\)](#) or deploy the [AWS Load Balancer Controller \(p. 329\)](#) and a sample application to load balance [application \(p. 378\)](#) or [network \(p. 372\)](#) traffic to IPv6 pods.

11. After you've finished with the cluster and nodes that you created for this tutorial, you should clean up the resources that you created with the following commands. Make sure that you're not using any of the resources outside of this tutorial before deleting them.
 - a. If you're completing this step in a different shell than you completed the previous steps in, set the values of all the variables used in previous steps, replacing the *example values* with the values you specified when you completed the previous steps. If you're completing this step in the same shell that you completed the previous steps in, skip to the next step.

```
export region_code=region-code
export vpc_stack_name=my-eks-ipv6-vpc
export cluster_name=my-cluster
export nodegroup_name=my-nodegroup
export account_id=111122223333
export node_role_name=AmazonEKSNodeRole
export cluster_role_name=myAmazonEKSClusterRole
```

- b. Delete your node group.

```
aws eks delete-nodegroup --region $region_code --cluster-name $cluster_name --
nodegroup-name $nodegroup_name
```

Deletion takes a few minutes. Run the following command. Don't proceed to the next step if any output is returned.

```
aws eks list-nodegroups --region $region_code --cluster-name $cluster_name --
query nodegroups --output text
```

- c. Delete the cluster.

```
aws eks delete-cluster --region $region_code --name $cluster_name
```

The cluster takes a few minutes to delete. Before continuing make sure that the cluster is deleted with the following command.

```
aws eks describe-cluster --region $region_code --name $cluster_name
```

Don't proceed to the next step until your output is similar to the following output.

```
An error occurred (ResourceNotFoundException) when calling the DescribeCluster
operation: No cluster found for name: my-cluster.
```

- d. Delete the IAM resources that you created. Replace *AmazonEKS_CNI_IPv6_Policy* with the name you chose, if you chose a different name than the one used in previous steps.

```
aws iam detach-role-policy --role-name $cluster_role_name --policy-arm
arn:aws:iam::aws:policy/AmazonEKSClusterPolicy
aws iam detach-role-policy --role-name $node_role_name --policy-arm
arn:aws:iam::aws:policy/AmazonEKSWorkerNodePolicy
aws iam detach-role-policy --role-name $node_role_name --policy-arm
arn:aws:iam::aws:policy/AmazonEC2ContainerRegistryReadOnly
aws iam detach-role-policy --role-name $node_role_name --policy-arm
arn:aws:iam::$account_id:policy/AmazonEKS_CNI_IPv6_Policy
aws iam delete-policy --policy-arm arn:aws:iam::
$account_id:policy/AmazonEKS_CNI_IPv6_Policy
aws iam delete-role --role-name $cluster_role_name
aws iam delete-role --role-name $node_role_name
```

- e. Delete the AWS CloudFormation stack that created the VPC.

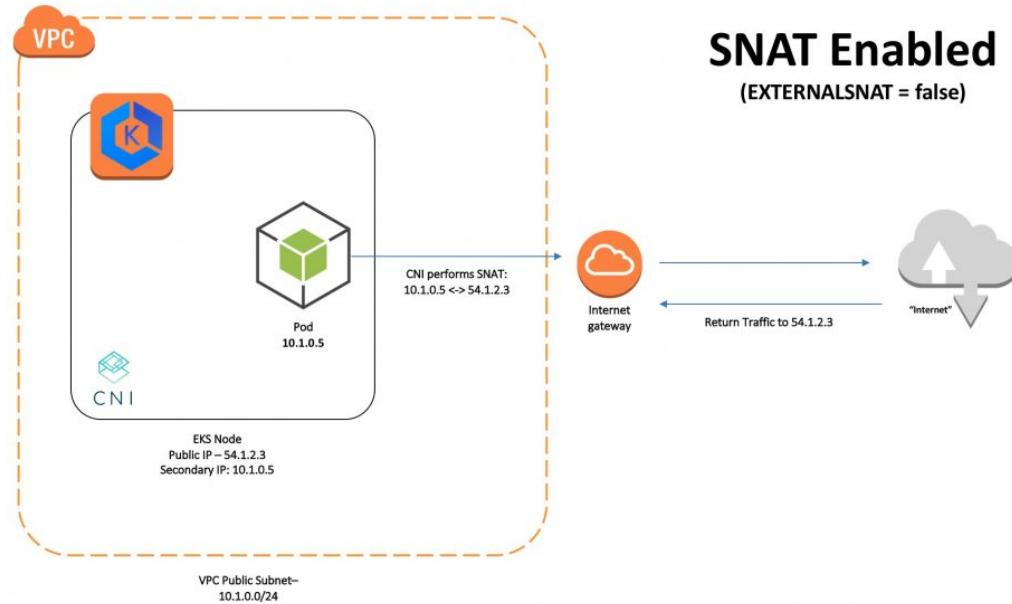
```
aws cloudformation delete-stack --region $region_code --stack-name
$vpc_stack_name
```

External source network address translation (SNAT)

Communication within a VPC (such as pod to pod) is direct between private IP addresses and requires no source network address translation (SNAT). When IP traffic is destined for an IPv4 address outside of the VPC, the [Amazon VPC CNI plugin for Kubernetes](#) translates the private IP address of each pod to the primary private IP address assigned to the primary [elastic network interface](#) of the Amazon EC2 node that the pod is running on, by default. IPv6 traffic is not source network address translated, so neither of the options in this topic apply to IPv6 traffic.

Elastic network interface is referred to as a *network interface* in the AWS Management Console and the Amazon EC2 API. Therefore, we use "network interface" in this documentation instead of "elastic network interface". The term "network interface" in this documentation always means "elastic network interface". SNAT:

- Enables pods to communicate bi-directionally with the internet. The node must be in a [public subnet](#) and have a [public](#) or [Elastic](#) IP address assigned to the primary private IP address of its primary network interface. The traffic is translated to and from the public or Elastic IP address and routed to and from the internet by an [internet gateway](#), as shown in the following picture.



SNAT is necessary because the internet gateway can only translate between the primary private and public or Elastic IP address assigned to the primary network interface of the Amazon EC2 instance node that pods are running on.

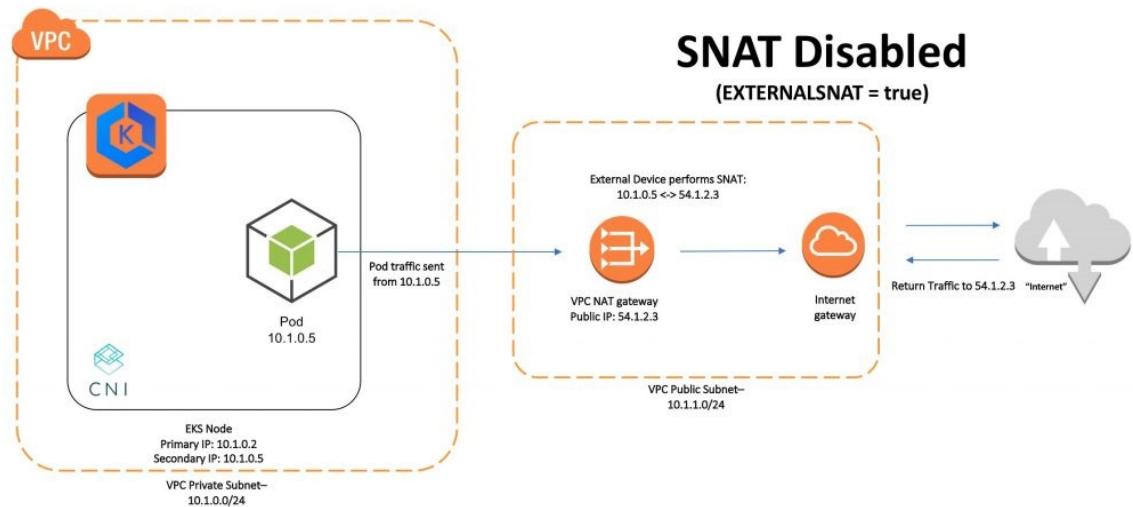
- Prevents a device in other private IP address spaces (for example, [VPC peering](#), [Transit VPC](#), or [Direct Connect](#)) from communicating directly to a pod that is not assigned the primary private IP address of the primary network interface of the Amazon EC2 instance node.

If the internet or devices in other private IP address spaces need to communicate with a pod that isn't assigned the primary private IP address assigned to the primary network interface of the Amazon EC2 instance node that the pod is running on, then:

- The node must be deployed in a private subnet that has a route to a [NAT device](#) in a public subnet.
- You need to enable external SNAT in the CNI plugin `aws-node` DaemonSet with the following command:

```
kubectl set env daemonset -n kube-system aws-node AWS_VPC_K8S_CNI_EXTERNALSNAT=true
```

After external SNAT is enabled, the CNI plugin doesn't translate a pod's private IP address to the primary private IP address assigned to the primary network interface of the Amazon EC2 instance node that the pod is running on when traffic is destined for an address outside of the VPC. Traffic from the pod's private IP address is translated by a NAT device to the public IP address of the NAT device and routed to and from the internet by an internet gateway, as shown in the following picture.



CNI custom networking

By default, when new network interfaces are allocated for pods, `ipamD` uses the security groups and subnet of the node's primary network interface. You might want your pods to use a different security group or subnet, within the same VPC as your control plane security group. For example:

- There are a limited number of IP addresses available in a subnet. This might limit the number of pods that can be created in the cluster. Using different subnets for pods allows you to increase the number of available IP addresses.
- For security reasons, your pods must use different security groups or subnets than the node's primary network interface.
- The nodes are configured in public subnets and you want the pods to be placed in private subnets using a NAT Gateway. For more information, see [External source network address translation \(SNAT\) \(p. 305\)](#).

Considerations

- The procedures in this topic require the [Amazon VPC CNI plugin for Kubernetes](#) version 1.6.3-eksbuild.2 or later.

- Enabling a custom network effectively removes an available network interface (and all of its available IP addresses for pods) from each node that uses it. The primary network interface for the node is not used for pod placement when a custom network is enabled.
- The procedure in this topic instructs the Amazon VPC CNI plugin to associate different security groups to secondary network interfaces than are associated to the primary network interface in the instance. All pods using the secondary network interfaces still share use of the secondary network interfaces and all use the same security groups.

If you want to assign different security groups to individual pods, then you can use the section called “[Security groups for pods](#)” (p. 313). Security groups for pods create additional network interfaces that can each be assigned a unique security group. Security groups for pods can be used with or without custom networking.

- You can't use custom networking if you created your cluster to use the IPv6 family. If you plan to use custom networking to help alleviate IP address exhaustion, you can use IPv6 instead. For more information, see [Assigning IPv6 addresses to pods and services](#) (p. 294).

To configure CNI custom networking

- Confirm that your currently-installed Amazon VPC CNI plugin version is 1.6.3-eksbuild.2 or later.

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

Output:

```
amazon-k8s-cni:1.6.3-eksbuild.2
```

If your version is earlier than 1.6.3-eksbuild.2, then you must update it. For more information, see the updating sections of [Managing the Amazon VPC CNI add-on](#) (p. 281).

- Associate a secondary CIDR block to your cluster's VPC. For more information, see [Associating a Secondary IPv4 CIDR Block with Your VPC](#) in the *Amazon VPC User Guide*.
- Create a subnet in your VPC for each Availability Zone, using your secondary CIDR block. Your custom subnets must be from a different VPC CIDR block than the subnet that your nodes were launched into. For more information, see [Creating a subnet in your VPC](#) in the *Amazon VPC User Guide*.
- Set the `AWS_VPC_K8S_CNI_CUSTOM_NETWORK_CFG` environment variable to `true` in the `aws-node` DaemonSet:

```
kubectl set env daemonset aws-node \
-n kube-system AWS_VPC_K8S_CNI_CUSTOM_NETWORK_CFG=true
```

- Create an ENIConfig custom resource for each subnet that you want to schedule pods in.

- Create a unique file for each network interface configuration. Each file must include the following contents with a unique value for `name`. We highly recommend using a value for `name` that matches the Availability Zone of the subnet because this makes deployment of multi-Availability Zone Auto Scaling groups simpler (see step 5c below).

In this example, a file named `us-west-2a.yaml` is created. Replace the `example values` with your own values. In this example, we follow best practices and set the value for `name` to the Availability Zone that the subnet is in. If you don't have a specific security group that you want to attach for your pods, you can leave that value empty for now. Later, you specify the node security group in the ENIConfig.

```
apiVersion: crd.k8s.amazonaws.com/v1alpha1
kind: ENIConfig
metadata:
  name: us-west-2a
spec:
  securityGroups:
    - sg-0dff111a1d11c1c11
  subnet: subnet-011b111c1f11fd11
```

Note

- Each subnet and security group combination requires its own custom resource. If you have multiple subnets in the same Availability Zone, use the following command to annotate the nodes in each subnet with the matching config name.

```
kubectl annotate node \
  node-name.region-code.compute.internal \
  k8s.amazonaws.com/eniConfig=subnet1ConfigName
```

- If you don't specify a valid security group for the VPC, and you're using version 1.8.0 or later of the VPC CNI plugin, then the security groups associated with the node's primary elastic network interface are used. If you're using a version of the plugin that is earlier than 1.8.0, then the default security group for the VPC is assigned to secondary elastic network interfaces.
 - If you specified a security group, ensure that the recommended or minimum required security group settings for the cluster, control plane and node security groups are met. For more information, see [Amazon EKS security group considerations \(p. 272\)](#).
- b. Apply each custom resource file that you created to your cluster with the following command:

```
kubectl apply -f us-west-2a.yaml
```

- c. (Optional, but recommended for multi-Availability Zone node groups) By default, Kubernetes applies the Availability Zone of a node to the `topology.kubernetes.io/zone` label. If you named your ENIConfig custom resources after each Availability Zone in your VPC, as recommended in step 5a, then you can enable Kubernetes to automatically apply the corresponding ENIConfig for the node's Availability Zone with the following command.

```
kubectl set env daemonset aws-node \
  -n kube-system ENI_CONFIG_LABEL_DEF=topology.kubernetes.io/zone
```

Note

Ensure that an annotation with the key `k8s.amazonaws.com/eniConfig` for the `ENI_CONFIG_ANNOTATION_DEF` environment variable doesn't exist in the container spec for the `aws-node` DaemonSet. If it exists, it overrides the `ENI_CONFIG_LABEL_DEF` value, and should be removed. You can check to see if the variable is set with the `kubectl describe daemonset aws-node -n kube-system | grep ENI_CONFIG_ANNOTATION_DEF` command. If no output is returned, then the variable is not set.

6. If you plan to deploy a managed node group without a launch template, or with a launch template that you haven't specified an AMI ID in, then skip to step 7 and use the **Managed, Without a launch template or with a launch template without an AMI ID specified** option. Managed node groups automatically calculates the maximum pods value for you.

If you're deploying a self-managed node group or a managed node group with a launch template that you have specified an AMI ID in, then you must determine the Amazon EKS recommend number of maximum pods for your nodes. Follow the instructions in [Amazon EKS recommended maximum](#)

pods for each Amazon EC2 instance type (p. 173), adding `--cni-custom-networking-enabled` to step 3. Note the output for use in a later step.

7. Create one of the following types of node groups. For additional instance selection criteria, see [Choosing an Amazon EC2 instance type \(p. 172\)](#). For the options that include `20`, replace it with either the value from the previous step (recommended) or your own value.

- **Self-managed** – Deploy the node group using the instructions in [Launching self-managed Amazon Linux nodes \(p. 133\)](#). Don't specify the subnets that you specified in the ENIConfig resources that you deployed. Specify the following text for the `BootstrapArguments` parameter.

```
--use-max-pods false --kubelet-extra-args '--max-pods=20'
```

- **Managed** – Deploy your node group using one of the following options:
 - **Without a launch template or with a launch template without an AMI ID specified** – Complete the procedure in [Creating a managed node group \(p. 112\)](#). Managed node groups automatically calculates the Amazon EKS recommended `max-pods` value for you.
 - **With a launch template with a specified AMI ID** – In your launch template, specify an Amazon EKS optimized AMI ID, or a custom AMI built off the Amazon EKS optimized AMI, then [deploy the node group using a launch template \(p. 124\)](#) and provide the following user data in the launch template. This user data passes arguments into the `bootstrap.sh` file. For more information about the bootstrap file, see [bootstrap.sh](#) on GitHub.

```
/etc/eks/bootstrap.sh my-cluster \
--use-max-pods false \
--kubelet-extra-args '--max-pods=20'
```

If you've created a custom AMI that is not built off the Amazon EKS optimized AMI, then you need to custom create the configuration yourself.

Note

If you want your nodes to support a significantly higher number of pods, run the script in [Amazon EKS recommended maximum pods for each Amazon EC2 instance type \(p. 173\)](#) again, adding the `--cni-prefix-delegation-enabled` option to the command. For example, `110` is returned for an `m5.large` instance type. To enable this capability, see [Increase the amount of available IP addresses for your Amazon EC2 nodes \(p. 310\)](#). You can use this capability with custom networking.

8. After your node group is created, record the security group that was created for the subnet and apply the security group to the associated ENIConfig. Edit each ENIConfig with the following command, replacing `eniconfig-name` with your value:

```
kubectl edit eniconfig.crd.k8s.amazonaws.com/eniconfig-name
```

If you followed best practices in step 5, the `eniconfig-name` corresponds to the Availability Zone name.

The `spec` section should look like the following example spec:

```
spec:
  securityGroups:
  - sg-0dff222a2d22c2c22
  subnet: subnet-022b222c2f22fdf22
```

Note

If you use the security group that was created, ensure that the recommended or minimum required security group settings for the cluster, control plane and node security groups are met. For more information, see [Amazon EKS security group considerations \(p. 272\)](#).

9. If you had any nodes in your cluster with running pods before you switched to the custom CNI networking feature, you should cordon and drain the nodes to gracefully shutdown the pods and then terminate the nodes. Only new nodes that are registered with the `k8s.amazonaws.com/eniConfig` label use the custom networking feature.

Increase the amount of available IP addresses for your Amazon EC2 nodes

By default, the number of IP addresses available to assign to pods is based on the number of IP addresses assigned to Elastic network interfaces, and the number of network interfaces attached to your Amazon EC2 node. You can configure version 1.9.0 or later of the Amazon VPC CNI add-on to assign /28 IPv4 address prefixes. With 1.10.1 or later of the add-on, you can still assign /28 IPv4 address prefixes, but if your cluster is version 1.21 or later, and you've [configured it for IPv6 \(p. 294\)](#), you can assign /80 IPv6 address prefixes instead.

When configured for prefix assignment, the CNI add-on can assign significantly more IP addresses to a network interface than it can when you assign individual IP addresses. The node can then assign significantly more available IP addresses to pods. For more information about the Amazon EC2 capability that enables the add-on to do this, see [Assigning prefixes to Amazon EC2 network interfaces](#) in the Amazon EC2 User Guide for Linux Instances.

Without enabling this capability, the add-on must make more Amazon EC2 application programming interface (API) calls to configure the network interfaces and IP addresses necessary for pod connectivity. The frequency of these API calls combined with a high number of network interfaces in each VPC can lead to longer pod and instance launch times as clusters grow to larger sizes. This results in scaling delays to meet the demand of large and spiky workloads, and adds cost and management overhead because you need to provision additional clusters and VPCs to meet scaling requirements. See the [Kubernetes scalability thresholds](#) for more information.

Considerations

- AWS Nitro-based nodes use this capability. Instances that aren't Nitro-based continue to allocate individual secondary IP addresses, but have a significantly lower number of IP addresses to assign to pods than Nitro-based instances.
- Once you configure the add-on to assign prefixes to network interfaces, you can't downgrade your Amazon VPC CNI add-on to a version lower than 1.9.0 (or 1.10.1) without removing all nodes in all node groups in your cluster.
- Your VPC must have enough available contiguous /28 IPv4 address blocks to support this capability.
- Each instance type supports a maximum number of pods. If your managed node group consists of multiple instance types, the smallest number of maximum pods for an instance in the cluster is applied to all nodes in the cluster.
- If you have an existing managed node group, the next AMI or launch template update of your node group results in new worker nodes coming up with the new IP address prefix assignment-enabled `max-pod` value.

Prerequisites

- An existing cluster. To deploy one, see [Creating an Amazon EKS cluster \(p. 24\)](#).

- Version 1.9.0 or later (for version 1.20 or earlier clusters or 1.21 or later clusters configured for IPv4) or 1.10.1 or later (for version 1.21 or later clusters configured for IPv6) of the Amazon VPC CNI add-on deployed to your cluster.

To increase the amount of available IP addresses for your Amazon EC2 nodes

1. Confirm that your currently-installed Amazon VPC CNI version is 1.9.0 or 1.10.1 or later.

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

Output:

```
amazon-k8s-cni:v1.10.1-eksbuild.1
```

If your version is earlier than 1.9.0, then you must update it. For more information, see the updating sections of [Managing the Amazon VPC CNI add-on \(p. 281\)](#).

2. Enable the parameter to assign prefixes to network interfaces for the Amazon VPC CNI DaemonSet. When you deploy a 1.21 or later cluster, version 1.10.1 or later of the VPC CNI add-on is deployed with it. If you created the cluster with the IPv6 family, this setting was set to `true` by default. If you created the cluster with the IPv4 family, this setting was set to `false` by default.

```
kubectl set env daemonset aws-node -n kube-system ENABLE_PREFIX_DELEGATION=true
```

Important

Even if your subnet has free IP addresses, if the subnet does not have any contiguous /28 blocks available, you will see the following error in the VPC CNI logs:

```
"failed to allocate a private IP/Prefix address: InsufficientCidrBlocks: The specified subnet does not have enough free cidr blocks to satisfy the request"
```

This can happen due to fragmentation of existing secondary IP addresses spread out across a subnet. To resolve this error, either create a new subnet and launch pods there, or use an Amazon EC2 subnet CIDR reservation to reserve space within a subnet for use with prefix assignment. For more information, see [Subnet CIDR reservations](#) in the Amazon VPC User Guide.

3. If you plan to deploy a managed node group without a launch template, or with a launch template that you haven't specified an AMI ID in, and you're using a version of the VPC add-on at or later than the versions listed in the prerequisites, then skip to the next step. Managed node groups automatically calculates the maximum number of pods for you.

If you're deploying a self-managed node group or a managed node group with a launch template that you have specified an AMI ID in, then you must determine the Amazon EKS recommend number of maximum pods for your nodes. Follow the instructions in [Amazon EKS recommended maximum pods for each Amazon EC2 instance type \(p. 173\)](#), adding `--cni-prefix-delegation-enabled` to step 3. Note the output for use in a later step.

Important

Managed node groups enforces a maximum number on the value of `maxPods`. For instances with less than 30 vCPUs the maximum number is 110 and for all other instances the maximum number is 250. This maximum number is applied whether prefix delegation is enabled or not.

4. If you're using a 1.21 or later cluster configured for IPv6, skip to the next step.

Specify the parameters in one of the following options. To determine which option is right for you and what value to provide for it, see [WARM_PREFIX_TARGET](#), [WARM_IP_TARGET](#), and [MINIMUM_IP_TARGET](#) on GitHub.

You can replace the *example values* with a value greater than zero.

- `WARM_PREFIX_TARGET`

```
kubectl set env ds aws-node -n kube-system WARM_PREFIX_TARGET=1
```

- `WARM_IP_TARGET` or `MINIMUM_IP_TARGET` – If either value is set, it overrides any value set for `WARM_PREFIX_TARGET`.

```
kubectl set env ds aws-node -n kube-system WARM_IP_TARGET=5
```

```
kubectl set env ds aws-node -n kube-system MINIMUM_IP_TARGET=2
```

5. Create one of the following types of node groups with at least one Amazon EC2 Nitro Amazon Linux 2 instance type. For a list of Nitro instance types, see [Instances built on the Nitro System](#) in the Amazon EC2 User Guide for Linux Instances. This capability is not supported on Windows. For the options that include `110`, replace it with either the value from step 3 (recommended), or your own value.

- **Self-managed** – Deploy the node group using the instructions in [Launching self-managed Amazon Linux nodes \(p. 133\)](#). Specify the following text for the `BootstrapArguments` parameter.

```
--use-max-pods false --kubelet-extra-args '--max-pods=110'
```

If you're using `eksctl` to create the node group, you can use the following command.

```
eksctl create nodegroup --cluster my-cluster --managed=false --max-pods-per-node 110
```

- **Managed** – Deploy your node group using one of the following options:

- **Without a launch template or with a launch template without an AMI ID specified** – Complete the procedure in [Creating a managed node group \(p. 112\)](#). Managed node groups automatically calculates the Amazon EKS recommended `max-pods` value for you.

- **With a launch template with a specified AMI ID** – In your launch template, specify an Amazon EKS optimized AMI ID, or a custom AMI built off the Amazon EKS optimized AMI, then [deploy the node group using a launch template \(p. 124\)](#) and provide the following user data in the launch template. This user data passes arguments into the `bootstrap.sh` file. For more information about the bootstrap file, see [bootstrap.sh](#) on GitHub.

```
/etc/eks/bootstrap.sh my-cluster \
--use-max-pods false \
--kubelet-extra-args '--max-pods=110'
```

If you're using `eksctl` to create the node group, you can use the following command.

```
eksctl create nodegroup --cluster my-cluster --max-pods-per-node 110
```

If you've created a custom AMI that is not built off the Amazon EKS optimized AMI, then you need to custom create the configuration yourself.

Note

If you also want to assign IP addresses to pods from a different subnet than the instance's, then you need to enable the capability in this step. For more information, see [CNI custom networking \(p. 306\)](#).

- Once your nodes are deployed, view the nodes in your cluster.

```
kubectl get nodes
```

Example output:

NAME	STATUS	ROLES	AGE	VERSION
ip-192-168-22-103.region-code.compute.internal <i>eks-6b7464</i>	Ready	<none>	19m	v1.20.4-
ip-192-168-97-94.region-code.compute.internal <i>eks-6b7464</i>	Ready	<none>	19m	v1.20.4-

- Describe one of the nodes to determine the `max-pods` for the node.

```
kubectl describe node node-name ip-192-168-22-103.region-code.compute.internal
```

Example output:

```
...
Allocatable:
  attachable-volumes-aws-ebs: 25
  cpu: 1930m
  ephemeral-storage: 76224326324
  hugepages-1Gi: 0
  hugepages-2Mi: 0
  memory: 7244720Ki
  pods: 110
...
```

In the previous output, 110 is the maximum number of pods that Kubernetes will deploy to the node.

Security groups for pods

Security groups for pods integrate Amazon EC2 security groups with Kubernetes pods. You can use Amazon EC2 security groups to define rules that allow inbound and outbound network traffic to and from pods that you deploy to nodes running on many Amazon EC2 instance types and Fargate. For a detailed explanation of this capability, see the [Introducing security groups for pods](#) blog post.

Considerations

Before deploying security groups for pods, consider the following limits and conditions:

- Pods running on Amazon EC2 nodes can be in any [Amazon EKS supported Kubernetes version \(p. 67\)](#), but pods running on [Fargate \(p. 153\)](#) must be in a 1.18 or later cluster.
- Security groups for pods can't be used with Windows nodes.
- Security groups for pods can't be used with clusters configured for the IPv6 family that contain Amazon EC2 nodes. You can however, use security groups for pods with clusters configured for the IPv6 family that contain only Fargate nodes. For more information, see [Assigning IPv6 addresses to pods and services \(p. 294\)](#).

- Security groups for pods are supported by most [Nitro-based Amazon EC2 instance families](#), including the `m5`, `c5`, `r5`, `p3`, `m6g`, `c6g`, and `r6g` instance families. The `t3` instance family is not supported. For a complete list of supported instances, see the [limits.go](#) file on Github. Your nodes must be one of the listed instance types that have `IsTrunkingCompatible: true` in that file.
- If you're also using pod security policies to restrict access to pod mutation, then the `eks-vpc-resource-controller` and `vpc-resource-controller` Kubernetes service accounts must be specified in the Kubernetes ClusterRoleBinding for the role that your psp is assigned to. If you're using the [default Amazon EKS psp, role, and ClusterRoleBinding \(p. 501\)](#), this is the `eks:podsecuritypolicy:authenticated` ClusterRoleBinding. For example, you add the service accounts to the `subjects:` section, as shown in the following example:

```
...
subjects:
  - kind: Group
    apiGroup: rbac.authorization.k8s.io
    name: system:authenticated
  - kind: ServiceAccount
    name: vpc-resource-controller
  - kind: ServiceAccount
    name: eks-vpc-resource-controller
```

- If you're using [custom networking \(p. 306\)](#) and security groups for pods together, the security group specified by security groups for pods is used instead of the security group specified in the `ENIconfig`.
- If you're using version 1.10.2 or earlier of the Amazon VPC CNI plugin and you include the `terminationGracePeriodSeconds` setting in your pod spec, the value for the setting can't be zero.
- If you're using version 1.10 or earlier of the Amazon VPC CNI plugin, or version 1.11 with `POD_SECURITY_GROUP_ENFORCING_MODE=strict`, which is the default setting, then Kubernetes services of type `NodePort` and `LoadBalancer` using instance targets with an `externalTrafficPolicy` set to `Local` are not supported with pods that you assign security groups to. For more information about using a load balancer with instance targets, see [Network load balancing on Amazon EKS \(p. 372\)](#). If you're using version 1.11 or later of the plugin with `POD_SECURITY_GROUP_ENFORCING_MODE=standard`, then instance targets with an `externalTrafficPolicy` set to `Local` are supported.
- If you're using version 1.10 or earlier of the Amazon VPC CNI plugin or version 1.11 with `POD_SECURITY_GROUP_ENFORCING_MODE=strict`, which is the default setting, source NAT is disabled for outbound traffic from pods with assigned security groups so that outbound security group rules are applied. To access the internet, pods with assigned security groups must be launched on nodes that are deployed in a private subnet configured with a NAT gateway or instance. Pods with assigned security groups deployed to public subnets are not able to access the internet.

If you're using version 1.11 or later of the plugin with `POD_SECURITY_GROUP_ENFORCING_MODE=standard`, then pod traffic destined for outside of the VPC is translated to the IP address of the instance's primary network interface. For this traffic, the rules in the security groups for the primary network interface are used, rather than the rules in the pod's security groups.

- To use Calico network policy with pods that have associated security groups, you must use version 1.11.0 or later of the Amazon VPC CNI plugin and set `POD_SECURITY_GROUP_ENFORCING_MODE=standard`. Otherwise, traffic flow to and from pods with associated security groups are not subjected to [Calico network policy \(p. 350\)](#) enforcement and are limited to Amazon EC2 security group enforcement only. To update your Amazon VPC CNI version, see [Managing the Amazon VPC CNI add-on \(p. 281\)](#).
- Pods running on Amazon EC2 nodes that use security groups in clusters that use `NodelocalDNSCache` are only supported with version 1.11.0 or later of the Amazon VPC CNI plugin and with `POD_SECURITY_GROUP_ENFORCING_MODE=standard`. To update your Amazon VPC CNI plugin version, see [Managing the Amazon VPC CNI add-on \(p. 281\)](#).

Configure the Amazon VPC CNI add-on for security groups for pods

To deploy security groups for pods

If you're using security groups for Fargate pods only, and don't have any Amazon EC2 nodes in your cluster, skip to [Deploy an example application \(p. 319\)](#).

1. Check your current CNI plugin version with the following command:

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

Example output:

```
amazon-k8s-cni-init:1.7.5-eksbuild.1
amazon-k8s-cni:1.7.5-eksbuild.1
```

If your CNI plugin version is earlier than 1.7.7, then update your CNI plugin to version 1.7.7 or later. For more information, see [Updating the Amazon VPC CNI Amazon EKS add-on \(p. 284\)](#).

2. Add the [AmazonEKSVPCCResourceController](#) managed policy to the [cluster role \(p. 473\)](#) that is associated with your Amazon EKS cluster. The policy allows the role to manage network interfaces, their private IP addresses, and their attachment and detachment to and from network instances.
 - a. Determine the ARN of your cluster IAM role. Replace `my-cluster` with the name of your cluster.

```
aws eks describe-cluster --name my-cluster --query cluster.roleArn --output text
```

Output

```
arn:aws:iam::111122223333:role/eksClusterRole
```

- b. The following command adds the policy to a cluster role named `eksClusterRole`. Replace `eksClusterRole` with the name of your role.

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

3. Enable the Amazon VPC CNI add-on to manage network interfaces for pods by setting the `ENABLE_POD_ENI` variable to `true` in the `aws-node` DaemonSet. Once this setting is set to `true`, for each node in the cluster the add-on adds a label with the value `vpc.amazonaws.com/has-trunk-attached=true`. The VPC resource controller creates and attaches one special network interface called a *trunk network interface* with the description `aws-k8s-trunk-eni`.

```
kubectl set env daemonset aws-node -n kube-system ENABLE_POD_ENI=true
```

Note

The trunk network interface is included in the maximum number of network interfaces supported by the instance type. For a list of the maximum number of network interfaces supported by each instance type, see [IP addresses per network interface per instance type](#) in the *Amazon EC2 User Guide for Linux Instances*. If your node already has the maximum number of standard network interfaces attached to it then the VPC resource controller will reserve a space. You will have to scale down your running pods enough for the controller

to detach and delete a standard network interface, create the trunk network interface, and attach it to the instance.

You can see which of your nodes have `aws-k8s-trunk-eni` set to `true` with the following command. If `No resources found` is returned, then wait several seconds and try again. The previous step requires restarting the CNI pods, which takes several seconds.

```
kubectl get nodes -o wide -l vpc.amazonaws.com/has-trunk-attached=true
```

Once the trunk network interface is created, pods are assigned secondary IP addresses from the trunk or standard network interfaces. The trunk interface is automatically deleted if the node is deleted.

When you deploy a security group for a pod in a later step, the VPC resource controller creates a special network interface called a *branch network interface* with a description of `aws-k8s-branch-eni` and associates the security groups to it. Branch network interfaces are created in addition to the standard and trunk network interfaces attached to the node. If you are using liveness or readiness probes, then you also need to disable TCP early demux, so that the `kubelet` can connect to pods on branch network interfaces using TCP. To disable TCP early demux, run the following command:

```
kubectl patch daemonset aws-node \
  -n kube-system \
  -p '{"spec": {"template": {"spec": {"initContainers": [{"env": [{"name": "DISABLE_TCP_EARLY_DEMUX", "value": "true"}], "name": "aws-vpc-cni-init"}]}}}}'
```

Note

If you're using 1.11.0 or later of the AWS VPC CNI add-on and set `POD_SECURITY_GROUP_ENFORCING_MODE=standard`, as described in the next step, then you don't need to run the previous command.

4. If your cluster uses NodeLocal DNSCache, or you want to use Calico network policy with your pods that have their own security groups, or you have Kubernetes services of type `NodePort` and `LoadBalancer` using instance targets with an `externalTrafficPolicy` set to `Local` for pods that you want to assign security groups to, then you must be using version 1.11.0 or later of the AWS VPC CNI add-on, and you must enable the following setting:

```
kubectl set env daemonset aws-node -n kube-system
  POD_SECURITY_GROUP_ENFORCING_MODE=standard
```

Important

- Pod security group rules aren't applied to traffic between pods or between pods and services, such as `kubelet` or `nodeLocalDNS`, that are on the same node.
- Outbound traffic from pods to addresses outside of the VPC is network address translated to the IP address of the instance's primary network interface (unless you've also set `AWS_VPC_K8S_CNI_EXTERNALSNAT=true`). For this traffic, the rules in the security groups for the primary network interface are used, rather than the rules in the pod's security groups.
- For this setting to apply to existing pods, you must restart the pods or the nodes that the pods are running on.

5. Create a namespace to deploy resources to.

```
kubectl create namespace my-namespace
```

6. Deploy an Amazon EKS SecurityGroupPolicy to your cluster.

- a. Save the following example security policy to a file named `my-security-group-policy.yaml`. You can replace `podSelector` with `serviceAccountSelector` if you'd rather select pods based on service account labels. You must specify one selector or the other. An empty `podSelector` (example: `podSelector: {}`) selects all pods in the namespace. An empty `serviceAccountSelector` selects all service accounts in the namespace. You must specify 1-5 security group IDs for `groupIds`. If you specify more than one ID, then the combination of all the rules in all the security groups are effective for the selected pods.

```
apiVersion: vpcresources.k8s.aws/v1beta1
kind: SecurityGroupPolicy
metadata:
  name: my-security-group-policy
  namespace: my-namespace
spec:
  podSelector:
    matchLabels:
      role: my-role
  securityGroups:
    groupIds:
      - sg-abc123
```

Important

- The security groups that you specify in the policy must exist. If they don't exist, then, when you deploy a pod that matches the selector, your pod remains stuck in the creation process. If you describe the pod, you'll see an error message similar to the following one: An `error occurred (InvalidSecurityGroupID.NotFound)` when calling the `CreateNetworkInterface` operation: The `securityGroup ID 'sg-abc123' does not exist`.
- The security group must allow inbound communication from the cluster security group (for `kubelet`) over any ports you've configured probes for.
- The security group must allow outbound communication to a security group that allows inbound TCP and UDP port 53 (for CoreDNS pods) from it.
- If you're using the security group policy with Fargate, make sure that your security group has rules that allow the pods to communicate with the Kubernetes control plane. The easiest way to do this is to specify the cluster security group as one of the security groups.
- Security group policies only apply to newly scheduled pods. They do not affect running pods.

- b. Deploy the policy.

```
kubectl apply -f my-security-group-policy.yaml
```

7. Deploy a sample application with a label that matches the `my-role` value for `podselector` that you specified in the previous step.

 - a. Save the following contents to a file.

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: my-deployment
  namespace: my-namespace
  labels:
    app: my-app
spec:
  replicas: 1
```

```
selector:
  matchLabels:
    app: my-app
template:
  metadata:
    labels:
      app: my-app
      role: my-role
spec:
  containers:
    - name: my-container
      image: my-image
      ports:
        - containerPort: 80
```

- b. Deploy the application with the following command. When you deploy the application, the CNI plugin matches the `role` label and the security groups that you specified in the previous step are applied to the pod.

```
kubectl apply -f my-security-group-policy.yaml
```

Note

- If your pod is stuck in the `Waiting` state and you see `Insufficient permissions: Unable to create Elastic Network Interface`, when you describe the pod, confirm that you added the IAM policy to the IAM cluster role in a previous step.
- If your pod is stuck in the `Pending` state, confirm that your node instance type has `IsTrunkingCompatible: true` in `limits.go` on GitHub and that the maximum number of branch network interfaces supported by the instance type multiplied times the number of nodes in your node group hasn't already been met. For example, an `m5.large` instance supports nine branch network interfaces. If your node group has five nodes, then a maximum of 45 branch network interfaces can be created for the node group. The 46th pod that you attempt to deploy will sit in `Pending` state until another pod that has associated security groups is deleted.

If you run `kubectl describe pod my-deployment-xxxxxxxxxxxxx -n my-namespace` and see a message similar to the following message, then it can be safely ignored. This message might appear when the CNI plugin tries to set up host networking and fails while the network interface is being created. The CNI plugin logs this event until the network interface is created.

```
Failed to create pod sandbox: rpc error: code = 500
Unknown desc = failed to set up sandbox container
"e24268322e55c8185721f52df6493684f6c2c3bf4fd59c9c121fd4cdc894579f" network for pod
"my-deployment-59f5f68b58-c89wx": networkPlugin
cni failed to set up pod "my-deployment-59f5f68b58-c89wx_my-namespace" network: add
cmd: failed to assign an IP address to container
```

You cannot exceed the maximum number of pods that can be run on the instance type. For a list of the maximum number of pods that you can run on each instance type, see [eni-max-pods.txt](#) on GitHub. When you delete a pod that has associated security groups, or delete the node that the pod is running on, the VPC resource controller deletes the branch network interface. If you delete a cluster with pods using pods for security groups, then the controller does not delete the branch network interfaces, so you'll need to delete them yourself.

Deploy an example application

To use security groups for pods, you must have an existing security group and [Deploy an Amazon EKS SecurityGroupPolicy \(p. 320\)](#) to your cluster, as described in the following procedure. The following steps show you how to use the security group policy for a pod. Unless otherwise noted, complete all steps from the same terminal because variables are used in the following steps that don't persist across terminals.

To deploy an example pod with a security group

1. Create a security group to use with your pod. The steps that follow help you create a simple security group for illustrative purposes only. In a production cluster, your rules will likely be different.
 - a. Retrieve the IDs of your cluster's VPC and cluster security group. Replace `my-cluster` with the name of your cluster.

```
my_cluster_name=my-cluster
my_cluster_vpc_id=$(aws eks describe-cluster \
    --name $my_cluster_name \
    --query cluster.resourcesVpcConfig.vpcId \
    --output text)
my_cluster_security_group_id=$(aws eks describe-cluster \
    --name $my_cluster_name \
    --query cluster.resourcesVpcConfig.clusterSecurityGroupId \
    --output text)
```

- b. Create the security group for your pod. Replace `my-pod-security-group` with your own. Note the ID of the security group returned in the output after running the commands. You'll use it in a later step.

```
my_pod_security_group_name=my-pod-security-group
aws ec2 create-security-group \
    --vpc-id $my_cluster_vpc_id \
    --group-name $my_pod_security_group_name \
    --description "My pod security group"
my_pod_security_group_id=$(aws ec2 describe-security-groups \
    --filters Name=group-name,Values=$my_pod_security_group_name \
    --query 'SecurityGroups[].GroupId' \
    --output text)
echo $my_pod_security_group_id
```

- c. Allow TCP and UDP port 53 traffic from the pod security group that you created in the previous step to your cluster security group. If you want the DNS traffic from your pod to flow to a different security group than your cluster security group, then replace `$my_cluster_security_group_id` with your own security group ID. Note the value returned for `SecurityGroupRuleId` in the output returned for each of the commands. You'll use them in a later step.

```
aws ec2 authorize-security-group-ingress \
    --group-id $my_cluster_security_group_id \
    --protocol tcp --port 53 --source-group $my_pod_security_group_id
aws ec2 authorize-security-group-ingress \
    --group-id $my_cluster_security_group_id \
    --protocol udp --port 53 --source-group $my_pod_security_group_id
```

- d. Allow inbound traffic to your pod security group from any pod the security group is associated to over any protocol and port. The security group has a default outbound rule that allows outbound traffic from the pods that your security group is associated with to any destination over any protocol and port.

```
aws ec2 authorize-security-group-ingress \
--group-id $my_pod_security_group_id \
--protocol -1 --port -1 --source-group $my_pod_security_group_id
```

2. Create a Kubernetes namespace to deploy resources to.

```
kubectl create namespace my-namespace
```

3. Deploy an Amazon EKS SecurityGroupPolicy to your cluster.

- a. Save the following example security policy to a file named *my-security-group-policy.yaml*. You can replace podSelector with serviceAccountSelector if you'd rather select pods based on service account labels. You must specify one selector or the other. An empty podSelector (example: podSelector: {}) selects all pods in the namespace. An empty serviceAccountSelector selects all service accounts in the namespace. You must specify 1-5 security group IDs for groupIds. If you specify more than one ID, then the combination of all the rules in all the security groups are effective for the selected pods. Replace *sg-05b1d815d1EXAMPLE* with the ID of the security group that you noted in a previous step when you created the security group for your pod.

```
apiVersion: vpcresources.k8s.aws/v1beta1
kind: SecurityGroupPolicy
metadata:
  name: my-security-group-policy
  namespace: my-namespace
spec:
  podSelector:
    matchLabels:
      role: my-role
  securityGroups:
    groupIds:
      - sg-05b1d815d1EXAMPLE
```

Important

The security group or groups that you specify for your pods must meet the following criteria:

- They must exist. If they don't exist, then, when you deploy a pod that matches the selector, your pod remains stuck in the creation process. If you describe the pod, you'll see an error message similar to the following one: An error occurred (InvalidSecurityGroupID.NotFound) when calling the CreateNetworkInterface operation: The securityGroup ID '*sg-05b1d815d1EXAMPLE*' does not exist.
- They must allow inbound communication from the [cluster security group \(p. 272\)](#) (for kubelet) over any ports that you've configured probes for.
- They must allow outbound communication over TCP and UDP ports 53 to a security group assigned to pods running CoreDNS. The security group for your CoreDNS pods must allow inbound TCP and UDP port 53 traffic from your security group.
- They must have necessary inbound and outbound rules to communicate with other pods that they need to communicate with.
- They must have rules that allow the pods to communicate with the Kubernetes control plane if you're using the security group with Fargate. The easiest way to do this is to specify the cluster security group as one of the security groups.

Security group policies only apply to newly scheduled pods. They do not affect running pods.

- b. Deploy the policy.

```
kubectl apply -f my-security-group-policy.yaml
```

4. Deploy a sample application with a label that matches the *my-role* value for *podSelector* that you specified in the previous step.

- a. Save the following contents to a file named *sample-application.yaml*.

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: my-deployment
  namespace: my-namespace
  labels:
    app: my-app
spec:
  replicas: 4
  selector:
    matchLabels:
      app: my-app
  template:
    metadata:
      labels:
        app: my-app
        role: my-role
    spec:
      terminationGracePeriodSeconds: 120
      containers:
        - name: nginx
          image: public.ecr.aws/nginx/nginx:1.21
          ports:
            - containerPort: 80
---
apiVersion: v1
kind: Service
metadata:
  name: my-app
  namespace: my-namespace
  labels:
    app: my-app
spec:
  selector:
    app: my-app
  ports:
    - protocol: TCP
      port: 80
      targetPort: 80
```

- b. Deploy the application with the following command. When you deploy the application, the CNI plugin matches the *role* label and the security groups that you specified in the previous step are applied to the pod.

```
kubectl apply -f sample-application.yaml
```

Note

- If your pod is stuck in the Waiting state and you see Insufficient permissions: Unable to create Elastic Network Interface. when you describe the pod, confirm that you added the IAM policy to the IAM cluster role in a previous step.
- If your pod is stuck in the Pending state, confirm that your node instance type is listed in `limits.go` and that the maximum number of branch network interfaces supported by the instance type multiplied times the number of nodes in your node group hasn't already been met. For example, an `m5.large` instance supports nine branch network interfaces. If your node group has five nodes, then a maximum of 45 branch network interfaces can be created for the node group. The 46th pod that you attempt to deploy will sit in Pending state until another pod that has associated security groups is deleted.

If you run `kubectl describe pod my-deployment-xxxxxxxxx-xxxx -n my-namespace` and see a message similar to the following message, then it can be safely ignored. This message might appear when the CNI plugin tries to set up host networking and fails while the network interface is being created. The CNI plugin logs this event until the network interface is created.

```
Failed to create pod sandbox: rpc error: code = 
Unknown desc = failed to set up sandbox container
"e24268322e55c8185721f52df6493684f6c2c3bf4fd59c9c121fd4cdc894579f" network for pod
"my-deployment-5df6f7687b-4fbjm": networkPlugin
cni failed to set up pod "my-deployment-5df6f7687b-4fbjm-c89wx_my-namespace"
network: add cmd: failed to assign an IP address to container
```

You can't exceed the maximum number of pods that can be run on the instance type. For a list of the maximum number of pods that you can run on each instance type, see [eni-max-pods.txt](#) on GitHub. When you delete a pod that has associated security groups, or delete the node that the pod is running on, the VPC resource controller deletes the branch network interface. If you delete a cluster with pods using pods for security groups, then the controller does not delete the branch network interfaces, so you'll need to delete them yourself.

5. View the pods deployed with the sample application. For the remainder of this topic, this terminal is referred to as TerminalA.

```
kubectl get pods -n my-namespace -o wide
```

Output

NAME	READY	STATUS	RESTARTS	AGE	IP
NODE			NOMINATED NODE	READINESS GATES	
my-deployment-5df6f7687b-4fbjm	1/1	Running	0	7m51s	192.168.53.48
ip-192-168-33-28.region-code.compute.internal		<none>			<none>
my-deployment-5df6f7687b-j9fl4	1/1	Running	0	7m51s	192.168.70.145
ip-192-168-92-33.region-code.compute.internal		<none>			<none>
my-deployment-5df6f7687b-rjxcz	1/1	Running	0	7m51s	192.168.73.207
ip-192-168-92-33.region-code.compute.internal		<none>			<none>
my-deployment-5df6f7687b-zmb42	1/1	Running	0	7m51s	192.168.63.27
ip-192-168-33-28.region-code.compute.internal		<none>			<none>

6. In a separate terminal, shell into one of the pods. For the remainder of this topic, this terminal is referred to as TerminalB. Replace `5df6f7687b-4fbjm` with the ID of one of the pods returned in your output from the previous step.

```
kubectl exec -it -n my-namespace my-deployment-5df6f7687b-4fbjm -- /bin/bash
```

7. From the shell in TerminalB, confirm that the sample application works.

```
curl my-app
```

Output

```
<!DOCTYPE html>
<html>
<head>
<title>Welcome to nginx!</title>
...

```

You received the output because all pods running the application are associated with the security group that you created. That group contains a rule that allows all traffic between all pods that the security group is associated to. DNS traffic is allowed outbound from that security group to the cluster security group, which is associated with your nodes. The nodes are running the CoreDNS pods, which your pods did a name lookup to.

8. From TerminalA, remove the security group rules that allow DNS communication to the cluster security group from your security group. If you didn't add the DNS rules to the cluster security group in a previous step, then replace `$my_cluster_security_group_id` with the ID of the security group that you created the rules in. Replace `sgr-0407e21efdEXAMPLE` and `sgr-04dedf9701EXAMPLE` with the IDs of the security group rules that you noted in a previous step.

```
aws ec2 revoke-security-group-ingress --group-id $my_cluster_security_group_id --
security-group-rule-ids sgr-0407e21efdEXAMPLE
aws ec2 revoke-security-group-ingress --group-id $my_cluster_security_group_id --
security-group-rule-ids sgr-04dedf9701EXAMPLE
```

9. From TerminalB, attempt to access the application again.

```
curl my-app
```

The attempt fails because the pod is no longer able to access the CoreDNS pods, which have the cluster security group associated to them. The cluster security group no longer has the security group rules that allow DNS communication from the security group associated to your pod.

If you attempt to access the application using the IP addresses returned for one of the pods in a previous step, you still receive a response because all ports are allowed between pods that have the security group associated to them and a name lookup isn't required.

10. Once you've finished experimenting, you can remove the sample security group policy, application, and security group that you created with the following commands.

```
kubectl delete namespace my-namespace
aws ec2 delete-security-group --group-id $my_pod_security_group_id
```

Multiple network interfaces for pods

Multus CNI is a container network interface (CNI) plugin for Amazon EKS that enables attaching multiple network interfaces to a pod. For more information, see the [Multus-CNI](#) documentation on GitHub.

In Amazon EKS, each pod has one network interface assigned by the Amazon VPC CNI plugin. With Multus, you can create a multi-homed pod that has multiple interfaces. This is accomplished by Multus acting as a "meta-plugin"; a CNI plugin that can call multiple other CNI plugins. AWS support for Multus comes configured with the Amazon VPC CNI plugin as the default delegate plugin.

Considerations

- Amazon EKS won't be building and publishing single root I/O virtualization (SR-IOV) and Data Plane Development Kit (DPDK) CNI plugins. However, you can achieve packet acceleration by connecting directly to Amazon EC2 Elastic Network Adapters (ENA) through Multus managed host-device and `ipvlan` plugins.
- Amazon EKS is supporting Multus, which provides a generic process that enables simple chaining of additional CNI plugins. Multus and the process of chaining is supported, but AWS won't provide support for all compatible CNI plugins that can be chained, or issues that may arise in those CNI plugins that are unrelated to the chaining configuration.
- Amazon EKS is providing support and life cycle management for the Multus plugin, but isn't responsible for any IP address or additional management associated with the additional network interfaces. The IP address and management of the default network interface utilizing the Amazon VPC CNI plugin remains unchanged.
- Only the Amazon VPC CNI plugin is officially supported as the default delegate plugin. You need to modify the published Multus installation manifest to reconfigure the default delegate plugin to an alternate CNI if you choose not to use the Amazon VPC CNI plugin for primary networking.
- Multus is only supported when using the Amazon VPC CNI as the primary CNI. We do not support the Amazon VPC CNI when used for higher order interfaces, secondary or otherwise.
- To prevent the Amazon VPC CNI plugin from trying to manage additional network interfaces assigned to pods, you can tag the network interfaces with `node.k8s.amazonaws.com/no_manage`.
- Multus is compatible with network policies, but the policy has to be enriched to include ports and IP addresses that may be part of additional network interfaces attached to pods.

For an implementation walk through, see the [Multus Setup Guide](#) on GitHub.

CNI metrics helper

The CNI metrics helper is a tool that you can use to scrape network interface and IP address information, aggregate metrics at the cluster level, and publish the metrics to Amazon CloudWatch. To learn more about the metrics helper, see [cni-metrics-helper](#) on GitHub.

When managing an Amazon EKS cluster, you may want to know how many IP addresses have been assigned and how many are available. The CNI metrics helper helps you to:

- Track these metrics over time
- Troubleshoot and diagnose issues related to IP assignment and reclamation
- Provide insights for capacity planning

When a node is provisioned, the CNI plugin automatically allocates a pool of secondary IP addresses from the node's subnet to the primary network interface (`eth0`). This pool of IP addresses is known as the *warm pool*, and its size is determined by the node's instance type. For example, a `c4.large` instance can support three network interfaces and nine IP addresses per interface. The number of IP addresses available for a given pod is one less than the maximum (of ten) because one of the IP addresses is reserved for the elastic network interface itself. For more information, see [IP Addresses Per Network Interface Per Instance Type](#) in the *Amazon EC2 User Guide for Linux Instances*.

As the pool of IP addresses is depleted, the plugin automatically attaches another elastic network interface to the instance and allocates another set of secondary IP addresses to that interface. This process continues until the node can no longer support additional elastic network interfaces.

The following metrics are collected for your cluster and exported to CloudWatch:

- The maximum number of network interfaces that the cluster can support
- The number of network interfaces have been allocated to pods
- The number of IP addresses currently assigned to pods
- The total and maximum numbers of IP addresses available
- The number of ipamD errors

Prerequisites

- An existing AWS Identity and Access Management (IAM) OpenID Connect (OIDC) provider for your cluster. To determine whether you already have one, or to create one, see [Create an IAM OIDC provider for your cluster \(p. 446\)](#).
- 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.
- The `kubectl` command line tool installed on your computer or AWS CloudShell. The version must be the same, or up to two versions later than your cluster version. To install or upgrade `kubectl`, see [Installing kubectl \(p. 4\)](#).
- If your cluster is 1.21 or later, make sure that your Amazon VPC CNI, `kube-proxy`, and `CoreDNS` add-ons are at the minimum versions listed in [Service account tokens \(p. 441\)](#).

Deploy the CNI metrics helper

Create an IAM policy and role and deploy the metrics helper.

To deploy the CNI metrics helper

1. Create an IAM policy that grants the CNI metrics helper `cloudwatch:PutMetricData` permissions to send metric data to CloudWatch.
 - a. Copy the following contents to a file named `cni-metrics-helper-policy.json`.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "cloudwatch:PutMetricData"
      ],
      "Resource": "*"
    }
  ]
}
```

- b. Create an IAM policy named `AmazonEKSVPCCNIMetricsHelperPolicy`.

```
aws iam create-policy --policy-name AmazonEKSVPCCNIMetricsHelperPolicy \
--description "Grants permission to write metrics to CloudWatch" \
--policy-document file://cni-metrics-helper-policy.json
```

2. Create an IAM role and attach the IAM policy to it. Create a Kubernetes service account. Annotate the Kubernetes service account with the IAM role ARN and the IAM role with the Kubernetes service account name. You can create the role using `eksctl` or the AWS CLI.

`eksctl`

Run the following command to create the IAM role. Replace `my-cluster` with your cluster name, `111122223333` with your account ID, and `region-code` with the AWS Region that your cluster is in.

```
eksctl create iamserviceaccount \
--name cni-metrics-helper \
--namespace kube-system \
--cluster my-cluster \
--role-name "AmazonEKSVPCCNIMetricsHelperRole" \
--attach-policy-arn
arn:aws:iam::111122223333:policy/AmazonEKSVPCCNIMetricsHelperPolicy \
--approve
```

AWS CLI

1. Determine your cluster's OIDC provider URL. Replace `my-cluster` with your cluster name. If the output from the command is None, review the [Prerequisites](#).

```
aws eks describe-cluster --name my-cluster --query "cluster.identity.oidc.issuer"
--output text
```

Example output:

```
oidc.eks.region-code.amazonaws.com/id/EXAMPLED539D4633E53DE1B71EXAMPLE
```

2. Create the IAM role, granting the Kubernetes service account the `AssumeRoleWithWebIdentity` action.
 - a. Copy the following contents to a file named `trust-policy.json`. Replace `111122223333` with your account ID. Replace `EXAMPLED539D4633E53DE1B71EXAMPLE` and `region-code` with the values returned in the previous step.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "Federated": "arn:aws:iam::111122223333:oidc-provider/oidc.eks.region-
code.amazonaws.com/id/EXAMPLED539D4633E53DE1B71EXAMPLE"
            },
            "Action": "sts:AssumeRoleWithWebIdentity",
            "Condition": {
                "StringEquals": {
                    "oidc.eks.region-code.amazonaws.com/
id/EXAMPLED539D4633E53DE1B71EXAMPLE:aud": "sts.amazonaws.com",
                    "oidc.eks.region-code.amazonaws.com/
id/EXAMPLED539D4633E53DE1B71EXAMPLE:sub": "system:serviceaccount:kube-
system:cni-metrics-helper"
                }
            }
        }
    ]
}
```

- b. Create the role.

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

3. Attach the IAM policy to the role. Replace **111122223333** with your account ID.

```
aws iam attach-role-policy \
--policy-arn
arn:aws:iam::111122223333:policy/AmazonEKSVPCCNIMetricsHelperPolicy \
--role-name AmazonEKSVPCCNIMetricsHelperRole
```

3. Use the following command for the AWS Region that your cluster is in to add the recommended version of the CNI metrics helper to your cluster.

Important

You should only update one minor version at a time. For example, if your current minor version is 1.9 and you want to update to 1.11, you should update to 1.10 first, then update to 1.11 by changing the version number in one of the following commands.

The recommended and latest version work with all Amazon EKS supported Kubernetes versions.

China (Beijing) (cn-north-1) or China (Ningxia) (cn-northwest-1)

```
kubectl apply -f https://raw.githubusercontent.com/aws/amazon-vpc-cni-k8s/v1.11.0/
config/master/cni-metrics-helper-cn.yaml
```

AWS GovCloud (US-East) (us-gov-east-1)

```
kubectl apply -f https://raw.githubusercontent.com/aws/amazon-vpc-cni-k8s/v1.11.0/
config/master/cni-metrics-helper-us-gov-east-1.yaml
```

AWS GovCloud (US-West) (us-gov-west-1)

```
kubectl apply -f https://raw.githubusercontent.com/aws/amazon-vpc-cni-k8s/v1.11.0/
config/master/cni-metrics-helper-us-gov-west-1.yaml
```

All other AWS Regions

1. Download the manifest file.

```
curl -o cni-metrics-helper.yaml https://raw.githubusercontent.com/aws/amazon-vpc-cni-
k8s/v1.11.0/config/master/cni-metrics-helper.yaml
```

2. If your cluster isn't in us-west-2, then replace **region-code** in the following command with the AWS Region that your cluster is in and then run the modified command to replace us-west-2 in the file with your AWS Region.

```
sed -i.bak -e 's/us-west-2/region-code/' cni-metrics-helper.yaml
```

3. If your cluster isn't in us-west-2, then replace **602401143452** in the following command with the account from [Amazon container image registries \(p. 386\)](#) for the AWS Region that your cluster is in and then run the modified command to replace 602401143452 in the file.

```
sed -i.bak -e 's/602401143452/602401143452/' cni-metrics-helper.yaml
```

4. Apply the manifest file to your cluster.

```
kubectl apply -f cni-metrics-helper.yaml
```

4. Annotate the `cni-metrics-helper` Kubernetes service account created in a previous step with the ARN of the IAM role that you created previously. Replace `111122223333` with your account ID, `my-cluster` with your cluster name, and `AmazonEKSVPCCNIMetricsHelperRole` with the name of the IAM role that you created in a previous step.

```
kubectl annotate serviceaccount cni-metrics-helper \
  -n kube-system \
  eks.amazonaws.com/role-
arn=arn:aws:iam::111122223333:role/AmazonEKSVPCCNIMetricsHelperRole
```

5. (Optional) Configure the AWS Security Token Service endpoint type used by your Kubernetes service account. For more information, see [Configure the AWS Security Token Service endpoint for a service account \(p. 452\)](#).
6. Restart the `cni-metrics-helper` deployment.

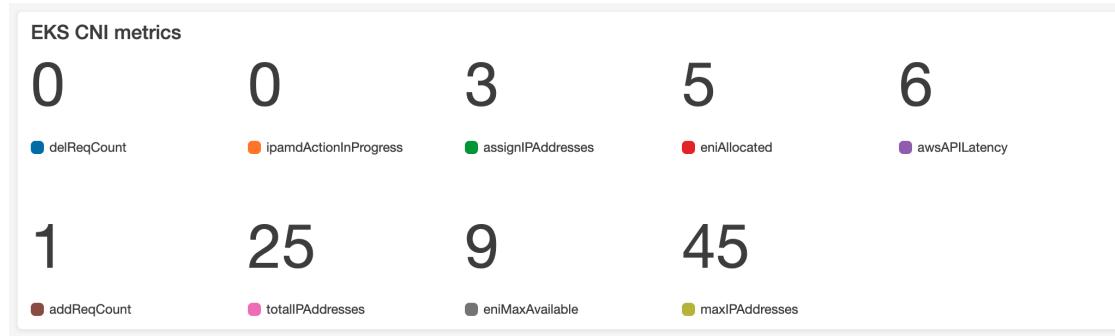
```
kubectl rollout restart \
  deployment cni-metrics-helper \
  -n kube-system
```

Creating a metrics dashboard

After you have deployed the CNI metrics helper, you can view the CNI metrics in the CloudWatch console. This topic helps you to create a dashboard for viewing your cluster's CNI metrics.

To create a CNI metrics dashboard

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. In the left navigation pane, choose **Metrics** and then select **All metrics**.
3. Under **Custom Namespaces**, select **Kubernetes**.
4. Select **CLUSTER_ID**.
5. On the **Metrics** tab, select the metrics you want to add to the dashboard.
6. At the upper right of the console, select **Actions**, and then **Add to dashboard**.
7. In the **Select a dashboard** section, select **Create new**, enter a name for your dashboard, such as **EKS-CNI-metrics**, and then select **Create**.
8. In the **Widget type** section, select **Number**.
9. In the **Customize widget title** section, enter a logical name for your dashboard title, such as **EKS CNI metrics**.
10. Select **Add to dashboard** to finish. Now your CNI metrics are added to a dashboard that you can monitor, as shown below.



Alternate compatible CNI plugins

Amazon EKS only officially supports the [Amazon VPC CNI plugin \(p. 276\)](#). Amazon EKS runs upstream Kubernetes and is certified Kubernetes conformant however, so alternate CNI plugins will work with Amazon EKS clusters. If you plan to use an alternate CNI plugin in production, then we strongly recommend that you either obtain commercial support, or have the in-house expertise to troubleshoot and contribute fixes to the open source CNI plugin project.

Amazon EKS maintains relationships with a network of partners that offer support for alternate compatible CNI plugins. See the following partners' documentation for details on supported Kubernetes versions and qualifications and testing performed.

Partner	Product	Documentation
Tigera	Calico	Installation instructions
Isovalent	Cilium	Installation instructions
Weaveworks	Weave Net	Installation instructions
VMware	Antrea	Installation instructions

Amazon EKS aims to give you a wide selection of options to cover all use cases. If you develop a commercially supported Kubernetes CNI plugin that is not listed here, then please contact our partner team at aws-container-partners@amazon.com for more information.

Installing the AWS Load Balancer Controller add-on

The AWS Load Balancer Controller manages AWS Elastic Load Balancers for a Kubernetes cluster. The controller provisions the following resources:

- An AWS Application Load Balancer (ALB) when you create a Kubernetes Ingress.
- An AWS Network Load Balancer (NLB) when you create a Kubernetes service of type LoadBalancer. In the past, the Kubernetes network load balancer was used for *instance* targets, but the AWS Load balancer Controller was used for *IP* targets. With the AWS Load Balancer Controller version 2.3.0 or later, you can create NLBs using either target type. For more information about NLB target types, see [Target type](#) in the User Guide for Network Load Balancers.

The AWS Load Balancer Controller controller was formerly named the *AWS ALB Ingress Controller*. It's an [open-source project](#) managed on GitHub. This topic describes how to install the controller using default options. You can view the full [documentation](#) for the controller on GitHub. Before deploying the controller, we recommend that you review the prerequisites and considerations in [Application load balancing on Amazon EKS \(p. 378\)](#) and [Network load balancing on Amazon EKS \(p. 372\)](#). Those topics also include steps on how to deploy a sample application that require the AWS Load Balancer Controller to provision AWS Application Load Balancers and Network Load Balancers.

Prerequisites

- An existing Amazon EKS cluster. To deploy one, see [Getting started with Amazon EKS \(p. 4\)](#). To use version 2.4.1 of the controller, which is the version used in this topic, your cluster must be 1.19 or later. If your cluster is earlier than 1.19, then we recommend using version 2.3.1.
- An existing AWS Identity and Access Management (IAM) OpenID Connect (OIDC) provider for your cluster. To determine whether you already have one, or to create one, see [Create an IAM OIDC provider for your cluster \(p. 446\)](#).
- If your cluster is 1.21 or later, make sure that your Amazon VPC CNI, kube-proxy, and CoreDNS add-ons are at the minimum versions listed in [Service account tokens \(p. 441\)](#).

To deploy the AWS Load Balancer Controller to an Amazon EKS cluster

In the following steps, replace the *example values* with your own values. If your cluster is earlier than 1.19, then change all instances of 2.4.1 to 2.3.1 and all instances of v2_4_1_full.yaml to v2_3_1_full.yaml.

1. Create an IAM policy.

- Download an IAM policy for the AWS Load Balancer Controller that allows it to make calls to AWS APIs on your behalf.
 - AWS GovCloud (US-East) or AWS GovCloud (US-East) AWS Regions

```
curl -o iam_policy_us-gov.json https://raw.githubusercontent.com/kubernetes-sigs/aws-load-balancer-controller/v2.4.1/docs/install/iam_policy_us-gov.json
```

- All other AWS Regions

```
curl -o iam_policy.json https://raw.githubusercontent.com/kubernetes-sigs/aws-load-balancer-controller/v2.4.1/docs/install/iam_policy.json
```

- Create an IAM policy using the policy downloaded in the previous step. If you downloaded iam_policy_us-gov.json, change iam_policy.json to iam_policy_us-gov.json before running the command.

```
aws iam create-policy \
--policy-name AWSLoadBalancerControllerIAMPolicy \
--policy-document file://iam_policy.json
```

Note

If you view the policy in the AWS Management Console, you may see warnings for **ELB**. These can be safely ignored because some of the actions only exist for ELB v2. You do not see warnings for ELB v2.

- Create an IAM role. Create a Kubernetes service account named aws-load-balancer-controller in the kube-system namespace for the AWS Load Balancer Controller and annotate the Kubernetes service account with the name of the IAM role.

You can use `eksctl` or the AWS CLI and `kubectl` to create the IAM role and Kubernetes service account.

`eksctl`

Replace `my-cluster` with the name of your cluster, `111122223333` with your account ID, and then run the command. If your cluster is in the AWS GovCloud (US-East) or AWS GovCloud (US-East) AWS Regions, then replace `arn:aws:` with `arn:aws-us-gov:` before running the following command.

```
eksctl create iamserviceaccount \
--cluster=my-cluster \
--namespace=kube-system \
--name=aws-load-balancer-controller \
--role-name "AmazonEKSLoadBalancerControllerRole" \
--attach-policy-arn=arn:aws:iam::111122223333:policy/AWSLoadBalancerControllerIAMPolicy \
--approve
```

AWS CLI and `kubectl`

Using the AWS CLI and `kubectl`

- View your cluster's OIDC provider URL.

```
aws eks describe-cluster --name my-cluster --query "cluster.identity.oidc.issuer" \
--output text
```

Example output:

```
oidc.eks.region-code.amazonaws.com/id/EXAMPLED539D4633E53DE1B71EXAMPLE
```

If no output is returned, then you must [create an IAM OIDC provider for your cluster \(p. 446\)](#).

- Copy the following contents to a file named `load-balancer-role-trust-policy.json`. Replace `111122223333` with your account ID. Replace `region-code` with the AWS Region that your cluster is in.. Replace `EXAMPLED539D4633E53DE1B71EXAMPLE` with the output returned in the previous step. If your cluster is in the AWS GovCloud (US-East) or AWS GovCloud (US-East) AWS Regions, then replace `arn:aws:` with `arn:aws-us-gov:`.

```
{ \
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "Federated": "arn:aws:iam::111122223333:oidc-provider/
oidc.eks.region-code.amazonaws.com/id/EXAMPLED539D4633E53DE1B71EXAMPLE"
            },
            "Action": "sts:AssumeRoleWithWebIdentity",
            "Condition": {
                "StringEquals": {
                    "oidc.eks.region-code.amazonaws.com/
id/EXAMPLED539D4633E53DE1B71EXAMPLE:aud": "sts.amazonaws.com",
                    "oidc.eks.region-code.amazonaws.com/
id/EXAMPLED539D4633E53DE1B71EXAMPLE:sub": "system:serviceaccount:kube-system:aws-
load-balancer-controller"
                }
            }
        }
    ]
}
```

```
        }
    ]
}
```

- c. Create the IAM role.

```
aws iam create-role \
--role-name AmazonEKSLoadBalancerControllerRole \
--assume-role-policy-document file:///"load-balancer-role-trust-policy.json"
```

- d. Attach the required Amazon EKS managed IAM policy to the IAM role. Replace **111122223333** with your account ID.

```
aws iam attach-role-policy \
--policy-arn
arn:aws:iam::111122223333:policy/AWSLoadBalancerControllerIAMPolicy \
--role-name AmazonEKSLoadBalancerControllerRole
```

- e. Save the following contents to a file that's named **aws-load-balancer-controller-service-account.yaml**, replacing **111122223333** with your account ID. If your cluster is in the AWS GovCloud (US-East) or AWS GovCloud (US-East) AWS Regions, then replace **arn:aws:** with **arn:aws-us-gov:**

```
apiVersion: v1
kind: ServiceAccount
metadata:
  labels:
    app.kubernetes.io/component: controller
    app.kubernetes.io/name: aws-load-balancer-controller
  name: aws-load-balancer-controller
  namespace: kube-system
  annotations:
    eks.amazonaws.com/role-arn:
      arn:aws:iam::111122223333:role/AmazonEKSLoadBalancerControllerRole
```

- f. Create the Kubernetes service account on your cluster. The Kubernetes service account named **aws-load-balancer-controller** is annotated with the IAM role that you created named **AmazonEKSLoadBalancerControllerRole**.

```
kubectl apply -f aws-load-balancer-controller-service-account.yaml
```

3. (Optional) Configure the AWS Security Token Service endpoint type used by your Kubernetes service account. For more information, see [Configure the AWS Security Token Service endpoint for a service account \(p. 452\)](#).
4. If you don't currently have the AWS ALB Ingress Controller for Kubernetes installed, or don't currently have the **0.1.x** version of the AWS Load Balancer Controller installed with Helm, then skip to the next step.

Uninstall the AWS ALB Ingress Controller or **0.1.x** version of the AWS Load Balancer Controller (only if installed with Helm). Complete the procedure using the tool that you originally installed it with. The AWS Load Balancer Controller replaces the functionality of the AWS ALB Ingress Controller for Kubernetes.

Helm

- a. If you installed the `incubator/aws-alb-ingress-controller` Helm chart, uninstall it.

```
helm delete aws-alb-ingress-controller -n kube-system
```

- b. If you have version `0.1.x` of the `eks-charts/aws-load-balancer-controller` chart installed, uninstall it. The upgrade from `0.1.x` to version `1.0.0` doesn't work due to incompatibility with the webhook API version.

```
helm delete aws-load-balancer-controller -n kube-system
```

Kubernetes manifest

- a. Check to see if the controller is currently installed.

```
kubectl get deployment -n kube-system alb-ingress-controller
```

This is the output if the controller isn't installed. Skip to the [install controller \(p. 334\)](#) step.

```
Error from server (NotFound): deployments.apps "alb-ingress-controller" not found
```

This is the output if the controller is installed.

NAME	READY	UP-TO-DATE	AVAILABLE	AGE
alb-ingress-controller	1/1	1	1	122d

- b. Enter the following commands to remove the controller.

```
kubectl delete -f https://raw.githubusercontent.com/kubernetes-sigs/aws-alb-ingress-controller/v1.1.8/docs/examples/alb-ingress-controller.yaml  
kubectl delete -f https://raw.githubusercontent.com/kubernetes-sigs/aws-alb-ingress-controller/v1.1.8/docs/examples/rbac-role.yaml
```

- c. Add the following IAM policy to the IAM role created in a [previous step \(p. 330\)](#). The policy allows the AWS Load Balancer Controller access to the resources that were created by the ALB Ingress Controller for Kubernetes.

1. Download the IAM policy. You can also [view the policy](#).

```
curl -o iam_policy_v1_to_v2_additional.json https://raw.githubusercontent.com/kubernetes-sigs/aws-load-balancer-controller/v2.4.1/docs/install/iam_policy_v1_to_v2_additional.json
```

2. If your cluster is in the AWS GovCloud (US-East) or AWS GovCloud (US-West) AWS Regions, then replace `arn:aws:` in the file with `arn:aws-us-gov:..`

```
sed -i.bak -e 's|arn:aws:|arn:aws-us-gov:|' iam_policy_v1_to_v2_additional.json
```

3. Create the IAM policy and note the ARN that is returned.

```
aws iam create-policy \  
--policy-name AWSLoadBalancerControllerAdditionalIAMPolicy \  
--policy-document file:///iam_policy_v1_to_v2_additional.json
```

4. Attach the IAM policy to the IAM role that you created in a [previous step \(p. 330\)](#). Replace `your-role-name` with the name of the role. If you created the role using `eksctl`, then to find the role name that was created, open the [AWS CloudFormation console](#) and select the `eksctl-your-cluster-name-addon-iamserviceaccount-kube-system-aws-load-balancer-controller` stack. Select the **Resources** tab. The role name is in the **Physical ID** column. If you used the AWS Management Console to create the role, then the role name is whatever you named it, such as `AmazonEKSLoadBalancerControllerRole`. If your

cluster is in the AWS GovCloud (US-East) or AWS GovCloud (US-East) AWS Regions, then replace `arn:aws:` with `arn:aws-us-gov:` before running the command.

```
aws iam attach-role-policy \
--role-name your-role-name \
--policy-arn
arn:aws:iam::111122223333:policy/AWSLoadBalancerControllerAdditionalIAMPolicy
```

5. Install the AWS Load Balancer Controller using [Helm V3 \(p. 430\)](#) or later or by applying a Kubernetes manifest. If you want to deploy the controller on Fargate, use the Helm procedure because it doesn't depend on `cert-manager`.

Helm

- a. Add the `eks-charts` repository.

```
helm repo add eks https://aws.github.io/eks-charts
```

- b. Update your local repo to make sure that you have the most recent charts.

```
helm repo update
```

- c. If your nodes don't have access to Amazon EKS Amazon ECR image repositories, then you need to pull the following container image and push it to a repository that your nodes have access to. For more information on how to pull, tag, and push an image to your own repository, see [Copy a container image from one repository to another repository \(p. 384\)](#). Replace `602401143452` and `region-code` with the values for your AWS Region listed in [Amazon container image registries \(p. 386\)](#).

```
602401143452.dkr.ecr.region-code.amazonaws.com/amazon/aws-load-balancer-
controller:v2.4.1
```

- d. Install the AWS Load Balancer Controller. If you're deploying the controller to Amazon EC2 nodes that have [restricted access to the Amazon EC2 instance metadata service \(IMDS\)](#), or if you're deploying to Fargate, then add the following flags to the `helm` command that follows:

- `--set region=region-code`
- `--set vpcId=vpc-xxxxxxxx`

If you're deploying to any AWS Region other than `us-west-2`, then add the following flag to the `helm` command, replacing `602401143452` and `region-code` with the values for your AWS Region listed in [Amazon container image registries \(p. 386\)](#). If you pulled the image and pushed it to your own repository, then replace the full registry and repository with your own.

```
--set image.repository=602401143452.dkr.ecr.region-code.amazonaws.com/amazon/aws-
load-balancer-controller
```

Replace `cluster-name` with your own. In the following command, `aws-load-balancer-controller` is the Kubernetes service account that you created in a previous step.

```
helm install aws-load-balancer-controller eks/aws-load-balancer-controller \
-n kube-system \
--set clusterName=cluster-name \
--set serviceAccount.create=false \
--set serviceAccount.name=aws-load-balancer-controller
```

Important

The deployed chart doesn't receive security updates automatically. You need to manually upgrade to a newer chart when it becomes available. When upgrading, change *install* to *upgrade* in the previous command, but run the following command to install the TargetGroupBinding custom resource definitions before running the previous command.

```
kubectl apply -k "github.com/aws/eks-charts/stable/aws-load-balancer-controller/crds?ref=master"
```

Kubernetes manifest

- a. Install cert-manager using one of the following methods to inject certificate configuration into the webhooks.
 - If your nodes have access to the quay.io container registry, install cert-manager to inject certificate configuration into the webhooks.

```
kubectl apply \  
  --validate=false \  
  -f https://github.com/jetstack/cert-manager/releases/download/v1.5.4/cert-  
  manager.yaml
```

- If your nodes don't have access to the quay.io container registry, then complete the following tasks:
 - i. Download the manifest.

```
curl -Lo cert-manager.yaml https://github.com/jetstack/cert-manager/releases/  
download/v1.5.4/cert-manager.yaml
```

- ii. Pull the following images and push them to a repository that your nodes have access to. For more information on how to pull, tag, and push the images to your own repository, see [Copy a container image from one repository to another repository \(p. 384\)](#).

```
quay.io/jetstack/cert-manager-cainjector:v1.5.4  
quay.io/jetstack/cert-manager-controller:v1.5.4  
quay.io/jetstack/cert-manager-webhook:v1.5.4
```

- iii. Replace quay.io in the manifest for the three images with your own registry name. The following command assumes that your private repository's name is the same as the source repository. Replace *111122223333.dkr.ecr.region-code.amazonaws.com* with your private registry.

```
sed -i.bak -e 's|quay.io|111122223333.dkr.ecr.region-code.amazonaws.com|' ./  
cert-manager.yaml
```

- iv. Apply the manifest.

```
kubectl apply \  
  --validate=false \  
  -f ./cert-manager.yaml
```

- b. Install the controller.
 - i. Download the controller specification. For more information about the controller, see the [documentation](#) on GitHub.

```
curl -Lo v2_4_1_full.yaml https://github.com/kubernetes-sigs/aws-load-balancer-controller/releases/download/v2.4.1/v2_4_1_full.yaml
```

ii. Make the following edits to the file.

- Replace `your-cluster-name` in the Deployment spec section of the file with the name of your cluster by replacing `my-cluster` with the name of your cluster.

```
sed -i.bak -e 's|your-cluster-name|my-cluster|' ./v2_4_1_full.yaml
```

- If your nodes don't have access to the Amazon EKS Amazon ECR image repositories, then you need to pull the following image and push it to a repository that your nodes have access to. For more information on how to pull, tag, and push an image to your own repository, see [Copy a container image from one repository to another repository \(p. 384\)](#).

```
amazon/aws-alb-ingress-controller:v2.4.1
```

Add your registry's name to the manifest. The following command assumes that your private repository's name is the same as the source repository and adds your private registry's name to the file. In the source file there is no registry specified because Kubernetes pulls from docker.io, by default. Replace `111122223333.dkr.ecr.region-code.amazonaws.com` with your registry. This line assumes that you named your private repository the same as the source repository. If not, change the `amazon/aws-alb-ingress-controller` text after your private registry name to your repository name.

```
sed -i.bak -e 's|amazon/aws-alb-ingress-controller|111122223333.dkr.ecr.region-code.amazonaws.com/amazon/aws-alb-ingress-controller|' ./v2_4_1_full.yaml
```

- Open the file in an editor and remove the following lines. Removing this section prevents the annotation with the IAM role that was added to the `aws-load-balancer-controller` Kubernetes service account that you created in a previous step from being overwritten when the controller is deployed. It also preserves the service account that you created in a previous step if you delete the controller.

```
apiVersion: v1
kind: ServiceAccount
metadata:
  labels:
    app.kubernetes.io/component: controller
    app.kubernetes.io/name: aws-load-balancer-controller
  name: aws-load-balancer-controller
  namespace: kube-system
---
```

- If you're deploying the controller to Amazon EC2 nodes that have [restricted access to the Amazon EC2 instance metadata service \(IMDS\)](#), or if you're deploying to Fargate, then add the **following parameters** under `- args:`

```
...
spec:
  containers:
    - args:
        - --cluster-name=your-cluster-name
        - --ingress-class=alb
        - --aws-vpc-id=vpc-xxxxxxxx
```

```
- --aws-region=region-code
```

...

iii. Apply the file.

```
kubectl apply -f v2_4_1_full.yaml
```

6. Verify that the controller is installed.

```
kubectl get deployment -n kube-system aws-load-balancer-controller
```

Example output:

NAME	READY	UP-TO-DATE	AVAILABLE	AGE
aws-load-balancer-controller	2/2	2	2	84s

You receive the previous output if you deployed using Helm. If you deployed using the Kubernetes manifest, you only have one replica.

7. Before using the controller to provision AWS resources, your cluster must meet specific requirements. For more information, see [Application load balancing on Amazon EKS \(p. 378\)](#) and [Network load balancing on Amazon EKS \(p. 372\)](#).

Managing the CoreDNS add-on

CoreDNS is a flexible, extensible DNS server that can serve as the Kubernetes cluster DNS. When you launch an Amazon EKS cluster with at least one node, two replicas of the CoreDNS image are deployed by default, regardless of the number of nodes deployed in your cluster. The CoreDNS pods provide name resolution for all pods in the cluster. The CoreDNS pods can be deployed to Fargate nodes if your cluster includes an [AWS Fargate profile \(p. 158\)](#) with a namespace that matches the namespace for the CoreDNS deployment. For more information about CoreDNS, see [Using CoreDNS for Service Discovery](#) in the Kubernetes documentation.

The following table lists the version of the CoreDNS add-on that is deployed with each Amazon EKS cluster version.

CoreDNS version deployed with each Amazon EKS supported cluster version

Kubernetes version	1.22	1.21	1.20	1.19	1.18	1.17
CoreDNS	1.8.7	1.8.4	1.8.3	1.8.0	1.7.0	1.6.6

If you have a 1.18 or later cluster that you have not added the CoreDNS Amazon EKS add-on to, you can add it using the procedure in [Adding the CoreDNS Amazon EKS add-on \(p. 338\)](#). If you created your 1.18 or later cluster using the AWS Management Console on or after May 19, 2021, the CoreDNS Amazon EKS add-on is already on your cluster. If you created your 1.18 or later cluster using any other tool, and want to use the CoreDNS Amazon EKS add-on, then you must add it to your cluster yourself.

If you've added the CoreDNS Amazon EKS add-on to your 1.18 or later cluster, you can manage it using the procedures in the [Updating the CoreDNS Amazon EKS add-on \(p. 339\)](#) and [Removing the CoreDNS Amazon EKS add-on \(p. 341\)](#) sections. For more information about Amazon EKS add-ons, see [Amazon EKS add-ons \(p. 388\)](#).

If you have not added the CoreDNS Amazon EKS add-on, the CoreDNS self-managed add-on is still running on your cluster. You can update the CoreDNS self-managed add-on using the procedure in [Updating the CoreDNS self-managed add-on \(p. 342\)](#).

Prerequisites

- An existing Amazon EKS cluster. To deploy one, see [Getting started with Amazon EKS \(p. 4\)](#).
- If your cluster is 1.21 or later, make sure that your Amazon VPC and `kube-proxy` add-ons are at the minimum versions listed in [Service account tokens \(p. 441\)](#).

Adding the CoreDNS Amazon EKS add-on

Select the tab with the name of the tool that you want to use to add the CoreDNS Amazon EKS add-on to your cluster with.

Important

Before adding the CoreDNS Amazon EKS add-on, confirm that you do not self-manage any settings that Amazon EKS will start managing. To determine which settings Amazon EKS manages, see [Amazon EKS add-on configuration \(p. 389\)](#).

`eksctl`

To add the CoreDNS Amazon EKS add-on using `eksctl`

Replace `my-cluster` with the name of your cluster and then run the following command.

```
eksctl create addon --name coredns --cluster my-cluster --force
```

If you remove the `--force` option and any of the Amazon EKS add-on settings conflict with your existing settings, then adding the Amazon EKS add-on fails, and you receive an error message to help you resolve the conflict. Before specifying this option, make sure that the Amazon EKS add-on doesn't manage settings that you need to manage, because those settings are overwritten with this option. For more information about Amazon EKS add-on configuration management, see [Amazon EKS add-on configuration \(p. 389\)](#).

AWS Management Console

To add the CoreDNS Amazon EKS add-on using the AWS Management Console

1. Open the Amazon EKS console at <https://console.aws.amazon.com/eks/home#/clusters>.
2. In the left navigation pane, choose Amazon EKS **Clusters**, and then select the name of the cluster that you want to configure the CoreDNS Amazon EKS add-on for.
3. Choose the **Configuration** tab and then choose the **Add-ons** tab.
4. Select **Add new**.
 - a. Select **CoreDNS** for **Name**.
 - b. Select the **Version** you'd like to use.
 - c. If you select **Override existing configuration for this add-on on the cluster**, then any setting for the existing add-on can be overwritten with the Amazon EKS add-on's settings. If you don't enable this option and any of the Amazon EKS add-on settings conflict with your existing settings, then adding the Amazon EKS add-on fails, and you receive an error message to help you resolve the conflict. Before selecting this option, make sure that the Amazon EKS add-on doesn't manage settings that you need to manage. For more information about Amazon EKS add-on configuration management, see [Amazon EKS add-on configuration \(p. 389\)](#).

- d. Select **Add**.

AWS CLI

To add the CoreDNS Amazon EKS add-on using the AWS CLI

Replace *my-cluster* with the name of your cluster and then run the following command.

```
aws eks create-addon \
--cluster-name my-cluster \
--addon-name coredns \
--resolve-conflicts OVERWRITE
```

If you remove the `--resolve-conflicts OVERWRITE` option and any of the Amazon EKS add-on settings conflict with your existing settings, then creating the add-on fails, and you receive an error message to help you resolve the conflict. Before specifying this option, make sure that the Amazon EKS add-on doesn't manage settings that you need to manage, because those settings are overwritten with this option. For more information about Amazon EKS add-on configuration management, see [Amazon EKS add-on configuration \(p. 389\)](#).

Updating the CoreDNS Amazon EKS add-on

This procedure is for updating the CoreDNS Amazon EKS add-on. If you haven't added the CoreDNS Amazon EKS add-on, complete the procedure in [Updating the CoreDNS self-managed add-on \(p. 342\)](#) instead, or [add the CoreDNS Amazon EKS add-on \(p. 338\)](#). Amazon EKS does not automatically update CoreDNS on your cluster when new versions are released or after you [update your cluster \(p. 34\)](#) to a new Kubernetes minor version. To update CoreDNS on an existing cluster, you must initiate the update and then Amazon EKS updates the Amazon EKS add-on for you.

eksctl

To update the CoreDNS Amazon EKS add-on using eksctl

1. Check the current version of your coredns Amazon EKS add-on. Replace *my-cluster* with your cluster name.

```
eksctl get addon --name coredns --cluster my-cluster
```

Example output:

NAME	VERSION	STATUS	ISSUES	IAMROLE	UPDATE	AVAILABLE
coredns	v1.8.0-eksbuild.1	ACTIVE	0			v1.8.3-eksbuild.1

2. Update the add-on to the version returned under UPDATE AVAILABLE in the output of the previous step.

```
eksctl update addon \
--name coredns \
--version v1.8.3-eksbuild.1 \
--cluster my-cluster \
--force
```

If you remove the `--force` option and any of the Amazon EKS add-on settings conflict with your existing settings, then updating the Amazon EKS add-on fails, and you receive an error message to help you resolve the conflict. Before specifying this option, make sure that the

Amazon EKS add-on doesn't manage settings that you need to manage, because those settings are overwritten with this option. For more information about Amazon EKS add-on configuration management, see [Amazon EKS add-on configuration \(p. 389\)](#).

AWS Management Console

To update the CoreDNS Amazon EKS add-on using the AWS Management Console

1. Open the Amazon EKS console at <https://console.aws.amazon.com/eks/home#/clusters>.
2. In the left navigation pane, choose Amazon EKS **Clusters**, and then select the name of the cluster that you want to update the CoreDNS Amazon EKS add-on for.
3. Choose the **Configuration** tab and then choose the **Add-ons** tab.
4. Select the radio button in the upper right of the **coredns** box and then choose **Edit**.
 - a. Select the **Version** of the Amazon EKS add-on that you want to use.
 - b. If you select **Override existing configuration for this add-on on the cluster**, then any setting for the existing add-on can be overwritten with the Amazon EKS add-on's settings. If you don't enable this option and any of the Amazon EKS add-on settings conflict with your existing settings, then updating the add-on to an Amazon EKS add-on fail, and you receive an error message to help you resolve the conflict. Before selecting this option, make sure that the Amazon EKS add-on doesn't manage settings that you need to manage. For more information about Amazon EKS add-on configuration management, see [Amazon EKS add-on configuration \(p. 389\)](#).
 - c. Select **Update**.

AWS CLI

To update the CoreDNS Amazon EKS add-on using the AWS CLI

1. Check the current version of your CoreDNS add-on. Replace **my-cluster** with your cluster name.

```
aws eks describe-addon \
--cluster-name my-cluster \
--addon-name coredns \
--query "addon.addonVersion" \
--output text
```

Output:

```
1.7.0
```

2. Determine which versions of the CoreDNS add-on are available for your cluster's version.

```
aws eks describe-addon-versions \
--addon-name coredns \
--kubernetes-version 1.19 \
--query "addons[].addonVersions[].addonVersion,
compatibilities[].defaultVersion" \
--output text
```

Example output:

```
1.8.0
True
```

1.7.0
False

The version with `True` underneath is the default version deployed with new clusters.

3. Update the add-on to the version with `True` returned in the output of the previous step. You can also update to a later version, if returned in the output.

```
aws eks update-addon \  
  --cluster-name my-cluster \  
  --addon-name coredns \  
  --addon-version 1.8.0 \  
  --resolve-conflicts OVERWRITE
```

If you remove the `--resolve-conflicts OVERWRITE` option and any of the Amazon EKS add-on settings conflict with your existing settings, then updating the add-on fails, and you receive an error message to help you resolve the conflict. Before specifying this option, make sure that the Amazon EKS add-on doesn't manage settings that you need to manage, because those settings are overwritten with this option. For more information about Amazon EKS add-on configuration management, see [Amazon EKS add-on configuration \(p. 389\)](#).

Removing the CoreDNS Amazon EKS add-on

You have two options when removing an Amazon EKS add-on:

- **Preserve the add-on's software on your cluster** – This option removes Amazon EKS management of any settings and the ability for Amazon EKS to notify you of updates and automatically update the Amazon EKS add-on after you initiate an update, but preserves the add-on's software on your cluster. This option makes the add-on a self-managed add-on, rather than an Amazon EKS add-on. There is no downtime for the add-on.
- **Removing the add-on software entirely from your cluster** – You should only remove the Amazon EKS add-on from your cluster if there are no resources on your cluster are dependent on the functionality that the add-on provides. After removing the Amazon EKS add-on, you can add it again if you want to.

If the add-on has an IAM account associated with it, the IAM account is not removed.

Select the tab with the name of the tool that you want to use to remove the CoreDNS Amazon EKS add-on from your 1.18 or later cluster with.

eksctl

To remove the CoreDNS Amazon EKS add-on using eksctl

Replace `my-cluster` with the name of your cluster and then run the following command. Removing `--preserve` removes the add-on software from your cluster.

```
eksctl delete addon --cluster my-cluster --name coredns --preserve
```

AWS Management Console

To remove the CoreDNS Amazon EKS add-on using the AWS Management Console

1. Open the Amazon EKS console at <https://console.aws.amazon.com/eks/home#/clusters>.
2. In the left navigation pane, choose Amazon EKS **Clusters**, and then select the name of the cluster that you want to remove the CoreDNS Amazon EKS add-on for.

3. Choose the **Configuration** tab and then choose the **Add-ons** tab.
4. Select the radio button in the upper right of the **coredns** box and then choose **Remove**. Type **coredns** and then select **Remove**.

AWS CLI

To remove the CoreDNS Amazon EKS add-on using the AWS CLI

Replace *my-cluster* with the name of your cluster and then run the following command. Removing `--preserve` removes the add-on software from your cluster.

```
aws eks delete-addon --cluster-name my-cluster --addon-name coredns --preserve
```

Updating the CoreDNS self-managed add-on

If you have a 1.17 or earlier cluster, or a 1.18 or later cluster that you have not added the CoreDNS Amazon EKS add-on to, complete the following steps to update the add-on. If you've added the CoreDNS Amazon EKS add-on, complete the procedure in [Updating the CoreDNS Amazon EKS add-on \(p. 339\)](#) instead.

To update the CoreDNS self-managed add-on using kubectl

1. Check the current version of your CoreDNS add-on.

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

Output:

```
coredns:v1.8.0
```

2. If your current CoreDNS version is 1.5.0 or later, but earlier than the version listed in the [CoreDNS versions \(p. 337\)](#) table, then skip this step. If your current version is earlier than 1.5.0, then you need to modify the ConfigMap for CoreDNS to use the forward add-on, rather than the proxy add-on.

- a. Open the configmap with the following command.

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

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

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

3. If you originally deployed your cluster on Kubernetes 1.17 or earlier, then you may need to remove a discontinued term from your CoreDNS manifest.

Important

You must complete this before upgrading to CoreDNS version 1.7.0, but it's recommended that you complete this step even if you're upgrading to an earlier version.

- a. Check to see if your CoreDNS manifest has the line.

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

If no output is returned, your manifest doesn't have the line and you can skip to the next step to update CoreDNS. If output is returned, then you need to remove the line.

- b. Edit the ConfigMap with the following command, removing the line in the file that has the word upstream in it. Do not change anything else in the file. Once the line is removed, save the changes.

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

4. Retrieve your current CoreDNS image:

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

5. If you're updating to CoreDNS 1.8.3 or later, then you need to add the endpointslices permission to the system:coredns Kubernetes clusterrole.

```
kubectl edit clusterrole system:coredns -n kube-system
```

Add the following lines under the existing permissions lines in the rules section of the file.

```
...
- apiGroups:
  - discovery.k8s.io
resources:
- endpointslices
verbs:
- list
- watch
...
...
```

6. Update the CoreDNS add-on by replacing `602401143452` and `region-code` with the values from the output returned in a previous step. Replace `1.8.7` with your cluster's [recommended CoreDNS version \(p. 337\)](#) or later:

```
kubectl set image --namespace kube-system deployment.apps/coredns \
coredns=602401143452.dkr.ecr.region-code.amazonaws.com/eks/coredns:v1.8.7-eksbuild.1
```

Managing the kube-proxy add-on

Kube-proxy maintains network rules on each Amazon EC2 node. It enables network communication to your pods. Kube-proxy is not deployed to Fargate nodes. For more information, see [kube-proxy](#) in the Kubernetes documentation. Two versions of the kube-proxy image are available for each Kubernetes version.

- **Default** – The version deployed by default with new clusters. This is the only version that you can use with the Amazon EKS add-on.
- **Minimal** – Unlike the default version, this image version is based on a [minimal base image](#) maintained by Amazon EKS Distro, which contains minimal packages and doesn't have shells. For more information, see [Amazon EKS Distro](#). This version is available as a self-managed add-on, but not as an

Amazon EKS add-on. If you choose to install this version, complete the steps in Step 2 of [the section called “Updating the self-managed add-on” \(p. 348\)](#), specifying this version.

kube-proxy image version for each Amazon EKS supported cluster version

Kubernetes version	1.22	1.21	1.20	1.19	1.18	1.17
kube-proxy (default version)	1.22.6-eksbuild.1	1.21.2-eksbuild.2	1.20.4-eksbuild.2	1.19.6-eksbuild.2	1.18.8-eksbuild.1	1.17.9-eksbuild.1
kube-proxy (minimal)	1.22.6-minimal-eksbuild.2	1.21.9-minimal-eksbuild.2	1.20.15-minimal-eksbuild.2	1.19.16-minimal-eksbuild.2	1.18.20-minimal-eksbuild.1	1.17.17-minimal-eksbuild.1

If you have a 1.18 or later cluster that you have not added the kube-proxy Amazon EKS add-on to, you can add it using the procedure in [Adding the kube-proxy Amazon EKS add-on \(p. 344\)](#). If you created your 1.18 or later cluster using the AWS Management Console after May 3, 2021, the kube-proxy Amazon EKS add-on is already on your cluster. If you created your 1.18 or later cluster using any other tool, and want to use the kube-proxy Amazon EKS add-on, then you must add it to your cluster yourself.

If you've added the kube-proxy Amazon EKS add-on to your 1.18 or later cluster, you can manage it using the procedures in the [Updating the kube-proxy Amazon EKS add-on \(p. 345\)](#) and [Removing the kube-proxy Amazon EKS add-on \(p. 347\)](#) sections. For more information about Amazon EKS add-ons, see [Amazon EKS add-ons \(p. 388\)](#).

If you have not added the kube-proxy Amazon EKS add-on, the kube-proxy self-managed add-on is still running on your cluster. You can manually update the kube-proxy self-managed add-on using the procedure in [Updating the kube-proxy self-managed add-on \(p. 348\)](#).

Prerequisites

- An existing Amazon EKS cluster. To deploy one, see [Getting started with Amazon EKS \(p. 4\)](#).
- If your cluster is 1.21 or later, make sure that your Amazon VPC and CoreDNS add-ons are at the minimum versions listed in [Service account tokens \(p. 441\)](#).

Adding the kube-proxy Amazon EKS add-on

Select the tab with the name of the tool that you want to use to add the kube-proxy Amazon EKS add-on to your cluster with.

Important

Before adding the kube-proxy Amazon EKS add-on, confirm that you do not self-manage any settings that Amazon EKS will start managing. To determine which settings Amazon EKS manages, see [Amazon EKS add-on configuration \(p. 389\)](#).

eksctl

To add the kube-proxy Amazon EKS add-on using eksctl

Replace `my-cluster` with the name of your cluster and then run the following command.

```
eksctl create addon --name kube-proxy --cluster my-cluster --force
```

If you remove the `--force` option and any of the Amazon EKS add-on settings conflict with your existing settings, then adding the Amazon EKS add-on fails, and you receive an error message to

help you resolve the conflict. Before specifying this option, make sure that the Amazon EKS add-on doesn't manage settings that you need to manage, because those settings are overwritten with this option. For more information about Amazon EKS add-on configuration management, see [Amazon EKS add-on configuration \(p. 389\)](#).

AWS Management Console

To add the `kube-proxy` Amazon EKS add-on using the AWS Management Console

1. Open the Amazon EKS console at <https://console.aws.amazon.com/eks/home#/clusters>.
2. In the left navigation pane, select Amazon EKS **Clusters**, and then select the name of the cluster that you want to configure the `kube-proxy` Amazon EKS add-on for.
3. Choose the **Configuration** tab and then choose the **Add-ons** tab.
4. Select **Add new**.
 - a. Select **`kube-proxy` for Name**.
 - b. Select the **Version** you'd like to use.
 - c. If you select **Override existing configuration for this add-on on the cluster**, then any setting for the existing add-on can be overwritten with the Amazon EKS add-on's settings. If you don't enable this option and any of the Amazon EKS add-on settings conflict with your existing settings, then adding the Amazon EKS add-on fails, and you receive an error message to help you resolve the conflict. Before selecting this option, make sure that the Amazon EKS add-on doesn't manage settings that you need to manage. For more information about Amazon EKS add-on configuration management, see [Amazon EKS add-on configuration \(p. 389\)](#).
 - d. Select **Add**.

AWS CLI

To add the `kube-proxy` Amazon EKS add-on using the AWS CLI

Replace `my-cluster` with the name of your cluster and then run the following command.

```
aws eks create-addon \
  --cluster-name my-cluster \
  --addon-name kube-proxy \
  --resolve-conflicts OVERWRITE
```

If you remove the `--resolve-conflicts OVERWRITE` option and any of the Amazon EKS add-on settings conflict with your existing settings, then creating the add-on fails, and you receive an error message to help you resolve the conflict. Before specifying this option, make sure that the Amazon EKS add-on doesn't manage settings that you need to manage, because those settings are overwritten with this option. For more information about Amazon EKS add-on configuration management, see [Amazon EKS add-on configuration \(p. 389\)](#).

Updating the `kube-proxy` Amazon EKS add-on

This procedure is for updating the `kube-proxy` Amazon EKS add-on. If you haven't added the `kube-proxy` Amazon EKS add-on, complete the procedure in [Updating the `kube-proxy` self-managed add-on \(p. 348\)](#) instead. Amazon EKS does not automatically update `kube-proxy` on your cluster when new versions are released or after you [update your cluster \(p. 34\)](#) to a new Kubernetes minor version. To update `kube-proxy` on an existing cluster, you must initiate the update and then Amazon EKS updates the add-on for you.

Important

Update your cluster and nodes to a new Kubernetes minor version before updating `kube-proxy` to the same minor version as your updated cluster's minor version.

eksctl

To update the `kube-proxy` Amazon EKS add-on using `eksctl`

1. Check the current version of your `kube-proxy` Amazon EKS add-on. Replace `my-cluster` with your cluster name.

```
eksctl get addon --name kube-proxy --cluster my-cluster
```

Output

NAME	VERSION	STATUS	ISSUES	IAMROLE	UPDATE	AVAILABLE
kube-proxy	v1.19.6-eksbuild.2	ACTIVE	0			v1.20.4-eksbuild.2

2. Update the add-on to the version returned under UPDATE AVAILABLE in the output of the previous step.

```
eksctl update addon \
--name kube-proxy \
--version v1.20.4-eksbuild.2 \
--cluster my-cluster \
--force
```

If you remove the `--force` option and any of the Amazon EKS add-on settings conflict with your existing settings, then updating the add-on fails, and you receive an error message to help you resolve the conflict. Before specifying this option, make sure that the Amazon EKS add-on doesn't manage settings that you need to manage, because those settings are overwritten with this option. For more information about Amazon EKS add-on configuration management, see [Amazon EKS add-on configuration \(p. 389\)](#).

AWS Management Console

To update the `kube-proxy` Amazon EKS add-on using the AWS Management Console

1. Open the Amazon EKS console at <https://console.aws.amazon.com/eks/home#/clusters>.
2. In the left navigation pane, select Amazon EKS **Clusters**, and then select the name of the cluster that you want to update the `kube-proxy` Amazon EKS add-on for.
3. Choose the **Configuration** tab and then choose the **Add-ons** tab.
4. Select the box in the top right of the **kube-proxy** box and then choose **Edit**.
 - a. Select the **Version** of the Amazon EKS add-on that you want to use.
 - b. If you select **Override existing configuration for this add-on on the cluster**, then any setting for the existing add-on can be overwritten with the Amazon EKS add-on's settings. If you don't enable this option and any of the Amazon EKS add-on settings conflict with your existing settings, then updating the add-on to an Amazon EKS add-on fail, and you receive an error message to help you resolve the conflict. Before selecting this option, make sure that the Amazon EKS add-on doesn't manage settings that you need to manage. For more information about Amazon EKS add-on configuration management, see [Amazon EKS add-on configuration \(p. 389\)](#).
 - c. Select **Update**.

AWS CLI

To update the `kube-proxy` Amazon EKS add-on using the AWS CLI

1. Check the current version of your `kube-proxy` Amazon EKS add-on. Replace `my-cluster` with your cluster name.

```
aws eks describe-addon \
--cluster-name my-cluster \
--addon-name kube-proxy \
--query "addon.addonVersion" \
--output text
```

Output:

```
v1.19.6-eksbuild.2
```

2. Determine which versions of the `kube-proxy` add-on are available for your cluster version.

```
aws eks describe-addon-versions \
--addon-name kube-proxy \
--kubernetes-version 1.20 \
--query "addons[].addonVersions[].addonVersion,
compatibilities[].defaultVersion" \
--output text
```

Output

```
v1.20.4-eksbuild.2
True
v1.19.6-eksbuild.2
False
v1.18.8-eksbuild.1
False
```

The version with `True` underneath is the default version deployed with new clusters with the version that you specified.

3. Update the add-on to the version with `True` returned in the output of the previous step. You can also update to a later version, if returned in the output.

```
aws eks update-addon \
--cluster-name my-cluster \
--addon-name kube-proxy \
--addon-version v1.20.4-eksbuild.2 \
--resolve-conflicts OVERWRITE
```

If you remove the `--resolve-conflicts OVERWRITE` option and any of the Amazon EKS add-on settings conflict with your existing settings, then updating the add-on fails, and you receive an error message to help you resolve the conflict. Before specifying this option, make sure that the Amazon EKS add-on doesn't manage settings that you need to manage, because those settings are overwritten with this option. For more information about Amazon EKS add-on configuration management, see [Amazon EKS add-on configuration \(p. 389\)](#).

Removing the `kube-proxy` Amazon EKS add-on

You have two options when removing an Amazon EKS add-on:

- **Preserve the add-on's software on your cluster** – This option removes Amazon EKS management of any settings and the ability for Amazon EKS to notify you of updates and automatically update the Amazon EKS add-on after you initiate an update, but preserves the add-on's software on your cluster. This option makes the add-on a self-managed add-on, rather than an Amazon EKS add-on. There is no downtime for the add-on.
- **Removing the add-on software entirely from your cluster** – You should only remove the Amazon EKS add-on from your cluster if there are no resources on your cluster are dependent on the functionality that the add-on provides. After removing the Amazon EKS add-on, you can add it again if you want to.

If the add-on has an IAM account associated with it, the IAM account is not removed.

Select the tab with the name of the tool that you want to use to remove the `kube-proxy` Amazon EKS add-on from your 1.18 or later cluster with.

`eksctl`

To remove the `kube-proxy` Amazon EKS add-on using `eksctl`

Replace `my-cluster` with the name of your cluster and then run the following command. Removing `--preserve` removes the add-on software from your cluster.

```
eksctl delete addon --cluster my-cluster --name kube-proxy --preserve
```

AWS Management Console

To remove the `kube-proxy` Amazon EKS add-on using the AWS Management Console

1. Open the Amazon EKS console at <https://console.aws.amazon.com/eks/home#/clusters>.
2. In the left navigation pane, select Amazon EKS **Clusters**, and then select the name of the cluster that you want to remove the `kube-proxy` Amazon EKS add-on for.
3. Choose the **Configuration** tab and then choose the **Add-ons** tab.
4. Select the check box in the top right of the `kube-proxy` box and then choose **Remove**. Type `kube-proxy` and then select **Remove**.

AWS CLI

To remove the `kube-proxy` Amazon EKS add-on using the AWS CLI

Replace `my-cluster` with the name of your cluster and then run the following command. Removing `--preserve` removes the add-on software from your cluster.

```
aws eks delete-addon --cluster-name my-cluster --addon-name kube-proxy --preserve
```

Updating the `kube-proxy` self-managed add-on

If you have a 1.17 or earlier cluster, or a 1.18 or later cluster that you have not added the `kube-proxy` Amazon EKS add-on to, complete the following steps to update the self-managed add-on. If you've added the `kube-proxy` Amazon EKS add-on, complete the procedure in [Updating the `kube-proxy` Amazon EKS add-on \(p. 345\)](#) instead.

Important

Update your cluster and nodes to a new Kubernetes minor version before updating `kube-proxy` to the same minor version as your updated cluster's minor version.

To update the kube-proxy self-managed add-on using kubectl

1. Check the current version of your kube-proxy deployment.

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

Example output:

```
602401143452.dkr.ecr.region-code.amazonaws.com/eks/kube-proxy:v1.21.2-eksbuild.2
```

2. Update the kube-proxy add-on by replacing `602401143452` and `region-code` with the values from your output. Replace `1.22.6-eksbuild.1` with the kube-proxy version listed in the [kube-proxy version deployed with each Amazon EKS supported cluster version \(p. 344\)](#) table for your cluster version.

```
kubectl set image daemonset.apps/kube-proxy \
-n kube-system \
kube-proxy=602401143452.dkr.ecr.region-code.amazonaws.com/eks/kube-proxy:v1.22.6-eksbuild.1
```

3. (Optional) If you're using x86 and Arm nodes in the same cluster and your cluster was deployed before August 17, 2020. Then, edit your kube-proxy manifest to include a node selector for multiple hardware architectures with the following command. This is a one-time operation. After you've added the selector to your manifest, you don't need to add it each time you update. If your cluster was deployed on or after August 17, 2020, then kube-proxy is already multi-architecture capable.

```
kubectl edit -n kube-system daemonset/kube-proxy
```

Add the following node selector to the file in the editor and then save the file. For an example of where to include this text in the editor, see the [CNI manifest](#) file on GitHub. This enables Kubernetes to pull the correct hardware image based on the node's hardware architecture.

```
- key: "kubernetes.io/arch"
operator: In
values:
- amd64
- arm64
```

4. (Optional) If your cluster was originally created with Kubernetes v1.14 or later, then you can skip this step because kube-proxy already includes this `Affinity Rule`. If you originally created an Amazon EKS cluster with Kubernetes version 1.13 or earlier and intend to use Fargate nodes, then edit your kube-proxy manifest to include a `NodeAffinity` rule to prevent kube-proxy pods from scheduling on Fargate nodes. This is a one-time edit. Once you've added the `Affinity Rule` to your manifest, you don't need to add it each time you upgrade your cluster. Edit your kube-proxy DaemonSet.

```
kubectl edit -n kube-system daemonset/kube-proxy
```

Add the following `Affinity Rule` to the `DaemonSet spec` section of the file in the editor and then save the file. For an example of where to include this text in the editor, see the [CNI manifest](#) file on GitHub.

```
- key: eks.amazonaws.com/compute-type
```

```
operator: NotIn
values:
- fargate
```

Installing the Calico add-on

[Project Calico](#) is a network policy engine for Kubernetes. With Calico network policy enforcement, you can implement network segmentation and tenant isolation. This is useful in multi-tenant environments where you must isolate tenants from each other or when you want to create separate environments for development, staging, and production. Network policies are similar to AWS security groups in that you can create network ingress and egress rules. Instead of assigning instances to a security group, you assign network policies to pods using pod selectors and labels.

Considerations

- Calico is not supported when using Fargate with Amazon EKS.
- Calico adds rules to `iptables` on the node that may be higher priority than existing rules that you've already implemented outside of Calico. Consider adding existing `iptables` rules to your Calico policies to avoid having rules outside of Calico policy overridden by Calico.
- If you're using the Amazon VPC CNI add-on version 1.10 or earlier, [security groups for pods \(p. 313\)](#) traffic flow to pods on branch network interfaces is not subjected to Calico network policy enforcement and is limited to Amazon EC2 security group enforcement only. If you're using 1.11.0 or later of the Amazon VPC CNI add-on, traffic flow to pods on branch network interfaces is subject to Calico network policy enforcement if you set `POD_SECURITY_GROUP_ENFORCING_MODE=standard` for the Amazon VPC CNI add-on.

Prerequisites

- An existing Amazon EKS cluster. To deploy one, see [Getting started with Amazon EKS \(p. 4\)](#).
- The `kubectl` command line tool installed on your computer or AWS CloudShell. The version must be the same, or up to two versions later than your cluster version. To install or upgrade `kubectl`, see [Installing kubectl \(p. 4\)](#).

The following procedure shows you how to install Calico on Linux nodes in your Amazon EKS cluster. To install Calico on Windows nodes, see [Using Calico on Amazon EKS Windows Containers](#).

Install Calico on your Amazon EKS Linux nodes

Important

Amazon EKS doesn't maintain the manifests or charts used in the following procedures. The recommended way to install Calico on Amazon EKS is by using the [Calico Operator](#) instead of these charts or manifests. For more information, see [Important Announcement: Amazon EKS will no longer maintain and update Calico charts in this repository](#) on GitHub. If you encounter issues during installation and usage of Calico, submit issues to [Calico Operator](#) and the [Calico](#) project directly. You should always contact Tigera for compatibility of any new Calico operator and Calico versions before installing them on your cluster.

You can install Calico using the procedure for Helm or manifests.

Helm

Prerequisite

Helm version 3.0 or later installed on your computer. To install or upgrade Helm, see [Using Helm with Amazon EKS \(p. 430\)](#).

To install Calico with Helm

1. Add Project Calico into your Helm repository.

```
helm repo add projectcalico https://docs.projectcalico.org/charts
```

2. If you already have Calico added, you may want to update it to get the latest released version.

```
helm repo update
```

3. Install version 3.21.4 or later of the Tigera Calico operator and custom resource definitions.

```
helm install calico projectcalico/tigera-operator --version v3.21.4
```

4. View the resources in the `tigera-operator` namespace.

```
kubectl get all -n tigera-operator
```

Example output:

The values in the DESIRED and READY columns for the `replicaset` should match. The values returned for you are different than the *values* in the following output.

NAME	READY	STATUS	RESTARTS	AGE
pod/tigera-operator- <i>c4b9549c7-h4zp5</i>	1/1	Running	0	110m
...				
NAME	DESIRED	CURRENT	READY	AGE
replicaset.apps/tigera-operator- <i>c4b9549c7</i>	1	1	1	2m35s

5. View the resources in the `calico-system` namespace.

```
kubectl get all -n calico-system
```

Example output:

The values in the DESIRED and READY columns for the `calico-node` DaemonSet should match. The values in the DESIRED and READY columns for the two `replicasets` should also match. The values returned for you are different than the *values* in the following output.

NAME	READY	STATUS	RESTARTS	AGE
pod/calico-kube-controllers- <i>579b45dcf-z5tsf</i>	1/1	Running	0	100m
pod/calico-node- <i>v9dhf</i>	1/1	Running	0	100m
pod/calico-typfa- <i>6f9c6786d-f2mc7</i>	1/1	Running	0	100m
...				
NAME	DESIRED	CURRENT	READY	UP-TO-DATE
AVAILABLE NODE SELECTOR	AGE			
daemonset.apps/calico-node	1	1	1	1
kubernetes.io/os=linux	100m			
...				
NAME	DESIRED	CURRENT	READY	AGE
replicaset.apps/calico-kube-controllers- <i>579b45dcf</i>	1	1	1	100m
replicaset.apps/calico-typfa- <i>6f9c6786d</i>	1	1	1	100m

6. Confirm that the logs for one of your `calico-node`, `calico-typfa`, and `tigera-operator` pods don't contain `ERROR`. Replace the values in the following commands with the values returned in your output for the previous steps.

```
kubectl logs tigera-operator-c4b9549c7-h4zp5 -n tigera-operator | grep ERROR
kubectl logs calico-node-v9dhf -n calico-system | grep ERROR
kubectl logs calico-typfa-6f9c6786d-f2mc7 -n calico-system | grep ERROR
```

If no output is returned from the previous commands, then `ERROR` doesn't exist in your logs and everything should be running correctly.

Manifests

To install Calico using manifests

1. Apply the Calico manifests to your cluster. These manifests create a DaemonSet in the `calico-system` namespace.

```
kubectl apply -f https://raw.githubusercontent.com/aws/amazon-vpc-cni-k8s/master/
config/master/calico-operator.yaml
kubectl apply -f https://raw.githubusercontent.com/aws/amazon-vpc-cni-k8s/master/
config/master/calico-crs.yaml
```

2. View the resources in the `calico-system` namespace.

```
kubectl get daemonset calico-node --namespace calico-system
```

Example output:

The values in the `DESIRED` and `READY` columns should match. The values returned for you are different than the `values` in the following output.

NAME	AGE	DESIRED	CURRENT	READY	UP-TO-DATE	AVAILABLE	NODE SELECTOR
calico-node	26m	1	1	1	1	1	kubernetes.io/

Stars policy demo

This section walks through the [Stars policy demo](#) provided by the Project Calico documentation and isn't necessary for Calico functionality on your cluster. The demo creates a front-end, back-end, and client service on your Amazon EKS cluster. The demo also creates a management graphical user interface that shows the available ingress and egress paths between each service. We recommend that you complete the demo on a cluster that you don't run production workloads on.

Before you create any network policies, all services can communicate bidirectionally. After you apply the network policies, you can see that the client can only communicate with the front-end service, and the back-end only accepts traffic from the front-end.

To run the Stars policy demo

1. Apply the front-end, back-end, client, and management user interface services:

```
kubectl apply -f https://docs.projectcalico.org/v3.5/getting-started/kubernetes/
tutorials/stars-policy/manifests/00-namespace.yaml
kubectl apply -f https://docs.projectcalico.org/v3.5/getting-started/kubernetes/
tutorials/stars-policy/manifests/01-management-ui.yaml
kubectl apply -f https://docs.projectcalico.org/v3.5/getting-started/kubernetes/
tutorials/stars-policy/manifests/02-backend.yaml
```

```
kubectl apply -f https://docs.projectcalico.org/v3.5/getting-started/kubernetes/tutorials/stars-policy/manifests/03-frontend.yaml
kubectl apply -f https://docs.projectcalico.org/v3.5/getting-started/kubernetes/tutorials/stars-policy/manifests/04-client.yaml
```

- View all pods on the cluster.

```
kubectl get pods -A
```

Example output:

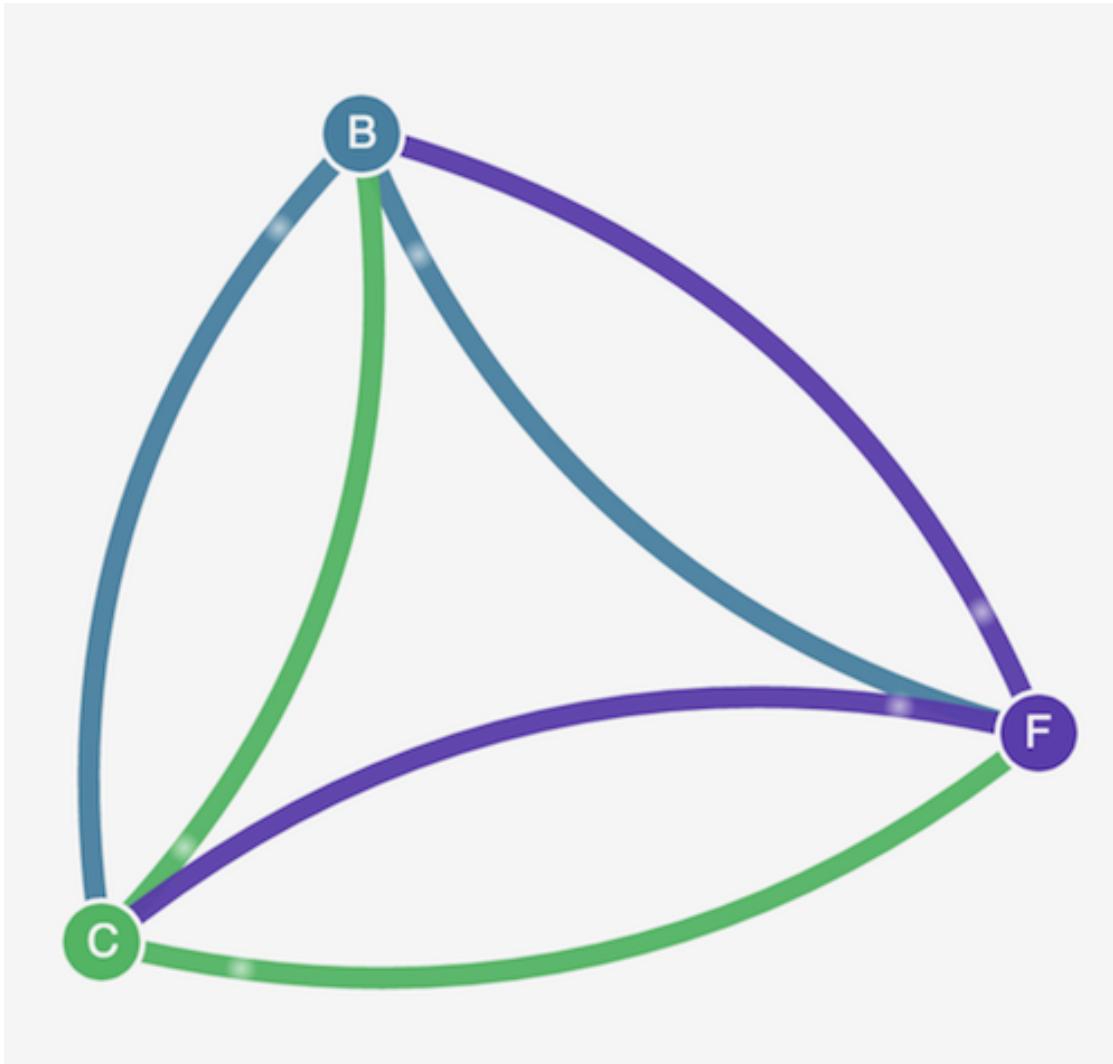
In your output, you should see pods in the namespaces shown in the following output. Your pod **NAMES** and the number of pods in the **READY** column are different than those in the following output. Don't continue until you see pods with similar names and they all have **Running** in the **STATUS** column.

NAMESPACE	NAME	READY	STATUS	RESTARTS
AGE				
...				
client	client- xlffc	1/1	Running	0
	5m19s			
...				
management-ui	management-ui- grb2g	1/1	Running	0
	5m24s			
stars	backend- sz87q	1/1	Running	0
	5m23s			
stars	frontend- cscnf	1/1	Running	0
	5m21s			
...				

- To connect to the management user interface, forward your local port 9001 to the `management-ui` service running on your cluster:

```
kubectl port-forward service/management-ui -n management-ui 9001
```

- Open a browser on your local system and point it to <http://localhost:9001/>. You should see the management user interface. The **C** node is the client service, the **F** node is the front-end service, and the **B** node is the back-end service. Each node has full communication access to all other nodes, as indicated by the bold, colored lines.



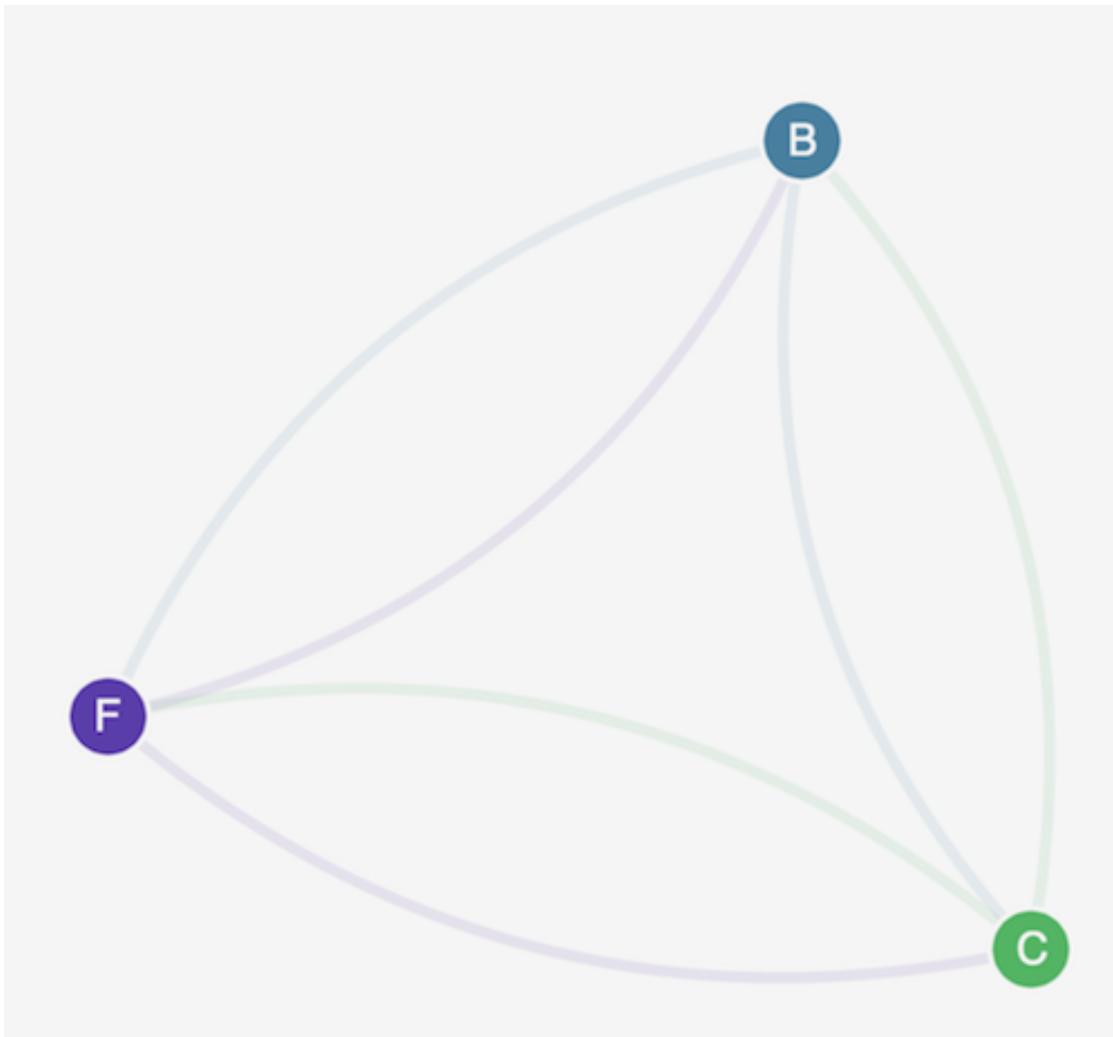
5. Apply the following network policies to isolate the services from each other:

```
kubectl apply -n stars -f https://docs.projectcalico.org/v3.5/getting-started/kubernetes/tutorials/stars-policy/policies/default-deny.yaml  
kubectl apply -n client -f https://docs.projectcalico.org/v3.5/getting-started/kubernetes/tutorials/stars-policy/policies/default-deny.yaml
```

6. Refresh your browser. You see that the management user interface can no longer reach any of the nodes, so they don't show up in the user interface.
7. Apply the following network policies to allow the management user interface to access the services:

```
kubectl apply -f https://docs.projectcalico.org/v3.5/getting-started/kubernetes/tutorials/stars-policy/policies/allow-ui.yaml  
kubectl apply -f https://docs.projectcalico.org/v3.5/getting-started/kubernetes/tutorials/stars-policy/policies/allow-ui-client.yaml
```

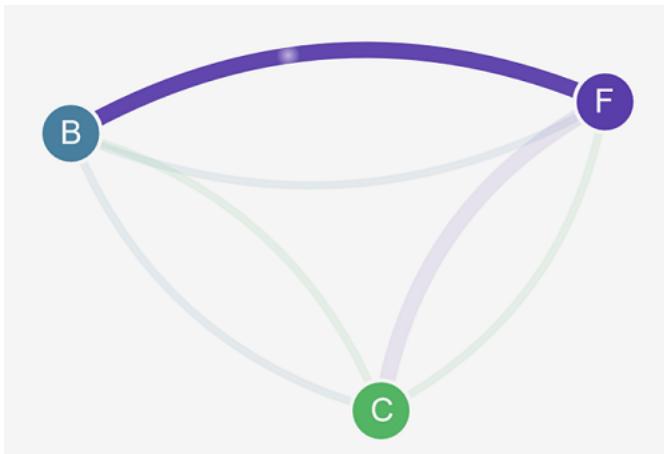
8. Refresh your browser. You see that the management user interface can reach the nodes again, but the nodes cannot communicate with each other.



9. Apply the following network policy to allow traffic from the front-end service to the back-end service:

```
kubectl apply -f https://docs.projectcalico.org/v3.5/getting-started/kubernetes/tutorials/stars-policy/policies/backend-policy.yaml
```

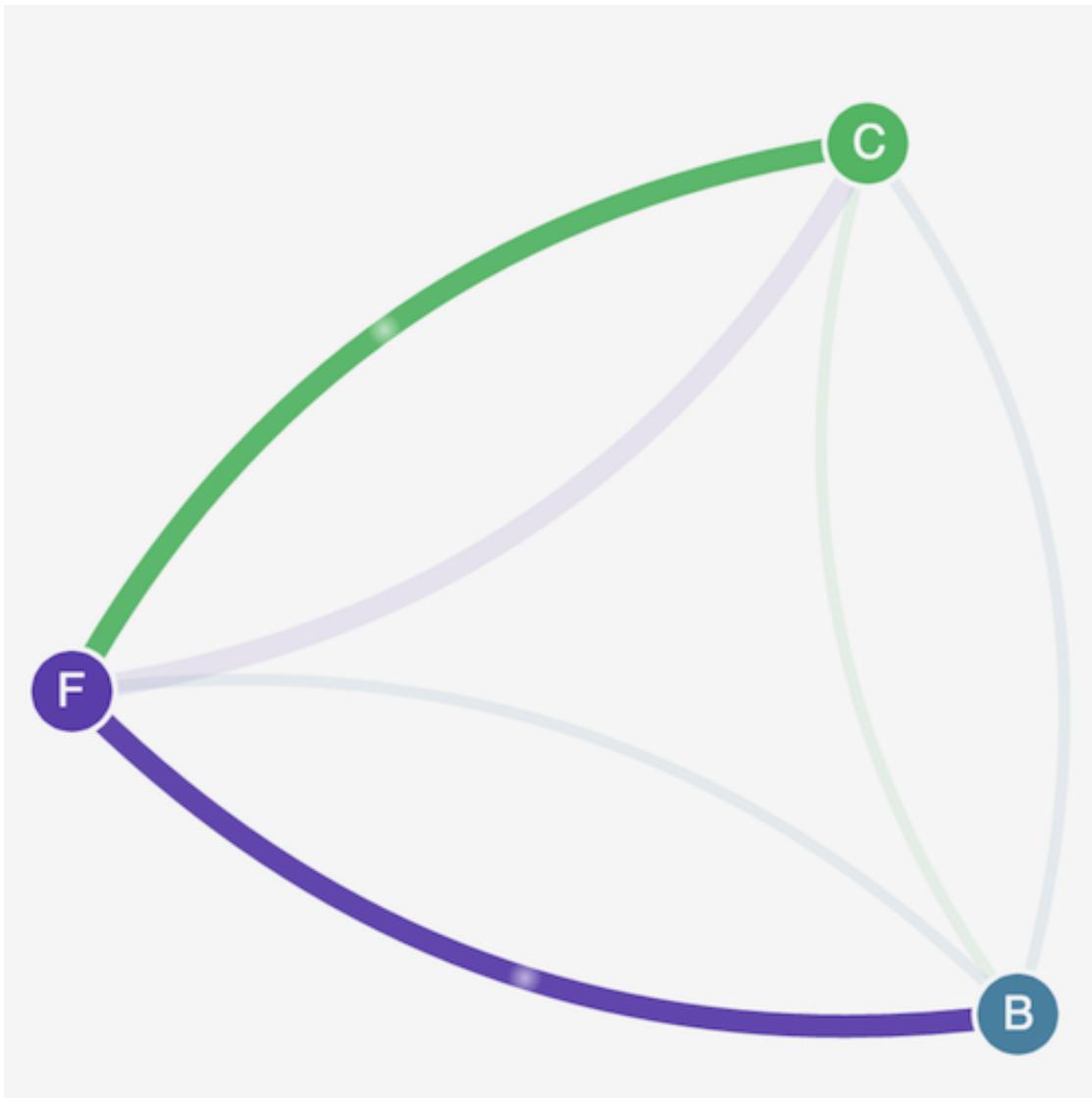
10. Refresh your browser. You see that the front-end can communicate with the back-end.



11. Apply the following network policy to allow traffic from the client to the front-end service.

```
kubectl apply -f https://docs.projectcalico.org/v3.5/getting-started/kubernetes/tutorials/stars-policy/policies/frontend-policy.yaml
```

12. Refresh your browser. You see that the client can communicate to the front-end service. The front-end service can still communicate to the back-end service.



13. (Optional) When you are done with the demo, you can delete its resources.

```
kubectl delete -f https://docs.projectcalico.org/v3.5/getting-started/kubernetes/tutorials/stars-policy/manifests/04-client.yaml
kubectl delete -f https://docs.projectcalico.org/v3.5/getting-started/kubernetes/tutorials/stars-policy/manifests/03-frontend.yaml
kubectl delete -f https://docs.projectcalico.org/v3.5/getting-started/kubernetes/tutorials/stars-policy/manifests/02-backend.yaml
kubectl delete -f https://docs.projectcalico.org/v3.5/getting-started/kubernetes/tutorials/stars-policy/manifests/01-management-ui.yaml
kubectl delete -f https://docs.projectcalico.org/v3.5/getting-started/kubernetes/tutorials/stars-policy/manifests/00-namespace.yaml
```

Even after deleting the resources, there can still be iptables rules on the nodes that might interfere in unexpected ways with networking in your cluster. The only sure way to remove Calico is to terminate all of the nodes and recycle them. To terminate all nodes, either set the Auto Scaling Group desired count to 0, then back up to the desired number, or just terminate the nodes. If you are unable to recycle the nodes, then see [Disabling and removing Calico Policy](#) in the Calico GitHub repository for a last resort procedure.

Remove Calico

Remove Calico using the method that you installed Calico with.

Helm

Remove Calico from your cluster.

```
helm uninstall calico
```

Manifests

Remove Calico from your cluster.

```
kubectl delete -f https://raw.githubusercontent.com/aws/amazon-vpc-cni-k8s/master/
config/master/calico-crs.yaml
kubectl delete -f https://raw.githubusercontent.com/aws/amazon-vpc-cni-k8s/master/
config/master/calico-operator.yaml
```

Workloads

Your workloads are deployed in containers, which are deployed in pods in Kubernetes. A pod includes one or more containers. Typically, one or more pods that provide the same service are deployed in a Kubernetes service. Once you've deployed multiple pods that provide the same service, you can:

- [View information about the workloads \(p. 506\)](#) running on each of your clusters using the AWS Management Console.
- Vertically scale pods up or down with the Kubernetes [Vertical Pod Autoscaler \(p. 366\)](#).
- Horizontally scale the number of pods needed to meet demand up or down with the Kubernetes [Horizontal Pod Autoscaler \(p. 370\)](#).
- Create an external (for internet-accessible pods) or an internal (for private pods) [network load balancer \(p. 372\)](#) to balance network traffic across pods. The load balancer routes traffic at Layer 4 of the OSI model.
- Create an [Application load balancing on Amazon EKS \(p. 378\)](#) to balance application traffic across pods. The application load balancer routes traffic at Layer 7 of the OSI model.
- If you're new to Kubernetes, this topic helps you [Deploy a sample application \(p. 359\)](#).
- You can [restrict IP addresses that can be assigned to a service \(p. 382\)](#) with `externalIPs`.

Deploy a sample application

In this topic, you deploy a sample application to your cluster.

Prerequisites

- An existing Kubernetes cluster. If you don't have an existing Amazon EKS cluster, you can deploy one using one of the [Getting started with Amazon EKS \(p. 4\)](#) guides. If you're deploying a Windows application, then you must have [Windows support \(p. 92\)](#) enabled for your cluster and at least one Amazon EC2 Windows node.
- `kubectl` installed on your computer. For more information, see [Installing kubectl \(p. 4\)](#).
- `kubectl` configured to communicate with your cluster. For more information, see [Create a kubeconfig for Amazon EKS \(p. 413\)](#).
- If you plan to deploy your sample workload to Fargate, then you must have an existing [Fargate profile \(p. 158\)](#) that includes the same namespace created in this tutorial, which is `eks-sample-app`, unless you change the name. If you used one of the [getting started guides \(p. 4\)](#) to create your cluster, then you'll have to create a new profile, or add the namespace to your existing profile, because the profile created in the getting started guides doesn't specify the namespace used in this tutorial. Your VPC must also have at least one private subnet.

To deploy a sample application

Though many variables are changeable in the following steps, we recommend only changing variable values where specified. Once you have a better understanding of Kubernetes pods, deployments, and services, you can experiment with changing other values.

1. Create a namespace. A namespace allows you to group resources in Kubernetes. For more information, see [Namespaces](#) in the Kubernetes documentation. If you plan to deploy your sample application to [AWS Fargate \(p. 153\)](#), make sure that the value for `namespace` in your [AWS Fargate profile \(p. 158\)](#) is `eks-sample-app`.

```
kubectl create namespace eks-sample-app
```

2. Create a Kubernetes deployment. This sample deployment pulls a container image from a public repository and deploys three replicas (individual pods) of it to your cluster. To learn more, see [Deployments](#) in the Kubernetes documentation. You can deploy the application to Linux or Windows nodes. If you're deploying to Fargate, then you can only deploy a Linux application.
 - a. Save the following contents to a file named `eks-sample-deployment.yaml`. The containers in the sample application don't use network storage, but you might have applications that need to. For more information, see [Storage \(p. 229\)](#).

Linux

The `amd64` or `arm64` values under the `kubernetes.io/arch` key mean that the application can be deployed to either hardware architecture (if you have both in your cluster). This is possible because this image is a multi-architecture image, but not all are. You can determine the hardware architecture that the image is supported on by viewing the [image details](#) in the repository that you're pulling it from. When deploying images that don't support a hardware architecture type, or that you don't want the image deployed to, remove that type from the manifest. For more information, see [Well-Known Labels](#), [Annotations and Taints](#) in the Kubernetes documentation.

The `kubernetes.io/os: linux` `nodeSelector` means that if you had Linux and Windows nodes (for example) in your cluster, the image would only be deployed to Linux nodes. For more information, see [Well-Known Labels](#), [Annotations and Taints](#) in the Kubernetes documentation.

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: eks-sample-linux-deployment
  namespace: eks-sample-app
  labels:
    app: eks-sample-linux-app
spec:
  replicas: 3
  selector:
    matchLabels:
      app: eks-sample-linux-app
  template:
    metadata:
      labels:
        app: eks-sample-linux-app
    spec:
      affinity:
        nodeAffinity:
          requiredDuringSchedulingIgnoredDuringExecution:
            nodeSelectorTerms:
              - matchExpressions:
                  - key: kubernetes.io/arch
                    operator: In
                    values:
                      - amd64
                      - arm64
      containers:
        - name: nginx
          image: public.ecr.aws/nginx/nginx:1.21
          ports:
            - name: http
              containerPort: 80
          imagePullPolicy: IfNotPresent
```

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

Windows

The `kubernetes.io/os: windows` nodeSelector means that if you had Windows and Linux nodes (for example) in your cluster, the image would only be deployed to Windows nodes. For more information, see [Well-Known Labels, Annotations and Taints](#) in the Kubernetes documentation.

```
apiVersion: apps/v1  
kind: Deployment  
metadata:  
  name: eks-sample-windows-deployment  
  namespace: eks-sample-app  
  labels:  
    app: eks-sample-windows-app  
spec:  
  replicas: 3  
  selector:  
    matchLabels:  
      app: eks-sample-windows-app  
  template:  
    metadata:  
      labels:  
        app: eks-sample-windows-app  
    spec:  
      affinity:  
        nodeAffinity:  
          requiredDuringSchedulingIgnoredDuringExecution:  
            nodeSelectorTerms:  
              - matchExpressions:  
                  - key: beta.kubernetes.io/arch  
                    operator: In  
                    values:  
                      - amd64  
      containers:  
        - name: windows-server-iis  
          image: mcr.microsoft.com/windows/servercore:ltsc2019  
          ports:  
            - name: http  
              containerPort: 80  
          imagePullPolicy: IfNotPresent  
          command:  
            - powershell.exe  
            - -command  
            - "Add-WindowsFeature Web-Server; Invoke-WebRequest -UseBasicParsing  
              -Uri 'https://dotnetbinaries.blob.core.windows.net/servicemonitor/2.0.1.6/  
              ServiceMonitor.exe' -OutFile 'C:\\ServiceMonitor.exe'; echo '<html><body><br>  
<br><marquee><H1>Hello EKS!!!<H1><marquee></body><html>' > C:\\inetpub\\  
              \\wwwroot\\default.html; C:\\ServiceMonitor.exe 'w3svc'; "  
          nodeSelector:  
            kubernetes.io/os: windows
```

- b. Apply the deployment manifest to your cluster.

```
kubectl apply -f eks-sample-deployment.yaml
```

3. Create a service. A service allows you to access all replicas through a single IP address or name. For more information, see [Service](#) in the Kubernetes documentation. Though not implemented in the sample application, if you have applications that need to interact with other AWS services, we recommend that you create Kubernetes service accounts for your pods, and associate them to AWS

IAM accounts. By specifying service accounts, your pods have only the minimum permissions that you specify for them to interact with other services. For more information, see [IAM roles for service accounts \(p. 442\)](#).

- Save the following contents to a file named `eks-sample-service.yaml`. Kubernetes assigns the service its own IP address that is accessible only from within the cluster. To access the service from outside of your cluster, deploy the [AWS Load Balancer Controller \(p. 329\)](#) to load balance [application \(p. 378\)](#) or [network \(p. 372\)](#) traffic to the service.

Linux

```
apiVersion: v1
kind: Service
metadata:
  name: eks-sample-linux-service
  namespace: eks-sample-app
  labels:
    app: eks-sample-linux-app
spec:
  selector:
    app: eks-sample-linux-app
  ports:
    - protocol: TCP
      port: 80
      targetPort: 80
```

Windows

```
apiVersion: v1
kind: Service
metadata:
  name: eks-sample-windows-service
  namespace: eks-sample-app
  labels:
    app: eks-sample-windows-app
spec:
  selector:
    app: eks-sample-windows-app
  ports:
    - protocol: TCP
      port: 80
      targetPort: 80
```

- Apply the service manifest to your cluster.

```
kubectl apply -f eks-sample-service.yaml
```

- View all resources that exist in the `eks-sample-app` namespace.

```
kubectl get all -n eks-sample-app
```

Example output:

If you deployed Windows resources, then all instances of `linux` in the following output are windows. The other `example values` may be different from your output.

NAME	READY	STATUS	RESTARTS	AGE
pod/eks-sample-linux-deployment- <code>65b7669776-m6qzx</code>	1/1	Running	0	27m
pod/eks-sample-linux-deployment- <code>65b7669776-mmxxvd</code>	1/1	Running	0	27m
pod/eks-sample-linux-deployment- <code>65b7669776-qzn22</code>	1/1	Running	0	27m

NAME	TYPE	CLUSTER-IP	EXTERNAL-IP	PORT(S)
service/eks-sample- linux -service	ClusterIP	10.100.74.8	<none>	80/TCP
32m				
NAME	READY	UP-TO-DATE	AVAILABLE	AGE
deployment.apps/eks-sample- linux -deployment	3/3	3	3	27m
27m				
NAME	DESIRED	CURRENT	READY	
replicaset.apps/eks-sample- linux -deployment- 776d8f8fd8	3	3	3	
27m				

In the output, you see the service and deployment that were specified in the sample manifests deployed in previous steps. You also see three pods. This is because 3 replicas were specified in the sample manifest. For more information about pods, see [Pods](#) in the Kubernetes documentation. Kubernetes automatically creates the replicaset resource, even though it isn't specified in the sample manifests. For more information about ReplicaSets, see [ReplicaSet](#) in the Kubernetes documentation.

Note

Kubernetes maintains the number of replicas that are specified in the manifest. If this were a production deployment and you wanted Kubernetes to horizontally scale the number of replicas or vertically scale the compute resources for the pods, use the [Horizontal Pod Autoscaler \(p. 370\)](#) and the [Vertical Pod Autoscaler \(p. 366\)](#) to do so.

- View the details of the deployed service. If you deployed a Windows service, replace **linux** with **windows**.

```
kubectl -n eks-sample-app describe service eks-sample-linux-service
```

Example output:

If you deployed Windows resources, then all instances of **linux** in the following output are **windows**. The other *example values* may be different from your output.

Name:	eks-sample- linux -service
Namespace:	eks-sample-app
Labels:	app=eks-sample- linux -app
Annotations:	<none>
Selector:	app=eks-sample- linux -app
Type:	ClusterIP
IP Families:	<none>
IP:	10.100.74.8
IPs:	10.100.74.8
Port:	<unset> 80/TCP
TargetPort:	80/TCP
Endpoints:	192.168.24.212:80,192.168.50.185:80,192.168.63.93:80
Session Affinity:	None
Events:	<none>

In the previous output, the value for **IP:** is a unique IP address that can be reached from any node or pod within the cluster, but it can't be reached from outside of the cluster. The values for **Endpoints** are IP addresses assigned from within your VPC to the pods that are part of the service.

- View the details of one of the pods listed in the output when you [viewed the namespace \(p. 362\)](#) in a previous step. If you deployed a Windows app, replace **linux** with **windows** and replace **776d8f8fd8-78w66** with the value returned for one of your pods.

```
kubectl -n eks-sample-app describe pod eks-sample-linux-deployment-65b7669776-m6qxz
```

Abbreviated output

If you deployed Windows resources, then all instances of `linux` in the following output are `windows`. The other *example values* may be different from your output.

```
Name:           eks-sample-linux-deployment-65b7669776-m6qxz
Namespace:      eks-sample-app
Priority:       0
Node:           ip-192-168-45-132.us-west-2.compute.internal/192.168.45.132
...
IP:             192.168.63.93
IPs:
  IP:           192.168.63.93
Controlled By: ReplicaSet/eks-sample-linux-deployment-65b7669776
...
Conditions:
  Type        Status
  Initialized  True
  Ready        True
  ContainersReady  True
  PodScheduled  True
...
Events:
  Type    Reason     Age      From
  Message
  ----  -----  ----  -----
  -----
  Normal  Scheduled  3m20s  default-scheduler
  Successfully assigned eks-sample-app/eks-sample-linux-deployment-65b7669776-m6qxz to
  ip-192-168-45-132.us-west-2.compute.internal
  ...
```

In the previous output, the value for `IP:` is a unique IP that's assigned to the pod from the CIDR block assigned to the subnet that the node is in. If you prefer to assign pods IP addresses from different CIDR blocks, you can change the default behavior. For more information, see [CNI custom networking \(p. 306\)](#). You can also see that the Kubernetes scheduler scheduled the pod on the Node with the IP address `192.168.45.132`.

Tip

Rather than using the command line, you can view many details about pods, services, deployments, and other Kubernetes resources in the AWS Management Console. For more information, see [View Kubernetes resources \(p. 506\)](#).

7. Run a shell on the pod that you described in the previous step, replacing `65b7669776-m6qxz` with the ID of one of your pods.

Linux

```
kubectl exec -it eks-sample-linux-deployment-65b7669776-m6qxz -n eks-sample-app
-- /bin/bash
```

Windows

```
kubectl exec -it eks-sample-windows-deployment-65b7669776-m6qxz -n eks-sample-app
-- powershell.exe
```

8. From the pod shell, view the output from the web server that was installed with your deployment in a previous step. You only need to specify the service name. It is resolved to the service's IP address by CoreDNS, which is deployed with an Amazon EKS cluster, by default.

Linux

```
curl eks-sample-linux-service
```

Example output:

```
<!DOCTYPE html>
<html>
<head>
<title>Welcome to nginx!</title>
...

```

Windows

```
Invoke-WebRequest -uri eks-sample-windows-service/default.html -UseBasicParsing
```

Example output:

```
Statuscode : 200
StatusDescription : OK
Content : < h t m l > < b o d y > < b r / > < b r / > < m a r q u e e > <
H 1 > H e l l o
                           E K S ! ! ! < H 1 > < m a r q u e e > < / b o d y > < h t m l
>
```

9. From the pod shell, view the DNS server for the pod.

Linux

```
cat /etc/resolv.conf
```

Example output:

```
nameserver 10.100.0.10
search eks-sample-app.svc.cluster.local svc.cluster.local cluster.local us-
west-2.compute.internal
options ndots:5
```

In the previous output, 10.100.0.10 is automatically assigned as the `nameserver` for all pods deployed to the cluster.

Windows

```
Get-NetIPConfiguration
```

Abbreviated output

```
InterfaceAlias      : vEthernet
...
IPv4Address         : 192.168.63.14
...
DNSServer          : 10.100.0.10
```

In the previous output, 10.100.0.10 is automatically assigned as the DNS server for all pods deployed to the cluster.

10. Disconnect from the pod by typing `exit`.
11. Once you're finished with the sample application, you can remove the sample namespace, service, and deployment with the following command.

```
kubectl delete namespace eks-sample-app
```

Vertical Pod Autoscaler

The Kubernetes [Vertical Pod Autoscaler](#) automatically adjusts the CPU and memory reservations for your pods to help "right size" your applications. This adjustment can improve cluster resource utilization and free up CPU and memory for other pods. This topic helps you to deploy the Vertical Pod Autoscaler to your cluster and verify that it is working.

Prerequisites

- You have an existing Amazon EKS cluster. If you don't, see [Getting started with Amazon EKS \(p. 4\)](#).
- You have the Kubernetes Metrics Server installed. For more information, see [Installing the Kubernetes Metrics Server \(p. 426\)](#).
- You are using a `kubectl` client that is [configured to communicate with your Amazon EKS cluster \(p. 18\)](#).
- OpenSSL 1.1.1 or later installed on your device.

Deploy the Vertical Pod Autoscaler

In this section, you deploy the Vertical Pod Autoscaler to your cluster.

To deploy the Vertical Pod Autoscaler

1. Open a terminal window and navigate to a directory where you would like to download the Vertical Pod Autoscaler source code.
2. Clone the [kubernetes/autoscaler](#) GitHub repository.

```
git clone https://github.com/kubernetes/autoscaler.git
```

3. Change to the `vertical-pod-autoscaler` directory.

```
cd autoscaler/vertical-pod-autoscaler/
```

4. (Optional) If you have already deployed another version of the Vertical Pod Autoscaler, remove it with the following command.

```
./hack/vpa-down.sh
```

5. If your nodes don't have internet access to the `k8s.gcr.io` container registry, then you need to pull the following images and push them to your own private repository. For more information about how to pull the images and push them to your own private repository, see [Copy a container image from one repository to another repository \(p. 384\)](#).

```
k8s.gcr.io/autoscaling/vpa-admission-controller:0.10.0
```

```
k8s.gcr.io/autoscaling/vpa-recommender:0.10.0
k8s.gcr.io/autoscaling/vpa-updater:0.10.0
```

If you're pushing the images to a private Amazon ECR repository, then replace `k8s.gcr.io` in the manifests with your registry. Replace `111122223333` with your account ID. Replace `region-code` with the AWS Region that your cluster is in. The following commands assume that you named your repository the same as the repository name in the manifest. If you named your repository something different, then you'll need to change it too.

```
sed -i.bak -e 's/k8s.gcr.io/111122223333.dkr.ecr.region-codeamazonaws.com/' ./deploy/
admission-controller-deployment.yaml
sed -i.bak -e 's/k8s.gcr.io/111122223333.dkr.ecr..dkr.ecr.region-codeamazonaws.com/' ./deploy/recommender-deployment.yaml
sed -i.bak -e 's/k8s.gcr.io/111122223333.dkr.ecr..dkr.ecr.region-codeamazonaws.com/' ./deploy/updater-deployment.yaml
```

- Deploy the Vertical Pod Autoscaler to your cluster with the following command.

```
./hack/vpa-up.sh
```

- Verify that the Vertical Pod Autoscaler pods have been created successfully.

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

Output:

NAME	READY	STATUS	RESTARTS	AGE
...				
metrics-server-8459fc497-kfj8w	1/1	Running	0	83m
vpa-admission-controller-68c748777d-ppspd	1/1	Running	0	7s
vpa-recommender-6fc8c67d85-gljpl	1/1	Running	0	8s
vpa-updater-786b96955c-bgp9d	1/1	Running	0	8s

Test your Vertical Pod Autoscaler installation

In this section, you deploy a sample application to verify that the Vertical Pod Autoscaler is working.

To test your Vertical Pod Autoscaler installation

- Deploy the `hamster.yaml` Vertical Pod Autoscaler example with the following command.

```
kubectl apply -f examples/hamster.yaml
```

- Get the pods from the `hamster` example application.

```
kubectl get pods -l app=hamster
```

Output:

hamster-c7d89d6db-rglf5	1/1	Running	0	48s
hamster-c7d89d6db-znvz5	1/1	Running	0	48s

- Describe one of the pods to view its `cpu` and `memory` reservation. Replace `c7d89d6db-rglf5` with one of the IDs returned in your output from the previous step.

```
kubectl describe pod hamster-c7d89d6db-rglf5
```

Output:

```
...
Containers:
  hamster:
    Container ID: docker:///
e76c2413fc720ac395c33b64588c82094fc8e5d590e373d5f818f3978f577e24
      Image:          k8s.gcr.io/ubuntu-slim:0.1
      Image ID:       docker-pullable://k8s.gcr.io/ubuntu-
slim@sha256:b6f8c3885f5880a4f1a7cf717c07242eb4858fdd5a84b5ffe35b1cf680ea17b1
      Port:           <none>
      Host Port:     <none>
      Command:       /bin/sh
      Args:
        -c
        while true; do timeout 0.5s yes >/dev/null; sleep 0.5s; done
      State:          Running
      Started:        Fri, 27 Sep 2019 10:35:16 -0700
      Ready:          True
      Restart Count:  0
      Requests:
        cpu:          100m
        memory:       50Mi
...
...
```

You can see that the original pod reserves 100 millicpu of CPU and 50 mebibytes of memory. For this example application, 100 millicpu is less than the pod needs to run, so it is CPU-constrained. It also reserves much less memory than it needs. The Vertical Pod Autoscaler `vpa-recommender` deployment analyzes the `hamster` pods to see if the CPU and memory requirements are appropriate. If adjustments are needed, the `vpa-updater` relaunches the pods with updated values.

4. Wait for the `vpa-updater` to launch a new `hamster` pod. This should take a minute or two. You can monitor the pods with the following command.

Note

If you are not sure that a new pod has launched, compare the pod names with your previous list. When the new pod launches, you will see a new pod name.

```
kubectl get --watch pods -l app=hamster
```

5. When a new `hamster` pod is started, describe it and view the updated CPU and memory reservations.

```
kubectl describe pod hamster-c7d89d6db-jxgfv
```

Output:

```
...
Containers:
  hamster:
    Container ID:
docker://2c3e7b6fb7ce0d8c86444334df654af6fb3fc88aad4c5d710ead3b1e7c58f7db
      Image:          k8s.gcr.io/ubuntu-slim:0.1
      Image ID:       docker-pullable://k8s.gcr.io/ubuntu-
slim@sha256:b6f8c3885f5880a4f1a7cf717c07242eb4858fdd5a84b5ffe35b1cf680ea17b1
      Port:           <none>
```

```

Host Port:      <none>
Command:
  /bin/sh
Args:
  -c
    while true; do timeout 0.5s yes >/dev/null; sleep 0.5s; done
State:          Running
  Started:      Fri, 27 Sep 2019 10:37:08 -0700
Ready:          True
Restart Count:  0
Requests:
  cpu:          587m
  memory:       262144k
...

```

In the previous output, you can see that the `cpu` reservation increased to 587 millicpu, which is over five times the original value. The `memory` increased to 262,144 Kilobytes, which is around 250 mebibytes, or five times the original value. This pod was under-resourced, and the Vertical Pod Autoscaler corrected the estimate with a much more appropriate value.

6. Describe the `hamster-vpa` resource to view the new recommendation.

```
kubectl describe vpa/hamster-vpa
```

Output:

```

Name:        hamster-vpa
Namespace:   default
Labels:      <none>
Annotations: kubectl.kubernetes.io/last-applied-configuration:
             {"apiVersion":"autoscaling.k8s.io/v1beta2","kind":"VerticalPodAutoscaler","metadata":{"annotations":{},"name":"hamster-vpa","namespace":"d...}}
API Version: autoscaling.k8s.io/v1beta2
Kind:        VerticalPodAutoscaler
Metadata:
  Creation Timestamp: 2019-09-27T18:22:51Z
  Generation:        23
  Resource Version:  14411
  Self Link:         /apis/autoscaling.k8s.io/v1beta2/namespaces/default/verticalpodautoscalers/hamster-vpa
  UID:               d0d85fb9-e153-11e9-ae53-0205785d75b0
Spec:
  Target Ref:
    API Version:  apps/v1
    Kind:         Deployment
    Name:         hamster
Status:
  Conditions:
    Last Transition Time: 2019-09-27T18:23:28Z
    Status:        True
    Type:         RecommendationProvided
  Recommendation:
    Container Recommendations:
      Container Name: hamster
    Lower Bound:
      Cpu:      550m
      Memory:  262144k
    Target:
      Cpu:      587m
      Memory:  262144k
    Uncapped Target:
      Cpu:      587m

```

```
Memory: 262144k
Upper Bound:
Cpu: 21147m
Memory: 387863636
Events: <none>
```

- When you finish experimenting with the example application, you can delete it with the following command.

```
kubectl delete -f examples/hamster.yaml
```

Horizontal Pod Autoscaler

The Kubernetes [Horizontal Pod Autoscaler](#) automatically scales the number of pods in a deployment, replication controller, or replica set based on that resource's CPU utilization. This can help your applications scale out to meet increased demand or scale in when resources are not needed, thus freeing up your nodes for other applications. When you set a target CPU utilization percentage, the Horizontal Pod Autoscaler scales your application in or out to try to meet that target.

The Horizontal Pod Autoscaler is a standard API resource in Kubernetes that simply requires that a metrics source (such as the Kubernetes metrics server) is installed on your Amazon EKS cluster to work. You do not need to deploy or install the Horizontal Pod Autoscaler on your cluster to begin scaling your applications. For more information, see [Horizontal Pod Autoscaler](#) in the Kubernetes documentation.

Use this topic to prepare the Horizontal Pod Autoscaler for your Amazon EKS cluster and to verify that it is working with a sample application.

Note

This topic is based on the [Horizontal pod autoscaler walkthrough](#) in the Kubernetes documentation.

Prerequisites

- You have an existing Amazon EKS cluster. If you don't, see [Getting started with Amazon EKS \(p. 4\)](#).
- You have the Kubernetes Metrics Server installed. For more information, see [Installing the Kubernetes Metrics Server \(p. 426\)](#).
- You are using a `kubectl` client that is [configured to communicate with your Amazon EKS cluster \(p. 18\)](#).

Run a Horizontal Pod Autoscaler test application

In this section, you deploy a sample application to verify that the Horizontal Pod Autoscaler is working.

Note

This example is based on the [Horizontal pod autoscaler walkthrough](#) in the Kubernetes documentation.

To test your Horizontal Pod Autoscaler installation

- Deploy a simple Apache web server application with the following command.

```
kubectl apply -f https://k8s.io/examples/application/php-apache.yaml
```

This Apache web server pod is given a 500 millicpu CPU limit and it is serving on port 80.

2. Create a Horizontal Pod Autoscaler resource for the php-apache deployment.

```
kubectl autoscale deployment php-apache --cpu-percent=50 --min=1 --max=10
```

This command creates an autoscaler that targets 50 percent CPU utilization for the deployment, with a minimum of one pod and a maximum of ten pods. When the average CPU load is below 50 percent, the autoscaler tries to reduce the number of pods in the deployment, to a minimum of one. When the load is greater than 50 percent, the autoscaler tries to increase the number of pods in the deployment, up to a maximum of ten. For more information, see [How does the Horizontal Pod Autoscaler work?](#) in the Kubernetes documentation.

3. Describe the autoscaler with the following command to view its details.

```
kubectl get hpa
```

Output:

NAME	REFERENCE	TARGETS	MINPODS	MAXPODS	REPLICAS	AGE
php-apache	Deployment/php-apache	0%/50%	1	10	1	51s

As you can see, the current CPU load is 0%, because there's no load on the server yet. The pod count is already at its lowest boundary (one), so it cannot scale in.

4. Create a load for the web server by running a container.

```
kubectl run -i \
    --tty load-generator \
    --rm --image=busybox \
    --restart=Never \
    -- /bin/sh -c "while sleep 0.01; do wget -q -O- http://php-apache; done"
```

5. To watch the deployment scale out, periodically run the following command in a separate terminal from the terminal that you ran the previous step in.

```
kubectl get hpa php-apache
```

Output:

NAME	REFERENCE	TARGETS	MINPODS	MAXPODS	REPLICAS	AGE
php-apache	Deployment/php-apache	250%/50%	1	10	5	4m44s

It may take over a minute for the replica count to increase. As long as actual CPU percentage is higher than the target percentage, then the replica count increases, up to 10. In this case, it's 250%, so the number of REPLICAS continues to increase.

Note

It may take a few minutes before you see the replica count reach its maximum. If only 6 replicas, for example, are necessary for the CPU load to remain at or under 50%, then the load won't scale beyond 6 replicas.

6. Stop the load. In the terminal window you're generating the load in, stop the load by holding down the `Ctrl+C` keys. You can watch the replicas scale back to 1 by running the following command again in the terminal that you're watching the scaling in.

```
kubectl get hpa
```

Example output:

NAME	REFERENCE	TARGETS	MINPODS	MAXPODS	REPLICAS	AGE
php-apache	Deployment/php-apache	0%/50%	1	10	1	25m

Note

The default timeframe for scaling back down is five minutes, so it will take some time before you see the replica count reach 1 again, even when the current CPU percentage is 0 percent. The timeframe is modifiable. For more information, see [Horizontal Pod Autoscaler](#) in the Kubernetes documentation.

- When you are done experimenting with your sample application, delete the php-apache resources.

```
kubectl delete deployment.apps/php-apache service/php-apache
horizontalpodautoscaler.autoscaling/php-apache
```

Network load balancing on Amazon EKS

Network traffic is load balanced at L4 of the OSI model. To load balance application traffic at L7, you deploy a Kubernetes ingress, which provisions an AWS Application Load Balancer. For more information, see [Application load balancing on Amazon EKS \(p. 378\)](#). To learn more about the differences between the two types of load balancing, see [Elastic Load Balancing features](#) on the AWS website.

When you create a Kubernetes Service of type `LoadBalancer`, the AWS cloud provider load balancer controller creates AWS [Classic Load Balancers](#) by default, but can also create AWS [Network Load Balancers](#). This controller is only receiving critical bug fixes in the future. For more information about using the AWS cloud provider load balancer, see [AWS cloud provider load balancer controller](#) in the Kubernetes documentation. Its use is not covered in this topic.

For new services deployed to 1.19 or later clusters, we recommend that you use version 2.4.1 or later of the [the section called “Installing the AWS Load Balancer Controller add-on” \(p. 329\)](#) instead of the AWS cloud provider load balancer controller. If your cluster is earlier than 1.19, then we recommend that you use version 2.3.1 of the controller. The AWS Load Balancer Controller creates AWS Network Load Balancers, but doesn't create AWS Classic Load Balancers. The remainder of this topic is about using the AWS Load Balancer Controller.

An AWS Network Load Balancer can load balance network traffic to pods deployed to Amazon EC2 IP and instance targets or to AWS Fargate IP targets. For more information, see [AWS Load Balancer Controller](#) on GitHub.

Prerequisites

Before you can load balance network traffic using the AWS Load Balancer Controller, you must meet the following requirements.

- Have an existing cluster. If you don't have an existing cluster, see [Getting started with Amazon EKS \(p. 4\)](#). If you need to update the version of an existing cluster, see [Updating a cluster \(p. 34\)](#).
- Have the AWS Load Balancer Controller deployed on your cluster. For more information, see [Installing the AWS Load Balancer Controller add-on \(p. 329\)](#). We recommend version 2.4.1 or later for 1.19 or later clusters. If your cluster is earlier than 1.19, then we recommend using version 2.3.1.
- At least one subnet. If multiple tagged subnets are found in an Availability Zone, the controller chooses the first subnet whose subnet ID comes first lexicographically. The subnet must have at least eight available IP addresses.
- If you're using the AWS Load Balancer Controller version v2.1.1 or earlier, subnets must be tagged as follows. If using version 2.1.2 or later, this tag is optional. You might want to tag a subnet if you have multiple clusters running in the same VPC, or multiple AWS services sharing subnets in a VPC,

and want more control over where load balancers are provisioned for each cluster. If you explicitly specify subnet IDs as an annotation on a service object, then Kubernetes and the AWS Load Balancer Controller use those subnets directly to create the load balancer. Subnet tagging isn't required if you choose to use this method for provisioning load balancers and you can skip the following private and public subnet tagging requirements. Replace `cluster-name` with your cluster name.

- **Key** – `kubernetes.io/cluster/cluster-name`
- **Value** – shared or owned
- Your public and private subnets must meet the following requirements, unless you explicitly specify subnet IDs as an annotation on a service or ingress object. If you provision load balancers by explicitly specifying subnet IDs as an annotation on a service or ingress object, then Kubernetes and the AWS Load Balancer Controller use those subnets directly to create the load balancer and the following tags aren't required.
 - **Private subnets** – Must be tagged in the following format. This is so that Kubernetes and the AWS Load Balancer Controller know that the subnets can be used for internal load balancers. If you use `eksctl` or an Amazon EKS AWS CloudFormation template to create your VPC after March 26, 2020, then the subnets are tagged appropriately when they're created. For more information about the Amazon EKS AWS CloudFormation VPC templates, see [Creating a VPC for your Amazon EKS cluster \(p. 265\)](#).
 - **Key** – `kubernetes.io/role/internal-elb`
 - **Value** – 1
 - **Public subnets** – Must be tagged in the following format. This is so that Kubernetes knows to use only those subnets for external load balancers instead of choosing a public subnet in each Availability Zone (based on the lexicographical order of the subnet IDs). If you use `eksctl` or an Amazon EKS AWS CloudFormation template to create your VPC after March 26, 2020, then the subnets are tagged appropriately when they're created. For more information about the Amazon EKS AWS CloudFormation VPC templates, see [Creating a VPC for your Amazon EKS cluster \(p. 265\)](#).
 - **Key** – `kubernetes.io/role/elb`
 - **Value** – 1

If the subnet role tags aren't explicitly added, the Kubernetes service controller examines the route table of your cluster VPC subnets to determine if the subnet is private or public. We recommend that you don't rely on this behavior, and instead explicitly add the private or public role tags. The AWS Load Balancer Controller doesn't examine route tables, and requires the private and public tags to be present for successful auto discovery.

Considerations

- The configuration of your load balancer is controlled by annotations that are added to the manifest for your service. Service annotations are different when using the AWS Load Balancer Controller than they are when using the AWS cloud provider load balancer controller. Make sure to review the [annotations](#) for the AWS Load Balancer Controller before deploying services.
- When using the [Amazon EKS VPC CNI plugin \(p. 276\)](#), the AWS Load Balancer Controller can load balance to Amazon EC2 IP or instance targets and Fargate IP targets. When using [Alternate compatible CNI plugins \(p. 329\)](#), the controller can only load balance to instance targets. For more information about Network Load Balancer target types, see [Target type](#) in the User Guide for Network Load Balancers
- If you want to add tags to the load balancer when or after it's created, add the following annotation in your service specification. For more information, see [AWS Resource Tags](#) in the AWS Load Balancer Controller documentation.

```
service.beta.kubernetes.io/aws-load-balancer-additional-resource-tags
```

- You can assign [Elastic IP addresses](#) to the Network Load Balancer by adding the following annotation. Replace the `example-values` with the Allocation IDs of your Elastic IP addresses. The number of

Allocation IDs must match the number of subnets that are used for the load balancer. For more information, see the [AWS Load Balancer Controller](#) documentation.

```
service.beta.kubernetes.io/aws-load-balancer-eip-allocations:  
  eipalloc-xxxxxxxxxxxxxxxxxx, eipalloc-yyyyyyyyyyyyyyyyyy
```

- Amazon EKS adds one inbound rule to the node's security group for client traffic and one rule for each load balancer subnet in the VPC for health checks for each Network Load Balancer that you create. Deployment of a service of type `LoadBalancer` can fail if Amazon EKS attempts to create rules that exceed the quota for the maximum number of rules allowed for a security group. For more information, see [Security groups](#) in Amazon VPC quotas in the Amazon VPC User Guide. Consider the following options to minimize the chances of exceeding the maximum number of rules for a security group:
 - Request an increase in your rules per security group quota. For more information, see [Requesting a quota increase](#) in the Service Quotas User Guide.
 - Use IP targets, rather than instance targets. With IP targets, you can share rules for the same target ports. You can manually specify load balancer subnets with an annotation. For more information, see [Annotations](#) on GitHub.
 - Use an ingress, instead of a service of type `LoadBalancer`, to send traffic to your service. The AWS Application Load Balancer requires fewer rules than Network Load Balancers. You can share an ALB across multiple ingresses. For more information, see [Application load balancing on Amazon EKS \(p. 378\)](#). You can't share a Network Load Balancer across multiple services.
 - Deploy your clusters to multiple accounts.
- If your pods run on Windows in an Amazon EKS cluster, a single service with a load balancer can support up to 64 backend pods. Each pod has its own unique IP address. This is a limitation of the Windows OS on the Amazon EC2 nodes.
- We recommend only creating new Network Load Balancers with the AWS Load Balancer Controller. Attempting to replace existing Network Load Balancers created with the AWS cloud provider load balancer controller can result in multiple Network Load Balancers that might cause application downtime.

Create a network load balancer

You can create a network load balancer with IP or instance targets.

IP targets

You can use IP targets with pods deployed to Amazon EC2 nodes or Fargate. Your Kubernetes service must be created as type `LoadBalancer`. For more information, see [Type LoadBalancer](#) in the Kubernetes documentation.

To create a load balancer that uses IP targets, add the following annotations to a service manifest and deploy your service. The `external` value for `aws-load-balancer-type` is what causes the AWS Load Balancer Controller, rather than the AWS cloud provider load balancer controller, to create the Network Load Balancer. You can view a [sample service manifest \(p. 377\)](#) with the annotations.

```
service.beta.kubernetes.io/aws-load-balancer-type: "external"  
service.beta.kubernetes.io/aws-load-balancer-nlb-target-type: "ip"
```

Note

If you're load balancing to IPv6 pods, add the following annotation. You can only load balance over IPv6 to IP targets, not instance targets. Without this annotation, load balancing is over IPv4.

```
service.beta.kubernetes.io/aws-load-balancer-ip-address-type: dualstack
```

Network Load Balancers are created with the `internal aws-load-balancer-scheme`, by default. For internal Network Load Balancers, your Amazon EKS cluster must be configured to use at least one private subnet in your VPC. Kubernetes examines the route table for your subnets to identify whether they are public or private. Public subnets have a route directly to the internet using an internet gateway, but private subnets do not.

If you want to create a Network Load Balancer in a public subnet to load balance to Amazon EC2 nodes (Fargate can only be private), specify `internet-facing` with the following annotation:

```
service.beta.kubernetes.io/aws-load-balancer-scheme: "internet-facing"
```

Note

The `service.beta.kubernetes.io/aws-load-balancer-type: "nlb-ip"` annotation is still supported for backwards compatibility. However, we recommend using the previous annotations for new load balancers instead of `service.beta.kubernetes.io/aws-load-balancer-type: "nlb-ip"`.

Important

Do not edit the annotations after creating your service. If you need to modify it, delete the service object and create it again with the desired value for this annotation.

Instance targets

The AWS cloud provider load balancer controller creates Network Load Balancers with instance targets only. Version 2.2.0 and later of the AWS Load Balancer Controller also creates Network Load Balancers with instance targets. We recommend using it, rather than the AWS cloud provider load balancer controller, to create new Network Load Balancers. You can use Network Load Balancer instance targets with pods deployed to Amazon EC2 nodes, but not to Fargate. To load balance network traffic across pods deployed to Fargate, you must use IP targets.

To deploy a Network Load Balancer to a private subnet, your service specification must have the following annotations. You can view a [sample service manifest \(p. 377\)](#) with the annotations. The external value for `aws-load-balancer-type` is what causes the AWS Load Balancer Controller, rather than the AWS cloud provider load balancer controller, to create the Network Load Balancer.

```
service.beta.kubernetes.io/aws-load-balancer-type: "external"
service.beta.kubernetes.io/aws-load-balancer-nlb-target-type: "instance"
```

Network Load Balancers are created with the `internal aws-load-balancer-scheme`, by default. For internal Network Load Balancers, your Amazon EKS cluster must be configured to use at least one private subnet in your VPC. Kubernetes examines the route table for your subnets to identify whether they are public or private. Public subnets have a route directly to the internet using an internet gateway, but private subnets do not.

If you want to create an Network Load Balancer in a public subnet to load balance to Amazon EC2 nodes, specify `internet-facing` with the following annotation:

```
service.beta.kubernetes.io/aws-load-balancer-scheme: "internet-facing"
```

Important

Do not edit the annotations after creating your service. If you need to modify it, delete the service object and create it again with the desired value for this annotation.

(Optional) Deploy a sample application

Prerequisites

- At least one public or private subnet in your cluster VPC.
- Have the AWS Load Balancer Controller deployed on your cluster. For more information, see [Installing the AWS Load Balancer Controller add-on \(p. 329\)](#). We recommend version 2.4.1 or later.

To deploy a sample application

1. If you're deploying to Fargate, make sure you have an available private subnet in your VPC and create a Fargate profile. If you're not deploying to Fargate, skip this step. You can create the profile by running the following command or in the [AWS Management Console \(p. 159\)](#) using the same values for name and namespace that are in the command. Replace the *example values* with your own.

```
eksctl create fargateprofile \
  --cluster my-cluster \
  --region region-code \
  --name nlb-sample-app \
  --namespace nlb-sample-app
```

2. Deploy a sample application.
 - a. Create a namespace for the application.

```
kubectl create namespace nlb-sample-app
```

- b. Save the following contents to a file named *sample-deployment.yaml* file on your computer.

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nlb-sample-app
  namespace: nlb-sample-app
spec:
  replicas: 3
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
        - name: nginx
          image: public.ecr.aws/nginx/nginx:1.21
          ports:
            - name: tcp
              containerPort: 80
```

- c. Apply the manifest to the cluster.

```
kubectl apply -f sample-deployment.yaml
```

3. Create a service with an internet-facing Network Load Balancer that load balances to IP targets.

- a. Save the following contents to a file named `sample-service.yaml` on your computer. If you're deploying to Fargate nodes, remove the `service.beta.kubernetes.io/aws-load-balancer-scheme: internet-facing` line.

```
apiVersion: v1
kind: Service
metadata:
  name: nlb-sample-service
  namespace: nlb-sample-app
  annotations:
    service.beta.kubernetes.io/aws-load-balancer-type: external
    service.beta.kubernetes.io/aws-load-balancer-nlb-target-type: ip
    service.beta.kubernetes.io/aws-load-balancer-scheme: internet-facing
spec:
  ports:
    - port: 80
      targetPort: 80
      protocol: TCP
    type: LoadBalancer
  selector:
    app: nginx
```

- b. Apply the manifest to the cluster.

```
kubectl apply -f sample-service.yaml
```

4. Verify that the service was deployed.

```
kubectl get svc nlb-sample-service -n nlb-sample-app
```

Example output:

NAME	TYPE	CLUSTER-IP	EXTERNAL-IP	PORT(S)	AGE
sample-service	LoadBalancer	10.100.240.137	k8s-nlbsampl-nlbsampl-xxxxxxxx-xxxxxxxxxxxxxx.elb.region-code.amazonaws.com		
		80:32400/TCP		16h	

Note

The values for `10.100.240.137` and `xxxxxxxx-xxxxxxxxxxxxxx` will be different than the example output (they will be unique to your load balancer) and `us-west-2` may be different for you, depending on which AWS Region that your cluster is in.

5. Open the [Amazon EC2 AWS Management Console](#). Select **Target Groups** (under **Load Balancing**) in the left navigation pane. In the **Name** column, select the target group's name where the value in the **Load balancer** column matches a portion of the name in the **EXTERNAL-IP** column of the output in the previous step. For example, you'd select the target group named `k8s-default-samplese-xxxxxxxxxx` if your output were the same as the output above. The **Target type** is **IP** because that was specified in the sample service manifest.
6. Select the **Target group** and then select the **Targets** tab. Under **Registered targets**, you should see three IP addresses of the three replicas deployed in a previous step. Wait until the status of all targets is **healthy** before continuing. It might take several minutes before all targets are healthy. The targets might be in an **unhealthy** state before changing to a **healthy** state.
7. Send traffic to the service replacing `xxxxxxxx-xxxxxxxxxxxxxx` and `us-west-2` with the values returned in the output for a [previous step \(p. 377\)](#) for **EXTERNAL-IP**. If you deployed to a private subnet, then you'll need to view the page from a device within your VPC, such as a bastion host. For more information, see [Linux Bastion Hosts on AWS](#).

```
curl k8s-default-samples-e-xxxxxxxxxx-xxxxxxxxxxxxxx.elb.region-code.amazonaws.com
```

Example output:

```
<!DOCTYPE html>
<html>
<head>
<title>Welcome to nginx!</title>
...

```

- When you're finished with the sample deployment, service, and namespace, remove them.

```
kubectl delete namespace nlb-sample-app
```

Application load balancing on Amazon EKS

When you create a Kubernetes `ingress`, an AWS Application Load Balancer (ALB) is provisioned that load balances application traffic. To learn more, see [What is an Application Load Balancer?](#) in the *Application Load Balancers User Guide* and [Ingress](#) in the Kubernetes documentation. ALBs can be used with pods that are deployed to nodes or to AWS Fargate. You can deploy an ALB to public or private subnets.

Application traffic is balanced at L7 of the OSI model. To load balance network traffic at L4, you deploy a Kubernetes service of the `LoadBalancer` type. This type provisions an AWS Network Load Balancer. For more information, see [Network load balancing on Amazon EKS \(p. 372\)](#). To learn more about the differences between the two types of load balancing, see [Elastic Load Balancing features](#) on the AWS website.

Prerequisites

Before you can load balance application traffic to an application, you must meet the following requirements.

- Have an existing cluster. If you don't have an existing cluster, see [Getting started with Amazon EKS \(p. 4\)](#). If you need to update the version of an existing cluster, see [Updating a cluster \(p. 34\)](#).
- Have the AWS Load Balancer Controller deployed on your cluster. For more information, see [Installing the AWS Load Balancer Controller add-on \(p. 329\)](#). We recommend version 2.4.1 or later for 1.19 or later clusters. If your cluster is earlier than 1.19, then we recommend using version 2.3.1.
- At least two subnets in different Availability Zones. The AWS load balancer controller chooses one subnet from each Availability Zone. When multiple tagged subnets are found in an Availability Zone, the controller chooses the subnet whose subnet ID comes first lexicographically. Each subnet must have at least eight available IP addresses.

If you're using multiple security groups attached to worker node, exactly one security group must be tagged as follows. Replace `cluster-name` with your cluster name.

- Key** – `kubernetes.io/cluster/cluster-name`
- Value** – shared or owned
- If you're using the AWS Load Balancer controller version v2.1.1 or earlier, subnets must be tagged in the format that follows. If you're using version 2.1.2 or later, tagging is optional. However, we recommend that you tag a subnet if any of the following is the case. You have multiple clusters that are running in the same VPC, or have multiple AWS services that share subnets in a VPC. Or, you want more control over where load balancers are provisioned for each cluster. Replace `cluster-name` with your cluster name.

- **Key** – `kubernetes.io/cluster/cluster-name`
- **Value** – shared or owned
- Your public and private subnets must meet the following requirements. This is unless you explicitly specify subnet IDs as an annotation on a service or ingress object. Assume that you provision load balancers by explicitly specifying subnet IDs as an annotation on a service or ingress object. In this situation, Kubernetes and the AWS load balancer controller use those subnets directly to create the load balancer and the following tags aren't required.
 - **Private subnets** – Must be tagged in the following format. This is so that Kubernetes and the AWS load balancer controller know that the subnets can be used for internal load balancers. If you use `eksctl` or an Amazon EKS AWS CloudFormation template to create your VPC after March 26, 2020, the subnets are tagged appropriately when created. For more information about the Amazon EKS AWS CloudFormation VPC templates, see [Creating a VPC for your Amazon EKS cluster \(p. 265\)](#).
 - **Key** – `kubernetes.io/role/internal-elb`
 - **Value** – 1
 - **Public subnets** – Must be tagged in the following format. This is so that Kubernetes knows to use only the subnets that were specified for external load balancers. This way, Kubernetes doesn't choose a public subnet in each Availability Zone (lexicographically based on their subnet ID). If you use `eksctl` or an Amazon EKS AWS CloudFormation template to create your VPC after March 26, 2020, the subnets are tagged appropriately when created. For more information about the Amazon EKS AWS CloudFormation VPC templates, see [Creating a VPC for your Amazon EKS cluster \(p. 265\)](#).
 - **Key** – `kubernetes.io/role/elb`
 - **Value** – 1

If the subnet role tags aren't explicitly added, the Kubernetes service controller examines the route table of your cluster VPC subnets. This is to determine if the subnet is private or public. We recommend that you don't rely on this behavior. Rather, explicitly add the private or public role tags. The AWS load balancer controller doesn't examine route tables. It also requires the private and public tags to be present for successful auto discovery.

Considerations

- The [AWS Load Balancer Controller](#) creates ALBs and the necessary supporting AWS resources whenever a Kubernetes ingress resource is created on the cluster with the `kubernetes.io/ingress.class: alb` annotation. The ingress resource configures the ALB to route HTTP or HTTPS traffic to different pods within the cluster. To ensure that your ingress objects use the AWS Load Balancer Controller, add the following annotation to your Kubernetes ingress specification. For more information, see [Ingress specification](#) on GitHub.

```
annotations:
  kubernetes.io/ingress.class: alb
```

Note

If you're load balancing to IPv6 pods, add the following annotation to your ingress spec. You can only load balance over IPv6 to IP targets, not instance targets. Without this annotation, load balancing is over IPv4.

```
alb.ingress.kubernetes.io/ip-address-type: dualstack
```

- The AWS Load Balancer Controller supports the following traffic modes:
 - **Instance** – Registers nodes within your cluster as targets for the ALB. Traffic reaching the ALB is routed to NodePort for your service and then proxied to your pods. This is the default traffic mode. You can also explicitly specify it with the `alb.ingress.kubernetes.io/target-type: instance` annotation.

Note

Your Kubernetes service must specify the `NodePort` or "LoadBalancer" type to use this traffic mode.

- **IP** – Registers pods as targets for the ALB. Traffic reaching the ALB is directly routed to pods for your service. You must specify the `alb.ingress.kubernetes.io/target-type: ip` annotation to use this traffic mode. The IP target type is required when target pods are running on Fargate.
- To tag ALBs created by the controller, add the following annotation to the controller: `alb.ingress.kubernetes.io/tags`. For a list of all available annotations supported by the AWS Load Balancer Controller, see [Ingress annotations](#) on GitHub.
- Upgrading or downgrading the ALB controller version can introduce breaking changes for features that rely on it. For more information about the breaking changes that are introduced in each release, see the [ALB controller release notes](#) on GitHub.

To share an application load balancer across multiple service resources using `IngressGroups`

To join an ingress to a group, add the following annotation to a Kubernetes ingress resource specification.

```
alb.ingress.kubernetes.io/group.name: my-group
```

The group name must:

- Be 63 or fewer characters in length.
- Consist of lower case letters, numbers, -, and .
- Start and end with a letter or number.

The controller automatically merges ingress rules for all ingresses in the same ingress group. It supports them with a single ALB. Most annotations that are defined on an ingress only apply to the paths defined by that ingress. By default, ingress resources don't belong to any ingress group.

Warning

Potential security risk: Specify an ingress group for an ingress only when all the Kubernetes users that have RBAC permission to create or modify ingress resources are within the same trust boundary. If you add the annotation with a group name, other Kubernetes users might create or modify their ingresses to belong to the same ingress group. Doing so can cause undesirable behavior, such as overwriting existing rules with higher priority rules.

You can add an order number of your ingress resource.

```
alb.ingress.kubernetes.io/group.order: '10'
```

The number can be 1-1000. The lowest number for all ingresses in the same ingress group is evaluated first. All ingresses without this annotation are evaluated with a value of zero. Duplicate rules with a higher number can overwrite rules with a lower number. By default, the rule order between ingresses within the same ingress group is determined lexicographically based namespace and name.

Important

Ensure that each ingress in the same ingress group has a unique priority number. You can't have duplicate order numbers across ingresses.

(Optional) Deploy a sample application

Prerequisites

- At least one public or private subnet in your cluster VPC.

- Have the AWS Load Balancer Controller deployed on your cluster. For more information, see [Installing the AWS Load Balancer Controller add-on \(p. 329\)](#). We recommend version 2.4.1 or later.

To deploy a sample application

You can run the sample application on a cluster that has Amazon EC2 nodes, Fargate pods, or both.

1. If you're not deploying to Fargate, skip this step. If you're deploying to Fargate, create a Fargate profile. You can create the profile by running the following command or in the [AWS Management Console \(p. 159\)](#) using the same values for name and namespace that are in the command. Replace the *example values* with your own.

```
eksctl create fargateprofile \
  --cluster my-cluster \
  --region region-code \
  --name alb-sample-app \
  --namespace game-2048
```

2. Deploy the game **2048** as a sample application to verify that the AWS Load Balancer Controller creates an AWS ALB as a result of the ingress object. Complete the steps for the type of subnet you're deploying to.
 - a. If you're deploying to pods in a cluster that you created with the IPv6 family, skip to the next step.

- **Public**

```
kubectl apply -f https://raw.githubusercontent.com/kubernetes-sigs/aws-load-balancer-controller/v2.4.1/docs/examples/2048/2048_full.yaml
```

- **Private**

1. Download the manifest.

```
curl -o 2048_full.yaml https://raw.githubusercontent.com/kubernetes-sigs/aws-load-balancer-controller/v2.4.1/docs/examples/2048/2048_full.yaml
```

2. Edit the file and find the line that says `alb.ingress.kubernetes.io/scheme: internet-facing`.
3. Change *internet-facing* to *internal* and save the file.
4. Apply the manifest to your cluster.

```
kubectl apply -f 2048_full.yaml
```

- b. If you're deploying to pods in a cluster that you created with the [IPv6 family \(p. 294\)](#), complete the following steps.

1. Download the manifest.

```
curl -o 2048_full.yaml https://raw.githubusercontent.com/kubernetes-sigs/aws-load-balancer-controller/v2.4.1/docs/examples/2048/2048_full.yaml
```

2. Open the file in an editor and add the following line to the annotations in the ingress spec.

```
alb.ingress.kubernetes.io/ip-address-type: dualstack
```

3. If you're load balancing to internal pods, rather than internet facing pods, change the line that says `alb.ingress.kubernetes.io/scheme: internet-facing` to `alb.ingress.kubernetes.io/scheme: internal`
4. Save the file.
5. Apply the manifest to your cluster.

```
kubectl apply -f 2048_full.yaml
```

3. After a few minutes, verify that the ingress resource was created with the following command.

```
kubectl get ingress/ingress-2048 -n game-2048
```

Output:

NAME	CLASS	HOSTS	ADDRESS
		PORTS	AGE
ingress-2048	<none>	*	k8s-game2048-ingress2-xxxxxxxxxx-yyyyyyyyyy.region-code.elb.amazonaws.com
		80	2m32s

Note

If you created the load balancer in a private subnet, the value under ADDRESS in the previous output is prefaced with `internal-`.

If your ingress wasn't successfully created after several minutes, run the following command to view the Load Balancer Controller logs. These logs might contain error messages that you can use to diagnose issues with your deployment.

```
kubectl logs -n kube-system deployment.apps/aws-load-balancer-controller
```

4. If you deployed to a public subnet, open a browser and navigate to the ADDRESS URL from the previous command output to see the sample application. If you don't see anything, refresh your browser and try again. If you deployed to a private subnet, then you'll need to view the page from a device within your VPC, such as a bastion host. For more information, see [Linux Bastion Hosts on AWS](#).
5. When you finish experimenting with your sample application, delete it by running one of the the following commands.
 - If you applied the manifest, rather than applying a copy that you downloaded, use the following command.

```
kubectl delete -f https://raw.githubusercontent.com/kubernetes-sigs/aws-load-balancer-controller/v2.4.1/docs/examples/2048/2048_full.yaml
```

- If you downloaded and edited the manifest, use the following command.

```
kubectl delete -f 2048_full.yaml
```

Restricting external IP addresses that can be assigned to services

Kubernetes services can be reached from inside of a cluster through:

- A cluster IP address that is assigned automatically by Kubernetes

- Any IP address that you specify for the `externalIPs` property in a service spec. External IP addresses are not managed by Kubernetes and are the responsibility of the cluster administrator. External IP addresses specified with `externalIPs` are different than the external IP address assigned to a service of type `LoadBalancer` by a cloud provider.

To learn more about Kubernetes services, see [Service](#) in the Kubernetes documentation. You can restrict the IP addresses that can be specified for `externalIPs` in a service spec.

To restrict the IP addresses that can be specified for `externalIPs` in a service spec

- Deploy cert-manager to manage webhook certificates. For more information, see the [cert-manager](#) documentation.

```
kubectl apply -f https://github.com/jetstack/cert-manager/releases/download/v1.5.4/cert-manager.yaml
```

- Verify that the cert-manager pods are running.

```
kubectl get pods -n cert-manager
```

Example output:

NAME	READY	STATUS	RESTARTS	AGE
cert-manager-58c8844bb8-nlx7q	1/1	Running	0	15s
cert-manager-cainjector-745768f6ff-696h5	1/1	Running	0	15s
cert-manager-webhook-67cc76975b-4v4nk	1/1	Running	0	14s

- Review your existing services to ensure that none of them have external IP addresses assigned to them that aren't contained within the CIDR block you want to limit addresses to.

```
kubectl get services -A
```

Example output:

NAMESPACE	EXTERNAL-IP	PORT(S)	AGE	TYPE
CLUSTER-IP				
cert-manager	<none>	cert-manager		ClusterIP
10.100.102.137		9402/TCP	20m	
cert-manager	<none>	cert-manager-webhook		ClusterIP
10.100.6.136		443/TCP	20m	
default		kubernetes		ClusterIP
10.100.0.1	<none>	443/TCP	2d1h	
externalip-validation-system		externalip-validation-webhook-service		ClusterIP
10.100.234.179		443/TCP	16s	
kube-system	<none>	kube-dns		ClusterIP
10.100.0.10		53/UDP, 53/TCP	2d1h	
my-namespace	<none>	my-service		ClusterIP
10.100.128.10	192.168.1.1	80/TCP	149m	

If any of the values are IP addresses that are not within the block you want to restrict access to, you'll need to change the addresses to be within the block, and redeploy the services. For example, the `my-service` service in the previous output has an external IP address assigned to it that isn't within the CIDR block example in step 5.

- Download the external IP webhook manifest. You can also view the [source code for the webhook](#) on GitHub.

```
curl -o externalip-webhook.yaml https://s3.us-west-2.amazonaws.com/amazon-eks/docs/externalip-webhook.yaml
```

5.

Specify CIDR blocks. Open the downloaded file in your editor and remove the # at the start of the following lines.

```
#args:  
#- --allowed-external-ip-cidrs=10.0.0.0/8
```

Replace 10.0.0.0/8 with your own CIDR block. You can specify as many blocks as you like. If specifying multiple blocks, add a comma between blocks.

6.

If your cluster is not in the us-west-2 AWS Region, then replace us-west-2, 602401143452, and amazonaws.com in the file with the following commands. Before running the commands, replace **region-code** and **111122223333** with the value for your AWS Region from the list in [Amazon container image registries \(p. 386\)](#).

```
sed -i.bak -e 's|602401143452|111122223333|' externalip-webhook.yaml  
sed -i.bak -e 's|us-west-2|region-code|' externalip-webhook.yaml  
sed -i.bak -e 's|amazonaws.com||' externalip-webhook.yaml
```

7. Apply the manifest to your cluster.

```
kubectl apply -f externalip-webhook.yaml
```

An attempt to deploy a service to your cluster with an IP address specified for **externalIPs** that is not contained in the blocks that you specified in the [Specify CIDR blocks \(p. 384\)](#) step will fail.

Copy a container image from one repository to another repository

This topic describes how to pull a container image from a repository that your nodes don't have access to and push the image to a repository that your nodes have access to. You can push the image to Amazon ECR or an alternative repository that your nodes have access to.

Prerequisites

- The Docker engine installed and configured on your computer. For instructions, see [Install Docker Engine](#) in the Docker documentation.
- 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 interface VPC endpoint for Amazon ECR if you want your nodes to pull container images from or push container images to a private Amazon ECR repository over Amazon's network. For more information, see [Create the VPC endpoints for Amazon ECR](#) in the Amazon Elastic Container Registry User Guide.

Complete the following steps to pull a container image from a repository and push it to your own repository. In the following examples that are provided in this topic, the image for the [CNI metrics helper \(p. 324\)](#) is pulled. When you follow these steps, make sure to replace the **example values** with your own values.

To copy a container image from one repository to another repository

1. If you don't already have an Amazon ECR repository or another repository, then create one that your nodes have access to. The following command creates an Amazon ECR private repository. An Amazon ECR private repository name must start with a letter. It can only contain lowercase letters, numbers, hyphens (-), underscores (_), and forward slashes (/). For more information, see [Creating a private repository](#) in the Amazon Elastic Container Registry User Guide.

You can replace `cni-metrics-helper` with whatever you choose. As a best practice, create a separate repository for each image. We recommend this because image tags must be unique within a repository. Replace `region-code` with an [AWS Region supported by Amazon ECR](#).

```
aws ecr create-repository --region region-code --repository-name cni-metrics-helper
```

2. Determine the registry, repository, and tag (optional) of the image that your nodes need to pull. This information is in the `registry/repository[:tag]` format.

Many of the Amazon EKS topics about installing images require that you apply a manifest file or install the image using a Helm chart. However, before you apply a manifest file or install a Helm chart, first view the contents of the manifest or chart's `values.yaml` file. That way, you can determine the registry, repository, and tag to pull.

For example, you can find the following line in the [manifest file](#) for the [CNI metrics helper \(p. 324\)](#). The registry is `602401143452.dkr.ecr.us-west-2.amazonaws.com`, which is an Amazon ECR private registry. The repository is `cni-metrics-helper`.

```
image: "602401143452.dkr.ecr.us-west-2.amazonaws.com/cni-metrics-helper:v1.11.0"
```

You may see the following variations for an image location:

- Only `repository-name:tag`. In this case, `docker.io` is usually the registry, but not specified since Kubernetes prepends it to a repository name by default if no registry is specified.
- `repository-name/repository-namespace/repository:tag`. A repository namespace is optional, but is sometimes specified by the repository owner for categorizing images. For example, all [Amazon EC2 images in the Amazon ECR Public Gallery](#) use the `aws-ec2` namespace.

Before installing an image with Helm, view the Helm `values.yaml` file to determine the image location. For example, the `values.yaml` file for the [CNI metrics helper \(p. 324\)](#) includes the following lines.

```
image:  
  region: us-west-2  
  tag: v1.11.0  
  account: "602401143452"  
  domain: "amazonaws.com"
```

3. Pull the container image specified in the manifest file.
 - a. If you're pulling from a public registry, such as the [Amazon ECR Public Gallery](#), you can skip to the next sub-step, because authentication isn't required. In this example, you authenticate to an Amazon ECR private registry that contains the repository for the CNI metrics helper image. Amazon EKS maintains the image in each registry listed in [Amazon container image registries \(p. 386\)](#). You can authenticate to any of the registries by replacing `602401143452` and `region-code` with the information for a different registry. A separate registry exists for each [AWS Region that Amazon EKS is supported in](#).

```
aws ecr get-login-password --region region-code | docker login --username AWS --password-stdin 602401143452.dkr.ecr.region-code.amazonaws.com
```

- b. Pull the image. In this example, you pull from the registry that you authenticated to in the previous sub-step. Replace *602401143452* and *region-code* with the information that you provided in the previous sub-step.

```
docker pull 602401143452.dkr.ecr.region-code.amazonaws.com/cni-metrics-helper:v1.11.0
```

4. Tag the image that you pulled with your registry, repository, and tag. The following example assumes that you pulled the image from the manifest file and are going to push it to the Amazon ECR private repository that you created in the first step. Replace *111122223333* with your account ID. Replace *region-code* with the AWS Region that you created your Amazon ECR private repository in.

```
docker tag cni-metrics-helper:v1.11.0 111122223333.dkr.ecr.region-code.amazonaws.com/cni-metrics-helper:v1.11.0
```

5. Authenticate to your registry. In this example, you authenticate to the Amazon ECR private registry that you created in the first step. For more information, see [Registry authentication](#) in the Amazon Elastic Container Registry User Guide.

```
aws ecr get-login-password --region region-code | docker login --username AWS --password-stdin 111122223333.dkr.ecr.region-code.amazonaws.com
```

6. Push the image to your repository. In this example, you push the image to the Amazon ECR private repository that you created in the first step. For more information, see [Pushing a Docker image](#) in the Amazon Elastic Container Registry User Guide.

```
docker push 111122223333.dkr.ecr.region-code.amazonaws.com/cni-metrics-helper:v1.11.0
```

7. Update the manifest file that you used to determine the image in a previous step with the `registry/repository:tag` for the image that you pushed. If you're installing with a Helm chart, there's often an option to specify the `registry/repository:tag`. When installing the chart, specify the `registry/repository:tag` for the image that you pushed to your repository.

Amazon container image registries

When you deploy add-ons such as the [Installing the AWS Load Balancer Controller add-on \(p. 329\)](#), the [Amazon VPC CNI \(p. 287\)](#), [kube-proxy \(p. 348\)](#), or [storage drivers \(p. 229\)](#) to your cluster, your nodes might pull the container image from an Amazon EKS Amazon ECR private repository. The image's registry, repository, and tag are specified in a manifest or Helm values.yaml file referenced in the topics for each add-on that you deploy.

Amazon EKS replicates the images to a repository in each Amazon EKS supported AWS Region. Your nodes can pull the container image over the internet from any of the following registries. Alternatively, your nodes can pull the image over Amazon's network if you created an [interface VPC endpoint for Amazon ECR \(AWS PrivateLink\)](#) in your VPC. The registries require authentication with an AWS IAM account. Your nodes authenticate using the [Amazon EKS node IAM role \(p. 474\)](#), which has the permissions in the [AmazonEC2ContainerRegistryReadOnly](#) managed IAM policy associated to it.

AWS Region	Registry
af-south-1	877085696533.dkr.ecr.af-south-1.amazonaws.com
ap-east-1	800184023465.dkr.ecr.ap-east-1.amazonaws.com
ap-northeast-1	602401143452.dkr.ecr.ap-northeast-1.amazonaws.com
ap-northeast-2	602401143452.dkr.ecr.ap-northeast-2.amazonaws.com
ap-northeast-3	602401143452.dkr.ecr.ap-northeast-3.amazonaws.com
ap-south-1	602401143452.dkr.ecr.ap-south-1.amazonaws.com
ap-southeast-1	602401143452.dkr.ecr.ap-southeast-1.amazonaws.com
ap-southeast-2	602401143452.dkr.ecr.ap-southeast-2.amazonaws.com
ap-southeast-3	296578399912.dkr.ecr.ap-southeast-3.amazonaws.com
ca-central-1	602401143452.dkr.ecr.ca-central-1.amazonaws.com
cn-north-1	918309763551.dkr.ecr.cn-north-1.amazonaws.com
cn-northwest-1	961992271922.dkr.ecr.cn-northwest-1.amazonaws.com
eu-central-1	602401143452.dkr.ecr.eu-central-1.amazonaws.com
eu-north-1	602401143452.dkr.ecr.eu-north-1.amazonaws.com
eu-south-1	590381155156.dkr.ecr.eu-south-1.amazonaws.com
eu-west-1	602401143452.dkr.ecr.eu-west-1.amazonaws.com
eu-west-2	602401143452.dkr.ecr.eu-west-2.amazonaws.com
eu-west-3	602401143452.dkr.ecr.eu-west-3.amazonaws.com
me-south-1	558608220178.dkr.ecr.me-south-1.amazonaws.com
sa-east-1	602401143452.dkr.ecr.sa-east-1.amazonaws.com
us-east-1	602401143452.dkr.ecr.us-east-1.amazonaws.com
us-east-2	602401143452.dkr.ecr.us-east-2.amazonaws.com

AWS Region	Registry
us-gov-east-1	151742754352.dkr.ecr.us-gov-east-1.amazonaws.com
us-gov-west-1	013241004608.dkr.ecr.us-gov-west-1.amazonaws.com
us-west-1	602401143452.dkr.ecr.us-west-1.amazonaws.com
us-west-2	602401143452.dkr.ecr.us-west-2.amazonaws.com

Amazon EKS add-ons

An add-on is software that provides supporting operational capabilities to Kubernetes applications, but is not specific to the application. This includes software like observability agents or Kubernetes drivers that allow the cluster to interact with underlying AWS resources for networking, compute, and storage. Add-on software is typically built and maintained by the Kubernetes community, cloud providers like AWS, or third-party vendors. Amazon EKS automatically installs self-managed add-ons such as the Amazon VPC CNI, `kube-proxy`, and CoreDNS for every cluster. You can change the default configuration of the add-ons and update them when desired.

Amazon EKS add-ons provide installation and management of a curated set of add-ons for Amazon EKS clusters. All Amazon EKS add-ons include the latest security patches, bug fixes, and are validated by AWS to work with Amazon EKS. Amazon EKS add-ons allow you to consistently ensure that your Amazon EKS clusters are secure and stable and reduce the amount of work that you need to do in order to install, configure, and update add-ons. If a self-managed add-on, such as `kube-proxy` is already running on your cluster and is available as an Amazon EKS add-on, then you can install the `kube-proxy` Amazon EKS add-on to start benefiting from the capabilities of Amazon EKS add-ons.

You can update specific Amazon EKS managed configuration fields for Amazon EKS add-ons through the Amazon EKS API. You can also modify configuration fields not managed by Amazon EKS directly within the Kubernetes cluster once the add-on starts. This includes defining specific configuration fields for an add-on where applicable. These changes are not overridden by Amazon EKS once they are made. This is made possible using the Kubernetes server-side apply feature. For more information, see [Amazon EKS add-on configuration \(p. 389\)](#).

Amazon EKS add-ons can be used with any 1.18 or later Amazon EKS cluster. The cluster can include self-managed and Amazon EKS managed node groups, and Fargate.

Considerations

- To configure add-ons for the cluster your IAM user must have IAM permissions to work with add-ons. For more information, see the actions with `Addon` in their name in [Actions defined by Amazon Elastic Kubernetes Service](#).
- Amazon EKS add-ons are only available with Amazon EKS clusters running Kubernetes version 1.18 and later.
- Amazon EKS add-ons run on the nodes that you provision or configure for your cluster. Node types include Amazon EC2 instances and Fargate.
- You can modify fields that aren't managed by Amazon EKS to customize the installation of an Amazon EKS add-on. For more information, see [Amazon EKS add-on configuration \(p. 389\)](#).
- If you create a cluster with the AWS Management Console, the Amazon EKS `kube-proxy`, Amazon VPC CNI, and CoreDNS Amazon EKS add-ons are automatically added to your cluster. If you use `eksctl` to create your cluster with a `config` file, `eksctl` can also create the cluster with Amazon EKS add-ons. If you create your cluster using `eksctl` without a `config` file or with any other tool, the self-managed `kube-proxy`, Amazon VPC CNI, and CoreDNS add-ons are installed, rather than the Amazon

EKS add-ons. You can either manage them yourself or add the Amazon EKS add-ons manually after cluster creation.

You can add, update, or delete Amazon EKS add-ons using the Amazon EKS API, AWS Management Console, AWS CLI, and `eksctl`. For detailed steps when using the AWS Management Console, AWS CLI, and `eksctl`, see the topics for the following add-ons:

- [Amazon VPC CNI \(p. 281\)](#)
- [CoreDNS \(p. 337\)](#)
- [kube-proxy \(p. 343\)](#)
- [ADOT \(p. 516\)](#)
- [Amazon EBS CSI \(p. 237\)](#)

You can also create Amazon EKS add-ons using [AWS CloudFormation](#).

Amazon EKS add-on configuration

Amazon EKS add-ons are installed to your cluster using standard, best practice configurations. For more information about Amazon EKS add-ons, see [Amazon EKS add-ons \(p. 388\)](#).

You may want to customize the configuration of an Amazon EKS add-on to enable advanced features. Amazon EKS uses the Kubernetes server-side apply feature to enable management of an add-on by Amazon EKS without overwriting your configuration for settings that aren't managed by Amazon EKS. For more information, see [Server-Side Apply](#) in the Kubernetes documentation. To achieve this, Amazon EKS manages a minimum set of fields for every add-on that it installs. You can modify all fields that aren't managed by Amazon EKS, or another Kubernetes control plane process such as `kube-controller-manager`, without issue.

Important

Modifying a field managed by Amazon EKS prevents Amazon EKS from managing the add-on and may result in your changes being overwritten when an add-on is updated.

Prerequisites

- An existing 1.18 or later Amazon EKS cluster.
- An Amazon EKS add-on added to the cluster. For more information about adding an Amazon EKS add-on to your cluster, see [Amazon EKS add-ons \(p. 388\)](#).

View field management status

You can use `kubectl` to see which fields are managed by Amazon EKS for any Amazon EKS add-on.

To see the management status of a field

1. Determine which add-on that you want to examine. To see all of the deployments and DaemonSets deployed to your cluster, see [View Kubernetes resources \(p. 506\)](#).
2. View the managed fields for an add-on by running the following command:

```
kubectl get type/add-on-name -n add-on-namespace -o yaml
```

For example, you can see the managed fields for the CoreDNS add-on with the following command.

```
kubectl get deployment/coredns -n kube-system -o yaml
```

Field management is listed in the following section in the returned output.

```
...  
managedFields:  
- apiVersion: apps/v1  
  fieldsType: FieldsV1  
  fieldsV1:  
...  
...
```

Note

If you don't see `managedFields` in the output, add `--show-managed-fields` to the command and run it again. The version of `kubectl` that you're using determines whether managed fields are returned by default.

Understanding field management syntax in the Kubernetes API

When you view details for a Kubernetes object, both managed and unmanaged fields are returned in the output. Managed fields can be either of the following types:

- **Fully managed** – All keys for the field are managed by Amazon EKS. Modifications to any value causes a conflict.
- **Partially managed** – Some keys for the field are managed by Amazon EKS. Only modifications to the keys explicitly managed by Amazon EKS cause a conflict.

Both types of fields are tagged with `manager: eks`.

Each key is either a `.` representing the field itself, which always maps to an empty set, or a string that represents a sub-field or item. The output for field management consists of the following types of declarations:

- `f:<name>`, where `<name>` is the name of a field in a list.
- `k:<keys>`, where `<keys>` is a map of a list item's fields.
- `v:<value>`, where `<value>` is the exact json formatted value of a list item.
- `i:<index>`, where `<index>` is position of an item in the list.

For more information, see [FieldsV1 v1 meta](#) in the Kubernetes documentation.

The following portions of output for the CoreDNS add-on illustrate the previous declarations:

- **Fully managed fields** – If a managed field has an `f:` (field) specified, but no `k:` (key), then the entire field is managed. Modifications to any values in this field cause a conflict.

In the following output, you can see that the container named `coredns` is managed by `eks`. The `args`, `image`, and `imagePullPolicy` sub-fields are also managed by `eks`. Modifications to any values in these fields cause a conflict.

```
...  
f:containers:  
k:{ "name": "coredns" }:  
.: {}  
f:args: {}  
f:image: {}  
f:imagePullPolicy: {}  
...  
manager: eks
```

...

- **Partially managed fields** – If a managed key has a value specified, the declared keys are managed for that field. Modifying the specified keys cause a conflict.

In the following output, you can see that eks manages the config-volume and tmp volumes set with the name key.

```
...
f:volumes:
  k:{ "name": "config-volume" }:
    .:
      f:configMap:
        f:items: {}
        f:name: {}
      f:name: {}
    k:{ "name": "tmp" }:
      .:
        f:name: {}
...
manager: eks
...
```

- **Adding keys to partially managed fields** – If only a specific key value is managed, you can safely add additional keys, such as arguments, to a field without causing a conflict. If you add additional keys, make sure that the field isn't managed first. Adding or modifying any value that is managed causes a conflict.

In the following output, you can see that both the name key and name field are managed. Adding or modifying any container name causes a conflict with this managed key.

```
...
f:containers:
  k:{ "name": "coredns" }:
    .:
      f:name: {}
...
manager: eks
...
```

Machine learning training using Elastic Fabric Adapter

This topic describes how to integrate Elastic Fabric Adapter (EFA) with pods deployed in your Amazon EKS cluster. Elastic Fabric Adapter (EFA) is a network interface for Amazon EC2 instances that enables you to run applications requiring high levels of inter-node communications at scale on AWS. Its custom-built operating system bypass hardware interface enhances the performance of inter-instance communications, which is critical to scaling these applications. With EFA, High Performance Computing (HPC) applications using the Message Passing Interface (MPI) and Machine Learning (ML) applications using NVIDIA Collective Communications Library (NCCL) can scale to thousands of CPUs or GPUs. As a result, you get the application performance of on-premises HPC clusters with the on-demand elasticity and flexibility of the AWS cloud. Integrating EFA with applications running on Amazon EKS clusters can reduce the time to complete large scale distributed training workloads without having to add additional instances to your cluster. For more information about EFA, [Elastic Fabric Adapter](#).

The EFA plugin described in this topic fully supports Amazon EC2 [P4d](#) instances, which represent the current state of the art in distributed machine learning in the cloud. Each [p4d.24xlarge](#) instance has

eight NVIDIA A100 GPUs, and 400 Gbps GPUDirectRDMA over EFA. GPUDirectRDMA enables you to have direct GPU-to-GPU communication across nodes with CPU bypass, increasing collective communication bandwidth and lowering latency. Amazon EKS and EFA integration with p4d instances provides a seamless method to take advantage of the highest performing Amazon EC2 computing instance for distributed machine learning training.

Prerequisites

- An existing 1.19 or later Amazon EKS cluster. If you don't have an existing cluster, use one of our [Getting started with Amazon EKS \(p. 4\)](#) guides to create one. Your cluster must be deployed in a VPC that has at least one private subnet with enough available IP addresses to deploy nodes in. The private subnet must have outbound internet access provided by an external device, such as a NAT gateway.

If you plan to use eksctl to create your node group, eksctl can also create a 1.19 cluster for you.

- 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.
- The kubectl command line tool installed on your computer or AWS CloudShell. The version must be the same, or up to two versions later than your cluster version. To install or upgrade kubectl, see [Installing kubectl \(p. 4\)](#).
- You must have the VPC CNI version 1.7.10 installed before launching worker nodes that support multiple Elastic Fabric Adapters, such as the p4d.24xlarge. For more information about updating your CNI version, see [Updating the Amazon VPC CNI self-managed add-on \(p. 287\)](#).

Create node group

The following procedure helps you create a node group with a p4d.24xlarge backed node group with EFA interfaces and GPUDirect RDMA, and run an example NVIDIA Collective Communications Library (NCCL) test for multi-node NCCL Performance using EFAs. The example can be used a template for distributed deep learning training on Amazon EKS using EFAs.

1. Determine which Availability Zones that Amazon EC2 instances that support EFA are available in for the region that your cluster is in
 - a. Determine which Amazon EC2 instance types that support EFA are available in the AWS Region that your cluster is in.

```
aws ec2 describe-instance-types \
--region region-code \
--filters Name=network-infoefa-supported,Values=true \
--query "InstanceTypes[*].[InstanceType]" \
--output text
```

The Availability Zone name is listed in the Location column of the output returned from the previous command.

2. Create a node group using either eksctl or the AWS CLI and AWS CloudFormation.
eksctl

Prerequisite

Version 0.97.0 or later of the eksctl command line tool installed on your computer or AWS CloudShell. To install or update eksctl, see [Installing eksctl \(p. 10\)](#).

- a. Copy the following contents to a file named *efa-cluster.yaml*. Replace the *example values* with your own. You can replace *p4d.24xlarge* with a different instance, but if you do, make sure that the values for availabilityZones are Availability Zones that were returned for the instance type in step 1.

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

metadata:
  name: my-efa-cluster
  region: region-code
  version: "1.19"

iam:
  withOIDC: true

availabilityZones: ["us-west-2a", "us-west-2c"]

managedNodeGroups:
  - name: my-efa-ng
    instanceType: p4d.24xlarge
    minSize: 1
    desiredCapacity: 2
    maxSize: 3
    availabilityZones: ["us-west-2a"]
    volumeSize: 300
    privateNetworking: true
    efaEnabled: true
```

- b. Create a managed node group in an existing cluster.

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

If you don't have an existing cluster, you can run the following command to create a cluster and the node group.

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

AWS CLI and AWS CloudFormation

There are several requirements for EFA networking, including creating an EFA specific security group, creating an Amazon EC2 [placement group](#), and creating a launch template that specifies one or more EFA interfaces, and includes EFA driver installation as part of Amazon EC2 user data. To learn more about EFA requirements, see [Get started with EFA and MPI](#) in the Amazon EC2 User Guide for Linux Instances. The following steps create all of this for you. Replace all *example values* with your own.

- a. Set a few variables used in later steps. Replace all of the *example values* with your own. Replace *my-cluster* with the name of your existing cluster. The value for `node_group_resources_name` is later used to create an AWS CloudFormation stack. The value for `node_group_name` is later used to create the node group in your cluster.

```
cluster_name="my-cluster"
cluster_region="region-code"
node_group_resources_name="my-efa-nodegroup-resources"
node_group_name="my-efa-nodegroup"
```

- b. Identify a private subnet in your VPC that is in the same Availability Zone as the instance type that you want to deploy is available in.

- i. Retrieve the version of your cluster and store it in a variable for use in a later step.

```
cluster_version=$(aws eks describe-cluster \
  --name $cluster_name \
  --query "cluster.version" \
```

```
--output text)
```

- ii. Retrieve the VPC ID that your cluster is in and store it in a variable for use in a later step.

```
vpc_id=$(aws eks describe-cluster \
--name $cluster_name \
--query "cluster.resourcesVpcConfig.vpcId" \
--output text)
```

- iii. Retrieve the ID of the control plane security group for your cluster and store it in a variable for use in a later step.

```
control_plane_security_group=$(aws eks describe-cluster \
--name $cluster_name \
--query "cluster.resourcesVpcConfig.clusterSecurityGroupId" \
--output text)
```

- iv. Get the list of subnet IDs in your VPC that are in an Availability Zone returned in step 1.

```
aws ec2 describe-subnets \
--filters "Name=vpc-id,Values=$vpc_id" "Name=availability-zone,Values=us-west-2a" \
--query 'Subnets[*].SubnetId' \
--output text
```

If no output is returned, try a different Availability Zone returned in step 1. If none of your subnets are in an Availability Zone returned in step 1, then you need to create a subnet in an Availability Zone returned in step 1. If you have no room in your VPC to create another subnet, then you may need to create a new cluster in a new VPC.

- v. Determine whether the subnet is a private subnet by checking the route table for the subnet.

```
aws ec2 describe-route-tables \
--filter Name=association.subnet-id,Values=subnet-0d403852a65210a29 \
--query "RouteTables[].Routes[].GatewayId" \
--output text
```

Example output:

```
local
```

If the output is `local igw-02adc64c1b72722e2`, then the subnet is a public subnet. You must select a private subnet in an Availability Zone returned in step 1. Once you've identified a private subnet, note its ID for use in a later step.

- vi. Set a variable with the private subnet ID from the previous step for use in later steps.

```
subnet_id=your-subnet-id
```

- c. Download the AWS CloudFormation template.

```
curl -o efa-p4d-managed-nodegroup.yaml https://raw.githubusercontent.com/aws-samples/aws-efa-eks/main/cloudformation/efa-p4d-managed-nodegroup.yaml
```

- d. Copy the following text to your computer. Replace `p4d.24xlarge` with an instance type from step 1. Replace `subnet-0d403852a65210a29` with the ID of the private subnet that you identified in step 2.b.v. Replace `path-to-downloaded-cfn-template` with the path to the `efa-p4d-managed-nodegroup.yaml` that you downloaded in the previous step.

Replace `your-public-key-name` with the name of your public key. Once you've made the replacements, run the modified command.

```
aws cloudformation create-stack \
--stack-name ${node_group_resources_name} \
--capabilities CAPABILITY_IAM \
--template-body file://path-to-downloaded-cfn-template \
--parameters \
  ParameterKey=ClusterName,ParameterValue=${cluster_name} \
  ParameterKey=ClusterControlPlaneSecurityGroup,ParameterValue=
${control_plane_security_group} \
  ParameterKey=VpcId,ParameterValue=${vpc_id} \
  ParameterKey=SubnetId,ParameterValue=${subnet_id} \
  ParameterKey=NodeGroupName,ParameterValue=${node_group_name} \
  ParameterKey=NodeImageIdSSMParam,ParameterValue=/aws/service/eks/optimized-
ami/${cluster_version}/amazon-linux-2-gpu/recommended/image_id \
  ParameterKey=KeyName,ParameterValue=your-public-key-name \
  ParameterKey=NodeInstanceType,ParameterValue=p4d.24xlarge
```

- e. Determine when the stack that you deployed in the previous step is deployed.

```
aws cloudformation wait stack-create-complete --stack-name
$node_group_resources_name
```

There is no output from the previous command, but your shell prompt doesn't return until the stack is created.

- f. Create your node group using the resources created by the AWS CloudFormation stack in the previous step.
 - i. Retrieve information from the deployed AWS CloudFormation stack and store it in variables.

```
node_instance_role=$(aws cloudformation describe-stacks \
--stack-name $node_group_resources_name \
--query='Stacks[].Outputs[?OutputKey==`NodeInstanceRole`].OutputValue' \
--output text)
launch_template=$(aws cloudformation describe-stacks \
--stack-name $node_group_resources_name \
--query='Stacks[].Outputs[?OutputKey==`LaunchTemplateID`].OutputValue' \
--output text)
```

- ii. Create a managed node group that uses the launch template and node IAM role that were created in the previous step.

```
aws eks create-nodegroup \
--cluster-name $cluster_name \
--nodegroup-name $node_group_name \
--node-role $node_instance_role \
--subnets $subnet_id \
--launch-template id=$launch_template,version=1
```

- iii. Confirm that the nodes were created.

```
aws eks describe-nodegroup \
--cluster-name ${cluster_name} \
--nodegroup-name ${node_group_name} | jq -r .nodegroup.status
```

Don't continue until the status returned from the previous command is ACTIVE. It can take several minutes for the nodes to become ready.

- g. Deploy the EFA Kubernetes device plugin.

The EFA Kubernetes device plugin detects and advertises EFA interfaces as allocatable resources to Kubernetes. An application can consume the extended resource type `vpc.amazonaws.com/efa` in a pod request spec just like CPU and memory. For more information, see [Consuming extended resources](#) in the Kubernetes documentation. Once requested, the plugin automatically assigns and mounts an EFA interface to the pod. Using the device plugin simplifies EFA setup and does not require a pod to run in privileged mode.

```
kubectl apply -f https://raw.githubusercontent.com/aws-samples/aws-efa-eks/main/manifest/efa-k8s-device-plugin.yaml
```

- h. If you deployed an instance type with a GPU, deploy the NVIDIA Kubernetes device plugin.

```
kubectl apply -f https://raw.githubusercontent.com/NVIDIA/k8s-device-plugin/v0.9.0/nvidia-device-plugin.yaml
```

(Optional) Deploy a sample EFA compatible application

Deploy the Kubeflow MPI Operator

For the NCCL tests you can apply the Kubeflow MPI Operator. The MPI Operator makes it easy to run Allreduce-style distributed training on Kubernetes. For more information, see [MPI Operator](#) on GitHub.

```
kubectl apply -f https://raw.githubusercontent.com/kubeflow/mpi-operator/master/deploy/v1alpha2/mpi-operator.yaml
```

Run the multi-node NCCL Performance Test to verify GPUDirectRDMA/EFA

To verify NCCL Performance with GPUDirectRDMA over EFA, run the standard NCCL Performance test. For more information, see the official [NCCL-Tests](#) repo on GitHub. You can use the sample [Dockerfile](#) that comes with this test already built for both CUDA 11.2 and the latest version of EFA.

Alternately, you can download an AWS Docker image available from an [Amazon ECR repo](#).

Important

An important consideration required for adopting EFA with Kubernetes is configuring and managing Huge Pages as a resource in the cluster. For more information, see [Manage Huge Pages](#) in the Kubernetes documentation. Amazon EC2 instances with the EFA driver installed pre-allocate 5128 2M Huge Pages, which you can request as resources to consume in your job specifications.

Complete the following steps to run a two node NCCL Performance Test. In the example NCCL test job, each worker requests eight GPUs, 5210Mi of hugepages-2Mi, four EFAs, and 8000Mi of memory, which effectively means each worker consumes all the resources of a p4d.24xlarge instance.

1. Create the NCCL-tests job.

```
kubectl apply -f https://raw.githubusercontent.com/aws-samples/aws-efa-eks/main/examples/simple/nccl-efa-tests.yaml
```

Example output:

```
mpijob.kubeflow.org/nccl-tests-efa created
```

2. View your running pods.

```
kubectl get pods
```

Example output:

NAME	READY	STATUS	RESTARTS	AGE
nccl-tests-efa-launcher- nbql9	0/1	Init:0/1	0	2m49s
nccl-tests-efa-worker-0	1/1	Running	0	2m49s
nccl-tests-efa-worker-1	1/1	Running	0	2m49s

The MPI Operator creates a launcher pod and 2 worker pods (one on each node).

- View the log for the efa-launcher pod. Replace **wzr8j** with the value from your output.

```
kubectl logs -f nccl-tests-efa-launcher-nbql9
```

For more examples, see the Amazon EKS [EFA samples](#) repository on GitHub.

Machine learning inference using AWS Inferentia

This topic describes how to create an Amazon EKS cluster with nodes running [Amazon EC2 Inf1](#) instances and (optionally) deploy a sample application. Amazon EC2 Inf1 instances are powered by [AWS Inferentia](#) chips, which are custom built by AWS to provide high performance and lowest cost inference in the cloud. Machine learning models are deployed to containers using [AWS Neuron](#), a specialized software development kit (SDK) consisting of a compiler, runtime, and profiling tools that optimize the machine learning inference performance of Inferentia chips. AWS Neuron supports popular machine learning frameworks such as TensorFlow, PyTorch, and MXNet.

Note

Neuron device logical IDs must be contiguous. If a pod requesting multiple Neuron devices is scheduled on an `inf1.6xlarge` or `inf1.24xlarge` instance type (which have more than one Neuron device), that pod will fail to start if the Kubernetes scheduler selects non-contiguous device IDs. For more information, see [Device logical IDs must be contiguous](#) on GitHub.

Prerequisites

- Have `eksctl` installed on your computer. If you don't have it installed, see [Installing eksctl \(p. 10\)](#) for installation instructions.
- Have `kubectl` installed on your computer. For more information, see [Installing kubectl \(p. 4\)](#).
- (Optional) Have `python3` installed on your computer. If you don't have it installed, then see [Python downloads](#) for installation instructions.

Create a cluster

To create a cluster with Inf1 Amazon EC2 instance nodes

- Create a cluster with Inf1 Amazon EC2 instance nodes. You can replace `inf1.2xlarge` with any [Inf1 instance type](#). The `eksctl` utility detects that you are launching a node group with an `Inf1` instance type and will start your nodes using one of the [Amazon EKS optimized accelerated Amazon Linux AMI \(p. 194\)](#).

Note

You can't use [IAM roles for service accounts \(p. 442\)](#) with TensorFlow Serving.

```
eksctl create cluster \
--name inferentia \
--region region-code \
--nodegroup-name ng-infl \
--node-type inf1.2xlarge \
--nodes 2 \
--nodes-min 1 \
--nodes-max 4 \
--ssh-access \
--ssh-public-key your-key \
--with-oidc
```

Note

Note the value of the following line of the output. It's used in a later (optional) step.

```
[9] adding identity "arn:aws:iam::111122223333:role/
eksctl-inferentia-nodegroup-ng-in-NodeInstanceRole-FI7HIYS3BS09" to auth
ConfigMap
```

When launching a node group with Inf1 instances, eksctl automatically installs the AWS Neuron Kubernetes device plugin. This plugin advertises Neuron devices as a system resource to the Kubernetes scheduler, which can be requested by a container. In addition to the default Amazon EKS node IAM policies, the Amazon S3 read only access policy is added so that the sample application, covered in a later step, can load a trained model from Amazon S3.

2. Make sure that all pods have started correctly.

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

Abbreviated output:

NAME	READY	STATUS	RESTARTS	AGE
...				
neuron-device-plugin-daemonset- 6djhfp	1/1	Running	0	5m
neuron-device-plugin-daemonset- hwjsj	1/1	Running	0	5m

(Optional) Deploy a TensorFlow Serving application image

A trained model must be compiled to an Inferentia target before it can be deployed on Inferentia instances. To continue, you will need a [Neuron optimized TensorFlow](#) model saved in Amazon S3. If you don't already have a SavedModel, please follow the tutorial for [creating a Neuron compatible ResNet50 model](#) and upload the resulting SavedModel to S3. ResNet-50 is a popular machine learning model used for image recognition tasks. For more information about compiling Neuron models, see [The AWS Inferentia Chip With DLAMI](#) in the AWS Deep Learning AMI Developer Guide.

The sample deployment manifest manages a pre-built inference serving container for TensorFlow provided by AWS Deep Learning Containers. Inside the container is the AWS Neuron Runtime and the TensorFlow Serving application. A complete list of pre-built Deep Learning Containers optimized for Neuron is maintained on GitHub under [Available Images](#). At start-up, the DLC will fetch your model from Amazon S3, launch Neuron TensorFlow Serving with the saved model, and wait for prediction requests.

The number of Neuron devices allocated to your serving application can be adjusted by changing the `aws.amazon.com/neuron` resource in the deployment yaml. Please note that communication between

TensorFlow Serving and the Neuron runtime happens over GRPC, which requires passing the `IPC_LOCK` capability to the container.

1. Add the `AmazonS3ReadOnlyAccess` IAM policy to the node instance role that was created in step 1 of [Create a cluster \(p. 397\)](#). This is necessary so that the sample application can load a trained model from Amazon S3.

```
aws iam attach-role-policy \
    --policy-arn arn:aws:iam::aws:policy/AmazonS3ReadOnlyAccess \
    --role-name eksctl-inferentia-nodegroup-ng-in-NodeInstanceRole-FI7HIYS3BS09
```

2. Create a file named `rn50_deployment.yaml` with the contents below. Update the region-code and model path to match your desired settings. The model name is for identification purposes when a client makes a request to the TensorFlow server. This example uses a model name to match a sample ResNet50 client script that will be used in a later step for sending prediction requests.

```
aws ecr list-images --repository-name neuron-rtd --registry-id 790709498068 --
region us-west-2
```

```
kind: Deployment
apiVersion: apps/v1
metadata:
  name: eks-neuron-test
  labels:
    app: eks-neuron-test
    role: master
spec:
  replicas: 2
  selector:
    matchLabels:
      app: eks-neuron-test
      role: master
  template:
    metadata:
      labels:
        app: eks-neuron-test
        role: master
    spec:
      containers:
        - name: eks-neuron-test
          image: 763104351884.dkr.ecr.us-east-1.amazonaws.com/tensorflow-inference-
neuron:1.15.4-neuron-py37-ubuntu18.04
          command:
            - /usr/local/bin/entrypoint.sh
          args:
            - --port=8500
            - --rest_api_port=9000
            - --model_name=resnet50_neuron
            - --model_base_path=s3://<your-bucket-of-models>/resnet50_neuron/
          ports:
            - containerPort: 8500
            - containerPort: 9000
      imagePullPolicy: IfNotPresent
  env:
    - name: AWS_REGION
      value: "us-east-1"
    - name: S3_USE_HTTPS
      value: "1"
    - name: S3_VERIFY_SSL
      value: "0"
    - name: S3_ENDPOINT
      value: s3.us-east-1.amazonaws.com
```

```
- name: AWS_LOG_LEVEL
  value: "3"
resources:
limits:
  cpu: 4
  memory: 4Gi
  aws.amazon.com/neuron: 1
requests:
  cpu: "1"
  memory: 1Gi
securityContext:
capabilities:
  add:
    - IPC_LOCK
```

3. Deploy the model.

```
kubectl apply -f rn50_deployment.yaml
```

4. Create a file named `rn50_service.yaml` with the following contents. The HTTP and gRPC ports are opened for accepting prediction requests.

```
kind: Service
apiVersion: v1
metadata:
  name: <eks-neuron-test>
  labels:
    app: <eks-neuron-test>
spec:
  type: ClusterIP
  ports:
    - name: http-tf-serving
      port: 8500
      targetPort: 8500
    - name: grpc-tf-serving
      port: 9000
      targetPort: 9000
  selector:
    app: <eks-neuron-test>
    role: master
```

5. Create a Kubernetes service for your TensorFlow model Serving application.

```
kubectl apply -f rn50_service.yaml
```

(Optional) Make predictions against your TensorFlow Serving service

1. To test locally, forward the gRPC port to the `eks-neuron-test` service.

```
kubectl port-forward service/eks-neuron-test 8500:8500 &
```

2. Create a Python script called `tensorflow-model-server-infer.py` with the following content. This script runs inference via gRPC, which is service framework.

```
import numpy as np
import grpc
import tensorflow as tf
```

```
from tensorflow.keras.preprocessing import image
from tensorflow.keras.applications.resnet50 import preprocess_input
from tensorflow_serving.apis import predict_pb2
from tensorflow_serving.apis import prediction_service_pb2_grpc
from tensorflow.keras.applications.resnet50 import decode_predictions

if __name__ == '__main__':
    channel = grpc.insecure_channel('localhost:8500')
    stub = prediction_service_pb2_grpc.PredictionServiceStub(channel)
    img_file = tf.keras.utils.get_file(
        "./kitten_small.jpg",
        "https://raw.githubusercontent.com/awslabs/mxnet-model-server/master/docs/images/kitten_small.jpg")
    img = image.load_img(img_file, target_size=(224, 224))
    img_array = preprocess_input(image.img_to_array(img)[None, ...])
    request = predict_pb2.PredictRequest()
    request.model_spec.name = 'resnet50_inf1'
    request.inputs['input'].CopyFrom(
        tf.make_tensor_proto(img_array, shape=img_array.shape))
    result = stub.Predict(request)
    prediction = tf.make_ndarray(result.outputs['output'])
    print(decode_predictions(prediction))
```

3. Run the script to submit predictions to your service.

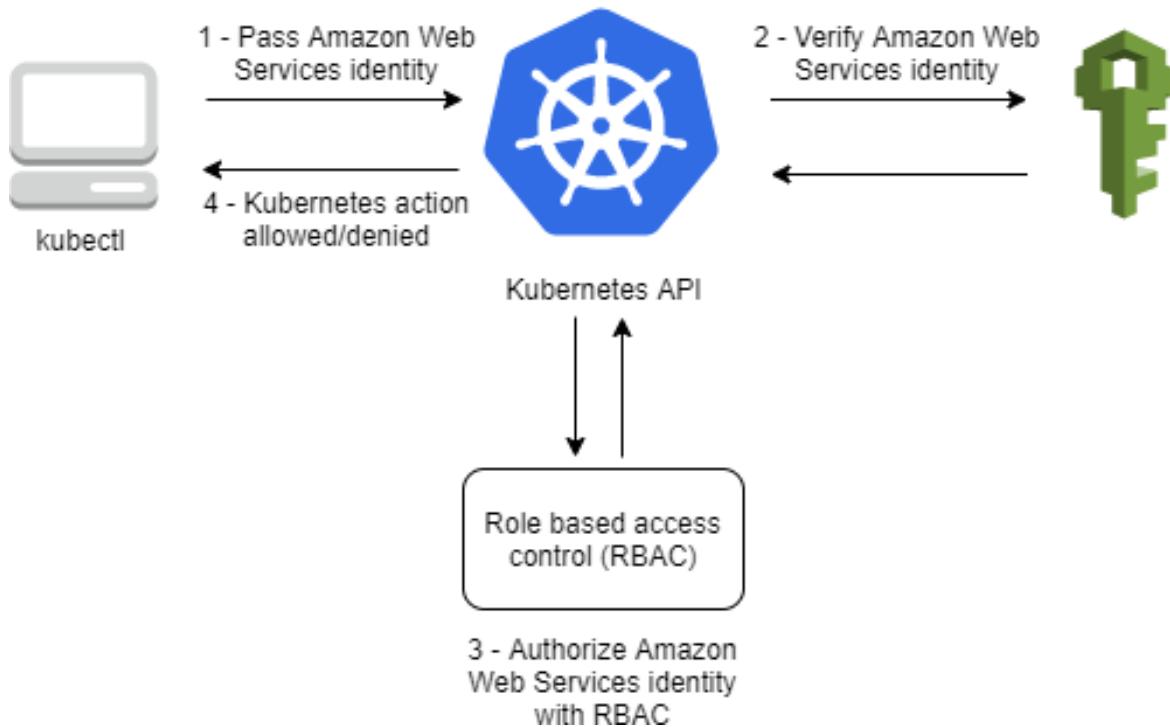
```
python3 tensorflow-model-server-infer.py
```

Example output:

```
[[(u'n02123045', u'tabby', 0.68817204), (u'n02127052', u'lynx', 0.12701613),
 (u'n02123159', u'tiger_cat', 0.08736559), (u'n02124075', u'Egyptian_cat',
 0.063844085), (u'n02128757', u'snow_leopard', 0.009240591)]]
```

Cluster authentication

Amazon EKS uses IAM to provide authentication to your Kubernetes cluster (through the `aws eks get-token` command, available in version 1.16.156 or later of the AWS CLI, or the [AWS IAM Authenticator for Kubernetes](#)), but it still relies on native Kubernetes [Role Based Access Control](#) (RBAC) for authorization. This means that IAM is only used for authentication of valid IAM entities. All permissions for interacting with your Amazon EKS cluster's Kubernetes API is managed through the native Kubernetes RBAC system. The following picture shows this relationship.



Note

Amazon EKS uses the authentication token to make the `sts:GetCallerIdentity` call. As a result, AWS CloudTrail events with the name `GetCallerIdentity` from the source `sts.amazonaws.com` can have Amazon EKS service IP addresses for their source IP address.

Topics

- [Enabling IAM user and role access to your cluster \(p. 402\)](#)
- [Authenticating users for your cluster from an OpenID Connect identity provider \(p. 409\)](#)
- [Create a kubeconfig for Amazon EKS \(p. 413\)](#)
- [Installing aws-iam-authenticator \(p. 417\)](#)
- [Default Amazon EKS Kubernetes roles and users \(p. 420\)](#)

Enabling IAM user and role access to your cluster

Access to your cluster using AWS Identity and Access Management (IAM); entities is enabled by the [AWS IAM Authenticator for Kubernetes](#), which runs on the Amazon EKS control plane. The authenticator gets its configuration information from the `aws-auth ConfigMap`. For all `aws-auth ConfigMap` settings, see [Full Configuration Format](#) on GitHub.

Add IAM users or roles to your Amazon EKS cluster

When you create an Amazon EKS cluster, the AWS Identity and Access Management (IAM) entity user or role, such as a [federated user](#) that creates the cluster, is automatically granted `system:masters` permissions in the cluster's role-based access control (RBAC) configuration in the Amazon EKS control plane. This IAM entity doesn't appear in any visible configuration, so make sure to keep track of which IAM entity originally created the cluster. To grant additional AWS users or roles the ability to interact with your cluster, you must edit the `aws-auth ConfigMap` within Kubernetes and create a Kubernetes `rolebinding` or `clusterrolebinding` with the name of a group that you specify in the `aws-auth ConfigMap`.

Note

For more information about different IAM identities, see [Identities \(Users, Groups, and Roles\)](#) in the *IAM User Guide*. For more information on Kubernetes role-based access control (RBAC) configuration, see [Using RBAC Authorization](#).

To add an IAM user or role to an Amazon EKS cluster

1. Determine which credentials `kubectl` is using to access your cluster. On your computer, you can see which credentials `kubectl` uses with the following command. Replace `~/.kube/config` with the path to your `kubeconfig` file if you don't use the default path.

```
cat ~/.kube/config
```

Example output:

```
...
contexts:
- context:
    cluster: my-cluster.region-code.eksctl.io
    user: admin@my-cluster.region-code.eksctl.io
    name: admin@my-cluster.region-code.eksctl.io
current-context: admin@my-cluster.region-code.eksctl.io
...
```

In the previous example output the credentials for a user named `admin` are configured for a cluster named `my-cluster`. If this is the user that created the cluster, then it already has access to your cluster. If it's not the user that created the cluster, then you need to complete the remaining steps to enable cluster access for other users. You can see which other roles or users currently have access to your cluster with the following command:

```
kubectl edit -n kube-system configmap/aws-auth
```

Example output:

```
Name: aws-auth
Namespace: kube-system
Labels: <none>
Annotations: <none>

Data
=====
mapRoles:
-----
- groups:
  - system:bootstrappers
  - system:nodes
  rolearn: arn:aws:iam::111122223333:role/my-node-role
```

```
username: system:node:{EC2PrivateDNSName}

BinaryData
====

Events: <none>
```

The previous example is a default aws-auth ConfigMap. Only the node instance role has access to the cluster.

2. Make sure that you have existing Kubernetes `roles` and `rolebindings` or `clusterroles` and `clusterrolebindings` that you can map IAM users or roles to. For more information about these resources, see [Using RBAC Authorization](#) in the Kubernetes documentation.
 1. View your existing Kubernetes `roles` or `clusterroles`. Roles are scoped to a namespace, but `clusterroles` are scoped to the cluster.

```
kubectl get roles -A
```

```
kubectl get clusterroles
```

2. View the details of any `role` or `clusterrole` returned in the previous output and confirm that it has the permissions (`rules`) that you want your IAM users to have in your cluster.

Replace `role-name` with a role name returned in the output from the previous command. Replace `kube-system` with the namespace of the role.

```
kubectl describe role role-name -n kube-system
```

Replace `cluster-role-name` with a `clusterrole` name returned in the output from the previous command.

```
kubectl describe clusterrole cluster-role-name
```

3. View your existing Kubernetes `rolebindings` or `clusterrolebindings`. `Rolebindings` are scoped to a namespace, but `clusterrolebindings` are scoped to the cluster.

```
kubectl get rolebindings -A
```

```
kubectl get clusterrolebindings
```

4. View the details of any `rolebinding` or `clusterrolebinding` and confirm that it has a `role` or `clusterrole` from the previous step listed as a `roleRef` and a group name listed for `subjects`.

Replace `role-binding-name` with a `rolebinding` name returned in the output from the previous command. Replace `kube-system` with the namespace of the `rolebinding`.

```
kubectl describe role role-binding-name -n kube-system
```

Example output:

```
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
```

```

name: eks-console-dashboard-restricted-access-role-binding
namespace: default
subjects:
- kind: Group
  name: eks-console-dashboard-restricted-access-group
  apiGroup: rbac.authorization.k8s.io
roleRef:
  kind: Role
  name: eks-console-dashboard-restricted-access-role
  apiGroup: rbac.authorization.k8s.io

```

Replace `cluster-role-binding-name` with a `clusterrolebinding` name returned in the output from the previous command.

```
kubectl describe clusterrole cluster-role-binding-name
```

Example output:

```

apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: eks-console-dashboard-full-access-binding
subjects:
- kind: Group
  name: eks-console-dashboard-full-access-group
  apiGroup: rbac.authorization.k8s.io
roleRef:
  kind: ClusterRole
  name: eks-console-dashboard-full-access-clusterrole
  apiGroup: rbac.authorization.k8s.io

```

3. Edit the `aws-auth` ConfigMap. You can use a tool such as `eksctl` to update the ConfigMap or you can update it manually by editing it.

Important

We recommend using `eksctl`, or another tool, to edit the ConfigMap. For information about other tools you can use, see [Use tools to make changes to the `aws-authConfigMap`](#) in the Amazon EKS best practices guides. An improperly formatted `aws-auth` ConfigMap can cause you to lose access to your cluster.

`eksctl`

Prerequisite

Version 0.97.0 or later of the `eksctl` command line tool installed on your computer or AWS CloudShell. To install or update `eksctl`, see [Installing `eksctl` \(p. 10\)](#).

1. View the current mappings in the ConfigMap. Replace `my-cluster` with the name of your cluster. Replace `region-code` with the AWS Region that your cluster is in.

```
eksctl get iamidentitymapping --cluster my-cluster --region=region-code
```

Example output:

ARN	USERNAME	GROUPS
ACCOUNT		

```
arn:aws:iam::111122223333:role/eksctl-my-cluster-my-nodegroup-
NodeInstanceRole-1XLS7754U3ZPA      system:node:{{EC2PrivateDNSName}}
system:bootstrappers,system:nodes
```

2. Add a mapping for a role. Replace *my-role* with your role name. Replace *eks-console-dashboard-full-access-group* with the name of the group specified in your Kubernetes rolebinding or clusterrolebinding. Replace *111122223333* with your account ID.

```
eksctl create iamidentitymapping \
--cluster my-cluster \
--region=region-code \
--arn arn:aws:iam::111122223333:role/my-role \
--group eks-console-dashboard-full-access-group \
--no-duplicate-arns
```

Important

The role ARN can't include a path such as *role/my-team/developers/my-role*. The format of the ARN must be *arn:aws:iam::111122223333:role/my-role*. In this example, *my-team/developers/* needs to be removed.

Example output:

```
...
2022-05-09 14:51:20 [#] adding identity "arn:aws:iam::111122223333:role/my-role"
to auth ConfigMap
```

3. Add a mapping for a user. Replace *my-user* with your user name. Replace *eks-console-dashboard-restricted-access-group* with the name of the group specified in your Kubernetes rolebinding or clusterrolebinding. Replace *111122223333* with your account ID.

```
eksctl create iamidentitymapping \
--cluster my-cluster \
--region=region-code \
--arn arn:aws:iam::111122223333:user/my-user \
--group eks-console-dashboard-restricted-access-group \
--no-duplicate-arns
```

Example output:

```
...
2022-05-09 14:53:48 [#] adding identity "arn:aws:iam::111122223333:user/my-user"
to auth ConfigMap
```

4. View the mappings in the ConfigMap again.

```
eksctl get iamidentitymapping --cluster my-cluster --region=region-code
```

Example output:

ARN	USERNAME	ACCOUNT	GROUPS
arn:aws:iam::111122223333:role/eksctl-my-cluster-my-nodegroup- NodeInstanceRole-1XLS7754U3ZPA		system:node:{{EC2PrivateDNSName}}	system:bootstrappers,system:nodes

```

arn:aws:iam::111122223333:role/my-role           eks-console-dashboard-
full-access-group
arn:aws:iam::111122223333:user/my-user           eks-console-dashboard-
restricted-access-group

```

Edit ConfigMap manually

1. Open the ConfigMap for editing.

```
kubectl edit -n kube-system configmap/aws-auth
```

Note

If you receive an error stating "Error from server (NotFound): configmaps "aws-auth" not found", then use the procedure in [Apply the aws-authConfigMap to your cluster \(p. 408\)](#) to apply the stock ConfigMap.

2. Add your IAM users, roles, or AWS accounts to the ConfigMap. You can't add IAM groups to the ConfigMap.

- **To add an IAM role (for example, for federated users):** Add the role details to the mapRoles section of the ConfigMap, under data. Add this section if it does not already exist in the file. Each entry supports the following parameters:
 - **rolearn:** The ARN of the IAM role to add. This value can't include a path. For example, you can't specify an ARN such as `arn:aws:iam::111122223333:role/my-team/developers/role-name`. The ARN needs to be `arn:aws:iam::111122223333:role/role-name` instead.
 - **username:** The user name within Kubernetes to map to the IAM role.
 - **groups:** The group or list of Kubernetes groups to map the role to. The group can be a default group, or a group specified in a clusterrolebinding or rolebinding. For more information, see [Default Roles and Role Bindings](#) in the Kubernetes documentation.
- **To add an IAM user:** Add the user details to the mapUsers section of the ConfigMap, under data. Add this section if it does not already exist in the file. Each entry supports the following parameters:
 - **userarn:** The ARN of the IAM user to add.
 - **username:** The user name within Kubernetes to map to the IAM user.
 - **groups:** The group, or list of Kubernetes groups to map the user to. The group can be a default group, or a group specified in a clusterrolebinding or rolebinding. For more information, see [Default Roles and Role Bindings](#) in the Kubernetes documentation.

For example, the YAML block below contains:

- A mapRoles section that maps the IAM node instance to Kubernetes groups so that nodes can register themselves with the cluster and the my-console-viewer-role IAM role that is mapped to a Kubernetes group that can view all Kubernetes resources for all clusters. For a list of the IAM and Kubernetes group permissions required for the my-console-viewer-role IAM role, see [Required permissions \(p. 506\)](#).
- A mapUsers section that maps the admin IAM user from the default AWS account to the system:masters Kubernetes group and the my-user user from a different AWS account that is mapped to a Kubernetes group that can view Kubernetes resources for a specific namespace. For a list of the IAM and Kubernetes group permissions required for the my-user IAM user, see [Required permissions \(p. 506\)](#).

Add or remove lines as necessary and replace all *example-values* with your own values.

```
# Please edit the object below. Lines beginning with a '#' will be ignored,  
# and an empty file will abort the edit. If an error occurs while saving this  
file will be  
# reopened with the relevant failures.  
#  
apiVersion: v1  
data:  
mapRoles: |  
  - groups:  
    - system:bootstrappers  
    - system:nodes  
      rolearn: arn:aws:iam::111122223333:role/my-role  
      username: system:node:{EC2PrivateDNSName}  
    - groups:  
      - eks-console-dashboard-full-access-group  
      rolearn: arn:aws:iam::111122223333:role/my-console-viewer-role  
      username: my-console-viewer-role  
mapUsers: |  
  - groups:  
    - system:masters  
      userarn: arn:aws:iam::111122223333:user/admin  
      username: admin  
    - groups:  
      - eks-console-dashboard-restricted-access-group  
      userarn: arn:aws:iam::444455556666:user/my-user  
      username: my-user
```

3. Save the file and exit your text editor.

Apply the aws-authConfigMap to your cluster

The aws-auth ConfigMap is automatically created and applied to your cluster when you create a managed node group or when you create a node group using eksctl. It is initially created to allow nodes to join your cluster, but you also use this ConfigMap to add role-based access control (RBAC) access to IAM users and roles. If you have not launched self-managed nodes and applied the aws-auth ConfigMap to your cluster, you can do so with the following procedure.

To apply the aws-authConfigMap to your cluster

1. Check to see if you have already applied the aws-auth ConfigMap.

```
kubectl describe configmap -n kube-system aws-auth
```

If you receive an error stating "Error from server (NotFound): configmaps "aws-auth" not found", then proceed with the following steps to apply the stock ConfigMap.

2. Download, edit, and apply the AWS authenticator configuration map.
 - a. Download the configuration map.
- ```
curl -o aws-auth-cm.yaml https://s3.us-west-2.amazonaws.com/amazon-eks/
cloudformation/2020-10-29/aws-auth-cm.yaml
```
- b. Open the file with a text editor. Replace <ARN of instance role (not instance profile)> with the Amazon Resource Name (ARN) of the IAM role associated with your nodes, and save the file. Do not modify any other lines in this file.

### Important

The role ARN can't include a path such as `role/my-team/developers/my-role`. The format of the ARN must be `arn:aws:iam::111122223333:role/my-role`. In this example, `my-team/developers/` needs to be removed.

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

You can inspect the AWS CloudFormation stack outputs for your node groups and look for the following values:

- **InstanceRoleARN** – For node groups that were created with `eksctl`
  - **NodeInstanceRole** – For node groups that were created with Amazon EKS vended AWS CloudFormation templates in the AWS Management Console
- c. Apply the configuration. This command may take a few minutes to finish.

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

### Note

If you receive any authorization or resource type errors, see [Unauthorized or access denied \(kubectl\) \(p. 547\)](#) in the troubleshooting section.

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

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

## Authenticating users for your cluster from an OpenID Connect identity provider

Amazon EKS supports using OpenID Connect (OIDC) identity providers as a method to authenticate users to your cluster. OIDC identity providers can be used with, or as an alternative to AWS Identity and Access Management (IAM). For more information about using IAM, see [Enabling IAM user and role access to your cluster \(p. 402\)](#). After configuring authentication to your cluster, you can create Kubernetes roles and clusterroles to assign permissions to the roles, and then bind the roles to the identities using Kubernetes rolebindings and clusterrolebindings. For more information, see [Using RBAC Authorization](#) in the Kubernetes documentation.

### Considerations

- You can associate one OIDC identity provider to your cluster.
- Kubernetes doesn't provide an OIDC identity provider. You can use an existing public OIDC identity provider, or you can run your own identity provider. For a list of certified providers, see [OpenID Certification](#) on the OpenID site.

- The issuer URL of the OIDC identity provider must be publicly accessible, so that Amazon EKS can discover the signing keys. Amazon EKS does not support OIDC identity providers with self-signed certificates.
- You can't disable the AWS IAM authenticator on your cluster, because it is still required for joining nodes to a cluster. For more information, see [AWS IAM Authenticator for Kubernetes](#) on GitHub.
- An Amazon EKS cluster must still be created by an AWS IAM user, rather than an OIDC identity provider user. This is because the cluster creator interacts with the Amazon EKS APIs, rather than the Kubernetes APIs.
- OIDC identity provider-authenticated users are listed in the cluster's audit log if CloudWatch logs are turned on for the control plane. For more information, see [Enabling and disabling control plane logs \(p. 65\)](#).
- You can't sign in to the AWS Management Console with an account from an OIDC provider. You can only [view Kubernetes resources \(p. 506\)](#) in the console by signing into the AWS Management Console with an AWS Identity and Access Management account.

## Associate an OIDC identity provider

Before you can associate an OIDC identity provider with your cluster, you need the following information from your provider:

- **Issuer URL** – The URL of the OIDC identity provider that allows the API server to discover public signing keys for verifying tokens. The URL must begin with `https://` and should correspond to the `iss` claim in the provider's OIDC ID tokens. In accordance with the OIDC standard, path components are allowed but query parameters are not. Typically the URL consists of only a host name, like `https://server.example.org` or `https://example.com`. This URL should point to the level below `.well-known/openid-configuration` and must be publicly accessible over the internet.
- **Client ID (also known as *audience*)** – The ID for the client application that makes authentication requests to the OIDC identity provider.

You can associate an identity provider using `eksctl` or the AWS Management Console.

`eksctl`

### To associate an OIDC identity provider to your cluster using `eksctl`

1. Create a file named `associate-identity-provider.yaml` with the following contents. Replace the `example values` with your own. The values in the `identityProviders` section are obtained from your OIDC identity provider. Values are only required for the `name`, `type`, `issuerUrl`, and `clientId` settings under `identityProviders`.

```

apiVersion: eksctl.io/v1alpha5
kind: ClusterConfig

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

identityProviders:
 - name: my-provider
 type: oidc
 issuerUrl: https://example.com
 clientId: kubernetes
 usernameClaim: email
 usernamePrefix: my-username-prefix
 groupsClaim: my-claim
```

```
groupsPrefix: my-groups-prefix
requiredClaims:
 string: string
tags:
 env: dev
```

**Important**

Don't specify `system:`, or any portion of that string, for `groupsPrefix` or `usernamePrefix`.

2. Create the provider.

```
eksctl associate identityprovider -f associate-identity-provider.yaml
```

3. To use `kubectl` to work with your cluster and OIDC identity provider, see [Using `kubectl`](#) in the Kubernetes documentation.

AWS Management Console

**To associate an OIDC identity provider to your cluster using the AWS Management Console**

1. Open the Amazon EKS console at <https://console.aws.amazon.com/eks/home#/clusters>.
2. Select your cluster.
3. Select the **Configuration** tab, and then select the **Authentication** tab.
4. On the **OIDC Identity Providers** page, select **Associate Identity Provider**.
5. On the **Associate OIDC Identity Provider** page, enter or select the following options, and then select **Associate**.
  - For **Name**, enter a unique name for the provider.
  - For **Issuer URL**, enter the URL for your provider. This URL must be accessible over the internet.
  - For **Client ID**, enter the OIDC identity provider's client ID (also known as **audience**).
  - For **Username claim**, enter the claim to use as the username.
  - For **Groups claim**, enter the claim to use as the user's group.
  - (Optional) Select **Advanced options**, enter or select the following information.
    - **Username prefix** – Enter a prefix to prepend to username claims. The prefix is prepended to username claims to prevent clashes with existing names. If you do not provide a value, and the username is a value other than `email`, the prefix defaults to the value for **Issuer URL**. You can use the value `-` to disable all prefixing. Don't specify `system:` or any portion of that string.
    - **Groups prefix** – Enter a prefix to prepend to groups claims. The prefix is prepended to group claims to prevent clashes with existing names (such as `system: groups`). For example, the value `oidc:` creates group names like `oidc:engineering` and `oidc:infra`. Don't specify `system:` or any portion of that string..
    - **Required claims** – Select **Add claim** and enter one or more key value pairs that describe required claims in the client ID token. The pairs describe required claims in the ID Token. If set, each claim is verified to be present in the ID token with a matching value.
6. To use `kubectl` to work with your cluster and OIDC identity provider, see [Using `kubectl`](#) in the Kubernetes documentation.

## Disassociate an OIDC identity provider from your cluster

If you disassociate an OIDC identity provider from your cluster, users included in the provider can no longer access the cluster. However, you can still access the cluster with AWS IAM users.

### To disassociate an OIDC identity provider from your cluster using the AWS Management Console

1. Open the Amazon EKS console at <https://console.aws.amazon.com/eks/home#/clusters>.
2. In the **OIDC Identity Providers** section, select **Disassociate**, enter the identity provider name, and then select **Disassociate**.

## Example IAM policy

If you want to prevent an OIDC identity provider from being associated with a cluster, create and associate the following IAM policy to the IAM accounts of your Amazon EKS administrators. For more information, see [Creating IAM policies](#) and [Adding IAM identity permissions](#) in the IAM User Guide and [Actions, resources, and condition keys for Amazon Elastic Kubernetes Service](#) in the Service Authorization Reference.

```
{
 "Version": "2012-10-17",
 "Statement": [
 {
 "Sid": "denyOIDC",
 "Effect": "Deny",
 "Action": [
 "eks:AssociateIdentityProviderConfig"
],
 "Resource": "arn:aws:eks:us-west-2.amazonaws.com:111122223333:cluster/*"
 },
 {
 "Sid": "eksAdmin",
 "Effect": "Allow",
 "Action": [
 "eks:*"
],
 "Resource": "*"
 }
]
}
```

The following example policy allows OIDC identity provider association if the `clientID` is `kubernetes` and the `issuerUrl` is `https://cognito-idp.us-west-2amazonaws.com/*`.

```
{
 "Version": "2012-10-17",
 "Statement": [
 {
 "Sid": "AllowCognitoOnly",
 "Effect": "Deny",
 "Action": "eks:AssociateIdentityProviderConfig",
 "Resource": "arn:aws:eks:us-west-2:111122223333:cluster/my-instance",
 "Condition": {
 "StringNotLikeIfExists": {
 "clientID": "kubernetes",
 "issuerUrl": "https://cognito-idp.us-west-2amazonaws.com/*"
 }
 }
 }
]
}
```

```
 "eks:issuerUrl": "https://cognito-idp.us-west-2.amazonaws.com/*"
 }
}
{
 "Sid": "DenyOtherClients",
 "Effect": "Deny",
 "Action": "eks:AssociateIdentityProviderConfig",
 "Resource": "arn:aws:eks:us-west-2:111122223333:cluster/my-instance",
 "Condition": {
 "StringNotEquals": {
 "eks:clientId": "kubernetes"
 }
 }
},
{
 "Sid": "AllowOthers",
 "Effect": "Allow",
 "Action": "eks:*",
 "Resource": "*"
}
]
```

## Create a kubeconfig for Amazon EKS

In this section, you create a `kubeconfig` file for your cluster (or update an existing one).

This section offers two procedures to create or update your `kubeconfig` file. You can quickly create or update a `kubeconfig` file with the AWS CLI `update-kubeconfig` command automatically by using the AWS CLI, or you can create a `kubeconfig` file manually using the AWS CLI or the `aws-iam-authenticator`.

Amazon EKS uses the `aws eks get-token` command, available in version 1.16.156 or later of the AWS CLI or the [AWS IAM Authenticator for Kubernetes](#) with `kubectl` for cluster authentication. If you have installed the AWS CLI on your system, then by default the AWS IAM Authenticator for Kubernetes uses the same credentials that are returned with the following command:

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

For more information, see [Configuring the AWS CLI](#) in the *AWS Command Line Interface User Guide*.

## Create kubeconfig file automatically

### To create your `kubeconfig` file with the AWS CLI

1. Make sure that you have version 1.23.11 or 2.6.3 or later installed. To install or upgrade the AWS CLI, see [Installing the AWS CLI](#) in the *AWS Command Line Interface User Guide*.

#### Note

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

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

```
aws --version
```

**Important**

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

2. Create or update a kubeconfig file for your cluster. Replace the *example values* with your own.
  - By default, the resulting configuration file is created at the default kubeconfig path (`.kube`) in your home directory or merged with an existing config file at that location. You can specify another path with the `--kubeconfig` option.
  - You can specify an IAM role ARN with the `--role-arn` option to use for authentication when you issue `kubectl` commands. Otherwise, the IAM entity in your default AWS CLI or SDK credential chain is used. You can view your default AWS CLI or SDK identity by running the `aws sts get-caller-identity` command.
  - For more information, see the help page with the `aws eks update-kubeconfig help` command or see [update-kubeconfig](#) in the [AWS CLI Command Reference](#).

**Note**

To run the following command, you must have permission to use the `eks:DescribeCluster` API action with the cluster that you specify. For more information, see [Amazon EKS identity-based policy examples \(p. 462\)](#).

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

3. Test your configuration.

```
kubectl get svc
```

**Note**

If you receive any authorization or resource type errors, see [Unauthorized or access denied \(kubectl\) \(p. 547\)](#) in the troubleshooting section.

Output:

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

## Create kubeconfig manually

### To create your kubeconfig file manually

1. Set values for a few variables by replacing the *example values* with your own and then running the modified commands.

```
export region_code=region-code
export cluster_name=my-cluster
export account_id=111122223333
```

2. Retrieve the endpoint for your cluster and store the value in a variable.

```
cluster_endpoint=$(aws eks describe-cluster \
--region $region_code \
--name $cluster_name \
--query "cluster.endpoint" \
```

```
--output text)
```

3. Retrieve the Base64-encoded certificate data required to communicate with your cluster and store the value in a variable.

```
certificate_data=$(aws eks describe-cluster \
--region $region_code \
--name $cluster_name \
--query "cluster.certificateAuthority.data" \
--output text)
```

4. Create the default ~/ .kube directory if it doesn't already exist.

```
mkdir -p ~/.kube
```

5. Run the command for your preferred client token method (AWS CLI or AWS IAM authenticator for Kubernetes) to create the config file in the ~/ .kube directory. You can specify the following before running one of the commands by modifying the command to include the following:

- **An IAM role** – Remove the # at the start of the lines under args:. Replace *my-role* with the name of the IAM role that you want to perform cluster operations with instead of the default AWS credential provider chain. For more information, see [Set up kubectl to use authentication tokens provided by AWS IAM Authenticator for Kubernetes](#) on GitHub.
- **An AWS CLI named profile** – Remove the # at the start of the env: line, and remove # at the start of the lines under it. Replace *aws-profile* with the name of the profile to use. If you don't specify a profile, then the default profile is used. For additional information, see [Specifying Credentials & Using AWS Profiles](#) on GitHub.

## AWS CLI

### Prerequisite

Version 1.16.156 or later of the AWS CLI must be installed on your device.

```
#!/bin/bash
read -r -d '' KUBECONFIG <<EOF
apiVersion: v1
clusters:
- cluster:
 certificate-authority-data: $certificate_data
 server: $cluster_endpoint
 name: arn:aws:eks:$region_code:$account_id:cluster/$cluster_name
contexts:
- context:
 cluster: arn:aws:eks:$region_code:$account_id:cluster/$cluster_name
 user: arn:aws:eks:$region_code:$account_id:cluster/$cluster_name
 name: arn:aws:eks:$region_code:$account_id:cluster/$cluster_name
 current-context: arn:aws:eks:$region_code:$account_id:cluster/$cluster_name
kind: Config
preferences: {}
users:
- name: arn:aws:eks:$region_code:$account_id:cluster/$cluster_name
 user:
 exec:
 apiVersion: client.authentication.k8s.io/v1alpha1
 command: aws
 args:
 - --region
 - $region_code
 - eks
EOF
```

```

 - get-token
 - --cluster-name
 - $cluster_name
 # - "-r"
 # - "arn:aws:iam::$account_id:role/my-role"
 # env:
 # name: "AWS_PROFILE"
 # value: "aws-profile"
EOF
echo "${KUBECONFIG}" > ~/.kube/config

```

## AWS IAM Authenticator for Kubernetes

### Prerequisite

The AWS IAM Authenticator for Kubernetes must be installed on your device. To install it, see [Installing aws-iam-authenticator \(p. 417\)](#).

```

#!/bin/bash
read -r -d '' KUBECONFIG <<EOF
apiVersion: v1
clusters:
- cluster:
 server: $cluster_endpoint
 certificate-authority-data: $certificate_data
 name: arn:aws:eks:$region_code:$account_id:cluster/$cluster_name
contexts:
- context:
 cluster: arn:aws:eks:$region_code:$account_id:cluster/$cluster_name
 user: arn:aws:eks:$region_code:$account_id:cluster/$cluster_name
 name: arn:aws:eks:$region_code:$account_id:cluster/$cluster_name
current-context: arn:aws:eks:$region_code:$account_id:cluster/$cluster_name
kind: Config
preferences: {}
users:
- name: arn:aws:eks:$region_code:$account_id:cluster/$cluster_name
 user:
 exec:
 apiVersion: client.authentication.k8s.io/v1alpha1
 command: aws-iam-authenticator
 args:
 - "token"
 - "-i"
 - "$cluster_name"
 # - "-r"
 # - "arn:aws:iam::$account_id:role/my-role"
 # env:
 # name: "AWS_PROFILE"
 # value: "aws-profile"
EOF
echo "${KUBECONFIG}" > ~/.kube/config

```

- Add the file path to your KUBECONFIG environment variable so that kubectl knows where to look for your cluster configuration.

- For Bash shells on macOS or Linux:

```
export KUBECONFIG=$KUBECONFIG:~/.kube/config
```

- For PowerShell on Windows:

```
$ENV:KUBECONFIG="{0};{1}" -f $ENV:KUBECONFIG, "$ENV:UserProfile\.kube\config"
```

7. (Optional) Add the configuration to your shell initialization file so that it is configured when you open a shell.

- For Bash shells on macOS:

```
echo 'export KUBECONFIG=$KUBECONFIG:~/.kube/config' >> ~/.bash_profile
```

- For Bash shells on Linux:

```
echo 'export KUBECONFIG=$KUBECONFIG:~/.kube/config' >> ~/.bashrc
```

- For PowerShell on Windows:

```
[System.Environment]::SetEnvironmentVariable('KUBECONFIG', $ENV:KUBECONFIG, 'Machine')
```

8. Test your configuration.

```
kubectl get svc
```

**Note**

If you receive any authorization or resource type errors, see [Unauthorized or access denied \(kubectl\) \(p. 547\)](#) in the troubleshooting section.

Output:

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

## Installing aws-iam-authenticator

Amazon EKS uses IAM to provide authentication to your Kubernetes cluster through the [AWS IAM authenticator for Kubernetes](#). You can configure the stock `kubectl` client to work with Amazon EKS by installing the AWS IAM authenticator for Kubernetes and [modifying your kubectl configuration file \(p. 413\)](#) to use it for authentication.

**Note**

If you're running the AWS CLI version 1.16.156 or later, then you don't need to install the authenticator. Instead, you can use the `aws eks get-token` command. For more information, see [Create kubeconfig manually \(p. 414\)](#).

If you're unable to use the AWS CLI version 1.16.156 or later to create the `kubeconfig` file, then you can install the AWS IAM authenticator for Kubernetes on macOS, Linux, or Windows.

macOS

### To install aws-iam-authenticator with Homebrew

The easiest way to install the `aws-iam-authenticator` is with [Homebrew](#).

1. If you do not already have [Homebrew](#) installed on your Mac, install it with the following command.

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

2. Install the `aws-iam-authenticator` with the following command.

```
brew install aws-iam-authenticator
```

3. Test that the aws-iam-authenticator binary works.

```
aws-iam-authenticator help
```

### To install aws-iam-authenticator on macOS

You can also install the AWS-vended version of the aws-iam-authenticator by following these steps.

1. Download the Amazon EKS vended aws-iam-authenticator binary from Amazon S3.

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

2. (Optional) Verify the downloaded binary with the SHA-256 sum provided in the same bucket prefix.

- a. Download the SHA-256 sum for your system.

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

- b. Check the SHA-256 sum for your downloaded binary.

```
openssl sha1 -sha256 aws-iam-authenticator
```

- c. Compare the generated SHA-256 sum in the command output against your downloaded aws-iam-authenticator.sha256 file. The two should match.

3. Apply execute permissions to the binary.

```
chmod +x ./aws-iam-authenticator
```

4. Copy the binary to a folder in your \$PATH. We recommend creating a \$HOME/bin/aws-iam-authenticator and ensuring that \$HOME/bin comes first in your \$PATH.

```
mkdir -p $HOME/bin && cp ./aws-iam-authenticator $HOME/bin/aws-iam-authenticator && export PATH=$PATH:$HOME/bin
```

5. Add \$HOME/bin to your PATH environment variable.

```
echo 'export PATH=$PATH:$HOME/bin' >> ~/.bash_profile
```

6. Test that the aws-iam-authenticator binary works.

```
aws-iam-authenticator help
```

### Linux

### To install aws-iam-authenticator on Linux

1. Download the Amazon EKS vended aws-iam-authenticator binary from Amazon S3 for your hardware platform.

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

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

2. (Optional) Verify the downloaded binary with the SHA-256 sum provided in the same bucket prefix for your hardware platform.

- a. Download the SHA-256 sum for your system.

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

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

- b. Check the SHA-256 sum for your downloaded binary.

```
openssl sha1 -sha256 aws-iam-authenticator
```

- c. Compare the generated SHA-256 sum in the command output against your downloaded aws-iam-authenticator.sha256 file. The two should match.

3. Apply execute permissions to the binary.

```
chmod +x ./aws-iam-authenticator
```

4. Copy the binary to a folder in your \$PATH. We recommend creating a \$HOME/bin/aws-iam-authenticator and ensuring that \$HOME/bin comes first in your \$PATH.

```
mkdir -p $HOME/bin && cp ./aws-iam-authenticator $HOME/bin/aws-iam-authenticator && export PATH=$PATH:$HOME/bin
```

5. Add \$HOME/bin to your PATH environment variable.

```
echo 'export PATH=$PATH:$HOME/bin' >> ~/.bashrc
```

6. Test that the aws-iam-authenticator binary works.

```
aws-iam-authenticator help
```

## Windows

### To install aws-iam-authenticator on Windows with Chocolatey

1. If you do not already have Chocolatey installed on your Windows system, see [Installing chocolatey](#).
2. Open a PowerShell terminal window and install the aws-iam-authenticator package with the following command:

```
choco install -y aws-iam-authenticator
```

3. Test that the aws-iam-authenticator binary works.

```
aws-iam-authenticator help
```

### To install aws-iam-authenticator on Windows

1. Open a PowerShell terminal window and download the Amazon EKS vended aws-iam-authenticator binary from Amazon S3.

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

2. (Optional) Verify the downloaded binary with the SHA-256 sum provided in the same bucket prefix.

- a. Download the SHA-256 sum for your system.

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

- b. Check the SHA-256 sum for your downloaded binary.

```
Get-FileHash aws-iam-authenticator.exe
```

- c. Compare the generated SHA-256 sum in the command output against your downloaded SHA-256 file. The two should match, although the PowerShell output will be uppercase.

3. Copy the binary to a folder in your PATH. If you have an existing directory in your PATH that you use for command line utilities, copy the binary to that directory. Otherwise, complete the following steps.

- a. Create a new directory for your command line binaries, such as C:\bin.
- b. Copy the aws-iam-authenticator.exe binary to your new directory.
- c. Edit your user or system PATH environment variable to add the new directory to your PATH.
- d. Close your PowerShell terminal and open a new one to pick up the new PATH variable.

4. Test that the aws-iam-authenticator binary works.

```
aws-iam-authenticator help
```

If you have an existing Amazon EKS cluster, create a kubeconfig file for that cluster. For more information, see [Create a kubeconfig for Amazon EKS \(p. 413\)](#). Otherwise, see [Creating an Amazon EKS cluster \(p. 24\)](#) to create a new Amazon EKS cluster.

## Default Amazon EKS Kubernetes roles and users

Amazon EKS creates identities for each of the Kubernetes components in 1.20 and later clusters. The following objects have been created by the Amazon EKS team to provide RBAC permissions for the components:

### Roles

- eks:authenticator
- eks:certificate-controller approver

- eks:certificate-controller
- eks:cluster-event-watcher
- eks:fargate-scheduler
- eks:k8s-metrics
- eks:nodewatcher
- eks:pod-identity-mutating-webhook

### RoleBindings

- eks:authenticator
- eks:certificate-controller-approver
- eks:certificate-controller
- eks:cluster-event-watcher
- eks:fargate-scheduler
- eks:k8s-metrics
- eks:nodewatcher
- eks:pod-identity-mutating-webhook

### Users

- eks:authenticator
- eks:certificate-controller
- eks:cluster-event-watcher
- eks:fargate-scheduler
- eks:k8s-metrics
- eks:nodewatcher
- eks:pod-identity-mutating-webhook

In addition to the objects above, Amazon EKS uses a special user identity `eks:cluster-bootstrap` for `kubectl` operations during cluster bootstrap. Amazon EKS also uses a special user identity `eks:support-engineer` for cluster management operations. All the user identities will appear in the `kube audit logs` available to customers through CloudWatch.

Run `kubectl describe clusterrole <rolename>` to see the permissions for each role.

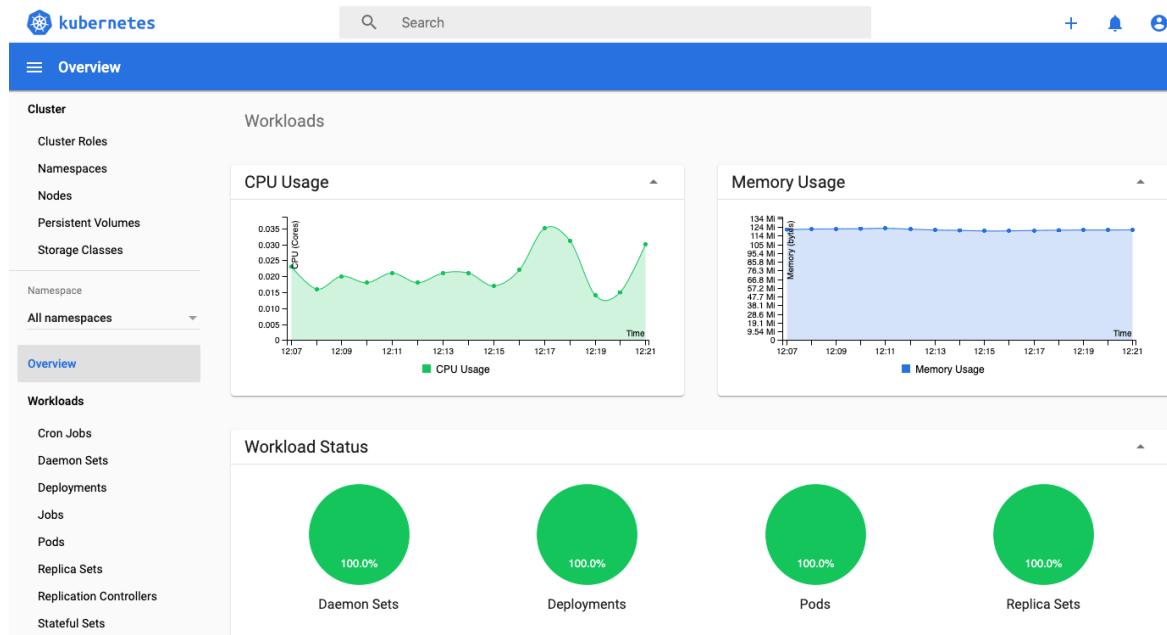
# Cluster management

This chapter includes the following topics to help you manage your cluster. You can also view information about your [Kubernetes resources \(p. 506\)](#) with the AWS Management Console.

- [Tutorial: Deploy the Kubernetes Dashboard \(web UI\) \(p. 422\)](#) – Learn how to install the dashboard, a web-based user interface for your Kubernetes cluster and applications.
- [Installing the Kubernetes Metrics Server \(p. 426\)](#) – The Kubernetes Metrics Server is an aggregator of resource usage data in your cluster. It is not deployed by default in your cluster, but is used by Kubernetes add-ons, such as the Kubernetes Dashboard and [Horizontal Pod Autoscaler \(p. 370\)](#). In this topic you learn how to install the Metrics Server.
- [Control plane metrics with Prometheus \(p. 427\)](#) – The Kubernetes API server exposes a number of metrics that are useful for monitoring and analysis. This topic explains how to deploy Prometheus and some of the ways that you can use it to view and analyze what your cluster is doing.
- [Using Helm with Amazon EKS \(p. 430\)](#) – The Helm package manager for Kubernetes helps you install and manage applications on your Kubernetes cluster. This topic helps you install and run the Helm binaries so that you can install and manage charts using the Helm CLI on your local computer.
- [Tagging your Amazon EKS resources \(p. 431\)](#) – To help you manage your Amazon EKS resources, you can assign your own metadata to each resource in the form of *tags*. This topic describes tags and shows you how to create them.
- [Amazon EKS service quotas \(p. 435\)](#) – Your AWS account has default quotas, formerly referred to as limits, for each AWS service. Learn about the quotas for Amazon EKS and how to increase them.

## Tutorial: Deploy the Kubernetes Dashboard (web UI)

This tutorial guides you through deploying the [Kubernetes Dashboard](#) to your Amazon EKS cluster, complete with CPU and memory metrics. It also helps you to create an Amazon EKS administrator service account that you can use to securely connect to the dashboard to view and control your cluster.



## Prerequisites

This tutorial assumes the following:

- You have created an Amazon EKS cluster by following the steps in [Getting started with Amazon EKS \(p. 4\)](#).
- You have the Kubernetes Metrics Server installed. For more information, see [Installing the Kubernetes Metrics Server \(p. 426\)](#).
- The security groups for your control plane elastic network interfaces and nodes follow the recommended settings in [Amazon EKS security group considerations \(p. 272\)](#).
- You are using a `kubectl` client that is [configured to communicate with your Amazon EKS cluster \(p. 18\)](#).

## Step 1: Deploy the Kubernetes dashboard

- For Regions other than Beijing and Ningxia China, apply the Kubernetes dashboard.

```
kubectl apply -f https://raw.githubusercontent.com/kubernetes/dashboard/v2.0.5/aio/deploy/recommended.yaml
```

- For the Beijing and Ningxia China Region, download, modify, and apply the Calico manifests to your cluster.

1. Download the Kubernetes Dashboard manifest with the following command.

```
curl -o recommended.yaml https://raw.githubusercontent.com/kubernetes/dashboard/blob/master/aio/deploy/recommended.yaml
```

2. Modify the manifest files.

- a. View the manifest file or files that you downloaded and note the name of the image. Download the image locally with the following command.

```
docker pull image:<tag>
```

- b. Tag the image to be pushed to an Amazon Elastic Container Registry repository in China with the following command.

```
docker tag image:<tag> <111122223333>.dkr.ecr.<cn-north-1>.amazonaws.com/image:<tag>
```

- c. Update the `recommended.yaml` to reference the Amazon ECR image URL in your Region.

- d. Update the `recommended.yaml` file to reference the Amazon ECR image repository in your Region by adding the following to the spec.

```
registry: <111122223333>.dkr.ecr.<cn-north-1>.amazonaws.com
```

- e. Apply the Kubernetes Dashboard manifest.

```
kubectl apply -f recommended.yaml
```

3. Apply the manifest to your cluster with the following command.

```
kubectl apply -f recommended.yaml
```

Output:

```
namespace/kubernetes-dashboard created
serviceaccount/kubernetes-dashboard created
service/kubernetes-dashboard created
secret/kubernetes-dashboard-certs created
secret/kubernetes-dashboard-csrf created
secret/kubernetes-dashboard-key-holder created
configmap/kubernetes-dashboard-settings created
role.rbac.authorization.k8s.io/kubernetes-dashboard created
clusterrole.rbac.authorization.k8s.io/kubernetes-dashboard created
rolebinding.rbac.authorization.k8s.io/kubernetes-dashboard created
clusterrolebinding.rbac.authorization.k8s.io/kubernetes-dashboard created
deployment.apps/kubernetes-dashboard created
service/dashboard-metrics-scraper created
deployment.apps/dashboard-metrics-scraper created
```

## Step 2: Create an eks-admin service account and cluster role binding

By default, the Kubernetes Dashboard user has limited permissions. In this section, you create an eks-admin service account and cluster role binding that you can use to securely connect to the dashboard with admin-level permissions. For more information, see [Managing Service Accounts](#) in the Kubernetes documentation.

### To create the eks-admin service account and cluster role binding

#### Important

The example service account created with this procedure has full `cluster-admin` (superuser) privileges on the cluster. For more information, see [Using RBAC authorization](#) in the Kubernetes documentation.

1. Create a file called `eks-admin-service-account.yaml` with the text below. This manifest defines a service account and cluster role binding called `eks-admin`.

```
apiVersion: v1
kind: ServiceAccount
metadata:
 name: eks-admin
 namespace: kube-system

apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
 name: eks-admin
 roleRef:
 apiGroup: rbac.authorization.k8s.io
 kind: ClusterRole
 name: cluster-admin
 subjects:
 - kind: ServiceAccount
 name: eks-admin
 namespace: kube-system
```

2. Apply the service account and cluster role binding to your cluster.

```
kubectl apply -f eks-admin-service-account.yaml
```

Output:

```
serviceaccount "eks-admin" created
clusterrolebinding.rbac.authorization.k8s.io "eks-admin" created
```

## Step 3: Connect to the dashboard

Now that the Kubernetes Dashboard is deployed to your cluster, and you have an administrator service account that you can use to view and control your cluster, you can connect to the dashboard with that service account.

### To connect to the Kubernetes dashboard

1. Retrieve an authentication token for the eks-admin service account. Copy the <authentication\_token> value from the output. You use this token to connect to the dashboard.

```
kubectl -n kube-system describe secret $(kubectl -n kube-system get secret | grep eks-admin | awk '{print $1}')
```

Output:

```
Name: eks-admin-token-b5zv4
Namespace: kube-system
Labels: <none>
Annotations: kubernetes.io/service-account.name=eks-admin
 kubernetes.io/service-account.uid=bce66ac-39be-11e8-97e8-026dce96b6e8

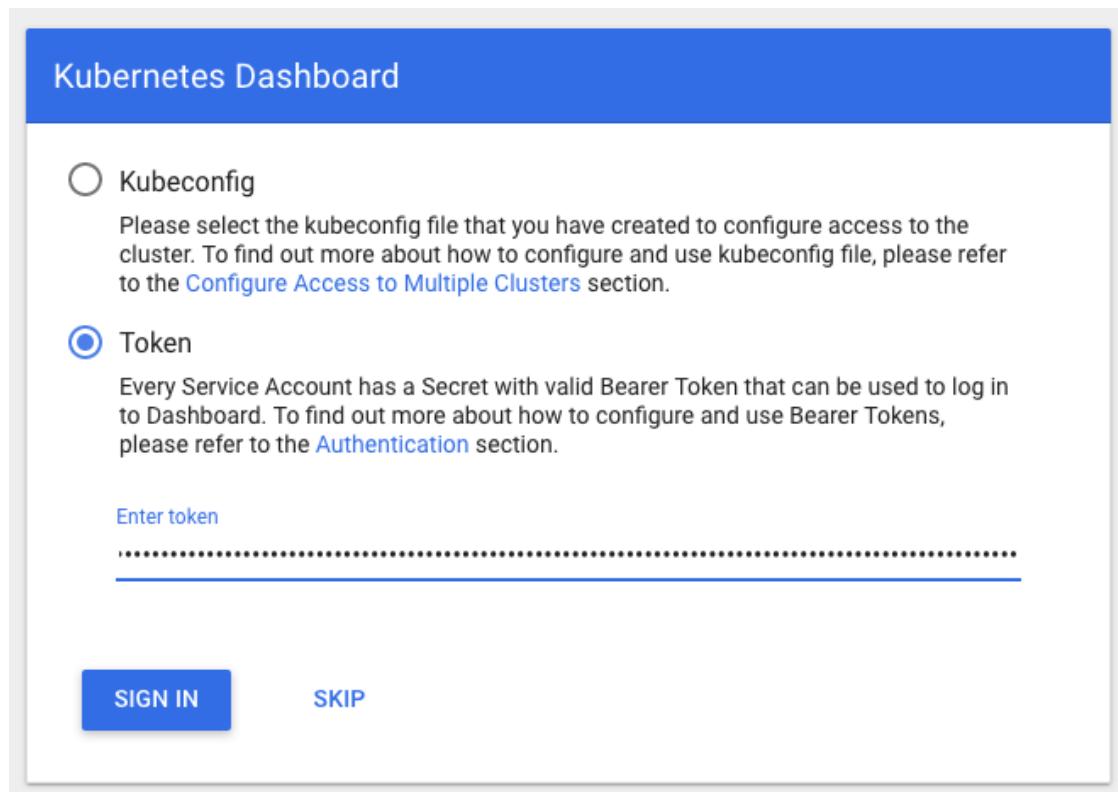
Type: kubernetes.io/service-account-token

Data
=====
ca.crt: 1025 bytes
namespace: 11 bytes
token: <authentication_token>
```

2. Start the `kubectl proxy`.

```
kubectl proxy
```

3. To access the dashboard endpoint, open the following link with a web browser: <http://localhost:8001/api/v1/namespaces/kubernetes-dashboard/services/https:kubernetes-dashboard:/proxy/#!/login>.
4. Choose **Token**, paste the <authentication\_token> output from the previous command into the **Token** field, and choose **SIGN IN**.



**Note**

It may take a few minutes before CPU and memory metrics appear in the dashboard.

## Step 4: Next steps

After you have connected to your Kubernetes Dashboard, you can view and control your cluster using your `eks-admin` service account. For more information about using the dashboard, see the [project documentation on GitHub](#).

## Installing the Kubernetes Metrics Server

The Kubernetes Metrics Server is an aggregator of resource usage data in your cluster, and it is not deployed by default in Amazon EKS clusters. For more information, see [Kubernetes Metrics Server](#) on GitHub. The Metrics Server is commonly used by other Kubernetes add ons, such as the [Horizontal Pod Autoscaler \(p. 370\)](#) or the [Kubernetes Dashboard \(p. 422\)](#). For more information, see [Resource metrics pipeline](#) in the Kubernetes documentation. This topic explains how to deploy the Kubernetes Metrics Server on your Amazon EKS cluster.

**Important**

Don't use Metrics Server when you need an accurate source of resource usage metrics or as a monitoring solution.

### Deploy the Metrics Server

1. Deploy the Metrics Server with the following command:

```
kubectl apply -f https://github.com/kubernetes-sigs/metrics-server/releases/latest/download/components.yaml
```

- Verify that the metrics-server deployment is running the desired number of pods with the following command.

```
kubectl get deployment metrics-server -n kube-system
```

Example output:

| NAME           | READY | UP-TO-DATE | AVAILABLE | AGE |
|----------------|-------|------------|-----------|-----|
| metrics-server | 1/1   | 1          | 1         | 6m  |

## Control plane metrics with Prometheus

The Kubernetes API server exposes a number of metrics that are useful for monitoring and analysis. These metrics are exposed internally through a metrics endpoint that refers to the /metrics HTTP API. Like other endpoints, this endpoint is exposed on the Amazon EKS control plane. This topic explains some of the ways you can use this endpoint to view and analyze what your cluster is doing.

### Viewing the raw metrics

To view the raw metrics output, use `kubectl` with the `--raw` flag. This command allows you to pass any HTTP path and returns the raw response.

```
kubectl get --raw /metrics
```

Example output:

```
...
HELP rest_client_requests_total Number of HTTP requests, partitioned by status code,
method, and host.
TYPE rest_client_requests_total counter
rest_client_requests_total{code="200",host="127.0.0.1:21362",method="POST"} 4994
rest_client_requests_total{code="200",host="127.0.0.1:443",method="DELETE"} 1
rest_client_requests_total{code="200",host="127.0.0.1:443",method="GET"} 1.326086e+06
rest_client_requests_total{code="200",host="127.0.0.1:443",method="PUT"} 862173
rest_client_requests_total{code="404",host="127.0.0.1:443",method="GET"} 2
rest_client_requests_total{code="409",host="127.0.0.1:443",method="POST"} 3
rest_client_requests_total{code="409",host="127.0.0.1:443",method="PUT"} 8
HELP ssh_tunnel_open_count Counter of ssh tunnel total open attempts
TYPE ssh_tunnel_open_count counter
ssh_tunnel_open_count 0
HELP ssh_tunnel_open_fail_count Counter of ssh tunnel failed open attempts
TYPE ssh_tunnel_open_fail_count counter
ssh_tunnel_open_fail_count 0
```

This raw output returns verbatim what the API server exposes. These metrics are represented in a [Prometheus format](#). This format allows the API server to expose different metrics broken down by line. Each line includes a metric name, tags, and a value.

```
<metric_name>{"<tag>"=<value>"[<, . . . >]"} <value>
```

While this endpoint is useful if you are looking for a specific metric, you typically want to analyze these metrics over time. To do this, you can deploy [Prometheus](#) into your cluster. Prometheus is a monitoring and time series database that scrapes exposed endpoints and aggregates data, allowing you to filter, graph, and query the results.

## Deploying Prometheus

This topic helps you deploy Prometheus into your cluster with Helm V3. If you already have Helm installed, you can check your version with the `helm version` command. Helm is a package manager for Kubernetes clusters. For more information about Helm and how to install it, see [Using Helm with Amazon EKS \(p. 430\)](#).

After you configure Helm for your Amazon EKS cluster, you can use it to deploy Prometheus with the following steps.

### To deploy Prometheus using Helm

1. Create a Prometheus namespace.

```
kubectl create namespace prometheus
```

2. Add the `prometheus-community` chart repository.

```
helm repo add prometheus-community https://prometheus-community.github.io/helm-charts
```

3. Deploy Prometheus.

```
helm upgrade -i prometheus prometheus-community/prometheus \
--namespace prometheus \
--set
alertmanager.persistentVolume.storageClass="gp2",server.persistentVolume.storageClass="gp2"
```

#### Note

If you get the error `Error: failed to download "stable/prometheus"` (hint: running `helm repo update` may help) when executing this command, run `helm repo update`, and then try running the Step 2 command again.

If you get the error `Error: rendered manifests contain a resource that already exists`, run `helm uninstall your-release-name -n namespace`, then try running the Step 3 command again.

4. Verify that all of the pods in the `prometheus` namespace are in the `READY` state.

```
kubectl get pods -n prometheus
```

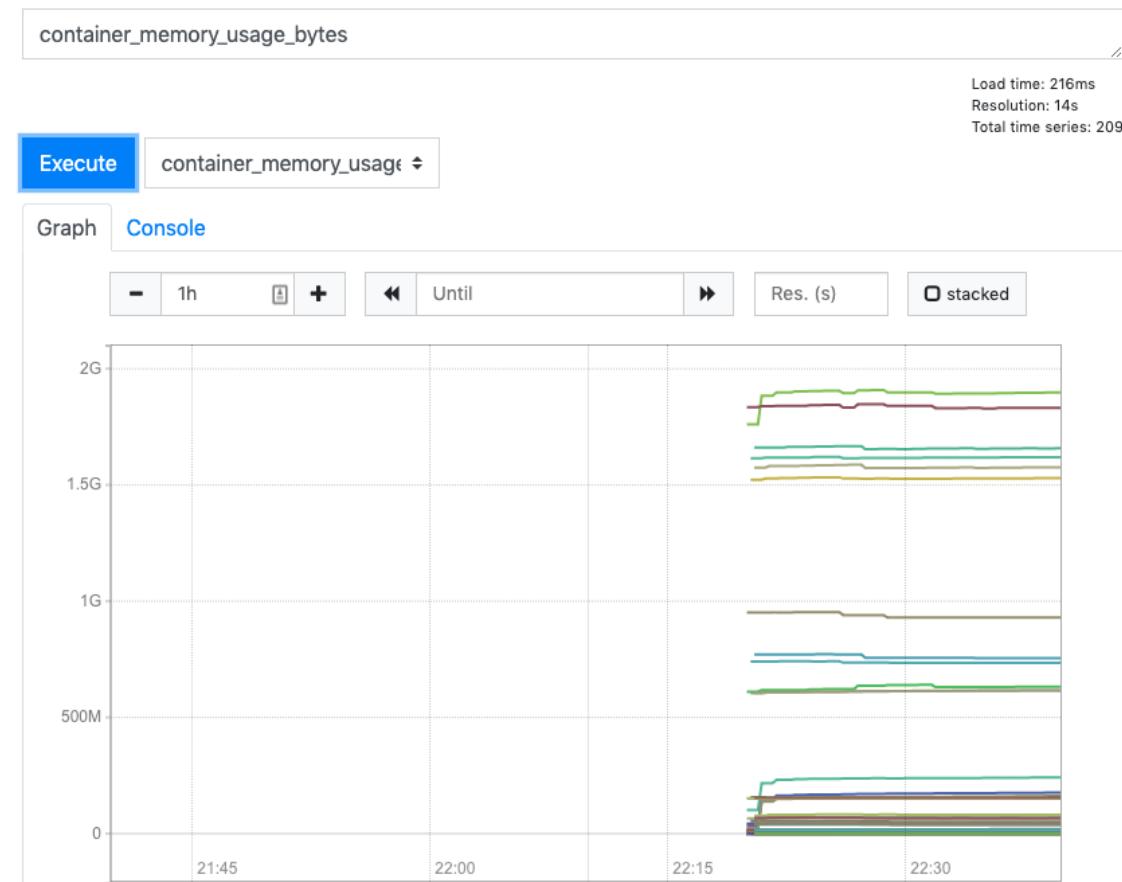
Output:

| NAME                                            | READY | STATUS  | RESTARTS | AGE |
|-------------------------------------------------|-------|---------|----------|-----|
| prometheus-alertmanager-59b4c8c744-r7bgp        | 1/2   | Running | 0        | 48s |
| prometheus-kube-state-metrics-7cf8d87cf99-jkz2f | 1/1   | Running | 0        | 48s |
| prometheus-node-exporter-jcjzq                  | 1/1   | Running | 0        | 48s |
| prometheus-node-exporter-jxv2h                  | 1/1   | Running | 0        | 48s |
| prometheus-node-exporter-vbdks                  | 1/1   | Running | 0        | 48s |
| prometheus-pushgateway-76c444b68c-82tnw         | 1/1   | Running | 0        | 48s |
| prometheus-server-775957f748-mmht9              | 1/2   | Running | 0        | 48s |

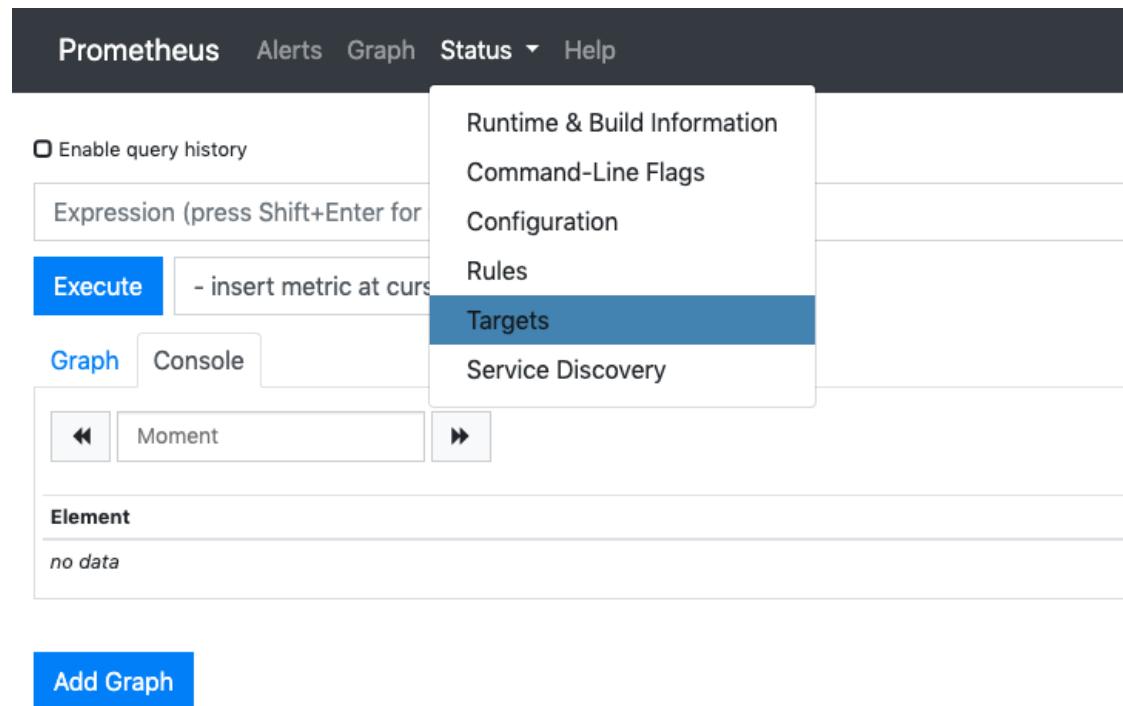
5. Use `kubectl` to port forward the Prometheus console to your local machine.

```
kubectl --namespace=prometheus port-forward deploy/prometheus-server 9090
```

6. Point a web browser to [localhost:9090](http://localhost:9090) to view the Prometheus console.
7. Choose a metric from the **- insert metric at cursor** menu, then choose **Execute**. Choose the **Graph** tab to show the metric over time. The following image shows `container_memory_usage_bytes` over time.



8. From the top navigation bar, choose **Status**, then **Targets**.



All of the Kubernetes endpoints that are connected to Prometheus using service discovery are displayed.

## Store your Prometheus metrics in Amazon Managed Service for Prometheus

Amazon Managed Service for Prometheus is a Prometheus-compatible monitoring and alerting service that makes it easy to monitor containerized applications and infrastructure at scale. It is a fully-managed service that automatically scales the ingestion, storage, querying, and alerting of your metrics. It also integrates with AWS security services to enable fast and secure access to your data. You can use the open-source PromQL query language to query your metrics and alert on them.

For more information, see [Getting started with Amazon Managed Service for Prometheus](#).

## Using Helm with Amazon EKS

The Helm package manager for Kubernetes helps you install and manage applications on your Kubernetes cluster. For more information, see the [Helm documentation](#). This topic helps you install and run the Helm binaries so that you can install and manage charts using the Helm CLI on your local system.

### Important

Before you can install Helm charts on your Amazon EKS cluster, you must configure `kubectl` to work for Amazon EKS. If you have not already done this, see [Create a kubeconfig for Amazon EKS \(p. 413\)](#) before proceeding. If the following command succeeds for your cluster, you're properly configured.

```
kubectl get svc
```

## To install the Helm binaries on your local system

1. Run the appropriate command for your client operating system.
  - If you're using macOS with [Homebrew](#), install the binaries with the following command.

```
brew install helm
```

- If you're using Windows with [Chocolatey](#), install the binaries with the following command.

```
choco install kubernetes-helm
```

- If you're using Linux, install the binaries with the following commands.

```
curl https://raw.githubusercontent.com/helm/helm/master/scripts/get-helm-3 >
get_helm.sh
chmod 700 get_helm.sh
./get_helm.sh
```

### Note

If you get a message that `openssl` must first be installed, one method to install it would be:

```
sudo yum install openssl
```

2. To pick up the new binary in your `PATH`, Close your current terminal window and open a new one.
3. Confirm that Helm is running with the following command.

```
helm help
```

4. At this point, you can run any Helm commands (such as `helm install <chart_name>`) to install, modify, delete, or query Helm charts in your cluster. If you're new to Helm and don't have a specific chart to install, you can:
  - Experiment by installing an example chart. See [Install an example chart](#) in the Helm [Quickstart guide](#).
  - Create an example chart and push it to Amazon ECR. For more information, see [Pushing a Helm chart](#) in the [Amazon Elastic Container Registry User Guide](#).
  - Install an Amazon EKS chart from the [eks-charts](#) GitHub repo or from [ArtifactHub](#).

## Tagging your Amazon EKS resources

To help you manage your Amazon EKS resources, you can assign your own metadata to each resource using `tags`. This topic provides an overview of the `tags` function and shows how you can create tags.

### Topics

- [Tag basics \(p. 432\)](#)
- [Tagging your resources \(p. 432\)](#)
- [Tag restrictions \(p. 433\)](#)
- [Working with tags using the console \(p. 433\)](#)
- [Working with tags using the CLI, API, or eksctl \(p. 434\)](#)

## Tag basics

A tag is a label that you assign to an AWS resource. Each tag consists of a *key* and an optional *value*, both of which you define.

Tags enable you to categorize your AWS resources by, for example, purpose, owner, or environment. When you have many resources of the same type, you can quickly identify a specific resource based on the tags you've assigned to it. For example, you can define a set of tags for your Amazon EKS clusters to help you track each cluster's owner and stack level. We recommend that you devise a consistent set of tag keys for each resource type. You can then search and filter the resources based on the tags that you add.

Tags are not automatically assigned to your resources. After you add a tag, you can edit tag keys and values or remove tags from a resource at any time. If you delete a resource, any tags for the resource are also deleted.

Tags don't have any semantic meaning to Amazon EKS and are interpreted strictly as a string of characters. You can set the value of a tag to an empty string, but you can't set the value of a tag to null. If you add a tag that has the same key as an existing tag on that resource, the new value overwrites the earlier value.

You can tag new or existing cluster resources using the AWS Management Console, the AWS CLI, or the Amazon EKS API. You can tag only new cluster resources using `eksctl`.

If you use AWS Identity and Access Management (IAM), you can control which users in your AWS account have permission to manage tags.

## Tagging your resources

You can tag new or existing Amazon EKS clusters and managed node groups.

If you're using the Amazon EKS console, then you can apply tags to new or existing resources at any time. You can do this by using the **Tags** tab on the relevant resource page. If you're using `eksctl`, then you can apply tags to resources when they are created using the `--tags` option.

If you're using the Amazon EKS API, the AWS CLI, or an AWS SDK, you can apply tags to new resources using the `tags` parameter on the relevant API action. You can apply tags to existing resources using the `TagResource` API action. For more information, see [TagResource](#).

Some resource-creating actions enable you to specify tags for a resource when the resource is created. If tags cannot be applied while a resource is being created, the resource fails to be created. This mechanism ensures that resources you intended to tag on creation are either created with specified tags or not created at all. If you tag resources at the time of creation, you don't need to run custom tagging scripts after creating a resource.

The following table describes the Amazon EKS resources that can be tagged and the resources that can be tagged on creation.

### Tagging support for Amazon EKS resources

| Resource            | Supports tags | Supports tag propagation                       | Supports tagging on creation (Amazon EKS API, AWS CLI, AWS SDK, and <code>eksctl</code> ) |
|---------------------|---------------|------------------------------------------------|-------------------------------------------------------------------------------------------|
| Amazon EKS clusters | Yes           | No. Cluster tags do not propagate to any other | Yes                                                                                       |

| Resource                       | Supports tags | Supports tag propagation                                                                                                                           | Supports tagging on creation (Amazon EKS API, AWS CLI, AWS SDK, and eksctl) |
|--------------------------------|---------------|----------------------------------------------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------|
|                                |               | resources associated with the cluster.                                                                                                             |                                                                             |
| Amazon EKS managed node groups | Yes           | No. Managed node group tags do not propagate to any other resources associated with the node group.                                                | Yes                                                                         |
| Amazon EKS Fargate profiles    | Yes           | No. Fargate profile tags do not propagate to any other resources associated with the Fargate profile, such as the pods that are scheduled with it. | Yes                                                                         |

## Tag restrictions

The following basic restrictions apply to tags:

- Maximum number of tags per resource – 50
- For each resource, each tag key must be unique, and each tag key can have only one value.
- Maximum key length – 128 Unicode characters in UTF-8
- Maximum value length – 256 Unicode characters in UTF-8
- If your tagging schema is used across multiple AWS services and resources, remember that other services may have restrictions on allowed characters. Generally allowed characters are letters, numbers, spaces representable in UTF-8, and the following characters: + - = . \_ : / @.
- Tag keys and values are case sensitive.
- Don't use aws :, AWS :, or any upper or lowercase combination of such as a prefix for either keys or values. These are reserved only for AWS use. You can't edit or delete tag keys or values with this prefix. Tags with this prefix do not count against your tags-per-resource limit.

## Working with tags using the console

Using the Amazon EKS console, you can manage the tags associated with new or existing clusters and managed node groups.

When you select a resource-specific page in the Amazon EKS console, it displays a list of those resources. For example, if you select Amazon EKS **Clusters** from the left navigation pane, the console displays a list of Amazon EKS clusters. When you select a resource from one of these lists (for example, a specific cluster) that supports tags, you can view and manage its tags on the **Tags** tab.

## Adding tags on an individual resource on creation

You can add tags to Amazon EKS clusters, managed node groups, and Fargate profiles when you create them. For more information, see [Creating an Amazon EKS cluster \(p. 24\)](#).

## Adding and deleting tags on an individual resource

Amazon EKS allows you to add or delete tags associated with your clusters directly from the resource's page.

### To add or delete a tag on an individual resource

1. Open the Amazon EKS console at <https://console.aws.amazon.com/eks/home#/clusters>.
2. From the navigation bar, select the Region to use.
3. In the left navigation pane, choose Amazon EKS **Clusters**.
4. Choose a specific cluster. Select the **Configuration** tab. On the **Tags** tab, choose **Manage tags**.
5. On the **Update tags** page, add or delete your tags as necessary.
  - To add a tag — choose **Add tag** and then specify the key and value for each tag.
  - To delete a tag — choose **Remove tag**.
6. Repeat this process for each tag you want to add or delete, and then choose **Update** to finish.

## Working with tags using the CLI, API, or eksctl

Use the following AWS CLI commands or Amazon EKS API operations to add, update, list, and delete the tags for your resources. You can only use eksctl to add tags to new resources.

### Tagging support for Amazon EKS resources

| Task                               | AWS CLI                     | AWS Tools for Windows PowerShell   | API action                 |
|------------------------------------|-----------------------------|------------------------------------|----------------------------|
| Add or overwrite one or more tags. | <code>tag-resource</code>   | <code>Add-EKSResourceTag</code>    | <code>TagResource</code>   |
| Delete one or more tags.           | <code>untag-resource</code> | <code>Remove-EKSResourceTag</code> | <code>UntagResource</code> |

The following examples show how to tag or untag resources using the AWS CLI.

### Example 1: Tag an existing cluster

The following command tags an existing cluster.

```
aws eks tag-resource --resource-arn <resource_ARN> --tags <team>=<devs>
```

### Example 2: Untag an existing cluster

The following command deletes a tag from an existing cluster.

```
aws eks untag-resource --resource-arn <resource_ARN> --tag-keys <tag_key>
```

### Example 3: List tags for a resource

The following command lists the tags associated with an existing resource.

```
aws eks list-tags-for-resource --resource-arn <resource_ARN>
```

Some resource-creating actions enable you to specify tags when you create the resource. The following actions support tagging when creating a resource.

| Task                         | AWS CLI                             | AWS Tools for Windows PowerShell   | API action                        | eksctl                             |
|------------------------------|-------------------------------------|------------------------------------|-----------------------------------|------------------------------------|
| Create a cluster             | <code>create-cluster</code>         | <code>New-EKSCluster</code>        | <code>CreateCluster</code>        | <code>create-cluster</code>        |
| Create a managed node group* | <code>create-nodegroup</code>       | <code>New-EKSNodegroup</code>      | <code>CreateNodegroup</code>      | <code>create-nodegroup</code>      |
| Create a Fargate profile     | <code>create-fargate-profile</code> | <code>New-EKSFargateProfile</code> | <code>CreateFargateProfile</code> | <code>create-fargateprofile</code> |

\*If you want to also tag the Amazon EC2 instances when creating a managed node group, create the managed node group using a launch template. For more information, see [Tagging Amazon EC2 instances \(p. 126\)](#). If your instances already exist, you can manually tag the instances. For more information, see [Tagging your resources](#) in the Amazon EC2 User Guide for Linux Instances.

## Amazon EKS service quotas

Amazon EKS has integrated with Service Quotas, an AWS service that enables you to view and manage your quotas from a central location. For more information, see [What Is Service Quotas?](#) in the *Service Quotas User Guide*. Service Quotas makes it easy to look up the value of your Amazon EKS and AWS Fargate service quotas using the AWS Management Console and AWS CLI.

### AWS Management Console

#### To view Amazon EKS and Fargate service quotas using the AWS Management Console

1. Open the Service Quotas console at <https://console.aws.amazon.com/servicequotas/>.
2. In the left navigation pane, choose **AWS services**.
3. From the **AWS services** list, search for and select **Amazon Elastic Kubernetes Service (Amazon EKS) or AWS Fargate**.

In the **Service quotas** list, you can see the service quota name, applied value (if it is available), AWS default quota, and whether the quota value is adjustable.

4. To view additional information about a service quota, such as the description, choose the quota name.
5. (Optional) To request a quota increase, select the quota that you want to increase, select **Request quota increase**, enter or select the required information, and select **Request**.

To work more with service quotas using the AWS Management Console see the [Service Quotas User Guide](#). To request a quota increase, see [Requesting a Quota Increase](#) in the *Service Quotas User Guide*.

### AWS CLI

#### To view Amazon EKS and Fargate service quotas using the AWS CLI

Run the following command to view your Amazon EKS quotas.

```
aws service-quotas list-aws-default-service-quotas \
```

```
--query 'Quotas[*].
{Adjustable:Adjustable,Name:QuotaName,Value:Value,Code:QuotaCode}' \
--service-code eks \
--output table
```

Run the following command to view your Fargate quotas.

```
aws service-quotas list-aws-default-service-quotas \
--query 'Quotas[*].
{Adjustable:Adjustable,Name:QuotaName,Value:Value,Code:QuotaCode}' \
--service-code fargate \
--output table
```

#### Note

The quota returned is the maximum number of Amazon ECS tasks or Amazon EKS pods running concurrently on Fargate in this account in the current AWS Region.

To work more with service quotas using the AWS CLI, see the [Service Quotas AWS CLI Command Reference](#). To request a quota increase, see the [request-service-quota-increase](#) command in the [AWS CLI Command Reference](#).

## Service quotas

| Name                                           | Default                    | Adjust | Description                                                                                                                       |
|------------------------------------------------|----------------------------|--------|-----------------------------------------------------------------------------------------------------------------------------------|
| Clusters                                       | Each supported Region: 100 | Yes    | The maximum number of EKS clusters in this account in the current Region.                                                         |
| Control plane security groups per cluster      | Each supported Region: 4   | No     | The maximum number of control plane security groups per cluster (these are specified when you create the cluster).                |
| Fargate profiles per cluster                   | Each supported Region: 10  | Yes    | The maximum number of Fargate profiles per cluster.                                                                               |
| Label pairs per Fargate profile selector       | Each supported Region: 5   | Yes    | The maximum number of label pairs per Fargate profile selector.                                                                   |
| Managed node groups per cluster                | Each supported Region: 30  | Yes    | The maximum number of managed node groups per cluster.                                                                            |
| Nodes per managed node group                   | Each supported Region: 450 | Yes    | The maximum number of nodes per managed node group.                                                                               |
| Public endpoint access CIDR ranges per cluster | Each supported Region: 40  | No     | The maximum number of public endpoint access CIDR ranges per cluster (these are specified when you create or update the cluster). |

| Name                          | Default                   | Adjust | Description                                                                      |
|-------------------------------|---------------------------|--------|----------------------------------------------------------------------------------|
| Registered clusters           | Each supported Region: 10 | Yes    | The maximum number of registered clusters in this account in the current Region. |
| Selectors per Fargate profile | Each supported Region: 5  | Yes    | The maximum number of selectors per Fargate profile.                             |

## AWS Fargate service quotas

The following are Amazon EKS on AWS Fargate service quotas.

These service quotas are listed under the AWS Fargate namespace in the Service Quotas console. To request a quota increase, see [Requesting a quota increase](#) in the *Service Quotas User Guide*.

| Service quota                    | Description                                                                                                                           | Default quota value | Adjustable |
|----------------------------------|---------------------------------------------------------------------------------------------------------------------------------------|---------------------|------------|
| Fargate On-Demand resource count | The maximum number of Amazon ECS tasks and Amazon EKS pods running concurrently on Fargate in this account in the current AWS Region. | 1,000               | Yes        |

# Security in Amazon EKS

Cloud security at AWS is the highest priority. As an AWS customer, you benefit from a data center and network architecture that is built to meet the requirements of the most security-sensitive organizations.

Security is a shared responsibility between AWS and you. The [shared responsibility model](#) describes this as security *of* the cloud and security *in* the cloud:

- **Security of the cloud** – AWS is responsible for protecting the infrastructure that runs AWS services in the AWS Cloud. For Amazon EKS, AWS is responsible for the Kubernetes control plane, which includes the control plane nodes and etcd database. Third-party auditors regularly test and verify the effectiveness of our security as part of the [AWS compliance programs](#). To learn about the compliance programs that apply to Amazon EKS, see [AWS Services in Scope by Compliance Program](#).
- **Security in the cloud** – Your responsibility includes the following areas.
  - The security configuration of the data plane, including the configuration of the security groups that allow traffic to pass from the Amazon EKS control plane into the customer VPC
  - The configuration of the nodes and the containers themselves
  - The node's operating system (including updates and security patches)
  - Other associated application software:
    - Setting up and managing network controls, such as firewall rules
    - Managing platform-level identity and access management, either with or in addition to IAM
  - The sensitivity of your data, your company's requirements, and applicable laws and regulations

This documentation helps you understand how to apply the shared responsibility model when using Amazon EKS. The following topics show you how to configure Amazon EKS to meet your security and compliance objectives. You also learn how to use other AWS services that help you to monitor and secure your Amazon EKS resources.

## Note

Linux containers are made up of control groups (cgroups) and namespaces that help limit what a container can access, but all containers share the same Linux kernel as the host Amazon EC2 instance. Running a container as the root user (UID 0) or granting a container access to host resources or namespaces such as the host network or host PID namespace are strongly discouraged, because doing so reduces the effectiveness of the isolation that containers provide.

## Topics

- [Certificate signing \(p. 439\)](#)
- [Kubernetes service accounts \(p. 440\)](#)
- [Identity and access management for Amazon EKS \(p. 455\)](#)
- [Compliance validation for Amazon Elastic Kubernetes Service \(p. 498\)](#)
- [Resilience in Amazon EKS \(p. 499\)](#)
- [Infrastructure security in Amazon EKS \(p. 500\)](#)
- [Configuration and vulnerability analysis in Amazon EKS \(p. 500\)](#)
- [Security best practices for Amazon EKS \(p. 501\)](#)
- [Pod security policy \(p. 501\)](#)
- [Using AWS Secrets Manager secrets with Kubernetes \(p. 504\)](#)

- [Amazon EKS Connector considerations \(p. 504\)](#)

## Certificate signing

The Kubernetes Certificates API enables automation of [X.509](#) credential provisioning. It features a command line interface for Kubernetes API clients to request and obtain [X.509 certificates](#) from a Certificate Authority (CA). A `CertificateSigningRequest` (CSR) resource is used to request that a denoted signer sign the certificate. Then, the request is either approved or denied before it's signed. Kubernetes supports both build-in signers and custom signers with well-defined behaviors. This way, clients can predict what happens to their CSRs. To learn more about certificate signing, see [signing requests](#).

One of the built-in signers is `kubernetes.io/legacy-unknown`. The v1beta1 API of CSR resource honored this legacy-unknown signer. However, the stable v1 API of CSR doesn't allow the `signerName` to be set to `kubernetes.io/legacy-unknown`.

Amazon EKS version 1.21 and earlier versions allowed the value `legacy-unknown` as the `signerName` in v1beta1 CSR API. This API enables the Amazon EKS Certificate Authority (CA) to generate certificates. However, in Kubernetes version 1.22, the v1beta1 CSR API is replaced by the v1 CSR API. This API doesn't support the `signerName` of "legacy-unknown." Given this, if you want to use Amazon EKS CA for generating certificates on version 1.22 and later, you need to use a [custom signer](#). It was introduced in Amazon EKS version 1.22. To use the CSR v1 API version and generate a new certificate, you must migrate any existing manifests and API clients. Existing certificates that were created with the older v1beta1 API are valid and function until the certificate expires. This includes the following:

1. Trust distribution: None. There's no standard trust or distribution for this signer in a Kubernetes cluster.
2. Permitted subjects: Any
3. Permitted x509 extensions: Honors `subjectAltName` and `key usage` extensions and discards other extensions
4. Permitted key usages: Must not include usages beyond ["key encipherment", "digital signature", "server auth"]
5. Expiration/certificate lifetime: 1 year (default and maximum)
6. CA bit allowed/disallowed: Not allowed

## Example CSR generation with signerName

These steps shows how to generate a serving certificate for DNS name `myserver.default.svc` using `signerName: beta.eks.amazonaws.com/app-serving`. Use this as a guide for your own environment.

1. Run the `openssl genrsa -out myserver.key 2048` command to generate an RSA private key.

```
openssl genrsa -out myserver.key 2048
```

2. Run the following command to generate a certificate request.

```
openssl req -new -key myserver.key -out myserver.csr -subj "/CN=myserver.default.svc"
```

3. Generate a base64 value for the CSR request. Later, you'll use this value for the `request` value in your CSR.

```
cat myserver.csr | base64 -w 0 | tr -d "\n"
```

4. Create a file named `mycsr.yaml` with the following contents. In the following example, `beta.eks.amazonaws.com/app-serving` is the `signerName`. Replace `base64-value` with the value returned in the previous step.

```
apiVersion: certificates.k8s.io/v1
kind: CertificateSigningRequest
metadata:
 name: myserver
spec:
 request: base64-value
 signerName: beta.eks.amazonaws.com/app-serving
 usages:
 - digital signature
 - key encipherment
 - server auth
```

5. Submit the CSR.

```
kubectl apply -f mycsr.yaml
```

6. Approve the serving certificate.

```
kubectl certificate approve myserver
```

7. Verify that the certificate was issued.

```
kubectl get csr myserver
```

Example output:

| NAME     | AGE   | SIGNERNAME                                             | REQUESTOR        | CONDITION |
|----------|-------|--------------------------------------------------------|------------------|-----------|
| myserver | 3m20s | beta.eks.amazonaws.com/app-serving<br>Approved, Issued | kubernetes-admin |           |

8. Export the issued certificate:

```
kubectl get csr myserver -o jsonpath='{.status.certificate}' | base64 -d > myserver.crt
```

## Kubernetes service accounts

A Kubernetes service account provides an identity for processes that run in a pod. For more information see [Managing Service Accounts](#) in the Kubernetes documentation. If your pod needs access to AWS services, you can map the service account to an AWS Identity and Access Management identity to grant that access. For more information, see [IAM roles for service accounts \(p. 442\)](#).

## Service account tokens

The `BoundServiceAccountTokenVolume` feature is enabled by default in Kubernetes version 1.21 and later. This feature improves the 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 have an expiration of one hour. In earlier Kubernetes versions, the tokens 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 v0.15.7 and later

- Python v12.0.0 and later
- Java v9.0.0 and later
- JavaScript v0.10.3 and later
- Ruby master branch
- Haskell v0.3.0.0
- C# v7.0.5 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 and later 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 client SDKs are the same or later than the versions listed above.

When the API server receives requests with tokens that are older than one hour, it annotates the API audit log event with `annotations.authentication.k8s.io/stale-token`. The value of the annotation looks like the following example:

```
subject: system:serviceaccount:common:fluent-bit, seconds after warning threshold: 4185802.
```

If your cluster has [control plane logging \(p. 64\)](#) enabled, then the annotations are in the audit logs. You can use the following [Cloudwatch Log Insights](#) query to identify all the pods in your Amazon EKS cluster that are using stale tokens:

```
fields @timestamp
| filter @logStream like /kube-apiserver-audit/
| filter @message like /seconds after warning threshold/
| parse @message "subject: *, seconds after warning threshold:*\\" as subject, elapsedtime
```

The `subject` refers to the service account that the pod used. The `elapsedtime` indicates the elapsed time (in seconds) after reading the latest token. The requests to the API server are denied when the `elapsedtime` exceeds 90 days. You should proactively update your applications' Kubernetes client SDK to use one of the version listed above that automatically refresh the token. If the service account token used is close to 90 days and you don't have sufficient time to update your client SDK versions before token expiration, then you can terminate existing pods and create new ones. This results in refetching of the service account token, giving you an additional 90 days to update your client version SDKs.

If the pod is part of a deployment, the suggested way to terminate pods while keeping high availability is to perform a roll out with the following command. Replace `my-deployment` with the name of your deployment.

```
kubectl rollout restart deployment/my-deployment
```

## Cluster add-ons

The following cluster add-ons have been updated to use the Kubernetes client SDKs that automatically refetch service account tokens. We recommend making sure that the listed versions, or later versions, are installed on your 1.21 or later cluster.

- Amazon VPC CNI and CNI metrics helper plugins version 1.8.0 and later. To check your current version or update it, see [Managing the Amazon VPC CNI add-on \(p. 281\)](#) and [CNI metrics helper \(p. 324\)](#).
- CoreDNS version 1.8.4 and later. To check your current version or update it, see [Managing the CoreDNS add-on \(p. 337\)](#).

- AWS Load Balancer Controller version 2.0.0 and later. To check your current version or update it, see [Installing the AWS Load Balancer Controller add-on \(p. 329\)](#).
- kube-proxy version 1.21.2-eksbuild.2 and later. To check your current version or update it, see [Managing the kube-proxy add-on \(p. 343\)](#).
- AWS for Fluent Bit version 2.25.0 or later. To update your current version, see [Releases](#) on GitHub.

## IAM roles for service accounts

You can associate an IAM role with a Kubernetes service account. This service account can then provide AWS permissions to the containers in any pod that uses that service account. With this feature, you no longer need to provide extended permissions to the [Amazon EKS node IAM role \(p. 474\)](#) so that pods on that node can call AWS APIs.

Applications must sign their AWS API requests with AWS credentials. This feature provides a strategy for managing credentials for your applications, similar to the way that Amazon EC2 instance profiles provide credentials to Amazon EC2 instances. Instead of creating and distributing your AWS credentials to the containers or using the Amazon EC2 instance's role, you can associate an IAM role with a Kubernetes service account. The applications in the pod's containers can then use an AWS SDK or the AWS CLI to make API requests to authorized AWS services.

### Important

Even if you assign an IAM role to a Kubernetes service account, the pod still also has the permissions assigned to the [Amazon EKS node IAM role \(p. 474\)](#), unless you block pod access to the IMDS. For more information, see [Restrict access to the instance profile assigned to the worker node](#).

The IAM roles for service accounts feature provides the following benefits:

- **Least privilege** — By using the IAM roles for service accounts feature, you no longer need to provide extended permissions to the node IAM role so that pods on that node can call AWS APIs. You can scope IAM permissions to a service account, and only pods that use that service account have access to those permissions. This feature also eliminates the need for third-party solutions such as `kiam` or `kube2iam`.
- **Credential isolation** — A container can only retrieve credentials for the IAM role that is associated with the service account to which it belongs. A container never has access to credentials that are intended for another container that belongs to another pod.
- **Auditability** — Access and event logging is available through CloudTrail to help ensure retrospective auditing.

### Enable service accounts to access AWS resources in three steps

1. [Create an IAM OIDC provider for your cluster \(p. 446\)](#) – You only need to do this once for a cluster.
2. [Create an IAM role and attach an IAM policy to it with the permissions that your service accounts need \(p. 447\)](#) – We recommend creating separate roles for each unique collection of permissions that pods need.
3. [Associate an IAM role with a service account \(p. 450\)](#) – Complete this task for each Kubernetes service account that needs access to AWS resources.
4. [Configure the AWS Security Token Service endpoint type for a service account \(p. 452\)](#) – You can optionally complete this task for a service account if your cluster is version 1.18 or later.

## Technical overview

In 2014, AWS Identity and Access Management added support for federated identities using OpenID Connect (OIDC). This feature allows you to authenticate AWS API calls with supported identity providers and receive a valid OIDC JSON web token (JWT). You can pass this token to the AWS STS

AssumeRoleWithWebIdentity API operation and receive IAM temporary role credentials. You can use these credentials to interact with any AWS service, like Amazon S3 and DynamoDB.

Kubernetes has long used service accounts as its own internal identity system. Pods can authenticate with the Kubernetes API server using an auto-mounted token (which was a non-OIDC JWT) that only the Kubernetes API server could validate. These legacy service account tokens do not expire, and rotating the signing key is a difficult process. In Kubernetes version 1.12, support was added for a new `ProjectedServiceAccountToken` feature, which is an OIDC JSON web token that also contains the service account identity, and supports a configurable audience.

Amazon EKS now hosts a public OIDC discovery endpoint per cluster containing the signing keys for the `ProjectedServiceAccountToken` JSON web tokens so external systems, like IAM, can validate and accept the OIDC tokens issued by Kubernetes.

## IAM role configuration

In IAM, you create an IAM role with a trust relationship that is scoped to your cluster's OIDC provider, the service account namespace, and (optionally) the service account name, and then attach the IAM policy that you want to associate with the service account. You can add multiple entries in the `StringEquals` and `StringLike` conditions below to use multiple service accounts or namespaces with the role.

- To scope a role to a specific service account:

```
{
 "Version": "2012-10-17",
 "Statement": [
 {
 "Effect": "Allow",
 "Principal": {
 "Federated": "arn:aws:iam::111122223333:oidc-provider/OIDC_PROVIDER"
 },
 "Action": "sts:AssumeRoleWithWebIdentity",
 "Condition": {
 "StringEquals": {
 "OIDC_PROVIDER:sub": "system:serviceaccount:SERVICE_ACCOUNT_NAMESPACE:SERVICE_ACCOUNT_NAME"
 }
 }
 }
]
}
```

- To scope a role to an entire namespace (to use the namespace as a boundary):

```
{
 "Version": "2012-10-17",
 "Statement": [
 {
 "Effect": "Allow",
 "Principal": {
 "Federated": "arn:aws:iam::111122223333:oidc-provider/OIDC_PROVIDER"
 },
 "Action": "sts:AssumeRoleWithWebIdentity",
 "Condition": {
 "StringLike": {
 "OIDC_PROVIDER:sub": "system:serviceaccount:SERVICE_ACCOUNT_NAMESPACE:/*"
 }
 }
 }
]
}
```

## Service account configuration

In Kubernetes, you define the IAM role to associate with a service account in your cluster by adding the `eks.amazonaws.com/role-arn` annotation to the service account.

```
apiVersion: v1
kind: ServiceAccount
metadata:
 annotations:
 eks.amazonaws.com/role-arn: arn:aws:iam::111122223333:role/iam-role-name
```

## Pod configuration

The [Amazon EKS Pod Identity Webhook](#) on the cluster watches for pods that are associated with service accounts with this annotation and applies the following environment variables to them.

```
AWS_ROLE_ARN=arn:aws:iam::111122223333:role/iam-role-name
AWS_WEB_IDENTITY_TOKEN_FILE=/var/run/secrets/eks.amazonaws.com/serviceaccount/token
```

### Note

Your cluster does not need to use the mutating web hook to configure the environment variables and token file mounts; you can choose to configure pods to add these environment variables manually.

[Supported versions of the AWS SDK \(p. 454\)](#) look for these environment variables first in the credential chain provider. The role credentials are used for pods that meet this criteria.

### Note

When a pod uses AWS credentials from an IAM role associated with a service account, the AWS CLI or other SDKs in the containers for that pod use the credentials provided by that role. The pod still has access to the credentials provided to the [Amazon EKS node IAM role \(p. 474\)](#) too, unless you restrict access to those credentials. For more information, see [Restrict access to the instance profile assigned to the worker node](#).

By default, only containers that run as `root` have the proper file system permissions to read the web identity token file. You can provide these permissions by having your containers run as `root`, or by providing the following security context for the containers in your manifest. The `fsGroup` ID is arbitrary, and you can choose any valid group ID. For more information about the implications of setting a security context for your pods, see [Configure a Security Context for a Pod or Container](#) in the Kubernetes documentation.

### Note

Providing this security context is not required for 1.19 or later clusters.

```
apiVersion: apps/v1
kind: Deployment
metadata:
 name: my-app
spec:
 template:
 metadata:
 labels:
 app: my-app
 spec:
 serviceAccountName: my-app
 containers:
 - name: my-app
 image: my-app:latest
```

```

 securityContext:
 fsGroup: 1337
...

```

The `kubelet` requests and stores the token on behalf of the pod. By default, the `kubelet` refreshes the token if it is older than 80 percent of its total TTL, or if the token is older than 24 hours. You can modify the expiration duration for any account, except the default service account, with settings in your pod spec. For more information, see [Service Account Token Volume Projection](#) in the Kubernetes documentation.

## Cross-account IAM permissions

You can configure cross-account IAM permissions either by creating an identity provider from another account's cluster or by using chained `AssumeRole` operations. In the following examples, Account A owns an Amazon EKS cluster that supports IAM roles for service accounts. Pods running on that cluster need to assume IAM permissions from Account B.

### **Example : Create an identity provider from another account's cluster**

#### **Example**

In this example, Account A would provide Account B with the OIDC issuer URL from their cluster. Account B follows the instructions in [Create an IAM OIDC provider for your cluster \(p. 446\)](#) and [Creating an IAM role and policy for your service account \(p. 447\)](#) using the OIDC issuer URL from Account A's cluster. Then a cluster administrator annotates the service account in Account A's cluster to use the role from Account B ([444455556666](#)).

```

apiVersion: v1
kind: ServiceAccount
metadata:
 annotations:
 eks.amazonaws.com/role-arn: arn:aws:iam::444455556666:role/account-b-role

```

### **Example : Use chained `AssumeRole` operations**

#### **Example**

In this example, Account B creates an IAM policy with the permissions to give to pods in Account A's cluster. Account B ([444455556666](#)) attaches that policy to an IAM role with a trust relationship that allows `AssumeRole` permissions to Account A ([111122223333](#)), as shown below.

```

{
 "Version": "2012-10-17",
 "Statement": [
 {
 "Effect": "Allow",
 "Principal": {
 "AWS": "arn:aws:iam::111122223333:root"
 },
 "Action": "sts:AssumeRole",
 "Condition": {}
 }
]
}

```

Account A creates a role with a trust policy that gets credentials from the identity provider created with the cluster's OIDC issuer URL, as shown below.

```
{
 "Version": "2012-10-17",
 "Statement": [
 {
 "Effect": "Allow",
 "Principal": {
 "Federated": "arn:aws:iam::111122223333:oidc-provider/oidc.eks.region-
code.amazonaws.com/id/EXAMPLED539D4633E53DE1B71EXAMPLE"
 },
 "Action": "sts:AssumeRoleWithWebIdentity"
 }
]
}
```

Account A attaches a policy to that role with the following permissions to assume the role that Account B created.

```
{
 "Version": "2012-10-17",
 "Statement": [
 {
 "Effect": "Allow",
 "Action": "sts:AssumeRole",
 "Resource": "arn:aws:iam::444455556666:role/account-b-role"
 }
]
}
```

The application code for pods to assume Account B's role uses two profiles: `account_b_role` and `account_a_role`. The `account_b_role` profile uses the `account_a_role` profile as its source. For the AWS CLI, the `~/.aws/config` file would look like the following example.

```
[profile account_b_role]
source_profile = account_a_role
role_arn=arn:aws:iam::444455556666:role/account-b-role

[profile account_a_role]
web_identity_token_file = /var/run/secrets/eks.amazonaws.com/serviceaccount/token
role_arn=arn:aws:iam::111122223333:role/account-a-role
```

To specify chained profiles for other AWS SDKs, consult their documentation.

## Create an IAM OIDC provider for your cluster

Your cluster has an [OpenID Connect](#) issuer URL associated with it. To use IAM roles for service accounts, an IAM OIDC provider must exist for your cluster.

### Prerequisites

An existing cluster. If you don't have one, you can create one using one of the [Getting started with Amazon EKS \(p. 4\)](#) guides.

You can create an OIDC provider for your cluster using `eksctl` or the AWS Management Console.

`eksctl`

### To create an IAM OIDC identity provider for your cluster with `eksctl`

1. Determine whether you have an existing IAM OIDC provider for your cluster.

View your cluster's OIDC provider URL.

```
aws eks describe-cluster --name my-cluster --query "cluster.identity.oidc.issuer"
--output text
```

Example output:

```
https://oidc.eks.region-code.amazonaws.com/id/EXAMPLED539D4633E53DE1B71EXAMPLE
```

List the IAM OIDC providers in your account. Replace *EXAMPLED539D4633E53DE1B71EXAMPLE* with the value returned from the previous command.

```
aws iam list-open-id-connect-providers | grep EXAMPLED539D4633E53DE1B71EXAMPLE
```

Example output:

```
"Arn": "arn:aws:iam::111122223333:oidc-provider/oidc.eks.region-code.amazonaws.com/
id/EXAMPLED539D4633E53DE1B71EXAMPLE"
```

If output is returned from the previous command, then you already have a provider for your cluster. If no output is returned, then you must create an IAM OIDC provider.

2. Create an IAM OIDC identity provider for your cluster with the following command. Replace *my-cluster* with your own value.

```
eksctl utils associate-iam-oidc-provider --cluster my-cluster --approve
```

## AWS Management Console

### To create an IAM OIDC identity provider for your cluster with the AWS Management Console

1. Open the Amazon EKS console at <https://console.aws.amazon.com/eks/home#/clusters>.
2. Select the name of your cluster and then select the **Configuration** tab.
3. In the **Details** section, note the value of the **OpenID Connect provider URL**.
4. Open the IAM console at <https://console.aws.amazon.com/iam/>.
5. In the left navigation pane, choose **Identity Providers** under **Access management**. If a **Provider** is listed that matches the URL for your cluster, then you already have a provider for your cluster. If a provider isn't listed that matches the URL for your cluster, then you must create one.
6. To create a provider, choose **Add Provider**.
7. For **Provider Type**, choose **OpenID Connect**.
8. For **Provider URL**, paste the OIDC issuer URL for your cluster, and then choose **Get thumbprint**.
9. For **Audience**, enter `sts.amazonaws.com` and choose **Add provider**.

## Creating an IAM role and policy for your service account

You must create an IAM policy that specifies the permissions that you would like the containers in your pods to have. You have several ways to create a new IAM permission policy. One way is to copy a complete AWS managed policy that already does some of what you're looking for and then customize it to your specific requirements. For more information, see [Creating a New Policy](#) in the *IAM User Guide*.

You must also create an IAM role for your Kubernetes service accounts to use before you associate it with a service account. The trust relationship is scoped to your cluster and service account so that each cluster and service account combination requires its own role. You can then attach a specific IAM policy to the role that gives the containers in your pods the permissions you desire. The following procedures describe how to do this.

## Create an IAM policy

In this procedure, we offer two example policies that you can use for your application:

- A policy to allow read-only access to an Amazon S3 bucket. You could store configuration information or a bootstrap script in this bucket, and the containers in your pod can read the file from the bucket and load it into your application.
- A policy to allow paid container images from AWS Marketplace.

1. Open the IAM console at <https://console.aws.amazon.com/iam/>.
2. In the left navigation pane, choose **Policies** and then choose **Create policy**.
3. Choose the **JSON** tab.
4. In the **Policy Document** field, paste one of the following policies to apply to your service accounts, or paste your own policy document into the field. You can also use the visual editor to construct your own policy.

The example below allows permission to the *my-pod-secrets-bucket* Amazon S3 bucket. You can modify the policy document to suit your specific needs.

```
{
 "Version": "2012-10-17",
 "Statement": [
 {
 "Effect": "Allow",
 "Action": [
 "s3:GetObject"
],
 "Resource": [
 "arn:aws:s3:::my-pod-secrets-bucket/*"
]
 }
]
}
```

The example below gives the required permissions to use a paid container image from AWS Marketplace.

```
{
 "Version": "2012-10-17",
 "Statement": [
 {
 "Action": [
 "aws-marketplace:RegisterUsage"
],
 "Effect": "Allow",
 "Resource": "*"
 }
]
}
```

5. Choose **Review policy**.
6. Enter a name and description for your policy and then choose **Create policy**.

7. Record the Amazon Resource Name (ARN) of the policy to use later when you create your role.

## Create an IAM role for a service account

Create an IAM role for your Kubernetes service account. You can use `eksctl`, the AWS Management Console, or the AWS CLI to create the role.

### Prerequisites

- An existing cluster. If you don't have one, you can create one using one of the [Getting started with Amazon EKS \(p. 4\)](#) guides.
- If using the AWS Management Console or AWS CLI to create the role, then you must have an existing IAM OIDC provider for your cluster. For more information, see [Create an IAM OIDC provider for your cluster \(p. 446\)](#).
- An existing IAM policy that includes the permissions for the AWS resources that your service account needs access to. For more information, see [Create an IAM policy \(p. 448\)](#).

You can create the IAM role with `eksctl` or the AWS CLI.

`eksctl`

Create the service account and IAM role with the following command. Replace `my-service-account` and `my-namespace` with the Kubernetes service account and namespace that you want to associate the IAM role to. Replace `my-cluster` with the name of your cluster and replace `my-role-name` with the name you want for the IAM role. Replace `111122223333` with your account ID (or with `aws`, if you're attaching an AWS managed policy) and replace `my-iam-policy` with the name of an existing IAM policy that you created or with the name of a managed AWS IAM policy.

```
eksctl create iamserviceaccount \
 --name my-service-account \
 --namespace my-namespace \
 --cluster my-cluster \
 --role-name "my-role-name" \
 --attach-policy-arn arn:aws:iam::111122223333:policy/my-iam-policy \
 --approve \
 --override-existing-serviceaccounts
```

The `eksctl` utility creates and deploys an AWS CloudFormation template. The template creates an IAM role and attaches the IAM policy that you specified to it. The role is associated with the Kubernetes service account that you specified. If the Kubernetes service account didn't exist, `eksctl` created it in the namespace that you specified and annotated it with `eks.amazonaws.com/role-arn:arn:aws:iam::111122223333:role/my-role-name`. This is the name of the IAM role created by the AWS CloudFormation template.

If the Kubernetes service account existed before running the command, but wasn't created with `eksctl`, then `eksctl` annotated the existing service account with `eks.amazonaws.com/role-arn:arn:aws:iam::111122223333:role/my-role-name`. If the Kubernetes service account existed before running the command and was created by `eksctl`, the command fails. To update a service account created by `eksctl`, use `eksctl update iamserviceaccount`.

### AWS CLI

1. Set your AWS account ID to an environment variable with the following command.

```
ACCOUNT_ID=$(aws sts get-caller-identity --query "Account" --output text)
```

2. Set your OIDC identity provider to an environment variable with the following command. Replace `my-cluster` with the name of your cluster.

### Important

You must use at least version 1.23.11 or 2.6.3 of the AWS CLI to receive the proper output from this command. For more information, see [Installing the AWS CLI](#) in the [AWS Command Line Interface User Guide](#).

```
OIDC_PROVIDER=$(aws eks describe-cluster --name my-cluster --query "cluster.identity.oidc.issuer" --output text | sed -e "s/^https:\/\///")
```

3. Copy the following code block to your computer. Replace *my-namespace* and *my-service-account* with the Kubernetes namespace and service account that you want to associate the role to.

```
read -r -d '' TRUST_RELATIONSHIP <<EOF
{
 "Version": "2012-10-17",
 "Statement": [
 {
 "Effect": "Allow",
 "Principal": {
 "Federated": "arn:aws:iam::${ACCOUNT_ID}:oidc-provider/${OIDC_PROVIDER}"
 },
 "Action": "sts:AssumeRoleWithWebIdentity",
 "Condition": {
 "StringEquals": {
 "${OIDC_PROVIDER}:aud": "sts.amazonaws.com",
 "${OIDC_PROVIDER}:sub": "system:serviceaccount:my-namespace:my-service-account"
 }
 }
]
 }
EOF
echo "${TRUST_RELATIONSHIP}" > trust.json
```

4. Run the modified code block from the previous step to create a file named *trust.json*.
5. Run the following AWS CLI command to create the role. Replace *my-iam-role* with a name for your IAM role, and *my-role-description* with a description for your role.

```
aws iam create-role --role-name my-iam-role --assume-role-policy-document file://trust.json --description "my-role-description"
```

6. Run the following command to attach an IAM policy to your role. Replace *my-iam-role* with the name of your IAM role, *111122223333* with your account ID (or with *aws*, if you're attaching an AWS managed policy), and *my-iam-policy* with the name of an existing policy that you created or an IAM AWS managed policy.

```
aws iam attach-role-policy --role-name my-iam-role --policy-arn=arn:aws:iam::111122223333:policy/my-iam-policy
```

## Associate an IAM role to a service account

In Kubernetes, you define the IAM role to associate with a service account in your cluster by adding the following annotation to the service account.

```
apiVersion: v1
kind: ServiceAccount
metadata:
```

```
annotations:
 eks.amazonaws.com/role-arn: arn:aws:iam::111122223333:role/iam-role-name
```

**Note**

If you [created an IAM role to use with your service account \(p. 449\)](#) using eksctl, eksctl annotated the service account that you specified when creating the role.

## Prerequisites

- An existing cluster. If you don't have one, you can create one using one of the [Getting started with Amazon EKS \(p. 4\)](#) guides.
- An existing IAM OIDC provider for your cluster. For more information, see [Create an IAM OIDC provider for your cluster \(p. 446\)](#).
- An existing Kubernetes service account. If you don't have one, see [Configure Service Accounts for Pods](#) in the Kubernetes documentation.
- An existing IAM role with an attached IAM policy. If you don't have one, see [Creating an IAM role and policy for your service account \(p. 447\)](#).

## To annotate a service account with an IAM role

1. Use the following command to annotate your service account with the ARN of the IAM role that you want to use with your service account. Replace `service-account-namespace` with the Kubernetes namespace of your service account, `service-account-name` with the name of your existing Kubernetes service account. Replace `111122223333` with your AWS account ID and `iam-role-name` with the name of your existing AWS Identity and Access Management (IAM) role.

```
kubectl annotate serviceaccount -n service-account-namespace service-account-name \
eks.amazonaws.com/role-arn=arn:aws:iam::111122223333:role/iam-role-name
```

**Note**

If you don't have an existing service account, then you need to create one. For more information, see [Configure Service Accounts for Pods](#) in the Kubernetes documentation. For the service account to be able to use Kubernetes permissions, you must create a Role, or ClusterRole and then bind the role to the service account. For more information, see [Using RBAC Authorization](#) in the Kubernetes documentation. When the [AWS VPC CNI plugin \(p. 276\)](#) is deployed, for example, the deployment manifest creates a service account, cluster role, and cluster role binding. You can view the [manifest](#) on GitHub to use as an example.

2. Delete and re-create any existing pods that are associated with the service account to apply the credential environment variables. The mutating web hook does not apply them to pods that are already running. For example, if you added the annotation to the service account used for the Amazon VPC CNI DaemonSet in a previous step, the following command deletes the existing `aws-node` DaemonSet pods and deploys them with the service account annotation. You can replace `pods`, `kube-system`, and `-l k8s-app=aws-node` with the information for the pods that you set your annotation for.

```
kubectl delete pods -n kube-system -l k8s-app=aws-node
```

3. Confirm that the pods all restarted.

```
kubectl get pods -n kube-system -l k8s-app=aws-node
```

4. View the environment variables for one of the pods and verify that the `AWS_WEB_IDENTITY_TOKEN_FILE` and `AWS_ROLE_ARN` environment variables exist. The following example command returns the variables for one of the pods created by the Amazon VPC CNI

DaemonSet. The pod named `aws-node-5v6ws` was returned in the output of the example used in the previous step.

```
kubectl exec -n kube-system aws-node-5v6ws -- env | grep AWS
```

Output:

```
...
AWS_WEB_IDENTITY_TOKEN_FILE=/var/run/secrets/eks.amazonaws.com/serviceaccount/token
...
AWS_ROLE_ARN=arn:aws:iam::111122223333:role/iam-role-name
...
```

5. (Optional) Depending on the version of your cluster, pods may be using the AWS Security Token Service regional or global endpoint. AWS recommends using regional endpoints instead of the global endpoint to reduce latency, build in redundancy, and increase session token validity. For more information about how to determine which endpoint type that your pods are using or to change the endpoint type, see [Configure the AWS Security Token Service endpoint for a service account \(p. 452\)](#).

## Configure the AWS Security Token Service endpoint for a service account

If you're using a Kubernetes service account with [IAM roles for service accounts \(p. 442\)](#), then you can configure the type of AWS Security Token Service endpoint used by the service account if your cluster and platform version are the same, or later than those listed in the following table. If your Kubernetes or platform version are earlier than those listed in the table, than your service accounts can only use the global endpoint.

| Kubernetes version | Platform version | Default endpoint type |
|--------------------|------------------|-----------------------|
| 1.22               | eks.1            | Regional              |
| 1.21               | eks.3            | Global                |
| 1.20               | eks.3            | Global                |
| 1.19               | eks.7            | Global                |
| 1.18               | eks.9            | Global                |

AWS recommends using the regional AWS STS endpoints instead of the global endpoint to reduce latency, build in redundancy, and increase session token validity. The AWS Security Token Service must be active in the AWS Region where the pod is running and your application should have redundancy built in to pick a different AWS Region in the event of a failure of the service in the AWS Region. For more information, see [Managing AWS STS in an AWS Region](#) in the IAM User Guide.

### Prerequisites

- An existing cluster. If you don't have one, you can create one using one of the [Getting started with Amazon EKS \(p. 4\)](#) guides.
- An existing IAM OIDC provider for your cluster. For more information, see [Create an IAM OIDC provider for your cluster \(p. 446\)](#).
- An existing Kubernetes service account configured for use with the [Amazon EKS IAM for service accounts \(p. 442\)](#) feature.

## To configure the endpoint type used by a Kubernetes service account

The following examples all use the `aws-node` Kubernetes service account used by the [Amazon VPC CNI plugin \(p. 277\)](#). You can replace the *example values* with your own service accounts, pods, namespaces, and other resources.

1. Determine which AWS Region that one of your pods running the service account that you want to configure the endpoint for is running in. Replace `aws-node-6mfgv` with your pod name and `kube-system` with your pod's namespace.

```
kubectl describe pod aws-node-6mfgv -n kube-system |grep Node:
```

Example output:

```
ip-192-168-79-166.us-west-2/192.168.79.166
```

In the previous output, the pod is running on a node in the us-west-2 AWS Region.

2. Determine the endpoint type that pods using your existing service account are currently using with the following command:

```
kubectl describe pod aws-node-6mfgv -n kube-system |grep AWS_STS_REGIONAL_ENDPOINTS
```

Example output:

```
AWS_STS_REGIONAL_ENDPOINTS: regional
```

If the current endpoint is global, then `global` is returned in the output. If no output is returned, then the default endpoint type is in use and has not been overridden.

3. If your cluster or platform version are the same or later than those listed in the table, then you can change the endpoint type used by your service account from the default type to a different type with one of the following commands. Replace `aws-node` with the name of your service account and `kube-system` with the namespace for your service account.
  - If your default or current endpoint type is global and you want to change it to regional:

```
kubectl annotate serviceaccount -n kube-system aws-node \
eks.amazonaws.com/sts-regional-endpoints=true
```

If you are using [IAM roles for service accounts \(p. 442\)](#) to generate pre-signed S3 URLs in your application running in pods' containers, the format of the URL for regional endpoints is similar to the following example:

```
https://bucket.s3.us-west-2.amazonaws.com/path?...&X-Amz-Credential=your-access-key-id/date/us-west-2/s3/aws4_request&...
```

- If your default or current endpoint type is regional and you want to change it to global:

```
kubectl annotate serviceaccount -n kube-system aws-node \
eks.amazonaws.com/sts-regional-endpoints=false
```

If your application is explicitly making requests to AWS STS global endpoints and you do not override the default behavior of using regional endpoints in Amazon EKS 1.22 and later clusters, then requests fail with an error. For more information, see [Pod containers receive the following error: An error occurred \(SignatureDoesNotMatch\) when calling the](#)

[GetCallerIdentity operation: Credential should be scoped to a valid region. \(p. 556\).](#)

If you are using [IAM roles for service accounts \(p. 442\)](#) to generate pre-signed S3 URLs in your application running in pods' containers, the format of the URL for global endpoints is similar to the following example:

```
https://bucket.s3.amazonaws.com/path?...&X-Amz-Credential=your-access-key-id/date/us-west-2/s3/aws4_request&...
```

If you have automation that expects the pre-signed URL in a certain format or if your application or downstream dependencies that use pre-signed URLs have expectations for the AWS Region targeted, then make the necessary changes to use the appropriate AWS STS endpoint.

4. Delete and re-create any existing pods that are associated with the service account to apply the credential environment variables. The mutating web hook does not apply them to pods that are already running. You can replace `pods`, `kube-system`, and `-l k8s-app=aws-node` with the information for the pods that you set your annotation for.

```
kubectl delete pods -n kube-system -l k8s-app=aws-node
```

5. Confirm that the all pods restarted.

```
kubectl get pods -n kube-system -l k8s-app=aws-node
```

6. View the environment variables for one of the pods and verify that the `AWS_STS_REGIONAL_ENDPOINTS` value is what you set it to in a previous step.

```
kubectl describe pod aws-node-kzbtr -n kube-system |grep AWS_STS_REGIONAL_ENDPOINTS
```

Example output:

```
AWS_STS_REGIONAL_ENDPOINTS=regional
```

## Using a supported AWS SDK

The containers in your pods must use an AWS SDK version that supports assuming an IAM role via an OIDC web identity token file. AWS SDKs that are included in Linux distribution package managers may not be new enough to support this feature. Be sure to use at least the minimum SDK versions listed below:

- Java (Version 2) — [2.10.11](#)
- Java — [1.11.704](#)
- Go — [1.23.13](#)
- Python (Boto3) — [1.9.220](#)
- Python (botocore) — [1.12.200](#)
- AWS CLI — [1.16.232](#)
- Node — [3.27.0](#)
- Ruby — [3.58.0](#)
- C++ — [1.7.174](#)
- .NET — [3.3.659.1](#)
- PHP — [3.110.7](#)

Many popular Kubernetes add-ons, such as the [Cluster Autoscaler](#) and the [Installing the AWS Load Balancer Controller add-on \(p. 329\)](#), support IAM roles for service accounts. The [Amazon VPC CNI plugin for Kubernetes](#) also supports IAM roles for service accounts.

To ensure that you are using a supported SDK, follow the installation instructions for your preferred SDK at [Tools for Amazon Web Services](#) when you build your containers.

## Identity and access management for Amazon EKS

AWS Identity and Access Management (IAM) is an AWS service that helps an administrator securely control access to AWS resources. IAM administrators control who can be *authenticated* (signed in) and *authorized* (have permissions) to use Amazon EKS resources. IAM is an AWS service that you can use with no additional charge.

### Audience

How you use AWS Identity and Access Management (IAM) differs, depending on the work that you do in Amazon EKS.

**Service user** – If you use the Amazon EKS service to do your job, then your administrator provides you with the credentials and permissions that you need. As you use more Amazon EKS features to do your work, you might need additional permissions. Understanding how access is managed can help you request the right permissions from your administrator. If you cannot access a feature in Amazon EKS, see [Troubleshooting Amazon EKS identity and access \(p. 498\)](#).

**Service administrator** – If you're in charge of Amazon EKS resources at your company, you probably have full access to Amazon EKS. It's your job to determine which Amazon EKS features and resources your employees should access. You must then submit requests to your IAM administrator to change the permissions of your service users. Review the information on this page to understand the basic concepts of IAM. To learn more about how your company can use IAM with Amazon EKS, see [How Amazon EKS works with IAM \(p. 459\)](#).

**IAM administrator** – If you're an IAM administrator, you might want to learn details about how you can write policies to manage access to Amazon EKS. To view example Amazon EKS identity-based policies that you can use in IAM, see [Amazon EKS identity-based policy examples \(p. 462\)](#).

## Authenticating with identities

Authentication is how you sign in to AWS using your identity credentials. For more information about signing in using the AWS Management Console, see [Signing in to the AWS Management Console as an IAM user or root user](#) in the *IAM User Guide*.

You must be *authenticated* (signed in to AWS) as the AWS account root user, an IAM user, or by assuming an IAM role. You can also use your company's single sign-on authentication or even sign in using Google or Facebook. In these cases, your administrator previously set up identity federation using IAM roles. When you access AWS using credentials from another company, you are assuming a role indirectly.

To sign in directly to the [AWS Management Console](#), use your password with your root user email address or your IAM user name. You can access AWS programmatically using your root user or IAM users access keys. AWS provides SDK and command line tools to cryptographically sign your request using your credentials. If you don't use AWS tools, you must sign the request yourself. Do this using *Signature Version 4*, a protocol for authenticating inbound API requests. For more information about authenticating requests, see [Signature Version 4 signing process](#) in the *AWS General Reference*.

Regardless of the authentication method that you use, you might also be required to provide additional security information. For example, AWS recommends that you use multi-factor authentication (MFA) to

increase the security of your account. To learn more, see [Using multi-factor authentication \(MFA\) in AWS](#) in the *IAM User Guide*.

## AWS account root user

When you first create an AWS account, you begin with a single sign-in identity that has complete access to all AWS services and resources in the account. This identity is called the AWS account *root user* and is accessed by signing in with the email address and password that you used to create the account. We strongly recommend that you do not use the root user for your everyday tasks, even the administrative ones. Instead, adhere to the [best practice of using the root user only to create your first IAM user](#). Then securely lock away the root user credentials and use them to perform only a few account and service management tasks.

## IAM users and groups

An *IAM user* is an identity within your AWS account that has specific permissions for a single person or application. An IAM user can have long-term credentials such as a user name and password or a set of access keys. To learn how to generate access keys, see [Managing access keys for IAM users](#) in the *IAM User Guide*. When you generate access keys for an IAM user, make sure you view and securely save the key pair. You cannot recover the secret access key in the future. Instead, you must generate a new access key pair.

An *IAM group* is an identity that specifies a collection of IAM users. You can't sign in as a group. You can use groups to specify permissions for multiple users at a time. Groups make permissions easier to manage for large sets of users. For example, you could have a group named *IAMAdmins* and give that group permissions to administer IAM resources.

Users are different from roles. A user is uniquely associated with one person or application, but a role is intended to be assumable by anyone who needs it. Users have permanent long-term credentials, but roles provide temporary credentials. To learn more, see [When to create an IAM user \(instead of a role\)](#) in the *IAM User Guide*.

## IAM roles

An *IAM role* is an identity within your AWS account that has specific permissions. It is similar to an IAM user, but is not associated with a specific person. You can temporarily assume an IAM role in the AWS Management Console by [switching roles](#). You can assume a role by calling an AWS CLI or AWS API operation or by using a custom URL. For more information about methods for using roles, see [Using IAM roles](#) in the *IAM User Guide*.

IAM roles with temporary credentials are useful in the following situations:

- **Temporary IAM user permissions** – An IAM user can assume an IAM role to temporarily take on different permissions for a specific task.
- **Federated user access** – Instead of creating an IAM user, you can use existing identities from AWS Directory Service, your enterprise user directory, or a web identity provider. These are known as *federated users*. AWS assigns a role to a federated user when access is requested through an *identity provider*. For more information about federated users, see [Federated users and roles](#) in the *IAM User Guide*.
- **Cross-account access** – You can use an IAM role to allow someone (a trusted principal) in a different account to access resources in your account. Roles are the primary way to grant cross-account access. However, with some AWS services, you can attach a policy directly to a resource (instead of using a role as a proxy). To learn the difference between roles and resource-based policies for cross-account access, see [How IAM roles differ from resource-based policies](#) in the *IAM User Guide*.
- **Cross-service access** – Some AWS services use features in other AWS services. For example, when you make a call in a service, it's common for that service to run applications in Amazon EC2 or store objects

in Amazon S3. A service might do this using the calling principal's permissions, using a service role, or using a service-linked role.

- **Principal permissions** – When you use an IAM user or role to perform actions in AWS, you are considered a principal. Policies grant permissions to a principal. When you use some services, you might perform an action that then triggers another action in a different service. In this case, you must have permissions to perform both actions. To see whether an action requires additional dependent actions in a policy, see [Actions, Resources, and Condition Keys for Amazon Elastic Kubernetes Service](#) in the *Service Authorization Reference*.
- **Service role** – A service role is an [IAM role](#) that a service assumes to perform actions on your behalf. An IAM administrator can create, modify, and delete a service role from within IAM. For more information, see [Creating a role to delegate permissions to an AWS service](#) in the *IAM User Guide*.
- **Service-linked role** – A service-linked role is a type of service role that is linked to an AWS service. The service can assume the role to perform an action on your behalf. Service-linked roles appear in your IAM account and are owned by the service. An IAM administrator can view, but not edit the permissions for service-linked roles.
- **Applications running on Amazon EC2** – You can use an IAM role to manage temporary credentials for applications that are running on an EC2 instance and making AWS CLI or AWS API requests. This is preferable to storing access keys within the EC2 instance. To assign an AWS role to an EC2 instance and make it available to all of its applications, you create an instance profile that is attached to the instance. An instance profile contains the role and enables programs that are running on the EC2 instance to get temporary credentials. For more information, see [Using an IAM role to grant permissions to applications running on Amazon EC2 instances](#) in the *IAM User Guide*.

To learn whether to use IAM roles or IAM users, see [When to create an IAM role \(instead of a user\)](#) in the *IAM User Guide*.

## Managing access using policies

You control access in AWS by creating policies and attaching them to IAM identities or AWS resources. A policy is an object in AWS that, when associated with an identity or resource, defines their permissions. You can sign in as the root user or an IAM user, or you can assume an IAM role. When you then make a request, AWS evaluates the related identity-based or resource-based policies. Permissions in the policies determine whether the request is allowed or denied. Most policies are stored in AWS as JSON documents. For more information about the structure and contents of JSON policy documents, see [Overview of JSON policies](#) in the *IAM User Guide*.

Administrators can use AWS JSON policies to specify who has access to what. That is, which **principal** can perform **actions** on what **resources**, and under what **conditions**.

Every IAM entity (user or role) starts with no permissions. In other words, by default, users can do nothing, not even change their own password. To give a user permission to do something, an administrator must attach a permissions policy to a user. Or the administrator can add the user to a group that has the intended permissions. When an administrator gives permissions to a group, all users in that group are granted those permissions.

IAM policies define permissions for an action regardless of the method that you use to perform the operation. For example, suppose that you have a policy that allows the `iam:GetRole` action. A user with that policy can get role information from the AWS Management Console, the AWS CLI, or the AWS API.

## Identity-based policies

Identity-based policies are JSON permissions policy documents that you can attach to an identity, such as an IAM user, group of users, or role. These policies control what actions users and roles can perform, on which resources, and under what conditions. To learn how to create an identity-based policy, see [Creating IAM policies](#) in the *IAM User Guide*.

Identity-based policies can be further categorized as *inline policies* or *managed policies*. Inline policies are embedded directly into a single user, group, or role. Managed policies are standalone policies that you can attach to multiple users, groups, and roles in your AWS account. Managed policies include AWS managed policies and customer managed policies. To learn how to choose between a managed policy or an inline policy, see [Choosing between managed policies and inline policies](#) in the *IAM User Guide*.

## Resource-based policies

Resource-based policies are JSON policy documents that you attach to a resource. Examples of resource-based policies are IAM *role trust policies* and Amazon S3 *bucket policies*. In services that support resource-based policies, service administrators can use them to control access to a specific resource. For the resource where the policy is attached, the policy defines what actions a specified principal can perform on that resource and under what conditions. You must [specify a principal](#) in a resource-based policy. Principals can include accounts, users, roles, federated users, or AWS services.

Resource-based policies are inline policies that are located in that service. You can't use AWS managed policies from IAM in a resource-based policy.

## Access control lists (ACLs)

Access control lists (ACLs) control which principals (account members, users, or roles) have permissions to access a resource. ACLs are similar to resource-based policies, although they do not use the JSON policy document format.

Amazon S3, AWS WAF, and Amazon VPC are examples of services that support ACLs. To learn more about ACLs, see [Access control list \(ACL\) overview](#) in the *Amazon Simple Storage Service Developer Guide*.

## Other policy types

AWS supports additional, less-common policy types. These policy types can set the maximum permissions granted to you by the more common policy types.

- **Permissions boundaries** – A permissions boundary is an advanced feature in which you set the maximum permissions that an identity-based policy can grant to an IAM entity (IAM user or role). You can set a permissions boundary for an entity. The resulting permissions are the intersection of entity's identity-based policies and its permissions boundaries. Resource-based policies that specify the user or role in the `Principal` field are not limited by the permissions boundary. An explicit deny in any of these policies overrides the allow. For more information about permissions boundaries, see [Permissions boundaries for IAM entities](#) in the *IAM User Guide*.
- **Service control policies (SCPs)** – SCPs are JSON policies that specify the maximum permissions for an organization or organizational unit (OU) in AWS Organizations. AWS Organizations is a service for grouping and centrally managing multiple AWS accounts that your business owns. If you enable all features in an organization, then you can apply service control policies (SCPs) to any or all of your accounts. The SCP limits permissions for entities in member accounts, including each AWS account root user. For more information about Organizations and SCPs, see [How SCPs work](#) in the *AWS Organizations User Guide*.
- **Session policies** – Session policies are advanced policies that you pass as a parameter when you programmatically create a temporary session for a role or federated user. The resulting session's permissions are the intersection of the user or role's identity-based policies and the session policies. Permissions can also come from a resource-based policy. An explicit deny in any of these policies overrides the allow. For more information, see [Session policies](#) in the *IAM User Guide*.

## Multiple policy types

When multiple types of policies apply to a request, the resulting permissions are more complicated to understand. To learn how AWS determines whether to allow a request when multiple policy types are involved, see [Policy evaluation logic](#) in the *IAM User Guide*.

## How Amazon EKS works with IAM

Before you use IAM to manage access to Amazon EKS, you should understand what IAM features are available to use with Amazon EKS. To get a high-level view of how Amazon EKS and other AWS services work with IAM, see [AWS services that work with IAM](#) in the *IAM User Guide*.

### Topics

- [Amazon EKS identity-based policies \(p. 459\)](#)
- [Amazon EKS resource-based policies \(p. 461\)](#)
- [Authorization based on Amazon EKS tags \(p. 461\)](#)
- [Amazon EKS IAM roles \(p. 461\)](#)

## Amazon EKS identity-based policies

With IAM identity-based policies, you can specify allowed or denied actions and resources as well as the conditions under which actions are allowed or denied. Amazon EKS supports specific actions, resources, and condition keys. To learn about all of the elements that you use in a JSON policy, see [IAM JSON policy elements reference](#) in the *IAM User Guide*.

### Actions

Administrators can use AWS JSON policies to specify who has access to what. That is, which **principal** can perform **actions** on what **resources**, and under what **conditions**.

The **Action** element of a JSON policy describes the actions that you can use to allow or deny access in a policy. Policy actions usually have the same name as the associated AWS API operation. There are some exceptions, such as *permission-only actions* that don't have a matching API operation. There are also some operations that require multiple actions in a policy. These additional actions are called *dependent actions*.

Include actions in a policy to grant permissions to perform the associated operation.

Policy actions in Amazon EKS use the following prefix before the action: `eks:`. For example, to grant someone permission to get descriptive information about an Amazon EKS cluster, you include the `DescribeCluster` action in their policy. Policy statements must include either an `Action` or `NotAction` element.

To specify multiple actions in a single statement, separate them with commas as follows:

```
"Action": ["eks:action1", "eks:action2"]
```

You can specify multiple actions using wildcards (\*). For example, to specify all actions that begin with the word `Describe`, include the following action:

```
"Action": "eks:Describe*"
```

To see a list of Amazon EKS actions, see [Actions Defined by Amazon Elastic Kubernetes Service](#) in the *Service Authorization Reference*.

## Resources

Administrators can use AWS JSON policies to specify who has access to what. That is, which **principal** can perform **actions** on what **resources**, and under what **conditions**.

The Resource JSON policy element specifies the object or objects to which the action applies. Statements must include either a Resource or a NotResource element. As a best practice, specify a resource using its [Amazon Resource Name \(ARN\)](#). You can do this for actions that support a specific resource type, known as *resource-level permissions*.

For actions that don't support resource-level permissions, such as listing operations, use a wildcard (\*) to indicate that the statement applies to all resources.

```
"Resource": "*"
```

The Amazon EKS cluster resource has the following ARN:

```
arn:${Partition}:eks:${Region}:${Account}:cluster/${ClusterName}
```

For more information about the format of ARNs, see [Amazon resource names \(ARNs\) and AWS service namespaces](#).

For example, to specify the dev cluster in your statement, use the following ARN:

```
"Resource": "arn:aws:eks:region-code:111122223333:cluster/dev"
```

To specify all clusters that belong to a specific account and AWS Region, use the wildcard (\*):

```
"Resource": "arn:aws:eks:region-code:111122223333:cluster/*"
```

Some Amazon EKS actions, such as those for creating resources, cannot be performed on a specific resource. In those cases, you must use the wildcard (\*).

```
"Resource": "*"
```

To see a list of Amazon EKS resource types and their ARNs, see [Resources Defined by Amazon Elastic Kubernetes Service](#) in the *Service Authorization Reference*. To learn with which actions you can specify the ARN of each resource, see [Actions Defined by Amazon Elastic Kubernetes Service](#).

## Condition keys

Amazon EKS defines its own set of condition keys and also supports using some global condition keys. To see all AWS global condition keys, see [AWS Global Condition Context Keys](#) in the *IAM User Guide*.

You can set condition keys when associating an OpenID Connect provider to your cluster. For more information, see [Example IAM policy \(p. 412\)](#).

All Amazon EC2 actions support the `aws:RequestedRegion` and `ec2:Region` condition keys. For more information, see [Example: Restricting Access to a Specific AWS Region](#).

For a list of Amazon EKS condition keys, see [Condition Keys for Amazon Elastic Kubernetes Service](#) in the *Service Authorization Reference*. To learn which actions and resources you can use a condition key with, see [Actions Defined by Amazon Elastic Kubernetes Service](#).

## Examples

To view examples of Amazon EKS identity-based policies, see [Amazon EKS identity-based policy examples \(p. 462\)](#).

When you create an Amazon EKS cluster, the AWS Identity and Access Management (IAM) entity user or role, such as a [federated user](#) that creates the cluster, is automatically granted `system:masters` permissions in the cluster's role-based access control (RBAC) configuration in the Amazon EKS control plane. This IAM entity doesn't appear in any visible configuration, so make sure to keep track of which IAM entity originally created the cluster. To grant additional AWS users or roles the ability to interact with your cluster, you must edit the `aws-auth ConfigMap` within Kubernetes and create a Kubernetes `rolebinding` or `clusterrolebinding` with the name of a group that you specify in the `aws-auth ConfigMap`.

For additional information about working with the `ConfigMap`, see [Enabling IAM user and role access to your cluster \(p. 402\)](#).

## Amazon EKS resource-based policies

Amazon EKS does not support resource-based policies.

## Authorization based on Amazon EKS tags

You can attach tags to Amazon EKS resources or pass tags in a request to Amazon EKS. To control access based on tags, you provide tag information in the [condition element](#) of a policy using the `eks:ResourceTag/<key-name>`, `aws:RequestTag/<key-name>`, or `aws:TagKeys` condition keys. For more information about tagging Amazon EKS resources, see [Tagging your Amazon EKS resources \(p. 431\)](#).

## Amazon EKS IAM roles

An [IAM role](#) is an entity within your AWS account that has specific permissions.

### Using temporary credentials with Amazon EKS

You can use temporary credentials to sign in with federation, assume an IAM role, or to assume a cross-account role. You obtain temporary security credentials by calling AWS STS API operations such as [AssumeRole](#) or [GetFederationToken](#).

Amazon EKS supports using temporary credentials.

### Service-linked roles

[Service-linked roles](#) allow AWS services to access resources in other services to complete an action on your behalf. Service-linked roles appear in your IAM account and are owned by the service. An IAM administrator can view but not edit the permissions for service-linked roles.

Amazon EKS supports service-linked roles. For details about creating or managing Amazon EKS service-linked roles, see [Using service-linked roles for Amazon EKS \(p. 465\)](#).

### Service roles

This feature allows a service to assume a [service role](#) on your behalf. This role allows the service to access resources in other services to complete an action on your behalf. Service roles appear in your IAM account and are owned by the account. This means that an IAM administrator can change the permissions for this role. However, doing so might break the functionality of the service.

Amazon EKS supports service roles. For more information, see [Amazon EKS cluster IAM role \(p. 472\)](#) and [Amazon EKS node IAM role \(p. 474\)](#).

## Choosing an IAM role in Amazon EKS

When you create a cluster resource in Amazon EKS, you must choose a role to allow Amazon EKS to access several other AWS resources on your behalf. If you have previously created a service role, then Amazon EKS provides you with a list of roles to choose from. It's important to choose a role that has the Amazon EKS managed policies attached to it. For more information, see [Check for an existing cluster role \(p. 472\)](#) and [Check for an existing node role \(p. 474\)](#).

# Amazon EKS identity-based policy examples

By default, IAM users and roles don't have permission to create or modify Amazon EKS resources. They also can't perform tasks using the AWS Management Console, AWS CLI, or AWS API. An IAM administrator must create IAM policies that grant users and roles permission to perform specific API operations on the specified resources they need. The administrator must then attach those policies to the IAM users or groups that require those permissions.

To learn how to create an IAM identity-based policy using these example JSON policy documents, see [Creating policies on the JSON tab](#) in the *IAM User Guide*.

When you create an Amazon EKS cluster, the AWS Identity and Access Management (IAM) entity user or role, such as a [federated user](#) that creates the cluster, is automatically granted `system:masters` permissions in the cluster's role-based access control (RBAC) configuration in the Amazon EKS control plane. This IAM entity doesn't appear in any visible configuration, so make sure to keep track of which IAM entity originally created the cluster. To grant additional AWS users or roles the ability to interact with your cluster, you must edit the `aws-auth ConfigMap` within Kubernetes and create a Kubernetes `rolebinding` or `clusterrolebinding` with the name of a group that you specify in the `aws-auth ConfigMap`.

For additional information about working with the `ConfigMap`, see [Enabling IAM user and role access to your cluster \(p. 402\)](#).

### Topics

- [Policy best practices \(p. 462\)](#)
- [Using the Amazon EKS console \(p. 463\)](#)
- [Allow users to view their own permissions \(p. 463\)](#)
- [Update a Kubernetes cluster \(p. 464\)](#)
- [List or describe all clusters \(p. 464\)](#)

## Policy best practices

Identity-based policies are very powerful. They determine whether someone can create, access, or delete Amazon EKS resources in your account. These actions can incur costs for your AWS account. When you create or edit identity-based policies, follow these guidelines and recommendations:

- **Get started using AWS managed policies** – To start using Amazon EKS quickly, use AWS managed policies to give your employees the permissions they need. These policies are already available in your account and are maintained and updated by AWS. For more information, see [Get started using permissions with AWS managed policies](#) in the *IAM User Guide*.
- **Grant least privilege** – When you create custom policies, grant only the permissions required to perform a task. Start with a minimum set of permissions and grant additional permissions as necessary. Doing so is more secure than starting with permissions that are too lenient and then trying to tighten them later. For more information, see [Grant least privilege](#) in the *IAM User Guide*.
- **Enable MFA for sensitive operations** – For extra security, require IAM users to use multi-factor authentication (MFA) to access sensitive resources or API operations. For more information, see [Using multi-factor authentication \(MFA\) in AWS](#) in the *IAM User Guide*.

- **Use policy conditions for extra security** – To the extent that it's practical, define the conditions under which your identity-based policies allow access to a resource. For example, you can write conditions to specify a range of allowable IP addresses that a request must come from. You can also write conditions to allow requests only within a specified date or time range, or to require the use of SSL or MFA. For more information, see [IAM JSON policy elements: Condition](#) in the *IAM User Guide*.

## Using the Amazon EKS console

To access the Amazon EKS console, an IAM entity, such as a user or role, must have a minimum set of permissions. These permissions allow the entity to list and view details about the Amazon EKS resources in your AWS account. If you create an identity-based policy that is more restrictive than the minimum required permissions, the console won't function as intended for entities with that policy attached to them.

To ensure that your IAM entities can still use the Amazon EKS console, create a policy with your own unique name, such as `AmazonEKSAdminPolicy`. Attach the policy to the entities. For more information, see [Adding permissions to a user](#) in the *IAM User Guide*.

### Important

The following example policy allows an entity to view information on the **Configuration** tab in the console. To view information on the **Overview** and **Resources** tabs in the AWS Management Console, the entity also needs Kubernetes permissions. For more information, see [Required permissions \(p. 506\)](#).

```
{
 "Version": "2012-10-17",
 "Statement": [
 {
 "Effect": "Allow",
 "Action": [
 "eks:*"
],
 "Resource": "*"
 },
 {
 "Effect": "Allow",
 "Action": "iam:PassRole",
 "Resource": "*",
 "Condition": {
 "StringEquals": {
 "iam:PassedToService": "eks.amazonaws.com"
 }
 }
 }
]
}
```

You don't need to allow minimum console permissions for users that are making calls only to the AWS CLI or the AWS API. Instead, allow access to only the actions that match the API operation that you're trying to perform.

## Allow users to view their own permissions

This example shows how you might create a policy that allows IAM users to view the inline and managed policies that are attached to their user identity. This policy includes permissions to complete this action on the console or programmatically using the AWS CLI or AWS API.

```
{
 "Version": "2012-10-17",
 "Statement": [
 {
 "Effect": "Allow",
 "Action": "iam:GetUser",
 "Resource": "arn:aws:iam::123456789012:user/
 }
]
}
```

```

"Statement": [
 {
 "Sid": "ViewOwnUserInfo",
 "Effect": "Allow",
 "Action": [
 "iam:GetUserPolicy",
 "iam>ListGroupsForUser",
 "iam>ListAttachedUserPolicies",
 "iam>ListUserPolicies",
 "iam GetUser"
],
 "Resource": ["arn:aws:iam::*:user/${aws:username}"]
 },
 {
 "Sid": "NavigateInConsole",
 "Effect": "Allow",
 "Action": [
 "iam:GetGroupPolicy",
 "iam:GetPolicyVersion",
 "iam GetPolicy",
 "iam>ListAttachedGroupPolicies",
 "iam>ListGroupPolicies",
 "iam>ListPolicyVersions",
 "iam>ListPolicies",
 "iam>ListUsers"
],
 "Resource": "*"
 }
]
}

```

## Update a Kubernetes cluster

This example shows how you can create a policy that allows a user to update the Kubernetes version of any *dev* cluster for an account, in any region.

```

{
 "Version": "2012-10-17",
 "Statement": [
 {
 "Effect": "Allow",
 "Action": "eks:UpdateClusterVersion",
 "Resource": "arn:aws:eks::<111122223333>:cluster/<dev>"
 }
]
}

```

## List or describe all clusters

This example shows how you can create a policy that allows a user read-only access to list or describe all clusters. An account must be able to list and describe clusters to use the `update-kubeconfig` AWS CLI command.

```

{
 "Version": "2012-10-17",
 "Statement": [
 {
 "Effect": "Allow",
 "Action": [
 "eks:DescribeCluster",
 "eks>ListClusters"
]
 }
]
}

```

```
],
 "Resource": "*"
 }
}
```

## Using service-linked roles for Amazon EKS

Amazon Elastic Kubernetes Service uses AWS Identity and Access Management (IAM) [service-linked roles](#). A service-linked role is a unique type of IAM role that is linked directly to Amazon EKS. Service-linked roles are predefined by Amazon EKS and include all the permissions that the service requires to call other AWS services on your behalf.

### Topics

- [Using roles for Amazon EKS clusters \(p. 465\)](#)
- [Using roles for Amazon EKS node groups \(p. 467\)](#)
- [Using roles for Amazon EKS Fargate profiles \(p. 468\)](#)
- [Using roles to connect a Kubernetes cluster to Amazon EKS \(p. 470\)](#)

## Using roles for Amazon EKS clusters

Amazon Elastic Kubernetes Service uses AWS Identity and Access Management (IAM) [service-linked roles](#). A service-linked role is a unique type of IAM role that is linked directly to Amazon EKS. Service-linked roles are predefined by Amazon EKS and include all the permissions that the service requires to call other AWS services on your behalf.

A service-linked role makes setting up Amazon EKS easier because you don't have to manually add the necessary permissions. Amazon EKS defines the permissions of its service-linked roles, and unless defined otherwise, only Amazon EKS can assume its roles. The defined permissions include the trust policy and the permissions policy, and that permissions policy cannot be attached to any other IAM entity.

You can delete a service-linked role only after first deleting their related resources. This protects your Amazon EKS resources because you can't inadvertently remove permission to access the resources.

For information about other services that support service-linked roles, see [AWS services that work with IAM](#) and look for the services that have **Yes** in the **Service-linked role** column. Choose a **Yes** with a link to view the service-linked role documentation for that service.

### Service-Linked Role Permissions for Amazon EKS

Amazon EKS uses the service-linked role named `AWSServiceRoleForAmazonEKS` – The role allows Amazon EKS to manage clusters in your account. The attached policies allow the role to manage the following resources: network interfaces, security groups, logs, and VPCs.

#### Note

The `AWSServiceRoleForAmazonEKS` service-linked role is distinct from the role required for cluster creation. For more information, see [Amazon EKS cluster IAM role \(p. 472\)](#).

The `AWSServiceRoleForAmazonEKS` service-linked role trusts the following services to assume the role:

- `eks.amazonaws.com`

The role permissions policy allows Amazon EKS to complete the following actions on the specified resources:

- [AmazonEKSServiceRolePolicy](#)

You must configure permissions to allow an IAM entity (such as a user, group, or role) to create, edit, or delete a service-linked role. For more information, see [Service-Linked Role Permissions](#) in the *IAM User Guide*.

## Creating a Service-Linked Role for Amazon EKS

You don't need to manually create a service-linked role. When you create a cluster in the AWS Management Console, the AWS CLI, or the AWS API, Amazon EKS creates the service-linked role for you.

If you delete this service-linked role, and then need to create it again, you can use the same process to recreate the role in your account. When you create a cluster, Amazon EKS creates the service-linked role for you again.

## Editing a service-linked role for Amazon EKS

Amazon EKS does not allow you to edit the `AWSServiceRoleForAmazonEKS` service-linked role. After you create a service-linked role, you cannot change the name of the role because various entities might reference the role. However, you can edit the description of the role using IAM. For more information, see [Editing a service-linked role](#) in the *IAM User Guide*.

## Deleting a service-linked role for Amazon EKS

If you no longer need to use a feature or service that requires a service-linked role, we recommend that you delete that role. That way you don't have an unused entity that is not actively monitored or maintained. However, you must clean up your service-linked role before you can manually delete it.

### Cleaning up a service-linked role

Before you can use IAM to delete a service-linked role, you must first delete any resources used by the role.

#### Note

If the Amazon EKS service is using the role when you try to delete the resources, then the deletion might fail. If that happens, wait for a few minutes and try the operation again.

### To delete Amazon EKS resources used by the `AWSServiceRoleForAmazonEKS` role.

1. Open the Amazon EKS console at <https://console.aws.amazon.com/eks/home#/clusters>.
2. In the left navigation pane, choose Amazon EKS **Clusters**.
3. If your cluster has any node groups or Fargate profiles, you must delete them before you can delete the cluster. For more information, see [Deleting a managed node group \(p. 131\)](#) and [Deleting a Fargate profile \(p. 160\)](#).
4. On the **Clusters** page, choose the cluster that you want to delete and choose **Delete**.
5. Type the name of the cluster in the deletion confirmation window, and then choose **Delete**.
6. Repeat this procedure for any other clusters in your account. Wait for all of the delete operations to finish.

### Manually delete the service-linked role

Use the IAM console, the AWS CLI, or the AWS API to delete the `AWSServiceRoleForAmazonEKS` service-linked role. For more information, see [Deleting a service-linked role](#) in the *IAM User Guide*.

## Supported regions for Amazon EKS service-linked roles

Amazon EKS supports using service-linked roles in all of the regions where the service is available. For more information, see [Amazon EKS Service Endpoints and Quotas](#).

## Using roles for Amazon EKS node groups

Amazon EKS uses AWS Identity and Access Management (IAM) [service-linked roles](#). A service-linked role is a unique type of IAM role that is linked directly to Amazon EKS. Service-linked roles are predefined by Amazon EKS and include all the permissions that the service requires to call other AWS services on your behalf.

A service-linked role makes setting up Amazon EKS easier because you don't have to manually add the necessary permissions. Amazon EKS defines the permissions of its service-linked roles, and unless defined otherwise, only Amazon EKS can assume its roles. The defined permissions include the trust policy and the permissions policy, and that permissions policy cannot be attached to any other IAM entity.

You can delete a service-linked role only after first deleting their related resources. This protects your Amazon EKS resources because you can't inadvertently remove permission to access the resources.

For information about other services that support service-linked roles, see [AWS services that work with IAM](#) and look for the services that have **Yes** in the **Service-linked role** column. Choose a **Yes** with a link to view the service-linked role documentation for that service.

### Service-linked role permissions for Amazon EKS

Amazon EKS uses the service-linked role named `AWSServiceRoleForAmazonEKSNodegroup` – The role allows Amazon EKS to manage node groups in your account. The attached policies allow the role to manage the following resources: Auto Scaling groups, security groups, launch templates and IAM instance profiles..

The `AWSServiceRoleForAmazonEKSNodegroup` service-linked role trusts the following services to assume the role:

- `eks-nodegroup.amazonaws.com`

The role permissions policy allows Amazon EKS to complete the following actions on the specified resources:

- [AWSServiceRoleForAmazonEKSNodegroup](#)

You must configure permissions to allow an IAM entity (such as a user, group, or role) to create, edit, or delete a service-linked role. For more information, see [Service-Linked Role Permissions](#) in the *IAM User Guide*.

### Creating a service-linked role for Amazon EKS

You don't need to manually create a service-linked role. When you `CreateNodegroup` in the AWS Management Console, the AWS CLI, or the AWS API, Amazon EKS creates the service-linked role for you.

#### Important

This service-linked role can appear in your account if you completed an action in another service that uses the features supported by this role. If you were using the Amazon EKS service before January 1, 2017, when it began supporting service-linked roles, then Amazon EKS created the `AWSServiceRoleForAmazonEKSNodegroup` role in your account. To learn more, see [A New Role Appeared in My IAM Account](#).

#### Creating a service-linked role in Amazon EKS (AWS API)

You don't need to manually create a service-linked role. When you create a managed node group in the AWS Management Console, the AWS CLI, or the AWS API, Amazon EKS creates the service-linked role for you.

If you delete this service-linked role, and then need to create it again, you can use the same process to recreate the role in your account. When you create another managed node group, Amazon EKS creates the service-linked role for you again.

## Editing a service-linked role for Amazon EKS

Amazon EKS does not allow you to edit the `AWSServiceRoleForAmazonEKSNodegroup` service-linked role. After you create a service-linked role, you cannot change the name of the role because various entities might reference the role. However, you can edit the description of the role using IAM. For more information, see [Editing a service-linked role](#) in the *IAM User Guide*.

## Deleting a service-linked role for Amazon EKS

If you no longer need to use a feature or service that requires a service-linked role, we recommend that you delete that role. That way you don't have an unused entity that is not actively monitored or maintained. However, you must clean up your service-linked role before you can manually delete it.

### Cleaning up a service-linked role

Before you can use IAM to delete a service-linked role, you must first delete any resources used by the role.

#### Note

If the Amazon EKS service is using the role when you try to delete the resources, then the deletion might fail. If that happens, wait for a few minutes and try the operation again.

### To delete Amazon EKS resources used by the `AWSServiceRoleForAmazonEKSNodegroup` role.

1. Open the Amazon EKS console at <https://console.aws.amazon.com/eks/home#/clusters>.
2. In the left navigation pane, choose Amazon EKS **Clusters**.
3. Select the **Configuration** tab and then choose the **Compute** tab.
4. In the **Node Groups** section, choose the node group to delete.
5. Type the name of the node group in the deletion confirmation window, and then choose **Delete**.
6. Repeat this procedure for any other node groups in the cluster. Wait for all of the delete operations to finish.

### Manually delete the service-linked role

Use the IAM console, the AWS CLI, or the AWS API to delete the `AWSServiceRoleForAmazonEKSNodegroup` service-linked role. For more information, see [Deleting a service-linked role](#) in the *IAM User Guide*.

## Supported regions for Amazon EKS service-linked roles

Amazon EKS supports using service-linked roles in all of the regions where the service is available. For more information, see [Amazon EKS Service Endpoints and Quotas](#).

## Using roles for Amazon EKS Fargate profiles

Amazon EKS uses AWS Identity and Access Management (IAM) [service-linked roles](#). A service-linked role is a unique type of IAM role that is linked directly to Amazon EKS. Service-linked roles are predefined by Amazon EKS and include all the permissions that the service requires to call other AWS services on your behalf.

A service-linked role makes setting up Amazon EKS easier because you don't have to manually add the necessary permissions. Amazon EKS defines the permissions of its service-linked roles, and unless

defined otherwise, only Amazon EKS can assume its roles. The defined permissions include the trust policy and the permissions policy, and that permissions policy cannot be attached to any other IAM entity.

You can delete a service-linked role only after first deleting their related resources. This protects your Amazon EKS resources because you can't inadvertently remove permission to access the resources.

For information about other services that support service-linked roles, see [AWS services that work with IAM](#) and look for the services that have **Yes** in the **Service-linked role** column. Choose a **Yes** with a link to view the service-linked role documentation for that service.

## Service-linked role permissions for Amazon EKS

Amazon EKS uses the service-linked role named `AWSServiceRoleForAmazonEKSForFargate` – The role allows Amazon EKS Fargate to configure VPC networking required for Fargate Pods. The attached policies allow the role to create and delete Elastic Network Interfaces and describe Elastic Network Interfaces and resources.

The `AWSServiceRoleForAmazonEKSForFargate` service-linked role trusts the following services to assume the role:

- `eks-fargate.amazonaws.com`

The role permissions policy allows Amazon EKS to complete the following actions on the specified resources:

- [AmazonEKSForFargateServiceRolePolicy](#)

You must configure permissions to allow an IAM entity (such as a user, group, or role) to create, edit, or delete a service-linked role. For more information, see [Service-linked role permissions](#) in the *IAM User Guide*.

## Creating a service-linked role for Amazon EKS

You don't need to manually create a service-linked role. When you create a Fargate profile in the AWS Management Console, the AWS CLI, or the AWS API, Amazon EKS creates the service-linked role for you.

### Important

This service-linked role can appear in your account if you completed an action in another service that uses the features supported by this role. If you were using the Amazon EKS service before December 13, 2019, when it began supporting service-linked roles, then Amazon EKS created the `AWSServiceRoleForAmazonEKSForFargate` role in your account. To learn more, see [A New Role Appeared in My IAM Account](#).

### Creating a service-linked role in Amazon EKS (AWS API)

You don't need to manually create a service-linked role. When you create a Fargate profile in the AWS Management Console, the AWS CLI, or the AWS API, Amazon EKS creates the service-linked role for you.

If you delete this service-linked role, and then need to create it again, you can use the same process to recreate the role in your account. When you create another managed node group, Amazon EKS creates the service-linked role for you again.

## Editing a service-linked role for Amazon EKS

Amazon EKS does not allow you to edit the `AWSServiceRoleForAmazonEKSForFargate` service-linked role. After you create a service-linked role, you cannot change the name of the role because

various entities might reference the role. However, you can edit the description of the role using IAM. For more information, see [Editing a service-linked role](#) in the *IAM User Guide*.

## Deleting a service-linked role for Amazon EKS

If you no longer need to use a feature or service that requires a service-linked role, we recommend that you delete that role. That way you don't have an unused entity that is not actively monitored or maintained. However, you must clean up your service-linked role before you can manually delete it.

### Cleaning up a service-linked role

Before you can use IAM to delete a service-linked role, you must first delete any resources used by the role.

#### Note

If the Amazon EKS service is using the role when you try to delete the resources, then the deletion might fail. If that happens, wait for a few minutes and try the operation again.

### To delete Amazon EKS resources used by the `AWSServiceRoleForAmazonEKSForFargate` role.

1. Open the Amazon EKS console at <https://console.aws.amazon.com/eks/home#/clusters>.
2. In the left navigation pane, choose Amazon EKS **Clusters**.
3. On the **Clusters** page, select your cluster.
4. Select the **Configuration** tab and then select the **Compute** tab.
5. If there are any Fargate profiles in the **Fargate Profiles** section, select each one individually, and then choose **Delete**.
6. Type the name of the profile in the deletion confirmation window, and then choose **Delete**.
7. Repeat this procedure for any other Fargate profiles in the cluster and for any other clusters in your account.

### Manually delete the service-linked role

Use the IAM console, the AWS CLI, or the AWS API to delete the `AWSServiceRoleForAmazonEKSForFargate` service-linked role. For more information, see [Deleting a service-linked role](#) in the *IAM User Guide*.

## Supported regions for Amazon EKS service-linked roles

Amazon EKS supports using service-linked roles in all of the regions where the service is available. For more information, see [Amazon EKS Service Endpoints and Quotas](#).

## Using roles to connect a Kubernetes cluster to Amazon EKS

Amazon EKS uses AWS Identity and Access Management (IAM) [service-linked roles](#). A service-linked role is a unique type of IAM role that is linked directly to Amazon EKS. Service-linked roles are predefined by Amazon EKS and include all the permissions that the service requires to call other AWS services on your behalf.

A service-linked role makes setting up Amazon EKS easier because you don't have to manually add the necessary permissions. Amazon EKS defines the permissions of its service-linked roles, and unless defined otherwise, only Amazon EKS can assume its roles. The defined permissions include the trust policy and the permissions policy, and that permissions policy cannot be attached to any other IAM entity.

You can delete a service-linked role only after first deleting their related resources. This protects your Amazon EKS resources because you can't inadvertently remove permission to access the resources.

For information about other services that support service-linked roles, see [AWS services that work with IAM](#) and look for the services that have **Yes** in the **Service-linked role** column. Choose a **Yes** with a link to view the service-linked role documentation for that service.

## Service-linked role permissions for Amazon EKS

Amazon EKS uses the service-linked role named `AWSServiceRoleForAmazonEKSConnector` – The role allows Amazon EKS to connect Kubernetes clusters. The attached policies allow the role to manage necessary resources to connect to your registered Kubernetes cluster.

The `AWSServiceRoleForAmazonEKSConnector` service-linked role trusts the following services to assume the role:

- `eks-connector.amazonaws.com`

The role permissions policy allows Amazon EKS to complete the following actions on the specified resources:

- [AmazonEKSConectorServiceRolePolicy](#)

You must configure permissions to allow an IAM entity (such as a user, group, or role) to create, edit, or delete a service-linked role. For more information, see [Service-linked role permissions](#) in the *IAM User Guide*.

## Creating a service-linked role for Amazon EKS

You don't need to manually create a service-linked role to connect a cluster. When you connect a cluster in the AWS Management Console, the AWS CLI, `eksctl`, or the AWS API, Amazon EKS creates the service-linked role for you.

If you delete this service-linked role, and then need to create it again, you can use the same process to recreate the role in your account. When you connect a cluster, Amazon EKS creates the service-linked role for you again.

## Editing a service-linked role for Amazon EKS

Amazon EKS does not allow you to edit the `AWSServiceRoleForAmazonEKSConnector` service-linked role. After you create a service-linked role, you cannot change the name of the role because various entities might reference the role. However, you can edit the description of the role using IAM. For more information, see [Editing a service-linked role](#) in the *IAM User Guide*.

## Deleting a service-linked role for Amazon EKS

If you no longer need to use a feature or service that requires a service-linked role, we recommend that you delete that role. That way you don't have an unused entity that is not actively monitored or maintained. However, you must clean up your service-linked role before you manually delete it.

### Cleaning up a service-linked role

Before you can use IAM to delete a service-linked role, you must first delete any resources used by the role.

#### Note

If the Amazon EKS service is using the role when you try to delete the resources, then the deletion might fail. If that happens, wait for a few minutes and try the operation again.

## To delete Amazon EKS resources used by the `AWSServiceRoleForAmazonEKSConnector` role.

1. Open the Amazon EKS console at <https://console.aws.amazon.com/eks/home#/clusters>.
2. In the left navigation pane, choose Amazon EKS **Clusters**.
3. On the **Clusters** page, select your cluster.
4. Select the **Deregister** tab and then select the **Ok** tab.

### Manually delete the service-linked role

Use the IAM console, the AWS CLI, or the AWS API to delete the `AWSServiceRoleForAmazonEKSConnector` service-linked role. For more information, see [Deleting a service-linked role](#) in the *IAM User Guide*.

## Supported regions for Amazon EKS service-linked roles

Amazon EKS supports using service-linked roles in all of the regions where the service is available. For more information, see [Amazon EKS Service Endpoints and Quotas](#).

## Amazon EKS cluster IAM role

Kubernetes clusters managed by Amazon EKS make calls to other AWS services on your behalf to manage the resources that you use with the service. Before you can create Amazon EKS clusters, you must create an IAM role with the following IAM policies:

- [AmazonEKSClusterPolicy](#)

### Note

Prior to April 16, 2020, [AmazonEKSServicePolicy](#) was also required and the suggested name was `eksServiceRole`. With the `AWSServiceRoleForAmazonEKS` service-linked role, that policy is no longer required for clusters created on or after April 16, 2020.

## Check for an existing cluster role

You can use the following procedure to check and see if your account already has the Amazon EKS cluster role.

### To check for the `eksClusterRole` in the IAM console

1. Open the IAM console at <https://console.aws.amazon.com/iam/>.
2. In the left navigation pane, choose **Roles**.
3. Search the list of roles for `eksClusterRole`. If a role that includes `eksClusterRole` doesn't exist, then see [Creating the Amazon EKS cluster role \(p. 473\)](#) to create the role. If a role that includes `eksClusterRole` does exist, then select the role to view the attached policies.
4. Choose **Permissions**.
5. Ensure that the **AmazonEKSClusterPolicy** managed policy is attached to the role. If the policy is attached, your Amazon EKS cluster role is properly configured.
6. Choose **Trust relationships**, and then choose **Edit trust policy**.
7. Verify that the trust relationship contains the following policy. If the trust relationship matches the policy below, choose **Cancel**. If the trust relationship doesn't match, copy the policy into the **Edit trust policy** window and choose **Update policy**.

{

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

## Creating the Amazon EKS cluster role

You can use the AWS Management Console or the AWS CLI to create the cluster role.

AWS Management Console

### To create your Amazon EKS cluster role in the IAM console

1. Open the IAM console at <https://console.aws.amazon.com/iam/>.
2. Choose **Roles**, then **Create role**.
3. Under **Trusted entity type**, select **AWS service**.
4. From the **Use cases for other AWS services** dropdown list, choose **EKS**.
5. Choose **EKS - Cluster** for your use case, and then choose **Next**.
6. On the **Add permissions** tab, choose **Next**.
7. For **Role name**, enter a unique name for your role, such as **eksClusterRole**.
8. For **Description**, enter descriptive text such as **Amazon EKS - Cluster role**.
9. Choose **Create role**.

AWS CLI

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

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

2. Create the role. You can replace **eksClusterRole** with any name that you choose.

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

3. Attach the required IAM policy to the role.

```
aws iam attach-role-policy \
```

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

## Amazon EKS node IAM role

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. Before you can launch nodes and register them into a cluster, you must create an IAM role for those nodes to use when they are launched. This requirement applies to nodes launched with the Amazon EKS optimized AMI provided by Amazon, or with any other node AMIs that you intend to use. Before you create nodes, you must create an IAM role with the following IAM policies:

- [AmazonEKSWorkerNodePolicy](#)
- [AmazonEC2ContainerRegistryReadOnly](#)
- Either the [AmazonEKS\\_CNI\\_Policy](#) managed policy (if you created your cluster with the IPv4 family) or an [IPv6 policy that you create \(p. 281\)](#) (if you created your cluster with the IPv6 family). Rather than attaching the policy to this role however, we recommend that you attach the policy to a separate role used specifically for the Amazon VPC CNI add-on. For more information about creating a separate role for the Amazon VPC CNI add-on, see [the section called “Configure plugin for IAM account” \(p. 277\)](#).

### Note

The Amazon EC2 node groups must have a different IAM role than the Fargate profile. For more information, see [Amazon EKS pod execution IAM role \(p. 477\)](#).

## Check for an existing node role

You can use the following procedure to check and see if your account already has the Amazon EKS node role.

### To check for the `eksNodeRole` in the IAM console

1. Open the IAM console at <https://console.aws.amazon.com/iam/>.
2. In the left navigation pane, choose **Roles**.
3. Search the list of roles for `eksNodeRole`, `AmazonEKSNodeRole`, or `NodeInstanceRole`. If a role with one of those names doesn't exist, then see [Creating the Amazon EKS node IAM role \(p. 475\)](#) to create the role. If a role that contains `eksNodeRole`, `AmazonEKSNodeRole`, or `NodeInstanceRole` does exist, then select the role to view the attached policies.
4. Choose **Permissions**.
5. Ensure that the [AmazonEKSWorkerNodePolicy](#) and [AmazonEC2ContainerRegistryReadOnly](#) managed policies are attached to the role. If the policies are attached, your Amazon EKS node role is properly configured.

### Note

If the [AmazonEKS\\_CNI\\_Policy](#) policy is attached to the role, we recommend removing it and attaching it to an IAM role that is mapped to the `aws-node` Kubernetes service account instead. For more information, see [Configuring the Amazon VPC CNI plugin to use IAM roles for service accounts \(p. 277\)](#).

6. Choose **Trust relationships**, and then choose **Edit trust policy**.
7. Verify that the trust relationship contains the following policy. If the trust relationship matches the policy below, choose **Cancel**. If the trust relationship doesn't match, copy the policy into the **Edit trust policy** window and choose **Update policy**.

```
{
```

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

## Creating the Amazon EKS node IAM role

You can create the node IAM role with the AWS Management Console or the AWS CLI. Select the tab with the name of the tool that you want to create the role with.

AWS Management Console

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

1. Open the IAM console at <https://console.aws.amazon.com/iam/>.
2. In the left navigation pane, choose **Roles**.
3. On the **Roles** page, choose **Create role**.
4. On the **Select trusted entity** page, do the following:
  - a. In the **Trusted entity type** section, choose **AWS service**.
  - b. Under **Use case**, choose **EC2**.
  - c. Choose **Next**.
5. On the **Add permissions** page, do the following:
  - a. In the **Filter policies** box, enter **AmazonEKSWorkerNodePolicy**.
  - b. Select the check box to the left of **AmazonEKSWorkerNodePolicy** in the search results.
  - c. Choose **Clear filters**.
  - d. In the **Filter policies** box, enter **AmazonEC2ContainerRegistryReadOnly**.
  - e. Select the check box to the left of **AmazonEC2ContainerRegistryReadOnly** in the search results.
6. On the **Name, review, and create** page, do the following:
  - a. For **Role name**, enter a unique name for your role, such as **AmazonEKSNodeRole**.
  - b. For **Description**, replace the current text with descriptive text such as **Amazon EKS - Node role**.
  - c. Under **Add tags (Optional)**, add metadata to the role by attaching tags as key-value pairs. For more information about using tags in IAM, see [Tagging IAM Entities](#) in the *IAM User Guide*.
  - d. Choose **Create role**.

## AWS CLI

1. Save the following contents to a file named *node-role-trust-relationship.json*.

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

2. Create the IAM role.

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

3. Attach two required IAM managed policies to the IAM role.

```
aws iam attach-role-policy \
--policy-arn arn:aws:iam::aws:policy/AmazonEKSWorkerNodePolicy \
--role-name AmazonEKSNodeRole
aws iam attach-role-policy \
--policy-arn arn:aws:iam::aws:policy/AmazonEC2ContainerRegistryReadOnly \
--role-name AmazonEKSNodeRole
```

4. Attach one of the following IAM policies to the IAM role depending on which IP family you created your cluster with. The policy must be attached to this role or to a role associated to the Kubernetes aws-node service account that's used for the Amazon EKS VPC CNI plugin. We recommend assigning the policy to the role associated to the Kubernetes service account. To assign the policy to the role associated to the Kubernetes service account, see [Configuring the Amazon VPC CNI plugin to use IAM roles for service accounts \(p. 277\)](#).

- IPv4

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

- IPv6

1. Copy the following text and save it to a file named *vpc-cni-ipv6-policy.json*.

```
{
 "Version": "2012-10-17",
 "Statement": [
 {
 "Effect": "Allow",
 "Action": [
 "ec2:AssignIpv6Addresses",
 "ec2:DescribeInstances",
 "ec2:DescribeTags",
 "ec2:DescribeNetworkInterfaces",
 "ec2:DescribeInstanceTypes"
],
 "Resource": "*"
 }
]
}
```

```
 },
 {
 "Effect": "Allow",
 "Action": [
 "ec2:CreateTags"
],
 "Resource": [
 "arn:aws:ec2:*.*:network-interface/*"
]
 }
]
```

2. Create the IAM policy.

```
aws iam create-policy \
--policy-name AmazonEKS_CNT_IPv6_Policy \
--policy-document file://vpc-cni-ipv6-policy.json
```

3. Attach the IAM policy to the IAM role. Replace **111122223333** with your account ID.

```
aws iam attach-role-policy \
--policy-arn arn:aws:iam::111122223333:policy/AmazonEKS_CNT_IPv6_Policy \
--role-name AmazonEKSNodeRole
```

## Amazon EKS pod execution IAM role

The Amazon EKS pod execution role is required to run pods on AWS Fargate infrastructure.

When your cluster creates pods on AWS Fargate infrastructure, the 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.

When you create a Fargate profile, you must specify a pod execution role for the Amazon EKS components that run on the Fargate infrastructure using the profile. This role is added to the cluster's Kubernetes [Role based access control](#) (RBAC) for authorization. This allows the `kubelet` that's running on the Fargate infrastructure to register with your Amazon EKS cluster so that it can appear in your cluster as a node.

**Note**

The Fargate profile must have a different IAM role than Amazon EC2 node groups.

**Important**

The containers running in the Fargate pod can't assume the IAM permissions associated with a pod execution role. To give the containers in your Fargate pod permissions to access other AWS services, you must use [IAM roles for service accounts \(p. 442\)](#).

Before you create a Fargate profile, you must create an IAM role with the [AmazonEKSFargatePodExecutionRolePolicy](#).

## Check for a correctly configured existing pod execution role

You can use the following procedure to check and see if your account already has a correctly configured Amazon EKS pod execution role. To avoid a confused deputy security problem, it's important that the role restricts access based on `SourceArn`. You can modify the execution role as needed to include support for Fargate profiles on other clusters.

### To check for an Amazon EKS pod execution role in the IAM console

1. Open the IAM console at <https://console.aws.amazon.com/iam/>.
2. In the left navigation pane, choose **Roles**.
3. On the **Roles** page, search the list of roles for **AmazonEKSFargatePodExecutionRole**. If the role doesn't exist, see [Creating the Amazon EKS pod execution role \(p. 478\)](#) to create the role. If the role does exist, choose the role.
4. On the **AmazonEKSFargatePodExecutionRole** page, do the following:
  - a. Choose **Permissions**.
  - b. Ensure that the **AmazonEKSFargatePodExecutionRolePolicy** Amazon managed policy is attached to the role.
  - c. Choose **Trust relationships**.
  - d. Choose **Edit trust policy**.
5. On the **Edit trust policy** page, verify that the trust relationship contains the following policy and has a line for Fargate profiles on your cluster. If so, choose **Cancel**.

```
{
 "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"
 }
]
}
```

If the policy matches but doesn't have a line specifying the Fargate profiles on your cluster, you can add the following line at the top of the `ArnLike` object. Replace `region-code` with the AWS Region that your cluster is in, `111122223333` with your account ID, and `my-cluster` with the name of your cluster.

```
"aws:SourceArn": "arn:aws:eks:region-code:111122223333:fargateprofile/my-cluster/*",
```

If the policy doesn't match, copy the full previous policy into the form and choose **Update policy**. 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 \*.

## Creating the Amazon EKS pod execution role

If you don't already have the Amazon EKS pod execution role for your cluster, you can use the AWS Management Console or the AWS CLI to create it.

AWS Management Console

**To create an AWS Fargate pod execution role with the AWS Management Console**

1. Open the IAM console at <https://console.aws.amazon.com/iam/>.
2. In the left navigation pane, choose **Roles**.
3. On the **Roles** page, choose **Create role**.
4. On the **Select trusted entity** page, do the following:
  - a. In the **Trusted entity type** section, choose **AWS service**.
  - b. From the **Use cases for other AWS services** dropdown list, choose **EKS**.
  - c. Choose **EKS - Fargate pod**.
  - d. Choose **Next**.
5. On the **Add permissions** page, choose **Next**.
6. On the **Name, review, and create** page, do the following:
  - a. For **Role name**, enter a unique name for your role, such as **AmazonEKSFargatePodExecutionRole**.
  - b. Under **Add tags (Optional)**, add metadata to the role by attaching tags as key–value pairs. For more information about using tags in IAM, see [Tagging IAM Entities](#) in the *IAM User Guide*.
  - c. Choose **Create role**.
7. On the **Roles** page, search the list of roles for **AmazonEKSFargatePodExecutionRole**. Choose the role.
8. On the **AmazonEKSFargatePodExecutionRole** page, do the following:
  - a. Choose **Trust relationships**.
  - b. Choose **Edit trust policy**.
9. On the **Edit trust policy** page, do the following:
  - a. Copy and paste the following contents into the **Edit trust policy** form. 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. Choose **Update policy**.

## AWS CLI

### To create an AWS Fargate pod execution role with the AWS CLI

1. Copy and paste 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"
 }
]
}
```

2. Create a pod execution IAM role.

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

3. 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
```

## Amazon EKS connector IAM role

You can connect Kubernetes clusters to view them in your AWS Management Console. To connect to a Kubernetes cluster, create an IAM role.

### Check for an existing connector role

You can use the following procedure to check and see if your account already has the Amazon EKS connector role.

#### To check for the `AmazonEKSCollectorAgentRole` in the IAM console

1. Open the IAM console at <https://console.aws.amazon.com/iam/>.
2. In the left navigation pane, choose **Roles**.
3. Search the list of roles for `AmazonEKSCollectorAgentRole`. If a role that includes `AmazonEKSCollectorAgentRole` doesn't exist, then see [Creating the Amazon EKS connector](#)

[agent role \(p. 481\)](#) to create the role. If a role that includes `AmazonEKSConnectorAgentRole` does exist, then select the role to view the attached policies.

4. Choose **Permissions**.
5. Ensure that the **AmazonEKSClusterPolicy** managed policy is attached to the role. If the policy is attached, your Amazon EKS cluster role is properly configured.
6. Choose **Trust relationships**, and then choose **Edit trust policy**.
7. Verify that the trust relationship contains the following policy. If the trust relationship matches the policy below, choose **Cancel**. If the trust relationship doesn't match, copy the policy into the **Edit trust policy** window and choose **Update policy**.

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

## Creating the Amazon EKS connector agent role

You can use the AWS Management Console or AWS CloudFormation to create the connector agent role. Select the tab with the name of the tool that you want to use to create the role.

### AWS CLI

1. Create a file named `eks-connector-agent-trust-policy.json` that contains the following JSON to use for the IAM role.

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

2. Create a file named `eks-connector-agent-policy.json` that contains the following JSON to use for the IAM role.

```
{
 "Version": "2012-10-17",
 "Statement": [
 {
 "Sid": "SsmControlChannel",
 "Effect": "Allow",
 "Action": [
 "sts:AssumeRole"
]
 }
]
}
```

```

 "ssmmessages:CreateControlChannel"
],
 "Resource": "arn:aws:eks:*:cluster/*"
},
{
 "Sid": "ssmDataplaneOperations",
 "Effect": "Allow",
 "Action": [
 "ssmmessages:CreateDataChannel",
 "ssmmessages:OpenDataChannel",
 "ssmmessages:OpenControlChannel"
],
 "Resource": "*"
}
]
}

```

3. Create the Amazon EKS Connector agent role using the trust policy and policy you created in the previous list items.

```
aws iam create-role \
--role-name AmazonEKSConnectorAgentRole \
--assume-role-policy-document file://eks-connector-agent-trust-policy.json
```

4. Attach the policy to your Amazon EKS Connector agent role.

```
aws iam put-role-policy \
--role-name AmazonEKSConnectorAgentRole \
--policy-name AmazonEKSConnectorAgentPolicy \
--policy-document file://eks-connector-agent-policy.json
```

## AWS CloudFormation

### To create your Amazon EKS connector agent role with AWS CloudFormation.

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

#### Note

This template also creates the service-linked role that would otherwise be created when the `registerCluster` API is called. See [Using roles to connect a Kubernetes cluster to Amazon EKS \(p. 470\)](#) for details.

```

AWSTemplateFormatVersion: '2010-09-09'
Description: 'Provisions necessary resources needed to register clusters in EKS'
Parameters: {}
Resources:
 EKSConnectorSLR:
 Type: AWS::IAM::ServiceLinkedRole
 Properties:
 AWSServiceName: eks-connector.amazonaws.com

 EKSConnectorAgentRole:
 Type: AWS::IAM::Role
 Properties:
 AssumeRolePolicyDocument:
 Version: '2012-10-17'
 Statement:
 - Effect: Allow
 Action: ['sts:AssumeRole']
 Principal:
```

```

Service: 'ssm.amazonaws.com'

EKSCollectorAgentPolicy:
 Type: AWS::IAM::Policy
 Properties:
 PolicyName: EKSCollectorAgentPolicy
 Roles:
 - {Ref: 'EKSCollectorAgentRole'}
 PolicyDocument:
 Version: '2012-10-17'
 Statement:
 - Effect: 'Allow'
 Action: ['ssmmessages:CreateControlChannel']
 Resource:
 - Fn::Sub: 'arn:${AWS::Partition}:eks:/*:cluster/*'
 - Effect: 'Allow'
 Action: ['ssmmessages:CreateDataChannel', 'ssmmessages:OpenDataChannel',
 'ssmmessages:OpenControlChannel']
 Resource: "*"
 Outputs:
 EKSCollectorAgentRoleArn:
 Description: The agent role that EKS connector uses to communicate with AWS
 services.
 Value: !GetAtt EKSCollectorAgentRole.Arn

```

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

## AWS managed policies for Amazon Elastic Kubernetes Service

To add permissions to users, groups, and roles, it is easier to use AWS managed policies than to write policies yourself. It takes time and expertise to [create IAM customer managed policies](#) that provide your team with only the permissions they need. To get started quickly, you can use our AWS managed policies. These policies cover common use cases and are available in your AWS account. For more information about AWS managed policies, see [AWS managed policies](#) in the *IAM User Guide*.

AWS services maintain and update AWS managed policies. You can't change the permissions in AWS managed policies. Services occasionally add additional permissions to an AWS managed policy to support new features. This type of update affects all identities (users, groups, and roles) where the policy is attached. Services are most likely to update an AWS managed policy when a new feature is launched or when new operations become available. Services do not remove permissions from an AWS managed policy, so policy updates won't break your existing permissions.

Additionally, AWS supports managed policies for job functions that span multiple services. For example, the **ReadOnlyAccess** AWS managed policy provides read-only access to all AWS services and resources. When a service launches a new feature, AWS adds read-only permissions for new operations and

resources. For a list and descriptions of job function policies, see [AWS managed policies for job functions in the IAM User Guide](#).

## AWS managed policy: AmazonEKS\_CNI\_Policy

You can attach the `AmazonEKS_CNI_Policy` to your IAM entities. Before you create an Amazon EC2 node group, this policy must be attached to either the [node IAM role \(p. 474\)](#), or to an IAM role that's used specifically by the AWS VPC CNI plugin. This is so that it can perform actions on your behalf. We recommend that you attach the policy to a role that's used only by the CNI plugin. For more information, see [Pod networking using the Amazon VPC Container network interface \(CNI\) plugin \(p. 276\)](#) and [Configuring the Amazon VPC CNI plugin to use IAM roles for service accounts \(p. 277\)](#).

### Permissions details

This policy includes the following permissions that allow Amazon EKS to complete the following tasks:

- **ec2** – Allows the Amazon VPC CNI plugin to perform actions such as provisioning Elastic Network Interfaces and IP addresses for pods to provide networking for applications that run in Amazon EKS.

```
{
 "Version": "2012-10-17",
 "Statement": [
 {
 "Effect": "Allow",
 "Action": [
 "ec2:AssignPrivateIpAddresses",
 "ec2:AttachNetworkInterface",
 "ec2>CreateNetworkInterface",
 "ec2>DeleteNetworkInterface",
 "ec2:DescribeInstances",
 "ec2:DescribeTags",
 "ec2:DescribeNetworkInterfaces",
 "ec2:DescribeInstanceTypes",
 "ec2:DetachNetworkInterface",
 "ec2:ModifyNetworkInterfaceAttribute",
 "ec2:UnassignPrivateIpAddresses"
],
 "Resource": "*"
 },
 {
 "Effect": "Allow",
 "Action": [
 "ec2>CreateTags"
],
 "Resource": [
 "arn:aws:ec2:*:network-interface/*"
]
 }
]
}
```

## AWS managed policy: AmazonEKSClusterPolicy

You can attach `AmazonEKSClusterPolicy` to your IAM entities. Before creating a cluster, you must have a [cluster IAM role \(p. 472\)](#) with this policy attached. Kubernetes clusters that are managed by Amazon EKS make calls to other AWS services on your behalf. They do this to manage the resources that you use with the service.

This policy includes the following permissions that allow Amazon EKS to complete the following tasks:

- **autoscaling** – Read and update the configuration of an Auto Scaling group. These permissions aren't used by Amazon EKS but remain in the policy for backwards compatibility.
- **ec2** – Work with volumes and network resources that are associated to Amazon EC2 nodes. This is required so that the Kubernetes control plane can join instances to a cluster and dynamically provision and manage Amazon EBS volumes that are requested by Kubernetes persistent volumes.
- **elasticloadbalancing** – Work with Elastic Load Balancers and add nodes to them as targets. This is required so that the Kubernetes control plane can dynamically provision Elastic Load Balancers requested by Kubernetes services.
- **iam** – Create a service-linked role. This is required so that the Kubernetes control plane can dynamically provision Elastic Load Balancers that are requested by Kubernetes services.
- **kms** – Read a key from AWS KMS. This is required for the Kubernetes control plane to support [secrets encryption](#) of Kubernetes secrets stored in etcd.

```
{
 "Version": "2012-10-17",
 "Statement": [
 {
 "Effect": "Allow",
 "Action": [
 "autoscaling:DescribeAutoScalingGroups",
 "autoscaling:UpdateAutoScalingGroup",
 "ec2:AttachVolume",
 "ec2:AuthorizeSecurityGroupIngress",
 "ec2:CreateRoute",
 "ec2:CreateSecurityGroup",
 "ec2:CreateTags",
 "ec2:CreateVolume",
 "ec2:DeleteRoute",
 "ec2:DeleteSecurityGroup",
 "ec2:DeleteVolume",
 "ec2:DescribeInstances",
 "ec2:DescribeRouteTables",
 "ec2:DescribeSecurityGroups",
 "ec2:DescribeSubnets",
 "ec2:DescribeVolumes",
 "ec2:DescribeVolumesModifications",
 "ec2:DescribeVpcs",
 "ec2:DescribeDhcpOptions",
 "ec2:DescribeNetworkInterfaces",
 "ec2:DetachVolume",
 "ec2:ModifyInstanceAttribute",
 "ec2:ModifyVolume",
 "ec2:RevokeSecurityGroupIngress",
 "ec2:DescribeAccountAttributes",
 "ec2:DescribeAddresses",
 "ec2:DescribeInternetGateways",
 "elasticloadbalancing:AddTags",
 "elasticloadbalancing:ApplySecurityGroupsToLoadBalancer",
 "elasticloadbalancing:AttachLoadBalancerToSubnets",
 "elasticloadbalancing:ConfigureHealthCheck",
 "elasticloadbalancing:CreateListener",
 "elasticloadbalancing:CreateLoadBalancer",
 "elasticloadbalancing:CreateLoadBalancerListeners",
 "elasticloadbalancing:CreateLoadBalancerPolicy",
 "elasticloadbalancing:CreateTargetGroup",
 "elasticloadbalancing:DeleteListener",
 "elasticloadbalancing:DeleteLoadBalancer",
 "elasticloadbalancing:DeleteLoadBalancerListeners",
 "elasticloadbalancing:DeleteTargetGroup",
 "elasticloadbalancing:DeregisterInstancesFromLoadBalancer",
 "elasticloadbalancing:DeregisterTargets",
 "elasticloadbalancing:DescribeLoadBalancers"
]
 }
]
}
```

```

 "elasticloadbalancing:DescribeListeners",
 "elasticloadbalancing:DescribeLoadBalancerAttributes",
 "elasticloadbalancing:DescribeLoadBalancerPolicies",
 "elasticloadbalancing:DescribeLoadBalancers",
 "elasticloadbalancing:DescribeTargetGroupAttributes",
 "elasticloadbalancing:DescribeTargetGroups",
 "elasticloadbalancing:DescribeTargetHealth",
 "elasticloadbalancing:DetachLoadBalancerFromSubnets",
 "elasticloadbalancing:ModifyListener",
 "elasticloadbalancing:ModifyLoadBalancerAttributes",
 "elasticloadbalancing:ModifyTargetGroup",
 "elasticloadbalancing:ModifyTargetGroupAttributes",
 "elasticloadbalancing:RegisterInstancesWithLoadBalancer",
 "elasticloadbalancing:RegisterTargets",
 "elasticloadbalancing:SetLoadBalancerPoliciesForBackendServer",
 "elasticloadbalancing:SetLoadBalancerPoliciesOfListener",
 "kms:DescribeKey"
],
 "Resource": "*"
},
{
 "Effect": "Allow",
 "Action": "iam:CreateServiceLinkedRole",
 "Resource": "*",
 "Condition": {
 "StringEquals": {
 "iam:AWSServiceName": "elasticloadbalancing.amazonaws.com"
 }
 }
}
]
}

```

## AWS managed policy: [AmazonEKSFargatePodExecutionRolePolicy](#)

You can attach [AmazonEKSFargatePodExecutionRolePolicy](#) to your IAM entities. Before you can create a Fargate profile, you must create a Fargate pod execution role and attach this policy to it. For more information, see [Create a Fargate pod execution role \(p. 156\)](#) and [AWS Fargate profile \(p. 158\)](#).

This policy grants the role the permissions that provide access to other AWS service resources that are required to run Amazon EKS pods on Fargate.

### Permissions details

This policy includes the following permissions that allow Amazon EKS to complete the following tasks:

- **ecr** – Allows Pods that are running on Fargate to pull container images that are stored in Amazon ECR.

```
{
 "Version": "2012-10-17",
 "Statement": [
 {
 "Effect": "Allow",
 "Action": [
 "ecr:GetAuthorizationToken",
 "ecr:BatchCheckLayerAvailability",
 "ecr:GetDownloadUrlForLayer",
 "ecr:BatchGetImage"
],
 "Resource": "*"
 }
]
}
```

```
]
 }
```

## AWS managed policy: AmazonEKSForFargateServiceRolePolicy

You can't attach `AmazonEKSForFargateServiceRolePolicy` to your IAM entities. This policy is attached to a service-linked role that allows Amazon EKS to perform actions on your behalf. For more information, see [AWS Service Role for Amazon EKS for Fargate](#).

This policy grants necessary permissions to Amazon EKS to run Fargate tasks. The policy is only used if you have Fargate nodes.

### Permissions details

This policy includes the following permissions that allow Amazon EKS to complete the following tasks.

- **ec2** – Create and delete Elastic Network Interfaces and describe Elastic Network Interfaces and resources. This is required so that the Amazon EKS Fargate service can configure the VPC networking that's required for Fargate Pods.

```
{
 "Version": "2012-10-17",
 "Statement": [
 {
 "Effect": "Allow",
 "Action": [
 "ec2:CreateNetworkInterface",
 "ec2:CreateNetworkInterfacePermission",
 "ec2:DeleteNetworkInterface",
 "ec2:DescribeNetworkInterfaces",
 "ec2:DescribeSecurityGroups",
 "ec2:DescribeSubnets",
 "ec2:DescribeVpcs",
 "ec2:DescribeDhcpOptions",
 "ec2:DescribeRouteTables"
],
 "Resource": "*"
 }
]
}
```

## AWS managed policy: AmazonEKSServicePolicy

You can attach `AmazonEKSServicePolicy` to your IAM entities. Clusters that were created before April 16, 2020, required you to create an IAM role and attach this policy to it. Clusters that were created on or after April 16, 2020, don't require you to create a role and don't require you to assign this policy. When you create a cluster using an IAM principal that has the `iam:CreateServiceLinkedRole` permission, the [AWS Service Role for Amazon EKS \(p. 465\)](#) service-linked role is automatically created for you. The service-linked role has the [AWS managed policy: AmazonEKSServiceRolePolicy \(p. 489\)](#) attached to it.

This policy allows Amazon EKS to create and manage the necessary resources to operate Amazon EKS clusters.

### Permissions details

This policy includes the following permissions that allow Amazon EKS to complete the following tasks.

- **eks** – Update the Kubernetes version of your cluster after you initiate an update. This permission isn't used by Amazon EKS but remains in the policy for backwards compatibility.

- **ec2** – Work with Elastic Network Interfaces and other network resources and tags. This is required by Amazon EKS to configure networking that facilitates communication between nodes and the Kubernetes control plane.
- **route53** – Associate a VPC with a hosted zone. This is required by Amazon EKS to enable private endpoint networking for your Kubernetes cluster API server.
- **logs** – Log events. This is required so that Amazon EKS can ship Kubernetes control plane logs to CloudWatch.
- **iam** – Create a service-linked role. This is required so that Amazon EKS can create the [AWSServiceRoleForAmazonEKS \(p. 465\)](#) service-linked role on your behalf.

```
{
 "Version": "2012-10-17",
 "Statement": [
 {
 "Effect": "Allow",
 "Action": [
 "ec2:CreateNetworkInterface",
 "ec2:CreateNetworkInterfacePermission",
 "ec2:DeleteNetworkInterface",
 "ec2:DescribeInstances",
 "ec2:DescribeNetworkInterfaces",
 "ec2:DetachNetworkInterface",
 "ec2:DescribeSecurityGroups",
 "ec2:DescribeSubnets",
 "ec2:DescribeVpcs",
 "ec2:ModifyNetworkInterfaceAttribute",
 "iam>ListAttachedRolePolicies",
 "eks:UpdateClusterVersion"
],
 "Resource": "*"
 },
 {
 "Effect": "Allow",
 "Action": [
 "ec2>CreateTags",
 "ec2>DeleteTags"
],
 "Resource": [
 "arn:aws:ec2::::vpc/*",
 "arn:aws:ec2::::subnet/*"
]
 },
 {
 "Effect": "Allow",
 "Action": "route53:AssociateVPCWithHostedZone",
 "Resource": "*"
 },
 {
 "Effect": "Allow",
 "Action": "logs>CreateLogGroup",
 "Resource": "*"
 },
 {
 "Effect": "Allow",
 "Action": [
 "logs>CreateLogStream",
 "logs>DescribeLogStreams"
],
 "Resource": "arn:aws:logs::::log-group:/aws/eks/*:*"
 },
 {
 "Effect": "Allow",
 "Action": [
 "logs:PutLogEvents"
],
 "Resource": "arn:aws:logs::::log-group:/aws/eks/*:*"
 }
]
}
```

```

 "Action": "logs:PutLogEvents",
 "Resource": "arn:aws:logs:***:log-group:/aws/eks/*:*:*"
 },
 {
 "Effect": "Allow",
 "Action": "iam:CreateServiceLinkedRole",
 "Resource": "*",
 "Condition": {
 "StringLike": {
 "iam:AWSServiceName": "eks.amazonaws.com"
 }
 }
 }
]
}

```

## AWS managed policy: AmazonEKSServiceRolePolicy

You can't attach `AmazonEKSServiceRolePolicy` to your IAM entities. This policy is attached to a service-linked role that allows Amazon EKS to perform actions on your behalf. For more information, see [Service-Linked Role Permissions for Amazon EKS \(p. 465\)](#). When you create a cluster using an IAM principal that has the `iam:CreateServiceLinkedRole` permission, the [AWSRoleforAmazonEKS \(p. 465\)](#) service-linked role is automatically created for you and this policy is attached to it.

This policy allows the service-linked role to call AWS services on your behalf.

### Permissions details

This policy includes the following permissions that allow Amazon EKS to complete the following tasks.

- **ec2** – Create and describe Elastic Network Interfaces and Amazon EC2 instances, the [cluster security group \(p. 272\)](#), and VPC that are required to create a cluster.
- **iam** – List all of the managed policies that attached to an IAM role. This is required so that Amazon EKS can list and validate all managed policies and permissions required to create a cluster.
- **Associate a VPC with a hosted zone** – This is required by Amazon EKS to enable private endpoint networking for your Kubernetes cluster API server.
- **Log event** – This is required so that Amazon EKS can ship Kubernetes control plane logs to CloudWatch.

```
{
 "Version": "2012-10-17",
 "Statement": [
 {
 "Effect": "Allow",
 "Action": [
 "ec2:CreateNetworkInterface",
 "ec2:DeleteNetworkInterface",
 "ec2:DetachNetworkInterface",
 "ec2:ModifyNetworkInterfaceAttribute",
 "ec2:DescribeInstances",
 "ec2:DescribeNetworkInterfaces",
 "ec2:DescribeSecurityGroups",
 "ec2:DescribeSubnets",
 "ec2:DescribeVpcs",
 "ec2:CreateNetworkInterfacePermission",
 "iam>ListAttachedRolePolicies",
 "ec2>CreateSecurityGroup"
],
 "Resource": "*"
 }
]
}
```

```

},
{
 "Effect": "Allow",
 "Action": [
 "ec2:DeleteSecurityGroup",
 "ec2:RevokeSecurityGroupIngress",
 "ec2:AuthorizeSecurityGroupIngress"
],
 "Resource": "arn:aws:ec2::::security-group/*",
 "Condition": {
 "ForAnyValue:StringLike": {
 "ec2:ResourceTag/Name": "eks-cluster-sg*"
 }
 }
},
{
 "Effect": "Allow",
 "Action": [
 "ec2>CreateTags",
 "ec2>DeleteTags"
],
 "Resource": [
 "arn:aws:ec2::::vpc/*",
 "arn:aws:ec2::::subnet/*"
],
 "Condition": {
 "ForAnyValue:StringLike": {
 "aws:TagKeys": [
 "kubernetes.io/cluster/*"
]
 }
 }
},
{
 "Effect": "Allow",
 "Action": [
 "ec2>CreateTags",
 "ec2>DeleteTags"
],
 "Resource": [
 "arn:aws:ec2::::security-group/*"
],
 "Condition": {
 "ForAnyValue:StringLike": {
 "aws:TagKeys": [
 "kubernetes.io/cluster/*"
],
 "aws:RequestTag/Name": "eks-cluster-sg*"
 }
 }
},
{
 "Effect": "Allow",
 "Action": "route53:AssociateVPCWithHostedZone",
 "Resource": "arn:aws:route53::::hostedzone/*"
},
{
 "Effect": "Allow",
 "Action": "logs>CreateLogGroup",
 "Resource": "arn:aws:logs::::log-group:/aws/eks/*"
},
{
 "Effect": "Allow",
 "Action": [
 "logs>CreateLogStream",
 "logs:DescribeLogStreams"
]
}

```

```
],
 "Resource": "arn:aws:logs:***:log-group:/aws/eks/*:*"
 },
 {
 "Effect": "Allow",
 "Action": "logs:PutLogEvents",
 "Resource": "arn:aws:logs:***:log-group:/aws/eks/*:*:*"
 }
]
```

## AWS managed policy: AmazonEKSVPCResourceController

You can attach the [AmazonEKSVPCResourceController](#) policy to your IAM identities. If you're using [security groups for pods \(p. 313\)](#), you must attach this policy to your [Amazon EKS cluster IAM role \(p. 472\)](#) to perform actions on your behalf.

This policy grants the cluster role permissions to manage Elastic Network Interfaces and IP addresses for nodes.

### Permissions details

This policy includes the following permissions that allow Amazon EKS to complete the following tasks:

- **ec2** – Manage Elastic Network Interfaces and IP addresses to support pod security groups and Windows nodes.

```
{
 "Version": "2012-10-17",
 "Statement": [
 {
 "Effect": "Allow",
 "Action": "ec2:CreateNetworkInterfacePermission",
 "Resource": "*",
 "Condition": {
 "ForAnyValue:StringEquals": {
 "ec2:ResourceTag/eks:eni:owner": "eks-vpc-resource-controller"
 }
 }
 },
 {
 "Effect": "Allow",
 "Action": [
 "ec2:CreateNetworkInterface",
 "ec2:DetachNetworkInterface",
 "ec2:ModifyNetworkInterfaceAttribute",
 "ec2:DeleteNetworkInterface",
 "ec2:AttachNetworkInterface",
 "ec2:UnassignPrivateIpAddresses",
 "ec2:AssignPrivateIpAddresses"
],
 "Resource": "*"
 }
]
}
```

## AWS managed policy: AmazonEKSWorkerNodePolicy

You can attach the [AmazonEKSWorkerNodePolicy](#) to your IAM entities. You must attach this policy to a [node IAM role \(p. 474\)](#) that you specify when you create Amazon EC2 nodes that allow Amazon EKS

to perform actions on your behalf. If you create a node group using `eksctl`, it creates the node IAM role and attaches this policy to the role automatically.

This policy grants Amazon EKS Amazon EC2 nodes permissions to connect to Amazon EKS clusters.

#### Permissions details

This policy includes the following permissions that allow Amazon EKS to complete the following tasks:

- **ec2** – Read instance volume and network information. This is required so that Kubernetes nodes can describe information about Amazon EC2 resources that are required for the node to join the Amazon EKS cluster.
- **eks** – Optionally describe the cluster as part of node bootstrapping.

```
{
 "Version": "2012-10-17",
 "Statement": [
 {
 "Effect": "Allow",
 "Action": [
 "ec2:DescribeInstances",
 "ec2:DescribeInstanceTypes",
 "ec2:DescribeRouteTables",
 "ec2:DescribeSecurityGroups",
 "ec2:DescribeSubnets",
 "ec2:DescribeVolumes",
 "ec2:DescribeVolumesModifications",
 "ec2:DescribeVpcs",
 "eks:DescribeCluster"
],
 "Resource": "*"
 }
]
}
```

## AWS managed policy: [AWS\*ServiceRoleForAmazonEKSNodegroup\*](#)

You can't attach `AWSServiceRoleForAmazonEKSNodegroup` to your IAM entities. This policy is attached to a service-linked role that allows Amazon EKS to perform actions on your behalf. For more information, see [Service-linked role permissions for Amazon EKS \(p. 467\)](#).

This policy grants the `AWSServiceRoleForAmazonEKSNodegroup` role permissions that allow it to create and manage Amazon EC2 node groups in your account.

#### Permissions details

This policy includes the following permissions that allow Amazon EKS to complete the following tasks:

- **ec2** – Work with security groups, tags, and launch templates. This is required for Amazon EKS managed node groups to enable remote access configuration. Additionally, Amazon EKS managed node groups create a launch template on your behalf. This is to configure the Amazon EC2 Auto Scaling group that backs each managed node group.
- **iam** – Create a service-linked role and pass a role. This is required by Amazon EKS managed node groups to manage instance profiles for the role being passed when creating a managed node group. This instance profile is used by Amazon EC2 instances launched as part of a managed node group. Amazon EKS needs to create service-linked roles for other services such as Amazon EC2 Auto Scaling groups. These are used in the creation of a managed node group

- **autoscaling** – Work with security Auto Scaling groups. This is required by Amazon EKS managed node groups to manage the Amazon EC2 Auto Scaling group that backs each managed node group. It's also used to support functionality such as evicting pods when nodes are terminated or recycled during node group updates.

```
{
 "Version": "2012-10-17",
 "Statement": [
 {
 "Sid": "SharedSecurityGroupRelatedPermissions",
 "Effect": "Allow",
 "Action": [
 "ec2:RevokeSecurityGroupIngress",
 "ec2:AuthorizeSecurityGroupEgress",
 "ec2:AuthorizeSecurityGroupIngress",
 "ec2:DescribeInstances",
 "ec2:RevokeSecurityGroupEgress",
 "ec2:DeleteSecurityGroup"
],
 "Resource": "*",
 "Condition": {
 "StringLike": {
 "ec2:ResourceTag/eks": "*"
 }
 }
 },
 {
 "Sid": "EKSCreatedSecurityGroupRelatedPermissions",
 "Effect": "Allow",
 "Action": [
 "ec2:RevokeSecurityGroupIngress",
 "ec2:AuthorizeSecurityGroupEgress",
 "ec2:AuthorizeSecurityGroupIngress",
 "ec2:DescribeInstances",
 "ec2:RevokeSecurityGroupEgress",
 "ec2:DeleteSecurityGroup"
],
 "Resource": "*",
 "Condition": {
 "StringLike": {
 "ec2:ResourceTag/eks:nodename": "*"
 }
 }
 },
 {
 "Sid": "LaunchTemplateRelatedPermissions",
 "Effect": "Allow",
 "Action": [
 "ec2>DeleteLaunchTemplate",
 "ec2>CreateLaunchTemplateVersion"
],
 "Resource": "*",
 "Condition": {
 "StringLike": {
 "ec2:ResourceTag/eks:nodename": "*"
 }
 }
 },
 {
 "Sid": "AutoscalingRelatedPermissions",
 "Effect": "Allow",
 "Action": [
 "autoscaling:UpdateAutoScalingGroup",
 "autoscaling:DescribeAutoScalingGroups",
 "autoscaling:DescribeAutoScalingInstances",
 "autoscaling:PutScalingPolicy",
 "autoscaling:PutMetricFilter",
 "autoscaling:PutScheduledUpdateGroupAction",
 "autoscaling:DeleteScalingPolicy",
 "autoscaling:DeleteMetricFilter",
 "autoscaling:DeleteScheduledUpdateGroupAction"
],
 "Resource": "*",
 "Condition": {
 "StringLike": {
 "ec2:ResourceTag/eks:nodename": "*"
 }
 }
 }
]
}
```

```

 "autoscaling>DeleteAutoScalingGroup",
 "autoscaling>TerminateInstanceInAutoScalingGroup",
 "autoscaling>CompleteLifecycleAction",
 "autoscaling>PutLifecycleHook",
 "autoscaling>PutNotificationConfiguration",
 "autoscaling>EnableMetricsCollection"
],
 "Resource": "arn:aws:autoscaling:*::*:autoScalingGroupName/eks-*"
},
{
 "Sid": "AllowAutoscalingToCreateSLR",
 "Effect": "Allow",
 "Condition": {
 "StringEquals": {
 "iam:AWSServiceName": "autoscaling.amazonaws.com"
 }
 },
 "Action": "iam>CreateServiceLinkedRole",
 "Resource": "*"
},
{
 "Sid": "AllowASGCreationByEKS",
 "Effect": "Allow",
 "Action": [
 "autoscaling>CreateOrUpdateTags",
 "autoscaling>CreateAutoScalingGroup"
],
 "Resource": "*",
 "Condition": {
 "ForAnyValue:StringEquals": {
 "aws:TagKeys": [
 "eks",
 "eks:cluster-name",
 "eks:nodename"
]
 }
 }
},
{
 "Sid": "AllowPassRoleToAutoscaling",
 "Effect": "Allow",
 "Action": "iam:PassRole",
 "Resource": "*",
 "Condition": {
 "StringEquals": {
 "iam:PassedToService": "autoscaling.amazonaws.com"
 }
 }
},
{
 "Sid": "AllowPassRoleToEC2",
 "Effect": "Allow",
 "Action": "iam:PassRole",
 "Resource": "*",
 "Condition": {
 "StringEqualsIfExists": {
 "iam:PassedToService": [
 "ec2.amazonaws.com",
 "ec2.amazonaws.com"
]
 }
 }
},
{
 "Sid": "PermissionsToManageResourcesForNodegroups",
 "Effect": "Allow",

```

```

 "Action": [
 "iam:GetRole",
 "ec2>CreateLaunchTemplate",
 "ec2:DescribeInstances",
 "iam:GetInstanceProfile",
 "ec2:DescribeLaunchTemplates",
 "autoscaling:DescribeAutoScalingGroups",
 "ec2>CreateSecurityGroup",
 "ec2:DescribeLaunchTemplateVersions",
 "ec2:RunInstances",
 "ec2:DescribeSecurityGroups",
 "ec2:GetConsoleOutput",
 "ec2:DescribeRouteTables",
 "ec2:DescribeSubnets"
],
 "Resource": "*"
},
{
 "Sid": "PermissionsToCreateAndManageInstanceProfiles",
 "Effect": "Allow",
 "Action": [
 "iam>CreateInstanceProfile",
 "iam>DeleteInstanceProfile",
 "iam:RemoveRoleFromInstanceProfile",
 "iam:AddRoleToInstanceProfile"
],
 "Resource": "arn:aws:iam::*:instance-profile/eks-*"
},
{
 "Sid": "PermissionsToManageEKSAndKubernetesTags",
 "Effect": "Allow",
 "Action": [
 "ec2>CreateTags",
 "ec2>DeleteTags"
],
 "Resource": "*",
 "Condition": {
 "ForAnyValue:StringLike": {
 "aws:TagKeys": [
 "eks",
 "eks:cluster-name",
 "eks:nodename",
 "kubernetes.io/cluster/*"
]
 }
 }
}
]
}

```

## AWS managed policy: AmazonEBSCSIDriverPolicy

The [AmazonEBSCSIDriverPolicy](#) allows the Amazon EBS Container Storage Interface (CSI) driver to create, modify, attach, detach, and delete volumes on your behalf. It also grants the EBS CSI driver permissions to create and delete snapshots, and to list your instances, volumes, and snapshots.

The [AmazonEBSCSIDriverPolicy](#) includes the following permissions:

```
{
 "Version": "2012-10-17",
 "Statement": [
 {
 "Effect": "Allow",

```

```

 "Action": [
 "ec2:CreateSnapshot",
 "ec2:AttachVolume",
 "ec2:DetachVolume",
 "ec2:ModifyVolume",
 "ec2:DescribeAvailabilityZones",
 "ec2:DescribeInstances",
 "ec2:DescribeSnapshots",
 "ec2:DescribeTags",
 "ec2:DescribeVolumes",
 "ec2:DescribeVolumesModifications"
],
 "Resource": "*"
},
{
 "Effect": "Allow",
 "Action": [
 "ec2:CreateTags"
],
 "Resource": [
 "arn:aws:ec2:*::volume/*",
 "arn:aws:ec2:*::snapshot/*"
],
 "Condition": {
 "StringEquals": {
 "ec2:CreateAction": [
 "CreateVolume",
 "CreateSnapshot"
]
 }
 }
},
{
 "Effect": "Allow",
 "Action": [
 "ec2:DeleteTags"
],
 "Resource": [
 "arn:aws:ec2:*::volume/*",
 "arn:aws:ec2:*::snapshot/*"
]
},
{
 "Effect": "Allow",
 "Action": [
 "ec2>CreateVolume"
],
 "Resource": "*",
 "Condition": {
 "StringLike": {
 "aws:RequestTag/ebs.csi.aws.com/cluster": "true"
 }
 }
},
{
 "Effect": "Allow",
 "Action": [
 "ec2:CreateVolume"
],
 "Resource": "*",
 "Condition": {
 "StringLike": {
 "aws:RequestTag/CSIVolumeName": "*"
 }
 }
},

```

```
{
 "Effect": "Allow",
 "Action": [
 "ec2:CreateVolume"
],
 "Resource": "*",
 "Condition": {
 "StringLike": {
 "aws:RequestTag/kubernetes.io/cluster/*": "owned"
 }
 }
},
{
 "Effect": "Allow",
 "Action": [
 "ec2:DeleteVolume"
],
 "Resource": "*",
 "Condition": {
 "StringLike": {
 "ec2:ResourceTag/ebs.csi.aws.com/cluster": "true"
 }
 }
},
{
 "Effect": "Allow",
 "Action": [
 "ec2:DeleteVolume"
],
 "Resource": "*",
 "Condition": {
 "StringLike": {
 "ec2:ResourceTag/CSIVolumeName": "*"
 }
 }
},
{
 "Effect": "Allow",
 "Action": [
 "ec2:DeleteVolume"
],
 "Resource": "*",
 "Condition": {
 "StringLike": {
 "ec2:ResourceTag/kubernetes.io/cluster/*": "owned"
 }
 }
},
{
 "Effect": "Allow",
 "Action": [
 "ec2:DeleteSnapshot"
],
 "Resource": "*",
 "Condition": {
 "StringLike": {
 "ec2:ResourceTag/CSIVolumeSnapshotName": "*"
 }
 }
},
{
 "Effect": "Allow",
 "Action": [
 "ec2:DeleteSnapshot"
],
 "Resource": "*",
}
```

```

 "Condition": {
 "StringLike": {
 "ec2:ResourceTag/ebs.csi.aws.com/cluster": "true"
 }
 }
]
}

```

## Amazon EKS updates to AWS managed policies

View details about updates to AWS managed policies for Amazon EKS since this service began tracking these changes. For automatic alerts about changes to this page, subscribe to the RSS feed on the [Amazon EKS Document history page](#).

| Change                                                                             | Description                                                                                                                                                                                                                        | Date              |
|------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------|
| Introduced <a href="#">AmazonEBSCSIDriverPolicy (p. 495)</a>                       | AWS introduced the <a href="#">AmazonEBSCSIDriverPolicy</a> .                                                                                                                                                                      | April 4, 2022     |
| Added permissions to <a href="#">AmazonEKSTaskNodePolicy (p. 181)</a>              | Added <code>ec2:DescribeInstanceTypes</code> to <code>Enable</code> permission to allow Amazon EKS-optimized AMIs that can auto discover instance level properties.                                                                | March 21, 2022    |
| Added permissions to <a href="#">AWSServiceRoleForAmazonEKSNetworking (p. 102)</a> | Added <code>AmazonEKSTaskNodePolicy</code> permission to allow Amazon EKS to enable metrics collection.                                                                                                                            | December 13, 2021 |
| Added permissions to <a href="#">AmazonEKSClusterPolicy (p. 494)</a>               | Added <code>ec2:DescribeAccountAttributes</code> , <code>ec2:DescribeAddresses</code> , and <code>ec2:DescribeInternetGateways</code> permissions to allow Amazon EKS to create a service-linked role for a Network Load Balancer. | June 17, 2021     |
| Amazon EKS started tracking changes.                                               | Amazon EKS started tracking changes for its AWS managed policies.                                                                                                                                                                  | June 17, 2021     |

## Troubleshooting Amazon EKS identity and access

To diagnose and fix common issues that you might encounter when working with Amazon EKS and IAM see [Troubleshooting IAM \(p. 554\)](#).

## Compliance validation for Amazon Elastic Kubernetes Service

Third-party auditors assess the security and compliance of AWS services as part of multiple AWS compliance programs, such as SOC, PCI, FedRAMP, and HIPAA.

To learn whether Amazon EKS or other AWS services are in scope of specific compliance programs, see [AWS Services in Scope by Compliance Program](#). For general information, see [AWS Compliance Programs](#).

You can download third-party audit reports using AWS Artifact. For more information, see [Downloading Reports in AWS Artifact](#).

Your compliance responsibility when using AWS services is determined by the sensitivity of your data, your company's compliance objectives, and applicable laws and regulations. AWS provides the following resources to help with compliance:

- [Security and Compliance Quick Start Guides](#) – These deployment guides discuss architectural considerations and provide steps for deploying baseline environments on AWS that are security and compliance focused.
- [Architecting for HIPAA Security and Compliance Whitepaper](#) – This whitepaper describes how companies can use AWS to create HIPAA-eligible applications.

**Note**

Not all AWS services are HIPAA eligible. For more information, see the [HIPAA Eligible Services Reference](#).

- [AWS Compliance Resources](#) – This collection of workbooks and guides might apply to your industry and location.
- [Evaluating Resources with Rules](#) in the *AWS Config Developer Guide* – The AWS Config service assesses how well your resource configurations comply with internal practices, industry guidelines, and regulations.
- [AWS Security Hub](#) – This AWS service provides a comprehensive view of your security state within AWS that helps you check your compliance with security industry standards and best practices.
- [AWS Audit Manager](#) – This AWS service helps you continuously audit your AWS usage to simplify how you manage risk and compliance with regulations and industry standards.

## Resilience in Amazon EKS

The AWS global infrastructure is built around AWS Regions and Availability Zones. AWS Regions provide multiple physically separated and isolated Availability Zones, which are connected with low-latency, high-throughput, and highly redundant networking. With Availability Zones, you can design and operate applications and databases that automatically fail over between Availability Zones without interruption. Availability Zones are more highly available, fault tolerant, and scalable than traditional single or multiple data center infrastructures.

Amazon EKS runs and scales the Kubernetes control plane across multiple AWS Availability Zones to ensure high availability. Amazon EKS automatically scales control plane instances based on load, detects and replaces unhealthy control plane instances, and automatically patches the control plane. After you initiate a version update, Amazon EKS updates your control plane for you, maintaining high availability of the control plane during the update.

This 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](#).

For more information about AWS Regions and Availability Zones, see [AWS global infrastructure](#).

## Infrastructure security in Amazon EKS

As a managed service, Amazon EKS is protected by the AWS global network security procedures that are described in the [Amazon Web Services: Overview of security processes](#) paper.

You use AWS published API calls to access Amazon EKS through the network. Clients must support Transport Layer Security (TLS) 1.2 or later. Clients must also support cipher suites with perfect forward secrecy (PFS) such as Ephemeral Diffie-Hellman (DHE) or Elliptic Curve Ephemeral Diffie-Hellman (ECDHE). Most modern systems such as Java 7 and later support these modes.

Additionally, requests must be signed by using an access key ID and a secret access key that is associated with an IAM principal. Or you can use the [AWS Security Token Service](#) (AWS STS) to generate temporary security credentials to sign requests.

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

For more information about VPC considerations, see [Cluster VPC and subnet considerations \(p. 269\)](#).

If you create your VPC and node groups with the AWS CloudFormation templates provided in the [Getting started with Amazon EKS \(p. 4\)](#) walkthrough, then your control plane and node security groups are configured with our recommended settings.

For more information about security group considerations, see [Amazon EKS security group considerations \(p. 272\)](#).

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.

For more information about modifying cluster endpoint access, see [Modifying cluster endpoint access \(p. 48\)](#).

You can implement network policies with tools such as [Project Calico \(p. 350\)](#). Project Calico is a third party open source project. For more information, see the [Project Calico documentation](#).

## Configuration and vulnerability analysis in Amazon EKS

Security is a critical consideration for configuring and maintaining Kubernetes clusters and applications. The [Center for Internet Security \(CIS\) Kubernetes Benchmark](#) provides guidance for Amazon EKS node security configurations. The benchmark:

- Is applicable to Amazon EC2 nodes (both managed and self-managed) where you are responsible for security configurations of Kubernetes components.
- Provides a standard, community-approved way to ensure that you have configured your Kubernetes cluster and nodes securely when using Amazon EKS.

- Consists of four sections; control plane logging configuration, node security configurations, policies, and managed services.
- Supports all of the Kubernetes versions currently available in Amazon EKS and can be run using [kube-bench](#), a standard open source tool for checking configuration using the CIS benchmark on Kubernetes clusters.

To learn more, see [Introducing The CIS Amazon EKS Benchmark](#).

Amazon EKS platform versions represent the capabilities of the cluster control plane, including which Kubernetes API server flags are enabled and the current Kubernetes patch version. New clusters are deployed with the latest platform version. For details, see [Amazon EKS platform versions \(p. 76\)](#).

You can [update an Amazon EKS cluster \(p. 34\)](#) to newer Kubernetes versions. As new Kubernetes versions become available in Amazon EKS, we recommend that you proactively update your clusters to use the latest available version. For more information about Kubernetes versions in EKS, see [Amazon EKS Kubernetes versions \(p. 67\)](#).

Track security or privacy events for Amazon Linux 2 at the [Amazon Linux Security Center](#) or subscribe to the associated [RSS feed](#). Security and privacy events include an overview of the issue affected, packages, and instructions for updating your instances to correct the issue.

You can use [Amazon Inspector](#) to check for unintended network accessibility of your nodes and for vulnerabilities on those Amazon EC2 instances.

## Security best practices for Amazon EKS

Amazon EKS security best practices are maintained on Github: <https://aws.github.io/aws-eks-best-practices/security/docs/>

## Pod security policy

The Kubernetes pod security policy admission controller validates pod creation and update requests against a set of rules. By default, Amazon EKS clusters ship with a fully permissive security policy with no restrictions. For more information, see [Pod Security Policies](#) in the Kubernetes documentation.

### Note

As of Kubernetes v1.21, this feature is deprecated. PodSecurityPolicy will be functional for several more releases, following Kubernetes deprecation guidelines. To learn more, read [PodSecurityPolicy Deprecation: Past, Present, and Future](#) and the [AWS blog](#).

## Amazon EKS default pod security policy

Amazon EKS clusters with Kubernetes version 1.13 and higher have a default pod security policy named `eks.privileged`. This policy has no restriction on what kind of pod can be accepted into the system, which is equivalent to running Kubernetes with the `PodSecurityPolicy` controller disabled.

### Note

This policy was created to maintain backwards compatibility with clusters that did not have the `PodSecurityPolicy` controller enabled. You can create more restrictive policies for your cluster and for individual namespaces and service accounts and then delete the default policy to enable the more restrictive policies.

You can view the default policy with the following command.

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

Output:

| NAME                           | PRIV | CAPS | SELINUX  | RUNASUSER | FSGROUP  | SUPGROUP | READONLYROOTFS |
|--------------------------------|------|------|----------|-----------|----------|----------|----------------|
| VOLUMES<br>eks.privileged<br>* | true | *    | RunAsAny | RunAsAny  | RunAsAny | RunAsAny | false          |

For more details, you can describe the policy with the following command.

```
kubectl describe psp eks.privileged
```

Output:

```
Name: eks.privileged

Settings:
 Allow Privileged: true
 Allow Privilege Escalation: 0xc0004ce5f8
 Default Add Capabilities: <none>
 Required Drop Capabilities: <none>
 Allowed Capabilities: *
 Allowed Volume Types: *
 Allow Host Network: true
 Allow Host Ports: 0-65535
 Allow Host PID: true
 Allow Host IPC: true
 Read Only Root Filesystem: false
 SELinux Context Strategy: RunAsAny
 User: <none>
 Role: <none>
 Type: <none>
 Level: <none>
 Run As User Strategy: RunAsAny
 Ranges: <none>
 FSGroup Strategy: RunAsAny
 Ranges: <none>
 Supplemental Groups Strategy: RunAsAny
 Ranges: <none>
```

You can view the full YAML file for the `eks.privileged` pod security policy, its cluster role, and cluster role binding in [Install or restore the default pod security policy \(p. 503\)](#).

## Delete the default Amazon EKS pod security policy

If you create more restrictive policies for your pods, then after doing so, you can delete the default Amazon EKS `eks.privileged` pod security policy to enable your custom policies.

### Important

If you are using version 1.7.0 or later of the CNI plugin and you assign a custom pod security policy to the `aws-node` Kubernetes service account used for the `aws-node` pods deployed by the Daemonset, then the policy must have `NET_ADMIN` in its `allowedCapabilities` section along with `hostNetwork: true` and `privileged: true` in the policy's spec.

### To delete the default pod security policy

1. Create a file named `privileged-podsecuritypolicy.yaml` with the contents in the example file in [Install or restore the default pod security policy \(p. 503\)](#).

2. Delete the YAML with the following command. This deletes the default pod security policy, the ClusterRole, and the ClusterRoleBinding associated with it.

```
kubectl delete -f privileged-podsecuritypolicy.yaml
```

## Install or restore the default pod security policy

If you are upgrading from an earlier version of Kubernetes, or have modified or deleted the default Amazon EKS eks.privileged pod security policy, you can restore it with the following steps.

### To install or restore the default pod security policy

1. Create a file called *privileged-podsecuritypolicy.yaml* with the following contents.

```
apiVersion: policy/v1beta1
kind: PodSecurityPolicy
metadata:
 name: eks.privileged
 annotations:
 kubernetes.io/description: 'privileged allows full unrestricted access to
 pod features, as if the PodSecurityPolicy controller was not enabled.'
 seccomp.security.alpha.kubernetes.io/allowedProfileNames: '*'
 labels:
 kubernetes.io/cluster-service: "true"
 eks.amazonaws.com/component: pod-security-policy
spec:
 privileged: true
 allowPrivilegeEscalation: true
 allowedCapabilities:
 - '*'
 volumes:
 - '*'
 hostNetwork: true
 hostPorts:
 - min: 0
 max: 65535
 hostIPC: true
 hostPID: true
 runAsUser:
 rule: 'RunAsAny'
 seLinux:
 rule: 'RunAsAny'
 supplementalGroups:
 rule: 'RunAsAny'
 fsGroup:
 rule: 'RunAsAny'
 readOnlyRootFilesystem: false

apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
 name: eks:podsecuritypolicy:privileged
 labels:
 kubernetes.io/cluster-service: "true"
 eks.amazonaws.com/component: pod-security-policy
rules:
- apiGroups:
 - policy
 resourceNames:
 - eks.privileged
```

```

resources:
- podsecuritypolicies
verbs:
- use

apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
 name: eks:podsecuritypolicy:authenticated
 annotations:
 kubernetes.io/description: 'Allow all authenticated users to create privileged pods.'
 labels:
 kubernetes.io/cluster-service: "true"
 eks.amazonaws.com/component: pod-security-policy
roleRef:
 apiGroup: rbac.authorization.k8s.io
 kind: ClusterRole
 name: eks:podsecuritypolicy:privileged
subjects:
- kind: Group
 apiGroup: rbac.authorization.k8s.io
 name: system:authenticated

```

2. Apply the YAML with the following command.

```
kubectl apply -f privileged-podsecuritypolicy.yaml
```

## Using AWS Secrets Manager secrets with Kubernetes

To show secrets from Secrets Manager and parameters from Parameter Store as files mounted in [Amazon EKS](#) pods, you can use the AWS Secrets and Configuration Provider (ASCP) for the [Kubernetes Secrets Store CSI Driver](#). The ASCP works with Amazon Elastic Kubernetes Service (Amazon EKS) version 1.17 or later.

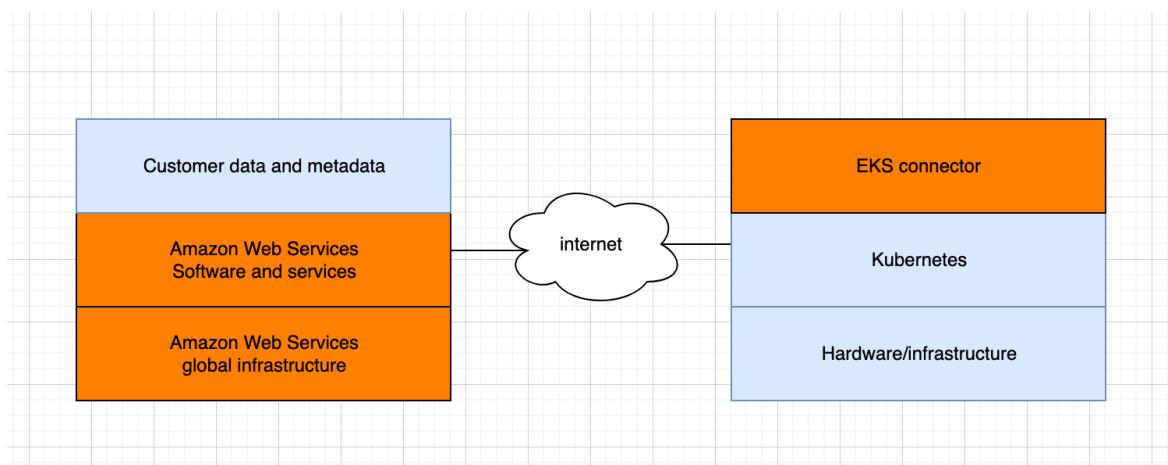
With the ASCP, you can store and manage your secrets in Secrets Manager and then retrieve them through your workloads running on Amazon EKS. You can use IAM roles and policies to limit access to your secrets to specific Kubernetes pods in a cluster. The ASCP retrieves the pod identity and exchanges the identity for an IAM role. ASCP assumes the IAM role of the pod, and then it can retrieve secrets from Secrets Manager that are authorized for that role.

If you use Secrets Manager automatic rotation for your secrets, you can also use the Secrets Store CSI Driver rotation reconciler feature to ensure you are retrieving the latest secret from Secrets Manager.

For more information, see [Using Secrets Manager secrets in Amazon EKS](#) in the AWS Secrets Manager User Guide.

## Amazon EKS Connector considerations

The Amazon EKS Connector is an open source component that runs on your Kubernetes cluster. This cluster can be located outside of the AWS environment. This creates additional considerations for security responsibilities. This configuration can be illustrated by the following diagram. Orange represents AWS responsibilities, and blue represents customer responsibilities:



This topic describes the differences in the responsibility model if the connected cluster is outside of AWS.

## AWS responsibilities

- Maintaining, building, and delivering Amazon EKS Connector, which is an [open source component](#) that runs on a customer's Kubernetes cluster and communicates with AWS.
- Maintaining transport and application layer communication security between the connected Kubernetes cluster and AWS services.

## Customer responsibilities

- Kubernetes cluster specific security, specifically along the following lines:
  - Kubernetes Secrets must be properly encrypted and protected.
  - Lock down access to the `eks-connector` namespace.
- Configuring role-based access control (RBAC) permissions to manage user access from AWS. For instructions, see [Granting access to a user to view a cluster](#).
- Installing and upgrading Amazon EKS Connector.
- Maintaining the hardware, software, and infrastructure that supports the connected Kubernetes cluster.
- Securing their AWS accounts (for example, through using [secure root user credentials](#)).

# View Kubernetes resources

You can view the Kubernetes resources deployed to your cluster with the AWS Management Console. You can't view Kubernetes resources with the AWS CLI or [eksctl \(p. 10\)](#). To view Kubernetes resources using a command-line tool, use [kubectl \(p. 4\)](#).

## Prerequisite

To view the **Overview** and **Resources** tabs in the AWS Management Console, the user that you're signed into the AWS Management Console as, or the role that you switch to once you're signed in, must have specific IAM and Kubernetes permissions. For more information, see [Required permissions \(p. 506\)](#).

## To view Kubernetes resources with the AWS Management Console

1. Open the Amazon EKS console at <https://console.aws.amazon.com/eks/home#/clusters>.
2. In the **Clusters** list, select the cluster that contains the Kubernetes resources that you want to view.
3. Select the **Resources** tab.
4. Select a **Resource type** group that you want to view resources for, such as **Workloads**. You see a list of resource types in that group.
5. Select a resource type, such as **Deployments**, in the **Workloads** group. You see a description of the resource type, a link to the Kubernetes documentation for more information about the resource type, and a list of resources of that type that are deployed on your cluster. If the list is empty, then there are no resources of that type deployed to your cluster.
6. Select a resource to view more information about it. Try the following examples:
  - Select the **Workloads** group, select the **Deployments** resource type, and then select the **coredns** resource. When you select a resource, you are in **Structured view**, by default. For some resource types, you see a **Pods** section in **Structured view**. This section lists the pods managed by the workload. You can select any pod listed to view information about the pod. Not all resource types display information in **Structured View**. If you select **Raw view** in the top right corner of the page for the resource, you see the complete JSON response from the Kubernetes API for the resource.
  - Select the **Cluster** group and then select the **Nodes** resource type. You see a list of all nodes in your cluster. The nodes can be any [Amazon EKS node type \(p. 105\)](#). This is the same list that you see in the **Nodes** section when you select the **Overview** tab for your cluster. Select a node resource from the list. In **Structured view**, you also see a **Pods** section. This section shows you all pods running on the node.

## Required permissions

To view the **Overview** and **Resources** tabs in the AWS Management Console, the user that you're signed into the AWS Management Console as, or the role that you switch to once you're signed in, must have specific minimum IAM and Kubernetes permissions. Complete the following steps to assign the required permissions to your users and roles.

1. Make sure that the `eks :AccessKubernetesApi`, and other necessary IAM permissions to view Kubernetes resources, are assigned to either the user that you sign into the AWS Management Console with, or the role that you switch to once you've signed in to the console. For more information about how to edit permissions for a user, see [Changing permissions for a user \(console\)](#) in the IAM User Guide. For more information about how to edit permissions for a role, see [Modifying a role permissions policy \(console\)](#) in the IAM User Guide.

The following example policy includes the necessary permissions for a user or role to view Kubernetes resources for all clusters in your account. Replace **111122223333** with your account ID.

```
{
 "Version": "2012-10-17",
 "Statement": [
 {
 "Effect": "Allow",
 "Action": [
 "eks>ListFargateProfiles",
 "eks>DescribeNodegroup",
 "eks>ListNodegroups",
 "eks>ListUpdates",
 "eks>AccessKubernetesApi",
 "eks>ListAddons",
 "eks>DescribeCluster",
 "eks>DescribeAddonVersions",
 "eks>ListClusters",
 "eks>ListIdentityProviderConfigs",
 "iam>ListRoles"
],
 "Resource": "*"
 },
 {
 "Effect": "Allow",
 "Action": "ssm:GetParameter",
 "Resource": "arn:aws:ssm:*:111122223333:parameter/*"
 }
]
}
```

To view nodes in connected clusters, the [Amazon EKS connector IAM role \(p. 480\)](#) should be able to impersonate the IAM user or role in the cluster. This allows the [Amazon EKS Connector \(p. 563\)](#) to map the IAM user or role to a Kubernetes user.

2. Create a Kubernetes `rolebinding` or `clusterrolebinding` that is bound to a Kubernetes `role` or `clusterrole` that has the necessary permissions to view the Kubernetes resources. To learn more about Kubernetes roles and role bindings, see [Using RBAC Authorization](#) in the Kubernetes documentation. You can apply one of the following manifests to your cluster that create a `role` and `rolebinding` or a `clusterrole` and `clusterrolebinding` with the necessary Kubernetes permissions:

- **View Kubernetes resources in all namespaces** – The group name in the file is `eks-console-dashboard-full-access-group`. Apply the manifest to your cluster with the following command:

```
kubectl apply -f https://s3.us-west-2.amazonaws.com/amazon-eks/docs/eks-console-full-access.yaml
```

- **View Kubernetes resources in a specific namespace** – The namespace in this file is `default`. The group name in the file is `eks-console-dashboard-restricted-access-group`. Apply the manifest to your cluster with the following command:

```
kubectl apply -f https://s3.us-west-2.amazonaws.com/amazon-eks/docs/eks-console-restricted-access.yaml
```

If you need to change the Kubernetes group name, namespace, permissions, or any other configuration in the file, then download the file and edit it before applying it to your cluster:

1. Download the file with one of the following commands:

```
curl -o eks-console-full-access.yaml https://s3.us-west-2.amazonaws.com/amazon-eks/docs/eks-console-full-access.yaml
```

```
curl -o eks-console-restricted-access.yaml https://s3.us-west-2.amazonaws.com/amazon-eks/docs/eks-console-restricted-access.yaml
```

2. Edit the file as necessary.

3. Apply the manifest to your cluster with one of the following commands:

```
kubectl apply -f eks-console-full-access.yaml
```

```
kubectl apply -f eks-console-restricted-access.yaml
```

3. Map the IAM user or role to the Kubernetes user or group in the `aws-auth` ConfigMap. You can use a tool such as `eksctl` to update the ConfigMap or you can update it manually by editing it.

**Important**

We recommend using `eksctl`, or another tool, to edit the ConfigMap. For information about other tools you can use, see [Use tools to make changes to the `aws-authConfigMap`](#) in the Amazon EKS best practices guides. An improperly formatted `aws-auth` ConfigMap can cause you to lose access to your cluster.

`eksctl`

**Prerequisite**

Version 0.97.0 or later of the `eksctl` command line tool installed on your computer or AWS CloudShell. To install or update `eksctl`, see [Installing `eksctl` \(p. 10\)](#).

1. View the current mappings in the ConfigMap. Replace `my-cluster` with the name of your cluster. Replace `region-code` with the AWS Region that your cluster is in.

```
eksctl get iamidentitymapping --cluster my-cluster --region=region-code
```

Example output:

| ARN                                                                                          | USERNAME                          | GROUPS |
|----------------------------------------------------------------------------------------------|-----------------------------------|--------|
| ACCOUNT                                                                                      |                                   |        |
| arn:aws:iam::111122223333:role/eksctl-my-cluster-my-nodegroup-NodeInstanceRole-1XLS7754U3ZPA | system:node:{{EC2PrivateDNSName}} |        |
|                                                                                              | system:bootstrappers,system:nodes |        |

2. Add a mapping for a role. This example assume that you attached the IAM permissions in the first step to a role named `my-console-viewer-role`. Replace `111122223333` with your account ID.

```
eksctl create iamidentitymapping \
--cluster my-cluster \
--region=region-code \
--arn arn:aws:iam::111122223333:role/my-console-viewer-role \
--group eks-console-dashboard-full-access-group \
--no-duplicate-arns
```

**Important**

The role ARN can't include a path such as `role/my-team/developers/my-role`.  
The format of the ARN must be `arn:aws:iam::111122223333:role/my-role`.  
In this example, `my-team/developers/` needs to be removed.

Example output:

```
...
2022-05-09 14:51:20 [#] adding identity "arn:aws:iam::111122223333:role/my-
console-viewer-role" to auth ConfigMap
```

3. Add a mapping for a user. This example assume that you attached the IAM permissions in the first step to a user named `my-user`. Replace `111122223333` with your account ID.

```
eksctl create iamidentitymapping \
--cluster my-cluster \
--region=region-code \
--arn arn:aws:iam::111122223333:user/my-user \
--group eks-console-dashboard-restricted-access-group \
--no-duplicate-arns
```

Example output:

```
...
2022-05-09 14:53:48 [#] adding identity "arn:aws:iam::111122223333:user/my-user"
to auth ConfigMap
```

4. View the mappings in the ConfigMap again.

```
eksctl get iamidentitymapping --cluster my-cluster --region=region-code
```

Example output:

| ARN                                                                                           | USERNAME                          | ACCOUNT | GROUPS                  |
|-----------------------------------------------------------------------------------------------|-----------------------------------|---------|-------------------------|
| arn:aws:iam::111122223333:role/eksctl-my-cluster-my-nodegroup- NodeInstanceRole-1XLS7754U3ZPA | system:node:{EC2PrivateDNSName}   |         |                         |
| arn:aws:iam::111122223333:role/my-console-viewer-role                                         | system:bootstrappers,system:nodes |         | eks-console-dashboard-  |
| arn:aws:iam::111122223333:user/my-user                                                        |                                   |         | restricted-access-group |

### Edit ConfigMap manually

For more information about adding users or roles to the `aws-auth` ConfigMap, see [Add IAM users, roles, or AWS accounts to the ConfigMap \(p. 403\)](#).

1. Open the ConfigMap for editing.

```
kubectl edit -n kube-system configmap/aws-auth
```

2. Add the mappings to the aws-auth ConfigMap, but don't replace any of the existing mappings. The following example adds mappings between IAM users and roles with permissions added in the first step and the Kubernetes groups created in the previous step:

- The `my-console-viewer-role` role and the `eks-console-dashboard-full-access-group`.
- The `my-user` user and the `eks-console-dashboard-restricted-access-group`.

These examples assume that you attached the IAM permissions in the first step to a role named `my-console-viewer-role` and a user named `my-user`. Replace `111122223333` with your account ID.

```
apiVersion: v1
data:
mapRoles: |
 - groups:
 - eks-console-dashboard-full-access-group
 rolearn: arn:aws:iam::111122223333:role/my-console-viewer-role
 username: my-console-viewer-role
mapUsers: |
 - groups:
 - eks-console-dashboard-restricted-access-group
 userarn: arn:aws:iam::111122223333:user/my-user
 username: my-user
```

**Important**

The role ARN can't include a path such as `role/my-team/developers/my-console-viewer-role`. The format of the ARN must be `arn:aws:iam::111122223333:role/my-console-viewer-role`. In this example, `my-team/developers/` needs to be removed.

3. Save the file and exit the editor.

# Observability in Amazon EKS

You can observe your data in Amazon EKS using many available monitoring or logging tools. With a Fargate log router configured, your Amazon EKS log data can be streamed to AWS services or to partner tools for data analysis. These AWS services are available in the AWS Management Console. You can monitor your Amazon EKS resources using Amazon CloudWatch. This topic describes the available functions and formats in Amazon EKS. There are many services available in the AWS Management Console that provide data for troubleshooting your Amazon EKS issues. For an overview of these services, see [AWS Management Console documentation](#).

You can view cluster health and details by selecting your cluster's name after selecting **Clusters** in the left navigation pane of the Amazon EKS console. To view details about any existing Kubernetes resources that are deployed to your cluster, see [View Kubernetes resources \(p. 506\)](#).

Monitoring is an important part of maintaining the reliability, availability, and performance of Amazon EKS and your AWS solutions. We recommend that you collect monitoring data from all of the parts of your AWS solution. That way, you can more easily debug a multi-point failure if one occurs. Before you start monitoring Amazon EKS, make sure that a monitoring plan that addresses the following questions.

- What are your goals? Do you need real-time notifications if your clusters scale dramatically?
- What resources need to be observed?
- How frequently do you need to observe these resources? Does your company want to respond quickly to risks?
- What tools do you intend to use? If you already run Fargate as part of your launch, then you can use the built-in [log router](#).
- Who do you intend to perform the monitoring tasks?
- Whom do you want notifications to be sent to when something goes wrong?

## Logging and monitoring in Amazon EKS

Amazon EKS provides built-in tools for logging and monitoring. You can choose your logging level, which starts at the control plane. Control plane logging records all API calls to your clusters, audit information capturing what users performed what actions to your clusters, as well as role-based logs.

Amazon EKS control plane logging provides audit and diagnostic logs directly from the Amazon EKS control plane to CloudWatch Logs in your account. These logs make it easy for you to secure and run your clusters. You can select the exact log types you need, and logs are sent as log streams to a group for each Amazon EKS cluster in CloudWatch. For more information, see [Amazon EKS Control Plane Logging \(p. 64\)](#).

### Note

When you check the Amazon EKS authenticator logs in Amazon CloudWatch, the entries are displayed that contain text similar to the following example text.

```
level=info msg="mapping IAM role" groups="[]"
role="arn:aws:iam::<111122223333>:role/<XXXXXXXXXXXXXXXXXX>-NodeManagerRole-<XXXXXXX>" username="eks:node-manager"
```

Entries that contain this text are expected. The `username` is an Amazon EKS internal service role that performs specific operations for managed node groups and Fargate. For low-level, customizable logging, then [Kubernetes logging](#) is available.

Amazon EKS is integrated with AWS CloudTrail, a service that provides a record of actions taken by a user, role, or an AWS service in Amazon EKS. CloudTrail captures all API calls for Amazon EKS as events. The calls captured include calls from the Amazon EKS console and code calls to the Amazon EKS API operations. For more information, see [Logging Amazon EKS API calls with AWS CloudTrail \(p. 514\)](#).

The Kubernetes API server exposes a number of metrics that are useful for monitoring and analysis. For more information, see [Control plane metrics with Prometheus \(p. 427\)](#).

To configure Fluent Bit for custom Amazon CloudWatch logs, see [Set up Fluent Bit as a DaemonSet to send logs to Amazon CloudWatch Logs](#) in the Amazon CloudWatch user guide.

## Amazon EKS monitoring and logging tools

Amazon Web Services provides various tools that you can use to monitor Amazon EKS. You can configure some tools to set up automatic monitoring, but some require manual calls. We recommend that you automate monitoring tasks as much as your environment and existing toolset allows.

### Logging

| Areas          | Tool                                                 | Logs                                                                                                                                                     | Setup                                                                                                                                    |
|----------------|------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------|
| Applications   | <a href="#">Amazon CloudWatch Container Insights</a> | It collects, aggregates, and summarizes metrics and logs from your containerized applications and microservices.                                         | <a href="#">Setup procedure</a><br>To set up Amazon CloudWatch Container Insights logging for ADOT, use <a href="#">this procedure</a> . |
| Containers     | <a href="#">AWS Fargate</a>                          | It provides compute capacity for containers, including resource usage.                                                                                   | <a href="#">Setup procedure</a>                                                                                                          |
| Control plane  | <a href="#">Prometheus</a>                           | CloudWatch Logs ingestion, archive storage, and data scanning rates apply to enabled control plane logs.                                                 | <a href="#">Setup procedure</a>                                                                                                          |
| Control plane  | <a href="#">AWS CloudTrail</a>                       | It logs API calls by a user, role, or service.                                                                                                           | <a href="#">Setup procedure</a>                                                                                                          |
| Multiple areas | Fargate log router                                   | It streams logs to AWS services or partner tools. Uses <a href="#">AWS for Fluent Bit</a> . Logs can be streamed to other AWS services or partner tools. | <a href="#">Setup procedure</a>                                                                                                          |

## Monitoring

| Areas        | Tool                                                 | Logs                                                                                                                                                                                                                       | Setup                                                          |
|--------------|------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----------------------------------------------------------------|
| Applications | <a href="#">Amazon CloudWatch Container Insights</a> | Amazon CloudWatch Container Insights collects, aggregates, and summarizes metrics and logs from your containerized applications and microservices.                                                                         | <a href="#">Setup procedure for the AWS Management Console</a> |
| Applications | <a href="#">AWS Distro for OpenTelemetry (ADOT)</a>  | <p>It collects and sends correlated metrics, trace data, and metadata to AWS monitoring services or partners.</p> <p>It can be set up through Amazon CloudWatch Container Insights.</p>                                    | <a href="#">Setup procedure</a>                                |
| Applications | <a href="#">Amazon DevOps Guru</a>                   | It detects node-level operational performance and availability.                                                                                                                                                            | <a href="#">Enable Amazon CloudWatch Container Insights</a>    |
| Applications | <a href="#">AWS X-Ray</a>                            | <p>It receives trace data about your application.</p> <p>This trace data includes ingoing and outgoing requests and metadata about the requests. For Amazon EKS, the implementation requires the OpenTelemetry add-on.</p> | <a href="#">Setup procedure</a>                                |
| Cluster      | <a href="#">Amazon CloudWatch</a>                    | It logs cluster resources.                                                                                                                                                                                                 | n/a                                                            |
| Containers   | <a href="#">AWS Fargate</a>                          | It provides compute capacity for containers, including resource usage.                                                                                                                                                     | <a href="#">Setup procedure</a>                                |

# Logging Amazon EKS API calls with AWS CloudTrail

Amazon EKS is integrated with AWS CloudTrail, a service that provides a record of actions that's taken by a user, role, or an AWS service in Amazon EKS. CloudTrail captures all API calls for Amazon EKS as events. This includes calls from the Amazon EKS console and from code calls to the Amazon EKS API operations.

If you create a trail, you can enable continuous delivery of CloudTrail events to an Amazon S3 bucket. This includes events for Amazon EKS. If you don't configure a trail, you can still view the most recent events in the CloudTrail console in **Event history**. Using the information that CloudTrail collects, you can determine several details about a request. For example, you can determine when the request was made to Amazon EKS, the IP address where the request was made from, and who made the request.

To learn more about CloudTrail, see the [AWS CloudTrail User Guide](#).

## Amazon EKS information in CloudTrail

When you create your AWS account, CloudTrail is also enabled on your AWS account. When any activity occurs in Amazon EKS, that activity is recorded in a CloudTrail event along with other AWS service events in **Event history**. You can view, search, and download recent events in your AWS account. For more information, see [Viewing events with CloudTrail event history](#).

For an ongoing record of events in your AWS account, including events for Amazon EKS, create a trail. A *trail* enables CloudTrail to deliver log files to an Amazon S3 bucket. By default, when you create a trail in the console, the trail applies to all AWS Regions. The trail logs events from all AWS Regions in the AWS partition and delivers the log files to the Amazon S3 bucket that you specify. Additionally, you can configure other AWS services to further analyze and act upon the event data that's collected in CloudTrail logs. For more information, see the following resources.

- [Overview for creating a trail](#)
- [CloudTrail supported services and integrations](#)
- [Configuring Amazon SNS notifications for CloudTrail](#)
- [Receiving CloudTrail log files from multiple regions](#) and [Receiving CloudTrail log files from multiple accounts](#)

All Amazon EKS actions are logged by CloudTrail and are documented in the [Amazon EKS API Reference](#). For example, calls to the `CreateCluster`, `ListClusters` and `DeleteCluster` sections generate entries in the CloudTrail log files.

Every event or log entry contains information about who generated the request. You can use the identity information to determine the following information.

- Whether the request was made with root or AWS Identity and Access Management (IAM) user credentials.
- Whether the request was made with temporary security credentials for a role or federated user.
- Whether the request was made by another AWS service.

For more information, see the [CloudTrail userIdentity element](#).

## Understanding Amazon EKS log file entries

A trail is a configuration that enables delivery of events as log files to an Amazon S3 bucket that you specify. CloudTrail log files contain one or more log entries. An event represents a single request from

any source and includes information about the requested action. This include information such as the date and time of the action and the request parameters that were used. CloudTrail log files aren't an ordered stack trace of the public API calls, so they don't appear in any specific order.

The following example shows a CloudTrail log entry that demonstrates the [CreateCluster](#) action.

```
{
 "eventVersion": "1.05",
 "userIdentity": {
 "type": "IAMUser",
 "principalId": "AKIAIOSFODNN7EXAMPLE",
 "arn": "arn:aws:iam::111122223333:user/username",
 "accountId": "111122223333",
 "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
 "userName": "username"
 },
 "eventTime": "2018-05-28T19:16:43Z",
 "eventSource": "eks.amazonaws.com",
 "eventName": "CreateCluster",
 "awsRegion": "region-code",
 "sourceIPAddress": "205.251.233.178",
 "userAgent": "PostmanRuntime/6.4.0",
 "requestParameters": {
 "resourcesVpcConfig": {
 "subnetIds": [
 "subnet-a670c2df",
 "subnet-4f8c5004"
]
 },
 "roleArn": "arn:aws:iam::111122223333:role/AWSServiceRoleForAmazonEKS-CAC1G1VH3ZKZ",
 "clusterName": "test"
 },
 "responseElements": {
 "cluster": {
 "clusterName": "test",
 "status": "CREATING",
 "createdAt": 1527535003.208,
 "certificateAuthority": {},
 "arn": "arn:aws:eks:region-code:111122223333:cluster/test",
 "roleArn": "arn:aws:iam::111122223333:role/AWSServiceRoleForAmazonEKS-CAC1G1VH3ZKZ",
 "version": "1.10",
 "resourcesVpcConfig": {
 "securityGroupIds": [],
 "vpcId": "vpc-21277358",
 "subnetIds": [
 "subnet-a670c2df",
 "subnet-4f8c5004"
]
 }
 }
 },
 "requestID": "a7a0735d-62ab-11e8-9f79-81ce5b2b7d37",
 "eventID": "eab22523-174a-499c-9dd6-91e7be3ff8e3",
 "readOnly": false,
 "eventType": "AwsApiCall",
 "recipientAccountId": "111122223333"
}
```

## Log Entries for Amazon EKS Service Linked Roles

The Amazon EKS service linked roles make API calls to AWS resources. CloudTrail log entries with `username: AWSServiceRoleForAmazonEKS` and `username: AWSServiceRoleForAmazonEKSNodegroup` appears for calls made by the Amazon EKS service linked

roles. For more information about Amazon EKS and service linked roles, see [Using service-linked roles for Amazon EKS \(p. 465\)](#).

The following example shows a CloudTrail log entry that demonstrates a `DeleteInstanceProfile` action that's made by the `AWSServiceRoleForAmazonEKSNodegroup` service linked role, noted in the `sessionContext`.

```
{
 "eventVersion": "1.05",
 "userIdentity": {
 "type": "AssumedRole",
 "principalId": "AROA3WHGPEZ7SJ2CW55C5:EKS",
 "arn": "arn:aws:sts::111122223333:assumed-role/AWSServiceRoleForAmazonEKSNodegroup/EKS",
 "accountId": "111122223333",
 "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
 "sessionContext": {
 "sessionIssuer": {
 "type": "Role",
 "principalId": "AROA3WHGPEZ7SJ2CW55C5",
 "arn": "arn:aws:iam::111122223333:role/aws-service-role/eks-nodegroup.amazonaws.com/AWSServiceRoleForAmazonEKSNodegroup",
 "accountId": "111122223333",
 "userName": "AWSServiceRoleForAmazonEKSNodegroup"
 },
 "webIdFederationData": {},
 "attributes": {
 "mfaAuthenticated": "false",
 "creationDate": "2020-02-26T00:56:33Z"
 }
 },
 "invokedBy": "eks-nodegroup.amazonaws.com"
 },
 "eventTime": "2020-02-26T00:56:34Z",
 "eventSource": "iam.amazonaws.com",
 "eventName": "DeleteInstanceProfile",
 "awsRegion": "region-code",
 "sourceIPAddress": "eks-nodegroup.amazonaws.com",
 "userAgent": "eks-nodegroup.amazonaws.com",
 "requestParameters": {
 "instanceProfileName": "eks-1111111-2222-3333-4444-abcdef123456"
 },
 "responseElements": null,
 "requestID": "1111111-2222-3333-4444-abcdef123456",
 "eventID": "1111111-2222-3333-4444-abcdef123456",
 "eventType": "AwsApiCall",
 "recipientAccountId": "111122223333"
}
```

## Amazon EKS add-on support for ADOT Operator

Amazon EKS supports using the AWS Management Console, AWS CLI and Amazon EKS API to install and manage the [AWS Distro for OpenTelemetry \(ADOT\) Operator](#). This enables a simplified experience for instrumenting your applications running on Amazon EKS to send metric and trace data to multiple monitoring service options like [Amazon CloudWatch](#), [Prometheus](#), and [X-Ray](#). This topic describes how to get started using ADOT for applications running on your Amazon EKS cluster.

- [Prerequisites \(p. 517\)](#)
- [Creating an IAM role \(p. 518\)](#)
- [Installing the add-on \(p. 519\)](#)

- [Deploying the ADOT Collector \(p. 521\)](#) for your preferred monitoring option
- [Deploying a sample application \(p. 535\)](#) to generate OTLP data

Before installing ADOT, ensure that you have read and understand the [considerations \(p. 517\)](#) and [prerequisites \(p. 517\)](#).

## ADOT considerations

Before you use ADOT, review the following considerations and prerequisites:

- The ADOT Operator doesn't yet support the arm64 architecture.
- Connected clusters cannot use this add-on.

## AWS Distro for OpenTelemetry (ADOT) prerequisites

Before installing the AWS Distro for OpenTelemetry (ADOT) add-on, you must meet the following prerequisites.

- Grant permissions to Amazon EKS add-ons to install ADOT:

```
kubectl apply -f https://amazon-eks.s3.amazonaws.com/docs/addons-otel-permissions.yaml
```

- Your Amazon EKS cluster must be using Kubernetes version 1.19 or higher. You can verify the version using the following command. To update your cluster, see [Updating a cluster \(p. 34\)](#).

```
kubectl version | grep "Server Version"
```

- Meet the [TLS certificate requirement \(p. 517\)](#) to ensure end-to-end encryption.

## TLS certificate requirement

The ADOT Operator uses [admission webhooks](#) to mutate and validate the Collector Custom Resource (CR) requests. In Kubernetes, the webhook requires a TLS certificate that the API server is configured to trust. There are multiple ways for you to generate the required TLS certificate. However, the default method is to install the [cert-manager](#) manually with a version of *less than 1.6.0*. The cert-manager generates a self-signed certificate.

### Important

The ADOT Operator is compatible with cert-manager versions of less than 1.6.0. Don't use version 1.6.0.

## Installing cert-manager

### Installing cert-manager

1. Install cert-manager using the following command. This creates the necessary cert-manager objects that allow end-to-end encryption. This must be done for each cluster that will have ADOT installed.

```
kubectl apply -f \
https://github.com/jetstack/cert-manager/releases/download/v1.5.0/cert-manager.yaml
```

2. Verify that cert-manager is ready using the following command.

```
kubectl get pod -w -n cert-manager
```

Example output:

```
NAME READY STATUS RESTARTS AGE
cert-manager-5597cff495-mnb2p 1/1 Running 0 12d
cert-manager-cainjector-bd5f9c764-8jp5g 1/1 Running 0 12d
cert-manager-webhook-5f57f59fbc-h9st8 1/1 Running 0 12d
```

## Create an IAM role

Each cluster where you [install AWS Distro for OpenTelemetry \(ADOT\) \(p. 519\)](#) must have this role to grant your AWS service account permissions. Follow these steps to create and associate your IAM role to your Amazon EKS service account using [IRSA \(p. 442\)](#):

1. [Create an IAM OIDC provider for your cluster \(p. 446\)](#).
2. Create your service account and IAM role. Note the following flags included in this command:
  - For the `--name` flag, specify the name of the service account you want to create. In this example, it is `adot-collector`.
  - For the `--namespace` flag, specify the namespace your service account will reside in; for this example you will use the `default` namespace.
  - For the `--cluster` flag, specify the name of your cluster.
  - Use the `--attach-policy-arn` parameter to specify the managed IAM policy for the integration you are using to the role. For example, if you are using ADOT Collector to send metric data to CloudWatch, you specify the `CloudWatchAgentServerPolicy` managed policy.
  - The `--override-existing-serviceaccounts` flag is for a service account already created in the cluster without an IAM role. You can exclude this if the service account does not have an IAM role.

Enter the following command considering the notes above:

```
eksctl create iamserviceaccount \
--name adot-collector \
--namespace default \
--cluster cluster_name \
--attach-policy-arn arn:aws:iam::aws:policy/AmazonPrometheusRemoteWriteAccess \
--attach-policy-arn arn:aws:iam::aws:policy/AWSXrayWriteOnlyAccess \
--attach-policy-arn arn:aws:iam::aws:policy/CloudWatchAgentServerPolicy \
--approve \
--override-existing-serviceaccounts
```

In the following collector configurations, the `serviceAccount: adot-collector` field to the configuration has been added to use IRSA.

## Manage the AWS Distro for OpenTelemetry Operator

The AWS Distro for OpenTelemetry (ADOT) Operator is available as an Amazon EKS add-on. After installing the ADOT Operator, you can configure the ADOT Collector to specify the deployment type and the service that will receive your application metric or trace data. This topic describes how to manage the ADOT add-on.

- [Install the AWS Distro for OpenTelemetry \(ADOT\) Operator \(p. 519\)](#)
- [Update the AWS Distro for OpenTelemetry \(ADOT\) Operator \(p. 520\)](#)
- [Remove the AWS Distro for OpenTelemetry \(ADOT\) Operator \(p. 521\)](#)

To install the ADOT Collector, see [Deploy the AWS Distro for OpenTelemetry Collector \(p. 521\)](#).

## Install the AWS Distro for OpenTelemetry (ADOT) Operator

Installing the ADOT add-on includes the ADOT Operator, which in turn deploys the ADOT Collector. The ADOT Operator is a custom controller which introduces a new object type called the `OpenTelemetryCollector` through [CustomResourceDefinition \(CRD\)](#). When the ADOT Operator detects the presence of the `OpenTelemetryCollector` resource, then it installs the ADOT Collector.

### Prerequisites

- You have met the [ADOT prerequisites \(p. 517\)](#).
- [kubectl is installed](#).
- Update your `kubeconfig` if necessary using the following command.

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

- [eksctl is installed](#).
- [AWS CLI version 2 is installed](#).
- An existing Amazon EKS cluster.
- An existing Amazon EKS service IAM role. If you don't have the role, you can follow [Create an IAM role \(p. 518\)](#) to create one.

### AWS Management Console

Install the ADOT Amazon EKS add-on to your Amazon EKS cluster using the following steps:

1. In the [Getting Started with the AWS Management Console](#), click the Amazon EKS cluster that you intend to install the add-on to enter the Cluster configuration screen.
2. Click **Add-ons**.
3. Click **Add new** and select **AWS Distro for OpenTelemetry** from the drop-down list.
4. The default version will be selected in the **Version** drop-down. Click **Override existing configuration for this add-on on the cluster** if a service account is already created in the cluster without an IAM Role.
5. Click **Add**.

### AWS CLI

1. Install the ADOT Amazon EKS add-on to your Amazon EKS cluster using the command:

```
aws eks create-addon --addon-name adot --cluster-name your-cluster
```

Example output:

```
{
 "addon": {
 "addonName": "adot",
 "clusterName": "adot",
 "status": "CREATING",
 "addonVersion": "v0.45.0-eksbuild.1",
 "health": {
 "issues": []
 }
 }
}
```

```
{
 "addonArn": "arn:aws:eks:region:111122223333:addon/adot/adot/c0bfbc67-ae88-d472-53e4-e88c79f7be7c",
 "createdAt": "2022-03-11T03:48:50.212000-08:00",
 "modifiedAt": "2022-03-11T03:48:50.234000-08 (tel:5023400008):00",
 "tags": {}
}
```

The status field value will be CREATING until complete.

2. Verify that ADOT is installed and running using the command:

```
aws eks describe-addon --addon-name adot --cluster-name your-cluster
```

Example output:

```
{
 "addon": {
 "addonName": "adot",
 "clusterName": "your-cluster",
 "status": "ACTIVE",
 "addonVersion": "v0.45.0-eksbuild.1",
 "health": {
 "issues": []
 },
 "addonArn": "arn:aws:eks:us-west-2:111122223333:addon/adot/adot/9ebe61e2-824d-14c2-408e-c491a7cc5fb8",
 "createdAt": "2021-10-27T15:00:41.536000-07:00",
 "modifiedAt": "2021-10-27T15:01:24.104000-07:00",
 "tags": {}
 }
}
```

The status value is now ACTIVE as creation is complete.

## Update the AWS Distro for OpenTelemetry (ADOT) Operator

Amazon EKS does not automatically update ADOT on your cluster. You must initiate the update and then Amazon EKS updates the Amazon EKS add-on for you.

### To update the ADOT Amazon EKS add-on using the AWS CLI

1. Check the current version of your ADOT add-on. Replace *my-cluster* with your cluster name.

```
aws eks describe-addon \
--cluster-name my-cluster \
--addon-name adot \
--query "addon.addonVersion" \
--output text
```

Example output:

```
1.7.0
```

2. Determine the ADOT versions are available that are supported by your cluster's version.

```
aws eks describe-addon-versions \
```

```
--addon-name adot \
--kubernetes-version 1.19 \
--query "addons[].addonVersions[].addonVersion, compatibilities[].defaultVersion"
\
--output text
```

Example output

```
1.8.0
True
1.7.0
False
```

The version with `True` underneath is the default version deployed with new clusters. Later versions can be returned and updated, if necessary.

- To update the ADOT version, use the following command and replace the `addon-version` value with the desired version.

```
aws eks update-addon \
--cluster-name my-cluster \
--addon-name adot \
--addon-version 1.8.0 \
--resolve-conflicts OVERWRITE
```

#### Note

The `--resolve-conflicts OVERWRITE` option will resolve any conflicts with Amazon EKS add-on settings by overwriting your settings. Ensure that your settings don't require management. Without this option, conflicts must be resolved manually. For more information, see [Amazon EKS add-on configuration \(p. 389\)](#).

## Remove the AWS Distro for OpenTelemetry (ADOT) Operator

- You must delete the ADOT Collector resource separately from the ADOT Collector. In this command, specify the YAML file that you used to deploy the ADOT Collector:

```
kubectl delete -f collector-config.yaml
```

- You can remove the ADOT Operator through either the AWS CLI or `eksctl`. If you remove the ADOT Operator, you must follow the [installation instructions \(p. 519\)](#) again to reinstall:

CLI

```
aws eks delete-addon --addon-name adot --cluster-name your-cluster
```

`eksctl`

```
eksctl delete addon --cluster your-cluster --name adot
```

## Deploy the AWS Distro for OpenTelemetry Collector

Once the [AWS Distro for OpenTelemetry \(ADOT\) Operator is installed \(p. 519\)](#) and running, you can deploy the ADOT Collector into your Amazon EKS cluster. The ADOT Collector can be deployed in one of four modes: Deployment, Daemonset, StatefulSet, and Sidecar and can be deployed for visibility by an

individual service. For more information on these deployment modes, see the ADOT Collector [installation instructions on Github](#).

You can deploy the ADOT Collector by applying a YAML file to your cluster. You should have already installed the [ADOT Operator \(p. 519\)](#). You can [Create an IAM role \(p. 518\)](#) with your Amazon EKS service account using [IRSA \(p. 442\)](#). By doing this, your service account can provide AWS permission to the containers you run in any pod that use that service account.

We provide ADOT Collector examples for the following services (in deployment modes):

- [Prometheus \(p. 522\)](#)
- [Amazon CloudWatch \(p. 528\)](#)
- [X-Ray \(p. 534\)](#)

## Deploy the AWS Distro for OpenTelemetry Collector for Amazon Managed Service for Prometheus in deployment mode

The following is an example YAML file with deployment as the mode value. This is the default mode and deploys the ADOT Collector similarly to a standalone application. This configuration receives OTLP metrics from the sample application and Amazon Managed Service for Prometheus metrics scraped from pods on the cluster. You can change the mode to Daemonset, StatefulSet, and Sidecar depending on your deployment strategy. For a daemonset deployment, see [the github page](#) or [click here](#) to download the daemonset example.

1. Copy the following text to a YAML file.

```

OpenTelemetry Collector configuration
Metrics pipeline with Prometheus Receiver and AWS Remote Write Exporter sending
metrics to Amazon Managed Prometheus

apiVersion: opentelemetry.io/v1alpha1
kind: OpenTelemetryCollector
metadata:
 name: my-collector-amp
spec:
 mode: deployment
 serviceAccount: adot-demo
 podAnnotations:
 prometheus.io/scrape: 'true'
 prometheus.io/port: '8888'
 config: |
 receivers:
 #
 # Scrape configuration for the Prometheus Receiver
 # This is the same configuration used when Prometheus is installed using
 # the community Helm chart
 #
 prometheus:
 config:
 global:
 scrape_interval: 15s
 scrape_timeout: 10s
 scrape_configs:
 - job_name: kubernetes-apiservers
 bearer_token_file: /var/run/secrets/kubernetes.io/serviceaccount/token
 kubernetes_sd_configs:
```

```

- role: endpoints
 relabel_configs:
 - action: keep
 regex: default;kubernetes;https
 source_labels:
 - __meta_kubernetes_namespace
 - __meta_kubernetes_service_name
 - __meta_kubernetes_endpoint_port_name
 scheme: https
 tls_config:
 ca_file: /var/run/secrets/kubernetes.io/serviceaccount/ca.crt
 insecure_skip_verify: true
- job_name: kubernetes-nodes
 bearer_token_file: /var/run/secrets/kubernetes.io/serviceaccount/token
 kubernetes_sd_configs:
 - role: node
 relabel_configs:
 - action: labelmap
 regex: __meta_kubernetes_node_label_(.+)
 - replacement: kubernetes.default.svc:443
 target_label: __address__
 - regex: (++)
 replacement: /api/v1/nodes/$$1/proxy/metrics
 source_labels:
 - __meta_kubernetes_node_name
 target_label: __metrics_path__
 scheme: https
 tls_config:
 ca_file: /var/run/secrets/kubernetes.io/serviceaccount/ca.crt
 insecure_skip_verify: true
- job_name: kubernetes-nodes-cadvisor
 bearer_token_file: /var/run/secrets/kubernetes.io/serviceaccount/token
 kubernetes_sd_configs:
 - role: node
 relabel_configs:
 - action: labelmap
 regex: __meta_kubernetes_node_label_(.+)
 - replacement: kubernetes.default.svc:443
 target_label: __address__
 - regex: (++)
 replacement: /api/v1/nodes/$$1/proxy/metrics/cadvisor
 source_labels:
 - __meta_kubernetes_node_name
 target_label: __metrics_path__
 scheme: https
 tls_config:
 ca_file: /var/run/secrets/kubernetes.io/serviceaccount/ca.crt
 insecure_skip_verify: true
- job_name: kubernetes-service-endpoints
 kubernetes_sd_configs:
 - role: endpoints
 relabel_configs:
 - action: keep
 regex: true
 source_labels:
 - __meta_kubernetes_service_annotation_prometheus_io_scrape
 - action: replace
 regex: (https?)
 source_labels:
 - __meta_kubernetes_service_annotation_prometheus_io_scheme
 target_label: __scheme__
 - action: replace
 regex: (++)
 source_labels:
 - __meta_kubernetes_service_annotation_prometheus_io_path
 target_label: __metrics_path__

```

```
- action: replace
 regex: ([^:]+)(?:\d+)?;(\d+)
 replacement: $$1.$$2
 source_labels:
 - __address__
 - __meta_kubernetes_service_annotation_prometheus_io_port
 target_label: __address__
- action: labelmap
 regex: __meta_kubernetes_service_annotation_prometheus_io_param_(.+)
 replacement: __param_$$1
- action: labelmap
 regex: __meta_kubernetes_service_label_(.+)
- action: replace
 source_labels:
 - __meta_kubernetes_namespace
 target_label: kubernetes_namespace
- action: replace
 source_labels:
 - __meta_kubernetes_service_name
 target_label: kubernetes_name
- action: replace
 source_labels:
 - __meta_kubernetes_pod_node_name
 target_label: kubernetes_node
- job_name: kubernetes-service-endpoints-slow
 kubernetes_sd_configs:
 - role: endpoints
 relabel_configs:
 - action: keep
 regex: true
 source_labels:
 - __meta_kubernetes_service_annotation_prometheus_io_scrape_slow
 - action: replace
 regex: (https?)
 source_labels:
 - __meta_kubernetes_service_annotation_prometheus_io_scheme
 target_label: __scheme__
 - action: replace
 regex: (++)
 source_labels:
 - __meta_kubernetes_service_annotation_prometheus_io_path
 target_label: __metrics_path__
 - action: replace
 regex: ([^:]+)(?:\d+)?;(\d+)
 replacement: $$1.$$2
 source_labels:
 - __address__
 - __meta_kubernetes_service_annotation_prometheus_io_port
 target_label: __address__
- action: labelmap
 regex: __meta_kubernetes_service_annotation_prometheus_io_param_(.+)
 replacement: __param_$$1
- action: labelmap
 regex: __meta_kubernetes_service_label_(.+)
- action: replace
 source_labels:
 - __meta_kubernetes_namespace
 target_label: kubernetes_namespace
- action: replace
 source_labels:
 - __meta_kubernetes_service_name
 target_label: kubernetes_name
- action: replace
 source_labels:
 - __meta_kubernetes_pod_node_name
 target_label: kubernetes_node
```

```
scrape_interval: 5m
scrape_timeout: 30s

- job_name: prometheus-pushgateway
 kubernetes_sd_configs:
 - role: service
 relabel_configs:
 - action: keep
 regex: pushgateway
 source_labels:
 - __meta_kubernetes_service_annotation_prometheus_io_probe
 - job_name: kubernetes-services
 kubernetes_sd_configs:
 - role: service
 metrics_path: /probe
 params:
 module:
 - http_2xx
 relabel_configs:
 - action: keep
 regex: true
 source_labels:
 - __meta_kubernetes_service_annotation_prometheus_io_probe
 - source_labels:
 - __address__
 target_label: __param_target
 - replacement: blackbox
 target_label: __address__
 - source_labels:
 - __param_target
 target_label: instance
 - action: labelmap
 regex: __meta_kubernetes_service_label_(.+)
 - source_labels:
 - __meta_kubernetes_namespace
 target_label: kubernetes_namespace
 - source_labels:
 - __meta_kubernetes_service_name
 target_label: kubernetes_name
 - job_name: kubernetes-pods
 kubernetes_sd_configs:
 - role: pod
 relabel_configs:
 - action: keep
 regex: true
 source_labels:
 - __meta_kubernetes_pod_annotation_prometheus_io_scrape
 - action: replace
 regex: (https?)
 source_labels:
 - __meta_kubernetes_pod_annotation_prometheus_io_scheme
 target_label: __scheme__
 - action: replace
 regex: (.)
 source_labels:
 - __meta_kubernetes_pod_annotation_prometheus_io_path
 target_label: __metrics_path__
 - action: replace
 regex: ([^:]+)(?::\d+)?;(\d+)
 replacement: $$1:$$2
 source_labels:
 - __address__
 - __meta_kubernetes_pod_annotation_prometheus_io_port
 target_label: __address__
 - action: labelmap
 regex: __meta_kubernetes_pod_annotation_prometheus_io_param_(.+)
```

```

 replacement: __param_$$1
 - action: labelmap
 regex: __meta_kubernetes_pod_label_(.+)
 - action: replace
 source_labels:
 - __meta_kubernetes_namespace
 target_label: kubernetes_namespace
 - action: replace
 source_labels:
 - __meta_kubernetes_pod_name
 target_label: kubernetes_pod_name
 - action: drop
 regex: Pending|Succeeded|Failed|Completed
 source_labels:
 - __meta_kubernetes_pod_phase
 - job_name: kubernetes-pods-slow
 scrape_interval: 5m
 scrape_timeout: 30s
 kubernetes_sd_configs:
 - role: pod
 relabel_configs:
 - action: keep
 regex: true
 source_labels:
 - __meta_kubernetes_annotation_prometheus_io_scrape_slow
 - action: replace
 regex: (https?)
 source_labels:
 - __meta_kubernetes_annotation_prometheus_io_scheme
 target_label: __scheme__
 - action: replace
 regex: (++)
 source_labels:
 - __meta_kubernetes_annotation_prometheus_io_path
 target_label: __metrics_path__
 - action: replace
 regex: ([^:]+)(:::\d+)?;(\d+)
 replacement: $$1:$$2
 source_labels:
 - __address__
 - __meta_kubernetes_annotation_prometheus_io_port
 target_label: __address__
 - action: labelmap
 regex: __meta_kubernetes_annotation_prometheus_io_param_(.+)
 replacement: __param_$$1
 - action: labelmap
 regex: __meta_kubernetes_pod_label_(.+)
 - action: replace
 source_labels:
 - __meta_kubernetes_namespace
 target_label: namespace
 - action: replace
 source_labels:
 - __meta_kubernetes_pod_name
 target_label: pod
 - action: drop
 regex: Pending|Succeeded|Failed|Completed
 source_labels:
 - __meta_kubernetes_pod_phase

processors:
batch/metrics:
 timeout: 60s
exporters:
aws_prometheus_remote_write:
 endpoint: "<YOUR_REMOTE_WRITE_ENDPOINT>""

```

```

aws_auth:
 region: "<AWS_REGION>"
 service: "aps"
service:
 pipelines:
 metrics:
 receivers: [prometheus]
 processors: [batch/metrics]
 exporters: [aws(prometheus)remote(write)]

apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
 name: otel-prometheus-role
rules:
 - apiGroups:
 - ""
 resources:
 - nodes
 - nodes/proxy
 - services
 - endpoints
 - pods
 verbs:
 - get
 - list
 - watch
 - apiGroups:
 - extensions
 resources:
 - ingresses
 verbs:
 - get
 - list
 - watch
 - nonResourceURLs:
 - /metrics
 verbs:
 - get

apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
 name: otel-prometheus-role-binding
roleRef:
 apiGroup: rbac.authorization.k8s.io
 kind: ClusterRole
 name: otel-prometheus-role
subjects:
 - kind: ServiceAccount
 name: adot-demo
 namespace: default

```

2. After you configured the YAML file, apply it to your cluster to deploy the ADOT Collector:

```
kubectl apply -f deployment.yaml
```

## Deploy the AWS Distro for OpenTelemetry Collector for CloudWatch

The AWS Distro for OpenTelemetry (ADOT) Collector can be deployed to receive OTLP metrics for export to Amazon CloudWatch. We have included an example YAML file that you can apply to your cluster. Replace the `region` value with your own.

```

OpenTelemetry Collector configuration
Metrics pipeline with Prometheus Receiver and AWS Amazon CloudWatch EMF Exporter
sending metrics to Amazon CloudWatch

apiVersion: opentelemetry.io/v1alpha1
kind: OpenTelemetryCollector
metadata:
 name: my-collector-cloudwatch
spec:
 mode: deployment
 serviceAccount: adot-demo
 podAnnotations:
 prometheus.io/scrape: 'true'
 prometheus.io/port: '8888'
 env:
 - name: CLUSTER_NAME
 value: <YOUR_EKS_CLUSTER_NAME>
 config: |
 receivers:
 #
 # Scrape configuration for the Prometheus Receiver
 # This is the same configuration used when Prometheus is installed using the
 # community Helm chart
 #
 prometheus:
 config:
 global:
 scrape_interval: 15s
 scrape_timeout: 10s
 scrape_configs:
 - job_name: kubernetes-apiservers
 bearer_token_file: /var/run/secrets/kubernetes.io/serviceaccount/token
 kubernetes_sd_configs:
 - role: endpoints
 relabel_configs:
 - action: keep
 regex: default;kubernetes;https
 source_labels:
 - __meta_kubernetes_namespace
 - __meta_kubernetes_service_name
 - __meta_kubernetes_endpoint_port_name
 scheme: https
 tls_config:
 ca_file: /var/run/secrets/kubernetes.io/serviceaccount/ca.crt
 insecure_skip_verify: true
 - job_name: kubernetes-nodes
 bearer_token_file: /var/run/secrets/kubernetes.io/serviceaccount/token
 kubernetes_sd_configs:
 - role: node
 relabel_configs:
 - action: labelmap
 regex: __meta_kubernetes_node_label_(.+)
 - replacement: kubernetes.default.svc:443
```

```
target_label: __address__
- regex: (.)
 replacement: /api/v1/nodes/$$1/proxy/metrics
 source_labels:
 - __meta_kubernetes_node_name
 target_label: __metrics_path__
 scheme: https
 tls_config:
 ca_file: /var/run/secrets/kubernetes.io/serviceaccount/ca.crt
 insecure_skip_verify: true
- job_name: kubernetes-nodes-cadvisor
 bearer_token_file: /var/run/secrets/kubernetes.io/serviceaccount/token
 kubernetes_sd_configs:
 - role: node
 relabel_configs:
 - action: labelmap
 regex: __meta_kubernetes_node_label_(.+)
 - replacement: kubernetes.default.svc:443
 target_label: __address__
 - regex: (.)
 replacement: /api/v1/nodes/$$1/proxy/metrics/cadvisor
 source_labels:
 - __meta_kubernetes_node_name
 target_label: __metrics_path__
 scheme: https
 tls_config:
 ca_file: /var/run/secrets/kubernetes.io/serviceaccount/ca.crt
 insecure_skip_verify: true
- job_name: kubernetes-service-endpoints
 kubernetes_sd_configs:
 - role: endpoints
 relabel_configs:
 - action: keep
 regex: true
 source_labels:
 - __meta_kubernetes_service_annotation_prometheus_io_scrape
 - action: replace
 regex: (https?)
 source_labels:
 - __meta_kubernetes_service_annotation_prometheus_io_scheme
 target_label: __scheme__
 - action: replace
 regex: (.)
 source_labels:
 - __meta_kubernetes_service_annotation_prometheus_io_path
 target_label: __metrics_path__
 - action: replace
 regex: ([^:]+)(?::\d+)?;(\d+)
 replacement: $$1:$$2
 source_labels:
 - __address__
 - __meta_kubernetes_service_annotation_prometheus_io_port
 target_label: __address__
 - action: labelmap
 regex: __meta_kubernetes_service_annotation_prometheus_io_param_(.+)
 replacement: __param_$$1
 - action: labelmap
 regex: __meta_kubernetes_service_label_(.+)
 - action: replace
 source_labels:
 - __meta_kubernetes_namespace
 target_label: kubernetes_namespace
 - action: replace
 source_labels:
 - __meta_kubernetes_service_name
 target_label: kubernetes_name
```

```
- action: replace
 source_labels:
 - __meta_kubernetes_pod_node_name
 target_label: kubernetes_node
- job_name: kubernetes-service-endpoints-slow
 kubernetes_sd_configs:
 - role: endpoints
 relabel_configs:
 - action: keep
 regex: true
 source_labels:
 - __meta_kubernetes_service_annotation_prometheus_io_scrape_slow
 - action: replace
 regex: (https?)
 source_labels:
 - __meta_kubernetes_service_annotation_prometheus_io_scheme
 target_label: __scheme__
 - action: replace
 regex: (.)
 source_labels:
 - __meta_kubernetes_service_annotation_prometheus_io_path
 target_label: __metrics_path__
 - action: replace
 regex: ([^:]+)(?::\d+)?;(\d+)
 replacement: $$1:$$2
 source_labels:
 - __address__
 - __meta_kubernetes_service_annotation_prometheus_io_port
 target_label: __address__
 - action: labelmap
 regex: __meta_kubernetes_service_annotation_prometheus_io_param_(.+)
 replacement: __param_$$1
 - action: labelmap
 regex: __meta_kubernetes_service_label_(.+)
 - action: replace
 source_labels:
 - __meta_kubernetes_namespace
 target_label: kubernetes_namespace
 - action: replace
 source_labels:
 - __meta_kubernetes_service_name
 target_label: kubernetes_name
 - action: replace
 source_labels:
 - __meta_kubernetes_pod_node_name
 target_label: kubernetes_node
 scrape_interval: 5m
 scrape_timeout: 30s

- job_name: prometheus-pushgateway
 kubernetes_sd_configs:
 - role: service
 relabel_configs:
 - action: keep
 regex: pushgateway
 source_labels:
 - __meta_kubernetes_service_annotation_prometheus_io_probe
- job_name: kubernetes-services
 kubernetes_sd_configs:
 - role: service
 metrics_path: /probe
 params:
 module:
 - http_2xx
 relabel_configs:
 - action: keep
```

```

 regex: true
 source_labels:
 - __meta_kubernetes_service_annotation_prometheus_io_probe
 - source_labels:
 - __address__
 target_label: __param_target
 - replacement: blackbox
 target_label: __address__
 - source_labels:
 - __param_target
 target_label: instance
 - action: labelmap
 regex: __meta_kubernetes_service_label_(.+)
 - source_labels:
 - __meta_kubernetes_namespace
 target_label: kubernetes_namespace
 - source_labels:
 - __meta_kubernetes_service_name
 target_label: kubernetes_name
 - job_name: kubernetes-pods
 kubernetes_sd_configs:
 - role: pod
 relabel_configs:
 - action: keep
 regex: true
 source_labels:
 - __meta_kubernetes_pod_annotation_prometheus_io_scrape
 - action: replace
 regex: (https?)
 source_labels:
 - __meta_kubernetes_pod_annotation_prometheus_io_scheme
 target_label: __scheme__
 - action: replace
 regex: (.+)
 source_labels:
 - __meta_kubernetes_pod_annotation_prometheus_io_path
 target_label: __metrics_path__
 - action: replace
 regex: ([^:]+)(?::\d+)?;(\d+)
 replacement: $$1:$$2
 source_labels:
 - __address__
 - __meta_kubernetes_pod_annotation_prometheus_io_port
 target_label: __address__
 - action: labelmap
 regex: __meta_kubernetes_pod_annotation_prometheus_io_param_(.+)
 replacement: __param_$$1
 - action: labelmap
 regex: __meta_kubernetes_pod_label_(.+)
 - action: replace
 source_labels:
 - __meta_kubernetes_namespace
 target_label: kubernetes_namespace
 - action: replace
 source_labels:
 - __meta_kubernetes_pod_name
 target_label: kubernetes_pod_name
 - action: drop
 regex: Pending|Succeeded|Failed|Completed
 source_labels:
 - __meta_kubernetes_pod_phase
 - job_name: kubernetes-pods-slow
 scrape_interval: 5m
 scrape_timeout: 30s
 kubernetes_sd_configs:

```

```

 - role: pod
 relabel_configs:
 - action: keep
 regex: true
 source_labels:
 - __meta_kubernetes_pod_annotation_prometheus_io_scrape_slow
 - action: replace
 regex: (https?)
 source_labels:
 - __meta_kubernetes_pod_annotation_prometheus_io_scheme
 target_label: __scheme__
 - action: replace
 regex: (.)
 source_labels:
 - __meta_kubernetes_pod_annotation_prometheus_io_path
 target_label: __metrics_path__
 - action: replace
 regex: ([^:]+)(?::(\d+)?);(\d+)
 replacement: $$1:$$2
 source_labels:
 - __address__
 - __meta_kubernetes_pod_annotation_prometheus_io_port
 target_label: __address__
 - action: labelmap
 regex: __meta_kubernetes_pod_annotation_prometheus_io_param_(.+)
 replacement: __param_$$1
 - action: labelmap
 regex: __meta_kubernetes_pod_label_(.+)
 - action: replace
 source_labels:
 - __meta_kubernetes_namespace
 target_label: namespace
 - action: replace
 source_labels:
 - __meta_kubernetes_pod_name
 target_label: pod
 - action: drop
 regex: Pending|Succeeded|Failed|Completed
 source_labels:
 - __meta_kubernetes_pod_phase

processors:
 batch/metrics:
 timeout: 60s
 #
 # Processor to transform the names of existing labels and/or add new labels to
 # the metrics identified
 #
 metricstransform/labelling:
 transforms:
 - include: .*
 match_type: regexp
 action: update
 operations:
 - action: add_label
 new_label: EKS_Cluster
 new_value: ${CLUSTER_NAME}
 - action: update_label
 label: kubernetes_pod_name
 new_label: EKS_PodName
 - action: update_label
 label: kubernetes_namespace
 new_label: EKS_Namespace
 exporters:
 #
 # AWS EMF exporter that sends metrics data as performance log events to

```

```

 # Amazon CloudWatch
 # Only the metrics that were filtered out by the processors get to this stage of
the pipeline
 # Under the metric_declarations field, add one or more sets of Amazon CloudWatch
dimensions
 # Each dimension must already exist as a label on the Prometheus metric
 # For each set of dimensions, add a list of metrics under the metric_name_selectors
field
 # Metrics names may be listed explicitly or using regular expressions
 # Data from performance log events will be aggregated by Amazon CloudWatch using
these
 # dimensions to create a Amazon CloudWatch custom metric.
 #
 awsemf:
 region: "<AWS_REGION>"
 namespace: ContainerInsights/Prometheus
 log_group_name: '/aws/containerinsights/${CLUSTER_NAME}/prometheus'
 resource_to_telemetry_conversion:
 enabled: true
 dimension_rollup_option: NoDimensionRollup
 parse_json_encoded_attr_values: [Sources, kubernetes]
 metric_declarations:
 - dimensions: [[EKS_Cluster, EKS_Namespace, EKS_PodName]]
service:
 pipelines:
 metrics:
 receivers: [prometheus]
 processors: [batch/metrics,metricstransform/labelling]
 exporters: [awsemf]

apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
 name: otel-prometheus-role
rules:
 - apiGroups:
 - ""
 resources:
 - nodes
 - nodes/proxy
 - services
 - endpoints
 - pods
 verbs:
 - get
 - list
 - watch
 - apiGroups:
 - extensions
 resources:
 - ingresses
 verbs:
 - get
 - list
 - watch
 - nonResourceURLs:
 - /metrics
 verbs:
 - get

apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
 name: otel-prometheus-role-binding
roleRef:

```

```
apiGroup: rbac.authorization.k8s.io
kind: ClusterRole
name: otel-prometheus-role
subjects:
- kind: ServiceAccount
 name: adot-demo
 namespace: default
```

1. Create a YAML file. In this example, it is named `collector-config-cloudwatch.yaml`.
2. Apply the YAML file using the command:

```
kubectl apply -f collector-config-cloudwatch.yaml
```

3. (Optional) Verify the metrics data is being sent to Amazon CloudWatch by opening the [Amazon CloudWatch console](#) and open the **Metrics** menu on the left. Select **All metrics** and click the **AOC Docker Demo/AOC Docker Demo Service** box under **custom namespaces**. You can view any metrics data by selecting any grouping.

## Deploy the AWS Distro for OpenTelemetry (ADOT) Collector for X-Ray

You can deploy the AWS Distro for OpenTelemetry (ADOT) Collector to send traces to X-Ray by applying a YAML configuration file. We have included an example YAML file called `collector-config-xray.yaml`:

```
apiVersion: opentelemetry.io/v1alpha1
kind: OpenTelemetryCollector
metadata:
 name: my-collector-xray
spec:
 mode: deployment
 serviceAccount: adot-demo
 config: |
 receivers:
 otlp:
 protocols:
 grpc:
 endpoint: 0.0.0.0:4317
 http:
 endpoint: 0.0.0.0:4318
 processors:
 exporters:
 awsxray:
 region: <AWS_REGION>
 service:
 pipelines:
 traces:
 receivers: [otlp]
 processors: []
 exporters: [awsxray]
```

1. Apply the YAML file using the command:

```
kubectl apply -f collector-config-xray.yaml
```

2. (Optional) Verify trace data is being sent to X-Ray by opening the [X-Ray console](#) and select **Traces** from the left menu. Your trace data should be present.

## Deploy a sample application to test the AWS Distro for OpenTelemetry Collector

The sample application will generate and send OTLP data to any of the services that you have configured through the AWS Distro for OpenTelemetry ([ADOT Collector deployment \(p. 521\)](#)). This step is optional if you already have an application running inside your cluster that can produce data. Consult your application's documentation to ensure that data is sent to the correct endpoints. The sample application set up includes applying two YAML files:

- A [traffic generator file](#) `traffic-generator.yaml`
- A [sample app file](#) `sample-app.yaml`

The sample application and traffic generator were largely taken from an example in the [ADOT Collector repository](#). A `docker-compose.yaml` file was translated to Kubernetes resources using the [Kompose tool](#).

`traffic-generator.yaml` makes http calls to the Kubernetes service `sample-app:4567`. This allows the traffic generator to interact with the sample application on port 4567. `sample-app` resolves to the IP address of the `sample-app` pod. Ensure that the `kind` value reflects your deployment mode.

`traffic-generator.yaml`

```
apiVersion: v1
metadata:
 labels:
 io.kompose.service: traffic-generator
 name: traffic-generator
spec:
 ports:
 - name: "80"
 port: 80
 targetPort: 80
 selector:
 io.kompose.service: traffic-generator
status:
 loadBalancer: {}

apiVersion: apps/v1
kind: Deployment
metadata:
 labels:
 io.kompose.service: traffic-generator
 name: traffic-generator
spec:
 replicas: 1
 selector:
 matchLabels:
 io.kompose.service: traffic-generator
 strategy: {}
 template:
 metadata:
 labels:
```

```

 io.kompose.service: traffic-generator
spec:
 containers:
 - args:
 - /bin/bash
 - -c
 - sleep 10; while :; do curl sample-app:4567/outgoing-http-call > /dev/null
1>&1; sleep 2; curl sample-app:4567/aws-sdk-call > /dev/null 2>&1; sleep 5; done
 image: ellerbrock/alpine-bash-curl-ssl:latest
 name: traffic-generator
 ports:
 - containerPort: 80
 resources: {}
 restartPolicy: Always
status: {}

```

#### sample-app.yaml

- The Service resource configures port:4567 to allow HTTP requests for the traffic generator.
- The Deployment resource configures some environment variables:
  - AWS\_REGION should be your AWS Region.
  - The LISTEN\_ADDRESS is configured to 0.0.0.0:4567 for HTTP requests from the traffic generator.
  - The OTEL\_EXPORTER\_OTLP\_ENDPOINT has a value of http://my-collector-collector:4317. my-collector-collector is the name of the Kubernetes service that allows the sample application to interact with the ADOT Collector on port 4317. In the ADOT Collector configuration, the ADOT Collector receives metrics and traces from an endpoint: 0.0.0.0:4317.

```

apiVersion: v1
kind: Service
metadata:
 labels:
 io.kompose.service: sample-app
 name: sample-app
spec:
 ports:
 - name: "4567"
 port: 4567
 targetPort: 4567
 selector:
 io.kompose.service: sample-app
status:
 loadBalancer: {}

apiVersion: apps/v1
kind: Deployment
metadata:
 labels:
 io.kompose.service: sample-app
 name: sample-app
spec:
 replicas: 1
 selector:
 matchLabels:
 io.kompose.service: sample-app
 strategy:
 type: Recreate
 template:
 metadata:
 labels:
 io.kompose.service: sample-app
 spec:

```

```
containers:
 - env:
 - name: AWS_REGION
 value: <AWS_REGION>
 - name: LISTEN_ADDRESS
 value: 0.0.0.0:4567
 - name: OTEL_EXPORTER_OTLP_ENDPOINT
 value: http://my-collector-collector:4317
 - name: OTEL_RESOURCE_ATTRIBUTES
 value: service.namespace=AOC Docker Demo,service.name=AOC Docker Demo Service
 image: public.ecr.aws/aws-otel-test/aws-otel-java-test-spark:v0.11.0
 name: sample-app
 ports:
 - containerPort: 4567
 resources: {}
 restartPolicy: Always
status: {}
```

## Deploy the traffic generator and sample application

To apply the traffic generator and sample application, use the following commands for each YAML file:

```
kubectl apply -f traffic-generator.yaml
```

```
kubectl apply -f sample-app.yaml
```

# AWS services integrated with Amazon EKS

Amazon EKS works with other AWS services to provide additional solutions for your business challenges. This topic identifies services that either use Amazon EKS to add functionality, or services that Amazon EKS uses to perform tasks.

## Topics

- [Creating Amazon EKS resources with AWS CloudFormation \(p. 538\)](#)
- [Use AWS App Mesh with Kubernetes \(p. 539\)](#)
- [Amazon EKS on AWS Outposts \(p. 539\)](#)
- [Amazon EKS on AWS Local Zones \(p. 545\)](#)
- [Deep Learning Containers \(p. 545\)](#)

## Creating Amazon EKS resources with AWS CloudFormation

Amazon EKS is integrated with AWS CloudFormation, a service that helps you model and set up your AWS resources so that you can spend less time creating and managing your resources and infrastructure. You create a template that describes all the AWS resources that you want, for example an Amazon EKS cluster, and AWS CloudFormation takes care of provisioning and configuring those resources for you.

When you use AWS CloudFormation, you can reuse your template to set up your Amazon EKS resources consistently and repeatedly. Just describe your resources once, and then provision the same resources over and over in multiple AWS accounts and Regions.

## Amazon EKS and AWS CloudFormation templates

To provision and configure resources for Amazon EKS and related services, you must understand [AWS CloudFormation templates](#). Templates are formatted text files in JSON or YAML. These templates describe the resources that you want to provision in your AWS CloudFormation stacks. If you're unfamiliar with JSON or YAML, you can use AWS CloudFormation Designer to help you get started with AWS CloudFormation templates. For more information, see [What is AWS CloudFormation Designer?](#) in the [AWS CloudFormation User Guide](#).

Amazon EKS supports creating clusters and node groups in AWS CloudFormation. For more information, including examples of JSON and YAML templates for your Amazon EKS resources, see [Amazon EKS resource type reference](#) in the [AWS CloudFormation User Guide](#).

## Learn more about AWS CloudFormation

To learn more about AWS CloudFormation, see the following resources:

- [AWS CloudFormation](#)
- [AWS CloudFormation User Guide](#)
- [AWS CloudFormation Command Line Interface User Guide](#)

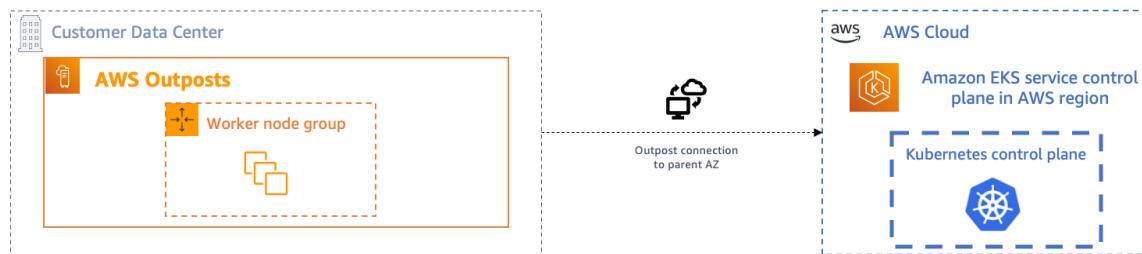
# Use AWS App Mesh with Kubernetes

AWS App Mesh (App Mesh) is a service mesh that makes it easy to monitor and control services. App Mesh standardizes how your services communicate, giving you end-to-end visibility and helping to ensure high availability for your applications. App Mesh gives you consistent visibility and network traffic controls for every service in an application. You can get started using App Mesh with Kubernetes by completing the [Getting started with AWS App Mesh and Kubernetes](#) tutorial in the AWS App Mesh User Guide. The tutorial recommends that you have existing services deployed to Kubernetes that you want to use App Mesh with.

## Amazon EKS on AWS Outposts

You can create and run Amazon EKS nodes on AWS Outposts. AWS Outposts enables native AWS services, infrastructure, and operating models in on-premises facilities for low latency, local data processing, and data residency needs. In AWS Outposts environments, you can use the same AWS APIs, tools, and infrastructure that you use in the AWS Cloud. For more information about AWS Outposts, see the [AWS Outposts User Guide](#).

You can use Amazon EKS to run Kubernetes applications on-premises with AWS Outposts. Amazon EKS on AWS Outposts supports extended clusters, with the Kubernetes control plane running in the parent AWS Region, and worker nodes running on AWS Outposts. The Kubernetes control plane is fully managed by AWS, and you can use the same Amazon EKS APIs, tools, and console to create and run Amazon EKS worker nodes on AWS Outposts.



## Prerequisites

The following are the prerequisites for using Amazon EKS nodes on AWS Outposts:

- You must have installed and configured an Outpost in your on-premises data center. For more information, see [Create an Outpost and order Outpost capacity](#) in the AWS Outposts User Guide.
- You must have a reliable network connection between your Outpost and its parent AWS Region. We recommend that you provide highly available, low-latency connectivity between your Outpost and its parent AWS Region. For more information, see [Outpost connectivity to the local network](#) in the AWS Outposts User Guide.
- The AWS Region for the Outpost must support Amazon EKS. For a list of supported AWS Regions, see [Amazon EKS service endpoints](#) in the AWS General Reference.

## Outpost considerations

### Architecture

- AWS Outposts are available in a variety of form factors including 1U and 2U Outposts servers and 42U Outposts racks. Amazon EKS is supported on the 42U Outposts racks only.

- A single subnet cannot span multiple logical Outposts, and inter-Outposts traffic must use customer-owned IP addresses and traverse the local network. Because of this, it is recommended to run a single Amazon EKS cluster per logical Outpost.
- Traffic from Amazon EKS worker nodes running on AWS Outposts to the control plane in an AWS Region stays within your VPC and traverses the service link connection. You can use private or public connectivity for your service link connection. For more information about the service link, see [Outposts Connectivity to AWS Regions](#) in the AWS Outposts User Guide.
- If network connectivity between your Outpost and its parent AWS Region is lost, your Amazon EKS worker nodes will continue to run. However, you cannot create new nodes or perform mutating management actions on existing deployments until connectivity is restored. The recommended course of action during network disconnects is to attempt to reconnect your AWS Outposts to the parent AWS Region following the [network connectivity checklist](#). In case of instance failures during periods of network disconnect, the instances will not be replaced automatically. The Amazon EKS Kubernetes control plane runs in the parent AWS Region, and missing kubelet heartbeats can lead to the following:
  - Pods on the AWS Outposts being marked as unhealthy
  - the Node status times out
  - the Pods will be marked for eviction

For more information, see [Node Controller](#) in the Kubernetes documentation.

## Operational

- When you create your Amazon EKS cluster, you must use subnets that run in the AWS Region because these are used for the creation of the Amazon EKS Kubernetes control plane. During cluster creation, do not use subnets that run on your AWS Outposts.
- You can run self-managed nodes on AWS Outposts only, managed nodes and Fargate are not supported. When you create a self-managed node group, you must pass the subnets that exist on your AWS Outposts.
- The Amazon EKS worker nodes' VPC must be associated with an LGW route table for Services running on the worker nodes to be accessible over the local network. If you are using customer-owned IPs, then the subnet in which the worker nodes run must have a route to the LGW.
- IPv6 cannot be used for service link or local network traffic on AWS Outposts. You shouldn't create your Amazon EKS cluster with the IPv6 IP family if you plan to run Amazon EKS worker nodes on AWS Outposts.
- The Kubernetes components for Amazon EKS worker nodes are pulled from Amazon Elastic Container Registry (Amazon ECR) in the parent AWS Region. When bootstrapping new Amazon EC2 instances to your Amazon EKS cluster, expect an increase in traffic over your service link connection.
- You can use Amazon ECR in the parent AWS Region to host your application container images. The size of your application container images will affect the service link bandwidth usage and the startup time when deploying new Pods that are not already cached. If you need to reduce your application deployment time, consider hosting a local container registry or cache for your application container images. If you have "isolated" subnets that do not have a path to the internet, you must set up [VPC Endpoints](#) for Amazon S3 and Amazon ECR to create Amazon EKS worker nodes on AWS Outposts.
- When creating your Amazon EKS worker nodes, you must use the gp2 volume type.

## Application

- With AWS Outposts, you can run some AWS services locally, and you can connect to a broad range of services available in the parent AWS Region. The locally available AWS services will have lower latency response times compared to those running in the parent AWS Region. The AWS services that you use in

the AWS Region will not be available during periods of network disconnect to the parent AWS Region. For information on the AWS services available locally, see the [AWS Outposts User Guide](#).

- You can connect to your applications running on Amazon EKS on AWS Outposts over your local network using the normal methods for Kubernetes [Services](#) and [Ingress](#). Application Load Balancer (ALB) is available for Kubernetes Ingress on Outposts racks. Network Load Balancer (Network Load Balancer) is not available on AWS Outposts. A common practice to conserve capacity on AWS Outposts is to use a single ALB deployment with [path-based routing](#) for each Kubernetes Service. For more information, see the [Application Load Balancer documentation](#).
- Amazon EBS volumes do not span physical or logical AWS Outposts, similar to the multi-Availability Zone behavior in AWS Regions. For example, if a Node running on physical Outpost A moves to physical Outpost B, the EBS volume will not move with it. If you are running stateful workloads on Amazon EKS with Pods backed by EBS volumes, consider implementing dual writes at the application layer or using an alternative storage mechanism if your application and its data must remain available during single-rack failures.

## Deploy an Amazon EKS cluster with worker nodes on AWS Outposts

This section describes how to create an Amazon EKS cluster and deploy Amazon EKS worker nodes on AWS Outposts using `eksctl` and the AWS CLI. You must have permissions to create and manage subnets on AWS Outposts. Click here to learn more about shareable Outpost resources<https://docs.aws.amazon.com/outposts/latest/userguide/sharing-outposts.html#sharing-resources>.

1. Create a public Amazon EKS cluster using a YAML file:

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

An example YAML file is shown below:

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

metadata:
 name: my-outposts-cluster
 version: 1.21
 region: us-west-2

cloudWatch:
 clusterLogging:
 enableTypes: ["api", "audit", "authenticator", "controllerManager", "scheduler"]

iam:
 withOIDC: true

addons:
 - name: vpc-cni
 - name: coredns
 - name: kube-proxy
```

To create a fully private cluster, add the following to your config YAML file:

```
privateCluster:
 enabled: true
```

Your output should look similar to this:

```
eksctl version 0.80.0
using region us-west-2
subnets for us-west-2a - public:192.168.0.0/19 private:192.168.96.0/19
subnets for us-west-2b - public:192.168.32.0/19 private:192.168.128.0/19
subnets for us-west-2c - public:192.168.64.0/19 private:192.168.160.0/19
using Kubernetes version 1.21
creating EKS cluster "my-outpost-cluster" in "us-west-2" region with
if you encounter any issues, check CloudFormation console or try 'eksctl utils
 describe-stacks --region=us-west-2 --cluster=my-outpost-cluster'
Kubernetes API endpoint access will use default of {publicAccess=true,
 privateAccess=false} for cluster "my-outpost-cluster" in "us-west-2"
CloudWatch logging will not be enabled for cluster "my-outpost-cluster" in "us-west-2"
you can enable it with 'eksctl utils update-cluster-logging --enable-types={SPECIFY-
YOUR-LOG-TYPES-HERE (e.g. all)} --region=us-west-2 --cluster=my-outpost-cluster'
2 sequential tasks: { create cluster control plane "my-outpost-cluster", wait for
 control plane to become ready
}
building cluster stack "eksctl-my-outpost-cluster-cluster"
deploying stack "eksctl-my-outpost-cluster-cluster"
waiting for CloudFormation stack "eksctl-my-outpost-cluster-cluster"
waiting for CloudFormation stack "eksctl-my-outpost-cluster-cluster"...
waiting for the control plane availability...
saved kubeconfig as "../.kube/config"
no tasks
all EKS cluster resources for "my-outpost-cluster" have been created
kubectl command should work with "../.kube/config", try 'kubectl get nodes'
EKS cluster "my-outpost-cluster" in "us-west-2" region is ready
```

You can confirm your cluster has been created in the Amazon EKS AWS Management Console or the command:

```
eksctl get cluster --name my-outposts-cluster
```

2. Identify the VPC created with your new cluster. This VPC will host the subnet that contains your worker nodes. You can find your `vpc-id` with the command:

```
eksctl get cluster --name my-outposts-cluster
```

3. Identify the subnet and CIDR for your Outpost. The CIDR shouldn't conflict with other IP addresses in use on your local network. This can be done by using a [subnet calculator](#) and determining the appropriate settings for your environment. Once you have calculated the CIDR, you can use this value in the next step. Your `availability-zone` is visible in the AWS Management Console.
4. Create a subnet on AWS Outposts using AWS CLI to host your worker nodes. Use the `vpc-id` that you retrieved in the previous step in the following command:

```
aws ec2 create-subnet \
 --region us-west-2 \
 --availability-zone us-west-2 \
 --outpost-arn your-outpost-arn \
 --vpc-id your-vpc-id \
 --cidr-block your-cidr-block
```

Note the subnet id that appears in the message after creation.

Your output should look similar to this:

```
{
 "Subnet": {
 "AvailabilityZone": "us-west-2b",
 "AvailabilityZoneId": "usw2-az1",
 "AvailableIpAddressCount": 11,
 "CidrBlock": "192.168.192.0/28",
 "DefaultForAz": false,
 "MapPublicIpOnLaunch": false,
 "State": "available",
 "SubnetId": "subnet-042e2c531a5713a5b",
 "VpcId": "vpc-08edbe538f4045578",
 "OwnerId": "[user-arn]",
 "AssignIpv6AddressOnCreation": false,
 "Ipv6CidrBlockAssociationSet": [],
 "SubnetArn": "arn:aws:ec2:us-west-2:[user-arn]:subnet/subnet-042e2c531a5713a5b",
 "OutpostArn": "arn:aws:outposts:us-west-2:[user-arn]:outpost/op-039edede540aa85d85",
 "EnableDns64": false,
 "Ipv6Native": false,
 "PrivateDnsNameOptionsOnLaunch": {
 "HostnameType": "ip-name",
 "EnableResourceNameDnsARecord": false,
 "EnableResourceNameDnsAAAARecord": false
 }
 }
}
```

You can verify the subnet creation in the AWS Management Console under your cluster details.

5. Create worker nodes by using a YAML file. Replace the `vpc-id` with the value created in step 1 and `subnet-id` created in step 2. The `security-group-id` was created in step 1 and can be retrieved with the command:

```
eksctl get cluster --name my-outposts-cluster
```

The nodes can be created with a YAML file by using the command:

```
eksctl create nodegroup -f worker-nodes-config.yaml
```

The following is an example YAML file to include in this step. For `instanceType`, select the instance size that is:

- slotted on your Outpost
- available for your workload

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

metadata:
 name: my-outposts-cluster
 region: us-west-2

nodeGroups:
 - name: outpost-worker-nodes
 instanceType: m5.large
 desiredCapacity: 1
 minSize: 1
 maxSize: 1
 volumeSize: 50
 volumeType: gp2
```

```
volumeEncrypted: true
subnets:
 - subnet-042e2c531a5713a5b
privateNetworking: true
```

To create private worker nodes, add this to the end of the YAML file:

```
subnets:
 - outpost-subnet
volumeType: gp2
privateNetworking: true
```

Your output should look similar to this:

```
all nodegroups have up-to-date cloudformation templates
eksctl version 0.80.0
using region us-west-2
will use version 1.21 for new nodegroup(s) based on control plane version
nodegroup "outpost-worker-nodes" will use "ami-085e8e02353a59de5" [AmazonLinux2/1.21]1
 nodegroup (outpost-worker-nodes) was included (based on the include/exclude rules)
 will create a CloudFormation stack for each of 1 nodegroups in cluster "my-outpost-
cluster"
2 sequential tasks: { fix cluster compatibility, 1 task: { 1 task: { create nodegroup
 "outpost-worker-nodes" } } }
}
checking cluster stack for missing resources
cluster stack has all required resources
building nodegroup stack "eksctl-my-outpost-cluster-nodegroup-outpost-worker-nodes"
deploying stack "eksctl-my-outpost-cluster-nodegroup-outpost-worker-nodes"
waiting for CloudFormation stack "eksctl-my-outpost-cluster-nodegroup-outpost-worker-
nodes"
waiting for CloudFormation stack "eksctl-my-outpost-cluster-nodegroup-outpost-worker-
nodes"...
no tasks
adding identity "arn:aws:iam::601017151385:role/eksctl-my-outpost-cluster-nodegro-
NodeInstanceRole-LHYFK4K4DV85" to auth ConfigMap
nodegroup "outpost-worker-nodes" has 0 node(s)
waiting for at least 1 node(s) to become ready in "outpost-worker-nodes"
nodegroup "outpost-worker-nodes" has 2 node(s)
node "ip-192-168-192-5.us-west-2.compute.internal" is not ready
node "ip-192-168-192-9.us-west-2.compute.internal" is ready
created 1 nodegroup(s) in cluster "my-outpost-cluster"
created 0 managed nodegroup(s) in cluster "my-outpost-cluster"
checking security group configuration for all nodegroups
```

6. (Optional) To expose worker nodes over your local network, see [How local gateways work](#).

Now that you have a configured Amazon EKS cluster with worker nodes running on AWS Outposts, you can install add-ons and deploying applications to your cluster. See the following for more information on how to extend your cluster's functionality:

- The IAM entity (user or role) that created the cluster is the only IAM user that can make calls to the API server using `kubectl`. To grant access to other users or roles, see [Apply the aws-authConfigMap to your cluster \(p. 408\)](#).
- [Deploy a sample application \(p. 359\)](#) to your cluster.
- Learn about important [Cluster management \(p. 422\)](#) tools.
- [Deploy a containerized application on AWS Outposts](#).
- [Build a modern application on AWS Outposts with Terraform examples](#).

# Amazon EKS on AWS Local Zones

An AWS Local Zone is an extension of an AWS Region in geographic proximity to your users. Local Zones have their own connections to the internet and support AWS Direct Connect. Resources created in a Local Zone can serve local users with low-latency communications. For more information, see [Local Zones](#).

Amazon EKS supports running certain infrastructure. This includes Amazon EC2 instances, Amazon EBS volumes, and Application Load Balancers (ALBs) from a Local Zone as part of your cluster. We recommend that you consider the following when using Local Zone infrastructure as part of your Amazon EKS cluster.

## Kubernetes versions

Only Amazon EKS clusters that run Kubernetes versions 1.17 and later can use Local Zone compute resources.

## Nodes

You can't create managed node groups in AWS Local Zones with Amazon EKS. However, you can create self-managed nodes in AWS Local Zones using the Amazon EC2 API, AWS CloudFormation, or `eksctl`. For more information, see [Self-managed nodes \(p. 132\)](#).

## Network architecture

The Amazon EKS managed Kubernetes control plane always runs in the AWS Region. The Amazon EKS managed Kubernetes control plane can't run in the Local Zone. Because Local Zones appear as a subnet within your VPC, Kubernetes sees your Local Zone resources as part of that subnet.

The Amazon EKS Kubernetes cluster communicates with the Amazon EC2 instances you run in the AWS Region or Local Zone using Amazon EKS managed [elastic network interfaces](#). To learn more about Amazon EKS networking architecture, see [Amazon EKS networking \(p. 263\)](#).

Unlike regional subnets, Amazon EKS can't place network interfaces into your Local Zone subnets. This means that you must not specify Local Zone subnets when you create your cluster.

After the cluster is created, tag your Local Zone subnets with the Amazon EKS cluster name. For more information, see [Subnet tagging \(p. 272\)](#). You can then deploy self-managed nodes to the Local Zone subnets and the nodes join your Amazon EKS cluster.

# Deep Learning Containers

AWS Deep Learning Containers are a set of Docker images for training and serving models in TensorFlow on Amazon EKS and Amazon Elastic Container Service (Amazon ECS). Deep Learning Containers provide optimized environments with TensorFlow, Nvidia CUDA (for GPU instances), and Intel MKL (for CPU instances) libraries and are available in Amazon ECR.

To get started using AWS Deep Learning Containers on Amazon EKS, see [AWS Deep Learning Containers on Amazon EKS](#) in the [AWS Deep Learning AMI Developer Guide](#).

# Amazon EKS troubleshooting

This chapter covers some common errors that you may see while using Amazon EKS and how to work around them.

## Insufficient capacity

If you receive the following error while attempting to create an Amazon EKS cluster, then one of the Availability Zones you specified does not have sufficient capacity to support a cluster.

```
Cannot create cluster 'example-cluster' because region-1d, the targeted Availability Zone, does not currently have sufficient capacity to support the cluster. Retry and choose from these Availability Zones: region-1a, region-1b, region-1c
```

Retry creating your cluster with subnets in your cluster VPC that are hosted in the Availability Zones returned by this error message.

## Nodes fail to join cluster

There are a few common reasons that prevent nodes from joining the cluster:

- The [aws-auth-cm.yaml \(p. 402\)](#) file does not have the correct IAM role ARN for your nodes. Ensure that the node IAM role ARN (not the instance profile ARN) is specified in your `aws-auth-cm.yaml` file. For more information, see [Launching self-managed Amazon Linux nodes \(p. 133\)](#).
- The **ClusterName** in your node AWS CloudFormation template does not exactly match the name of the cluster you want your nodes to join. Passing an incorrect value to this field results in an incorrect configuration of the node's `/var/lib/kubelet/kubeconfig` file, and the nodes will not join the cluster.
- The node is not tagged as being *owned* by the cluster. Your nodes must have the following tag applied to them, where `cluster-name` is replaced with the name of your cluster.

| Key                                                    | Value              |
|--------------------------------------------------------|--------------------|
| <code>kubernetes.io/cluster/<i>cluster-name</i></code> | <code>owned</code> |

- The nodes may not be able to access the cluster using a public IP address. Ensure that nodes deployed in public subnets are assigned a public IP address. If not, you can associate an Elastic IP address to a node after it's launched. For more information, see [Associating an Elastic IP address with a running instance or network interface](#). If the public subnet is not set to automatically assign public IP addresses to instances deployed to it, then we recommend enabling that setting. For more information, see [Modifying the public IPv4 addressing attribute for your subnet](#). If the node is deployed to a private subnet, then the subnet must have a route to a NAT gateway that has a public IP address assigned to it.
- The STS endpoint for the AWS Region that you're deploying the nodes to is not enabled for your account. To enable the region, see [Activating and deactivating AWS STS in an AWS Region](#).
- The worker node does not have a private DNS entry, resulting in the kubelet log containing a `node "" not found` error. Ensure that the VPC where the worker node is created has values set for `domain-name` and `domain-name-servers` as Options in a DHCP options set.

The default values are `domain-name:<region>.compute.internal` and `domain-name-servers:AmazonProvidedDNS`. For more information, see [DHCP options sets](#) in the Amazon VPC User Guide.

## Unauthorized or access denied (kubectl)

If you receive one of the following errors while running `kubectl` commands, then your `kubectl` is not configured properly for Amazon EKS or the IAM user or role credentials that you are using do not map to a Kubernetes RBAC user with sufficient permissions in your Amazon EKS cluster.

- `could not get token: AccessDenied: Access denied`
- `error: You must be logged in to the server (Unauthorized)`
- `error: the server doesn't have a resource type "svc"`

This could be because the cluster was created with one set of AWS credentials (from an IAM user or role), and `kubectl` is using a different set of credentials.

When an Amazon EKS cluster is created, the IAM entity (user or role) that creates the cluster is added to the Kubernetes RBAC authorization table as the administrator (with `system:masters` permissions). Initially, only that IAM user can make calls to the Kubernetes API server using `kubectl`. For more information, see [Enabling IAM user and role access to your cluster \(p. 402\)](#). If you use the console to create the cluster, you must ensure that the same IAM user credentials are in the AWS SDK credential chain when you are running `kubectl` commands on your cluster.

If you install and configure the AWS CLI, you can configure the IAM credentials for your user. For more information, see [Configuring the AWS CLI in the AWS Command Line Interface User Guide](#).

If you assumed a role to create the Amazon EKS cluster, you must ensure that `kubectl` is configured to assume the same role. Use the following command to update your `kubeconfig` file to use an IAM role. For more information, see [Create a kubeconfig for Amazon EKS \(p. 413\)](#).

```
aws eks update-kubeconfig \
 --region region-code \
 --name my-cluster \
 --role-arn arn:aws:iam::111122223333:role/role_name
```

To map an IAM user to a Kubernetes RBAC user, see [Enabling IAM user and role access to your cluster \(p. 402\)](#).

## aws-iam-authenticator Not found

If you receive the error "`aws-iam-authenticator`: executable file not found in \$PATH", then your `kubectl` is not configured for Amazon EKS. For more information, see [Installing aws-iam-authenticator \(p. 417\)](#).

### Note

The `aws-iam-authenticator` isn't required if you have the AWS CLI version 1.16.156 or higher installed.

## hostname doesn't match

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

## getsockopt: no route to host

Docker runs in the 172.17.0.0/16 CIDR range in Amazon EKS clusters. We recommend that your cluster's VPC subnets do not overlap this range. Otherwise, you will receive the following error:

```
Error: : error upgrading connection: error dialing backend: dial tcp
172.17.<nn>.<nn>:10250: getsockopt: no route to host
```

## Managed node group errors

If you receive the error "Instances failed to join the Kubernetes cluster" in the AWS Management Console, ensure that either the cluster's private endpoint access is enabled, or that you have correctly configured CIDR blocks for public endpoint access. For more information, see [Amazon EKS cluster endpoint access control \(p. 47\)](#).

If your managed node group encounters a hardware health issue, Amazon EKS returns an error message to help you to diagnose the issue. These health checks don't detect software issues because they are based on [Amazon EC2 health checks](#). The following error messages and their associated descriptions are shown below.

- **AccessDenied**: Amazon EKS or one or more of your managed nodes is failing to authenticate or authorize with your Kubernetes cluster API server. For more information about resolving this error, see [Fixing AccessDenied errors for managed node groups \(p. 549\)](#).
- **AmildNotFound**: We couldn't find the AMI Id associated with your Launch Template. Make sure that the AMI exists and is shared with your account.
- **AutoScalingGroupNotFound**: We couldn't find the Auto Scaling group associated with the managed node group. You may be able to recreate an Auto Scaling group with the same settings to recover.
- **ClusterUnreachable**: Amazon EKS or one or more of your managed nodes is unable to communicate with your Kubernetes cluster API server. This can happen if there are network disruptions or if API servers are timing out processing requests.
- **Ec2SecurityGroupNotFound**: We couldn't find the cluster security group for the cluster. You must recreate your cluster.
- **Ec2SecurityGroupDeletionFailure**: We could not delete the remote access security group for your managed node group. Remove any dependencies from the security group.
- **Ec2LaunchTemplateNotFound**: We couldn't find the Amazon EC2 launch template for your managed node group. You must recreate your node group to recover.
- **Ec2LaunchTemplateVersionMismatch**: The Amazon EC2 launch template version for your managed node group does not match the version that Amazon EKS created. You may be able to revert to the version that Amazon EKS created to recover.
- **IamInstanceProfileNotFound**: We couldn't find the IAM instance profile for your managed node group. You may be able to recreate an instance profile with the same settings to recover.
- **IamNodeRoleNotFound**: We couldn't find the IAM role for your managed node group. You may be able to recreate an IAM role with the same settings to recover.
- **AsgInstanceLaunchFailures**: Your Auto Scaling group is experiencing failures while attempting to launch instances.
- **NodeCreationFailure**: Your launched instances are unable to register with your Amazon EKS cluster. Common causes of this failure are insufficient [node IAM role \(p. 474\)](#) permissions or lack of outbound internet access for the nodes. Your nodes must be able to access the internet using a public IP address to function properly. For more information, see [VPC IP addressing \(p. 270\)](#). Your nodes must also have ports open to the internet. For more information, see [Amazon EKS security group considerations \(p. 272\)](#).
- **InstanceLimitExceeded**: Your AWS account is unable to launch any more instances of the specified instance type. You may be able to request an Amazon EC2 instance limit increase to recover.

- **InsufficientFreeAddresses:** One or more of the subnets associated with your managed node group does not have enough available IP addresses for new nodes.
- **InternalFailure:** These errors are usually caused by an Amazon EKS server-side issue.

## Fixing AccessDenied errors for managed node groups

The most common cause of `AccessDenied` errors when performing operations on managed node groups is missing the `eks:node-manager` `ClusterRole` or `ClusterRoleBinding`. Amazon EKS sets up these resources in your cluster as part of onboarding with managed node groups, and these are required for managing the node groups.

The `ClusterRole` may change over time, but it should look similar to the following example:

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
 name: eks:node-manager
rules:
- apiGroups:
 - ''
 resources:
 - pods
 verbs:
 - get
 - list
 - watch
 - delete
- apiGroups:
 - ''
 resources:
 - nodes
 verbs:
 - get
 - list
 - watch
 - patch
- apiGroups:
 - ''
 resources:
 - pods/eviction
 verbs:
 - create
```

The `ClusterRoleBinding` may change over time, but it should look similar to the following example:

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
 name: eks:node-manager
roleRef:
 apiGroup: rbac.authorization.k8s.io
 kind: ClusterRole
 name: eks:node-manager
subjects:
- apiGroup: rbac.authorization.k8s.io
 kind: User
 name: eks:node-manager
```

Verify that the `eks:node-manager` `ClusterRole` exists.

```
kubectl describe clusterrole eks:node-manager
```

If present, compare the output to the previous ClusterRole example.

Verify that the eks:node-manager ClusterRoleBinding exists.

```
kubectl describe clusterrolebinding eks:node-manager
```

If present, compare the output to the previous ClusterRoleBinding example.

If you've identified a missing or broken ClusterRole or ClusterRoleBinding as the cause of an AccessDenied error while requesting managed node group operations, you can restore them. Save the following contents to a file named `eks-node-manager-role.yaml`.

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
 name: eks:node-manager
rules:
- apiGroups:
 - ''
 resources:
 - pods
 verbs:
 - get
 - list
 - watch
 - delete
- apiGroups:
 - ''
 resources:
 - nodes
 verbs:
 - get
 - list
 - watch
 - patch
- apiGroups:
 - ''
 resources:
 - pods/eviction
 verbs:
 - create

apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
 name: eks:node-manager
roleRef:
 apiGroup: rbac.authorization.k8s.io
 kind: ClusterRole
 name: eks:node-manager
subjects:
- apiGroup: rbac.authorization.k8s.io
 kind: User
 name: eks:node-manager
```

Apply the file.

```
kubectl apply -f eks-node-manager-role.yaml
```

Retry the node group operation to see if that resolved your issue.

## CNI log collection tool

The Amazon VPC CNI plugin for Kubernetes has its own troubleshooting script that is available on nodes at `/opt/cni/bin/aws-cni-support.sh`. You can use the script to collect diagnostic logs for support cases and general troubleshooting.

Use the following command to run the script on your node:

```
sudo bash /opt/cni/bin/aws-cni-support.sh
```

### Note

If the script is not present at that location, then the CNI container failed to run. You can manually download and run the script with the following command:

```
curl -O https://raw.githubusercontent.com/awslabs/amazon-eks-ami/master/log-
collector-script/linux/eks-log-collector.sh
sudo bash eks-log-collector.sh
```

The script collects the following diagnostic information. The CNI version that you have deployed can be earlier than the script version.

```
This is version 0.6.1. New versions can be found at https://github.com/awslabs/
amazon-eks-ami

Trying to collect common operating system logs...
Trying to collect kernel logs...
Trying to collect mount points and volume information...
Trying to collect SELinux status...
Trying to collect iptables information...
Trying to collect installed packages...
Trying to collect active system services...
Trying to collect Docker daemon information...
Trying to collect kubelet information...
Trying to collect L-IPAMD information...
Trying to collect sysctls information...
Trying to collect networking information...
Trying to collect CNI configuration information...
Trying to collect running Docker containers and gather container data...
Trying to collect Docker daemon logs...
Trying to archive gathered information...

Done... your bundled logs are located in /var/log/eks_i-0717c9d54b6cfcaa19_2020-03-24_0103-
UTC_0.6.1.tar.gz
```

The diagnostic information is collected and stored at:

```
/var/log/eks_i-0717c9d54b6cfcaa19_2020-03-24_0103-UTC_0.6.1.tar.gz
```

## Container runtime network not ready

You may receive a `Container runtime network not ready` error and authorization errors similar to the following:

```
4191 kubelet.go:2130] Container runtime network not ready: NetworkReady=false
reason:NetworkPluginNotReady message:docker: network plugin is not ready: cni config
uninitialized
```

```
4191 reflector.go:205] k8s.io/kubernetes/pkg/kubelet/kubelet.go:452: Failed to list
 *v1.Service: Unauthorized
4191 kubelet_node_status.go:106] Unable to register node "ip-10-40-175-122.ec2.internal"
 with API server: Unauthorized
4191 reflector.go:205] k8s.io/kubernetes/pkg/kubelet/kubelet.go:452: Failed to list
 *v1.Service: Unauthorized
```

The errors are most likely because the AWS IAM Authenticator (`aws-auth`) configuration map isn't applied to the cluster. The configuration map provides the `system:bootstrappers` and `system:nodes` Kubernetes RBAC permissions for nodes to register to the cluster. To apply the configuration map to your cluster, see [Apply the `aws-authConfigMap` to your cluster \(p. 408\)](#).

The authenticator does not recognize a **Role ARN** if it includes a **path** other than `/`, such as the following example:

```
arn:aws:iam::111122223333:role/development/apps/prod-iam-role-NodeInstanceRole-621LVEEXAMPLE
```

When specifying a **Role ARN** in the configuration map that includes a path other than `/`, you must drop the path. The ARN above should be specified as the following:

```
arn:aws:iam::111122223333:role/prod-iam-role-NodeInstanceRole-621LVEEXAMPLE
```

## TLS handshake timeout

When a node is unable to establish a connection to the public API server endpoint, you may see an error similar to the following error.

```
server.go:233] failed to run Kubelet: could not init cloud provider "aws": error finding
instance i-1111f222f333e44c: "error listing AWS instances: \"RequestError: send request
failed\\ncaused by: Post net/http: TLS handshake timeout\""
```

The `kubelet` process will continually respawn and test the API server endpoint. The error can also occur temporarily during any procedure that performs a rolling update of the cluster in the control plane, such as a configuration change or version update.

To resolve the issue, check the route table and security groups to ensure that traffic from the nodes can reach the public endpoint.

## InvalidClientTokenId

If you're using IAM roles for service accounts for a pod or `DaemonSet` deployed to a cluster in a China AWS Region, and haven't set the `AWS_DEFAULT_REGION` environment variable in the spec, the pod or `DaemonSet` may receive the following error:

```
An error occurred (InvalidClientTokenId) when calling the GetCallerIdentity operation: The
security token included in the request is invalid
```

To resolve the issue, you need to add the `AWS_DEFAULT_REGION` environment variable to your pod or `DaemonSet` spec, as shown in the following example pod spec.

```
apiVersion: v1
kind: Pod
metadata:
 name: envar-demo
 labels:
```

```
purpose: demonstrate-envars
spec:
 containers:
 - name: envar-demo-container
 image: gcr.io/google-samples/node-hello:1.0
 env:
 - name: AWS_DEFAULT_REGION
 value: "region-code"
```

## VPC admission webhook certificate expiration

If the certificate used to sign the VPC admission webhook expires, the status for new Windows pod deployments stays at `ContainerCreating`.

To resolve the issue if you have legacy Windows support on your data plane, see [the section called "Renew VPC admission webhook certificate" \(p. 99\)](#). If your cluster and platform version are later than a version listed in the [Windows support prerequisites \(p. 93\)](#), then we recommend that you remove legacy Windows support on your data plane and enable it for your control plane. Once you do, you don't need to manage the webhook certificate. For more information, see [Windows support \(p. 92\)](#).

## Node groups must match Kubernetes version before updating control plane

Before you update a control plane to a new Kubernetes version, the minor version of the managed and Fargate nodes in your cluster must be the same as the version of your control plane's current version. The EKS `update-cluster-version` API rejects requests until you update all EKS managed nodes to the current cluster version. EKS provides APIs to update managed nodes. For information on updating managed node group Kubernetes versions, see [Updating a managed node group \(p. 119\)](#). To update the version of a Fargate node, delete the pod that's represented by the node and redeploy the pod after you update your control plane. For more information, see [Updating a cluster \(p. 34\)](#).

## When launching many nodes, there are Too Many Requests errors

If you launch many nodes simultaneously, you may see an error message in the [Amazon EC2 user data execution logs](#) that says `Too Many Requests`. This can occur because the control plane is being overloaded with `describeCluster` calls. The overloading results in throttling, nodes failing to run the bootstrap script, and nodes failing to join the cluster altogether.

Make sure that `--apiserver-endpoint`, `--b64-cluster-ca`, and `--dns-cluster-ip` arguments are being passed to the worker node bootstrap script. When including these arguments, there's no need for the bootstrap script to make a `describeCluster` call, which helps prevent the control plane from being overloaded. For more information, see [Provide user data to pass arguments to the bootstrap.sh file included with an Amazon EKS optimized AMI \(p. 128\)](#).

## HTTP 401 unauthorized error response on Kubernetes API server requests

You see these errors if a pod's service account token has expired on a 1.21 or later cluster.

Your Amazon EKS version 1.21 or later cluster's Kubernetes API server rejects requests with tokens older than 90 days. In previous Kubernetes versions, tokens did not have an expiration. This means that clients that rely on these tokens must refresh them within an hour. To prevent the Kubernetes API server from

rejecting your request due to an invalid token, the Kubernetes client SDK version used by your workload must be the same, or later than the following versions:

- Go v0.15.7 and later
- Python v12.0.0 and later
- Java v9.0.0 and later
- JavaScript v0.10.3 and later
- Ruby master branch
- Haskell v0.3.0.0
- C# v7.0.5 and later

You can identify all existing pods in your cluster that are using stale tokens. For more information, see [Kubernetes service accounts \(p. 441\)](#).

## Troubleshooting IAM

This topic covers some common errors that you may see while using Amazon EKS with IAM and how to work around them.

### AccessDeniedException

If you receive an `AccessDeniedException` when calling an AWS API operation, then the AWS Identity and Access Management (IAM) user or role credentials that you are using do not have the required permissions to make that call.

```
An error occurred (AccessDeniedException) when calling the DescribeCluster operation:
User: arn:aws:iam::111122223333:user/user_name is not authorized to perform:
eks:DescribeCluster on resource: arn:aws:eks:region:111122223333:cluster/my-cluster
```

In the above example message, the user does not have permissions to call the Amazon EKS `DescribeCluster` API operation. To provide Amazon EKS admin permissions to a user, see [Amazon EKS identity-based policy examples \(p. 462\)](#).

For more general information about IAM, see [Controlling access using policies](#) in the *IAM User Guide*.

### Can't see Nodes on the Overview tab or anything on the Resources tab and you receive an error in the AWS Management Console

You may see a console error message that says `Your current user or role does not have access to Kubernetes objects on this EKS cluster`. Make sure that the IAM user that you're signed into the AWS Management Console with (or role that you switched to after signing in) has the necessary permissions. For more information, see [Required permissions \(p. 506\)](#).

### aws-auth ConfigMap does not grant access to the cluster

`AWS IAM Authenticator` does not permit a path in the role ARN used in the `ConfigMap`. Therefore, before you specify `rolearn`, remove the path. For example, change `arn:aws:iam::111122223333:role/team/developers/eks-admin` to `arn:aws:iam::111122223333:role/eks-admin`.

## I Am not authorized to perform iam:PassRole

If you receive an error that you're not authorized to perform the `iam:PassRole` action, then you must contact your administrator for assistance. Your administrator is the person that provided you with your user name and password. Ask that person to update your policies to allow you to pass a role to Amazon EKS.

Some AWS services allow you to pass an existing role to that service, instead of creating a new service role or service-linked role. To do this, you must have permissions to pass the role to the service.

The following example error occurs when an IAM user named `marymajor` tries to use the console to perform an action in Amazon EKS. However, the action requires the service to have permissions granted by a service role. Mary does not have permissions to pass the role to the service.

```
User: arn:aws:iam::123456789012:user/marymajor is not authorized to perform: iam:PassRole
```

In this case, Mary asks her administrator to update her policies to allow her to perform the `iam:PassRole` action.

## I want to view my access keys

After you create your IAM user access keys, you can view your access key ID at any time. However, you can't view your secret access key again. If you lose your secret key, you must create a new access key pair.

Access keys consist of two parts: an access key ID (for example, `AKIAIOSFODNN7EXAMPLE`) and a secret access key (for example, `wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY`). Like a user name and password, you must use both the access key ID and secret access key together to authenticate your requests. Manage your access keys as securely as you do your user name and password.

**Important**

Do not provide your access keys to a third party, even to help [find your canonical user ID](#). By doing this, you might give someone permanent access to your account.

When you create an access key pair, you are prompted to save the access key ID and secret access key in a secure location. The secret access key is available only at the time you create it. If you lose your secret access key, you must add new access keys to your IAM user. You can have a maximum of two access keys. If you already have two, you must delete one key pair before creating a new one. To view instructions, see [Managing access keys](#) in the *IAM User Guide*.

## I'm an administrator and want to allow others to access Amazon EKS

To allow others to access Amazon EKS, you must create an IAM entity (user or role) for the person or application that needs access. They will use the credentials for that entity to access AWS. You must then attach a policy to the entity that grants them the correct permissions in Amazon EKS.

To get started right away, see [Creating your first IAM delegated user and group](#) in the *IAM User Guide*.

## I want to allow people outside of my AWS account to access my Amazon EKS resources

You can create a role that users in other accounts or people outside of your organization can use to access your resources. You can specify who is trusted to assume the role. For services that support resource-based policies or access control lists (ACLs), you can use those policies to grant people access to your resources.

To learn more, consult the following:

- To learn whether Amazon EKS supports these features, see [How Amazon EKS works with IAM \(p. 459\)](#).
- To learn how to provide access to your resources across AWS accounts that you own, see [Providing access to an IAM user in another AWS account that you own](#) in the *IAM User Guide*.
- To learn how to provide access to your resources to third-party AWS accounts, see [Providing access to AWS accounts owned by third parties](#) in the *IAM User Guide*.
- To learn how to provide access through identity federation, see [Providing access to externally authenticated users \(identity federation\)](#) in the *IAM User Guide*.
- To learn the difference between using roles and resource-based policies for cross-account access, see [How IAM roles differ from resource-based policies](#) in the *IAM User Guide*.

## Pod containers receive the following error: An error occurred (`SignatureDoesNotMatch`) when calling the `GetCallerIdentity` operation: Credential should be scoped to a valid region.

Your containers receive this error if your application is explicitly making requests to the AWS STS global endpoint (`https://sts.amazonaws`) and your Kubernetes service account is configured to use a regional endpoint. You can resolve the issue with one of the following options:

- Update your application code to remove explicit calls to the AWS STS global endpoint.
- Update your application code to make explicit calls to regional endpoints such as `https://sts.us-west-2.amazonaws.com`. Your application should have redundancy built in to pick a different AWS Region in the event of a failure of the service in the AWS Region. For more information, see [Managing AWS STS in an AWS Region](#) in the *IAM User Guide*.
- Configure your service accounts to use the global endpoint. All versions earlier than 1.22 use the global endpoint by default, but version 1.22 and later clusters use the regional endpoint by default. If your code makes explicit calls to the global endpoint on a 1.22 or later cluster, you'll receive the error if you're using the default endpoint type. For more information, see [Configure the AWS Security Token Service endpoint for a service account \(p. 452\)](#).

# Troubleshooting issues in Amazon EKS Connector

This topic covers some of the common errors that you might encounter while using the Amazon EKS Connector, including instructions on how to resolve them, workarounds, and frequently asked questions.

## Common issues

This section describes how you can troubleshoot some functional issues that you might encounter while using Amazon EKS Connector. It provides solutions and workarounds for these issues.

### Console error: the cluster is stuck in the Pending state

If, after you registered the cluster, the cluster gets stuck in the Pending state on the Amazon EKS console, it might be because Amazon EKS Connector didn't successfully connect the cluster to AWS yet. For a registered cluster, the Pending state means that the connection wasn't already successfully established. To resolve this issue, make sure that you have applied the manifest to the target Kubernetes cluster. If you did apply it to the cluster but the cluster is still in the Pending state, then most likely Amazon EKS Connector might be unhealthy in your cluster. To troubleshoot this issue, see the section that's named [Amazon EKS connector pods are crash looping \(p. 557\)](#) in this topic.

## Console error: User “system:serviceaccount:eks-connector:eks-connector” can't impersonate resource “users” in API group “” at cluster scope

Amazon EKS Connector uses Kubernetes [user impersonation](#) to act on behalf of users from the AWS Management Console. For each IAM identity that accesses the Kubernetes API from the AWS eks-connector service account, they must be granted permission to impersonate the corresponding Kubernetes user with an IAM ARN as their user name. In the following examples, the IAM ARN is mapped to a Kubernetes user.

- IAM user *john* from AWS account *111122223333* is mapped to a Kubernetes user.

```
arn:aws:iam::111122223333:user/john
```

- IAM role *admin* from AWS account *111122223333* is mapped to a Kubernetes user:

```
arn:aws:iam::111122223333:role/admin
```

The result is an IAM role ARN, instead of the STS session ARN.

For instructions on how to configure the `ClusterRole` and `ClusterRoleBinding` to grant eks-connector service account privilege to impersonate the mapped user, see [Granting access to a user to view Kubernetes resources on a cluster \(p. 567\)](#). Make sure that, in the template, `%IAM_ARN%` is replaced with IAM ARN of the AWS Management Console user.

## Console error: ... is forbidden: User ... cannot list resource “... in API group” at the cluster scope

Consider the following problem. The Amazon EKS Connector has successfully impersonated the requesting AWS Management Console user in the target Kubernetes cluster. However, the impersonated user doesn't have RBAC permission on Kubernetes API operations.

To resolve this issue, as the cluster administrator, you must grant the appropriate level of RBAC privileges to individual Kubernetes users. For more information and examples, see [Granting access to a user to view Kubernetes resources on a cluster \(p. 567\)](#).

## Console error: Amazon EKS can't communicate with your Kubernetes cluster API server. The cluster must be in an ACTIVE state for successful connection. Try again in few minutes.

If the Amazon EKS service can't communicate with Amazon EKS connector in the target cluster, it might be because of one of the following reasons:

- Amazon EKS Connector in the target cluster is unhealthy.
- Poor connectivity or an interrupted connection between the target cluster and the AWS Region.

To resolve this, check the [Inspect logs of Amazon EKS Connector \(p. 560\)](#). If you don't see an error for the Amazon EKS connector, retry the connection after a few minutes. If you regularly experience high latency or intermittent connectivity for the target cluster, consider re-registering the cluster to an AWS Region that's located closer to you.

## Amazon EKS connector pods are crash looping

There are many reasons that can cause an EKS connector pod to enter the CrashLoopBackOff status. This issue likely involves the connector-init container. In the following example, it was the connector-init container that entered the CrashLoopBackOff status.

```
kubectl get pods -n eks-connector
NAME READY STATUS RESTARTS AGE
eks-connector-0 0/2 Init:CrashLoopBackOff 1 7s
```

To troubleshoot this issue, [Inspect logs of Amazon EKS Connector \(p. 560\)](#).

### failed to initiate eks-connector: InvalidActivation

When you start Amazon EKS Connector for the first time, it registers an `activationId` and `activationCode` with Amazon Web Services. The registration might fail, which can cause the `connector-init` container to crash.

```
F1116 20:30:47.261469 1 init.go:43] failed to initiate eks-connector:
InvalidActivation:
```

To troubleshoot this issue, consider the following causes and recommended fixes:

- Registration might have failed because the `activationId` and `activationCode` aren't in your manifest file. If this is the case, make sure that they are the correct values that were returned from the `RegisterCluster` API operation, and that the `activationCode` is in the manifest file. The `activationCode` is added to Kubernetes secrets, so it must be base64 encoded. For more information, see [Step 1: Registering the cluster \(p. 564\)](#).
- Registration might have failed because your activation expired. This is because, for security reasons, you must activate the EKS connector within 3 days after registering the cluster. To resolve this issue, make sure that the EKS connector manifest is applied to the target Kubernetes cluster before the expiry date and time. To confirm your activation expiry date, call the `DescribeCluster` API operation. In the following example response, the expiry date and time is recorded as `2021-11-12T22:28:51.101000-08:00`.

```
aws eks describe-cluster --name my-cluster
{
 "cluster": {
 "name": "my-cluster",
 "arn": "arn:aws:eks:us-east-1:123456789012:cluster/my-cluster",
 "createdAt": "2021-11-09T22:28:51.449000-08:00",
 "status": "FAILED",
 "tags": {},
 "connectorConfig": {
 "activationId": "00000000-0000-0000-000000000000",
 "activationExpiry": "2021-11-12T22:28:51.101000-08:00",
 "provider": "OTHER",
 "roleArn": "arn:aws:iam::123456789012:role/my-connector-role"
 }
 }
}
```

If the `activationExpiry` passed, deregister the cluster and register it again. Do this generates a new activation.

### Cluster worker node is missing outbound connectivity

To work properly, EKS connector requires outbound connectivity to several AWS endpoints. You can't connect a private cluster without outbound connectivity to a target AWS Region. To resolve this issue, you must add the necessary outbound connectivity. For information about connector requirements, see [Amazon EKS Connector considerations \(p. 563\)](#).

## Amazon EKS connector pods are in ImagePullBackOff state

If you run the `getpods` command and pods are in the `ImagePullBackOff` state, they can't work properly. If EKS Connector pod is in `ImagePullBackOff` state, they can't work properly. See the example below:

```
kubectl get pods -n eks-connector
NAME READY STATUS RESTARTS AGE
eks-connector-0 0/2 Init:ImagePullBackOff 0 4s
```

The default Amazon EKS Connector manifest file references images from [Amazon ECR Public Registry](#). It is possible that the target Kubernetes cluster cannot pull images from Amazon ECR Public. Either resolve the Amazon ECR Public image pull issue, or consider mirroring the images in the private container registry of your choice.

## Frequently asked questions

### **Q: How does the underlying technology behind Amazon EKS Connector work?**

A: The Amazon EKS Connector is based on the System Manager (SSM) Agent. The EKS Connector runs as a StatefulSet on your Kubernetes cluster. It establishes a connection and proxies the communication between the API server of your cluster and Amazon Web Services. It does this to display cluster data in the Amazon EKS console until you disconnect the cluster from AWS. SSM agent is an open source project. For more information about this project, see the [GitHub project page](#).

### **Q: I have an on-premises Kubernetes cluster that I want to connect. Do I need to open firewall ports to connect it?**

A: No, you don't need to open any firewall ports. The Kubernetes cluster only requires outbound connection to AWS Regions. AWS services never access resources in your on-premises network. The Amazon EKS Connector runs on your cluster and initiates the connection to AWS. When the cluster registration completes, AWS only issues commands to the Amazon EKS Connector after you start an action from the Amazon EKS console that requires information from the Kubernetes API server on your cluster.

### **Q: What data is sent from my cluster to AWS by the Amazon EKS Connector?**

A: The Amazon EKS Connector sends technical information that's necessary for your cluster to be registered on AWS. It also sends cluster and workload metadata for the Amazon EKS console features that customers request. Amazon EKS Connector only gathers or sends this data if you start an action from the Amazon EKS console or the Amazon EKS API that necessitates the data to be sent to AWS. Other than the Kubernetes version number, AWS doesn't store any data by default. It stores them only if you authorize it.

### **Q: Can I connect a cluster outside of an AWS Region?**

A: Yes, you can connect a cluster from any location to Amazon EKS. Moreover, your EKS service can be located in any AWS public commercial AWS Region. This works with a valid network connection from your cluster to the target AWS Region. We recommend that you pick an AWS Region, which is the closest to your cluster location for UI performance optimization. For example, if you have a cluster running in Tokyo, connect your cluster to the AWS Region in Tokyo (that is, the `ap-northeast-1` AWS Region) for low latency. You can connect a cluster from any location to the Amazon EKS in any of the public commercial AWS Regions, except the China or GovCloud AWS Regions.

## Basic troubleshooting

This section describes steps to diagnose the issue if it's unclear.

## Check Amazon EKS Connector status

Check the Amazon EKS Connector status by using the following commands.

```
kubectl get pods -n eks-connector
```

## Inspect logs of Amazon EKS Connector

The Amazon EKS Connector Pod consists of three containers. You can inspect the logs of all three of them.

- connector-init
- connector-proxy
- connector-agent

To retrieve full logs for all of these containers so that you can inspect them, run the following commands.

- ```
# Retrieve log for connector-init container
kubectl logs eks-connector-0 --container connector-init -n eks-connector
kubectl logs eks-connector-1 --container connector-init -n eks-connector
```
- ```
Retrieve log for connector-proxy container
kubectl logs eks-connector-0 --container connector-proxy -n eks-connector
kubectl logs eks-connector-1 --container connector-proxy -n eks-connector
```
- ```
# Retrieve log for connector-agent container
kubectl exec eks-connector-0 --container connector-agent -n eks-connector -- cat /var/
log/amazon/ssm/amazon-ssm-agent.log
kubectl exec eks-connector-1 --container connector-agent -n eks-connector -- cat /var/
log/amazon/ssm/amazon-ssm-agent.log
```

Get the effective cluster name

Amazon EKS clusters are uniquely identified by `clusterName` within a single AWS account and AWS Region. If you have multiple connected clusters in Amazon EKS, you can confirm which Amazon EKS cluster that the current Kubernetes cluster is registered. To do this, enter the following to find out the `clusterName` of the current cluster.

```
kubectl exec eks-connector-0 --container connector-agent -n eks-connector \
-- cat /var/log/amazon/ssm/amazon-ssm-agent.log \
| grep -m1 -oE "eks_c:[a-zA-Z0-9_-]+" \
| sed -E "s/^.*eks_c:([a-zA-Z0-9_-]+)_([a-zA-Z0-9]+.*$)/\1/""
kubectl exec eks-connector-1 --container connector-agent -n eks-connector \
-- cat /var/log/amazon/ssm/amazon-ssm-agent.log \
| grep -m1 -oE "eks_c:[a-zA-Z0-9_-]+" \
| sed -E "s/^.*eks_c:([a-zA-Z0-9_-]+)_([a-zA-Z0-9]+.*$)/\1/"
```

Miscellaneous commands

The following commands are useful to retrieve information that you need to troubleshoot issues.

- Use the following command to gather images that's used by pods in Amazon EKS Connector.

```
kubectl get pods -n eks-connector -o jsonpath=".items[*].spec.containers[*].image" \
```

```
| tr -s '[:space:]' '\n'
```

- Use the following command to gather worker node names that Amazon EKS Connector is running on.

```
kubectl get pods -n eks-connector -o jsonpath=".items[*].spec.nodeName" \  
| tr -s '[:space:]' '\n'
```

- Run the following command to get your Kubernetes client and server versions.

```
kubectl version
```

- Run the following command to get information about your worker nodes.

```
kubectl get nodes -o wide --show-labels
```

Troubleshooting issues in ADOT Amazon EKS add-on

This topic covers some of the common errors that you might encounter while using the ADOT Amazon EKS add-on, including instructions on how to resolve them, workarounds, and frequently asked questions.

Common issues

This section describes functional issues that you might encounter while using ADOT Amazon EKS add-on and troubleshooting steps.

`error: "code": "K8sResourceNotFound", "message": "cert-manager is not installed on this cluster. During preview, you are required to have previously installed cert-manager."`

Ensure that the [cert-manager](#) version installed is supported by the ADOT Amazon EKS add-on and under version 1.6.0.

`error: "code": "AccessDenied", "message":
"roles.rbac.authorization.k8s.io \"opentelemetry-operator-leader-election-role\" is forbidden: User
\"eks:addon-manager\" cannot patch resource \"roles\" in
API group \"rbac.authorization.k8s.io\" in the namespace
\"opentelemetry-operator-system\""`

You do not have permission to install the ADOT for Amazon EKS add-on. See [Install the AWS Distro for OpenTelemetry \(ADOT\) Operator \(p. 519\)](#). If you have deleted the add-on and are now reinstalling, ensure that you have [applied permissions \(p. 517\)](#).

Create add-on error: "status": "CREATE_FAILED"

- Possibly a conflict. You can overwrite conflicts by adding the flag: `--resolve-conflicts=OVERWRITE` and run the create command again.
- An unsupported architecture, such as ARM64. Consult your logs to determine if this is the case.

Delete add-on error: "status": "DELETE_FAILED"

You can remove the ADOT Operator from being managed by Amazon EKS add-ons by adding the --preserve flag to your delete command.

Amazon EKS Connector

You can use Amazon EKS Connector to register and connect any conformant Kubernetes cluster to AWS and visualize it in the Amazon EKS console. After a cluster is connected, you can see the status, configuration, and workloads for that cluster in the Amazon EKS console. You can use this feature to view connected clusters in Amazon EKS console, but you can't manage them. The Amazon EKS Connector can connect the following types of Kubernetes clusters to Amazon EKS. The Amazon EKS Connector is also an [open source project on Github](#). For additional technical content, including frequently asked questions and troubleshooting, see [Troubleshooting issues in Amazon EKS Connector \(p. 556\)](#)

- On-premises Kubernetes clusters
- Self-managed clusters that are running on Amazon EC2
- Managed clusters from other cloud providers

Amazon EKS Connector considerations

Before you use Amazon EKS Connector, understand the following:

- You must have administrative privileges to the Kubernetes cluster to connect the cluster to Amazon EKS.
- The Kubernetes cluster must have Linux 64-bit (x86) worker nodes present before connecting. ARM worker nodes aren't supported.
- You must have worker nodes in your Kubernetes cluster that have outbound access to the `ssm.` and `ssmmessages.` Systems Manager endpoints. For more information, see [Systems Manager endpoints](#) in the *AWS General Reference*.
- By default, you can connect up to 10 clusters in a Region. You can request an increase through the [service quota console](#). See [Requesting a quota increase](#) for more information.
- Only the Amazon EKS `RegisterCluster`, `ListClusters`, `DescribeCluster`, and `DeregisterCluster` APIs are supported for external Kubernetes clusters.
- You must have the following permissions to register a cluster:
 - `eks:RegisterCluster`
 - `ssm>CreateActivation`
 - `ssm>DeleteActivation`
 - `iam:PassRole`
- You must have the following permissions to deregister a cluster:
 - `eks:DeregisterCluster`
 - `ssm>DeleteActivation`
 - `ssm:DeregisterManagedInstance`

Required IAM roles for Amazon EKS Connector

Using the Amazon EKS Connector requires the following two IAM roles:

- The [Amazon EKS Connector](#) service-linked role is created when you register the cluster.
- The Amazon EKS Connector agent IAM role must be created manually. See [Amazon EKS connector IAM role \(p. 480\)](#) for details.

To enable cluster and workload view permission for another user, you must apply the `eks-connector` and Amazon EKS Connector cluster roles to your cluster. Follow the steps in [Granting access to a user to view Kubernetes resources on a cluster \(p. 567\)](#).

Connecting a cluster

Step 1: Registering the cluster

You can connect an external Kubernetes cluster to Amazon EKS with AWS CLI and the AWS Management Console. This process involves two steps: registering the cluster with Amazon EKS and applying a YAML manifest file to enable connectivity. To allow another user to view the cluster, follow the instructions in [Granting access to a user to view Kubernetes resources on a cluster \(p. 567\)](#).

You must have the following permissions to register a cluster:

- `eks:RegisterCluster`
- `ssm>CreateActivation`
- `ssm>DeleteActivation`
- `iam:PassRole`

`eksctl`

Prerequisites

- `eksctl` v0.68 or above must be installed. To install or upgrade it, see [Getting started with eksctl](#).
- The Amazon EKS Connector agent IAM role was created. For more information, see [Connector IAM role](#).

To register your cluster with `eksctl`

1. Register the cluster by providing a name, provider, and region.

```
eksctl register cluster --name <my-first-registered-cluster> --provider <provider>
--region <region>
```

This creates two files on your local drive: `my-first-registered-cluster.yaml` and `eks-connector-binding.yaml`. These two files must be applied to the external cluster within three days, or the registration expires.

2. In the cluster's native environment, apply the `eks-connector-binding.yaml` file:

```
kubectl apply -f eks-connector-binding.yaml
```

AWS CLI

Prerequisites

- AWS CLI must be installed. To install or upgrade it, see [Installing the AWS CLI](#).
- Ensure the Amazon EKS Connector agent role was created. .

To register your cluster with the AWS CLI

1. For the Connector configuration, specify your Amazon EKS Connector agent IAM role. For more information, see [Required IAM roles for Amazon EKS Connector \(p. 563\)](#).

```
aws eks register-cluster \
    --name my-first-registered-cluster \
    --connector-config roleArn=arn:aws:iam::111122223333:role/
AmazonEKSConnectorAgentRole,provider="OTHER" \
    --region AWS_REGION
```

Output:

```
{
    "cluster": {
        "name": "my-first-registered-cluster",
        "arn": "arn:aws:eks:region:111122223333:cluster/my-first-registered-
cluster",
        "createdAt": 1627669203.531,
        "ConnectorConfig": {
            "activationId": "xxxxxxxxxACTIVATION_IDxxxxxxxxx",
            "activationCode": "xxxxxxxxxACTIVATION_CODExxxxxxxxx",
            "activationExpiry": 1627672543.0,
            "provider": "OTHER",
            "roleArn": "arn:aws:iam::111122223333:role/AmazonEKSConnectorAgentRole"
        },
        "status": "CREATING"
    }
}
```

You use the region, activationId, and activationCode values in a later step.

2. Download the Amazon EKS Connector YAML file.

```
curl -o eks-connector.yaml https://amazon-eks.s3.us-west-2.amazonaws.com/eks-
connector/manifests/eks-connector/latest/eks-connector.yaml
```

3. Edit the Amazon EKS Connector YAML file to replace all references of %AWS_REGION%, %EKS_ACTIVATION_ID%, %EKS_ACTIVATION_CODE% with the region, activationId, and activationCode from the output of your registration command.

The following example command can replace these values.

```
sed -i "s-%AWS_REGION%-$AWS_REGION-g; s-%EKS_ACTIVATION_ID%-$EKS_ACTIVATION_ID-g;
s-%EKS_ACTIVATION_CODE%-$(echo -n $EKS_ACTIVATION_CODE | base64)-g" eks-
connector.yaml
```

Important

Ensure that your activation code is in the base64 format.

AWS Management Console

Prerequisites

- Ensure the Amazon EKS Connector agent role was created. .

To register your Kubernetes cluster with the console.

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

2. Choose **Add cluster** and select **Register** to bring up the configuration page.
3. On the **Configure cluster** section, fill in the following fields:
 - **Name** – A unique name for your cluster.
 - **Provider** – Choose to display the dropdown list of Kubernetes cluster providers. If you don't know the specific provider, select Other.
 - **EKS Connector role** – Select the role to use for connecting the cluster.
4. Select **Register cluster**.
5. The Cluster overview page displays. Choose **Download YAML file** to download the manifest file to your local drive.

Important

- This is your only opportunity to download this file. Don't navigate away from this page, as the link will not be accessible and you must deregister the cluster and start the steps from the beginning.
- The manifest file can be used only once for the registered cluster. If you delete resources from the Kubernetes cluster, you must re-register the cluster and obtain a new manifest file.

Continue to the next step to apply the manifest file to your Kubernetes cluster.

Step 2: Applying the manifest file

Complete the connection by applying the Amazon EKS Connector manifest file to your Kubernetes cluster. To do this, you must use the AWS CLI or eksctl for the registration methods described previously. If the manifest is applied within three days, the Amazon EKS Connector registration expires. If the cluster connection expires, the cluster must be deregistered before connecting the cluster again.

1. In the native environment of the cluster, you can apply the updated manifest file by running the following command:

```
kubectl apply -f eks-connector.yaml
```

2. After the Amazon EKS Connector manifest and role binding YAML files are applied to your Kubernetes cluster, confirm that the cluster is now connected.

```
aws eks describe-cluster \
--name "my-first-registered-cluster" \
--region AWS_REGION
```

The output should include status=ACTIVE.

3. You can now add Tags to your cluster (optional). See [Tagging your Amazon EKS resources for more information](#).

To grant additional IAM users access to the Amazon EKS console to view Kubernetes resources in a connected cluster, see [Granting access to a user to view Kubernetes resources on a cluster \(p. 567\)](#).

Granting access to a user to view Kubernetes resources on a cluster

Grant additional IAM users access to the Amazon EKS console to view information about Kubernetes resources running on your connected cluster.

Prerequisites

The IAM user or role that you use to access the AWS Management Console must meet the following requirements.

- It has the `eks:AccessKubernetesApi` permission.
- The Amazon EKS Connector Service account can impersonate the IAM or role in the cluster. This allows the eks-connector to map the IAM user or role to a Kubernetes user.

To create and apply the Amazon EKS Connector cluster role

1. Download the eks-connector cluster role template.

```
curl -o eks-connector-clusterrole.yaml https://s3.us-west-2.amazonaws.com/amazon-eks/eks-connector/manifests/eks-connector-console-roles/eks-connector-clusterrole.yaml
```

2. Edit the cluster role template YAML file. Replace references of `%IAM_ARN%` with the Amazon Resource Name (ARN) of your IAM user or role.
3. Apply the Amazon EKS Connector cluster role YAML to your Kubernetes cluster.

```
kubectl apply -f eks-connector-clusterrole.yaml
```

For an IAM user or role to view Kubernetes resources in Amazon EKS console, the user or role must be associated with a Kubernetes `role` or `clusterrole` with necessary permissions to read the resources. For more information, see [Using RBAC Authorization](#) in the Kubernetes documentation.

To configure an IAM user to access the connected cluster

1. You can download the example manifest file to create a `clusterrole` and `clusterrolebinding` or a `role` and `rolebinding`:
 - **View Kubernetes resources in all namespaces** – The `eks-connector-console-dashboard-full-access-clusterrole` cluster role gives access to all namespaces and resources that can be visualized in the console. You can change the name of the `role`, `clusterrole` and their corresponding binding before applying it to your cluster. Use the following command to download a sample file.

```
curl -o eks-connector-console-dashboard-full-access-group.yaml https://s3.us-west-2.amazonaws.com/amazon-eks/eks-connector/manifests/eks-connector-console-roles/eks-connector-console-dashboard-full-access-group.yaml
```

- **View Kubernetes resources in a specific namespace** – The namespace in this file is default, so if you want to specify a different namespace, edit the file before applying it to your cluster. Use the following command to download a sample file.

```
curl -o eks-connector-console-dashboard-restricted-access-group.yaml https://s3.us-west-2.amazonaws.com/amazon-eks/eks-connector/manifests/eks-connector-console-roles/eks-connector-console-dashboard-restricted-access-group.yaml
```

2. Edit the full access or restricted access YAML file to replace references of %IAM_ARN% with the Amazon Resource Name (ARN) of your IAM user or role.
3. Apply the full access or restricted access YAML files to your Kubernetes cluster. Replace the YAML file value with your own.

```
kubectl apply -f eks-connector-console-dashboard-full-access-group.yaml
```

To view Kubernetes resources in your connected cluster, see [View Kubernetes resources \(p. 506\)](#). Data for some resource types on the **Resources** tab isn't available for connected clusters.

Deregistering a cluster

If you are finished using a connected cluster, you can deregister it. After it's deregistered, the cluster is no longer visible in the Amazon EKS console.

You must have the following permissions to call the `deregisterCluster` API:

- `eks:DeregisterCluster`
- `ssm>DeleteActivation`
- `ssm:DeregisterManagedInstance`

`eksctl`

Prerequisites

- `eksctl` v0.68 or above must be installed. To install or upgrade it, see [Getting started with eksctl](#).
- Ensure the Amazon EKS Connector agent role was created. .

To deregister your cluster with `eksctl`

- For the Connector configuration, specify your Amazon EKS Connector agent IAM role. For more information, see [Required IAM roles for Amazon EKS Connector \(p. 563\)](#).

```
eksctl deregister cluster --name
```

AWS CLI

Prerequisites

- AWS CLI must be installed. To install or upgrade it, see [Installing the AWS CLI](#).
- Ensure the Amazon EKS Connector agent role was created. .

Deregister the connected cluster.

```
aws eks deregister-cluster \  
  --name "my-first-registered-cluster" \  
  --region AWS_REGION
```

AWS Management Console

1. Open the Amazon EKS console at <https://console.aws.amazon.com/eks/home#/clusters>.
2. Choose **Clusters**.
3. On the **Clusters** page, select the connected cluster and select **Deregister**.
4. Confirm that you want to deregister the cluster.

To clean up the resources on your Kubernetes cluster.

1. Delete the Amazon EKS Connector YAML file from your Kubernetes cluster.

```
kubectl delete -f eks-connector.yaml
```

2. If you created `clusterrole` or `clusterrolebinding` for an additional IAM user to access the cluster, ensure that you delete them from your Kubernetes cluster.

Related projects

These open-source projects extend the functionality of Kubernetes clusters running on or outside of AWS, including clusters managed by Amazon EKS.

Management tools

Related management tools for Amazon EKS and Kubernetes clusters.

eksctl

`eksctl` is a simple CLI tool for creating clusters on Amazon EKS.

- [Project URL](#)
- [Project documentation](#)
- AWS open source blog: [eksctl: Amazon EKS cluster with one command](#)

AWS controllers for Kubernetes

With AWS Controllers for Kubernetes, you can create and manage AWS resources directly from your Kubernetes cluster.

- [Project URL](#)
- AWS open source blog: [AWS service operator for Kubernetes now available](#)

Flux CD

Flux is a tool that you can use to manage your cluster configuration using Git. It uses an operator in the cluster to trigger deployments inside of Kubernetes. For more information about operators, see [Awesome Operators in the Wild](#) on GitHub.

- [Project URL](#)
- [Project documentation](#)

CDK for Kubernetes

With the CDK for Kubernetes (`cdk8s`), you can define Kubernetes apps and components using familiar programming languages. `cdk8s` apps synthesize into standard Kubernetes manifests, which can be applied to any Kubernetes cluster.

- [Project URL](#)
- [Project documentation](#)
- AWS containers blog: [Introducing cdk8s+: Intent-driven APIs for Kubernetes objects](#)

Networking

Related networking projects for Amazon EKS and Kubernetes clusters.

Amazon VPC CNI plugin for Kubernetes

Amazon EKS supports native VPC networking through the Amazon VPC CNI plugin for Kubernetes. The plugin assigns an IP address from your VPC to each pod.

- [Project URL](#)
- [Project documentation](#)

AWS Load Balancer Controller for Kubernetes

The AWS Load Balancer Controller helps manage AWS Elastic Load Balancers for a Kubernetes cluster. It satisfies Kubernetes Ingress resources by provisioning AWS Application Load Balancers. It satisfies Kubernetes Service resources by provisioning AWS Network Load Balancers.

- [Project URL](#)
- [Project documentation](#)

ExternalDNS

ExternalDNS synchronizes exposed Kubernetes services and ingresses with DNS providers including Amazon Route 53 and AWS Service Discovery.

- [Project URL](#)
- [Project documentation](#)

App Mesh Controller

The App Mesh Controller for Kubernetes helps to manage App Mesh for your cluster. With the controller, you can manage the service mesh using custom resources within your cluster. The controller manages the injection of networking proxy sidecars to pods to enable the mesh.

- [Project URL](#)
- [Project documentation](#)
- AWS blog: [Getting started with App Mesh and Amazon EKS](#)

Security

Related security projects for Amazon EKS and Kubernetes clusters.

AWS IAM authenticator

A tool to use AWS IAM credentials to authenticate to a Kubernetes cluster if you're not using the AWS CLI version 1.16.156 or higher. For more information, see [Installing aws-iam-authenticator \(p. 417\)](#).

- [Project URL](#)
- [Project documentation](#)
- AWS open source blog: [Deploying the AWS IAM authenticator to kops](#)

Machine learning

Related machine learning projects for Amazon EKS and Kubernetes clusters.

Kubeflow

A machine learning toolkit for Kubernetes.

- [Project URL](#)
- [Project documentation](#)
- AWS open source blog: [Kubeflow on Amazon EKS](#)

Auto Scaling

Related auto scaling projects for Amazon EKS and Kubernetes clusters.

Cluster autoscaler

Cluster Autoscaler is a tool that automatically adjusts the size of the Kubernetes cluster based on CPU and memory pressure.

- [Project URL](#)
- [Project documentation](#)
- Amazon EKS workshop: https://eksworkshop.com/scaling/deploy_ca/

Escalator

Escalator is a batch or job optimized horizontal autoscaler for Kubernetes.

- [Project URL](#)
- [Project documentation](#)

Monitoring

Related monitoring projects for Amazon EKS and Kubernetes clusters.

Prometheus

Prometheus is an open-source systems monitoring and alerting toolkit.

- [Project URL](#)
- [Project documentation](#)

- Amazon EKS workshop: https://eksworkshop.com/intermediate/240_monitoring/

Continuous integration / continuous deployment

Related CI/CD projects for Amazon EKS and Kubernetes clusters.

Jenkins X

CI/CD solution for modern cloud applications on Amazon EKS and Kubernetes clusters.

- [Project URL](#)
- [Project documentation](#)

Amazon EKS new features and roadmap

You can learn about new Amazon EKS features by scrolling to the What's New feed on the [What's New with AWS](#) page. You can also review the [roadmap](#) on GitHub, which lets you know about upcoming features and priorities so that you can plan how you want to use Amazon EKS in the future. You can provide direct feedback to us about the roadmap priorities.

Document history for Amazon EKS

The following table describes the major updates and new features for the Amazon EKS User Guide. We also update the documentation frequently to address the feedback that you send us.

update-history-change	update-history-description	update-history-date
View Kubernetes resources in the AWS Management Console	You can now view information about the Kubernetes resources deployed to your cluster using the AWS Management Console.	May 3, 2022
Observation page and ADOT add-on support	Added Observability page and AWS Distro for OpenTelemetry (ADOT).	April 21, 2022
Kubernetes version 1.22	Added Kubernetes version 1.22 support for new clusters and version upgrades.	April 4, 2022
Added Fargate pod patching details	When upgrading Fargate pods, Amazon EKS first tries to evict pods based on your pod disruption budgets. You can create event rules to react to failed evictions before the pods are deleted.	April 1, 2022
Full release: Add-on support for Amazon EBS CSI driver	You can now use the AWS Management Console, AWS CLI, and API to manage the Amazon EBS CSI driver.	March 31, 2022
AWS Outposts content update	Instructions to deploy an Amazon EKS cluster on AWS Outposts.	March 22, 2022
Windows containerd support	You can now select the containerd runtime for Windows nodes.	March 14, 2022
Added EKS Connector considerations to security documentation	Describes the shared responsibility model as it relates to connected clusters.	February 25, 2022
Assign IPv6 addresses to your pods and services	You can now create a 1.21 or later cluster that assigns IPv6 addresses to your pods and services.	January 6, 2022
Preview release: Add-on support for Amazon EBS CSI driver	You can now preview using the AWS Management Console, AWS CLI, and API to manage the Amazon EBS CSI driver.	December 9, 2021

Karpenter autoscaler support	You can now use the Karpenter open-source project to autoscale your nodes.	November 29, 2021
Fluent Bit Kubernetes filter support in Fargate logging	You can now use the Fluent Bit Kubernetes filter with Fargate logging.	November 10, 2021
Windows support available in the control plane	Windows support is now available in your control plane. You no longer need to enable it in your data plane.	November 9, 2021
Bottlerocket added as an AMI type for managed node groups	Previously, Bottlerocket was only available as a self-managed node option. Now it can be configured as a managed node group, reducing the effort that's required to meet node compliance requirements.	October 28, 2021
DL1 driver support	Custom Amazon Linux AMIs now support deep learning workloads for AWS Linux 2. This enablement allows a generic on-premises or cloud baseline configuration.	October 25, 2021
VT1 video support	Custom Amazon Linux AMIs now support VT1 for some distributions. This enablement advertises Xilinx U30 devices on your Amazon EKS cluster.	September 13, 2021
Amazon EKS Connector is now available	You can use Amazon EKS Connector to register and connect any conformant Kubernetes cluster to AWS and visualize it in the Amazon EKS console.	September 8, 2021
Amazon EKS Anywhere is now available	Amazon EKS Anywhere is a new deployment option for Amazon EKS that you can use to create and operate Kubernetes clusters on-premises.	September 8, 2021
Amazon FSx for NetApp ONTAP CSI driver	Added topic that summarizes the Amazon FSx for NetApp ONTAP CSI driver and gives links to other references.	September 2, 2021
Managed node groups now auto-calculates the Amazon EKS recommended maximum pods for nodes	Managed node groups now auto-calculate the Amazon EKS maximum pods for nodes that you deploy without a launch template, or with a launch template that you haven't specified an AMI ID in.	August 30, 2021

Remove Amazon EKS management of add-on settings without removing the Amazon EKS add-on software	You can now remove an Amazon EKS add-on without removing the add-on software from your cluster.	August 20, 2021
Create multi-homed pods using Multus	You can now add multiple network interfaces to a pod using Multus.	August 2, 2021
Add more IP addresses to your Linux Amazon EC2 nodes	You can now add significantly more IP addresses to your Linux Amazon EC2 nodes. This means that you can run a higher density of pods on each node.	July 27, 2021
Kubernetes version 1.21	Added Kubernetes version 1.21 support.	July 19, 2021
Containerd runtime bootstrap	The Amazon EKS optimized accelerated Amazon Linux Amazon Machine Image (AMI) now contains a bootstrap flag that you can use to enable the containerd runtime in Amazon EKS optimized and Bottlerocket AMIs. This flag is available in all supported Kubernetes versions of the AMI.	July 19, 2021
Added managed policies topic	A list of all Amazon EKS IAM managed policies and changes that were made to them since June 17, 2021.	June 17, 2021
Use security groups for pods with Fargate	You can now use security groups for pods with Fargate, in addition to using them with Amazon EC2 nodes.	June 1, 2021
Added CoreDNS and kubeProxy Amazon EKS add-ons	Amazon EKS can now help you manage the CoreDNS and kubeProxy Amazon EKS add-ons for your cluster.	May 19, 2021
Kubernetes version 1.20	Added Kubernetes version 1.20 support for new clusters and version upgrades.	May 18, 2021
AWS Load Balancer Controller 2.2.0 released	You can now use the AWS Load Balancer Controller to create Elastic Load Balancers using instance or IP targets.	May 14, 2021
Node taints for managed node groups	Amazon EKS now supports adding node taints to managed node groups.	May 11, 2021

Secrets encryption for existing clusters	Amazon EKS now supports adding secrets encryption to existing clusters.	February 26, 2021
Kubernetes version 1.19	Added Kubernetes version 1.19 support for new clusters and version upgrades.	February 16, 2021
Amazon EKS now supports OpenID Connect (OIDC) identity providers as a method to authenticate users to your cluster that's version 1.16 or later.	OIDC identity providers can be used with, or as an alternative to AWS Identity and Access Management (IAM).	February 12, 2021
View node and workload resources in the AWS Management Console	You can now view details about your managed, self-managed, and Fargate nodes and your deployed Kubernetes workloads in the AWS Management Console.	December 1, 2020
Deploy Spot Instance types in a managed node group	You can now deploy multiple Spot or On-Demand Instance types to a managed node group.	December 1, 2020
Amazon EKS can now manage specific add-ons for your cluster	You can manage add-ons yourself, or allow Amazon EKS control the launch and version of an add-on through the Amazon EKS API for clusters that are running Kubernetes version 1.18 with platform version eks.3 or later.	December 1, 2020
Share an ALB across multiple Ingresses	You can now share an AWS Application Load Balancer (ALB) across multiple Kubernetes Ingresses. In the past, you had to deploy a separate ALB for each Ingress.	October 23, 2020
NLB IP target support	You can now deploy a network load balancer (NLB) with IP targets. This means that you can use an NLB to load balance network traffic to Fargate pods and directly to pods that are running on Amazon EC2 nodes.	October 23, 2020
Kubernetes version 1.18	Added Kubernetes version 1.18 support for new clusters and version upgrades.	October 13, 2020
Specify a custom CIDR block for Kubernetes service IP address assignment.	You can now specify a custom CIDR block that Kubernetes assigns service IP addresses from.	September 29, 2020

Assign security groups to individual pods	You can now associate different security groups to some of the individual pods that are running on many Amazon EC2 instance types.	September 9, 2020
Deploy Bottlerocket on your nodes	You can now deploy nodes that are running Bottlerocket .	August 31, 2020
The ability to launch Arm nodes is generally available	You can now launch Arm nodes in managed and self-managed node groups.	August 17, 2020
Managed node group launch templates and custom AMI	You can now deploy a managed node group that uses an Amazon EC2 launch template. The launch template can specify a custom AMI, if you choose.	August 17, 2020
EFS support for AWS Fargate	You can now use Amazon EFS with AWS Fargate.	August 17, 2020
Amazon EKS platform version update	This is a new platform version with security fixes and enhancements. This includes UDP support for services of type <code>LoadBalancer</code> when using NLB with Kubernetes 1.15 or later. For more information, see the Allow UDP for AWS NLB issue on GitHub.	August 12, 2020
Amazon EKS AWS Region expansion (p. 575)	Amazon EKS is now available in the Africa (Cape Town) (<code>af-south-1</code>) and Europe (Milan) (<code>eu-south-1</code>) AWS Regions.	August 6, 2020
Fargate usage metrics	AWS Fargate provides CloudWatch usage metrics that provide visibility into your account's usage of Fargate On-Demand resources.	August 3, 2020
Kubernetes version 1.17	Added Kubernetes version 1.17 support for new clusters and version upgrades.	July 10, 2020
Create and manage App Mesh resources from within Kubernetes with the App Mesh controller for Kubernetes	You can create and manage App Mesh resources from within Kubernetes. The controller also automatically injects the Envoy proxy and init containers into pods that you deploy.	June 18, 2020
Amazon EKS now supports Amazon EC2 Inf1 nodes	You can add Amazon EC2 Inf1 nodes to your cluster.	June 4, 2020

Amazon EKS AWS Region expansion (p. 575)	Amazon EKS is now available in the AWS GovCloud (US-East) (<code>us-gov-east-1</code>) and AWS GovCloud (US-West) (<code>us-gov-west-1</code>) AWS Regions.	May 13, 2020
Kubernetes 1.12 is no longer supported on Amazon EKS	Kubernetes version 1.12 is no longer supported on Amazon EKS. Update any 1.12 clusters to version 1.13 or later to avoid service interruption.	May 12, 2020
Kubernetes version 1.16 (p. 575)	Added Kubernetes version 1.16 support for new clusters and version upgrades.	April 30, 2020
Added the <code>AWSServiceRoleForAmazonEKS</code> service-linked role	Added the <code>AWSServiceRoleForAmazonEKS</code> service-linked role.	April 16, 2020
Kubernetes version 1.15 (p. 575)	Added Kubernetes version 1.15 support for new clusters and version upgrades.	March 10, 2020
Amazon EKS AWS Region expansion (p. 575)	Amazon EKS is now available in the Beijing (<code>cn-north-1</code>) and Ningxia (<code>cn-northwest-1</code>) AWS Regions.	February 26, 2020
FSx for Lustre CSI driver	Added topic for installing the FSx for Lustre CSI Driver on Kubernetes 1.14 Amazon EKS clusters.	December 23, 2019
Restrict network access to the public access endpoint of a cluster	With this update, you can use Amazon EKS to restrict the CIDR ranges that can communicate to the public access endpoint of the Kubernetes API server.	December 20, 2019
Resolve the private access endpoint address for a cluster from outside of a VPC	With this update, you can use Amazon EKS to resolve the private access endpoint of the Kubernetes API server from outside of a VPC.	December 13, 2019
(Beta) Amazon EC2 A1 Amazon EC2 instance nodes	Launch Amazon EC2 A1 Amazon EC2 instance nodes that register with your Amazon EKS cluster.	December 4, 2019
Creating a cluster on AWS Outposts	Amazon EKS now supports creating clusters on AWS Outposts.	December 3, 2019
AWS Fargate on Amazon EKS	Amazon EKS Kubernetes clusters now support running pods on Fargate.	December 3, 2019

Amazon EKS Region expansion (p. 575)	Amazon EKS is now available in the Canada (Central) (ca-central-1) AWS Region.	November 21, 2019
Managed node groups	Amazon EKS managed node groups automate the provisioning and lifecycle management of nodes (Amazon EC2 instances) for Amazon EKS Kubernetes clusters.	November 18, 2019
Amazon EKS platform version update	New platform versions to address CVE-2019-11253 .	November 6, 2019
Kubernetes 1.11 is no longer supported on Amazon EKS	Kubernetes version 1.11 is no longer supported on Amazon EKS. Please update any 1.11 clusters to version 1.12 or higher to avoid service interruption.	November 4, 2019
Amazon EKS AWS Region expansion (p. 575)	Amazon EKS is now available in the South America (São Paulo) (sa-east-1) AWS Region.	October 16, 2019
Windows support	Amazon EKS clusters running Kubernetes version 1.14 now support Windows workloads.	October 7, 2019
Autoscaling	Added a chapter to cover some of the different types of Kubernetes autoscaling that are supported on Amazon EKS clusters.	September 30, 2019
Kubernetes Dashboard update	Updated topic for installing the Kubernetes Dashboard on Amazon EKS clusters to use the beta 2.0 version.	September 28, 2019
Amazon EFS CSI driver	Added topic for installing the Amazon EFS CSI Driver on Kubernetes 1.14 Amazon EKS clusters.	September 19, 2019
Amazon EC2 Systems Manager parameter for Amazon EKS optimized AMI ID	Added topic for retrieving the Amazon EKS optimized AMI ID using an Amazon EC2 Systems Manager parameter. The parameter eliminates the need for you to look up AMI IDs.	September 18, 2019
Amazon EKS resource tagging	You can manage the tagging of your Amazon EKS clusters.	September 16, 2019
Amazon EBS CSI driver	Added topic for installing the Amazon EBS CSI driver on Kubernetes 1.14 Amazon EKS clusters.	September 9, 2019

New Amazon EKS optimized AMI patched for CVE-2019-9512 and CVE-2019-9514	Amazon EKS has updated the Amazon EKS optimized AMI to address CVE-2019-9512 and CVE-2019-9514 .	September 6, 2019
Announcing deprecation of Kubernetes 1.11 in Amazon EKS	Amazon EKS discontinued support for Kubernetes version 1.11 on November 4, 2019.	September 4, 2019
Kubernetes version 1.14 (p. 575)	Added Kubernetes version 1.14 support for new clusters and version upgrades.	September 3, 2019
IAM roles for service accounts	With IAM roles for service accounts on Amazon EKS clusters, you can associate an IAM role with a Kubernetes service account. With this feature, you no longer need to provide extended permissions to the node IAM role. This way, pods on that node can call AWS APIs.	September 3, 2019
Amazon EKS AWS Region expansion (p. 575)	Amazon EKS is now available in the Middle East (Bahrain) (<code>me-south-1</code>) AWS Region.	August 29, 2019
Amazon EKS platform version update	New platform versions to address CVE-2019-9512 and CVE-2019-9514 .	August 28, 2019
Amazon EKS platform version update	New platform versions to address CVE-2019-11247 and CVE-2019-11249 .	August 5, 2019
Amazon EKS Region expansion (p. 575)	Amazon EKS is now available in the Asia Pacific (Hong Kong) (<code>ap-east-1</code>) AWS Region.	July 31, 2019
Kubernetes 1.10 no longer supported on Amazon EKS	Kubernetes version 1.10 is no longer supported on Amazon EKS. Update any 1.10 clusters to version 1.11 or higher to avoid service interruption.	July 30, 2019
Added topic on ALB ingress controller	The AWS ALB Ingress Controller for Kubernetes is a controller that causes an ALB to be created when ingress resources are created.	July 11, 2019
New Amazon EKS optimized AMI	Removing unnecessary <code>kubectl</code> binary from AMIs.	July 3, 2019
Kubernetes version 1.13 (p. 575)	Added Kubernetes version 1.13 support for new clusters and version upgrades.	June 18, 2019

New Amazon EKS optimized AMI patched for AWS-2019-005	Amazon EKS has updated the Amazon EKS optimized AMI to address the vulnerabilities that are described in AWS-2019-005 .	June 17, 2019
Announcing discontinuation of support of Kubernetes 1.10 in Amazon EKS	Amazon EKS stopped supporting Kubernetes version 1.10 on July 22, 2019.	May 21, 2019
Amazon EKS platform version update	New platform version for Kubernetes 1.11 and 1.10 clusters to support custom DNS names in the Kubelet certificate and improve etcd performance.	May 21, 2019
AWS CLIget-token command (p. 575)	The <code>aws eks get-token</code> command was added to the AWS CLI. You no longer need to install the AWS IAM Authenticator for Kubernetes to create client security tokens for cluster API server communication. Upgrade your AWS CLI installation to the latest version to use this new functionality. For more information, see Installing the AWS Command Line Interface in the AWS Command Line Interface User Guide .	May 10, 2019
Getting started with eksctl	This getting started guide describes how you can install all of the required resources to get started with Amazon EKS using <code>eksctl</code> . This is a simple command line utility for creating and managing Kubernetes clusters on Amazon EKS.	May 10, 2019
Amazon EKS platform version update	New platform version for Kubernetes 1.12 clusters to support custom DNS names in the Kubelet certificate and improve etcd performance. This fixes a bug that caused node Kubelet daemons to request a new certificate every few seconds.	May 8, 2019
Prometheus tutorial	Added topic for deploying Prometheus to your Amazon EKS cluster.	April 5, 2019

Amazon EKS control plane logging	With this update, you can get audit and diagnostic logs directly from the Amazon EKS control pane. You can use these CloudWatch logs in your account as reference for securing and running clusters.	April 4, 2019
Kubernetes version 1.12 (p. 575)	Added Kubernetes version 1.12 support for new clusters and version upgrades.	March 28, 2019
Added App Mesh getting started guide	Added documentation for getting started with App Mesh and Kubernetes.	March 27, 2019
Amazon EKS API server endpoint private access	Added documentation for disabling public access for your Amazon EKS cluster's Kubernetes API server endpoint.	March 19, 2019
Added topic for installing the Kubernetes Metrics Server	The Kubernetes Metrics Server is an aggregator of resource usage data in your cluster.	March 18, 2019
Added list of related open source projects	These open source projects extend the functionality of Kubernetes clusters running on AWS, including clusters that are managed by Amazon EKS.	March 15, 2019
Added topic for installing Helm locally	The <code>helm</code> package manager for Kubernetes helps you install and manage applications on your Kubernetes cluster. This topic shows how to install and run the <code>helm</code> and <code>tiller</code> binaries locally. That way, you can install and manage charts using the <code>helm</code> CLI on your local system.	March 11, 2019
Amazon EKS platform version update	New platform version that updates Amazon EKS Kubernetes 1.11 clusters to patch level 1.11.8 to address CVE-2019-1002100 .	March 8, 2019
Increased cluster limit	Amazon EKS has increased the number of clusters that you can create in an AWS Region from 3 to 50.	February 13, 2019
Amazon EKS AWS Region expansion (p. 575)	Amazon EKS is now available in the Europe (London) (<code>eu-west-2</code>), Europe (Paris) (<code>eu-west-3</code>), and Asia Pacific (Mumbai) (<code>ap-south-1</code>) AWS Regions.	February 13, 2019

New Amazon EKS optimized AMI patched for ALAS-2019-1156	Amazon EKS has updated the Amazon EKS optimized AMI to address the vulnerability that's described in ALAS-2019-1156 .	February 11, 2019
New Amazon EKS optimized AMI patched for ALAS2-2019-1141	Amazon EKS has updated the Amazon EKS optimized AMI to address the CVEs that are referenced in ALAS2-2019-1141 .	January 9, 2019
Amazon EKS AWS Region expansion (p. 575)	Amazon EKS is now available in the Asia Pacific (Seoul) (ap-northeast-2) AWS Region.	January 9, 2019
Amazon EKS region expansion (p. 575)	Amazon EKS is now available in the following additional AWS Regions: Europe (Frankfurt) (eu-central-1), Asia Pacific (Tokyo) (ap-northeast-1), Asia Pacific (Singapore) (ap-southeast-1), and Asia Pacific (Sydney) (ap-southeast-2).	December 19, 2018
Amazon EKS cluster updates	Added documentation for Amazon EKS cluster Kubernetes version updates and node replacement .	December 12, 2018
Amazon EKS AWS Region expansion (p. 575)	Amazon EKS is now available in the Europe (Stockholm) (eu-north-1) AWS Region.	December 11, 2018
Amazon EKS platform version update	New platform version updating Kubernetes to patch level 1.10.11 to address CVE-2018-1002105 .	December 4, 2018
Added version 1.0.0 support for the ALB ingress controller	The ALB ingress controller releases version 1.0.0 with formal support from AWS.	November 20, 2018
Added support for CNI network configuration	The Amazon VPC CNI plugin for Kubernetes version 1.2.1 now supports custom network configuration for secondary pod network interfaces.	October 16, 2018
Added support for MutatingAdmissionWebhook and ValidatingAdmissionWebhook	Amazon EKS platform version 1.10-eks.2 now supports MutatingAdmissionWebhook and ValidatingAdmissionWebhook admission controllers.	October 10, 2018
Added partner AMI information	Canonical has partnered with Amazon EKS to create node AMIs that you can use in your clusters.	October 3, 2018

Added instructions for AWS CLI update-kubeconfig command	Amazon EKS has added the update-kubeconfig to the AWS CLI to simplify the process of creating a kubeconfig file for accessing your cluster.	September 21, 2018
New Amazon EKS optimized AMIs	Amazon EKS has updated the Amazon EKS optimized AMIs (with and without GPU support) to provide various security fixes and AMI optimizations.	September 13, 2018
Amazon EKS AWS Region expansion (p. 575)	Amazon EKS is now available in the Europe (Ireland) (eu-west-1) Region.	September 5, 2018
Amazon EKS platform version update	New platform version with support for Kubernetes aggregation layer and the Horizontal Pod Autoscaler(HPA).	August 31, 2018
New Amazon EKS optimized AMIs and GPU support	Amazon EKS has updated the Amazon EKS optimized AMI to use a new AWS CloudFormation node template and bootstrap script. In addition, a new Amazon EKS optimized AMI with GPU support is available.	August 22, 2018
New Amazon EKS optimized AMI patched for ALAS2-2018-1058	Amazon EKS has updated the Amazon EKS optimized AMI to address the CVEs that are referenced in ALAS2-2018-1058 .	August 14, 2018
Amazon EKS optimized AMI build scripts	Amazon EKS has open-sourced the build scripts that are used to build the Amazon EKS optimized AMI. These build scripts are now available on GitHub.	July 10, 2018
Amazon EKS initial release (p. 575)	Initial documentation for service launch	June 5, 2018