Amazon EKS

User Guide

Amazon EKS: User Guide

Table of Contents

What is Amazon EKS?

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

- Runs and scales the Kubernetes control plane across multiple AWS Availability Zones to ensure high availability.
- Automatically scales control plane instances based on load, detects and replaces unhealthy control plane instances, and it provides automated version updates and patching for them.
- Is integrated with many AWS services to provide scalability and security for your applications, including the following capabilities:
  - Amazon ECR for container images
  - Elastic Load Balancing for load distribution
  - IAM for authentication
  - Amazon VPC for isolation
- Runs up-to-date versions of the open-source Kubernetes software, so you can use all of the existing plugins and tooling from the Kubernetes community. Applications that are running on Amazon EKS are fully compatible with applications running on any standard Kubernetes environment, no matter whether they're running in on-premises data centers or public clouds. This means that you can easily migrate any standard Kubernetes application to Amazon EKS without any code modification.

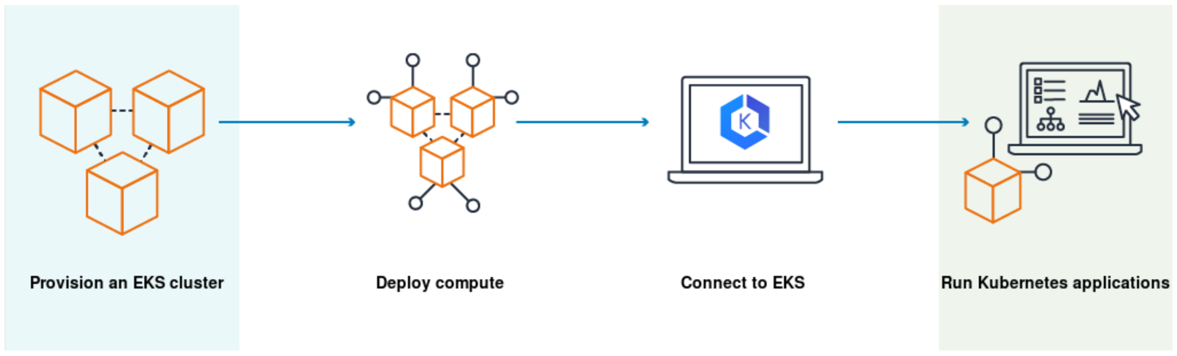
Amazon EKS control plane architecture

Amazon EKS runs a single tenant Kubernetes control plane for each cluster. The control plane infrastructure is not 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 a 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 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).

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.

Aligning with Amazon EKS for your self-managed Kubernetes clusters

Amazon EKS Distro is a distribution of the same open-source Kubernetes software and dependencies deployed by Amazon EKS in the cloud. With Amazon EKS Distro, you can create reliable and secure clusters wherever your applications are deployed. You can rely on the same versions of Kubernetes deployed by Amazon EKS, etcd, CoreDNS, upstream CNI, and CSI sidecars with the latest updates, and extended security patching support. Amazon EKS Distro follows the same Kubernetes version release cycle as Amazon EKS and is provided as an open-source project.

Note

The source code for the Amazon EKS Distro is available on GitHub. The latest documentation is available on the Amazon EKS Distro website. If you find any issues, you can report them with Amazon EKS Distro by connecting with us on GitHub. There you can open issues, provide feedback, and report bugs.

Getting started with Amazon EKS

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. 4) – 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. 9) – 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.

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. 9).

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 1.19 or later. For more information, see Installing `kubectl` (p. 303).
- **`eksctl`** – A command line tool for working with EKS clusters that automates many individual tasks. This guide requires that you use version 0.43.0 or later. For more information, see The `eksctl` command line utility (p. 308).
- **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

Create your 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. 17) and Amazon EKS nodes (p. 83).

You can create a cluster with one of the following node types. To learn more about each type, see Amazon EKS nodes (p. 83). 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.
- **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. 112) and Bottlerocket (p. 110) nodes to your cluster. A cluster must contain at least one Linux node, even if all your workloads are Windows.

Select the tab with the name of the node type that you'd like to create a cluster with.

Fargate – Linux

To create your cluster with Fargate Linux nodes

1. Create your Amazon EKS cluster with an the section called "Fargate profile" (p. 129) and the section called "Pod execution role" (p. 343) with the following command. Replace *my-cluster* with your own value. Though you can create a cluster in any Amazon EKS supported Region, in this tutorial, it's created in **US West (Oregon) us-west-2**.

```
eksctl create cluster \
--name my-cluster \
--region us-west-2 \
--fargate
```

The previous command creates a cluster and Fargate profile using primarily default settings. After creation is complete, view the stack named *eksctl-<my-cluster>-cluster* in the AWS CloudFormation console to review all resources that were created. For a list of all settings and options, enter `eksctl create cluster -h`. For documentation of all settings and options, see Creating and Managing Clusters in the `eksctl` documentation.

Output

You'll see several lines of output as the cluster and Fargate profile are created. Creation takes several minutes. The last line of output is similar to the following example line.

```
...
[#]  EKS cluster "my-cluster" in "us-west-2" region is ready
```

If nodes fail to join the cluster, then see Nodes fail to join cluster (p. 374) in the Troubleshooting guide.

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

2. Test your configuration.

```
kubectl get svc
```

Output

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

Managed nodes – Linux

To create your cluster with Amazon EC2 Linux managed nodes

- Create your cluster and Linux managed node group. Replace `my-cluster` with your own value and `us-west-2` with any Amazon EKS supported Region. If you're deploying to the Africa (Cape Town), Asia Pacific (Hong Kong), Europe (Milan), or Middle East (Bahrain) Regions, the endpoint must be enabled for your account. For more information, see Activating and deactivating AWS STS in an AWS Region. The endpoint is enabled by default for all other Regions.

  Replace *<your-key>* (including *<>*) with the name of an existing key pair. If you don't have a key pair, you can create one with the following command. If necessary, change `us-west-2` to the Region that you create your cluster in. Be sure to save the return output in a file on your local computer. For more information, see Creating or importing a key pair in the Amazon EC2 User Guide for Linux Instances.

  Though the key isn't required in this guide, you can only specify a key to use when you create the node group. Specifying the key allows you to SSH to nodes once they're created. To run the command, you need to have the AWS CLI version 2.1.26 or later or 1.19.7 or later. For more information, see Installing, updating, and uninstalling the AWS CLI in the AWS Command Line Interface User Guide.

  ```
  aws ec2 create-key-pair --region us-west-2 --key-name myKeyPair
  ```

  Create your cluster and nodes with the following command. Though you can create a cluster in any Amazon EKS supported Region, in this tutorial, it's created in **US West (Oregon) us-west-2**.

  ```
  eksctl create cluster \
  --name my-cluster \
  --region us-west-2 \
  --with-oidc \
  --ssh-access \
  --ssh-public-key <your-key> \
  --managed
  ```

  The previous command creates a cluster with nodes using primarily default Amazon EKS settings. To see all resources created, view the stack named `eksctl-<my-cluster>-cluster` in the AWS CloudFormation console at https://console.aws.amazon.com/cloudformation. For a list of all settings and options, enter `eksctl create cluster -h`. For documentation of all settings and options, see Creating and Managing Clusters in the `eksctl` documentation.

  Output

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

  ```
  ...
  [#]  EKS cluster "my-cluster" in "us-west-2" 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`.

Step 2: View resources

1. View your cluster nodes.

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

Amazon EC2 node output

```
NAME                                          STATUS   ROLES    AGE    VERSION
      INTERNAL-IP      EXTERNAL-IP     OS-IMAGE         KERNEL-VERSION
 CONTAINER-RUNTIME
ip-192-168-12-49.us-west-2.compute.internal    Ready    <none>   6m7s
 v1.18.9-eks-d1db3c    192.168.12.49    52.35.116.65    Amazon Linux 2
 4.14.209-160.335.amzn2.x86_64   docker://19.3.6
ip-192-168-72-129.us-west-2.compute.internal   Ready    <none>   6m4s
 v1.18.9-eks-d1db3c    192.168.72.129   44.242.140.21   Amazon Linux 2
 4.14.209-160.335.amzn2.x86_64   docker://19.3.6
```

Fargate node output

```
NAME                                                   STATUS   ROLES    AGE
 VERSION              INTERNAL-IP       EXTERNAL-IP   OS-IMAGE         KERNEL-VERSION
               CONTAINER-RUNTIME
fargate-ip-192-168-141-147.us-west-2.compute.internal   Ready    <none>
 8m3s     v1.18.8-eks-7c9bda    192.168.141.147   <none>        Amazon Linux 2
 4.14.209-160.335.amzn2.x86_64   containerd://1.3.2
fargate-ip-192-168-164-53.us-west-2.compute.internal    Ready    <none>
 7m30s    v1.18.8-eks-7c9bda    192.168.164.53    <none>        Amazon Linux 2
 4.14.209-160.335.amzn2.x86_64   containerd://1.3.2
```

For more information about what you see here, see the section called "View nodes" (p. 86).

2. View the workloads running on your cluster.

```
kubectl get pods --all-namespaces -o wide
```

Amazon EC2 output

```
NAMESPACE     NAME                       READY   STATUS    RESTARTS   AGE     IP
        NODE                                        NOMINATED NODE   READINESS GATES
kube-system   aws-node-6ctpm             1/1     Running   0          7m43s
 192.168.72.129   ip-192-168-72-129.us-west-2.compute.internal   <none>
 <none>
kube-system   aws-node-cbntg             1/1     Running   0          7m46s
 192.168.12.49    ip-192-168-12-49.us-west-2.compute.internal    <none>
 <none>
kube-system   coredns-559b5db75d-26t47   1/1     Running   0          14m
 192.168.78.81    ip-192-168-72-129.us-west-2.compute.internal   <none>
 <none>
kube-system   coredns-559b5db75d-9rvnk   1/1     Running   0          14m
 192.168.29.248   ip-192-168-12-49.us-west-2.compute.internal    <none>
 <none>
kube-system   kube-proxy-l8pbd           1/1     Running   0          7m46s
 192.168.12.49    ip-192-168-12-49.us-west-2.compute.internal    <none>
 <none>
kube-system   kube-proxy-zh85h           1/1     Running   0          7m43s
 192.168.72.129   ip-192-168-72-129.us-west-2.compute.internal   <none>
 <none>
```

Fargate output

```
NAMESPACE     NAME                           READY   STATUS    RESTARTS   AGE   IP
           NODE                                                    NOMINATED NODE
 READINESS GATES
kube-system   coredns-69dfb8f894-9z95l   1/1     Running   0          18m
 192.168.164.53    fargate-ip-192-168-164-53.us-west-2.compute.internal    <none>
     <none>
kube-system   coredns-69dfb8f894-c8v66   1/1     Running   0          18m
 192.168.141.147   fargate-ip-192-168-141-147.us-west-2.compute.internal   <none>
     <none>
```

For more information about what you see here, see the section called "View workloads" (p. 253).

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. If you want to do more with this cluster before you clean up, see the section called "Next steps" (p. 8).

Delete your cluster and nodes.

```
eksctl delete cluster --name my-cluster --region us-west-2
```

Next steps

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

- The IAM entity (user or role) that created 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`. If you want other users to have access to your cluster, then you must add them to the `aws-auth ConfigMap`. For more information, see Managing users or IAM roles for your cluster (p. 288).
- Restrict access to IMDS (p. 359) – If you plan to assign IAM roles to all of your Kubernetes service accounts so that pods only have the minimum permissions that they need, and no pods in the cluster require access to the Amazon EC2 instance metadata service (IMDS) for other reasons, such as retrieving the current Region, then we recommend blocking pod access to IMDS. For more information, see the section called "IAM roles for service accounts" (p. 345) and the section called "Restricting access to the instance profile" (p. 359).
- Cluster Autoscaler (p. 45) – Configure the Kubernetes Cluster Autoscaler to automatically adjust the number of nodes in your node groups.
- Deploy a sample Linux workload (p. 255) – Deploy a sample Linux application to test your cluster and Linux nodes.
- Cluster management (p. 303) – Learn how to use important tools for managing your cluster.

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. 4).

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 2.1.26 or later or 1.19.7 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 1.19 or later. For more information, see Installing `kubectl` (p. 303).
- **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

Create an Amazon EKS cluster.

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. 17) and Amazon EKS nodes (p. 83).

To create your cluster

1. Create an Amazon VPC with public and private subnets that meets Amazon EKS requirements.

```
aws cloudformation create-stack \
  --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
```

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 Region selected in the top right of your console is **Oregon**. If not, select the drop-down next to the Region name and select **US West (Oregon) us-west-2**. Though you can create a cluster in any Amazon EKS supported Region, in this tutorial, it's created in **US West (Oregon) us-west-2**.
4. Select **Create cluster**. If you don't see this option, in the **Create EKS cluster** box, enter a name for your cluster, such as `my-cluster`, and select **Next step**.
5. On the **Configure cluster** page enter a name for your cluster, such as *`my-cluster`* and select ***`myAmazonEKSClusterRole`*** for **Cluster Service Role**. Leave the remaining settings at their default values and select **Next**.
6. On the **Specify networking** page, select ***`vpc-00x0000x000x0x000`*** **|** ***`my-eks-vpc-stack-VPC`*** from the **VPC** drop down list. Leave the remaining settings at their default values and select **Next**.
7. On the **Configure logging** page, select **Next**.
8. On the **Review and create** page, select **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. 374).

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. If necessary, replace *`us-west-2`* with the Region that you created your cluster in.

   ```
   aws eks update-kubeconfig \
     --region us-west-2 \
     --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. 375) 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 an IAM OpenID Connect (OIDC) provider

Create an IAM OpenID Connect (OIDC) provider for your cluster so that Kubernetes service accounts used by workloads can access AWS resources. You only need to complete this step one time for a cluster.

1. Select the **Configuration** tab.
2. In the **Details** section, copy the value for **OpenID Connect provider URL**.
3. Open the IAM console at https://console.aws.amazon.com/iam/.
4. In the navigation panel, choose **Identity Providers**.
5. Choose **Add Provider**.
6. For **Provider Type**, choose **OpenID Connect**.
7. For **Provider URL**, paste the OIDC provider URL for your cluster from step two, and then choose **Get thumbprint**.
8. For **Audience**, enter `sts.amazonaws.com` and choose **Add provider**.

Step 4: Create nodes

You can create a cluster with one of the following node types. To learn more about each type, see Amazon EKS nodes (p. 83). After your cluster is deployed, you can add other node types.

- **Fargate – Linux** – Select this type if you want to run Linux applications on AWS Fargate.
- **Managed nodes – Linux** – Select this type 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. 112) and Bottlerocket (p. 110) nodes to your cluster. A cluster must contain at least one Linux node, even if all your workloads are Windows.

Select the tab with the name of the node type that you'd like to create.

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 need to make calls to AWS APIs on your behalf to do things like 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*.

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

   b. Create a pod execution IAM role.

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

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

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

2. Open the Amazon EKS console at https://console.aws.amazon.com/eks/home#/clusters.
3. Choose the cluster to create a Fargate profile for and select the **Configuration** tab, then the **Compute** tab.
4. Under **Fargate profiles**, choose **Add Fargate profile**.

5. On the **Configure Fargate profile** page, enter the following information and choose **Next**.

   a. For **Name**, enter a unique name for your Fargate profile, such as *my-profile*.
   b. For **Pod execution role**, choose the ***myAmazonEKSFargatePodExecutionRole*** role that you created in step one.
   c. Select the **Subnets** dropdown and unselect any subnet with `Public` in its name. Only private subnets are supported for pods running on Fargate.

6. On the **Configure pods selection** page, enter the following information and choose **Next**.

   - For **Namespace**, enter `default`.

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

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 an IAM role for the Amazon VPC CNI plugin and attach the required Amazon EKS IAM managed policy to it. The Amazon EKS Amazon VPC CNI plugin is installed on a cluster, by default. The plugin allows Kubernetes pods to have the same IP address inside the pod as they do on the VPC network.

   a. Copy the following contents to a file named *cni-role-trust-policy.json*. Replace *<111122223333>* (including *<>*) with your account ID and replace *<XXXXXXXXXX45D83924220DC4815XXXXX>* with the value after the last `/` of your **OpenID Connect provider URL** (p. 11).

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Federated": "arn:aws:iam::<111122223333>:oidc-provider/oidc.eks.us-
west-2.amazonaws.com/id/<XXXXXXXXXX45D83924220DC4815XXXXX>"
      },
      "Action": "sts:AssumeRoleWithWebIdentity",
      "Condition": {
        "StringEquals": {
          "oidc.eks.us-west-2.amazonaws.com/
id/<XXXXXXXXXX45D83924220DC4815XXXXX>:sub": "system:serviceaccount:kube-
system:aws-node"
        }
      }
    }
  ]
}
```

   b. Create an IAM role for the Amazon VPC CNI plugin.

```
aws iam create-role \
  --role-name myAmazonEKSCNIRole \
  --assume-role-policy-document file://"cni-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/AmazonEKS_CNI_Policy \
  --role-name myAmazonEKSCNIRole
```

2. Associate the Kubernetes service account used by the VPC CNI plugin to the IAM role. Replace *<111122223333>* (including <>) with your account ID.

```
aws eks update-addon \
  --cluster-name my-cluster \
  --addon-name vpc-cni \
  --service-account-role-arn arn:aws:iam::<111122223333>:role/myAmazonEKSCNIRole
```

3. 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 Amazon EKS 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
```

4. Open the Amazon EKS console at https://console.aws.amazon.com/eks/home#/clusters.
5. Choose the name of the cluster that you created in the section called “Step 1: Create cluster” (p. 9), such as ***my-cluster***.
6. Select the **Configuration** tab.
7. On the **Configuration** tab, select the **Compute** tab, and then choose **Add Node Group**.
8. On the **Configure node group** page, fill out the parameters accordingly, accept the remaining default values, and then choose **Next**.

   - **Name** – Enter a unique name for your managed node group, such as *my-nodegroup*.
   - **Node IAM role name** – Choose ***myAmazonEKSNodeRole***. In this getting started guide, this role must only be used for this node group and no other node groups.
9. On the **Set compute and scaling configuration** page, accept the default values and select **Next**.

10. On the **Specify networking** page, select an existing key pair to use for `SSH key pair` and then choose **Next**. If you don't have a key pair, you can create one with the following command. If necessary, change `us-west-2` to the Region that you created your cluster in. Be sure to save the return output in a file on your local computer. For more information, see Creating or importing a key pair in the Amazon EC2 User Guide for Linux Instances. Though the key isn't required in this guide, you can only specify a key to use when you create the node group. Specifying the key allows you to SSH to the node once it's created.

```
aws ec2 create-key-pair --region us-west-2 --key-name myKeyPair
```

11. On the **Review and create** page, review your managed node group configuration and choose **Create**.
12. 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 5: View resources

You can view your nodes and Kubernetes workloads.

To view your nodes

1. In the left pane, select **Clusters**, and then in the list of **Clusters**, select the name of the cluster that you created, such as ***my-cluster***.
2. On the **Overview** tab, you see the list of **Nodes** that were deployed for the cluster. You can select the name of a node to see more information about it. For more information about what you see here, see the section called “View nodes” (p. 86).
3. On the **Workloads** tab of the cluster, you see a list of the workloads that are deployed by default to an Amazon EKS cluster. You can select the name of a workload to see more information about it. For more information about what you see here, see the section called “View workloads” (p. 253).

Step 6: 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. If you want to do more with this cluster before you clean up, see the section called “Next steps” (p. 16).

To delete your cluster and nodes

1. 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, select **Clusters**, and then in the list of clusters, select the name of the cluster that you want to delete.
   c. Select the **Configuration** tab. On the **Compute** tab, select:

      - The node group that you created in a previous step and select **Delete**. Enter the name of the node group, and then select **Delete**.
      - The **Fargate Profile** that you created in a previous step and select **Delete**. Enter the name of the profile, and then select **Delete**.
2. 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**.
3. Delete the VPC AWS CloudFormation stack that you created in this guide.
   a. Open the AWS CloudFormation console at https://console.aws.amazon.com/cloudformation.
   b. Select the VPC stack to delete, and choose **Delete**.
   c. On the **Delete Stack** confirmation screen, 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, select **Roles**.
   c. Select the ***`myAmazonEKSClusterRole`*** from the list. Select **Delete role**, and then select **Yes, Delete**. Delete the ***`myAmazonEKSFargatePodExecutionRole`*** or ***`myAmazonEKSNodeRole`*** role that you created and the ***`myAmazonEKSCNIRole`*** role, if you created one.

Next steps

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

- The IAM entity (user or role) that created 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`. If you want other users to have access to your cluster, then you must add them to the `aws-auth` `ConfigMap`. For more information, see Managing users or IAM roles for your cluster (p. 288).
- Restrict access to IMDS (p. 359) – If you plan to assign IAM roles to all of your Kubernetes service accounts so that pods only have the minimum permissions that they need, and no pods in the cluster require access to the Amazon EC2 instance metadata service (IMDS) for other reasons, such as retrieving the current Region, then we recommend blocking pod access to IMDS. For more information, see the section called “IAM roles for service accounts” (p. 345) and the section called “Restricting access to the instance profile” (p. 359).
- Cluster Autoscaler (p. 45) – Configure the Kubernetes Cluster Autoscaler to automatically adjust the number of nodes in your node groups.
- Deploy a sample Linux workload (p. 255) – Deploy a sample Linux application to test your cluster and Linux nodes.
- Cluster management (p. 303) – Learn how to use important tools for managing your cluster.

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).

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.

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.

Important
When an Amazon EKS cluster is created, the IAM entity (user or role) that creates the cluster is added to the Kubernetes RBAC authorization table as the administrator (with `system:masters` permissions). Initially, only that IAM user can make calls to the Kubernetes API server using `kubectl`. For more information, see Managing users or IAM roles for your cluster (p. 288). 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.

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

eksctl

Prerequisite

`eksctl` version 0.43.0 or later installed. To install it or upgrade, see the section called “Installing `eksctl`” (p. 308).

Create a cluster with the Amazon EKS latest Kubernetes version in your default Region. Replace the *`<example-values>`* (including `<>`) with your own values. You can replace *`<1.19>`* with any supported version (p. 58).

```
eksctl create cluster \
 --name <my-cluster> \
 --version <1.19> \
 --with-oidc \
 --without-nodegroup
```

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
If you create a cluster using a config file with the `secretsEncryption` option, which requires an existing AWS Key Management Service key, and the key that you use is ever deleted, then there is no path to recovery for the cluster. If you enable envelope encryption, the Kubernetes secrets are encrypted using the customer master key (CMK) that you select. The CMK must be symmetric, created in the same Region as the cluster, and if the CMK was created in a different account, the user must have access to the CMK. For more information, see Allowing users in other accounts to use a CMK in the *AWS Key Management Service Developer Guide*. Kubernetes secrets encryption with an AWS KMS CMK requires Kubernetes version 1.13 or later.
By default, the `create-key` command creates a symmetric key with 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 key policy for the principal that will be calling the `create-cluster` API. Amazon EKS does not support the key policy condition `kms:GrantIsForAWSResource`. Creating a cluster will not work if this action is in the 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
```

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 considerations (p. 208) and Amazon EKS security group considerations (p. 210). If you don't have a VPC, you can follow Creating a VPC for your Amazon EKS cluster (p. 204) to create one.
- An existing Amazon EKS cluster IAM role. If you don't have the role, you can follow Amazon EKS IAM roles (p. 331) to create one.

To create your cluster with the console

1. Open the Amazon EKS console at https://console.aws.amazon.com/eks/home#/clusters.
2. Choose **Create cluster**.
3. On the **Configure cluster** page, fill in the following fields:

- **Name** – A unique name for your cluster.
- **Kubernetes version** – The version of Kubernetes to use for your cluster.
- **Cluster service role** – Choose the Amazon EKS cluster role to allow the Kubernetes control plane to manage AWS resources on your behalf. For more information, see Amazon EKS cluster IAM role (p. 339).
- **Secrets encryption** – (Optional) Choose to enable envelope encryption of Kubernetes secrets using the AWS Key Management Service (AWS KMS). If you enable envelope encryption, the Kubernetes secrets are encrypted using the customer master key (CMK) that you select. The CMK must be symmetric, created in the same region as the cluster, and if the CMK was created in a different account, the user must have access to the CMK. For more information, see Allowing users in other accounts to use a CMK in the *AWS Key Management Service Developer Guide*.

  Kubernetes secrets encryption with an AWS KMS CMK requires Kubernetes version 1.13 or later. If no keys are listed, you must create one first. For more information, see Creating keys.

  Note
  By default, the `create-key` command creates a symmetric key with a key policy that gives the account's root user 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 key policy for the principal that will be calling the `create-cluster` API.
  Amazon EKS does not support the key policy condition `kms:GrantIsForAWSResource`. Creating a cluster will not work if this action is in the key policy statement.

  Warning
  Deletion of the CMK will permanently put the cluster in a degraded state. If any CMKs used for cluster creation are scheduled for deletion, verify that this is the intended action before deletion. Once the key is deleted, there is no path to recovery for the cluster.
- **Tags** – (Optional) Add any tags to your cluster. For more information, see Tagging your Amazon EKS resources (p. 318).

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

   - **VPC** – Select an existing VPC to use for your cluster. If none are listed, then you need to create one first. For more information, see Creating a VPC for your Amazon EKS cluster (p. 204).
   - **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 considerations (p. 208).

     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. 204).
     - 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. 204). This security group has **ControlPlaneSecurityGroup** in the drop-down 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.

- (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. 41).

6. If you selected Kubernetes version 1.17 or earlier on the previous page, skip to the next step. If you selected version 1.18, accept the defaults in the **Networking add-ons** section to install the latest version of the AWS VPC CNI (p. 214) Amazon EKS add-on. You can only use Amazon EKS add-ons with 1.18 clusters because Amazon EKS add-ons require the Server-side Apply Kubernetes feature, which wasn't available until Kubernetes 1.18. If you selected a different Kubernetes version for your cluster, then this option isn't shown.

   > **Important**
   > The AWS VPC CNI add-on is configured to use the IAM permissions assigned to the Amazon EKS node IAM role (p. 341). After the cluster is created, but before you deploy any Amazon EC2 nodes to your cluster, you must ensure that the `AmazonEKS_CNI_Policy` IAM policy 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 VPC CNI plugin to use IAM roles for service accounts (p. 217). Once your cluster and IAM role are created, you can update the add-on (p. 33) to use the IAM role that you create.

7. Select **Next**.

8. On the **Configure logging** page, you can optionally choose which log types that you want to enable. By default, each log type is **Disabled**. For more information, see Amazon EKS control plane logging (p. 55).
9. Select **Next**.
10. 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. 374).

    Cluster provisioning takes several minutes.
11. Follow the procedures in Create a `kubeconfig` for Amazon EKS (p. 295) to enable communication with your new cluster.
12. (Optional) To use Amazon EKS add-ons, or to enable individual Kubernetes workloads to have specific IAM permissions, you need to enable an OpenID Connect (OIDC) provider for your cluster. To configure an OIDC provider for your cluster, see the section called "Create OIDC provider" (p. 349). You only need to enable an OIDC provider for your cluster once. To learn more about Amazon EKS add-ons, see Configure an Amazon EKS add-on (p. 33). To learn more about assigning specific IAM permissions to your workloads, see Technical overview (p. 346).
13. (Optional) Before deploying nodes to your cluster, we recommend configuring the AWS VPC CNI plugin that was deployed with the cluster to use IAM roles for service accounts (p. 345). For more information, see the section called "Configure plugin for IAM account" (p. 217).

AWS CLI

Prerequisites

- An existing VPC and a dedicated security group that meet the requirements for an Amazon EKS cluster. For more information, see Cluster VPC considerations (p. 208) and Amazon EKS security group considerations (p. 210). If you don't have a VPC, you can follow Creating a VPC for your Amazon EKS cluster (p. 204) to create one.
- An existing Amazon EKS cluster IAM role. If you don't have the role, you can follow Amazon EKS IAM roles (p. 331) to create one.
- The AWS CLI version 2.1.26 or later or 1.19.7 or later installed. To install or upgrade, see Installing, updating, and uninstalling the AWS CLI in the AWS Command Line Interface User Guide. We recommend that you also configure the AWS CLI. For more information, see Quick configuration with aws configure in the AWS Command Line Interface User Guide.

To create your cluster with the AWS CLI

1. 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. 339) and the subnet and security group IDs for the VPC that you created in Creating a VPC for your Amazon EKS cluster (p. 204). Replace *`<my-cluster>`* (including `<>`) with your cluster name and *`<region-code>`* with a supported Region. You can replace *`<1.19>`* with any supported version (p. 58).

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.

```
aws eks create-cluster \
   --region <region-code> \
   --name <my-cluster> \
   --kubernetes-version <1.19> \
   --role-arn <arn:aws:iam::111122223333:role/eks-service-role-
AWSServiceRoleForAmazonEKS-EXAMPLEBKZRQR> \
   --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. 331).

Output:

```
{
    "cluster": {
        "name": "<my-cluster>",
        "arn": "arn:aws:eks:<region-code>:<111122223333>:cluster/<my-cluster>",
        "createdAt": <1527785885.159>,
        "version": "<1.19>",
        "roleArn": "arn:aws:iam::<111122223333>:role/eks-service-role-
AWSServiceRoleForAmazonEKS-<AFNL4H8HB71F>",
        "resourcesVpcConfig": {
            "subnetIds": [
                "<subnet-a9189fe2>",
                "<subnet-50432629>"
            ],
            "securityGroupIds": [
                "<sg-f5c54184>"
            ],
            "vpcId": "<vpc-a54041dc>",
            "endpointPublicAccess": true,
            "endpointPrivateAccess": false
        },
        "status": "CREATING",
        "certificateAuthority": {}
    }
}
```

Note
You might receive an error that one of the Availability Zones in your request doesn't have sufficient capacity to create an Amazon EKS cluster. If this happens, the error output contains the Availability Zones that can support a new cluster. Retry creating your cluster with at least two subnets that are located in the supported Availability Zones for your account. For more information, see Insufficient capacity (p. 374).

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

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

Note
By default, the `create-key` command creates a symmetric key with a key policy that gives the account's root user 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 key policy for the principal that will be calling the `create-cluster` API.
Amazon EKS does not support the key policy condition `kms:GrantIsForAWSResource`. Creating a cluster will not work if this action is in the 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 CMK. The CMK must be symmetric, created in the same Region as the cluster, and if the CMK was created in a different account, the user must have access to the CMK. For more information, see Allowing users in other accounts to use a CMK in the *AWS Key Management Service Developer Guide*. Kubernetes secrets encryption with an AWS KMS CMK requires Kubernetes version 1.13 or later.

Warning
Deletion of the CMK will permanently put the cluster in a degraded state. If any CMKs used for cluster creation are scheduled for deletion, verify that this is the intended action before deletion. Once the key is deleted, there is no path to recovery for the cluster.

2. 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 --region <region-code> describe-cluster --name <my-cluster> --query
 "cluster.status"
```

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

   a. Retrieve the `endpoint`.

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

   b. Retrieve the `certificateAuthority.data`.

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

4. Follow the procedures in Create a `kubeconfig` for Amazon EKS (p. 295) to enable communication with your new cluster.

5. (Optional) To use Amazon EKS add-ons, or to enable individual Kubernetes workloads to have specific IAM permissions, you need to enable an OpenID Connect (OIDC) provider for your cluster. To configure an OIDC provider for your cluster, see the section called "Create OIDC provider" (p. 349). You only need to enable an OIDC provider for your cluster once. To learn more about Amazon EKS add-ons, see Configure an Amazon EKS add-on (p. 33). To learn more about assigning specific IAM permissions to your workloads, see Technical overview (p. 346).

6. (Optional) Before deploying nodes to your cluster, we recommend configuring the AWS VPC CNI plugin that was deployed with the cluster to use IAM roles for service accounts (p. 345). For more information, see the section called "Configure plugin for IAM account" (p. 217).

Updating a cluster

You can update an existing cluster to a new Kubernetes version or configure managed add-ons for your cluster.

Updating an Amazon EKS cluster Kubernetes version

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

Important
We recommend that before updating to a new Kubernetes version that you review the information in Amazon EKS Kubernetes versions (p. 58) and 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 achieve 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 two to three 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 during cluster creation have been deleted, the cluster update process can fail.

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

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

Kubernetes version	1.19	1.18	1.17	1.16	1.15
Amazon VPC CNI plug-in	1.7.5	1.7.5	1.7.5	1.7.5	1.7.5
DNS (CoreDNS)	1.8.0	1.7.0	1.6.6	1.6.6	1.6.6

Kubernetes version	1.19	1.18	1.17	1.16	1.15
KubeProxy	1.19.6	1.18.8	1.17.9	1.16.13	1.15.11

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

Update an existing cluster

Update the cluster and Kubernetes add-ons.

To update an existing 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 following command.

   ```
   kubectl version --short
   ```

   - Get the Kubernetes version of your nodes with the following 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
   ```

   The Kubernetes 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 before you update your control plane to a new Kubernetes version. For example, if your control plane is running version 1.18 and any of your nodes are running version 1.17, update your nodes to version 1.18 before updating your control plane's Kubernetes version to 1.19. 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 the section called “Updating a managed node group” (p. 97) and the section called “Updates” (p. 117). To update the version of a Fargate node, delete the pod that is represented by the node and redeploy the pod after you update your control plane.
2. The pod security policy admission controller is enabled by default on Amazon EKS clusters. Before updating your cluster, ensure that the proper pod security policies are in place before you update to avoid any issues. You can check for the default policy with the following command:

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

   If you receive the following error, see default pod security policy (p. 363) 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, then you may 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, your manifest doesn't have the line and you can skip to the next step to update your cluster. If the word `upstream` is returned, then you need to remove the line.

b. Edit the configmap, removing the line near the top of the file that only has the word `upstream`. 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. See Kubernetes Version and Version Skew Support Policy for the rationale behind this requirement. Therefore, if your current version is 1.17 and you want to update to 1.19, then you must first update your cluster to 1.18 and then update it from 1.18 to 1.19.
- 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 also recommend that your self-managed nodes are at the same version as the control plane, though they can be up to one version behind the control plane's current version.
- Updating a cluster from 1.16 to 1.17 will fail if you have any AWS Fargate pods that have a `kubelet` minor version earlier than 1.16. Before updating your cluster from 1.16 to 1.17, you need to recycle your Fargate pods so that their `kubelet` is 1.16 before attempting to update the cluster to 1.17.
- You may need to update some of your deployed resources before you can update to 1.16. For more information, see Kubernetes 1.16 update prerequisites (p. 31).
- Updating your cluster to a newer version may overwrite custom configurations.

eksctl

This procedure requires `eksctl` version `0.43.0` or later. You can check your version with the following command:

```
eksctl version
```

For more information about installing or updating `eksctl`, see Installing or upgrading `eksctl` (p. 308).

Update your Amazon EKS control plane's Kubernetes version 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 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**, type the name of your cluster and choose **Confirm**.

   The update takes several minutes to complete.

AWS CLI

a. Update your cluster with the following AWS CLI command. Replace the *<example-values>* (including *<>*) with your own.

```
aws eks update-cluster-version \
 --region <region-code> \
 --name <my-cluster> \
 --kubernetes-version <1.19>
```

Output:

```
{
    "update": {
        "id": "<b5f0ba18-9a87-4450-b5a0-825e6e84496f>",
        "status": "InProgress",
        "type": "VersionUpdate",
        "params": [
            {
                "type": "Version",
                "value": "1.19"
            },
            {
                "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. Your update is complete when the status appears as `Successful`. The update takes several minutes to complete.

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

Output:

```
{
    "update": {
        "id": "b5f0ba18-9a87-4450-b5a0-825e6e84496f",
        "status": "Successful",
        "type": "VersionUpdate",
        "params": [
            {
                "type": "Version",
                "value": "1.19"
            },
            {
                "type": "PlatformVersion",
                "value": "eks.1"
            }
        ],
...
```

```
            "errors": []
        }
}
```

5. Patch the `kube-proxy` daemonset to use the image that corresponds to your cluster's Region and current Kubernetes version (in this example, `1.19.6`).

Kubernetes version	**1.19**	**1.18**	**1.17**	**1.16**	**1.15**
KubeProxy	1.19.6	1.18.8	1.17.9	1.16.13	1.15.11

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

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

   Example output

   ```
   602401143452.dkr.ecr.us-west-2.amazonaws.com/eks/kube-proxy:v1.18.8-eksbuild.1
   ```

   b. Update `kube-proxy` to the recommended version by replacing *`602401143452`* and *`us-west-2`* with the values from your output and replace *`1.19.6`* with your cluster's recommended `kube-proxy` version. If you're deploying a version that is earlier than `1.19.6`, then replace *`eksbuild.2`* with `eksbuild.1`.

   ```
   kubectl set image daemonset.apps/kube-proxy \
     -n kube-system \
     kube-proxy=602401143452.dkr.ecr.us-west-2.amazonaws.com/eks/kube-
   proxy:v1.19.6-eksbuild.2
   ```

   c. (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 do 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: "beta.kubernetes.io/arch"
                       operator: In
                       values:
                         - amd64
                         - arm64
   ```

   d. (Optional) If your cluster was oriniginally 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 earilier and intend to use Fargate nodes, then edit your `kube-proxy` manifest to include a `NodeAffinity` rule to prevent `kube-proxy` pods from sheduling on Fargate nodes. This is a one-time edit. Once you've added the `Affinity Rule` to your manifest, you don't need to do 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
```

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

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

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

   If the output shows `coredns` in the pod names, you're already running CoreDNS in your cluster. If not, see Installing or upgrading CoreDNS (p. 243) to install CoreDNS on your cluster, update it to the recommended version, return here, and skip steps 7-8.

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

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

   Output:

```
coredns:v<1.6.6>
```

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

Kubernetes version	1.19	1.18	1.17	1.16	1.15
CoreDNS	1.8.0	1.7.0	1.6.6	1.6.6	1.6.6

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

   a. Open the configmap with the following command.

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

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

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

9. Retrieve your current `coredns` image:

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

10. Update `coredns` to the recommended version by replacing *`602401143452`*, *`us-west-2`*, and *`amazonaws.com`* in the following command with your output from the previous command. Replace *`1.8.0`* with the version recommended for your cluster's Kubernetes version and then run the modified command.

```
kubectl set image --namespace kube-system deployment.apps/coredns \
            coredns=602401143452.dkr.ecr.us-west-2.amazonaws.com/eks/coredns:v1.8.0-
eksbuild.1
```

11. (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 `coredns` 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 do it each time you update. If you cluster was deployed on or after August 17, 2020, then `coredns` is already multi-architecture capable.

```
kubectl edit -n kube-system deployment/coredns
```

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.

```
- key: "beta.kubernetes.io/arch"
                    operator: In
                    values:
                      - amd64
                      - arm64
```

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

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

The output is as follows:

```
amazon-k8s-cni:<1.6.3>
```

If your CNI version is earlier than 1.7.5, then use the appropriate command below to update your CNI version to the latest recommended version.

Important
Any changes you've made to the plugin's default settings on your cluster can be overwritten with default settings when applying the new version of the manifest. To prevent loss of your custom settings, download the manifest, change the default settings as necessary, and then apply the modified manifest to your cluster.

- US West (Oregon) (`us-west-2`)

```
kubectl apply -f https://raw.githubusercontent.com/aws/amazon-vpc-cni-k8s/v1.7.5/
config/v1.7/aws-k8s-cni.yaml
```

- AWS GovCloud (US-East) (`us-gov-east-1`)

```
kubectl apply -f https://raw.githubusercontent.com/aws/amazon-vpc-cni-k8s/v1.7.5/
config/v1.7/aws-k8s-cni-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/v1.7.5/
config/v1.7/aws-k8s-cni-us-gov-west-1.yaml
```

- For all other Regions
  - Download the manifest file.

  ```
  curl -o aws-k8s-cni.yaml https://raw.githubusercontent.com/aws/amazon-vpc-cni-k8s/
  v1.7.5/config/v1.7/aws-k8s-cni.yaml
  ```

  - Replace *`<region-code>`* in the following command with the Region that your cluster is in. Then, run the modified command to replace the Region code in the file (currently `us-west-2`).

  ```
  sed -i -e 's/us-west-2/<region-code>/' aws-k8s-cni.yaml
  ```

  - Apply the modified manifest file to your cluster.

  ```
  kubectl apply -f aws-k8s-cni.yaml
  ```

13. (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 1.19 find the latest Cluster Autoscaler release that begins with 1.19. Record the semantic version number (`<1.19.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 *`1.19.n`* with your own value.

    ```
    kubectl -n kube-system set image deployment.apps/cluster-autoscaler cluster-
    autoscaler=k8s.gcr.io/autoscaling/cluster-autoscaler:v1.19.n
    ```

14. (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/v0.8.0/
    nvidia-device-plugin.yml
    ```

15. After your cluster update is complete, update your nodes to the same Kubernetes version of your updated cluster. For more information, see Self-managed node updates (p. 117) or Updating a managed node group (p. 97). Any new pods launched on Fargate will have a `kubelet` version that matches your cluster version. Existing Fargate pods won't be changed.

Kubernetes 1.16 update prerequisites

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

Warning
If you don't change these APIs before updating to 1.16, workloads fail after the update is complete.

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

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

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

If you originally created an Amazon EKS cluster with Kubernetes version 1.11 or earlier and haven't removed the `--resource-container` flag from the `kube-proxy` DaemonSet, then updating to Kubernetes 1.16 will cause `kube-proxy` failures. This flag is no longer supported in Kubernetes 1.16. For more information, see `kube-proxy` in Kubernetes 1.16 Deprecations and removals. You must remove this flag before updating to Kubernetes 1.16.

What you need to do before updating to 1.16

- Change your YAML files to reference the new APIs.
- Update custom integrations and controllers to call the new APIs.
- Ensure that you use an updated version of any third party tools, such as ingress controllers, continuous delivery systems, and other tools that call the new APIs.

  To easily check for discontinued API usage in your cluster, make sure that the `audit` control plane log (p. 55) is enabled, and specify `v1beta` as a filter for the events. All of the replacement APIs are in Kubernetes versions later than 1.10. Applications on any supported version of Amazon EKS can begin using the updated APIs now.
- Remove the `--resource-container=""` flag from your `kube-proxy` DaemonSet, if your cluster was originally deployed with Kubernetes 1.11 or earlier or use a kube-proxy configuration file (recommended). To determine whether your current version of `kube-proxy` has the flag, enter the following command.

```
kubectl get daemonset kube-proxy --namespace kube-system -o yaml | grep 'resource-
container='
```

  If you receive no output, then you don't need to remove anything. If you receive output similar to `--resource-container=""`, then you need to remove the flag. Enter the following command to edit your current `kube-proxy` config.

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

With the editor open, remove the `--resource-container=""` line, and save the file. We recommend that you instead, start using a kube-proxy configuration file. To do so, download the following manifest.

```
curl -o kube-proxy-daemonset.yaml https://amazon-eks.s3-us-west-2.amazonaws.com/
cloudformation/2020-06-10/kube-proxy-daemonset.yaml
```

Determine your cluster's endpoint with the following command.

```
aws eks describe-cluster \
    --name <cluster-name> \
    --region <region-code> \
    --query 'cluster.endpoint' \
    --output text
```

The output is as follows:

```
https://<A89DBB2140C8AC0C2F920A36CCC6E18C>.sk1.<region-code>.eks.amazonaws.com
```

Edit the `kube-proxy-daemonset.yaml` file that you downloaded. In your editor, replace `<MASTER_ENDPOINT>` (including `<>`) with the output from the previous command. Replace `<REGION>` with your cluster's Region. On the same line, replace the version with the version of your cluster if necessary. Apply the file with the following command.

```
kubectl apply -f kube-proxy-daemonset.yaml
```

Configure an Amazon EKS add-on

An add-on is Kubernetes operational software that provides capabilities like observability, scaling, networking, and AWS cloud resource integrations for your Amazon EKS cluster. You can manage add-ons yourself, or let Amazon EKS control the launch and version of the add-on through the Amazon EKS API for clusters running Kubernetes version 1.18 with platform version `eks.3` or later. Amazon EKS add-ons use the Kubernetes *Server-Side Apply* feature, which is only available with Kubernetes version 1.18 or later. For more information, see Server-Side Apply in the Kubernetes documentation.

You can configure an `eksctl` using eksctl or the AWS Management Console.

eksctl

Replace the *`<example values>`* (including `<>`) with your own values.

1. Add an Amazon EKS add-on to your cluster.

   a. Determine which Amazon EKS add-ons are available for your cluster.

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

   b. Add an Amazon EKS add-on to your cluster. When specifying `<name-of-addon>`, specify a name returned from the previous command. The `--force` option overwrites any existing configuration if the add-on is already installed on your cluster, but you're managing it yourself. You can specify the ARN of an IAM role, policy, or both. The add-on's Kubernetes service account is then annotated with the role. If you don't specify an existing role or policy, `eksctl` creates a default role and attaches the necessary policy to it for you.

```
eksctl create addon --name <name--of-addon-from-previous-command> --
cluster <my-cluster> --force
```

2. You can update an add-on to a newer version. See other update options with `eksctl update addon -h`.

   a. Determine which versions are available for an add-on.

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

   b. Update the add-on to a newer version.

   ```
   eksctl update addon --name <name-from-previous-command> --version <version-
   from-previous-command> --cluster <my-cluster>
   ```

3. Delete an Amazon EKS add-on from your cluster.

   a. See which add-ons are currently enabled for your cluster.

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

   b. Delete an Amazon EKS add-on from your cluster. Deleting the add-on also deletes any IAM roles associated to it.

   ```
   eksctl delete addon --cluster <my-cluster> --name <addon-name-from-previous-
   command>
   ```

AWS Management Console

To configure an 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, select **Clusters**, and then select the name of the cluster that you want to configure an Amazon EKS add-on for.
3. Choose the **Configuration** tab and then choose the **Add-ons** tab.
4. Configure the add-on. To add a new Amazon EKS add-on, select **Add new**. To change an existing Amazon EKS add-on, select the add-on and choose **Edit**. To remove an add-on, select the add-on and select **Remove**. Removing the add-on also removes it from your cluster. You can always manually add the add-on back to your cluster, but be aware that its removal and addition could cause service disruption to your cluster.

   a. Select the **Name** of the Amazon EKS add-on that you want to add or edit.
   b. Select the **Version** of the Amazon EKS add-on that you want to use.
   c. Select the **Service account role** that you want the add-on to run as. If you don't select an existing IAM role, then the the section called "Node IAM role" (p. 341) is used. However, we recommend specifying your own role so that the node IAM role isn't assigned more than the minimum permissions that it requires.

      Whichever role you choose must be assigned the permissions required by the add-on. For example, the role that you specify for the **vpc-cni** add-on must have the `AmazonEKS_CNI_Policy` IAM policy assigned to it, and the role must have unique settings in its trust relationship. For more information, see Create your CNI plugin IAM role with the AWS Management Console (p. 218). If the Kubernetes service account used by the add-on isn't currently annotated with the IAM role that you specify, then the API will annotate the service account with the IAM role for you.

> **Important**
> To specify an IAM role, you must have an IAM OpenID Connect (OIDC) provider for your cluster. To create an OIDC provider, see the section called "Create OIDC provider" (p. 349).

d. If you currently have the add-on deployed to your cluster, are managing it yourself, and want Amazon EKS to manage the add-on you can **Enable Override existing configuration for this add-on on the cluster**. If you enable this option, then any setting for the existing add-on can be overwritten with the Amazon EKS add-on's settings. If you currently have the add-on deployed to your cluster and don't enable this option and any of the Amazon EKS add-on settings conflict with your existing settings, then migrating the add-on to an Amazon EKS add-on will fail, and you'll receive an error message to help you resolve the conflict.

e. Select **Add** or **Update**.

Enabling envelope encryption on an existing cluster

If you enable secret encryption, the Kubernetes secrets are encrypted using the AWS Key Management Service (KMS) customer master key (CMK) that you select. The CMK must be symmetric, created in the same region as the cluster, and if the CMK was created in a different account, the user must have access to the CMK. For more information, see Allowing users in other accounts to use a CMK in the *AWS Key Management Service Developer Guide*. Enabling envelope encryption on an existing cluster is supported for Kubernetes version `1.13` or later.

> **Warning**
> You cannot disable envelope 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:

```
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:

```
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 automatically re-encrypt your secrets:

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

To opt-out of automatically re-encrypting your secrets:

```
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 to which you want to add KMS encryption.
3. Click on the **Configuration** tab.
4. Scroll down to the **Secrets encryption** section and click on the **Enable** button.
5. Select a key from the dropdown menu and click the **Enable** button. If no keys are listed, you must create one first. For more information, see Creating keys
6. Click the **Confirm** button to use the chosen key.

AWS CLI

1. Associate envelope encryption configuration with your cluster using the following AWS CLI command. Replace the *<example-values>* (including <>) 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>"}}]'
```

Output:

```
{
  "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 `cluster name` and `update ID` that was returned in the **Output** of the step above. Your update is complete when the status is shown as `Successful`.

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

Output:

```
{
  "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 will contain `EncryptionConfig`.

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

After you have enabled encryption on your cluster, you will need to encrypt all existing secrets with the new key:

Note
`eksctl` users do not need to run the following command unless they chose to opt-out of re-encrypting their secrets automatically.

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

Warning
If you enable envelope encryption for an existing cluster and the key that you use is ever deleted, then there is no path to recovery for the cluster. Deletion of the CMK will permanently put the cluster in a degraded state.

Note
By default, the `create-key` command creates a symmetric key with a key policy that gives the account's root user 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 key policy for the principal that will be calling the `create-cluster` API. Amazon EKS does not support the key policy condition `kms:GrantIsForAWSResource`. Creating a cluster will not work if this action is in the key policy statement.

Deleting a cluster

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

Important

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

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 `0.43.0` or later. You can check your version with the following command:

```
eksctl version
```

For more information on installing or upgrading `eksctl`, see Installing or upgrading `eksctl` (p. 308).

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, select **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. 88) 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 **Actions**, **Delete Stack**.
   c. On the **Delete Stack** confirmation screen, choose **Yes, Delete**. 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. Select the VPC stack to delete and choose **Actions** and then **Delete Stack**.
   b. On the **Delete Stack** confirmation screen, choose **Yes, Delete**.

AWS CLI

To delete an Amazon EKS cluster with the AWS CLI

1. List all services running in your cluster.

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

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

```
kubectl delete svc <service-name>
```

3. Delete 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. 88) only.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Amazon EKS cluster endpoint access control

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

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

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

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

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

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

Modifying cluster endpoint access

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

API server endpoint access options

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

Endpoint public access	Endpoint private access	Behavior
		• Your cluster API server is accessible from the internet. You can, optionally, limit the CIDR blocks that can access the public endpoint.
Disabled	Enabled	• All traffic to your cluster API server must come from within your cluster's VPC or a connected network. • There is no public access to your API server from the internet. Any `kubectl` commands must come from within the VPC or a connected network. For connectivity options, see Accessing a private only API server (p. 45). • The cluster's API server endpoint is resolved by public DNS servers to a private IP address from the VPC. In the past, the endpoint could only be resolved from within the VPC. If your endpoint does not resolve to a private IP address within the VPC for an existing cluster, you can: • Enable public access and then disable it again. You only need to do so once for a cluster and the endpoint will resolve to a private IP address from that point forward. • Update (p. 24) 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. 322). 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 1.19.7 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. 322). 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. 210).

  When you configure `kubectl` for your bastion host, be sure to use AWS credentials that are already mapped to your cluster's RBAC configuration, or add the IAM user or role that your bastion will use to the RBAC configuration before you remove endpoint public access. For more information, see Managing users or IAM roles for your cluster (p. 288) and Unauthorized or access denied (`kubectl`) (p. 375).
- **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. 210).

  When you configure `kubectl` for your AWS Cloud9 IDE, be sure to use AWS credentials that are already mapped to your cluster's RBAC configuration, or add the IAM user or role that your IDE will use to the RBAC configuration before you remove endpoint public access. For more information, see Managing users or IAM roles for your cluster (p. 288) and Unauthorized or access denied (`kubectl`) (p. 375).

Cluster Autoscaler

The Kubernetes Cluster Autoscaler automatically adjusts the number of nodes in your cluster when pods fail or are rescheduled onto other nodes. That is, the AWS Cloud Provider implementation within the Kubernetes Cluster Autoscaler controls the `.DesiredReplicas` field of Amazon EC2 Auto Scaling groups. The Cluster Autoscaler is typically installed as a Deployment in your cluster. It uses leader election to ensure high availability, but scaling is one done by a single replica at a time.

Before you deploy the Cluster Autoscaler, it's important to understand how Kubernetes concepts relate to 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 the AWS platform (for example 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 do exist as an abstraction in the Cluster Autoscaler, Cluster API, and other components. Nodes that exist within a single node group might share several properties like labels and taints. However, they can still consist of multiple Availability Zones or instance types.

- **Amazon EC2 Auto Scaling groups** – A feature of AWS that is 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. 88) are implemented using Amazon EC2 Auto Scaling groups, and are compatible with the Cluster Autoscaler.

This topic shows 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.

- Have an existing Kubernetes cluster – If you don't have a cluster, see Creating an Amazon EKS cluster (p. 17).
- An existing IAM OIDC provider for your cluster. To determine whether you have one, or to create one if you don't, see Create an IAM OIDC provider for your cluster (p. 349).
- 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` to create your node groups, then you must manually tag your Auto Scaling groups with the following tags. For more information, see Tagging your Amazon EC2 resources in the Amazon EC2 User Guide for Linux Instances.

Key	Value
`k8s.io/cluster-autoscaler/<cluster-name>`	`owned`
`k8s.io/cluster-autoscaler/enabled`	`TRUE`

Create an IAM policy and role

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

1. Create an IAM policy.

   a. Save the following contents to a file 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": [
        {
            "Action": [
                "autoscaling:DescribeAutoScalingGroups",
                "autoscaling:DescribeAutoScalingInstances",
                "autoscaling:DescribeLaunchConfigurations",
                "autoscaling:DescribeTags",
                "autoscaling:SetDesiredCapacity",
                "autoscaling:TerminateInstanceInAutoScalingGroup",
                "ec2:DescribeLaunchTemplateVersions"
            ],
            "Resource": "*",
            "Effect": "Allow"
        }
    ]
}
```

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
```

Note the ARN returned in the output for use 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 tab with the name of the tool that you'd like to create the role with.

eksctl

1. Run the following command if you created your 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-<cluster-name>-nodegroup-ng-<xxxxxxxx>-PolicyAutoScaling`.

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

2. If you created your node groups using the `--asg-access` option, we recommend that you detach the IAM policy that `eksctl` created and attached to the the section called "Node IAM role" (p. 341) that `eksctl` created for your node groups. Detaching the policy from the node IAM role enables the Cluster Autoscaler to function properly, but 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 navigation panel, choose **Roles**, **Create Role**.
c. In the **Select type of trusted entity** section, choose **Web identity**.
d. In the **Choose a web identity provider** section:
   i. For **Identity provider**, choose the URL for your cluster.
   ii. For **Audience**, choose `sts.amazonaws.com`.
e. Choose **Next: Permissions**.
f. In the **Attach Policy** section, select the `AmazonEKSClusterAutoscalerPolicy` policy that you created in step 1 to use for your service account.
g. Choose **Next: Tags**.
h. On the **Add tags (optional)** screen, you can add tags for the account. Choose **Next: Review**.
i. For **Role Name**, enter a name for your role, such as `AmazonEKSClusterAutoscalerRole`, and then choose **Create Role**.

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

k. Choose the **Trust relationships** tab, and then choose **Edit trust relationship**.

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

```
"oidc.eks.us-west-2.amazonaws.com/id/EXAMPLED539D4633E53DE1B716D3041E:aud":
 "sts.amazonaws.com"
```

Change the line to look like the following line. Replace `<EXAMPLED539D4633E53DE1B716D3041E>` (including <>)with your cluster's OIDC provider ID and replace <region-code> with the Region code that your cluster is in.

```
"oidc.eks.<region-code>.amazonaws.com/id/<EXAMPLED539D4633E53DE1B716D3041E>:sub":
 "system:serviceaccount:kube-system:cluster-autoscaler"
```

m. Choose **Update Trust Policy** to finish.

Deploy the Cluster Autoscaler

Complete the following steps to deploy the Cluster Autoscaler. We recommend you review Deployment considerations (p. 50) and optimize the Cluster Autoscaler deployment before you deploy it to a production cluster.

To deploy the Cluster Autoscaler

1. Deploy the Cluster Autoscaler.

```
kubectl apply -f https://raw.githubusercontent.com/kubernetes/autoscaler/master/
cluster-autoscaler/cloudprovider/aws/examples/cluster-autoscaler-autodiscover.yaml
```

2. 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::<AWS_ACCOUNT_ID>:role/<AmazonEKSClusterAutoscalerRole>
```

3. 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"}}}}}'
```

4. Edit the Cluster Autoscaler deployment with the following command.

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

Edit the `cluster-autoscaler` container command to replace *`<YOUR CLUSTER NAME>`* (including *`<>`*) with your cluster's name, and add the following options.

- `--balance-similar-node-groups`
- `--skip-nodes-with-system-pods=false`

```
    spec:
      containers:
      - command:
        - ./cluster-autoscaler
        - --v=4
        - --stderrthreshold=info
        - --cloud-provider=aws
        - --skip-nodes-with-local-storage=false
        - --expander=least-waste
        - --node-group-auto-discovery=asg:tag=k8s.io/cluster-autoscaler/enabled,k8s.io/
cluster-autoscaler/<YOUR CLUSTER NAME>
        - --balance-similar-node-groups
        - --skip-nodes-with-system-pods=false
```

Save and close the file to apply the changes.

5. Open the Cluster Autoscaler releases page from GitHub 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 1.19, find the latest Cluster Autoscaler release that begins with 1.19. Record the semantic version number (`1.19.n`) for that release to use in the next step.
6. Set the Cluster Autoscaler image tag to the version that you recorded in the previous step with the following command. Replace *`1.19.n`* with your own value.

```
kubectl set image deployment cluster-autoscaler \
  -n kube-system \
  cluster-autoscaler=k8s.gcr.io/autoscaling/cluster-autoscaler:v<1.19.n>
```

View your Cluster Autoscaler logs

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

View your Cluster Autoscaler logs with the following command.

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

Output:

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

Deployment considerations

Review the following considerations to optimize your Cluster Autoscaler deployment.

Scaling considerations

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

Availability Zones

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

Node group composition

The Cluster Autoscaler makes assumptions about how you are using node groups, such as which instance types you use within a group. To align with these assumptions, it's important to configure your node group based on these considerations:

- Each node in a node group has identical scheduling properties, such as labels, taints, and resources.
  - For `MixedInstancePolicies`, the instance types must be of the same shape for CPU, memory, and GPU.
  - The first instance type specified in the policy simulates scheduling.
  - If your policy has additional instance types with more resources, resources may be wasted after scale out.
  - If your policy has additional instance types with fewer resources than the original instance types, pods may fail to schedule on the instances.
- We recommend that you configure a smaller number of node groups with a larger number of nodes, rather than the opposite. This is because the opposite configuration negatively affects scalability.
- We recommend that you use Amazon EC2 features whenever both systems provide support. (For example, use Regions and `MixedInstancePolicy`.)

If possible, we recommend that you use Managed node groups (p. 88). Managed node groups come with powerful management features, including features for Cluster Autoscaler like automatic Amazon EC2 Auto Scaling group discovery and graceful node termination.

EBS volumes

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, but you are limited to doing this within a specific zone. For more information, see How do I use persistent storage in Amazon EKS?. Stateful applications can be highly available if sharded (split) across multiple Availability Zones using a separate Amazon EBS volume for each Availability Zone. As such, 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`.
- Node groups are configured with identical settings except for different Availability Zones and 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. This can be achieved by setting pod affinity for all co-scheduled pods or node affinity using `topologyKey: failure-domain.beta.kubernetes.io/zone`. Based on this configuration, the Cluster Autoscaler scales out a specific zone to match demands. You might want to 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=false`.
- 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.
  - If zonal pods schedule onto regional node groups, this results in imbalanced capacity for your regional pods.
  - If your zonal workloads can tolerate disruption and relocation, configure pod preemption to enable regionally scaled pods to force preemption and rescheduling on a less contested zone.

Accelerators and GPUs

Some clusters take advantage of specialized hardware accelerators such as a dedicated GPU. When scaling out, the accelerator device plugin 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 node’s available resources, 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 expensive. 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, you can 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, which can yield significant cost savings. It 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 specific `NodeSelectors` or `Taints`, which can't be discovered from the `LaunchConfiguration`. The Cluster Autoscaler can account for these factors by discovering them from the following tags on the Auto Scaling group.

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

Note
When scaling to zero, your capacity is returned to Amazon EC2 and may be unavailable in the future.

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

The primary items that you can change to tune the performance and scalability of the Cluster Autoscaler are the resources provided to the process, the scan interval of the algorithm, and the number of node groups in the cluster. There are other factors involved in the true runtime complexity of this algorithm, such as scheduling plug-in complexity and the number of pods. These are considered to be unconfigurable parameters, because they are natural to the cluster's workload 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 scalability limits are reached, the Cluster Autoscaler's performance and functionality degrades. Additionally, when it exceeds its scalability limits, 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 stimuli, such as a pod becoming unschedulable.

Familiarizing yourself with the autoscaling algorithm's runtime complexity makes tuning the Cluster Autoscaler to continue operating smoothly in large clusters (with more than 1,000 nodes) easier.

The Cluster Autoscaler loads the entire cluster's state into memory, including pods, nodes, and node groups. On each scan interval, the algorithm identifies unschedulable pods and simulates scheduling for each node group. Tuning these factors come with different tradeoffs, which should be carefully considered.

Vertical autoscaling

The simplest way to scale the Cluster Autoscaler to larger clusters is to increase the resource requests for its deployment. Both memory and CPU should be increased for large clusters, though this varies significantly with 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. Increasing resources is typically done manually. If you find that constant resource tuning is creating an operational burden, consider using the Addon Resizer or Vertical Pod Autoscaler.

Reducing the number of node groups

Minimizing the number of node groups is one way to ensure that the Cluster Autoscaler performs well on large clusters. This may be challenging if you structure your node groups on an individual team or application basis. 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 reasons to use multiple node groups, such as 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.

- Pod isolation is done using namespaces rather than node groups.
  - This may not be possible in low-trust multi-tenant clusters.
  - Pod `ResourceRequests` and `ResourceLimits` are properly set to avoid resource contention.
  - Larger instance types result in more optimal bin packing and reduced system pod overhead.
- `NodeTaints` or `NodeSelectors` are used to schedule pods as the exception, not as the rule.
- Regional resources are defined as a single Amazon EC2 Auto Scaling group with multiple Availability Zones.

Reducing the scan interval

A low scan interval, such as 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, changing the interval to one minute results in a trade-off of 6x reduced API calls for 38% slower scale ups.

Sharding across node groups

The Cluster Autoscaler can be configured to operate on a specific set of node groups. Using this functionality, it’s possible to deploy multiple instances of the Cluster Autoscaler, each configured to operate on a different set of node groups. When you use this strategy, you can use arbitrarily large numbers of node groups, trading cost for scalability. However, we only recommend using this strategy as a last resort for improving performance.

The Cluster Autoscaler wasn't originally designed for this configuration, so there are some side effects. Because the shards don't communicate, it’s possible for multiple autoscalers to attempt to schedule an unschedulable pod. This 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 like 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 associated with prematurely terminating pods due to an aggressive scale down decision.

Spot instances

You can use Spot Instances in your node groups and 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 may 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 autoscaler’s scheduling simulator uses the first instance type in the Mixed Instance Policy. If subsequent instance types are larger, resources may be wasted after a scale up. If 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. 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 may 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 will try to scale up the Amazon EC2 Auto Scaling group matching the name `p2-node-group`. If this operation doesn't succeed within `--max-node-provision-time`, it will attempt to scale an Amazon EC2 Auto Scaling group matching the name `p3-node-group`. This value defaults to 15 minutes and can be reduced for more responsive node group selection, though if the value is too low, it can cause unnecessary scale outs.

Overprovisioning

The Cluster Autoscaler minimizes costs by ensuring that nodes are only added to the cluster when needed and are removed when unused. This significantly impacts deployment latency because many pods will be forced to wait for a node 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.

Choosing the right amount of overprovisioned capacity is important. One way to make this decision is to determine your average scale up frequency and divide this number by the amount 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. This can reduce scheduling latency by 30 seconds at the cost of a single additional Amazon EC2 instance. To improve zonal scheduling decisions, you can overprovision the number of nodes to be equal to 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 are interrupted. The Cluster Autoscaler functions to scale down any node under the `scale-down-utilization-threshold`. This interrupts any remaining pods on the node. However, you can prevent this 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`.

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 authentication (p. 288).
- **Controller manager (`controllerManager`)** – The controller manager manages the core control loops that are shipped with Kubernetes. For more information, see kube-controller-manager in the Kubernetes documentation.
- **Scheduler (`scheduler`)** – The scheduler component manages when and where to run pods in your cluster. For more information, see kube-scheduler in the Kubernetes documentation.

Enabling and disabling control plane logs

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

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

To enable or disable control plane logs with the console

1. Open the Amazon EKS console at https://console.aws.amazon.com/eks/home#/clusters.
2. Choose the name of the cluster to display your cluster information.
3. 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. Substitute your cluster name and desired endpoint access values.

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

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

   Output:

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

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

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

   Output:

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

Viewing cluster control plane logs

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

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

To view your cluster control plane logs on the CloudWatch console

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

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

   - **Kubernetes API server component logs (`api`)** – `kube-apiserver-<nnn...>`
   - **Audit (`audit`)** – `kube-apiserver-audit-<nnn...>`
   - **Authenticator (`authenticator`)** – `authenticator-<nnn...>`
   - **Controller manager (`controllerManager`)** – `kube-controller-manager-<nnn...>`
   - **Scheduler (`scheduler`)** – `kube-scheduler-<nnn...>`

Amazon EKS Kubernetes versions

The Kubernetes project is rapidly evolving with new features, design updates, and bug fixes. The community releases new Kubernetes minor versions, such as 1.19, as generally available approximately every three months, and each minor version is supported for approximately twelve months after it is first released.

Available Amazon EKS Kubernetes versions

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

- 1.19.6
- 1.18.9
- 1.17.12
- 1.16.15
- 1.15.12

Unless your application requires a specific version of Kubernetes, we recommend that you choose the latest available Kubernetes version supported by Amazon EKS for your clusters. As new Kubernetes versions become available in Amazon EKS, we recommend that you proactively update your clusters to use the latest available version. For more information, see Updating a cluster (p. 24). For more information, see Amazon EKS Kubernetes release calendar (p. 63) and Amazon EKS version support and FAQ (p. 64).

Kubernetes 1.19

Kubernetes 1.19 is now available in Amazon EKS. For more information about Kubernetes 1.19, see the official release announcement.

Important

- Starting with 1.19, Amazon EKS no longer adds the `kubernetes.io/cluster/<cluster-name>` tag to subnets passed in during cluster creation. 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 "Cluster VPC considerations" (p. 208).
  - Subnet tags are not 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 the section called "AWS Load Balancer Controller" (p. 238). For more information about subnet tagging when using a load balancer, see the section called "Application load balancing" (p. 271) and the section called "Network load balancing" (p. 266).
- You're no longer required to provide a security context for non-root containers that need to access the web identity token file for use with IAM roles for service accounts. For more information, see the section called "IAM roles for service accounts" (p. 345) andproposal for file permission handling in projected service account volume on GitHub.
- The pod identity webhook has been updated to address the missing startup probes GitHub issue. The webhook also now supports an annotation to control token expiration. For more information, see the GitHub pull request.
- CoreDNS version 1.8.0 is the recommended version for Amazon EKS 1.19 clusters. This version is installed by default in new Amazon EKS 1.19 clusters. For more information, see the section called "Installing or upgrading CoreDNS" (p. 243).
- Amazon EKS optimized Amazon Linux 2 AMIs include the Linux kernel version 5.4 for Kubernetes version 1.19. For more information, see the section called "Amazon EKS optimized Amazon Linux AMI" (p. 150).
- The `CertificateSigningRequest API` has been promoted to stable `certificates.k8s.io/v1` with the following changes:
  - `spec.signerName` is now required. You can't create requests for `kubernetes.io/legacy-unknown` with the `certificates.k8s.io/v1` API.
  - You can continue to create CSRs with the `kubernetes.io/legacy-unknown` signer name with the `certificates.k8s.io/v1beta1` API.
  - You can continue to request that a CSR to is signed for a non-node server cert, webhooks, for example, with the `certificates.k8s.io/v1beta1` API. These CSRs aren't auto-approved.
  - To approve certificates, a privileged user requires `kubectl` 1.18.8 or later.

  For more details on the certificate v1 API, see Certificate Signing Requests in the Kubernetes documentation.

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

Permission	Kind	Namespace	Reason
`eks:certificate-controller`	`Rolebinding`	`kube-system`	Impacts signer and approver functionality in the control plane.

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

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

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

Kubernetes 1.18 is now available in Amazon EKS. 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, allowing 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. 271) (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.

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

Kubernetes 1.17

Kubernetes 1.17 is now available in Amazon EKS. For more information about Kubernetes 1.17, see the official release announcement.

Important

- EKS has not enabled the `CSIMigrationAWS` feature flag. This will be enabled in a future release, along with detailed migration instructions. For more info on CSI migration, see the Kubernetes blog.
- Updating a cluster from 1.16 to 1.17 will fail if any of your AWS Fargate pods have a `kubelet` minor version earlier than 1.16. Before updating your cluster from 1.16 to 1.17, you need to recycle your Fargate pods so that their `kubelet` is 1.16 before attempting to update the cluster to 1.17. To recycle a Kubernetes deployment on a 1.15 or later cluster, use the following command.

```
kubectl rollout restart deployment <deployment-name>
```

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

- Cloud Provider Labels have reached general availability. If you are 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. This feature allows you 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 is not 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, allowing you 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.

Kubernetes 1.16

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

Important

- Kubernetes 1.16 removes a number of discontinued APIs. Changes to your applications may be required before updating your cluster to 1.16. Carefully follow the 1.16 update prerequisites (p. 31) before updating.
- Starting with 1.16, the Amazon EKS certificate authority will honor certificate signing requests with SAN X.509 extensions, which resolves the EKS CA should honor SAN x509 extension feature request from GitHub.

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

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

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

Kubernetes 1.15

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

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

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

- For more information about VPC tagging, see VPC tagging requirement (p. 210).

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

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

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

  Note
  When running CoreDNS on Amazon EC2, we recommend not using `force_tcp` in the configuration and ensuring that `options use-vc` is not set in `/etc/resolv.conf`.

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

Kubernetes version	Upstream release	Amazon EKS release	Amazon EKS end of support
1.15	June 19, 2019	March 10, 2020	May 3, 2021
1.16	September 8, 2019	April 30, 2020	July, 2021
1.17	December 9, 2019	July 10, 2020	September, 2021
1.18	March 23, 2020	October 13, 2020	November, 2021

Kubernetes version	Upstream release	Amazon EKS release	Amazon EKS end of support
1.19	August 26, 2020	February 16, 2021	April, 2022
1.20	December 8, 2020	April, 2021	June, 2022

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 fully supported for 14 months after first being available on Amazon EKS. This is true even if upstream Kubernetes is no longer supporting a version available on Amazon EKS. We backport security patches that are applicable to the Kubernetes versions 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 Personal Health Dashboard approximately 12 months after the Kubernetes version was released on Amazon EKS. The notice includes the end of support date, which 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 are no longer able to create new Amazon EKS clusters with the unsupported version. Existing control planes are automatically updated by Amazon EKS to the oldest supported version through a gradual deployment process after the end of support date. After the automatic control plane update, you must manually update cluster add-ons and Amazon EC2 nodes. For more information, see the section called "Update an existing cluster" (p. 25).

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

A: Amazon EKS is unable to provide specific timeframes. Automatic updates can happen at any time after the end of support date. We recommend that you take proactive action and update your control plane without relying on the Amazon EKS automatic update process. For more information, see the section called "Updating a cluster" (p. 24).

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

A: No. Cloud security at AWS is the highest priority. Amazon EKS does not allow control planes to stay on a version that has 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, as well as beta features which are enabled by default. Alpha features are not 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. If Amazon EKS automatically updates your control plane, the Kubernetes version on your managed node group may be more than one version earlier than your control plane. If a managed node group contains instances that are running a version of Kubernetes that is 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. 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. 97). 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. 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. 117).

The Kubernetes project tests compatibility between the control plane and nodes for up to two minor versions. For example, 1.17 nodes continue to operate when orchestrated by a 1.19 control plane. However, running a cluster with nodes that are persistently two minor versions behind the control plane is not 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. 324). Amazon EKS uses the Kubernetes eviction API to attempt to gracefully drain pods running on Fargate. For more information, see The Eviction API in the Kubernetes documentation. If a pod can't be evicted, then Amazon EKS issues a Kubernetes `delete pod` command. We strongly recommend running Fargate pods as part of a replication controller like a Kubernetes deployment 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 is the same version as your updated cluster control plane version.

> **Important**
> If you update the control plane, you must 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 is the same version as your cluster.

Amazon EKS platform versions

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

When a new Kubernetes minor version is available in Amazon EKS, such as 1.19, the initial Amazon EKS platform version for that Kubernetes minor version starts at `eks.1`. However, Amazon EKS releases new platform versions periodically to enable new Kubernetes control plane settings and to provide security fixes.

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

- The Amazon EKS platform version number is incremented (`eks.<n+1>`).
- Amazon EKS automatically upgrades all existing clusters to the latest Amazon EKS platform version for their corresponding Kubernetes minor version.
- Amazon EKS might publish a new 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.

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

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

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

Kubernetes version 1.19

Kubernetes version	Amazon EKS platform version	Enabled admission controllers	Release notes
`1.19.6`	`eks.2`	`NamespaceLifecycle, LimitRanger, ServiceAccount, DefaultStorageClass, ResourceQuota, DefaultTolerationSeconds, NodeRestriction, MutatingAdmissionWebhook, ValidatingAdmissionWebhook, PodSecurityPolicy, TaintNodesByCondition, Priority, StorageObjectInUseProtection, PersistentVolumeClaimResize, ExtendedResourceToleration`	New platform version with security fixes and enhancements.
`1.19.6`	`eks.1`	`NamespaceLifecycle, LimitRanger, ServiceAccount, DefaultStorageClass, ResourceQuota, DefaultTolerationSeconds, NodeRestriction, MutatingAdmissionWebhook, ValidatingAdmissionWebhook, PodSecurityPolicy, TaintNodesByCondition, Priority, StorageObjectInUseProtection, PersistentVolumeClaimResize, ExtendedResourceToleration`	Initial release of Kubernetes 1.19 for Amazon EKS. For more information, see Kubernetes 1.19 (p. 59).

Kubernetes version 1.18

Kubernetes version	Amazon EKS platform version	Enabled admission controllers	Release notes
`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. 33) and Fargate logging (p. 135).
`1.18.9`	`eks.2`	`NamespaceLifecycle, LimitRanger, ServiceAccount, DefaultStorageClass, ResourceQuota, DefaultTolerationSeconds, NodeRestriction, MutatingAdmissionWebhook, ValidatingAdmissionWebhook, PodSecurityPolicy, TaintNodesByCondition, Priority, StorageObjectInUseProtection, PersistentVolumeClaimResize`	New platform version with security fixes and enhancements.
`1.18.8`	`eks.1`	`NamespaceLifecycle, LimitRanger, ServiceAccount, DefaultStorageClass, ResourceQuota, DefaultTolerationSeconds, NodeRestriction,`	Initial release of Kubernetes 1.18 for Amazon EKS. For more information, see Kubernetes 1.18 (p. 60).

Kubernetes version	Amazon EKS platform version	Enabled admission controllers	Release notes
		MutatingAdmissionWebhook, ValidatingAdmissionWebhook, PodSecurityPolicy, TaintNodesByCondition, Priority, StorageObjectInUseProtection, PersistentVolumeClaimResize	

Kubernetes version 1.17

Kubernetes version	Amazon EKS platform version	Enabled admission controllers	Release notes
`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.
`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. 135).
`1.17.12`	`eks.4`	NamespaceLifecycle, LimitRanger, ServiceAccount, DefaultStorageClass, ResourceQuota, DefaultTolerationSeconds, NodeRestriction, MutatingAdmissionWebhook, ValidatingAdmissionWebhook, PodSecurityPolicy,	New platform version with security fixes and enhancements.

Kubernetes version	Amazon EKS platform version	Enabled admission controllers	Release notes
		`TaintNodesByCondition`, `Priority`, `StorageObjectInUseProtection`, `PersistentVolumeClaimResize`	
`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. 219). This release creates a `vpc-resource-controller` service account that is required for the VPC resource controller.
`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. 61).

Kubernetes version 1.16

Kubernetes version	Amazon EKS platform version	Enabled admission controllers	Release notes
`1.16.15`	`eks.6`	`NamespaceLifecycle, LimitRanger, ServiceAccount, DefaultStorageClass, ResourceQuota, DefaultTolerationSeconds, NodeRestriction, MutatingAdmissionWebhook, ValidatingAdmissionWebhook, PodSecurityPolicy, TaintNodesByCondition, Priority, StorageObjectInUseProtection, PersistentVolumeClaimResize`	New platform version with security fixes and enhancements.
`1.16.15`	`eks.5`	`NamespaceLifecycle, LimitRanger, ServiceAccount, DefaultStorageClass, ResourceQuota, DefaultTolerationSeconds, NodeRestriction, MutatingAdmissionWebhook, ValidatingAdmissionWebhook, PodSecurityPolicy, TaintNodesByCondition, Priority, StorageObjectInUseProtection, PersistentVolumeClaimResize`	Includes support for Fargate logging (p. 135).
`1.16.15`	`eks.4`	`NamespaceLifecycle, LimitRanger, ServiceAccount, DefaultStorageClass, ResourceQuota, DefaultTolerationSeconds, NodeRestriction, MutatingAdmissionWebhook, ValidatingAdmissionWebhook, PodSecurityPolicy, TaintNodesByCondition, Priority, StorageObjectInUseProtection, PersistentVolumeClaimResize`	New platform version with security fixes and enhancements.
`1.16.13`	`eks.3`	`NamespaceLifecycle, LimitRanger, ServiceAccount, DefaultStorageClass, ResourceQuota, DefaultTolerationSeconds, NodeRestriction,`	New platform version with security fixes and enhancements, including UDP support for services of type `LoadBalancer` when using NLB. For more

Kubernetes version	Amazon EKS platform version	Enabled admission controllers	Release notes
		MutatingAdmissionWebhook, ValidatingAdmissionWebhook, PodSecurityPolicy, TaintNodesByCondition, Priority, StorageObjectInUseProtection, PersistentVolumeClaimResize	information, see the Allow UDP for AWS NLB issue on GitHub.
1.16.8	eks.2	NamespaceLifecycle, LimitRanger, ServiceAccount, DefaultStorageClass, ResourceQuota, DefaultTolerationSeconds, NodeRestriction, MutatingAdmissionWebhook, ValidatingAdmissionWebhook, PodSecurityPolicy, TaintNodesByCondition, Priority, StorageObjectInUseProtection, PersistentVolumeClaimResize	New platform version with security fixes.
1.16.8	eks.1	NamespaceLifecycle, LimitRanger, ServiceAccount, DefaultStorageClass, ResourceQuota, DefaultTolerationSeconds, NodeRestriction, MutatingAdmissionWebhook, ValidatingAdmissionWebhook, PodSecurityPolicy, TaintNodesByCondition, Priority, StorageObjectInUseProtection, PersistentVolumeClaimResize	Initial release of Kubernetes 1.16 for Amazon EKS. For more information, see Kubernetes 1.16 (p. 62).

Kubernetes version 1.15

Kubernetes version	Amazon EKS platform version	Enabled admission controllers	Release notes
1.15.12	eks.7	NamespaceLifecycle, LimitRanger, ServiceAccount, DefaultStorageClass, ResourceQuota, DefaultTolerationSeconds, NodeRestriction, MutatingAdmissionWebhook, ValidatingAdmissionWebhook, PodSecurityPolicy,	New platform version with security fixes and enhancements.

Kubernetes version	Amazon EKS platform version	Enabled admission controllers	Release notes
		`TaintNodesByCondition`, `Priority`, `StorageObjectInUseProtection`, `PersistentVolumeClaimResize`	
`1.15.12`	`eks.6`	`NamespaceLifecycle`, `LimitRanger`, `ServiceAccount`, `DefaultStorageClass`, `ResourceQuota`, `DefaultTolerationSeconds`, `NodeRestriction`, `MutatingAdmissionWebhook`, `ValidatingAdmissionWebhook`, `PodSecurityPolicy`, `TaintNodesByCondition`, `Priority`, `StorageObjectInUseProtection`, `PersistentVolumeClaimResize`	Includes support for Fargate logging (p. 135).
`1.15.12`	`eks.5`	`NamespaceLifecycle`, `LimitRanger`, `ServiceAccount`, `DefaultStorageClass`, `ResourceQuota`, `DefaultTolerationSeconds`, `NodeRestriction`, `MutatingAdmissionWebhook`, `ValidatingAdmissionWebhook`, `PodSecurityPolicy`, `TaintNodesByCondition`, `Priority`, `StorageObjectInUseProtection`, `PersistentVolumeClaimResize`	New platform version with security fixes and enhancements.
`1.15.11`	`eks.4`	`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. For more information, see the Allow UDP for AWS NLB issue on GitHub.

Kubernetes version	Amazon EKS platform version	Enabled admission controllers	Release notes
`1.15.11`	`eks.3`	`NamespaceLifecycle, LimitRanger, ServiceAccount, DefaultStorageClass, ResourceQuota, DefaultTolerationSeconds, NodeRestriction, MutatingAdmissionWebhook, ValidatingAdmissionWebhook, PodSecurityPolicy, TaintNodesByCondition, Priority, StorageObjectInUseProtection, PersistentVolumeClaimResize`	New platform version with security fixes.
`1.15.11`	`eks.2`	`NamespaceLifecycle, LimitRanger, ServiceAccount, DefaultStorageClass, ResourceQuota, DefaultTolerationSeconds, NodeRestriction, MutatingAdmissionWebhook, ValidatingAdmissionWebhook, PodSecurityPolicy, TaintNodesByCondition, Priority, StorageObjectInUseProtection, PersistentVolumeClaimResize`	New platform version with bug fixes and enhancements, including an update to the server side AWS IAM Authenticator, with IAM role traceability improvements.
`1.15.10`	`eks.1`	`NamespaceLifecycle, LimitRanger, ServiceAccount, DefaultStorageClass, ResourceQuota, DefaultTolerationSeconds, NodeRestriction, MutatingAdmissionWebhook, ValidatingAdmissionWebhook, PodSecurityPolicy, TaintNodesByCondition, Priority, StorageObjectInUseProtection, PersistentVolumeClaimResize`	Initial release of Kubernetes 1.15 for Amazon EKS. For more information, see Kubernetes 1.15 (p. 63).

Windows support

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

Considerations

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

- Amazon EC2 instance types C3, C4, D2, I2, M4 (excluding m4.16xlarge), and R3 instances are not supported for Windows workloads.
- Host networking mode is not supported for Windows workloads.
- Amazon EKS clusters must contain one or more Linux nodes to run core system pods that only run on Linux, such as `coredns` and the VPC resource controller.
- The `kubelet` and `kube-proxy` event logs are redirected to the `EKS` Windows Event Log and are set to a 200 MB limit.
- You can't use Security groups for pods (p. 219) with pods running on Windows nodes.
- 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 Linux Instances*.
- Group Managed Service Accounts (GMSA) for Windows pods and containers is not supported by Amazon EKS versions earlier than 1.16. You can follow the instructions in the Kubernetes documentation to enable and test this alpha feature on clusters that are earlier than 1.16.
- 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.
- You can't deploy Windows managed or Fargate nodes. You can only create self-managed Windows nodes. For more information, see the section called "Windows" (p. 112).

Enabling Windows support

The following steps help you to enable Windows support for your Amazon EKS cluster. You can use `eksctl`, a Windows client, or a macOS or Linux client to enable Windows support for your cluster.

eksctl

To enable Windows support for your cluster with `eksctl`

Prerequisite

This procedure requires `eksctl` version `0.43.0` 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. 308).

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
   ```
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. 112).

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

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

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

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

Windows

To enable Windows support for your cluster with a Windows client

In the following steps, replace *us-west-2* with the 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/us-west-2/
   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/us-west-2/vpc-admission-webhook/
      latest/vpc-admission-webhook-deployment.yaml;
      curl -o Setup-VPCAdmissionWebhook.ps1 https://s3.us-west-2.amazonaws.com/
      amazon-eks/manifests/us-west-2/vpc-admission-webhook/latest/Setup-
      VPCAdmissionWebhook.ps1;
      curl -o webhook-create-signed-cert.ps1 https://s3.us-west-2.amazonaws.com/
      amazon-eks/manifests/us-west-2/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/us-west-2/vpc-admission-webhook/latest/webhook-patch-ca-
      bundle.ps1;
      ```

   b. Install OpenSSL and jq.

   c. Set up and deploy the VPC admission webhook.

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

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

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

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

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

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

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

Apply the configuration to the cluster.

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

4. After you have enabled Windows support, you can launch a Windows node group into your cluster. For more information, see Launching self-managed Windows nodes (p. 112).

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

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

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

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

macOS and Linux

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

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

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

1. Deploy the VPC resource controller to your cluster.

```
kubectl apply -f https://s3.us-west-2.amazonaws.com/amazon-eks/manifests/us-west-2/
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/us-west-2/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/us-west-2/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/us-west-2/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
```

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. 112).

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

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

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

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

Deploy a Windows sample application

To deploy a Windows sample application

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

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

```
      nodeSelector:
        kubernetes.io/os: windows
---
apiVersion: v1
kind: Service
metadata:
  name: windows-server-iis-service
  namespace: default
spec:
  ports:
  - port: 80
    protocol: TCP
    targetPort: 80
  selector:
    app: windows-server-iis
    tier: backend
    track: stable
  sessionAffinity: None
  type: LoadBalancer
```

2. Deploy the application to the cluster.

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

3. Get the status of the pod.

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

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

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

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

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

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

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

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. 55). 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. Click on 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 a 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. 81).
- 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. 40).
- You may need to include the VPC endpoints found at VPC endpoints for private clusters (p. 82).
- You must include the following text to the bootstrap arguments when launching self-managed nodes. This text bypasses the Amazon EKS introspection and does not require access to the Amazon EKS API from within the VPC. Replace <cluster-endpoint> and <cluster-certificate-authority> with the values from your Amazon EKS cluster.

```
--apiserver-endpoint <cluster-endpoint> --b64-cluster-ca <cluster-certificate-authority>
```

- The `aws-auth` ConfigMap must be created from within the VPC. For more information about create the `aws-auth` ConfigMap, see Managing users or IAM roles for your cluster (p. 288).

Considerations

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

- AWS X-Ray is not supported with private clusters.
- Amazon CloudWatch Logs is supported with private clusters, but you must use an Amazon CloudWatch Logs VPC endpoint. For more information, see VPC endpoints for private clusters (p. 82).
- Self-managed and managed nodes (p. 104) 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.
- IAM roles for service accounts (p. 345) is supported. You must include the STS VPC endpoint. For more information, see VPC endpoints for private clusters (p. 82).
- The Amazon EBS CSI driver (p. 181) is supported. Before deploying, the kustomization.yaml file must be changed to set the container images to use the same Region as the Amazon EKS cluster.
- The Amazon EFS CSI driver (p. 186) is supported. Before deploying, the kustomization.yaml file must be changed to set the container images to use the same Region as the Amazon EKS cluster.
- The Amazon FSx for Lustre CSI driver (p. 197) is not supported.
- AWS Fargate (p. 124) is supported with private clusters. You must include the STS VPC endpoint. For more information, see VPC endpoints for private clusters (p. 82). You can use the AWS load balancer controller to deploy AWS Application Load Balancers 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 the section called “Application load balancing” (p. 271) and the section called “Load balancer – IP targets” (p. 269).
- App Mesh is supported with private clusters when you use the App Mesh Envoy VPC endpoint. For more information, see VPC endpoints for private clusters (p. 82).
  - 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 is not supported. For more information, see App Mesh controller on GitHub.

Creating local copies of container images

Because a private cluster has no outbound internet access, container images cannot 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 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
   Be 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
```

VPC endpoints for private clusters

The following VPC endpoints may be required.

- `com.amazonaws.<region>.ec2`
- `com.amazonaws.<region>.ecr.api`
- `com.amazonaws.<region>.ecr.dkr`
- `com.amazonaws.<region>.s3` – For pulling container images
- `com.amazonaws.<region>.logs` – For CloudWatch Logs
- `com.amazonaws.<region>.sts` – If using AWS Fargate or IAM roles for service accounts
- `com.amazonaws.<region>.elasticloadbalancing` – If using Application Load Balancers
- `com.amazonaws.<region>.autoscaling` – If using Cluster Autoscaler
- `com.amazonaws.<region>.appmesh-envoy-management` – If using App Mesh

Amazon EKS nodes

Your Amazon EKS cluster can schedule pods on any combination of Self-managed nodes (p. 104), Amazon EKS Managed node groups (p. 88), and AWS Fargate (p. 124). The following table provides several criteria to evaluate when deciding which options best meets your requirements. We recommend reviewing this page often because the data in this table changes frequently as new capabilities are introduced to Amazon EKS. To view details about any existing nodes deployed in your cluster, see View nodes (p. 86).

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. 371).	No
Can be deployed to AWS Local Zones	No	Yes – For more information, see Amazon EKS on AWS Local Zones (p. 373).	No
Can run containers that require Windows	No	Yes (p. 73) – 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. 283) – Amazon Linux nodes only	Yes (p. 283) – Amazon Linux only	No
Can run workloads that require a GPU	Yes (p. 148) – Amazon Linux nodes only	Yes (p. 148) – Amazon Linux only	No
Can run workloads that require Arm processors	Yes (p. 149)	Yes (p. 149)	No
Can run AWS Bottlerocket	No	Yes (p. 110)	No – There is no node.
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

Criteria	EKS managed node groups	Self-managed nodes	AWS Fargate
			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. 91) – 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. 105), Linux (Arm) (p. 149), or Windows (p. 73) 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. 100) with a custom AMI	Yes – For more information, view the bootstrap script usage information on GitHub.	No – There is no node.
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. 100) with a custom AMI	Yes, using CNI custom networking (p. 230).	No – There is no node.
Can SSH into node	Yes	Yes	No – There is no node host operating system to SSH to.
Can deploy your own custom AMI to nodes	Yes – Using a launch template (p. 100)	Yes	No – You don't manage nodes.
Can deploy your own custom CNI to nodes	Yes – Using a launch template (p. 100) with a custom AMI	Yes	No – You don't manage nodes.

Criteria	EKS managed node groups	Self-managed nodes	AWS Fargate
Must update node AMI yourself	Yes – If you deployed an Amazon EKS optimized AMI, then you're notified in the Amazon EKS console when updates are available and can perform the update with one click in the console. If you deployed a custom AMI, then you're not notified in the Amazon EKS console when updates are available and must perform the update yourself.	Yes - Using tools other than the Amazon EKS console, because self-managed nodes can't be managed with the Amazon EKS console.	No – You don't manage nodes.
Must update node Kubernetes version yourself	Yes – If you deployed an Amazon EKS optimized AMI, then you're notified in the Amazon EKS console when updates are available and can perform the update with one click in the console. If you deployed a custom AMI, then you're not notified in the Amazon EKS console when updates are available and must perform the update yourself.	Yes – Using tools other than the Amazon EKS console, 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. 181)	Yes (p. 181)	No
Can use Amazon EFS storage with pods	Yes (p. 186)	Yes (p. 186)	Yes (p. 186)
Can use Amazon FSx for Lustre storage with pods	Yes (p. 197)	Yes (p. 197)	No

Criteria	EKS managed node groups	Self-managed nodes	AWS Fargate
Can use Network Load Balancer for services	Yes (p. 266)	Yes (p. 266)	Yes, when using the the section called “Load balancer – IP targets” (p. 269)
Pods can run in a public subnet	Yes	Yes	No
Can assign different VPC security groups to individual pods	Yes (p. 219) – Linux nodes only	Yes (p. 219) – Linux nodes only	No
Can run Kubernetes DaemonSets	Yes	Yes	No
Support `HostPort` and `HostNetwork` in the pod manifest	Yes	Yes	No
Region availability	All Amazon EKS supported regions	All Amazon EKS supported regions	Some Amazon EKS supported regions (p. 124)
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.

View nodes

The Amazon EKS console shows information about all of your cluster's nodes, including Amazon EKS managed nodes, self-managed nodes, and Fargate. Nodes represent the compute resources provisioned for your cluster from the perspective of the Kubernetes API. For more information, see Nodes in the Kubernetes documentation. To learn more about the different types of Amazon EKS nodes that you can deploy your workloads (p. 253) to, see *Nodes* (p. 83).

Prerequisites

The IAM user or IAM role that you sign into the AWS Management Console with must meet the following requirements.

- Have the `eks:AccessKubernetesApi` and other necessary IAM permissions to view nodes attached to it. For an example IAM policy, see the section called “View nodes and workloads for all clusters in the AWS Management Console” (p. 333) .
- Is mapped to Kubernetes user or group in the `aws-auth` `configmap`. For more information, see the section called “Managing users or IAM roles for your cluster” (p. 288).
- The Kubernetes user or group that the IAM user or role is mapped to in the configmap must be bound to a Kubernetes `role` or `clusterrole` that has permissions to view the resources in the namespaces that you want to view. For more information, see Using RBAC Authorization in the Kubernetes documentation. You can download the following example manifests that create a `clusterrole` and `clusterrolebinding` or a `role` and `rolebinding`:

- **View Kubernetes resources in all namespaces** – The group name in the file is `eks-console-dashboard-full-access-group`, which is the group that your IAM user or role needs to be mapped to in the `aws-auth` configmap. You can change the name of the group before applying it to your cluster, if desired, and then map your IAM user or role to that group in the configmap. To download the file, select the appropriate link for the Region that your cluster is in.
  - All Regions other than Beijing and Ningxia China
  - Beijing and Ningxia China Regions
- **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. The group name in the file is `eks-console-dashboard-restricted-access-group`, which is the group that your IAM user or role needs to be mapped to in the `aws-auth` configmap. You can change the name of the group before applying it to your cluster, if desired, and then map your IAM user or role to that group in the configmap. To download the file, select the appropriate link for the Region that your cluster is in.
  - All Regions other than Beijing and Ningxia China
  - Beijing and Ningxia China Regions

To view nodes 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 panel, select **Clusters**, and then in the **Clusters** list, select the cluster that you want to view compute resources for.
3. On the **Overview** tab, you see a list of all compute **Nodes** for your cluster and the nodes' status.

   Important
   If you can't see any **Nodes** on the **Overview** tab, or you see a **Your current user or role does not have access to Kubernetes objects on this EKS cluster** error, see the prerequisites for this topic. If you don't resolve the issue, you can still view and manage your Amazon EKS cluster on the **Configuration** tab, but you won't see self-managed nodes or some of the information that you see for managed nodes and Fargate under **Nodes**.

   Note
   Each pod that runs on Fargate is registered as a separate Kubernetes node within the cluster. This is because Fargate runs each pod in an isolated compute environment and independently connects to the cluster control plane. For more information, see the section called "AWS Fargate" (p. 124).
4. In the **Nodes** list, you see a list of all of the managed, self-managed, and Fargate nodes for your cluster. Selecting the link for one of the nodes provides the following information about the node:
   - The Amazon EC2 **Instance type**, **Kernel version**, **Kubelet version**, **Container runtime**, **OS** and **OS image** for managed and self-managed nodes.
   - Deep links to the Amazon EC2 console and the Amazon EKS managed node group (if applicable) for the node.
   - The **Resource allocation**, which shows baseline and allocatable capacity for the node.
   - **Conditions** describe the current operational status of the node. This is useful information for troubleshooting issues on the node.

     Conditions are reported back to the Kubernetes control plane by the Kubernetes agent `kubelet` that runs locally on each node. For more information, see kubelet in the Kubernetes documentation. Conditions on the node are always reported as part of the node detail and the **Status** of each condition along with its **Message** indicates the health of the node for that condition. The following common conditions are reported for a node:
     - **Ready** – This condition is **TRUE** if the node is healthy and can accept pods. The condition is **FALSE** if the node is not ready and will not accept pods. **UNKNOWN** indicates that the

Kubernetes control plane has not recently received a heartbeat signal from the node. The heartbeat timeout period is set to the Kubernetes default of 40 seconds for Amazon EKS clusters.

  - **Memory pressure** – This condition is **FALSE** under normal operation and **TRUE** if node memory is low.
  - **Disk pressure** – This condition is **FALSE** under normal operation and **TRUE** if disk capacity for the node is low.
  - **PID pressure** – This condition is **FALSE** under normal operation and **TRUE** if there are too many processes running on the node. On the node, each container runs as a process with a unique *Process ID*, or PID.
  - **NetworkUnavailable** – This condition is **FALSE**, or not present, under normal operation. If **TRUE**, the network for the node is not properly configured.
- The Kubernetes **Labels** and **Annotations** assigned to the node. These could have been assigned by you, by Kubernetes, or by the Amazon EKS API when the node was created. These values can be used by your workloads for scheduling pods.

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. Nodes run using the latest Amazon EKS optimized AMIs in your AWS account. Node updates and terminations automatically and gracefully drain nodes to ensure that your applications stay available.

All managed nodes are provisioned as part of an Amazon EC2 Auto Scaling group that's managed for you by Amazon EKS. All resources including the instances and Auto Scaling groups run within your AWS account. Each node group uses the Amazon EKS optimized Amazon Linux 2 AMI and can run across multiple Availability Zones that you define.

You can add a managed node group to new or existing clusters using the Amazon EKS console, `eksctl`, AWS CLI; AWS API, or infrastructure as code tools including AWS CloudFormation. Nodes launched as part of a managed node group are automatically tagged for auto-discovery by the Kubernetes cluster autoscaler. 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. 9).

To add a managed node group to an existing cluster, see Creating a managed node group (p. 91).

Managed node groups concepts

- Amazon EKS managed node groups create and manage Amazon EC2 instances for you.
- All managed nodes are provisioned as part of an Amazon EC2 Auto Scaling group that's managed for you by Amazon EKS. Moreover, all resources including Amazon EC2 instances and Auto Scaling groups run within your AWS account.

- A managed node group's Auto Scaling group spans all of the subnets that you specify when you create the group.
- Amazon EKS tags managed node group resources so that they are configured to use the Kubernetes Cluster Autoscaler (p. 45).

  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. 45), you should configure multiple node groups, each scoped to a single Availability Zone. In addition, you should enable the `--balance-similar-node-groups` feature.
- By default, instances in a managed node group use the latest version of the Amazon EKS optimized Amazon Linux 2 AMI for its cluster's Kubernetes version. You can choose between standard and GPU variants of the Amazon EKS optimized Amazon Linux 2 AMI. If you deploy using a launch template, you can also use a custom AMI. For more information, see Launch template support (p. 100).
- 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. 97).
- 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. 204) on or after March 26, 2020, then this setting is already set to true. If the public subnets were created 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.
- 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. 371) 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 a health issue, Amazon EKS returns an error message to help you to diagnose the issue. For more information, see Managed node group errors (p. 376).
- 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, you must encrypt all new Amazon EBS volumes created in your account by default. 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, you need to 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, we recommend that you 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. Cordoning 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 such as 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 that can tolerate periods where the required capacity is not available.
- We recommend that you use On-Demand for applications that are fault intolerant, including 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 the section called "Cluster Autoscaler" (p. 45), we recommend using a flexible set of instance types with the same amount of vCPU and memory resources to ensure that the nodes in your cluster scale as expected. For example, if you need four vCPUs and eight GiB memory, we recommend that you 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, each using 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 instead of passing a single instance type through the launch template. For more information about deploying a node group using a launch template, see the section called "Launch template support" (p. 100).

Creating a managed node group

This topic describes how you can launch an Amazon EKS managed node group of Linux 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 the section called "Amazon Linux" (p. 105), the section called "Bottlerocket" (p. 110), and the section called "Windows" (p. 112).

Prerequisite

An existing cluster. If you don't have an existing cluster, we recommend that you follow one of the *Getting started with Amazon EKS* (p. 4) guides to create your cluster and node group.

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 `0.43.0` or later. You can check your version with the following command:

```
eksctl version
```

For more information on installing or upgrading `eksctl`, see Installing or upgrading `eksctl` (p. 308).

You can create your node group with or without a launch template. A launch template allows for greater customization of a node group, to include deploying a custom AMI. If you plan to use Security groups for pods (p. 219), then make sure to specify a supported Amazon EC2 instance type. For more information, see Amazon EC2 supported instances and branch network interfaces (p. 224). If specifying an Arm Amazon EC2 instance type, then review the considerations in Amazon EKS optimized Arm Amazon Linux AMIs (p. 149) before deploying.

1. (Optional) If the **AmazonEKS_CNI_Policy** managed IAM policy is attached to your the section called "Node IAM role" (p. 341), 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 VPC CNI plugin to use IAM roles for service accounts (p. 217).
2. Create your managed node group with or without using a custom launch template. For a complete list of all available options and defaults, enter the following command.

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

   Replace the *`<example values>`* (including the `<>`) with your own values.

   - **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. For a complete list of supported values for `--node-type`, see the list in `amazon-eks-nodegroup.yaml` on GitHub. 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*.

     If you plan to assign IAM roles to all of your Kubernetes service accounts so that pods only have the minimum permissions that they need, and no pods in the cluster require access to the Amazon EC2 instance metadata service (IMDS) for other reasons, such as retrieving the current Region, then we recommend blocking pod access to IMDS. For more information, see the section called "IAM roles for service accounts" (p. 345) and the section called "Restricting access to the instance profile" (p. 359). 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 <us-west-2> \
  --name <my-mng> \
  --node-type <m5.large> \
  --nodes <3> \
  --nodes-min <2> \
  --nodes-max <4> \
  --ssh-access \
  --ssh-public-key <my-key> \
  --managed
```

- **With a launch template** – The launch template must already exist and must meet the requirements specified in Launch template configuration basics (p. 100). If you plan to assign IAM roles to all of your Kubernetes service accounts so that pods only have the minimum permissions that they need, and no pods in the cluster require access to the Amazon EC2 instance metadata service (IMDS) for other reasons, such as retrieving the current Region, then we recommend blocking pod access to IMDS. For more information, see the section called "IAM roles for service accounts" (p. 345) and the section called "Restricting access to the instance profile" (p. 359). 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 llist of `eksctl` config file settings, see Config file schema in the `eksctl` documentation.

  b. Deploy the nodegroup with the following command.

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

AWS Management Console

To create your managed node group using the AWS Management Console

1. Wait for your cluster status to show as `ACTIVE`. You cannot create a managed node group for a cluster that is not yet `ACTIVE`.
2. Open the Amazon EKS console at https://console.aws.amazon.com/eks/home#/clusters.
3. Choose the name of the cluster that you want to create your managed node group in.
4. 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 name** – Choose the node instance role to use with your node group. For more information, see Amazon EKS node IAM role (p. 341).

  Important
  We recommend using a role that is not currently in use by any self-managed node group, or that you plan to use with a new self-managed node group. For more information, see Deleting a managed node group (p. 103).
- **Use launch template** – (Optional) Choose if you want to use an existing launch template and then select a **Launch template version** (Optional). 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, including allowing you to deploy a custom AMI. The launch template must meet the requirements in Launch template support (p. 100). 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. 345), assign necessary permissions directly to all pods that require access to AWS services, and no pods in your cluster require access to IMDS for other reasons, such as retrieving the current 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 the section called "Restricting access to the instance profile" (p. 359).
- **Kubernetes labels** – (Optional) You can choose to apply Kubernetes labels to the nodes in your managed node group.
- **Tags** – (Optional) You can choose to tag your Amazon EKS managed node group. These tags do not propagate to other resources in the node group, such as Auto Scaling groups or instances. For more information, see Tagging your Amazon EKS resources (p. 318).

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

Node group compute configuration

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

  If you are deploying Arm instances, be sure to review the considerations in the section called "Arm" (p. 149) before deploying.

  If you specified a launch template on the previous page, and specified an AMI in the launch template, then you cannot select a value. The value from the template is displayed. The AMI specified in the template must meet the requirements in Using a custom AMI (p. 103).
- **Capacity type** – Select a capacity type. For more information about choosing a capacity type, see the section called "Managed node group capacity types" (p. 89). You cannot 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 type** – One or more instance type is specified by default. 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. The console displays a set of commonly used instance types. For the complete set of supported instance types, see the list in `amazon-eks-nodegroup.yaml` on GitHub. If you need to create a managed node group with an instance type that is 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 cannot select a value because it 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. For more information about selecting instance types, see **Considerations** in the section called "Managed node group capacity types" (p. 89).

Each Amazon EC2 instance type supports a maximum number of elastic network interfaces (ENIs) and each ENI supports a maximum number of IP addresses. Since 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. Some instance types might not be available in all Regions.

If you plan to use Security groups for pods (p. 219), then make sure to specify a supported Amazon EC2 instance type. For more information, see Amazon EC2 supported instances and branch network interfaces (p. 224). If specifying an Arm Amazon EC2 instance type, then review the considerations in Amazon EKS optimized Arm Amazon Linux AMIs (p. 149) before deploying.

- **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 cannot select a value because it must be specified in the launch template.

Node group scaling configuration

Note
Amazon EKS does not automatically scale your node group in or out. However, you can configure the Kubernetes Cluster Autoscaler (p. 45) to do this for you.

- **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.

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. 45), 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 endpoing 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. 204) 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.
     - f you use a launch template and specify multiple network interfaces, Amazon EC2 will not 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.

- **Allow remote access to nodes** (Optional, but default). 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 your node group. You cannot 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. 101).
- 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.
- For **Allow 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. 374) 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/v0.8.0/
nvidia-device-plugin.yml
```

12. (Optional) After you add Linux worker nodes to your cluster, follow the procedures in Windows support (p. 73) to add Windows support to your cluster and to add Windows worker nodes. All Amazon EKS clusters must contain at least one Linux worker node, even if you only want to run Windows workloads in your cluster.

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 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`. If you want other users to have access to your cluster, then you must add them to the `aws-auth ConfigMap`. For more information, see Managing users or IAM roles for your cluster (p. 288).
- Restrict access to IMDS (p. 359) – If you plan to assign IAM roles to all of your Kubernetes service accounts so that pods only have the minimum permissions that they need, and no pods in the cluster require access to the Amazon EC2 instance metadata service (IMDS) for other reasons, such as retrieving the current Region, then we recommend blocking pod access to IMDS. For more information, see the section called “IAM roles for service accounts” (p. 345) and the section called “Restricting access to the instance profile” (p. 359).
- Cluster Autoscaler (p. 45) – Configure the Kubernetes Cluster Autoscaler to automatically adjust the number of nodes in your node groups.

- Deploy a sample Linux workload (p. 255) – Deploy a sample Linux application to test your cluster and Linux nodes.
- Cluster management (p. 303) – 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 and gracefully updates your nodes for you, completing the steps listed in the section called "Managed node update behavior" (p. 99). 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 the section called "View versions" (p. 150).
- 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.

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 action or update, the pods in that node are drained first. For more information, see Managed node update behavior (p. 99).

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 later 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 the *<example values>* (include *<>*) with your own.

```
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>` to the preceding command. The launch template must meet the requirements described in Launch

template support (p. 100). If the launch template includes a custom AMI, the AMI must meet the requirements in Using a custom AMI (p. 103). When you upgrade your node group to a newer version of your launch template, all of your nodes are 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 1.19, you can upgrade nodes currently running Kubernetes 1.18 to version 1.19 with the following command.

```
eksctl upgrade nodegroup --name=<node-group-name> --cluster=<cluster-name> --
kubernetes-version=<1.19>
```

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, all of your nodes are 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 is 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 do not propagate to other resources, such as subnets or Amazon EC2 instances in the node group.
   - **Kubernetes labels** – Add or remove Kubernetes labels to the nodes in your node group. The labels shown here are only the labels that you have applied with Amazon EKS. Other labels may exist on your nodes that aren't shown here.
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 the section called "Service quotas" (p. 322).
   - **Desired size** – Specify the current number of nodes that the managed node group should maintain.
6. When you're finished editing, choose **Save changes**.

Managed node update behavior

When you update a managed node group version to the latest AMI release version for your node group's Kubernetes version or to a newer Kubernetes version to match your cluster, Amazon EKS automatically completes the following tasks:

1. Creates a new Amazon EC2 launch template version for the Auto Scaling group associated with your node group. The new template uses the target AMI for the update.
2. Updates the Auto Scaling group to use the latest launch template with the new AMI.
3. Increments the Auto Scaling group maximum size and desired size by one up to twice the number of Availability Zones in the Region that the Auto Scaling group is deployed in. This ensures that at least one new instance comes up in every Availability Zone in the Region that your node group is deployed in.
4. Checks the nodes in the node group for the `eks.amazonaws.com/nodegroup-image` label, and applies a `eks.amazonaws.com/nodegroup=unschedulable:NoSchedule` taint on all of the nodes in the node group that aren't labeled with the latest AMI ID. This prevents nodes that have already been updated from a previous failed update from being tainted.
5. Randomly selects a node in the node group and evicts all pods from it.
6. Cordons the node after all of the pods are evicted. 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 healthy, active nodes.
7. Sends a termination request to the Auto Scaling group for the cordoned node.
8. Repeats steps 5-7 until there are no nodes in the node group that are deployed with the earlier version of the launch template.
9. Decrements the Auto Scaling group maximum size and desired size by 1 to return to your pre-update values.

Launch template support

Managed node groups are always deployed with an Amazon EC2 Auto Scaling Group launch template. If you don't specify your own launch template to use when creating a managed node group, the Amazon EKS API creates a launch template with default values in your account. Creating your own launch template and a managed node group from that template has the benefit of providing you with a greater level of flexibility and customization when deploying managed nodes than is provided by the default launch template. For the highest level of customization, you can deploy managed nodes using your own launch template and a custom AMI.

After you've deployed a managed node group with your own launch template, you can then 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 of the nodes in the group are recycled to match the new configuration of the specified launch template version. Existing node groups that do not 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 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 the node group configuration or the launch template, but not both. 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 and which similar settings (if any) 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 settings that are prohibited in a managed node group configuration and which similar settings (if any) 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.	**AMI** under **Launch template contents** – You must specify if you're using a custom AMI. If you specify an AMI that doesn't meet the requirements listed in Using a custom AMI (p. 103), the node group deployment will fail.

Amazon EKS node group configuration – Prohibited	Launch template
If an AMI type was not specified in the launch template, then you can select an AMI in the node group configuration.	
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)**.
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. 101).

Note

- If you deploy a node group using a launch template, then you can specify zero or one **Instance type** under **Launch template contents** in a launch template *or* you can specify 0-20 instance types for **Instance types** on the **Set compute and scaling configuration** page in the console, or 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, then the `t3.medium` instance type is used, by default. If your node group is using the Spot capacity type, then we recommend specifying multiple instance types using the console. For more information, see the section called "Managed node group capacity types" (p. 89).
- If any containers that you deploy to the node group use the Instance Metadata Service Version 2, then 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. 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 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. Therefore, you must ensure that the inbound and outbound rules of your security groups enable communication with your cluster's endpoint. Incorrect security group rules result in worker nodes being unable to join the cluster. To learn about the security group rules that you should need to apply, see Amazon EKS security group considerations (p. 210).
- If you need SSH access to the instances in your node group, be sure to include a security group that allows that access.

Amazon EC2 user data

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. This is because your user data is merged with Amazon EKS user data required for nodes to join the cluster. Do not specify any commands in your user data that starts or modifies `kubelet`, as this will be 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**
> If you have a need for advanced `kubelet` customization, including manually starting it or passing in custom configuration parameters, see Using a custom AMI (p. 103) for more information. Amazon EKS doesn't merge user data when a custom AMI ID is specified in a launch template.

You can combine multiple user data blocks together into a single MIME multi-part file. For example, you can combine a cloud boothook 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="==BOUNDARY=="`
- 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 – `--==BOUNDARY==`
  - 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: `--==BOUNDARY==--`

  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==--\
```

Using a custom AMI

If your organization needs to run a custom AMI due to specific security, compliance, or internal policy requirements, you can deploy such AMIs to managed node groups by using a launch template. 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.

> **Note**
> When using a custom 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.

To use a custom AMI with managed node groups, specify an AMI ID in the `imageId` field of the launch template. To update your node group to a newer version of a custom AMI, create a new version of the launch template with an updated AMI ID, and update the node group with the new launch template version.

Limitations of using custom AMIs with managed node groups

- You must create a new node group to switch between using custom AMIs and Amazon EKS optimized AMIs.
- The following fields can't be set in the API if you're using a custom AMI
  - `amiType`
  - `releaseVersion`
  - `version`
- The custom AMI can't be Windows 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 will first set the minimum, maximum, and desired size of your Auto Scaling group to zero, which will trigger a scale down of your node group. Before each instance is terminated, Amazon EKS will send a signal to drain the pods from that node and wait a few minutes. If the pods haven't drained after a few minutes, Amazon EKS will let Auto Scaling continue the termination of the instance. Once all instances are 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. 288). If any self-managed node groups in the cluster are using the same node IAM role, the self-managed nodes move to the `NotReady` status, and the cluster operation are also disrupted. You can add the mapping back to the ConfigMap to minimize disruption.

To delete a managed node group

1. Open the Amazon EKS console at https://console.aws.amazon.com/eks/home#/clusters.
2. Choose the cluster that contains the node group to delete.
3. Select the **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, type 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 your cluster's control plane via 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:

- Be the same instance type
- Be running the same Amazon Machine Image (AMI)
- Use the same Amazon EKS node IAM role (p. 341)

A cluster can contain several node groups. As long as 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, and you are billed for them based on normal EC2 prices. For more information, see Amazon EC2 pricing.

Amazon EKS provides specialized Amazon Machine Images (AMI) 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 your cluster's public endpoint using CIDR blocks, it is recommended that you also enable private endpoint access 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. 40).

To add self-managed nodes to your Amazon EKS cluster, see the topics that follow. If you launch self-managed nodes manually, then you must 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, then the required tag is automatically added to nodes for you.

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

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

Topics

- Self-managed node updates (p. 117)

Launching self-managed Amazon Linux nodes

This topic helps you to 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 launch self-managed Amazon Linux 2 nodes with `eksctl` or the AWS Management Console.

eksctl

Prerequisites

- An existing Amazon EKS cluster that was created using `eksctl`.
- `eksctl` version `0.43.0` or later. For more information about installing or upgrading `eksctl`, see Installing or upgrading `eksctl` (p. 308).

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. 341), 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 VPC CNI plugin to use IAM roles for service accounts (p. 217).
2. The following command creates a node group in an existing cluster. Replace the *`<example values>`* (including `<>`) with your own values. The nodes are created with the same Kubernetes version as the control plane, by default.

   If you want to deploy on Amazon EC2 Arm instances, then replace `t3.medium` with an Arm instance type. If specifying an Arm Amazon EC2 instance type, then review the considerations in Amazon EKS optimized Arm Amazon Linux AMIs (p. 149) before deploying. For a complete list of supported values for `--node-type`, see the list in `amazon-eks-nodegroup.yaml` on GitHub.

   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, then the AWS Outposts, AWS Wavelength, or AWS Local Zones subnets must not have been passed in when you created the cluster. You must 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.

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

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. 374) in the Troubleshooting guide.

Output:

You'll see several lines of output as 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) If you plan to assign IAM roles to all of your Kubernetes service accounts so that pods only have the minimum permissions that they need, and no pods in the cluster require access to the Amazon EC2 instance metadata service (IMDS) for other reasons, such as retrieving the current Region, then we recommend blocking pod access to IMDS. For more information, see the section called "IAM roles for service accounts" (p. 345) and the section called "Restricting access to the instance profile" (p. 359).

AWS Management Console

Prerequisites

- An existing VPC and security group that meet the requirements for an Amazon EKS cluster. For more information, see Cluster VPC considerations (p. 208) and Amazon EKS security group considerations (p. 210). The Getting started with Amazon EKS (p. 4) guide creates a VPC that meets the requirements, or you can also follow Creating a VPC for your Amazon EKS cluster (p. 204) to create one manually.
- An existing Amazon EKS cluster that uses a VPC and security group that meet the requirements of an Amazon EKS cluster. For more information, see Creating an Amazon EKS cluster (p. 17). 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.

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 will 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://raw.githubusercontent.com/awslabs/amazon-
eks-ami/master/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, fill out the following parameters accordingly:
   - **Stack name**: Choose a stack name for your AWS CloudFormation stack. For example, you can call it **cluster-name-nodes**.
   - **ClusterName**: Enter the name that you used when you created your Amazon EKS cluster.

> **Important**
> This name must exactly match the name you used in Step 1: Create your Amazon EKS cluster (p. 9); otherwise, your nodes cannot join the cluster.

- **ClusterControlPlaneSecurityGroup**: Choose the **SecurityGroups** value from the AWS CloudFormation output that you generated when you created your VPC (p. 204).
- **NodeGroupName**: Enter a name for your node group. This name can be used later to identify the Auto Scaling node group that is created for your 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. You can also view the list in the `amazon-eks-nodegroup.yaml` file on GitHub. Before choosing an Arm instance type, make sure to review the considerations in Amazon EKS optimized Arm Amazon Linux AMIs (p. 149).

  > **Note**
  > The supported instance types for the latest version of the Amazon VPC CNI plugin for Kubernetes are shown here. You may need to update your CNI version to take advantage of the latest supported instance types. For more information, see Amazon VPC CNI plugin for Kubernetes upgrades (p. 236).

  > **Important**
  > Some instance types might not be available in all Regions.

- **NodeImageIdSSMParam**: Pre-populated with the Amazon EC2 Systems Manager parameter of a recent Amazon EKS optimized Amazon Linux AMI ID for a Kubernetes version. If you want to use a different Kubernetes minor version supported with Amazon EKS, then you can replace `1.x` with a different supported version (p. 58). We recommend specifying the same Kubernetes version as your cluster.

  If you want to use the Amazon EKS optimized accelerated AMI, then replace `amazon-linux-2` with `amazon-linux-2-gpu`. If you want to use the Amazon EKS optimized Arm AMI, then 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 are using your own custom AMI (instead of the Amazon EKS optimized AMI), enter a node AMI ID for your Region. If you specify a value here, it overrides any values in the **NodeImageIdSSMParam** field.
- **NodeVolumeSize**: Specify a root volume size for your 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 do not 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.

Note

- If you are launching nodes into a private VPC without outbound internet access, then you need to include the following arguments.

```
--apiserver-endpoint <cluster-endpoint> --b64-cluster-ca <cluster-
certificate-authority>
```

- If you want to assign IP addresses to pods that are from a different CIDR block than the block that includes the IP address for the node, then you may need to add a CIDR block to your VPC and specify an argument to support the capability. For more information, see CNI custom networking (p. 230).

- **DisableIMDSv1**: Each node supports the Instance Metadata Service Version 1 (IMDSv1) and IMDSv2 by default, but you can disable IMDSv1. Select **true** if you don't want any nodes in the node group, or any pods scheduled on the nodes in the node group to use IMDSv1. For more information about IMDS, see Configuring the instance metadata service. For more information about restricting access to it on your nodes, see the section called "Restricting access to the instance profile" (p. 359).
- **VpcId**: Enter the ID for the VPC (p. 204) that you created.
- **Subnets**: Choose the subnets that you created for your VPC. If you created your VPC using the steps described in Creating a VPC for your Amazon EKS cluster (p. 204), 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 is not enabled for the public subnet, then any nodes that you deploy to that public subnet will not be assigned a public IP address and will not be able to communicate with the cluster or other AWS services. If the subnet was deployed before March 26, 2020 using either of the Amazon EKS AWS CloudFormation VPC templates (p. 204), 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 is able to communicate with the cluster and other AWS services through a NAT gateway.
- If the subnets do not have internet access, then make sure that you're aware of the considerations and extra steps in Private clusters (p. 80).
- 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`* (including *<>*) with the name of your cluster and 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, then you must manually add the tag to the subnet. For more information, see the section called "Subnet tagging" (p. 210).
- 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.

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.

To enable nodes to join your cluster

Note
If you launched nodes inside a private VPC without outbound internet access, then you must 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:

   ```
   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
      Do not modify any other lines in this file.

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

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

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

      Note
      If you receive any authorization or resource type errors, see Unauthorized or access denied (`kubectl`) (p. 375) in the troubleshooting section.

   If nodes fail to join the cluster, then see Nodes fail to join cluster (p. 374) 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/v0.8.0/
   nvidia-device-plugin.yml
   ```

4. (Optional) Deploy a sample application (p. 255) to test your cluster and Linux nodes.
5. (Optional) If the **AmazonEKS_CNI_Policy** managed IAM policy is attached to your the section called "Node IAM role" (p. 341), 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 VPC CNI plugin to use IAM roles for service accounts (p. 217).

6. (Optional) If you plan to assign IAM roles to all of your Kubernetes service accounts so that pods only have the minimum permissions that they need, and no pods in the cluster require access to the Amazon EC2 instance metadata service (IMDS) for other reasons, such as retrieving the current Region, then we recommend blocking pod access to IMDS. For more information, see the section called "IAM roles for service accounts" (p. 345) and the section called "Restricting access to the instance profile" (p. 359).

Launching self-managed Bottlerocket nodes

This topic helps you 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 that is purpose-built by AWS 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 the documentation 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, but not to instances that have GPUs or Inferentia chips.
- You can't deploy to the following regions: China (Beijing) (`cn-north-1`), China (Ningxia) (`cn-northwest-1`), AWS GovCloud (US-East) (`us-gov-east-1`), or AWS GovCloud (US-West) (`us-gov-west-1`).
- There is no AWS CloudFormation template to deploy nodes with.

To launch Bottlerocket nodes using `eksctl`

This procedure requires `eksctl` version `0.43.0` or later. You can check your version with the following command:

```
eksctl version
```

For more information on installing or upgrading `eksctl`, see Installing or upgrading `eksctl` (p. 308).

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

1. Create a file named *`bottlerocket.yaml`* with the following contents. Replace the *`example values`* with your own values. If you want to deploy on Arm instances, then replace *`m5.large`* with an Arm instance type. If specifying an Arm Amazon EC2 instance type, then review the considerations in Amazon EKS optimized Arm Amazon Linux AMIs (p. 149) before deploying. If you want to deploy using a custom AMI, then see Building Bottlerocket on GitHub and Custom AMI support in the `eksctl` documentation. If you want to deploy a managed node group then you must deploy a custom AMI using a launch template. For more information, see the section called "Launch template support" (p. 100).

   Important
   If you want to deploy a node group to AWS Outposts, AWS Wavelength, or AWS Local Zones subnets, then the AWS Outposts, AWS Wavelength, or AWS Local Zones subnets must not have been passed in when you created the cluster, and 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.

   ```
   ---
   ```

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

metadata:
  name: my-cluster
  region: us-west-2
  version: '1.19'

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
    ssh:
        allow: true
        publicKeyName: YOUR_EC2_KEYPAIR_NAME
```

2. Deploy your nodes with the following command.

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

Output:

You'll see several lines of output as 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 are not included with Bottlerocket. For more information about creating a storage class using the driver, see Amazon EBS CSI driver (p. 181).
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. If you prefer to keep Bottlerocket's default setting, then 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 as shown 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. 255) to test your Bottlerocket nodes.

6. (Optional) If you plan to assign IAM roles to all of your Kubernetes service accounts so that pods only have the minimum permissions that they need, and no pods in the cluster require access to the Amazon EC2 instance metadata service (IMDS) for other reasons, such as retrieving the current Region, then we recommend blocking pod access to IMDS. For more information, see the section called "IAM roles for service accounts" (p. 345) and the section called "Restricting access to the instance profile" (p. 359).

Launching self-managed Windows nodes

This topic helps you 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.

You must enable Windows support for your cluster and we recommend that you review important considerations before you launch a Windows node group. For more information, see Enabling Windows support (p. 74).

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 `0.43.0`. You can check your version with the following command:

```
eksctl version
```

For more information on installing or upgrading `eksctl`, see Installing or upgrading `eksctl` (p. 308).

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

1. (Optional) If the **AmazonEKS_CNI_Policy** managed IAM policy is attached to your the section called "Node IAM role" (p. 341), 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 VPC CNI plugin to use IAM roles for service accounts (p. 217).
2. This procedure assumes that you have an existing cluster named *`my-cluster`* in the *`us-west-2`* Region. For a different existing cluster, change the values. If you don't already have an Amazon EKS cluster and an Amazon Linux 2 node group to add a Windows node group to, then we recommend that you follow the Getting started with Amazon EKS – `eksctl` (p. 4) guide instead. The guide provides a complete end-to-end walkthrough for creating an Amazon EKS cluster with Amazon Linux nodes.

   Create your node group with the following command. Replace the *`<example values>`* (including *`<>`*) with your own values.

   Important
   If you want to deploy a node group to AWS Outposts, AWS Wavelength, or AWS Local Zones subnets, then the AWS Outposts, AWS Wavelength, or AWS Local Zones subnets must not have been passed in when you created the cluster. You must 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.

```
eksctl create nodegroup \
  --region <us-west-2> \
  --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. 374) in the Troubleshooting guide.
- For more information on the available options for `eksctl` commands, enter the following command.

```
eksctl <command> -help
```

Output:

You'll see several lines of output as 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. 78) to test your cluster and Windows nodes.
4. (Optional) If you plan to assign IAM roles to all of your Kubernetes service accounts so that pods only have the minimum permissions that they need, and no pods in the cluster require access to the Amazon EC2 instance metadata service (IMDS) for other reasons, such as retrieving the current Region, then we recommend blocking pod access to IMDS. For more information, see the section called "IAM roles for service accounts" (p. 345) and the section called "Restricting access to the instance profile" (p. 359).

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 provide a complete end-to-end walkthrough for creating 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 considerations (p. 208) and Amazon EKS security group considerations (p. 210). The Getting started with Amazon EKS (p. 4) guide creates a VPC that meets the requirements, or you can also follow Creating a VPC for your Amazon EKS cluster (p. 204) to create one manually.
- An existing Amazon EKS cluster that uses a VPC and security group that meet the requirements of an Amazon EKS cluster. For more information, see Creating an Amazon EKS cluster (p. 17). 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.

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 will fail to register with the cluster and you will have to relaunch them.
2. Open the AWS CloudFormation console at https://console.aws.amazon.com/cloudformation
3. Choose **Create stack**.
4. For **Specify template**, select **Amazon S3 URL**, 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, fill out the following parameters accordingly:

   - **Stack name**: Choose a stack name for your AWS CloudFormation stack. For example, you can call it **<cluster-name>-nodes**.
   - **ClusterName**: Enter the name that you used when you created your Amazon EKS cluster.

     Important
     This name must exactly match the name you used in Step 1: Create your Amazon EKS cluster (p. 9); otherwise, your nodes cannot join the cluster.
   - **ClusterControlPlaneSecurityGroup**: Choose the **SecurityGroups** value from the AWS CloudFormation output that you generated when you created your VPC (p. 204).
   - **NodeGroupName**: Enter a name for your node group. This name can be used later to identify the Auto Scaling node group that is created for your 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.

     Note
     The supported instance types for the latest version of the Amazon VPC CNI plugin for Kubernetes are shown here. You may need to update your CNI version to take advantage of the latest supported instance types. For more information, see Amazon VPC CNI plugin for Kubernetes upgrades (p. 236).
   - **NodeImageIdSSMParam**: Pre-populated with the Amazon EC2 Systems Manager parameter of the current recommended Amazon EKS optimized Windows Core AMI ID. If you want to use the full version of Windows, then replace <Core> with `Full`.
   - **NodeImageId**: (Optional) If you are using your own custom AMI (instead of the Amazon EKS optimized AMI), enter a node AMI ID for your Region. If you specify a value here, it overrides any values in the **NodeImageIdSSMParam** field.
   - **NodeVolumeSize**: Specify a root volume size for your 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 Windows Instances*.

     Note
     If you do not 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 using `-KubeletExtraArgs`.

- **DisableIMDSv1**: Each node supports the Instance Metadata Service Version 1 (IMDSv1) and IMDSv2 by default, but you can disable IMDSv1. Select **true** if you don't want any nodes in the node group, or any pods scheduled on the nodes in the node group to use IMDSv1. For more information about IMDS, see Configuring the instance metadata service.
- **VpcId**: Select the ID for the VPC (p. 204) that you created.
- **NodeSecurityGroups**: Select the security group that was created for your Linux node group when you created your VPC (p. 204). If your Linux nodes have more than one security group attached to them (for example, if the Linux node group was created with `eksctl`), specify all of them here.
- **Subnets**: Choose the subnets that you created. If you created your VPC using the steps described at Creating a VPC for your Amazon EKS cluster (p. 204), 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 is not enabled for the public subnet, then any nodes that you deploy to that public subnet will not be assigned a public IP address and will not be able to communicate with the cluster or other AWS services. If the subnet was deployed before March 26, 2020 using either of the Amazon EKS AWS CloudFormation VPC templates (p. 204), 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 is able to communicate with the cluster and other AWS services through a NAT gateway.
  - If the subnets do not have internet access, then make sure that you're aware of the considerations and extra steps in Private clusters (p. 80).
  - 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`* (including *<>*) with the name of your cluster and 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, then you must manually add the tag to the subnet. For more information, see the section called “Subnet tagging” (p. 210).
  - 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.

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 with your favorite 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
Do not modify any other lines in this file.

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: aws-auth
  namespace: kube-system
data:
  mapRoles: |
    - rolearn: <ARN of instance role (not instance profile) of **Linux** 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 may 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. 375) in the troubleshooting section.

If nodes fail to join the cluster, then see Nodes fail to join cluster (p. 374) 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. 78) to test your cluster and Windows nodes.
4. (Optional) If the **AmazonEKS_CNI_Policy** managed IAM policy is attached to your the section called "Node IAM role" (p. 341), 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 VPC CNI plugin to use IAM roles for service accounts (p. 217).
5. (Optional) If you plan to assign IAM roles to all of your Kubernetes service accounts so that pods only have the minimum permissions that they need, and no pods in the cluster require access to the Amazon EC2 instance metadata service (IMDS) for other reasons, such as retrieving the current Region, then we recommend blocking pod access to IMDS. For more information, see the section called "IAM roles for service accounts" (p. 345) and the section called "Restricting access to the instance profile" (p. 359).

Self-managed node updates

When a new Amazon EKS optimized AMI is released, you should 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, you should also 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. 88), see Updating a managed node group (p. 97).

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. 117) – 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, 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. 122) – Update the AWS CloudFormation stack for an existing node group to use the new AMI. This method is not supported for node groups that were created with `eksctl`.

Migrating to a new node group

This topic helps you to create a new node group, gracefully migrate your existing applications to the new group, and then 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 `0.43.0` or later. You can check your version with the following command:

```
eksctl version
```

For more information on installing or upgrading `eksctl`, see Installing or upgrading `eksctl` (p. 308).

> **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>` (including `<>`) with your cluster name.

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

   Output:

   ```
   CLUSTER      NODEGROUP          CREATED               MIN SIZE      MAX SIZE
    DESIRED CAPACITY     INSTANCE TYPE     IMAGE ID
   default      standard-nodes    2019-05-01T22:26:58Z  1             4            3
                t3.medium          ami-05a71d034119ffc12
   ```

2. Launch a new node group with `eksctl` with the following command, replacing the *`<example values>`* (including `<>`)with your own values. The version number can't be later than your control plane's Kubernetes version and can't be more than two minor versions earlier than your control plane's Kubernetes version, though we recommend that you use the same version as your control plane. If you plan to assign IAM roles to all of your Kubernetes service accounts so that pods only have the minimum permissions that they need, and no pods in the cluster require access to the

Amazon EC2 instance metadata service (IMDS) for other reasons, such as retrieving the current Region, then we recommend blocking pod access to IMDS. For more information, see the section called "IAM roles for service accounts" (p. 345) and the section called "Restricting access to the instance profile" (p. 359). If you want to block pod access to IMDS, then 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 <1.19> \
  --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, replacing the `<example values>` (including <>) 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 outlined in Launching self-managed Amazon Linux nodes (p. 105).
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 have attached any additional IAM policies to your old node group IAM role, such as adding permissions for the Kubernetes Cluster Autoscaler, you should attach those same policies to your new node group IAM role to maintain that functionality on the new group.
4. Update the security groups for both node groups so that they can communicate with each other. For more information, see Amazon EKS security group considerations (p. 210).

   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 the `<example values>` (including <>) with your own.

```
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 ingress 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 are migrating your workload to the new group.

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

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

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

   Add a new `mapRoles` entry for the new node group.

```
apiVersion: v1
data:
  mapRoles: |
    - rolearn: <ARN of instance role (not instance profile)>
      username: system:node:{{EC2PrivateDNSName}}
      groups:
        - system:bootstrappers
        - system:nodes>
    - rolearn: <arn:aws:iam::111122223333:role/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. 118), 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 are using the Kubernetes Cluster Autoscaler, scale the deployment down to 0 replicas to avoid conflicting scaling actions.

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

8. Use the following command to taint each of the nodes that you want to remove with `NoSchedule` so that new pods are not scheduled or rescheduled on the nodes you are replacing:

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

If you are upgrading your nodes to a new Kubernetes version, you can identify and taint all of the nodes of a particular Kubernetes version (in this case, 1.17) with the following code snippet. The version number can't be later than your control plane's Kubernetes version and can't be more than two minor versions earlier than your control plane's Kubernetes version, though we recommend that you use the same version as your control plane.

```
K8S_VERSION=<1.17>
nodes=$(kubectl get nodes -o jsonpath="{.items[?(@.status.nodeInfo.kubeletVersion==\"v
$K8S_VERSION\")].metadata.name}")
for node in ${nodes[@]}
do
    echo "Tainting $node"
    kubectl taint nodes $node key=value:NoSchedule
done
```

9. Determine your cluster's DNS provider.

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

Output (this cluster is using `kube-dns` for DNS resolution, but your cluster may return `coredns` instead):

```
NAME       DESIRED   CURRENT   UP-TO-DATE   AVAILABLE   AGE
<kube-dns>    1         1         1            1           31m
```

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

```
kubectl scale deployments/<kube-dns> --replicas=2 -n kube-system
```

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

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

If you are upgrading your nodes to a new Kubernetes version, you can identify and drain all of the nodes of a particular Kubernetes version (in this case, *1.17*) with the following code snippet.

```
K8S_VERSION=<1.17>
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 have finished draining, revoke the security group ingress rules you authorized earlier, and then delete the AWS CloudFormation stack to terminate the instances.

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

a. Revoke the ingress rules that you created for your 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 **Actions**, then **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.

```
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/<YOUR CLUSTER NAME>`) and update your Cluster Autoscaler deployment's command to point to the newly tagged Auto Scaling group. For more information, see Cluster Autoscaler on AWS.

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

15. (Optional) Verify that you are using the latest version of the Amazon VPC CNI plugin for Kubernetes. You may need to update your CNI version to take advantage of the latest supported instance types. For more information, see Amazon VPC CNI plugin for Kubernetes upgrades (p. 236).
16. If your cluster is using `kube-dns` for DNS resolution (see previous step (p. 120)), 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 helps you to 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, or 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. If you are using Managed node groups (p. 88), see Updating a managed node group (p. 97).

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 is not 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. 117).

To update an existing node group

1. Determine your cluster's DNS provider.

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

   Output (this cluster is using `kube-dns` for DNS resolution, but your cluster may return `coredns` instead):

```
NAME       DESIRED   CURRENT   UP-TO-DATE   AVAILABLE   AGE
<kube-dns>    1          1          1            1           31m
```

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

```
kubectl scale deployments/<kube-dns> --replicas=2 -n kube-system
```

3. (Optional) If you are using the Kubernetes Cluster Autoscaler, scale the deployment down to zero replicas to avoid conflicting scaling actions.

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

4. Determine the instance type and desired instance count of your current node group. You will enter these values later when you update the AWS CloudFormation template for the group.

   a. Open the Amazon EC2 console at https://console.aws.amazon.com/ec2/.
   b. Choose **Launch Configurations** in the left navigation, and note the instance type for your existing node launch configuration.

   c. Choose **Auto Scaling Groups** in the left navigation 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 are using the latest version of the node AWS CloudFormation template, and 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. 122), or enter a new desired number of nodes to scale to when your stack is updated.
   - **NodeAutoScalingGroupMaxSize** – Enter the maximum number of nodes to which your node Auto Scaling group can scale out. **This value must be at least one node greater than your desired capacity so that you can perform a rolling update of your nodes without reducing your node count during the update.**
   - **NodeInstanceType** – Choose the instance type your recorded in a previous step (p. 122), or choose a different instance type for your nodes. Each Amazon EC2 instance type supports a maximum number of elastic network interfaces (ENIs) and each ENI supports a maximum number of IP addresses. Since 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. Some instance types might not be available in all Regions.

     Note
     The supported instance types for the latest version of the Amazon VPC CNI plugin for Kubernetes are shown here. You may need to update your CNI version to take advantage of the latest supported instance types. For more information, see Amazon VPC CNI plugin for Kubernetes upgrades (p. 236).

     Important
     Some instance types might not be available in all Regions.
   - **NodeImageIdSSMParam** – The Amazon EC2 Systems Manager parameter of the AMI ID that you want to update to. The following value uses the latest Amazon EKS optimized AMI for Kubernetes version 1.19.

```
/aws/service/eks/optimized-ami/<1.19>/<amazon-linux-2>/recommended/image_id
```

     You can replace *`<1.19>`* (including *`<>`*) with a supported Kubernetes version (p. 65) that is the same, or 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 lookup and specify an AMI ID. If your AWS CloudFormation stack is using this value, any stack update will always launch the latest recommended Amazon EKS optimized AMI for your specified Kubernetes version, even if you don't change any values in the template.

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

  Important
  This value overrides any value specified for **NodeImageIdSSMParam**. If you want to use the **NodeImageIdSSMParam** value, ensure that the value for **NodeImageId** is blank.
- **DisableIMDSv1** – Each node supports the Instance Metadata Service Version 1 (IMDSv1) and IMDSv2 by default, but you can disable IMDSv1. Select **true** if you don't want any nodes in the node group, or any pods scheduled on the nodes 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, have assigned necessary permissions directly to all pods that require access to AWS services, and no pods in your cluster require access to IMDS for other reasons, such as retrieving the current Region, then you can also disable access to IMDSv2 for pods that don't use host networking. For more information, see the section called "Restricting access to the instance profile" (p. 359).

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 are using the latest version of the Amazon VPC CNI plugin for Kubernetes. You may need to update your CNI version to take advantage of the latest supported instance types. For more information, see Amazon VPC CNI plugin for Kubernetes upgrades (p. 236).

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 no longer have to provision, configure, or scale groups of virtual machines to run containers. This removes the need to choose server types, decide when to scale your node groups, or optimize cluster packing. You can control which pods start on Fargate and how they run with Fargate profiles (p. 129), which are defined as part of your Amazon EKS cluster.

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

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

AWS Fargate considerations

Here's 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 running on Fargate has its own isolation boundary and does not share the underlying kernel, CPU resources, memory resources, or elastic network interface with another pod.
- Network Load Balancers can be used with IP targets only. You can also use AWS Application Load Balancers with Fargate. For more information, see the section called "Load balancer – IP targets" (p. 269) and the section called "Application load balancing" (p. 271).
- Pods must match a Fargate profile at the time that they are scheduled in order to run on Fargate. Pods which do not match a Fargate profile may be stuck as `Pending`. If a matching Fargate profile exists, you can delete pending pods that you have created to reschedule them onto Fargate.
- Daemonsets are not supported on Fargate. If your application requires a daemon, you should reconfigure that daemon to run as a sidecar container in your pods.
- Privileged containers are not supported on Fargate.
- Pods running on Fargate cannot specify `HostPort` or `HostNetwork` in the pod manifest.
- GPUs are currently not available on Fargate.
- You cannot use Security groups for pods (p. 219) with pods running on Fargate.
- Pods running on Fargate are only supported on private subnets (with NAT gateway access to AWS services, but not a direct route to an Internet Gateway), so your cluster's VPC must have private subnets available. For clusters without outbound internet access, see Private clusters (p. 80).
- You can use the Vertical Pod Autoscaler (p. 259) to initially right size the CPU and memory for your Fargate pods, and then use the Horizontal Pod Autoscaler (p. 263) to scale those pods. If you want the Vertical Pod Autoscaler to automatically re-deploy pods to Fargate with larger CPU and memory combinations, then set the Vertical Pod Autoscaler's mode 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.
- Fargate runs each pod in a VM-isolated environment without sharing resources with other pods. However, because Kubernetes is a single-tenant orchestrator, Fargate cannot guarantee pod-level security isolation. You should run sensitive workloads or untrusted workloads that need complete security isolation using separate Amazon EKS clusters.
- Fargate profiles support specifying subnets from VPC secondary CIDR blocks. You may want to specify a secondary CIDR block because there are a limited number of IP addresses available in a subnet. As a result, there are a limited 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 more information, see Adding IPv4 CIDR blocks to a VPC.
- The Amazon EC2 instance metadata service (IMDS) is not available to pods deployed to Fargate nodes. If you have pods deployed to Fargate that need IAM credentials, assign them to your pods using the section called "IAM roles for service accounts" (p. 345). If your pods need access to other information available through IMDS, such as the Region or Availability Zone that a pod is deployed to, then you must hard code this information into your pod spec.
- You can't deploy Fargate pods to AWS Outposts, AWS Wavelength or AWS Local Zones.

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 your cluster's public endpoint using CIDR blocks, it is recommended that you also enable private endpoint access so that Fargate pods can communicate with the cluster. Without the private endpoint enabled, the CIDR blocks that you specify for public access must include the egress sources from your VPC. For more information, see Amazon EKS cluster endpoint access control (p. 40).

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 do not 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 are working with a new cluster with no nodes, or a cluster with only managed node groups (p. 88), you can skip to Create a Fargate pod execution role (p. 126).

If you are working with an existing cluster that already has nodes associated with it, you need to make sure that pods on these nodes can communicate freely with pods running on Fargate. Pods running on Fargate are automatically configured to use the cluster security group for the cluster that they are associated with. You must ensure that any existing nodes in your cluster can send and receive traffic to and from the cluster security group. Managed node groups (p. 88) are automatically configured to use the cluster security group as well, so you do not need to modify or check them for this compatibility.

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

You can check for a cluster security group for your cluster in the AWS Management Console under the cluster's **Networking** section, or with the following AWS CLI command. Replace *<cluster_name>* (including <>) with the name of your cluster.

```
aws eks describe-cluster --name <cluster_name> --query
 cluster.resourcesVpcConfig.clusterSecurityGroupId
```

Create a Fargate pod execution role

When your cluster creates pods on AWS Fargate, the components running on the Fargate infrastructure need to make calls to AWS APIs on your behalf to do things like 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.

Note
If you created your cluster with `eksctl` using the `--fargate` option, then your cluster already has a pod execution role and you can skip ahead to Create a Fargate profile for your cluster (p. 127). Similarly, if you use `eksctl` to create your Fargate profiles, `eksctl` will create your pod execution role if one does not already exist.

When you create a Fargate profile, you must specify a pod execution role to use with your pods. This role is added to the cluster's Kubernetes Role based access control (RBAC) for authorization. This allows the `kubelet` that is running on the Fargate infrastructure to register with your Amazon EKS cluster so that it can appear in your cluster as a node. For more information, see Pod execution role (p. 343).

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. 345).

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

Create a Fargate profile for your cluster

Before you can schedule pods running on Fargate in your cluster, you must define a Fargate profile that specifies which pods should use Fargate when they are launched. For more information, see AWS Fargate profile (p. 129).

> **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 `0.43.0` or later. You can check your version with the following command:

```
eksctl version
```

For more information on installing or upgrading `eksctl`, see Installing or upgrading `eksctl` (p. 308).

eksctl

To create a Fargate profile with `eksctl`

Create your Fargate profile with the following `eksctl` command, replacing the `<variable text>` (including `<>`) with your own values. You must specify a namespace, but the `--labels` option is not required.

```
eksctl create fargateprofile \
    --cluster <cluster_name> \
    --name <fargate_profile_name> \
    --namespace <kubernetes_namespace> \
    --labels <key=value>
```

AWS Management Console

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

1. Open the Amazon EKS console at https://console.aws.amazon.com/eks/home#/clusters.
2. Choose the cluster to create a Fargate profile for.

3. Under **Fargate profiles**, choose **Add Fargate profile**.
4. On the **Configure Fargate profile** page, enter the following information and choose **Next**.

   a. For **Name**, enter a unique name for your Fargate profile.

   b. For **Pod execution role**, choose the pod execution role to use with your Fargate profile. Only IAM roles with the `eks-fargate-pods.amazonaws.com` service principal are shown. If you do not see any roles listed here, you must create one. For more information, see Pod execution role (p. 343).

   c. For **Subnets**, choose the subnets to use for your pods. By default, all subnets in your cluster's VPC are selected. Only private subnets are supported for pods running on Fargate; you must deselect any public subnets.

   d. For **Tags**, you can optionally tag your Fargate profile. These tags do not propagate to other resources associated with the profile, such as its pods.
5. On the **Configure pods selection** page, enter the following information and choose **Next**.

   a. For **Namespace**, enter a namespace to match for pods, such as `kube-system` or `default`.

   b. (Optional) Add Kubernetes labels to the selector that pods in the specified namespace must have to match the selector. For example, you could add the label `infrastructure: fargate` to the selector so that only pods in the specified namespace that also have the `infrastructure: fargate` Kubernetes label match the selector.
6. On the **Review and create** page, review the information for your Fargate profile and choose **Create**.

(Optional) Update CoreDNS

By default, CoreDNS is configured to run on Amazon EC2 infrastructure on Amazon EKS clusters. If you want to *only* run your pods on Fargate in your cluster, you need to modify the CoreDNS deployment to remove the `eks.amazonaws.com/compute-type : ec2` annotation. You would also need to create a Fargate profile to target the CoreDNS pods. The following Fargate profile JSON file does this.

Note
If you created your cluster with `eksctl` using the `--fargate` option, then `coredns` has already been patched to run on Fargate and you can skip ahead to Next steps (p. 129).

```
{
    "fargateProfileName": "coredns",
    "clusterName": "<dev>",
    "podExecutionRoleArn": "<arn:aws:iam::111122223333:role/
AmazonEKSFargatePodExecutionRole>",
    "subnets": [
        "subnet-<0b64dd020cdff3864>",
        "subnet-<00b03756df55e2b87>",
        "subnet-<0418fcb68ed294abf>"
    ],
    "selectors": [
        {
            "namespace": "kube-system",
            "labels": {
                "k8s-app": "kube-dns"
            }
        }
    ]
}
```

You could apply this Fargate profile to your cluster with the following AWS CLI command. First, create a file called `coredns.json` and paste the JSON file from the previous step into it, replacing the <variable text> with your own cluster values.

```
aws eks create-fargate-profile --cli-input-json file://coredns.json
```

Then, use the following `kubectl` command to remove the `eks.amazonaws.com/compute-type : ec2` annotation from the CoreDNS pods.

```
kubectl patch deployment coredns \
    -n kube-system \
    --type json \
    -p='[{"op": "remove", "path": "/spec/template/metadata/annotations/
eks.amazonaws.com~1compute-type"}]'
```

Next steps

- You can start migrating your existing applications to run on Fargate with the following workflow.
  1. Create a Fargate profile (p. 130) 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. 271) to allow Ingress objects for your pods running on Fargate.
- You can use the Vertical Pod Autoscaler (p. 259) to initially right size the CPU and memory for your Fargate pods, and then use the Horizontal Pod Autoscaler (p. 263) to scale those pods. If you want the Vertical Pod Autoscaler to automatically re-deploy pods to Fargate with larger CPU and memory combinations, then set the Vertical Pod Autoscaler's mode to either `Auto` or `Recreate` to ensure correct functionality. For more information, see the Vertical Pod Autoscaler documentation on GitHub.

AWS Fargate profile

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

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

If a pod matches multiple Fargate profiles, Amazon EKS picks one of the matches at random. In this case, you can specify which profile a pod should use by adding the following Kubernetes label to the pod specification: `eks.amazonaws.com/fargate-profile: <profile_name>`. However, the pod must still match a selector in that profile in order to be scheduled onto Fargate.

When you create a Fargate profile, you must specify a pod execution role for the 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 is running on the Fargate infrastructure can register with your Amazon EKS cluster and appear in your cluster as a node. The pod execution role also provides IAM permissions to the Fargate infrastructure to allow read access to Amazon ECR image repositories. For more information, see Pod execution role (p. 343).

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

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

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

Fargate profile components

The following components are contained in a Fargate profile.

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

  When you create a Fargate profile, you must specify a pod execution role to use with your pods. This role is added to the cluster's Kubernetes Role Based Access Control (RBAC) for authorization, so that the `kubelet` that is running on the Fargate infrastructure can register with your Amazon EKS cluster and appear in your cluster as a node. For more information, see Pod execution role (p. 343).
- **Subnets** – The IDs of subnets to launch pods into that use this profile. At this time, pods running on Fargate are not assigned public IP addresses, so only private subnets (with no direct route to an Internet Gateway) are accepted for this parameter.
- **Selectors** – The selectors to match for pods to use this Fargate profile. Each selector must have an associated namespace. Optionally, you can also specify labels for a namespace. You may specify up to five selectors in a Fargate profile. A pod only needs to match one selector to run using the Fargate profile.
- **Namespace** – You must specify a namespace for a selector. The selector only matches pods that are created in this namespace, but you can create multiple selectors to target multiple namespaces.
- **Labels** – You can optionally specify Kubernetes labels to match for the selector. The selector only matches pods that have all of the labels that are specified in the selector.

Creating a Fargate profile

This topic helps you to create a Fargate profile. 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 Pod execution role (p. 343). Pods running on Fargate are only supported on private subnets (with NAT gateway access to AWS services, but not a direct route to an Internet Gateway), so your cluster's VPC must have private subnets available. 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 `0.43.0` or later. You can check your version with the following command:

```
eksctl version
```

For more information on installing or upgrading `eksctl`, see Installing or upgrading `eksctl` (p. 308).

eksctl

To create a Fargate profile with `eksctl`

Create your Fargate profile with the following `eksctl` command, replacing the `<variable text>` (including <>) with your own values. You must specify a namespace, but the `--labels` option is not required.

```
eksctl create fargateprofile --cluster <cluster_name> --name <fargate_profile_name> --
namespace <kubernetes_namespace> --labels <key=value>
```

AWS Management Console

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

1. Open the Amazon EKS console at https://console.aws.amazon.com/eks/home#/clusters.
2. Choose the cluster to create a Fargate profile for.
3. Click on the **Configuration** tab, then the **Compute** tab.
4. Under **Fargate profiles**, choose **Add Fargate profile**.
5. On the **Configure Fargate profile** page, enter the following information and choose **Next**.

   a. For **Name**, enter a unique name for your Fargate profile.
   b. For **Pod execution role**, choose the pod execution role to use with your Fargate profile. Only IAM roles with the `eks-fargate-pods.amazonaws.com` service principal are shown. If you do not see any roles listed here, you must create one. For more information, see Pod execution role (p. 343).
   c. For **Subnets**, choose the subnets to use for your pods. By default, all subnets in your cluster's VPC are selected. Only private subnets are supported for pods running on Fargate; you must deselect any public subnets.
   d. For **Tags**, you can optionally tag your Fargate profile. These tags do not propagate to other resources associated with the profile, such as its pods.
6. On the **Configure pods selection** page, enter the following information and choose **Next**.

   a. For **Namespace**, enter a namespace to match for pods, such as `kube-system` or `default`.
   b. (Optional) Add Kubernetes labels to the selector that pods in the specified namespace must have to match the selector. For example, you could add the label `infrastructure: fargate` to the selector so that only pods in the specified namespace that also have the `infrastructure: fargate` Kubernetes label match the selector.
7. On the **Review and create** page, review the information for your Fargate profile and choose **Create**.

Deleting a Fargate profile

This topic helps you to delete a Fargate profile.

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

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

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

Use the following command to delete a profile from a cluster. Replace the *<example values>* (including <>) with your own.

```
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. Choose the cluster that you want to delete the Fargate profile from.
3. Choose the **Configuration** tab, then choose the **Compute** tab.
4. Choose the Fargate profile to delete and then choose **Delete**.
5. On the **Delete *<cluster_name>*** page, type the name of the cluster and choose **Confirm** to delete.

AWS CLI

Use the following command to delete a profile from a cluster. Replace the *<example values>* (including <>) with your own.

```
aws eks delete fargateprofile --faragate-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

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

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

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

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

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

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

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

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>` (including `<>`) with the name of your pod.

```
kubectl describe pod <pod-name>
```

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 on 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 5/28/2020, 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 running on Fargate is less than 20GB because some space is used by the `kubelet` and other Kubernetes modules that are loaded inside the pod.

AWS Fargate usage metrics

You can use CloudWatch usage metrics to provide visibility into your accounts 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. 322).

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 `Fargate`.
`Type`	The type of entity that is being reported. Currently, the only valid value for AWS Fargate usage metrics is `Resource`.
`Resource`	The type of resource that is running. Currently, AWS Fargate returns information on your Fargate On-Demand usage. The resource value for Fargate On-Demand usage is `OnDemand`. **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 `FARGATE` capacity provider.
`Class`	The class of resource being tracked. Currently, AWS Fargate does not 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 and configure alarms that alert you when your usage approaches a service quota. For more information, see AWS Fargate usage metrics (p. 133).

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 navigation pane, choose **AWS services**.
3. From the **AWS services** list, search for and select **AWS Fargate**.
4. In the **Service quotas** list, select the Fargate usage quota you want to create an alarm for.
5. In the Amazon CloudWatch Events 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 with Fargate supports a built-in log router, which means there are no sidecar containers to install or maintain. 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 Elasticsearch Service, and Amazon Kinesis Data Firehose destinations such as Amazon S3, Amazon Kinesis Data Streams, and partner tools. 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.

Prerequisites

- An existing Amazon EKS cluster. The cluster must be running one of the following (or later) platform versions.

EKS version	Platform version
1.19	`eks.1`
1.18	`eks.4`
1.17	`eks.5`
1.16	`eks.5`
1.15	`eks.6`

  If you don't have an existing cluster at one of these platform versions, or later, see *Getting started with Amazon EKS* (p. 4) to deploy a cluster.
- An existing Fargate profile that specifies an existing Kubernetes namespace that you deploy Fargate pods to. For more information, see the section called "Create a Fargate profile for your cluster" (p. 127).

To send logs to a destination of your choice

Apply a `ConfigMap` to your Amazon EKS cluster with a `Fluent Conf` data value that defines where container logs are shipped to. `Fluent Conf` is Fluent Bit, which is a fast and lightweight log processor configuration language that is used to route container logs to a log destination of your choice. For more information, see Configuration File in the Fluent Bit documentation.

In the following steps, replace the `<example values>` with your own values.

1. Create a Kubernetes namespace.

   a. Save the following contents to a file named `aws-observability-namespace.yaml` on your computer. The name must be `aws-observability`.

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

   b. Create the namespace.

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

2. Configure Kubernetes to send Fargate logs to one of the following destinations. The `ConfigMap` that you create must be created in the `aws-observability` namespace.

   a. (Optional) Send logs to CloudWatch. You have two output options when using CloudWatch:

      - `cloudwatch_logs` – An output plugin written in C.
      - `cloudwatch` – An output plugin written in Golang.

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

      i. Save the following contents to a file named `aws-logging-cloudwatch-configmap.yaml`.

```
kind: ConfigMap
apiVersion: v1
metadata:
  name: aws-logging
  namespace: aws-observability
  labels:
data:
  output.conf: |
    [OUTPUT]
        Name cloudwatch_logs
        Match   *
        region <us-east-1>
        log_group_name fluent-bit-cloudwatch
        log_stream_prefix from-fluent-bit-
        auto_create_group true
```

      ii. Apply the manifest to your cluster.

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

      iii. 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
```

   b. (Optional) Send logs to Amazon Elasticsearch 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 Elasticsearch.

      i. Save the following contents to a file named `aws-logging-elasticsearch-configmap.yaml`.

```
kind: ConfigMap
apiVersion: v1
metadata:
  name: aws-logging
  namespace: aws-observability
  labels:
data:
  output.conf: |
    [OUTPUT]
```

```
      Name  es
      Match *
      Host  192.168.2.3
      Port  9200
      Index my_index
      Type  my_type
      AWS_Auth On
      AWS_Region <us-east-1>
```

ii. Apply the manifest to your cluster.

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

iii. Download the Elasticsearch 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
```

c. (Optional) Send logs to Kinesis Data Firehose. You have two output options when using 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.

i. Save the following contents to a file named `aws-logging-firehose-configmap.yaml`.

```
kind: ConfigMap
apiVersion: v1
metadata:
  name: aws-logging
  namespace: aws-observability
  labels:
data:
  output.conf: |
    [OUTPUT]
     Name  kinesis_firehose
     Match *
     region <us-east-1>
     delivery_stream my-stream-firehose
```

ii. Apply the manifest to your cluster.

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

iii. 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.

```
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.

   ```
   aws iam attach-role-policy \
     --policy-arn arn:aws:iam::<111122223333>:policy/<eks-fargate-logging-policy> \
     --role-name <your-pod-execution-role>
   ```

5. Deploy a sample pod.

   a. Save the following contents to a `yaml` file 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 <name-of-file-from-previous-step>.yaml
      ```

6. View your logs using the tool that you sent your logs to.

Size considerations

We suggest that you plan for up to 50 MB of memory for your logs. 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 is 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.

Validation strategy

The main sections included in a typical `Fluent Conf` are `Service`, `Input`, `Filter`, and `Output`. `Filter`, `Output`, and `Parser`. `Service` and `Input` are generated by Fargate. Fargate only validates the `Filter`, `Output`, and `Parser` specified in the `Fluent Conf`. Any sections provided other than `Filter`, `Output`, and `Parser` are ignored. The following rules are used to validate the `Filter`, `Output`, and `Parser` fields.

1. `[FILTER]`, `[OUTPUT]`, and `[PARSER]` are supposed to be specified under each corresponding key : `filters.conf`, `output.conf`, and `parsers.conf`. For example, `[FILTER]` must be under `filters.conf`. You can have many `[FILTER]`s under `filters.conf`. The same is true for `[OUTPUT]` and `[PARSER]`.
2. 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.
3. Environment variables such as `${ENV_VAR}` are not allowed in the `configmap`.
4. 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.
5. Fargate validates against the following supported filters: `grep`, `parser`, `record_modifier`, `rewrite_tag`, `throttle`, `nest`, and `modify`.
6. Fargate validates against the following supported output: `es`, `firehose`, `kinesis_firehose`, `cloudwatch`, `cloudwatch_logs`, and `kinesis`.
7. At least one supported `Output` plugin has to be provided in the `ConfigMap` to enable logging. `Filter` and `Parser` are not required to enable logging.

For more information about `Fluent Conf` see Configuration File in the Fluent Bit documentation. You can also run Fluent Bit on Amazon EC2 using the desired configuration to troubleshoot any issues that arise from validation.

Amazon EKS optimized AMIs

You can deploy nodes with pre-built Amazon EKS optimized Amazon Machine Images (AMI), or your own custom AMIs. For more information about each type of Amazon EKS optimized AMI, see one of the following topics. For more information about creating your own custom AMI, see Amazon EKS optimized Amazon Linux AMI build script (p. 158).

Topics

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. 148) and Amazon EKS optimized Arm Amazon Linux AMIs (p. 149).

Select a link in one of the following tables to view the latest Amazon EKS optimized Amazon Linux 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 Amazon Linux AMI IDs (p. 157).

1.19.6

Kubernetes version 1.19.6

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

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
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

1.18.9

Kubernetes version 1.18.9

Region	x86	x86 accelerated	Arm
US East (Ohio) (`us-east-2`)	View AMI ID	View AMI ID	View AMI ID

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
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

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

1.17.12

Kubernetes version 1.17.12

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

Region	x86	x86 accelerated	Arm
Asia Pacific (Sydney) (`ap-southeast-2`)	View AMI ID	View AMI ID	View AMI ID
Canada (Central) (`ca-central-1`)	View AMI ID	View AMI ID	View AMI ID
China (Beijing) (`cn-north-1`)	View AMI ID	View AMI ID	View AMI ID
China (Ningxia) (`cn-northwest-1`)	View AMI ID	View AMI ID	View AMI ID
Europe (Frankfurt) (`eu-central-1`)	View AMI ID	View AMI ID	View AMI ID
Europe (Ireland) (`eu-west-1`)	View AMI ID	View AMI ID	View AMI ID
Europe (London) (`eu-west-2`)	View AMI ID	View AMI ID	View AMI ID
Europe (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

1.16.15

Kubernetes version 1.16.15

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

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
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

Region	x86	x86 accelerated	Arm
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

1.15.12

Kubernetes version 1.15.12

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
Asia Pacific (Sydney) (`ap-southeast-2`)	View AMI ID	View AMI ID	View AMI ID

Region	x86	x86 accelerated	Arm
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; they will fail to join your cluster. Be sure to update any existing AWS CloudFormation node stacks with the latest template (URL shown below) 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. 105).

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, and is 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. Be 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 end 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. 283).

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/v0.8.0/
nvidia-device-plugin.yml
```

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
```

Output:

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

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

Amazon EKS optimized 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

- You can only deploy Arm AMIs in 1.15 or later clusters.
- If your cluster was deployed before August 17, 2020, then you must do a one-time upgrade of critical cluster add-on manifests 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 Update an existing cluster (p. 25). 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. 197) with Arm.
- If you have any DaemonSets 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 ensure 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. 158) 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. 157).

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 tables below list the current and previous versions of the Amazon EKS optimized Amazon Linux AMI.

Kubernetes version 1.19

Kubernetes version 1.19

AMI version	`kubelet` version	Docker version	Kernel version	Packer version
1.19.6-20210329	1.19.6	19.03.13-ce-1	5.4.105	v20210329
1.19.6-20210322	1.19.6	19.03.13-ce-1	5.4.95	v20210322
1.19.6-20210310	1.19.6	19.03.13-ce-1	5.4.95	v20210310
1.19.6-20210302	1.19.6	19.03.13-ce-1	5.4.95	v20210302
1.19.6-20210208	1.19.6	19.03.6-ce-4	5.4.91	v20210208

Kubernetes version 1.18

Kubernetes version 1.18

AMI version	`kubelet` version	Docker version	Kernel version	Packer version
1.18.9-20210329	1.18.9	19.03.13-ce-1	4.14.225	v20210329
1.18.9-20210322	1.18.9	19.03.13-ce-1	4.14.225	v20210322
1.18.9-20210310	1.18.9	19.03.13-ce-1	4.14.219	v20210310
1.18.9-20210302	1.18.9	19.03.13-ce-1	4.14.219	v20210302
1.18.9-20210208	1.18.9	19.03.6-ce-4	4.14.214	v20210208
1.18.9-20210125	1.18.9	19.03.6-ce-4	4.14.209	v20210125

AMI version	`kubelet` version	Docker version	Kernel version	Packer version
1.18.9-20210112	1.18.9	19.03.6-ce-4	4.14.209	v20210112
1.18.9-20201211	1.18.9	19.03.6-ce-4	4.14.209	v20201211
1.18.9-20201126	1.18.9	19.03.6-ce-4	4.14.203	v20201126
1.18.9-20201117	1.18.9	19.03.6-ce-4	4.14.203	v20201117
1.18.9-20201112	1.18.9	19.03.6-ce-4	4.14.203	v20201112
1.18.8-20201007	1.18.8	19.03.6-ce-4	4.14.198	v20201007

Kubernetes version 1.17

Kubernetes version 1.17

AMI version	`kubelet` version	Docker version	Kernel version	Packer version
1.17.12-20210329	1.17.12	19.03.13-ce-1	4.14.225	v20210329
1.17.12-20210322	1.17.12	19.03.13-ce-1	4.14.225	v20210322
1.17.12-20210310	1.17.12	19.03.13-ce-1	4.14.219	v20210310
1.17.12-20210302	1.17.12	19.03.13-ce-1	4.14.219	v20210302
1.17.12-20210208	1.17.12	19.03.6-ce-4	4.14.214	v20210208
1.17.12-20210125	1.17.12	19.03.6-ce-4	4.14.209	v20210125
1.17.12-20210112	1.17.12	19.03.6-ce-4	4.14.209	v20210112
1.17.12-20201211	1.17.12	19.03.6-ce-4	4.14.209	v20201211
1.17.12-20201126	1.17.12	19.03.6-ce-4	4.14.203	v20201126
1.17.12-20201117	1.17.12	19.03.6-ce-4	4.14.203	v20201117
1.17.12-20201112	1.17.12	19.03.6-ce-4	4.14.203	v20201112
1.17.11-20201007	1.17.11	19.03.6-ce-4	4.14.198	v20201007
1.17.11-20201002	1.17.11	19.03.6-ce-4	4.14.198	v20201002
1.17.11-20200921	1.17.11	19.03.6-ce-4	4.14.193	v20200921
1.17.9-20200904	1.17.9	19.03.6-ce-4	4.14.193	v20200904
1.17.9-20200814	1.17.9	19.03.6-ce-4	4.14.186	v20200814
1.17.9-20200723	1.17.9	19.03.6-ce	4.14.181	v20200723
1.17.7-20200710	1.17.7	19.03.6-ce	4.14.181	v20200710
1.17.7-20200709	1.17.7	19.03.6-ce	4.14.181	v20200709

Kubernetes version 1.16

Kubernetes version 1.16

AMI version	`kubelet` version	Docker version	Kernel version	Packer version
1.16.15-20210329	1.16.15	19.03.13-ce-1	4.14.225	v20210329
1.16.15-20210322	1.16.15	19.03.13-ce-1	4.14.225	v20210322
1.16.15-20210310	1.16.15	19.03.13-ce-1	4.14.219	v20210310
1.16.15-20210302	1.16.15	19.03.13-ce-1	4.14.219	v20210302
1.16.15-20210208	1.16.15	19.03.6-ce-4	4.14.214	v20210208
1.16.15-20210125	1.16.15	19.03.6-ce-4	4.14.209	v20210125
1.16.15-20210112	1.16.15	19.03.6-ce-4	4.14.209	v20210112
1.16.15-20201211	1.16.15	19.03.6-ce-4	4.14.209	v20201211
1.16.15-20201126	1.16.15	19.03.6-ce-4	4.14.203	v20201126
1.16.15-20201117	1.16.15	19.03.6-ce-4	4.14.203	v20201117
1.16.15-20201112	1.16.15	19.03.6-ce-4	4.14.203	v20201112
1.16.13-20201007	1.16.13	19.03.6-ce-4	4.14.198	v20201007
1.16.13-20201002	1.16.13	19.03.6-ce-4	4.14.198	v20201002
1.16.13-20200921	1.16.13	19.03.6-ce-4	4.14.193	v20200921
1.16.13-20200904	1.16.13	19.03.6-ce-4	4.14.193	v20200904
1.16.13-20200904	1.16.13	19.03.6-ce-4	4.14.193	v20200904
1.16.13-20200814	1.16.13	19.03.6-ce-4	4.14.186	v20200814
1.16.13-20200723	1.16.13	19.03.6-ce	4.14.181	v20200723
1.16.12-20200710	1.16.12	19.03.6-ce	4.14.181	v20200710
1.16.12-20200709	1.16.12	19.03.6-ce	4.14.181	v20200709
1.16.8-20200615	1.16.8	19.03.6-ce	4.14.181	v20200615
1.16.8-20200609	1.16.8	19.03.6-ce	4.14.181	v20200609
1.16.8-20200531	1.16.8	18.09.9-ce	4.14.177	v20200531
1.16.8-20200507	1.16.8	18.09.9-ce	4.14.177	v20200507
1.16.8-20200423	1.16.8	18.09.9-ce	4.14.173	v20200423

Kubernetes version 1.15

Kubernetes version 1.15

AMI version	`kubelet` version	Docker version	Kernel version	Packer version
1.15.12-20210329	1.15.12	19.03.13-ce-1	4.14.225	v20210329
1.15.12-20210322	1.15.12	19.03.13-ce-1	4.14.225	v20210322
1.15.12-20210310	1.15.12	19.03.13-ce-1	4.14.219	v20210310
1.15.12-20210302	1.15.12	19.03.13-ce-1	4.14.219	v20210302
1.15.12-20210208	1.15.12	19.03.6-ce-4	4.14.214	v20210208
1.15.12-20210125	1.15.12	19.03.6-ce-4	4.14.209	v20210125
1.15.12-20210112	1.15.12	19.03.6-ce-4	4.14.209	v20210112
1.15.12-20201211	1.15.12	19.03.6-ce-4	4.14.209	v20201211
1.15.12-20201126	1.15.12	19.03.6-ce-4	4.14.203	v20201126
1.15.12-20201117	1.15.12	19.03.6-ce-4	4.14.203	v20201117
1.15.12-20201112	1.15.12	19.03.6-ce-4	4.14.203	v20201112
1.15.11-20201007	1.15.11	19.03.6-ce-4	4.14.198	v20201007
1.15.11-20201002	1.15.11	19.03.6-ce-4	4.14.198	v20201002
1.15.11-20200921	1.15.11	19.03.6-ce-4	4.14.193	v20200921
1.15.11-20200904	1.15.11	19.03.6-ce-4	4.14.193	v20200904
1.15.11-20200814	1.15.11	19.03.6-ce-4	4.14.186	v20200814
1.15.11-20200723	1.15.11	19.03.6-ce	4.14.181	v20200723
1.15.11-20200710	1.15.11	19.03.6-ce	4.14.181	v20200710
1.15.11-20200709	1.15.11	19.03.6-ce	4.14.181	v20200709
1.15.11-20200615	1.15.11	19.03.6-ce	4.14.181	v20200615
1.15.11-20200609	1.15.11	19.03.6-ce	4.14.181	v20200609
1.15.11-20200531	1.15.11	18.09.9-ce	4.14.177	v20200531
1.15.11-20200507	1.15.11	18.09.9-ce	4.14.177	v20200507
1.15.11-20200423	1.15.11	18.09.9-ce	4.14.173	v20200423
1.15.10-20200406	1.15.10	18.09.9-ce	4.14.173	v20200406
1.15.10-20200228	1.15.10	18.09.9-ce	4.14.165	v20200228

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 1.19

Kubernetes version 1.19

AMI version	`kubelet` version	Docker version	Kernel version	Packer version	NVIDIA driver version
1.18.9-20210329	1.19.6	19.03.13-ce-1	5.4.105	v20210329	460.32.03
1.18.9-20210322	1.19.6	19.03.13-ce-1	5.4.95	v20210322	460.27.04
1.18.9-20210310	1.19.6	19.03.13-ce-1	5.4.95	v20210310	460.27.04
1.18.9-20210302	1.19.6	19.03.13-ce-1	5.4.95	v20210302	450.51.06
1.18.9-20210208	1.19.6	19.03.6-ce-4	5.4.91	v20210208	450.51.06

Kubernetes version 1.18

Kubernetes version 1.18

AMI version	`kubelet` version	Docker version	Kernel version	Packer version	NVIDIA driver version
1.18.9-20210329	1.18.9	19.03.13-ce-1	4.14.225	v20210329	460.32.03
1.18.9-20210322	1.18.9	19.03.13-ce-1	4.14.225	v20210322	460.27.04
1.18.9-20210310	1.18.9	19.03.13-ce-1	4.14.219	v20210310	460.27.04
1.18.9-20210302	1.18.9	19.03.13-ce-1	4.14.219	v20210302	450.51.06
1.18.9-20210208	1.18.9	19.03.6-ce-4	4.14.214	v20210208	450.51.06
1.18.9-20210125	1.18.9	19.03.6-ce-4	4.14.209	v20210125	450.51.06
1.18.9-20210112	1.18.9	19.03.6-ce-4	4.14.209	v20210112	450.51.06
1.18.9-20201211	1.18.9	19.03.6-ce-4	4.14.209	v20201211	450.51.06
1.18.9-20201126	1.18.9	19.03.6-ce-4	4.14.203	v20201126	450.51.06
1.18.9-20201117	1.18.9	19.03.6-ce-4	4.14.203	v20201117	450.51.06
1.18.9-20201112	1.18.9	19.03.6-ce-4	4.14.203	v20201112	450.51.06
1.18.8-20201007	1.18.8	19.03.6-ce-4	4.14.198	v20201007	418.87.00

Kubernetes version 1.17

Kubernetes version 1.17

AMI version	`kubelet` version	Docker version	Kernel version	Packer version	NVIDIA driver version
1.17.12-20210329	1.17.12	19.03.13-ce-1	4.14.225	v20210329	460.32.03
1.17.12-20210322	1.17.12	19.03.13-ce-1	4.14.225	v20210322	460.27.04
1.17.12-20210310	1.17.12	19.03.13-ce-1	4.14.219	v20210310	460.27.04

AMI version	`kubelet` version	Docker version	Kernel version	Packer version	NVIDIA driver version
1.17.12-20210302	1.17.12	19.03.13-ce-1	4.14.219	v20210302	450.51.06
1.17.12-20210208	1.17.12	19.03.6-ce-4	4.14.214	v20210208	450.51.06
1.17.12-20210125	1.17.12	19.03.6-ce-4	4.14.209	v20210125	450.51.06
1.17.12-20210112	1.17.12	19.03.6-ce-4	4.14.209	v20210112	450.51.06
1.17.12-20201211	1.17.12	19.03.6-ce-4	4.14.209	v20201211	450.51.06
1.17.12-20201126	1.17.12	19.03.6-ce-4	4.14.203	v20201126	450.51.06
1.17.12-20201117	1.17.12	19.03.6-ce-4	4.14.203	v20201117	450.51.06
1.17.12-20201112	1.17.12	19.03.6-ce-4	4.14.203	v20201112	450.51.06
1.17.11-20201007	1.17.11	19.03.6-ce-4	4.14.198	v20201007	418.87.00
1.17.11-20201002	1.17.11	19.03.6-ce-4	4.14.198	v20201002	418.87.00
1.17.11-20200921	1.17.11	19.03.6-ce-4	4.14.193	v20200921	418.87.00
1.17.9-20200904	1.17.9	19.03.6-ce-4	4.14.193	v20200904	418.87.00
1.17.9-20200814	1.17.9	19.03.6-ce-4	4.14.186	v20200814	418.87.00
1.17.9-20200723	1.17.9	19.03.6-ce	4.14.181	v20200723	418.87.00
1.17.7-20200710	1.17.7	19.03.6-ce	4.14.181	v20200710	418.87.00
1.17.7-20200709	1.17.7	19.03.6-ce	4.14.181	v20200709	418.87.00

Kubernetes version 1.16

Kubernetes version 1.16

AMI version	`kubelet` version	Docker version	Kernel version	Packer version	NVIDIA driver version
1.16.15-20210329	1.16.15	19.03.13-ce-1	4.14.225	v20210329	460.32.03
1.16.15-20210322	1.16.15	19.03.13-ce-1	4.14.225	v20210322	460.27.04
1.16.15-20210310	1.16.15	19.03.13-ce-1	4.14.219	v20210310	460.27.04
1.16.15-20210302	1.16.15	19.03.13-ce-1	4.14.219	v20210302	450.51.06
1.16.15-20210208	1.16.15	19.03.6-ce-4	4.14.214	v20210208	450.51.06
1.16.15-20210125	1.16.15	19.03.6-ce-4	4.14.209	v20210125	450.51.06
1.16.15-20210112	1.16.15	19.03.6-ce-4	4.14.209	v20210112	450.51.06
1.16.15-20201211	1.16.15	19.03.6-ce-4	4.14.209	v20201211	450.51.06
1.16.15-20201126	1.16.15	19.03.6-ce-4	4.14.203	v20201126	450.51.06
1.16.15-20201117	1.16.15	19.03.6-ce-4	4.14.203	v20201117	450.51.06

AMI version	`kubelet` version	Docker version	Kernel version	Packer version	NVIDIA driver version
1.16.15-20201112	1.16.15	19.03.6-ce-4	4.14.203	v20201112	450.51.06
1.16.13-20201007	1.16.13	19.03.6-ce-4	4.14.198	v20201007	418.87.00
1.16.13-20201002	1.16.13	19.03.6-ce-4	4.14.198	v20201002	418.87.00
1.16.13-20200921	1.16.13	19.03.6-ce-4	4.14.193	v20200921	418.87.00
1.16.13-20200904	1.16.13	19.03.6-ce-4	4.14.193	v20200904	418.87.00
1.16.13-20200814	1.16.13	19.03.6-ce-4	4.14.186	v20200814	418.87.00
1.16.13-20200723	1.16.13	19.03.6-ce	4.14.181	v20200723	418.87.00
1.16.12-20200710	1.16.12	19.03.6-ce	4.14.181	v20200710	418.87.00
1.16.12-20200709	1.16.12	19.03.6-ce	4.14.181	v20200709	418.87.00
1.16.8-20200615	1.16.8	19.03.6-ce	4.14.181	v20200615	418.87.00
1.16.8-20200609	1.16.8	19.03.6-ce	4.14.181	v20200609	418.87.00
1.16.8-20200531	1.16.8	18.09.9-ce	4.14.177	v20200531	418.87.00
1.16.8-20200507	1.16.8	18.09.9-ce	4.14.177	v20200507	418.87.00
1.16.8-20200423	1.16.8	18.09.9-ce	4.14.173	v20200423	418.87.00

Kubernetes version 1.15

Kubernetes version 1.15

AMI version	`kubelet` version	Docker version	Kernel version	Packer version	NVIDIA driver version
1.15.12-20210329	1.15.12	19.03.13-ce-1	4.14.225	v20210329	460.32.03
1.15.12-20210322	1.15.12	19.03.13-ce-1	4.14.225	v20210322	460.27.04
1.15.12-20210310	1.15.12	19.03.13-ce-1	4.14.219	v20210310	460.27.04
1.15.12-20210302	1.15.12	19.03.13-ce-1	4.14.219	v20210302	450.51.06
1.15.12-20210208	1.15.12	19.03.6-ce-4	4.14.214	v20210208	450.51.06
1.15.12-20210125	1.15.12	19.03.6-ce-4	4.14.209	v20210125	450.51.06
1.15.12-20210112	1.15.12	19.03.6-ce-4	4.14.209	v20210112	450.51.06
1.15.12-20201211	1.15.12	19.03.6-ce-4	4.14.209	v20201211	450.51.06
1.15.12-20201126	1.15.12	19.03.6-ce-4	4.14.203	v20201126	450.51.06
1.15.12-20201117	1.15.12	19.03.6-ce-4	4.14.203	v20201117	450.51.06
1.15.12-20201112	1.15.12	19.03.6-ce-4	4.14.203	v20201112	450.51.06
1.15.11-20201007	1.15.11	19.03.6-ce-4	4.14.198	v20201007	418.87.00

AMI version	**`kubelet`** version	Docker version	Kernel version	Packer version	NVIDIA driver version
1.15.11-20201002	1.15.11	19.03.6-ce-4	4.14.198	v20201002	418.87.00
1.15.11-20200921	1.15.11	19.03.6-ce-4	4.14.193	v20200921	418.87.00
1.15.11-20200904	1.15.11	19.03.6-ce-4	4.14.193	v20200904	418.87.00
1.15.11-20200814	1.15.11	19.03.6-ce-4	4.14.186	v20200814	418.87.00
1.15.11-20200723	1.15.11	19.03.6-ce	4.14.181	v20200723	418.87.00
1.15.11-20200710	1.15.11	19.03.6-ce	4.14.181	v20200710	418.87.00
1.15.11-20200709	1.15.11	19.03.6-ce	4.14.181	v20200709	418.87.00
1.15.11-20200615	1.15.11	19.03.6-ce	4.14.181	v20200615	418.87.00
1.15.11-20200609	1.15.11	19.03.6-ce	4.14.181	v20200609	418.87.00
1.15.11-20200531	1.15.11	18.09.9-ce	4.14.177	v20200531	418.87.00
1.15.11-20200507	1.15.11	18.09.9-ce	4.14.177	v20200507	418.87.00
1.15.11-20200423	1.15.11	18.09.9-ce	4.14.173	v20200423	418.87.00
1.15.10-20200406	1.15.10	18.09.9-ce	4.14.173	v20200406	418.87.00
1.15.10-20200228	1.15.10	18.09.9-ce	4.14.165	v20200228	418.87.00

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 *`<1.19>`* with a supported version (p. 65) 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.

```
aws ssm get-parameter --name /aws/service/eks/optimized-ami/<1.19>/<amazon-linux-2>/
recommended/image_id --region <region-code> --query "Parameter.Value" --output text
```

  Example output:

```
ami-<abcd1234efgh5678i>
```

- **AWS Management Console** – You can query for the recommended Amazon EKS optimized AMI ID using a URL. The URL opens the Amazon EC2 Systems Manager console with the value of the ID for the parameter. In the following URL, replace *`<1.19>`* with a supported version (p. 65) 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/%252Faws%252Fservice%252Feks
%252Foptimized-ami%252F<1.19>%252F<amazon-linux-2>%252Frecommended%252Fimage_id/
description?region=<region-code>
```

Amazon EKS optimized Amazon Linux AMI build script

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

The Amazon EKS optimized 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 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 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. The AMI is configured to work with Amazon EKS and it includes containerd and `kubelet`.

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. 164).

1.19.6

Kubernetes version 1.19.6

Region	x86	Arm
US East (Ohio) (`us-east-2`)	View AMI ID	View AMI ID

Region	x86	Arm
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
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

1.18.8

Kubernetes version 1.18.8

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
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

1.17.11

Kubernetes version 1.17.11

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
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

1.16.13

Kubernetes version 1.16.13

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
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

1.15.11

Kubernetes version 1.15.11

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
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

Retrieving Amazon EKS optimized Bottlerocket 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 Bottlerocket AMI with the following AWS CLI command by using the sub-parameter `image_id`. Replace *<1.19>* with a supported version (p. 65) 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-<1.19>/x86_64/latest/
image_id --region <region-code> --query "Parameter.Value" --output text
```

  Example output:

```
ami-<068ed1c8e99b4810c>
```

- **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 *<1.19>* with a supported version (p. 65) 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-<1.19>/x86_64/latest/image_id/description?region=<region-code>
```

Amazon EKS optimized Windows AMIs

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

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

The AMI IDs for the latest Amazon EKS optimized AMI are shown in the following tables. Windows Server 2019 is a Long-Term Servicing Channel (LTSC) release and Windows Server, version 2004 is a Semi-Annual Channel (SAC) release. For more information, see Windows Server servicing channels: LTSC and SAC in the Microsoft documentation. 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. 178).

1.19.6

Kubernetes version 1.19.6

Region	Windows Server 2019 Full	Windows Server 2019 Core	Windows Server 2004 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
Canada (Central) (`ca-central-1`)	View AMI ID	View AMI ID	View AMI ID
China (Beijing) (`cn-north-1`)	View AMI ID	View AMI ID	View AMI ID
China (Ningxia) (`cn-northwest-1`)	View AMI ID	View AMI ID	View AMI ID
Europe (Frankfurt) (`eu-central-1`)	View AMI ID	View AMI ID	View AMI ID
Europe (Ireland) (`eu-west-1`)	View AMI ID	View AMI ID	View AMI ID
Europe (London) (`eu-west-2`)	View AMI ID	View AMI ID	View AMI ID

Region	Windows Server 2019 Full	Windows Server 2019 Core	Windows Server 2004 Core
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

1.18.9

Kubernetes version 1.18.9

Region	Windows Server 2019 Full	Windows Server 2019 Core	Windows Server 2004 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

Region	Windows Server 2019 Full	Windows Server 2019 Core	Windows Server 2004 Core
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
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

1.17.12

Kubernetes version 1.17.12

Region	Windows Server 2019 Full	Windows Server 2019 Core	Windows Server 2004 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
Canada (Central) (`ca-central-1`)	View AMI ID	View AMI ID	View AMI ID
China (Beijing) (`cn-north-1`)	View AMI ID	View AMI ID	View AMI ID
China (Ningxia) (`cn-northwest-1`)	View AMI ID	View AMI ID	View AMI ID
Europe (Frankfurt) (`eu-central-1`)	View AMI ID	View AMI ID	View AMI ID
Europe (Ireland) (`eu-west-1`)	View AMI ID	View AMI ID	View AMI ID
Europe (London) (`eu-west-2`)	View AMI ID	View AMI ID	View AMI ID

Region	Windows Server 2019 Full	Windows Server 2019 Core	Windows Server 2004 Core
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

1.16.15

Kubernetes version 1.16.15

Region	Windows Server 2019 Full	Windows Server 2019 Core	Windows Server 2004 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

Region	Windows Server 2019 Full	Windows Server 2019 Core	Windows Server 2004 Core
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
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

1.15.12

Kubernetes version 1.15.12

Region	Windows Server 2019 Full	Windows Server 2019 Core	Windows Server 2004 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
Canada (Central) (`ca-central-1`)	View AMI ID	View AMI ID	View AMI ID
China (Beijing) (`cn-north-1`)	View AMI ID	View AMI ID	View AMI ID
China (Ningxia) (`cn-northwest-1`)	View AMI ID	View AMI ID	View AMI ID
Europe (Frankfurt) (`eu-central-1`)	View AMI ID	View AMI ID	View AMI ID
Europe (Ireland) (`eu-west-1`)	View AMI ID	View AMI ID	View AMI ID
Europe (London) (`eu-west-2`)	View AMI ID	View AMI ID	View AMI ID

Region	Windows Server 2019 Full	Windows Server 2019 Core	Windows Server 2004 Core
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

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 2004 Core	8/19/2020	
Windows Server 2019 Full	10/7/2019	
Windows Server 2019 Core	10/7/2019	
Windows Server 1909 Core	10/7/2019	December 8, 2020

Amazon EKS optimized Windows AMI versions

This topic lists versions of the Amazon EKS optimized Windows AMIs and their corresponding versions of `kubelet` and Docker.

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. 178).

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 2004 Core AMI

The tables below list the current and previous versions of the Amazon EKS optimized Windows AMI.

Kubernetes version 1.19

Kubernetes version 1.19

AMI version	`kubelet` version	Docker version
1.19-2021.03.10	1.19.6	19.03.14
1.19-2021.02.18	1.19.6	19.03.14

Kubernetes version 1.18

Kubernetes version 1.18

AMI version	`kubelet` version	Docker version
1.18-2021.03.10	1.18.9	19.03.14
1.18-2021.02.10	1.18.9	19.03.14
1.18-2021.01.13	1.18.9	19.03.14
1.18-2020.12.11	1.18.9	19.03.13
1.18-2020.11.12	1.18.9	18.09.7
1.18-2020.10.29	1.18.8	18.09.7
1.18-2020.10.08	1.18.8	18.09.7

Kubernetes version 1.17

Kubernetes version 1.17

AMI version	`kubelet` version	Docker version
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

Kubernetes version 1.16

Kubernetes version 1.16

AMI version	`kubelet` version	Docker version
1.16-2021.03.10	1.16.15	19.03.14
1.16-2021.02.10	1.16.15	19.03.14
1.16-2021.01.13	1.16.15	19.03.14
1.16-2020.12.11	1.16.15	19.03.13
1.16-2020.11.12	1.16.15	18.09.7
1.16-2020.10.29	1.16.13	18.09.7
1.16-2020.09.09	1.16.13	18.09.7
1.16-2020.08.13	1.16.13	18.09.7

Kubernetes version 1.15

The most recent version is the last version we're releasing for Amazon EKS 1.15 clusters. It will be available until the 1.15 end of support date. For more information, see the section called “Amazon EKS Kubernetes release calendar” (p. 63). For newer AMI releases, update your cluster to a later Kubneretes version. For more information, see the section called “Updating a cluster” (p. 24).

Kubernetes version 1.15

AMI version	`kubelet` version	Docker version
1.15-2021.02.10	1.15.12	19.03.14
1.15-2021.01.13	1.15.12	19.03.14
1.15-2020.12.11	1.15.12	19.03.13
1.15-2020.11.12	1.15.12	18.09.7
1.15-2020.10.29	1.15.11	18.09.7
1.15-2020.09.09	1.15.11	18.09.7
1.15-2020.08.13	1.15.11	18.09.7

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 1.19

Kubernetes version 1.19

AMI version	`kubelet` version	Docker version
1.19-2021.03.10	1.19.6	19.03.14
1.19-2021.02.18	1.19.6	19.03.14

Kubernetes version 1.18

Kubernetes version 1.18

AMI version	`kubelet` version	Docker version
1.18-2021.03.10	1.18.9	19.03.14
1.18-2021.02.10	1.18.9	19.03.14
1.18-2021.01.13	1.18.9	19.03.14
1.18-2020.12.11	1.18.9	19.03.13
1.18-2020.11.12	1.18.9	18.09.7
1.18-2020.10.29	1.18.8	18.09.7
1.18-2020.10.08	1.18.8	18.09.7

Kubernetes version 1.17

Kubernetes version 1.17

AMI version	`kubelet` version	Docker version
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

Kubernetes version 1.16

Kubernetes version 1.16

AMI version	`kubelet` version	Docker version
1.16-2021.03.10	1.16.15	19.03.14
1.16-2021.02.10	1.16.15	19.03.14
1.16-2021.01.13	1.16.15	19.03.14
1.16-2020.12.11	1.16.15	19.03.13
1.16-2020.11.12	1.16.15	18.09.7
1.16-2020.10.29	1.16.13	18.09.7

AMI version	`kubelet` version	Docker version
1.16-2020.09.09	1.16.13	18.09.7
1.16-2020.08.13	1.16.13	18.09.7

Kubernetes version 1.15

The most recent version is the last version we're releasing for Amazon EKS 1.15 clusters. It will be available until the 1.15 end of support date. For more information, see the section called "Amazon EKS Kubernetes release calendar" (p. 63). For newer AMI releases, update your cluster to a later Kubneretes version. For more information, see the section called "Updating a cluster" (p. 24).

Kubernetes version 1.15

AMI version	`kubelet` version	Docker version
1.15-2021.02.10	1.15.12	19.03.14
1.15-2021.01.13	1.15.12	19.03.14
1.15-2020.12.11	1.15.12	19.03.13
1.15-2020.11.12	1.15.12	18.09.7
1.15-2020.10.29	1.15.11	18.09.7
1.15-2020.09.09	1.15.11	18.09.7
1.15-2020.08.13	1.15.11	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 1.19

Kubernetes version 1.19

AMI version	`kubelet` version	Docker version
1.19-2021.02.18	1.19.6	19.03.14

Kubernetes version 1.18

Kubernetes version 1.18

AMI version	`kubelet` version	Docker version
1.18-2021.03.10	1.18.9	19.03.14
1.18-2021.02.10	1.18.9	19.03.14
1.18-2021.01.13	1.18.9	19.03.14
1.18-2020.12.11	1.18.9	19.03.13

AMI version	`kubelet` version	Docker version
1.18-2020.11.12	1.18.9	18.09.7
1.18-2020.10.29	1.18.8	18.09.7
1.18-2020.10.08	1.18.8	18.09.7

Kubernetes version 1.17

Kubernetes version 1.17

AMI version	`kubelet` version	Docker version
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

Kubernetes version 1.16

Kubernetes version 1.16

AMI version	`kubelet` version	Docker version
1.16-2021.03.10	1.16.15	19.03.14
1.16-2021.02.10	1.16.15	19.03.14
1.16-2021.01.13	1.16.15	19.03.14
1.16-2020.12.11	1.16.15	19.03.13
1.16-2020.11.12	1.16.15	18.09.7
1.16-2020.10.29	1.16.13	18.09.7
1.16-2020.09.09	1.16.13	18.09.7
1.16-2020.08.13	1.16.13	18.09.7

Kubernetes version 1.15

The most recent version is the last version we're releasing for Amazon EKS 1.15 clusters. It will be available until the 1.15 end of support date. For more information, see the section called "Amazon EKS Kubernetes release calendar" (p. 63). For newer AMI releases, update your cluster to a later Kubneretes version. For more information, see the section called "Updating a cluster" (p. 24).

Kubernetes version 1.15

AMI version	`kubelet` version	Docker version
1.15-2021.02.10	1.15.12	19.03.14
1.15-2021.01.13	1.15.12	19.03.14
1.15-2020.12.11	1.15.12	19.03.13
1.15-2020.11.12	1.15.12	18.09.7
1.15-2020.10.29	1.15.11	18.09.7
1.15-2020.09.09	1.15.11	18.09.7
1.15-2020.08.13	1.15.11	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 *<1.19>* (including *<>*) 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 `2004` for the `Core` version only.

```
aws ssm get-parameter --name /aws/service/ami-windows-latest/Windows_Server-<2019>-
English-<Core>-EKS_Optimized-<1.19>/image_id --region <region-code> --query
 "Parameter.Value" --output text
```

  Example output:

```
ami-<ami-00a053f1635fffea0>
```

- **AWS Management Console** – You can query for the recommended Amazon EKS optimized AMI ID using a URL. The URL opens the Amazon EC2 Systems Manager console with the value of the ID for the parameter. In the following URL, you can replace *<1.19>* (including *<>*) 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 `2004` for the `Core` version only.

```
https://console.aws.amazon.com/systems-manager/parameters/%252Faws
%252Fservice%252Fami-windows-latest%252FWindows_Server-<2019>-English-<Core>-
EKS_Optimized-<1.19>%252Fimage_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 drop-down, select an Amazon EKS supported Windows Server version (p. 164).
  - **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 drop-down 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>` (including <>) with a supported Kubernetes version (p. 58). If you enter 1.19.x as the component version, your Image Builder pipeline builds an AMI with the latest 1.19.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.

You should 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. 180) 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.

Topics

Storage classes

Amazon EKS clusters that were created prior to Kubernetes version 1.11 were not created with any storage classes. You must define storage classes for your cluster to use and you should define a default storage class for your persistent volume claims. For more information, see Storage classes in the Kubernetes documentation.

Note
This topic uses the in-tree Amazon EBS storage provisioner. 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.

To create an AWS storage class for your Amazon EKS cluster

1. Determine which storage classes your cluster already has.

```
kubectl get storageclass
```

   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. 180) chapter. Once deployed, you can set one of the storage classes as your default (p. 181) 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 provides a CSI interface that allows Amazon Elastic Kubernetes Service (Amazon EKS) clusters to manage the lifecycle of Amazon EBS volumes for persistent volumes.

This topic shows you how to deploy the Amazon EBS CSI Driver to your Amazon EKS cluster and verify that it works. We recommend using version v0.9.0 of the driver.

> **Note**
> The driver is not supported on Fargate. Alpha features of the Amazon EBS CSI Driver are not supported on Amazon EKS clusters. The driver is in Beta release. It is well tested and supported by Amazon EKS for production use. Support for the driver will not be dropped, though details may change. If the schema or schematics of the driver changes, instructions for migrating to the next version will be provided.

For detailed descriptions of the available parameters and complete examples that demonstrate the driver's features, see the Amazon EBS Container Storage Interface (CSI) driver project on GitHub.

Prerequisites

- An existing cluster. If you don't have one, see *Getting started with Amazon EKS* (p. 4) to create one.
- An existing IAM OpenID Connect (OIDC) provider for your cluster. To determine whether you already have one, or to create one, see the section called "Create OIDC provider" (p. 349).
- AWS CLI version 1.19.7 or later or 2.1.26 installed on your computer or AWS CloudShell. To install or upgrade the AWS CLI , see Installing, updating, and uninstalling the AWS CLI.
- `kubectl` version 1.15 or later installed on your computer or AWS CloudShell. To install or upgrade kubectl, see the section called "Installing `kubectl`" (p. 303).

To deploy the Amazon EBS 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. You can view the policy document on GitHub.

   a. Download the IAM policy document from GitHub.

   ```
   curl -o example-iam-policy.json https://raw.githubusercontent.com/kubernetes-sigs/
   aws-ebs-csi-driver/v0.9.0/docs/example-iam-policy.json
   ```

   b. Create the policy. You can change `AmazonEKS_EBS_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_EBS_CSI_Driver_Policy \
     --policy-document file://example-iam-policy.json
   ```

2. Create an IAM role and attach the IAM policy to it.

   a. View your cluster's OIDC provider URL. Replace `<cluster_name>` (including `<>`) with your cluster name. If the output from the command is `None`, review the **Prerequisites**.

   ```
   aws eks describe-cluster --name <cluster_name> --query
    "cluster.identity.oidc.issuer" --output text
   ```

   Output

   ```
   https://oidc.eks.us-west-2.amazonaws.com/id/XXXXXXXXXX45D83924220DC4815XXXXX
   ```

   b. Create the IAM role.

      1. Copy the following contents to a file named `trust-policy.json`. Replace `<AWS_ACCOUNT_ID>` (including `<>`) with your account ID and `<XXXXXXXXXX45D83924220DC4815XXXXX>` with the value returned in the previous step.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Federated": "arn:aws:iam::<AWS_ACCOUNT_ID>:oidc-provider/oidc.eks.us-
west-2.amazonaws.com/id/<XXXXXXXXXX45D83924220DC4815XXXXX>"
      },
      "Action": "sts:AssumeRoleWithWebIdentity",
      "Condition": {
        "StringEquals": {
          "oidc.eks.us-west-2.amazonaws.com/id/
<XXXXXXXXXX45D83924220DC4815XXXXX>:sub": "system:serviceaccount:kube-system:ebs-
csi-controller-sa"
        }
      }
    }
  ]
}
```

2. Create the role. You can change `AmazonEKS_EBS_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_EBS_CSI_DriverRole \
  --assume-role-policy-document file://"trust-policy.json"
```

c. Attach the IAM policy to the role. Replace `<AWS_ACCOUNT_ID>` (including `<>`) with your account ID.

```
aws iam attach-role-policy \
  --policy-arn arn:aws:iam::<AWS_ACCOUNT_ID>:policy/AmazonEKS_EBS_CSI_Driver_Policy
 \
  --role-name AmazonEKS_EBS_CSI_DriverRole
```

3. Deploy the driver using one of the following options.

- Deploy the driver so that it tags all Amazon EBS volumes that it creates with tags that your specify.

  a. Clone the Amazon EBS Container Storage Interface (CSI) driver GitHub repository to your computer.

```
git clone https://github.com/kubernetes-sigs/aws-ebs-csi-driver.git
```

  b. Navigate to the `base` example folder.

```
cd aws-ebs-csi-driver/deploy/kubernetes/base/
```

  c. Edit the `controller.yaml` file. Find the section of the file with the following text and add `--extra-tags` to it. The following text shows the section of the file with the existing and added text. This example will cause the controller to add `department` and `environment` tags to all volumes it creates.

```
...
containers:
        - name: ebs-plugin
          image: amazon/aws-ebs-csi-driver:latest
          imagePullPolicy: IfNotPresent
          args:
            # - {all,controller,node} # specify the driver mode
            - --endpoint=$(CSI_ENDPOINT)
            - --logtostderr
            - --v=5
            - --extra-tags=department=accounting,environment=dev
...
```

d. Apply the modified manifest to your cluster.

```
kubectl apply -k ../base
```

- Deploy the driver so that it doesn't tag the Amazon EBS volumes that it creates with tags that your specify. Use the command that corresponds to the Region that your cluster is in.

  Note
  This commands require version 1.14 or later of `kubectl`. You can see your `kubectl` version with the following command. To install or upgrade your `kubectl` version, see Installing `kubectl` (p. 303).

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

  - **Note**
    To see or download the `yaml` file manually, you can find it on the aws-ebs-csi-driver Github.

  All Regions other than China Regions.

  ```
  kubectl apply -k "github.com/kubernetes-sigs/aws-ebs-csi-driver/deploy/kubernetes/
  overlays/stable/?ref=master"
  ```

  - Beijing and Ningxia China Regions.

  ```
  kubectl apply -k "github.com/kubernetes-sigs/aws-ebs-csi-driver/deploy/kubernetes/
  overlays/stable-cn/?ref=master"
  ```

4. Annotate the `ebs-csi-controller-sa` Kubernetes service account with the ARN of the IAM role that you created previously. Replace the `<AWS_ACCOUNT_ID>` (including `<>`) with your account ID.

```
kubectl annotate serviceaccount ebs-csi-controller-sa \
  -n kube-system \
  eks.amazonaws.com/role-arn=arn:aws:iam::<AWS_ACCOUNT_ID>:role/
AmazonEKS_EBS_CSI_DriverRole
```

5. Delete the driver pods. They're automatically redeployed with the IAM permissions from the IAM policy assigned to the role.

```
kubectl delete pods \
  -n kube-system \
  -l=app=ebs-csi-controller
```

To deploy a sample application and verify that the CSI driver is working

This procedure uses the Dynamic volume provisioning example from the Amazon EBS Container Storage Interface (CSI) driver GitHub repository to consume a dynamically-provisioned Amazon EBS volume.

1. Clone the Amazon EBS Container Storage Interface (CSI) driver GitHub repository to your local system.

```
git clone https://github.com/kubernetes-sigs/aws-ebs-csi-driver.git
```

2. Navigate to the `dynamic-provisioning` example directory.

```
cd aws-ebs-csi-driver/examples/kubernetes/dynamic-provisioning/
```

3. Deploy the `ebs-sc` storage class, `ebs-claim` persistent volume claim, and `app` sample application from the `specs` directory.

```
kubectl apply -f specs/
```

4. Describe the `ebs-sc` storage class.

```
kubectl describe storageclass ebs-sc
```

   Output:

```
Name:            ebs-sc
IsDefaultClass:  No
Annotations:     kubectl.kubernetes.io/last-applied-
configuration={"apiVersion":"storage.k8s.io/v1","kind":"StorageClass","metadata":
{"annotations":{},"name":"ebs-
sc"},"provisioner":"ebs.csi.aws.com","volumeBindingMode":"WaitForFirstConsumer"}

Provisioner:           ebs.csi.aws.com
Parameters:            <none>
AllowVolumeExpansion:  <unset>
MountOptions:          <none>
ReclaimPolicy:         Delete
VolumeBindingMode:     WaitForFirstConsumer
Events:                <none>
```

   Note that the storage class uses the `WaitForFirstConsumer` volume binding mode. This means that volumes are not dynamically provisioned until a pod makes a persistent volume claim. For more information, see Volume Binding Mode in the Kubernetes documentation.

5. Watch the pods in the default namespace and wait for the `app` pod's status to become `Running`.

```
kubectl get pods --watch
```

6. List the persistent volumes in the default namespace. Look for a persistent volume with the `default/ebs-claim` claim.

```
kubectl get pv
```

   Output:

```
NAME                                       CAPACITY   ACCESS MODES   RECLAIM POLICY
 STATUS   CLAIM               STORAGECLASS   REASON   AGE
pvc-37717cd6-d0dc-11e9-b17f-06fad4858a5a   4Gi        RWO            Delete
 Bound    default/ebs-claim   ebs-sc                  30s
```

7. Describe the persistent volume, replacing `<pvc-37717cd6-d0dc-11e9-b17f-06fad4858a5a>` (including <>) with the value from the output in the previous step.

```
kubectl describe pv <pvc-37717cd6-d0dc-11e9-b17f-06fad4858a5a>
```

Output:

```
Name:               pvc-37717cd6-d0dc-11e9-b17f-06fad4858a5a
Labels:             <none>
Annotations:        pv.kubernetes.io/provisioned-by: ebs.csi.aws.com
Finalizers:         [kubernetes.io/pv-protection external-attacher/ebs-csi-aws-com]
StorageClass:       ebs-sc
Status:             Bound
Claim:              default/ebs-claim
Reclaim Policy:     Delete
Access Modes:       RWO
VolumeMode:         Filesystem
Capacity:           4Gi
Node Affinity:
  Required Terms:
    Term 0:         topology.ebs.csi.aws.com/zone in [us-west-2d]
Message:
Source:
    Type:               CSI (a Container Storage Interface (CSI) volume source)
    Driver:             ebs.csi.aws.com
    VolumeHandle:       <vol-0d651e157c6d93445>
    ReadOnly:           false
    VolumeAttributes:       storage.kubernetes.io/
csiProvisionerIdentity=1567792483192-8081-ebs.csi.aws.com
Events:                 <none>
```

The Amazon EBS volume ID is listed as the `VolumeHandle`.

8. Verify that the pod is successfully writing data to the volume.

```
kubectl exec -it app -- cat /data/out.txt
```

Output:

```
Fri Jan 8 15:34:20 UTC 2021
Fri Jan 8 15:34:25 UTC 2021
Fri Jan 8 15:34:30 UTC 2021
Fri Jan 8 15:34:35 UTC 2021
Fri Jan 8 15:34:40 UTC 2021
Fri Jan 8 15:34:45 UTC 2021
...
```

9. When you finish experimenting, delete the resources for this sample application to clean up.

```
kubectl delete -f specs/
```

Amazon EFS CSI driver

The Amazon EFS Container Storage Interface (CSI) driver provides a CSI interface that allows 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 are not 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

- 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. 58).

Prerequisites

- **Existing cluster with an OIDC provider** – If you don't have a cluster, you can create one using one of the *Getting started with Amazon EKS* (p. 4) guides. To determine whether you have an OIDC provider for an existing cluster, or to create one, see the section called "Create OIDC provider" (p. 349).
- **AWS CLI** – A command line tool for working with AWS services, including Amazon EKS. This guide requires that you use version 2.1.26 or later or 1.19.7 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 1.19 or later. For more information, see Installing `kubectl` (p. 303).

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/v1.2.0/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

   The following command creates the IAM role and Kubernetes service account. It also attach 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. If you don't have an IAM OIDC provider for your cluster, the command also create the IAM OIDC provider.

```
eksctl create iamserviceaccount \
    --name efs-csi-controller-sa \
    --namespace kube-system \
    --cluster <cluster-name> \
    --attach-policy-arn arn:aws:iam::<AWS account
 ID>:policy/AmazonEKS_EFS_CSI_Driver_Policy \
    --approve \
    --override-existing-serviceaccounts \
    --region us-west-2
```

AWS CLI

1. Determine your cluster's OIDC provider URL. Replace `<cluster_name>` (including `<>`) with your cluster name. If the output from the command is `None`, review the **Prerequisites**.

   ```
   aws eks describe-cluster --name <cluster-name> --query
    "cluster.identity.oidc.issuer" --output text
   ```

   Output

   ```
   https://oidc.eks.us-west-2.amazonaws.com/id/EXAMPLEXXX45D83924220DC4815XXXXX
   ```

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 `<AWS_ACCOUNT_ID>` (including `<>`) with your account ID and `<EXAMPLEXXX45D83924220DC4815XXXXX>` and `us-west-2` with the value returned in the previous step.

   ```
   {
     "Version": "2012-10-17",
     "Statement": [
       {
         "Effect": "Allow",
         "Principal": {
           "Federated": "arn:aws:iam::<AWS_ACCOUNT_ID>:oidc-provider/oidc.eks.us-
   west-2.amazonaws.com/id/<EXAMPLEXXX45D83924220DC4815XXXXX>"
         },
         "Action": "sts:AssumeRoleWithWebIdentity",
         "Condition": {
           "StringEquals": {
             "oidc.eks.us-west-2.amazonaws.com/
   id/<EXAMPLEXXX45D83924220DC4815XXXXX>: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 `<AWS_ACCOUNT_ID>` (including `<>`) with your account ID.

```
aws iam attach-role-policy \
  --policy-arn
 arn:aws:iam::<AWS_ACCOUNT_ID>:policy/AmazonEKS_EFS_CSI_Driver_Policy \
  --role-name AmazonEKS_EFS_CSI_DriverRole
```

4. Create a Kubernetes service account that is annotated with the ARN of the IAM role that you created.

   a. Save the following contents to a file named *efs-service-account.yaml*.

   ```
   ---
   apiVersion: v1
   kind: ServiceAccount
   metadata:
     name: efs-csi-controller-sa
     namespace: kube-system
     labels:
       app.kubernetes.io/name: aws-efs-csi-driver
     annotations:
       eks.amazonaws.com/role-arn:
    arn:aws:iam::<AWS_ACCOUNT_ID>:role/AmazonEKS_EFS_CSI_DriverRole
   ```

   b. Apply the manifest.

   ```
   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 1.2.0 version of the driver, which requires a 1.17 or later cluster. If you're installing the driver on a cluster that is 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 the section called "Using Helm" (p. 317).

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 the chart. If your cluster isn't in the *us-west-2* Region, then change *602401143452*.dkr.ecr.*us-west-2*.*amazonaws.com* to the address (p. 275) for your Region.

```
helm upgrade -i aws-efs-csi-driver aws-efs-csi-driver/aws-efs-csi-driver \
    --namespace kube-system \
    --set image.repository=602401143452.dkr.ecr.us-west-2.amazonaws.com/eks/aws-efs-
csi-driver \
    --set serviceAccount.controller.create=false \
    --set serviceAccount.controller.name=efs-csi-controller-sa
```

Manifest

1. Download the manifest.

```
kubectl kustomize \
    "github.com/kubernetes-sigs/aws-efs-csi-driver/deploy/kubernetes/overlays/stable/
ecr?ref=release-1.2" > driver.yaml
```

2. Edit the file and remove the following lines that create a Kubernetes service account. This isn't necessary since 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. Find the following line. If your cluster is not in the `us-west-2` region, replace the following address with the address for your Region (p. 275). Once you've made the change, save your modified manifest.

```
image: 602401143452.dkr.ecr.us-west-2.amazonaws.com/eks/aws-efs-csi-driver:v1.2.0
```

4. Apply the manifest.

```
kubectl apply -f 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 *<cluster-name>* (including *<>*) with your cluster name.

```
vpc_id=$(aws eks describe-cluster \
    --name <cluster-name> \
    --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.

   a. Create a file system.

```
file_system_id=$(aws efs create-file-system \
    --region us-west-2 \
    --performance-mode generalPurpose \
    --query 'FileSystemId' \
    --output text)
```

   b. Create mount targets.

      i. Determine the IP address of your cluster nodes.

```
kubectl get nodes
```

Output

```
NAME                                           STATUS   ROLES    AGE   VERSION
ip-192-168-56-0.us-west-2.compute.internal    Ready    <none>   19m   v1.19.6-
eks-49a6c0
```

      ii. 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
```

Output

```
|                            DescribeSubnets                           |
+-------------------+--------------------+----------------------------+
| AvailabilityZone  |     CidrBlock      |          SubnetId          |
+-------------------+--------------------+----------------------------+
|  us-west-2c       |  192.168.128.0/19  |  subnet-EXAMPLE6e421a0e97  |
|  us-west-2b       |  192.168.96.0/19   |  subnet-EXAMPLEd0503db0ec  |
|  us-west-2c       |  192.168.32.0/19   |  subnet-EXAMPLEe2ba886490  |
|  us-west-2b       |  192.168.0.0/19    |  subnet-EXAMPLE123c7c5182  |
|  us-west-2a       |  192.168.160.0/19  |  subnet-EXAMPLE0416ce588p  |
|  us-west-2a       |  192.168.64.0/19   |  subnet-EXAMPLE12c68ea7fb  |
+-------------------+--------------------+----------------------------+
```

iii. 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
```

(Optional) Deploy a sample application

You can deploy a sample app that dynamically creates a persistent volume, or you can manually create a persistent volume.

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. 24).

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 EFS access points and a Persistent Volume Claim (PVC) that is 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. 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
```

   b. Edit the file, replacing the value for `fileSystemId` with your file system ID.
   c. 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
   ```

3. Determine the names of the pods running the controller.

   ```
   kubectl get pods -n kube-system | grep efs-csi-controller
   ```

   Output

   ```
   efs-csi-controller-74ccf9f566-q5989   3/3     Running   0          40m
   efs-csi-controller-74ccf9f566-wswg9   3/3     Running   0          40m
   ```

4. 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
   ```

   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.

5. Confirm that a persistent volume was created with a status of `Bound` to a `PersistentVolumeClaim`:

   ```
   kubectl get pv
   ```

   Output

   ```
   NAME                                       CAPACITY   ACCESS MODES   RECLAIM POLICY
    STATUS   CLAIM                STORAGECLASS   REASON   AGE
   pvc-5983ffec-96cf-40c1-9cd6-e5686ca84eca   20Gi       RWX            Delete
    Bound    default/efs-claim    efs-sc                  7m57s
   ```

6. View details about the `PersistentVolumeClaim` that was created.

```
kubectl get pvc
```

Output

```
NAME        STATUS   VOLUME                                     CAPACITY   ACCESS
 MODES   STORAGECLASS   AGE
efs-claim   Bound    pvc-5983ffec-96cf-40c1-9cd6-e5686ca84eca   20Gi       RWX
     efs-sc          9m7s
```

7. View the sample app pod's status.

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

Output

```
NAME            READY   STATUS    RESTARTS   AGE   IP               NODE
                            NOMINATED NODE   READINESS GATES
efs-example   1/1     Running   0          10m   192.168.78.156
 ip-192-168-73-191.us-west-2.compute.internal   <none>           <none>
```

Confirm that the data is written to the volume.

```
kubectl exec efs-app -- bash -c "cat data/out"
```

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
...
```

8. (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 step 7 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 does not enforce any file system capacity limits. The actual storage capacity value in persistent volumes and persistent volume claims is not used when creating the file system. However, since storage capacity is a required field in Kubernetes, you must specify a valid value, such as, `5Gi` in this example. This value does not limit the size of your Amazon EFS file system.

5. Deploy the `efs-sc` storage class, `efs-claim` persistent volume claim, 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"],"capaci...
                    pv.kubernetes.io/bound-by-controller: yes
   Finalizers:      [kubernetes.io/pv-protection]
   StorageClass:    efs-sc
   Status:          Bound
   Claim:           default/efs-claim
   Reclaim Policy:  Retain
   Access Modes:    RWX
   VolumeMode:      Filesystem
   Capacity:        5Gi
   Node Affinity:   none
   Message:
   Source:
       Type:              CSI (a Container Storage Interface (CSI) volume source)
       Driver:            efs.csi.aws.com
       VolumeHandle:      fs-582a03f3
       ReadOnly:          false
       VolumeAttributes:  none
   Events:                none
   ```

   The Amazon EFS file system ID is listed as the `VolumeHandle`.

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

Important
This capability is not available in China Regions.

The Amazon FSx for Lustre Container Storage Interface (CSI) driver provides a CSI interface that allows Amazon EKS clusters to manage the lifecycle of Amazon FSx for Lustre file systems.

This topic shows you how to deploy the Amazon FSx for Lustre CSI Driver to your Amazon EKS cluster and verify that it works. We recommend using version 0.4.0 of the driver.

Note
This driver is supported on Kubernetes version 1.19 and later Amazon EKS clusters and nodes. The driver is not supported on Fargate or Arm nodes. Alpha features of the Amazon FSx for Lustre CSI Driver are not supported on Amazon EKS clusters. The driver is in Beta release. It is well tested and supported by Amazon EKS for production use. Support for the driver will not be dropped, though details may change. If the schema or schematics of the driver changes, instructions for migrating to the next version will be provided.

For detailed descriptions of the available parameters and complete examples that demonstrate the driver's features, see the Amazon FSx for Lustre Container Storage Interface (CSI) driver project on GitHub.

Prerequisites

You must have:

- Version 1.19.7 or later of the AWS CLI installed. You can check your currently-installed version with the `aws --version` command. To install or upgrade the AWS CLI, see Installing the AWS CLI.
- An existing Amazon EKS cluster. If you don't currently have a cluster, see Getting started with Amazon EKS (p. 4) to create one.
- Version 0.43.0 or later of `eksctl` installed. You can check your currently-installed version with the `eksctl version` command. To install or upgrade `eksctl`, see Installing or upgrading `eksctl` (p. 308).
- The latest version of `kubectl` installed that aligns to your cluster version. You can check your currently-installed version with the `kubectl version --short --client` command. For more information, see Installing `kubectl` (p. 303).

To deploy the Amazon FSx for Lustre CSI driver to an Amazon EKS cluster

1. Create an AWS Identity and Access Management OIDC provider and associate it with your cluster.

```
eksctl utils associate-iam-oidc-provider \
    --region <region-code> \
    --cluster <prod> \
    --approve
```

2. Create an IAM policy and service account that allows the driver to make calls to AWS APIs on your behalf.

   a. Copy the following text and save it to a file named `fsx-csi-driver.json`.

```
{
   "Version":"2012-10-17",
   "Statement":[
      {
         "Effect":"Allow",
         "Action":[
            "iam:CreateServiceLinkedRole",
            "iam:AttachRolePolicy",
            "iam:PutRolePolicy"
         ],
         "Resource":"arn:aws:iam::*:role/aws-service-role/s3.data-
source.lustre.fsx.amazonaws.com/*"
      },
      {
         "Action":"iam:CreateServiceLinkedRole",
         "Effect":"Allow",
         "Resource":"*",
         "Condition":{
            "StringLike":{
               "iam:AWSServiceName":[
                  "fsx.amazonaws.com"
               ]
            }
         }
      },
      {
         "Effect":"Allow",
         "Action":[
            "s3:ListBucket",
            "fsx:CreateFileSystem",
            "fsx:DeleteFileSystem",
            "fsx:DescribeFileSystems"
         ],
         "Resource":[
            "*"
         ]
      }
   ]
}
```

   b. Create the policy.

```
aws iam create-policy \
    --policy-name <Amazon_FSx_Lustre_CSI_Driver> \
    --policy-document file://fsx-csi-driver.json
```

   Take note of the policy Amazon Resource Name (ARN) that is returned.

3. Create a Kubernetes service account for the driver and attach the policy to the service account. Replacing the ARN of the policy with the ARN returned in the previous step.

```
eksctl create iamserviceaccount \
    --region <region-code> \
    --name fsx-csi-controller-sa \
    --namespace kube-system \
    --cluster <prod> \
    --attach-policy-arn arn:aws:iam::<111122223333:policy/Amazon_FSx_Lustre_CSI_Driver>
 \
    --approve
```

   Output:

   You'll see several lines of output as the service account is created. The last line of output is similar to the following example line.

```
[#]  created serviceaccount "kube-system/fsx-csi-controller-sa"
```

   Note the name of the AWS CloudFormation stack that was deployed. In the example output above, the stack is named `eksctl-prod-addon-iamserviceaccount-kube-system-fsx-csi-controller-sa`.

4. Note the **Role ARN** for the role that was created.

   a. Open the AWS CloudFormation console at https://console.aws.amazon.com/cloudformation.
   b. Ensure that the console is set to the Region that you created your IAM role in and then select **Stacks**.
   c. Select the stack named `eksctl-prod-addon-iamserviceaccount-kube-system-fsx-csi-controller-sa`.
   d. Select the **Outputs** tab. The **Role ARN** is listed on the **Output(1)** page.

5. Deploy the driver with the following command.

   Note
   To see or download the `yaml` file manually, you can find it on the aws-fsx-csi-driver Github.

```
kubectl apply -k "github.com/kubernetes-sigs/aws-fsx-csi-driver/deploy/kubernetes/
overlays/stable/?ref=master"
```

   Output

```
Warning: kubectl apply should be used on resource created by either kubectl create --
save-config or kubectl apply
serviceaccount/fsx-csi-controller-sa configured
clusterrole.rbac.authorization.k8s.io/fsx-csi-external-provisioner-role created
clusterrolebinding.rbac.authorization.k8s.io/fsx-csi-external-provisioner-binding
 created
deployment.apps/fsx-csi-controller created
daemonset.apps/fsx-csi-node created
csidriver.storage.k8s.io/fsx.csi.aws.com created
```

6. Patch the driver deployment to add the service account that you created in step 3, replacing the ARN with the ARN that you noted in step 4.

```
kubectl annotate serviceaccount -n kube-system <fsx-csi-controller-sa> \
 eks.amazonaws.com/role-arn=<arn:aws:iam::111122223333:role/eksctl-prod-addon-
iamserviceaccount-kube-sys-Role1-NPFTLHJ5PJF5> --overwrite=true
```

To deploy a Kubernetes storage class, persistent volume claim, and sample application to verify that the CSI driver is working

This procedure uses the Dynamic volume provisioning for Amazon S3 from the Amazon FSx for Lustre Container Storage Interface (CSI) driver GitHub repository to consume a dynamically-provisioned Amazon FSx for Lustre volume.

1. Create an Amazon S3 bucket and a folder within it named `export` by creating and copying a file to the bucket.

```
aws s3 mb s3://<fsx-csi>
echo test-file >> testfile
aws s3 cp testfile s3://<fsx-csi>/export/testfile
```

2. Download the `storageclass` manifest with the following command.

```
curl -o storageclass.yaml https://raw.githubusercontent.com/kubernetes-sigs/aws-fsx-
csi-driver/master/examples/kubernetes/dynamic_provisioning_s3/specs/storageclass.yaml
```

3. Edit the file and replace the existing, `<example values>` with your own.

```
parameters:
  subnetId: <subnet-056da83524edbe641>
  securityGroupIds: <sg-086f61ea73388fb6b>
  s3ImportPath: s3://<ml-training-data-000>
  s3ExportPath: s3://<ml-training-data-000/export>
  deploymentType: <SCRATCH_2>
```

   - **subnetId** – The subnet ID that the Amazon FSx for Lustre file system should be created in. Amazon FSx for Lustre is not supported in all Availability Zones. Open the Amazon FSx for Lustre console at https://console.aws.amazon.com/fsx/ to confirm that the subnet that you want to use is in a supported Availability Zone. The subnet can include your nodes, or can be a different subnet or VPC. If the subnet that you specify is not 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.
   - **s3ImportPath** – The Amazon Simple Storage Service data repository that you want to copy data from to the persistent volume. Specify the `fsx-csi` bucket that you created in step 1.
   - **s3ExportPath** – The Amazon S3 data repository that you want to export new or modified files to. Specify the `fsx-csi/export` folder that you created in step 1.
   - **deploymentType** – The file system deployment type. Valid values are `SCRATCH_1`, `SCRATCH_2`, and `PERSISTENT_1`. For more information about deployment types, see Create your Amazon FSx for Lustre file system.

   Note
   The Amazon S3 bucket for `s3ImportPath` and `s3ExportPath` must be the same, otherwise the driver cannot create the Amazon FSx for Lustre file system. The `s3ImportPath` can stand alone. A random path will be created automatically like `s3://ml-training-data-000/FSxLustre20190308T012310Z`. The `s3ExportPath` cannot be used without specifying a value for `S3ImportPath`.

4. Create the `storageclass`.

```
kubectl apply -f storageclass.yaml
```

5. Download the persistent volume claim manifest.

```
curl -o claim.yaml https://raw.githubusercontent.com/kubernetes-sigs/aws-fsx-csi-
driver/master/examples/kubernetes/dynamic_provisioning_s3/specs/claim.yaml
```

6. (Optional) Edit the `claim.yaml` file. Change the following <value> to one of the increment values listed below, based on your storage requirements and the `deploymentType` that you selected in a previous step.

   ```
   storage: <1200Gi>
   ```

   - `SCRATCH_2` and `PERSISTENT` – 1.2 TiB, 2.4 TiB, or increments of 2.4 TiB over 2.4 TiB.
   - `SCRATCH_1` – 1.2 TiB, 2.4 TiB, 3.6 TiB, or increments of 3.6 TiB over 3.6 TiB.
7. Create the persistent volume claim.

   ```
   kubectl apply -f claim.yaml
   ```
8. Confirm that the file system is provisioned.

   ```
   kubectl get pvc
   ```

   Output.

   ```
   NAME        STATUS   VOLUME                                     CAPACITY   ACCESS MODES
      STORAGECLASS   AGE
   fsx-claim   Bound    pvc-15dad3c1-2365-11ea-a836-02468c18769e   1200Gi     RWX
      fsx-sc         7m37s
   ```

   Note
   The `STATUS` may show as `Pending` for 5-10 minutes, before changing to `Bound`. Don't continue with the next step until the `STATUS` is `Bound`.
9. Deploy the sample application.

   ```
   kubectl apply -f https://raw.githubusercontent.com/kubernetes-sigs/aws-fsx-csi-driver/
   master/examples/kubernetes/dynamic_provisioning_s3/specs/pod.yaml
   ```
10. Verify that the sample application is running.

    ```
    kubectl get pods
    ```

    Output

    ```
    NAME      READY   STATUS              RESTARTS   AGE
    fsx-app   1/1     Running             0          8s
    ```

Access Amazon S3 files from the Amazon FSx for Lustre file system

If you only want to import data and read it without any modification and creation, then you don't need a value for `s3ExportPath` in your `storageclass.yaml` file. Verify that data was written to the Amazon FSx for Lustre file system by the sample app.

```
kubectl exec -it fsx-app ls /data
```

Output.

```
export  out.txt
```

The sample app wrote the `out.txt` file to the file system.

Archive files to the `s3ExportPath`

For new files and modified files, you can use the Lustre user space tool to archive the data back to Amazon S3 using the value that you specified for `s3ExportPath`.

1. Export the file back to Amazon S3.

   ```
   kubectl exec -ti fsx-app -- lfs hsm_archive /data/out.txt
   ```

   Note

   - New files aren't synced back to Amazon S3 automatically. In order to sync files to the `s3ExportPath`, you need to install the Lustre client in your container image and manually run the `lfs hsm_archive` command. The container should run in privileged mode with the `CAP_SYS_ADMIN` capability.
   - This example uses a lifecycle hook to install the Lustre client for demonstration purpose. A normal approach is building a container image with the Lustre client.

2. Confirm that the `out.txt` file was written to the `s3ExportPath` folder in Amazon S3.

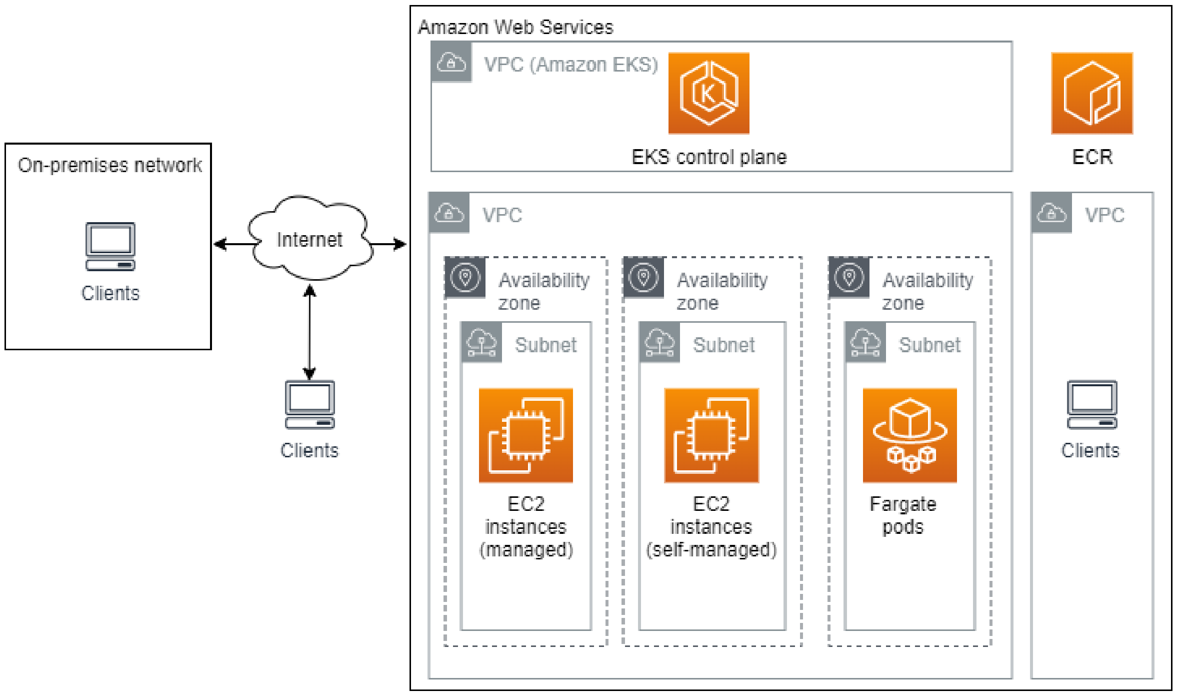
   ```
   aws s3 ls fsx-csi/export/
   ```

   Output

   ```
   2019-12-23 12:11:35       4553 out.txt
   2019-12-23 11:41:21         10 testfile
   ```

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 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 VPC tagging requirement (p. 210) and Subnet tagging (p. 210). If you deploy the VPC using an Amazon EKS provided AWS CloudFormation template (p. 206) 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. 80).
  - Any public subnets that you use must be configured to auto-assign public IP addresses for Amazon EC2 instances launched within them. For more information, see VPC IP addressing (p. 209).
  - 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. 210).
  - 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 the section called “Installing Calico on Amazon EKS” (p. 246).

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. 308), 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. 206).

- **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. 40).

  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. 206), then nodes deployed to public subnets are automatically assigned a public IP address by the subnet. Each node is deployed with the Pod networking (CNI) (p. 214) which, by default, assigns each pod a private IP address from the CIDR block assigned to the subnet that the node is in and adds the IP 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 CIDR blocks to your VPC and enabling CNI custom networking (p. 230), 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. 219).

  By default, the source IP 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 IP address to the NAT device's IP address. For more information, see External source network address translation (SNAT) (p. 215).
- **Fargate pods** – Deployed to private subnets only. Each pod is assigned a private IP 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. 125).

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 considerations (p. 208). This topic guides you through creating a VPC for your cluster using one of the following configurations:

- **Public and private subnets** – This VPC has two public and two private subnets. One public and one private subnet are deployed to the same Availability Zone. The other public and private subnets are deployed to a second Availability Zone in the same Region. We recommend this option for all production deployments. This option allows you to deploy your 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 IP addresses are automatically assigned to resources deployed to one of the public subnets, but public IP addresses are not assigned to any resources deployed to the private subnets. The nodes in private subnets can communicate with the cluster and other AWS services, and pods can communicate outbound to the internet through a NAT gateway that is deployed in each Availability Zone. A security group is deployed that denies all inbound traffic and allows all outbound traffic. The subnets are tagged so that Kubernetes is able to deploy load balancers to them. For more information about subnet tagging, see Subnet tagging (p. 210). 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 IP addresses and can send and receive internet traffic through an internet gateway. A security group is deployed that denies all inbound traffic and allows all outbound traffic. The subnets are tagged so that Kubernetes can deploy load balancers to them. For more information about subnet tagging, see Subnet tagging (p. 210). 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 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. 210). 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, 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. 41).

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. 91) – 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. 105), Windows (p. 112), or Arm (p. 149) 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.

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 a Region that supports Amazon EKS.
3. Choose **Create stack**, **With new resources (standard)**.
4. For **Choose a template**, select **Specify an Amazon S3 template URL**.
5. Paste the following URL into the text area and choose **Next**:

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

6. On the **Specify Details** page, fill out the parameters accordingly, and then choose **Next**.
   - **Stack name**: Choose a stack name for your AWS CloudFormation stack. For example, you can call it **eks-vpc**.
   - **VpcBlock**: Choose a CIDR range for your VPC. Each worker node, pod, and load balancer that you deploy is assigned an IP address from this block. The default value provides 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.
   - **PublicSubnet01Block**: Specify a 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
   - **PublicSubnet02Block**: Specify a 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
   - **PrivateSubnet01Block**: Specify a 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
   - **PrivateSubnet02Block**: Specify a 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
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 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.

Only public subnets

To create your cluster VPC with only public subnets

1. Open the AWS CloudFormation console at https://console.aws.amazon.com/cloudformation.
2. From the navigation bar, select a Region that supports Amazon EKS.
3. Choose **Create stack**, **With new resources (standard)**.
4. For **Choose a template**, select **Specify an Amazon S3 template URL**.
5. Paste the following URL into the text area and choose **Next**:

```
https://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**.
   - **VpcBlock**: Choose a CIDR block for your VPC. Each worker node, pod, and load balancer that you deploy is assigned an IP address from this block. The default value provides 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.

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 a Region that supports Amazon EKS.
3. Choose **Create stack**, **With new resources (standard)**.
4. For **Choose a template**, select **Specify an Amazon S3 template URL**.
5. Paste the following URL into the text area and choose **Next**:

```
https://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**.
   - **VpcBlock**: Choose a CIDR block for your VPC. Each worker node, pod, and load balancer that you deploy is assigned an IP address from this block. The default value provides 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.

Next steps

After you have created your VPC, you can try the Getting started with Amazon EKS (p. 4) walkthrough.

Cluster VPC 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 the section called "Private clusters" (p. 80).

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. 211) 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 their description.

Make sure that the subnets that you specify during cluster creation have enough available IP addresses for the Amazon EKS created network interfaces. 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 the section called "Subnet tagging" (p. 210).
- 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.
- There is a known issue where Amazon EKS cannot communicate with nodes launched in subnets from additional CIDR blocks added to a VPC after a cluster is first created. If you are experiencing this issue, file a support ticket so Amazon EKS can manually update your cluster to recognize the additional CIDR blocks added to the VPC.
- Do not select a subnet in AWS Outposts, AWS Wavelength, or an AWS Local Zone when creating your cluster.

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

Nodes must be able to communicate with the control plane and other AWS services. If your nodes are deployed in a private subnet, then the subnet must meet one of the following requirements:

- Has a default route to a NAT gateway. The NAT gateway must be assigned a public IP address to provide internet access for the nodes.
- Is configured with the necessary settings and requirements in the section called "Private clusters" (p. 80).

If self-managed nodes are deployed to a public subnet, the subnet must be configured to auto-assign public IP addresses. Otherwise, your node instances must be assigned a public IP address when they're launched. For more information, see Assigning a public IPv4 address during instance launch in the Amazon VPC User Guide. If managed nodes are deployed to a public subnet, the subnet must be configured to auto-assign public IP addresses. If the subnet is not configured to auto-assign public IP addresses, then the nodes aren't assigned a public IP address. Determine whether your public subnets are configured to auto-assign public IP addresses with the following command. Replace the *`<example values>`* (including *`<>`*) with your own values.

```
aws ec2 describe-subnets \
    --filters "Name=vpc-id,Values=<VPC-ID>" | grep 'SubnetId\|MapPublicIpOnLaunch'
```

Output

```
"MapPublicIpOnLaunch": <false>,"SubnetId": "<subnet-
aaaaaaaaaaaaaaaaa>","MapPublicIpOnLaunch": <false>,"SubnetId": "<subnet-
bbbbbbbbbbbbbbbbb>",
```

For any subnets that have `MapPublicIpOnLaunch` set to `false`, change the setting to `true`.

```
aws ec2 modify-subnet-attribute --map-public-ip-on-launch --subnet-id <subnet-
aaaaaaaaaaaaaaaaa>
```

Important
If you used an Amazon EKS AWS AWS CloudFormation template (p. 204) 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) 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 IP 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 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, the tag is not removed from the subnets.

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

You can optionally use this tag to control where Elastic Load Balancers are provisioned, in addition to the required subnet tags for using automatically provisioned Elastic Load Balancers. For more information about load balancer subnet tagging, see the section called “Application load balancing” (p. 271) and the section called “Network load balancing” (p. 266).

VPC tagging requirement

If you created a 1.14 or earlier Amazon EKS cluster, Amazon EKS tagged the VPC containing the subnets you specified in the following way:

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

This tag is not required or created by Amazon EKS for 1.15 or later clusters. If you create a 1.15 or later cluster in a VPC that already has this tag, the tag is not removed. You can safely remove this tag from any VPC used by an Amazon EKS cluster running version 1.15 or later.

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.

Cluster security group

Amazon EKS clusters, starting with Kubernetes version 1.14 and platform version (p. 65) `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. 88) 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 <cluster_name> --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. 88) 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

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	
Minimum outbound traffic	TCP	443		Cluster security group
Minimum outbound traffic*	TCP	10250		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. 80). 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. 65) `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 <cluster_name> --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. 40) is enabled:** Any security groups that generate API server client traffic (such as `kubectl` commands on a bastion host within your cluster's VPC)	

	Protocol	Port range	Source	Destination
Recommended inbound traffic	TCP	443	All node security groups **When cluster endpoint private access (p. 40) is enabled:** Any security groups that generate API server client traffic (such as `kubectl` commands on a bastion host within your cluster's VPC)	
Minimum outbound traffic	TCP	10250		All 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	
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

*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. 80).

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. 320).

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

Pod networking (CNI)

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. 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 AWS 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. 238).

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.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.large` instance supports three network interfaces and ten private IP addresses for each network interface. Even though an `m5.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 addressess per
 network interface - 1)) + 2
```

  Using this formula, an `m5.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. System pods count towards the maximum pods. For example, the CNI plugin and `kube-proxy` pods run on every node in a cluster, so you're only able to deploy 27 additional pods to an `m5.large` instance, not 29. Further, CoreDNS runs on some of the nodes in the cluster, which decrements the maximum pods by another one for the nodes it runs on.

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. 230). 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. 219).

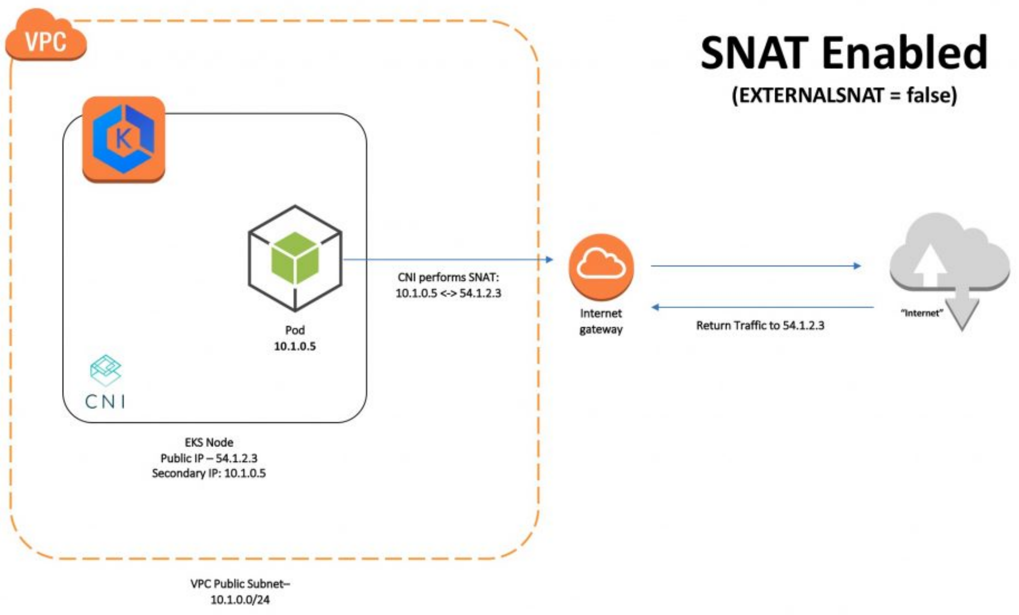
- **CNI plugin** – Responsible for wiring the host network (for example, configuring the network interfaces and virtual Ethernet pairs) and adding the correct network interface to the pod namespace.

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`. For more information, see the section called “Pod security policy” (p. 363).

External source network address translation (SNAT)

Communication within a VPC (such as pod to pod) is direct between private IP addresses and requires no source network address translation (SNAT). When traffic is destined for an address outside of the VPC, the Amazon VPC CNI plugin for Kubernetes translates the private IP address of each pod to the primary private IP address assigned to the primary network interface (network interface) of the Amazon EC2 node that the pod is running on, by default. 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.

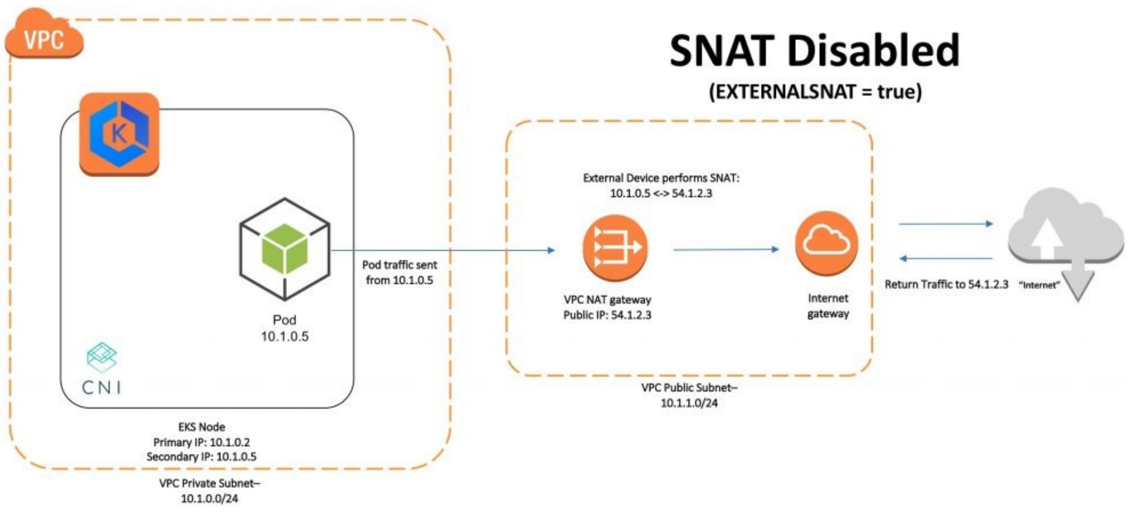
- Prevents a device in other private IP address spaces (for example, VPC peering, Transit VPC, or Direct Connect) from communicating directly to a pod that is not assigned the primary private IP address of the primary network interface of the Amazon EC2 instance node.

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 to the internet is externally translated to and from the public IP address of the NAT device and routed to and from the internet by an internet gateway, as shown in the following picture.



Configuring the 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 CNI plugin is responsible for allocating VPC IP addresses to Kubernetes nodes and configuring the necessary networking for pods on each node. The plugin:

- Requires IAM permissions, provided by the AWS managed policy `AmazonEKS_CNI_Policy`, to make calls to AWS APIs on your behalf.
- Creates and is configured to use a 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
Regardless of whether you configure the VPC CNI plugin to use IAM roles for service accounts, the pods also have access to the permissions assigned to the the section called “Node IAM role” (p. 341), unless you block access to IMDS. For more information, see the section called “Restricting access to the instance profile” (p. 359).

You can use `eksctl` or the AWS Management Console to create your CNI plugin IAM role.

eksctl

1. Create an IAM role and attach the `AmazonEKS_CNI_Policy` managed IAM policy with the following command. Replace *<cluster_name>* (including *<>*) with your own value. This command creates an IAM OIDC provider for your cluster if it doesn't already exist. It then deploys an AWS CloudFormation stack that creates an IAM role, attaches the `AmazonEKS_CNI_Policy` AWS managed policy to it, and annotates the existing `aws-node` service account with the ARN of the IAM role.

```
eksctl create iamserviceaccount \
    --name aws-node \
    --namespace kube-system \
    --cluster <cluster_name> \
    --attach-policy-arn arn:aws:iam::aws:policy/AmazonEKS_CNI_Policy \
    --approve \
    --override-existing-serviceaccounts
```

2. Describe one of the pods and verify that the `AWS_WEB_IDENTITY_TOKEN_FILE` and `AWS_ROLE_ARN` environment variables exist.

   ```
   kubectl exec -n kube-system aws-node-<9rgzw> env | grep AWS
   ```

   Output:

   ```
   AWS_VPC_K8S_CNI_LOGLEVEL=DEBUG
   AWS_ROLE_ARN=arn:aws:iam::<111122223333>:role/eksctl-prod-addon-iamserviceaccount-kube-sys-Role1-<V66K5I6JLDGK>
   AWS_WEB_IDENTITY_TOKEN_FILE=/var/run/secrets/eks.amazonaws.com/serviceaccount/token
   ```

AWS Management Console

Prerequisite

You must have an existing IAM OIDC provider for your cluster. To determine whether you already do or to create one, see Create an IAM OIDC provider for your cluster (p. 349).

To create your CNI plugin IAM role with the AWS Management Console

1. In the navigation panel, choose **Roles**, **Create Role**.
2. In the **Select type of trusted entity** section, choose **Web identity**.
3. In the **Choose a web identity provider** section:
   1. For **Identity provider**, choose the URL for your cluster.
   2. For **Audience**, choose `sts.amazonaws.com`.
4. Choose **Next: Permissions**.
5. In the **Attach Policy** section, select the `AmazonEKS_CNI_Policy` policy to use for your service account.
6. Choose **Next: Tags**.
7. On the **Add tags (optional)** screen, you can add tags for the account. Choose **Next: Review**.
8. For **Role Name**, enter a name for your role, such as *`AmazonEKSCNIRole`*, and then choose **Create Role**.
9. After the role is created, choose the role in the console to open it for editing.
10. Choose the **Trust relationships** tab, and then choose **Edit trust relationship**.
11. Find the line that looks similar to the following:

    ```
    "oidc.eks.us-west-2.amazonaws.com/id/EXAMPLED539D4633E53DE1B716D3041E:aud":
     "sts.amazonaws.com"
    ```

    Change the line to look like the following line. Replace *`<EXAMPLED539D4633E53DE1B716D3041E>`* (including *`<>`*)with your cluster's OIDC provider ID and replace *`<region-code>`* with the Region code that your cluster is in.

    ```
    "oidc.eks.<region-code>.amazonaws.com/id/<EXAMPLED539D4633E53DE1B716D3041E>:sub":
     "system:serviceaccount:kube-system:aws-node"
    ```
12. Choose **Update Trust Policy** to finish.

To annotate the `aws-node` Kubernetes service account with the IAM role

1. If you're using the Amazon EKS add-on with a 1.18 or later Amazon EKS cluster with platform version **eks.3** or later, see the section called "Configure an Amazon EKS add-on" (p. 33), instead of completing this procedure. If you're not using the Amazon VPC CNI Amazon EKS add-on, then use the following command to annotate the `aws-node` service account with the ARN of the IAM role that you created previously. Be sure to substitute your own values for the `<example values>` to use with your pods.

```
kubectl annotate serviceaccount \
  -n kube-system aws-node \
  eks.amazonaws.com/role-arn=arn:aws:iam::<AWS_ACCOUNT_ID>:role/<AmazonEKSCNIRole>
```

2. Delete and re-create any existing pods that are associated with the service account to apply the credential environment variables. The mutating web hook does not apply them to pods that are already running. The following command deletes the existing the `aws-node` DaemonSet pods and deploys them with the service account annotation.

```
kubectl delete pods -n kube-system -l k8s-app=aws-node
```

3. Confirm that the pods all restarted.

```
kubectl get pods -n kube-system  -l k8s-app=aws-node
```

4. Describe one of the pods and verify that the `AWS_WEB_IDENTITY_TOKEN_FILE` and `AWS_ROLE_ARN` environment variables exist.

```
kubectl exec -n kube-system aws-node-<9rgzw> env | grep AWS
```

Output:

```
AWS_VPC_K8S_CNI_LOGLEVEL=DEBUG
AWS_ROLE_ARN=arn:aws:iam::<AWS_ACCOUNT_ID>:role/<IAM_ROLE_NAME>
AWS_WEB_IDENTITY_TOKEN_FILE=/var/run/secrets/eks.amazonaws.com/serviceaccount/token
```

Remove the CNI policy from the node IAM role

If your the section called "Node IAM role" (p. 341) currently has the `AmazonEKS_CNI_Policy` IAM policy 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.

1. Open the IAM console at https://console.aws.amazon.com/iam/.
2. In the left navigation, choose **Roles**, and then search for your node instance role.
3. Choose the **Permissions** tab for your node instance role and then choose the **X** to the right of the `AmazonEKS_CNI_Policy`.
4. Choose **Detach** to finish.

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. 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:

- Your Amazon EKS cluster must be running Kubernetes version 1.17 and Amazon EKS platform version `eks.3` or later. You can't use security groups for pods on Kubernetes clusters that you deployed to Amazon EC2.
- Traffic flow to and from pods with associated security groups are not subjected to Calico network policy (p. 246) enforcement and are limited to Amazon EC2 security group enforcement only. Community effort is underway to remove this limitation.
- Security groups for pods can't be used with pods deployed to Fargate.
- Security groups for pods can't be used with Windows nodes.
- 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 section called "Supported instances" (p. 224). Your nodes must be one of the supported instance types.
- 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.
- 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 the section called "Load balancer – Instance targets" (p. 268).
- 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 the `Role` that your `psp` is assigned to. If you're using the default Amazon EKS `psp`, `Role`, and `ClusterRoleBinding` (p. 363), this is the `eks:podsecuritypolicy:authenticated` `ClusterRoleBinding`. For example, you would 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. 230) 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`.
- Pods using security groups must contain `terminationGracePeriodSeconds` in their pod spec. This is because the Amazon EKS VPC CNI plugin queries the API server to retrieve the pod IP address before deleting the pod network on the host. Without this setting, the plugin doesn't remove the pod network on the host.

Deploy security groups for pods

To deploy security groups for pods

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
   ```

   The output is similar to the following output.

   ```
   amazon-k8s-cni:<1.7.7>
   ```

   If your CNI plugin version is earlier than 1.7.7, then upgrade your CNI plugin to version 1.7.7 or later. For more information, see Amazon VPC CNI plugin for Kubernetes upgrades (p. 236).
2. Add the `AmazonEKSVPCResourceController` managed policy to the cluster role (p. 340) 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 instances. The following command adds the policy to a cluster role named `<eksClusterRole>`.

   ```
   aws iam attach-role-policy \
       --policy-arn arn:aws:iam::aws:policy/AmazonEKSVPCResourceController \
       --role-name <eksClusterRole>
   ```
3. Enable the CNI plugin 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 plugin 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 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.

   ```
   kubectl get nodes -o wide -l vpc.amazonaws.com/has-trunk-attached=true
   ```

   Once the trunk network interface is created, pods can be 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 are you using liveness or readiness probes, you also need to disable TCP early demux, so that the `kubelet` can connect to pods on branch network interfaces via 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"}]}}}}'
```

4. Create a namespace to deploy resources to.

```
kubectl create namespace <my-namespace>
```

5. 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 the cluster security group (for CoreDNS) over TCP and UDP port 53. The cluster security group must also allow inbound TCP and UDP port 53 communication from all security groups associated to pods.

   b. Deploy the policy.

```
kubectl apply -f <my-security-group-policy.yaml>
```

6. 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 <file-name-you-used-in-previous-step.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 the section called "Supported instances" (p. 224) and that 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-xxxxxxxxxx-xxxxx> -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-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.

Amazon EC2 supported instances and branch network interfaces

The following table lists the number of branch network interfaces that you can use with each supported Amazon EC2 instance type.

Instance type	Branch network interfaces
a1.medium	10
a1.large	9
a1.xlarge	18
a1.2xlarge	38
a1.4xlarge	54
a1.metal	54
c5.large	9
c5.xlarge	18
c5.2xlarge	38
c5.4xlarge	54
c5.9xlarge	54
c5.12xlarge	54
c5.18xlarge	107
c5.24xlarge	107
c5.metal	107
c5a.large	9
c5a.xlarge	18
c5a.2xlarge	38
c5a.4xlarge	54
c5a.8xlarge	54
c5a.12xlarge	54
c5a.16xlarge	107
c5a.24xlarge	107

Instance type	Branch network interfaces
c5d.large	9
c5d.xlarge	18
c5d.2xlarge	38
c5d.4xlarge	54
c5d.9xlarge	54
c5d.12xlarge	54
c5d.18xlarge	107
c5d.24xlarge	107
c5d.metal	107
c5n.large	9
c5n.xlarge	18
c5n.2xlarge	38
c5n.4xlarge	54
c5n.9xlarge	54
c5n.18xlarge	107
c5n.metal	107
c6g.medium	4
c6g.large	9
c6g.xlarge	18
c6g.2xlarge	38
c6g.4xlarge	54
c6g.8xlarge	54
c6g.12xlarge	54
c6g.16xlarge	107
c6g.metal	107
g4dn.xlarge	39
g4dn.2xlarge	39
g4dn.4xlarge	59
g4dn.8xlarge	58
g4dn.12xlarge	54
g4dn.16xlarge	118

Instance type	Branch network interfaces
g4dn.metal	107
i3en.large	5
i3en.xlarge	12
i3en.2xlarge	28
i3en.3xlarge	38
i3en.6xlarge	54
i3en.12xlarge	114
i3en.24xlarge	107
i3en.metal	107
inf1.xlarge	38
inf1.2xlarge	38
inf1.6xlarge	54
inf1.24xlarge	107
m5.large	9
m5.xlarge	18
m5.2xlarge	38
m5.4xlarge	54
m5.8xlarge	54
m5.12xlarge	54
m5.16xlarge	107
m5.24xlarge	107
m5.metal	107
m5a.large	9
m5a.xlarge	18
m5a.2xlarge	38
m5a.4xlarge	54
m5a.8xlarge	54
m5a.12xlarge	54
m5a.16xlarge	107
m5a.24xlarge	107
m5ad.large	9

Instance type	Branch network interfaces
m5ad.xlarge	18
m5ad.2xlarge	38
m5ad.4xlarge	54
m5ad.8xlarge	54
m5ad.12xlarge	54
m5ad.16xlarge	107
m5ad.24xlarge	107
m5d.large	9
m5d.xlarge	18
m5d.2xlarge	38
m5d.4xlarge	54
m5d.8xlarge	54
m5d.12xlarge	54
m5d.16xlarge	107
m5d.24xlarge	107
m5d.metal	107
m5dn.large	9
m5dn.xlarge	18
m5dn.2xlarge	38
m5dn.4xlarge	54
m5dn.8xlarge	54
m5dn.12xlarge	54
m5dn.16xlarge	107
m5dn.24xlarge	107
m5n.large	9
m5n.xlarge	18
m5n.2xlarge	38
m5n.4xlarge	54
m5n.8xlarge	54
m5n.12xlarge	54
m5n.16xlarge	107

Instance type	Branch network interfaces
m5n.24xlarge	107
m6g.medium	4
m6g.large	9
m6g.xlarge	18
m6g.2xlarge	38
m6g.4xlarge	54
m6g.8xlarge	54
m6g.12xlarge	54
m6g.16xlarge	107
m6g.metal	107
p3.2xlarge	38
p3.8xlarge	54
p3.16xlarge	114
p3dn.24xlarge	107
r5.large	9
r5.xlarge	18
r5.2xlarge	38
r5.4xlarge	54
r5.8xlarge	54
r5.12xlarge	54
r5.16xlarge	107
r5.24xlarge	107
r5.metal	107
r5a.large	9
r5a.xlarge	18
r5a.2xlarge	38
r5a.4xlarge	54
r5a.8xlarge	54
r5a.12xlarge	54
r5a.16xlarge	107
r5a.24xlarge	107

Instance type	Branch network interfaces
r5ad.large	9
r5ad.xlarge	18
r5ad.2xlarge	38
r5ad.4xlarge	54
r5ad.8xlarge	54
r5ad.12xlarge	54
r5ad.16xlarge	107
r5ad.24xlarge	107
r5d.large	9
r5d.xlarge	18
r5d.2xlarge	38
r5d.4xlarge	54
r5d.8xlarge	54
r5d.12xlarge	54
r5d.16xlarge	107
r5d.24xlarge	107
r5d.metal	107
r5dn.large	9
r5dn.xlarge	18
r5dn.2xlarge	38
r5dn.4xlarge	54
r5dn.8xlarge	54
r5dn.12xlarge	54
r5dn.16xlarge	107
r5dn.24xlarge	107
r5n.large	9
r5n.xlarge	18
r5n.2xlarge	38
r5n.4xlarge	54
r5n.8xlarge	54
r5n.12xlarge	54

Instance type	Branch network interfaces
r5n.16xlarge	107
r5n.24xlarge	107
r6g.medium	4
r6g.large	9
r6g.xlarge	18
r6g.2xlarge	38
r6g.4xlarge	54
r6g.8xlarge	54
r6g.12xlarge	54
r6g.16xlarge	107
z1d.large	13
z1d.xlarge	28
z1d.2xlarge	58
z1d.3xlarge	54
z1d.6xlarge	54
z1d.12xlarge	107
z1d.metal	107

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. 215).

Note

- You can configure custom networking for self-managed node groups or for managed node groups that were created with a launch template that uses a custom AMI (p. 103). The procedures in this topic require the Amazon VPC CNI plugin for Kubernetes version 1.4.0 or later. To check your CNI version, and upgrade if necessary, see Amazon VPC CNI plugin for Kubernetes upgrades (p. 236).

- 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 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 will still share use of the secondary network interfaces and will all use the same security groups. If you want to assign different security groups to individual pods, then you can use Security groups for pods (p. 219). 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.

To configure CNI custom networking

1. Associate a secondary CIDR block to your cluster's VPC. For more information, see Associating a Secondary IPv4 CIDR Block with Your VPC in the *Amazon VPC User Guide*.
2. Create a subnet in your VPC for each Availability Zone, using your secondary CIDR block. Your custom subnets must be from a different VPC CIDR block than the subnet that your nodes were launched into. For more information, see Creating a subnet in your VPC in the *Amazon VPC User Guide*.
3. Set the `AWS_VPC_K8S_CNI_CUSTOM_NETWORK_CFG` environment variable to `true` in the `aws-node` DaemonSet:

```
kubectl set env daemonset aws-node -n kube-system
 AWS_VPC_K8S_CNI_CUSTOM_NETWORK_CFG=true
```

4. View the currently-installed CNI version.

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

   Output:

```
amazon-k8s-cni:1.7.5
```

5. If you have version 1.3 or later of the CNI installed, you can skip to step 6. Define a new `ENIConfig` custom resource for your cluster.

   a. Create a file called `ENIConfig.yaml` and paste the following content into it:

```
apiVersion: apiextensions.k8s.io/v1beta1
kind: CustomResourceDefinition
metadata:
  name: eniconfigs.crd.k8s.amazonaws.com
spec:
  scope: Cluster
  group: crd.k8s.amazonaws.com
  version: v1alpha1
  names:
    plural: eniconfigs
    singular: eniconfig
    kind: ENIConfig
```

   b. Apply the file to your cluster with the following command:

```
kubectl apply -f ENIConfig.yaml
```

6. Create an `ENIConfig` custom resource for each subnet that you want to schedule pods in.

   a. Create a unique file for each 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 6c below).

      In this example, a file named `us-west-2a.yaml` is created. Replace the *`<example values>`* (including `<>`) for `name`, `subnet`, and `securityGroups` with your own values. In this example, we follow best practices and set the value for `name` to the Availability Zone that the subnet is in. If you don't have a specific security group that you want to attach for your pods, you can leave that value empty for now. Later, you will specify the 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-011b111c1f11fdf11>
      ```

      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>.compute.internal
         k8s.amazonaws.com/eniConfig=<subnet1ConfigName>
        ```

      - If you don't specify a valid security group for the VPC, the default security group for the VPC is assigned to secondary ENI's.

   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 `failure-domain.beta.kubernetes.io/zone` label. If you named your `ENIConfig` custom resources after each Availability Zone in your VPC, as recommended in step 6a, 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=failure-
      domain.beta.kubernetes.io/zone
      ```

      Note
      Ensure that an annotation with the key `k8s.amazonaws.com/eniConfig` for the `ENI_CONFIG_ANNOTATION_DEF` environment variable doesn't exist in the container spec for the `aws-node` daemonset. If it exists, it overrides the `ENI_CONFIG_LABEL_DEF` value, and should be removed. You can check to see if the variable is set with the `kubectl describe daemonset aws-node -n kube-system | grep ENI_CONFIG_ANNOTATION_DEF` command. If no output is returned, then the variable is not set.

7. Create a new self-managed node group. For managed node groups, use a custom AMI with a launch template (p. 103).

a. Determine the maximum number of pods that can be scheduled on each node using the following formula.

```
maxPods = (number of interfaces - 1) * (max IPv4 addresses per interface - 1) + 2
```

For example, the `m5.large` instance type supports three network interfaces and ten IPv4 addresses for each interface. Inserting the values into the formula, the instance can support up to 20 pods, as shown in the following calculation.

```
maxPods = (3 - 1) * (10 - 1) + 2 = 20
```

For more information about the maximum number of network interfaces for each instance type, see IP addresses per network interface per instance type in the *Amazon EC2 User Guide for Linux Instances*.

b. Follow the steps for **Self-managed nodes** in Launching self-managed Amazon Linux nodes (p. 105) to create a new self-managed node group. Do not specify the subnets that you specified in the `ENIConfig` resources that you deployed. After you've opened the AWS CloudFormation template, enter values as described in the instructions. For the **BootstrapArguments** field, enter the following value.

```
--use-max-pods false --kubelet-extra-args '--max-pods=<20>'
```

8. After your 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 from steps 6a and 6c, the `eniconfig-name` corresponds to the Availability Zone name.

The `spec` section should look like this:

```
spec:
  securityGroups:
  - <sg-0dff222a2d22c2c22>
  subnet: <subnet-022b222c2f22fdf22>
```

9. If you 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.

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.

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

Deploying the CNI metrics helper

The CNI metrics helper requires `cloudwatch:PutMetricData` permissions to send metric data to CloudWatch. This section helps you to create an IAM policy with those permissions, apply it to your node instance role, and then deploy the CNI metrics helper.

To create an IAM policy for the CNI metrics helper

1. Create a file called `allow_put_metrics_data.json` and populate it with the following policy document.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "cloudwatch:PutMetricData",
      "Resource": "*"
    }
  ]
}
```

2. Create an IAM policy called `CNIMetricsHelperPolicy` for your node instance profile that allows the CNI metrics helper to make calls to AWS APIs on your behalf. Use the following AWS CLI command to create the IAM policy in your AWS account.

```
aws iam create-policy --policy-name CNIMetricsHelperPolicy \
--description "Grants permission to write metrics to CloudWatch" \
--policy-document file://allow_put_metrics_data.json
```

   Take note of the policy ARN that is returned.

3. Get the IAM role name for your nodes. Use the following command to print the `aws-auth` configmap.

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

Output:

```
Name:         aws-auth
Namespace:    kube-system
Labels:       <none>
Annotations:  <none>

Data
====
mapRoles:
----
- groups:
  - system:bootstrappers
  - system:nodes
  rolearn: arn:aws:iam::<111122223333>:role/<eksctl-prod-nodegroup-standard-wo-
NodeInstanceRole-GKNS581EASPU>
  username: system:node:{{EC2PrivateDNSName}}

Events:  <none>
```

Record the role name for any `rolearn` values that have the `system:nodes` group assigned to them. In the above example output, the role name is <eksctl-prod-nodegroup-standard-wo-NodeInstanceRole-GKNS581EASPU>. You should have one value for each node group in your cluster.

4. Attach the new `CNIMetricsHelperPolicy` IAM policy to each of the node IAM roles you identified earlier with the following command, substituting the red text with your own AWS account number and node IAM role name.

```
aws iam attach-role-policy \
--policy-arn arn:aws:iam::<111122223333>:policy/CNIMetricsHelperPolicy \
--role-name <eksctl-prod-nodegroup-standard-wo-NodeInstanceRole-GKNS581EASPU>
```

To deploy the CNI metrics helper

- Apply the CNI metrics helper manifest with the command that corresponds to the Region that your cluster is in.
  - All regions other than China Regions.

```
kubectl apply -f https://raw.githubusercontent.com/aws/amazon-vpc-cni-k8s/
release-1.7/config/v1.7/cni-metrics-helper.yaml
```

  - Beijing and Ningxia China Regions.

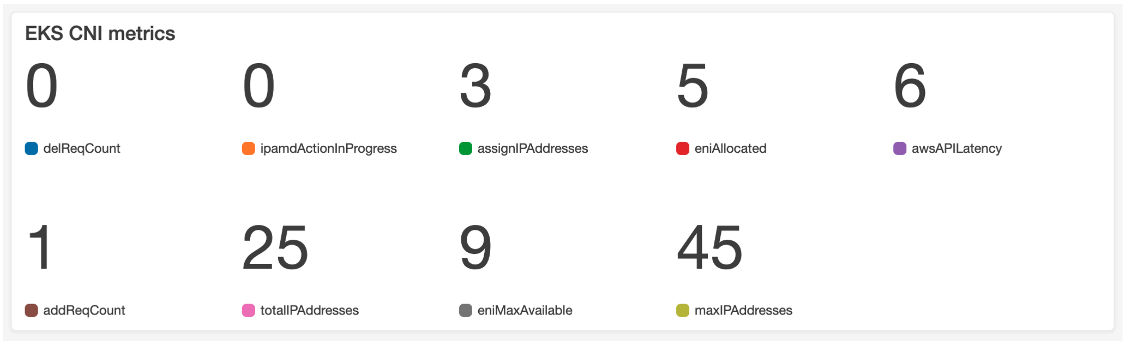
```
kubectl apply -f https://raw.githubusercontent.com/aws/amazon-vpc-cni-k8s/master/
config/v1.7/cni-metrics-helper-cn.yaml
```

Creating a metrics dashboard

After you have deployed the CNI metrics helper, you can view the CNI metrics in the CloudWatch console. This topic helps you to create a dashboard for viewing your cluster's CNI metrics.

To create a CNI metrics dashboard

1. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.
2. In the left navigation, choose **Metrics**.
3. Under **Custom Namespaces**, choose **Kubernetes**.
4. Choose **CLUSTER_ID**.
5. On the **All metrics** tab, select the metrics you want to add to the dashboard.
6. Choose **Actions**, and then **Add to dashboard**.
7. In the **Select a dashboard** section, choose **Create new** and enter a name for your dashboard, such as "EKS-CNI-metrics".
8. In the **Select a widget type** section, choose **Number**.
9. In the **Customize the widget title** section, enter a logical name for your dashboard title, such as "EKS CNI metrics".
10. Choose **Add to dashboard** to finish. Now your CNI metrics are added to a dashboard that you can monitor, as shown below.


EKS CNI metrics

0	0	3	5	6
delReqCount	ipamdActionInProgress	assignIPAddresses	eniAllocated	awsAPILatency
1	25	9	45	
addReqCount	totalIPAddresses	eniMaxAvailable	maxIPAddresses	

Amazon VPC CNI plugin for Kubernetes upgrades

When you launch an Amazon EKS cluster, a recent version of the Amazon VPC CNI plugin for Kubernetes is deployed to your cluster. The absolute latest version of the plugin is available on GitHub for a short grace period before new clusters are switched over to use it. Amazon EKS does not automatically upgrade the CNI plugin on your cluster when new versions are released. To get a newer version of the CNI plugin on existing clusters, you must manually upgrade the plugin.

The latest version that we recommend is version 1.7.5. You can view the different releases available for the plugin, and read the release notes for each version on GitHub. With version 1.7.0 and later, the `privileged` container capability was removed from the CNI pod (`aws-node`). The pod has the `NET_ADMIN` capability in its `securityContext capabilities`, which is required for the `aws-node` container to add `iptables`, routes, and rules to setup pod networking. An `init` container was also added to the `aws-node` pod, which has the `privileged` capability, so that it can setup reverse path filters and copy loopback plugins during `aws-node` pod start up.

Important
If you have assigned a custom pod security policy to the `aws-node` Kubernetes service account used for the `aws-node` pod, then the policy must have `NET_ADMIN` in its `allowedCapabilities` section along with `hostNetwork: true` and `privileged: true` in the policy's `spec`. For more information, see the section called "Pod security policy" (p. 363).

Use the following procedures to check your CNI plugin version and upgrade to the latest recommended version.

To check your Amazon VPC CNI plugin for Kubernetes version

- Use the following command to print your cluster's CNI version:

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

  Output:

  ```
  amazon-k8s-cni:1.6.3
  ```

  In this example output, the CNI version is 1.6.3, which is earlier than the current recommended version, 1.7.5. Use the following procedure to upgrade the CNI.

To upgrade the Amazon VPC CNI plugin for Kubernetes

- If your CNI version is earlier than 1.7.5, and you are managing the plugin yourself, then use the appropriate command below to update your CNI version to the latest recommended version. If your cluster is running Kubernetes `1.18` or later with `eks.3` platform version or later, and the plugin is managed by Amazon EKS, then to update the plugin, see the section called "Configure an Amazon EKS add-on" (p. 33).

  Important
  Any changes you've made to the plugin's default settings on your cluster can be overwritten with default settings when applying the new version of the manifest. To prevent loss of your custom settings, download the manifest, change the default settings as necessary, and then apply the modified manifest to your cluster.

  - US West (Oregon) (`us-west-2`)

    ```
    kubectl apply -f https://raw.githubusercontent.com/aws/amazon-vpc-cni-k8s/v1.7.5/
    config/v1.7/aws-k8s-cni.yaml
    ```

  - China (Beijing) (`cn-north-1`) or China (Ningxia) (`cn-northwest-1`)

    ```
    kubectl apply -f https://raw.githubusercontent.com/aws/amazon-vpc-cni-k8s/v1.7.5/
    config/v1.7/aws-k8s-cni-cn.yaml
    ```

  - AWS GovCloud (US-East) (`us-gov-east-1`)

    ```
    kubectl apply -f https://raw.githubusercontent.com/aws/amazon-vpc-cni-k8s/v1.7.5/
    config/v1.7/aws-k8s-cni-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/v1.7.5/
    config/v1.7/aws-k8s-cni-us-gov-west-1.yaml
    ```

  - For all other Regions
    - Download the manifest file.

      ```
      curl -o aws-k8s-cni.yaml https://raw.githubusercontent.com/aws/amazon-vpc-cni-k8s/
      v1.7.5/config/v1.7/aws-k8s-cni.yaml
      ```

- If necessary, replace `<region-code>` in the following command with the Region that your cluster is in and then run the modified command to replace the Region code in the file (currently `us-west-2`).

```
sed -i.bak -e 's/us-west-2/<region-code>/' aws-k8s-cni.yaml
```

- Apply the manifest file to your cluster.

```
kubectl apply -f aws-k8s-cni.yaml
```

Alternate compatible CNI plugins

Amazon EKS only officially supports the Amazon VPC CNI plugin (p. 214). 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. Refer to 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.

AWS Load Balancer Controller

The AWS Load Balancer Controller manages AWS Elastic Load Balancers for a Kubernetes cluster. The controller provisions:

- 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` using IP targets on 1.18 or later Amazon EKS clusters. If you're load balancing network traffic to instance targets, then you use the in-tree Kubernetes load balancer controller and don't need to install this controller. For more information about NLB target types, see Target type in the User Guide for Network Load Balancers.

The controller was formerly named the *AWS ALB Ingress Controller*. It is an open source project managed on GitHub. This topic helps you 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 the section called “Application load balancing” (p. 271) and the section called “Network load balancing” (p. 266). Those topics also include steps to deploy a sample application that require the controller to provision AWS resources.

Prerequisite

An existing cluster. If you don't have an existing cluster, see *Getting started with Amazon EKS* (p. 4).

To deploy the AWS Load Balancer Controller to an Amazon EKS cluster

In the following steps, replace the `<example values>` (including `<>`) with your own values.

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 <cluster_name> --query "cluster.identity.oidc.issuer"
    --output text
   ```

   Example output:

   ```
   https://oidc.eks.us-west-2.amazonaws.com/id/EXAMPLED539D4633E53DE1B716D3041E
   ```

   List the IAM OIDC providers in your account. Replace *<EXAMPLED539D4633E53DE1B716D3041E>* (including *<>*) with the value returned from the previous command.

   ```
   aws iam list-open-id-connect-providers | grep <EXAMPLED539D4633E53DE1B716D3041E>
   ```

   Example output

   ```
   "Arn": "arn:aws:iam::111122223333:oidc-provider/oidc.eks.us-west-2.amazonaws.com/
   id/EXAMPLED539D4633E53DE1B716D3041E"
   ```

   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.

   To create an IAM OIDC provider, see Create an IAM OIDC provider for your cluster (p. 349).
2. Download an IAM policy for the AWS Load Balancer Controller that allows it to make calls to AWS APIs on your behalf. You can view the policy document on GitHub.

   ```
   curl -o iam_policy.json https://raw.githubusercontent.com/kubernetes-sigs/aws-load-
   balancer-controller/v2.1.3/docs/install/iam_policy.json
   ```
3. Create an IAM policy using the policy downloaded in the previous step.

   ```
   aws iam create-policy \
       --policy-name AWSLoadBalancerControllerIAMPolicy \
       --policy-document file://iam_policy.json
   ```

   Take note of the policy ARN that is returned.
4. Create an IAM role and annotate the Kubernetes service account named `aws-load-balancer-controller` in the `kube-system` namespace for the AWS Load Balancer Controller using `eksctl` or the AWS Management Console and `kubectl`.

   eksctl

      Use the following command:

```
  eksctl create iamserviceaccount \
  --cluster=my-cluster \
  --namespace=kube-system \
  --name=aws-load-balancer-controller \
  --attach-policy-
arn=arn:aws:iam::<AWS_ACCOUNT_ID>:policy/AWSLoadBalancerControllerIAMPolicy \
  --override-existing-serviceaccounts \
  --approve
```

AWS Management Console

Using the AWS Management Console and `kubectl`

a. Open the IAM console at https://console.aws.amazon.com/iam/.
b. In the navigation panel, choose **Roles**, **Create Role**.
c. In the **Select type of trusted entity** section, choose **Web identity**.
d. In the **Choose a web identity provider** section:
   i. For **Identity provider**, choose the URL for your cluster.
   ii. For **Audience**, choose `sts.amazonaws.com`.
e. Choose **Next: Permissions**.
f. In the **Attach Policy** section, select the *`AWSLoadBalancerControllerIAMPolicy`* policy that you created in step 3 to use for your service account.
g. Choose **Next: Tags**.
h. On the **Add tags (optional)** screen, you can add tags for the account. Choose **Next: Review**.
i. For **Role Name**, enter a name for your role, such as *`AmazonEKSLoadBalancerControllerRole`*, and then choose **Create Role**.
j. After the role is created, choose the role in the console to open it for editing.
k. Choose the **Trust relationships** tab, and then choose **Edit trust relationship**.
l. Find the line that looks similar to the following:

```
"oidc.eks.us-west-2.amazonaws.com/id/EXAMPLED539D4633E53DE1B716D3041E:aud":
 "sts.amazonaws.com"
```

Change the line to look like the following line. Replace *`<EXAMPLED539D4633E53DE1B716D3041E>`* (including *`<>`*) with your cluster's OIDC provider ID and replace *`<region-code>`* with the Region code that your cluster is in.

```
"oidc.eks.<region-code>.amazonaws.com/id/<EXAMPLED539D4633E53DE1B716D3041E>:sub":
 "system:serviceaccount:kube-system:aws-load-balancer-controller"
```

m. Choose **Update Trust Policy** to finish.
n. Note the ARN of the role for use in a later step.
o. Save the following contents to a file named *`aws-load-balancer-controller-service-account.yaml`*.

```
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::<AWS_ACCOUNT_ID>:role/AmazonEKSLoadBalancerControllerRole
```

p. Create the service account on your cluster.

```
kubectl apply -f aws-load-balancer-controller-service-account.yaml
```

5. If you currently have the AWS ALB Ingress Controller for Kubernetes installed, uninstall it. The AWS Load Balancer Controller replaces the functionality of the AWS ALB Ingress Controller for Kubernetes.

   a. Check to see if the controller is currently installed.

   ```
   kubectl get deployment -n kube-system alb-ingress-controller
   ```

   Output (if it is installed). Skip to step 5b.

   ```
   NAME                   READY UP-TO-DATE AVAILABLE AGE
   alb-ingress-controller 1/1   1          1         122d
   ```

   Output (if it isn't installed). If it isn't installed, skip to step 6.

   ```
   Error from server (NotFound): deployments.apps "alb-ingress-controller" not found
   ```

   b. If the controller is installed, enter the following commands to remove it.

   ```
   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. If you removed the AWS ALB Ingress Controller for Kubernetes, add the following IAM policy to the IAM role created in step 4. 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/v2.1.3/docs/install/
      iam_policy_v1_to_v2_additional.json
      ```

      2. Create the IAM policy and note the ARN returned.

      ```
      aws iam create-policy \
        --policy-name AWSLoadBalancerControllerAdditionalIAMPolicy \
        --policy-document file://iam_policy_v1_to_v2_additional.json
      ```

      3. Attach the IAM policy to the IAM role that you created in step 4. Replace `<your-role-name>` (including <>) 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`.

      ```
      aws iam attach-role-policy \
      ```

```
  --role-name eksctl-<your-role name>\
  --policy-arn arn:aws:iam::111122223333:policy/
AWSLoadBalancerControllerAdditionalIAMPolicy
```

6. Install the AWS Load Balancer Controller using Helm V3 or later or by applying a Kubernetes manifest.

   Helm V3 or later

   a. Install the `TargetGroupBinding` custom resource definitions.

   ```
   kubectl apply -k "github.com/aws/eks-charts/stable/aws-load-balancer-controller//
   crds?ref=master"
   ```

   b. Add the `eks-charts` repository.

   ```
   helm repo add eks https://aws.github.io/eks-charts
   ```

   c. Install the AWS Load Balancer Controller using the command that corresponds to the Region that your cluster is in.

      Important
      If you are deploying the controller to Amazon EC2 nodes that you have restricted access to the Amazon EC2 instance metadata service (IMDS) (p. 359) from, or if you are deploying to Fargate, then you need to add the following flags to the command that you run:

      - `--set region=<region-code>`
      - `--set vpcId=<vpc-xxxxxxxx>`

   ```
   helm upgrade -i aws-load-balancer-controller eks/aws-load-balancer-controller \
     --set clusterName=<cluster-name> \
     --set serviceAccount.create=false \
     --set serviceAccount.name=aws-load-balancer-controller \
     -n kube-system
   ```

      Important
      The deployed chart does not receive security updates automatically. You need to manually upgrade to a newer chart when it becomes available.

   Kubernetes manifest

   a. Install `cert-manager` to inject certificate configuration into the webhooks.

      - Install on Kubernetes `1.16` or later.

      ```
      kubectl apply --validate=false -f https://github.com/jetstack/cert-manager/
      releases/download/v1.1.1/cert-manager.yaml
      ```

      - Install on Kubernetes `1.15`.

      ```
       kubectl apply --validate=false -f https://github.com/jetstack/cert-manager/
      releases/download/v1.1.1/cert-manager-legacy.yaml
      ```

   b. Install the controller.

      1. Download the controller specification. For more information about the controller, see the documentation on GitHub.

```
curl -o v2_1_3_full.yaml https://raw.githubusercontent.com/kubernetes-sigs/aws-load-balancer-controller/v2.1.3/docs/install/v2_1_3_full.yaml
```

2. Make the following edits to the `v2_1_3_full.yaml` file:
   - Delete the `ServiceAccount` section from the specification. Doing so prevents the annotation with the IAM role from being overwritten when the controller is deployed and preserves the service account that you created in step 4 if you delete the controller.
   - Set the `--cluster-name` value to your Amazon EKS cluster name in the `Deployment` `spec` section.
3. Apply the file.

```
kubectl apply -f v2_1_3_full.yaml
```

7. Verify that the controller is installed.

```
kubectl get deployment -n kube-system aws-load-balancer-controller
```

Output

```
NAME                           READY   UP-TO-DATE   AVAILABLE   AGE
aws-load-balancer-controller   1/1     1            1           84s
```

8. Before using the controller to provision AWS resources, your cluster must meet specific requirements. For more information, see the section called "Application load balancing" (p. 271) and the section called "Network load balancing" (p. 266).

Installing or upgrading CoreDNS

CoreDNS is supported on Amazon EKS clusters with Kubernetes version 1.16 or later. Clusters that were created with Kubernetes version 1.10 shipped with `kube-dns` as the default DNS and service discovery provider. If you have updated from a 1.10 cluster and you want to use CoreDNS for DNS and service discovery, then you must install CoreDNS and remove `kube-dns`.

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

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

If the output shows `coredns` in the pod names, then you're already running CoreDNS in your cluster. If not, use the following procedure to update your DNS and service discovery provider to CoreDNS.

Note
The service for CoreDNS is still called `kube-dns` for backward compatibility.

To install CoreDNS on an updated Amazon EKS cluster with `kubectl`

1. Add the `{"eks.amazonaws.com/component": "kube-dns"}` selector to the `kube-dns` deployment for your cluster. This prevents the two DNS deployments from competing for control of the same set of labels.

```
kubectl patch -n kube-system deployment/kube-dns --patch \
'{"spec":{"selector":{"matchLabels":{"eks.amazonaws.com/component":"kube-dns"}}}}'
```

2. Deploy CoreDNS to your cluster.

a. Set your cluster's DNS IP address to the `DNS_CLUSTER_IP` environment variable.

```
export DNS_CLUSTER_IP=$(kubectl get svc -n kube-system kube-dns -o
 jsonpath='{.spec.clusterIP}')
```

b. Replace *<region-code>* (including *<>*) with the Region code that your cluster is in.

```
export REGION="<region-code>"
```

c. Download the CoreDNS manifest.

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

d. Replace the variable placeholders in the `dns.yaml` file with your environment variable values and apply the updated manifest to your cluster. The following command completes this in one step.

```
cat dns.yaml | sed -e "s/REGION/$REGION/g" | sed -e "s/DNS_CLUSTER_IP/
$DNS_CLUSTER_IP/g" | kubectl apply -f -
```

e. Fetch the `coredns` pod name from your cluster.

```
COREDNS_POD=$(kubectl get pod -n kube-system -l eks.amazonaws.com/component=coredns
 \
-o jsonpath='{.items[0].metadata.name}')
```

f. Query the `coredns` pod to ensure that it's receiving requests.

```
kubectl get --raw /api/v1/namespaces/kube-system/pods/$COREDNS_POD:9153/proxy/
metrics \
| grep 'coredns_dns_request_count_total'
```

Note
It might take several minutes for the expected output to return properly, depending on the rate of DNS requests in your cluster.

In the following expected output, the number `23` is the DNS request count total.

```
# HELP coredns_dns_request_count_total Counter of DNS requests made per zone,
 protocol and family.
# TYPE coredns_dns_request_count_total counter
coredns_dns_request_count_total{family="1",proto="udp",server="dns://:53",zone="."}
 23
```

3. Update CoreDNS to the recommended version for your cluster by completing the steps in Upgrading CoreDNS (p. 245).

4. Scale down the `kube-dns` deployment to zero replicas.

```
kubectl scale -n kube-system deployment/kube-dns --replicas=0
```

5. Clean up the old `kube-dns` resources.

```
kubectl delete -n kube-system deployment/kube-dns serviceaccount/kube-dns configmap/
kube-dns
```

Upgrading CoreDNS

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

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

   Output:

   ```
   coredns:v<1.1.3>
   ```

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

Kubernetes version	1.19	1.18	1.17	1.16	1.15
CoreDNS	1.8.0	1.7.0	1.6.6	1.6.6	1.6.6

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

   a. Open the configmap with the following command.

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

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

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

3. 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. Update `coredns` to the recommended version by replacing the *account ID* and *Region* using the output from the previous step and and replacing *<1.8.0>* (including *<>*) with your cluster's recommended `coredns` version:

   ```
   kubectl set image --namespace kube-system deployment.apps/coredns \
               coredns=<602401143452.dkr.ecr.us-west-2.amazonaws.com>/eks/
   coredns:v<1.8.0>-eksbuild.1
   ```

Installing Calico on Amazon EKS

Project Calico is a network policy engine for Kubernetes. With Calico network policy enforcement, you can implement network segmentation and tenant isolation. This is useful in multi-tenant environments where you must isolate tenants from each other or when you want to create separate environments for development, staging, and production. Network policies are similar to AWS security groups in that you can create network ingress and egress rules. Instead of assigning instances to a security group, you assign network policies to pods using pod selectors and labels. The following procedure shows you how to install Calico on Linux nodes in your Amazon EKS cluster. To install Calico on Windows nodes, see Using Calico on Amazon EKS Windows Containers.

Note

- 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 security groups for pods (p. 219), 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. Community effort is underway to remove this limitation.

To install Calico on your Amazon EKS Linux nodes

1. Apply the Calico manifests to your cluster by completing the option that corresponds to the Region that your cluster is in.

   - All regions other than China (Ningxia) or China (Beijing) – Apply the Calico manifests from the `aws/amazon-vpc-cni-k8s` GitHub project. These manifests create DaemonSets 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
     ```

   - China (Ningxia) or China (Beijing)

     1. Download the Calico manifests with the following commands.

        ```
        curl -o calico-operator.yaml https://raw.githubusercontent.com/aws/amazon-vpc-cni-
        k8s/master/config/master/calico-operator.yaml
        curl -o calico-crs.yaml https://raw.githubusercontent.com/aws/amazon-vpc-cni-k8s/
        master/config/master/calico-crs.yaml
        ```

     2. Modify the manifests.

        a. View the manifest file or files that you downloaded and note the name of the images. Download the images locally with the following commands.

```
docker pull quay.io/tigera/operator:v1.13.2
docker pull quay.io/calico/node:v3.17.1
docker pull quay.io/calico/typha:v3.17.1
```

b. Tag the images to be pushed to an Amazon Elastic Container Registry repository in China with the following command.

```
docker tag image:<tag> <aws_account_id>.dkr.ecr.<cn-north-1>.amazonaws.com/
image:<tag>
```

c. Push the images to a China Amazon ECR repository with the following command.

```
docker push image:<tag> <aws_account_id>.dkr.ecr.<cn-north-1>.amazonaws.com/
image:<tag>
```

d. Update the `calico-operator.yaml` file to reference the Amazon ECR image URL in your Region.

e. Update the `calico-crs.yaml` file to reference the Amazon ECR image repository in your Region by adding the following to the spec.

```
registry: <aws_account_id>.dkr.ecr.<cn-north-1>.amazonaws.com.cn
```

3. Apply the Calico manifests. These manifests create DaemonSets in the `calico-system` namespace.

```
kubectl apply -f calico-operator.yaml
kubectl apply -f calico-crs.yaml
```

2. Watch the `calico-system` DaemonSets and wait for the `calico-node` DaemonSet to have the `DESIRED` number of pods in the `READY` state. When this happens, Calico is working.

```
kubectl get daemonset calico-node --namespace calico-system
```

Output:

```
NAME           DESIRED    CURRENT    READY     UP-TO-DATE    AVAILABLE    NODE SELECTOR
 AGE
calico-node    3          3          3         3             3            <none>
 38s
```

To delete Calico from your Amazon EKS cluster

- If you are done using Calico in your Amazon EKS cluster, you can delete it with the following commands:

```
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
```

Stars policy demo

This section walks through the Stars policy demo provided by the Project Calico documentation. The demo creates a frontend, backend, and client service on your Amazon EKS cluster. The demo also creates a management GUI that shows the available ingress and egress paths between each service.

Before you create any network policies, all services can communicate bidirectionally. After you apply the network policies, you can see that the client can only communicate with the frontend service, and the backend only accepts traffic from the frontend.

To run the Stars policy demo

1. Apply the frontend, backend, client, and management UI services:

```
kubectl apply -f https://docs.projectcalico.org/v3.3/getting-started/kubernetes/
tutorials/stars-policy/manifests/00-namespace.yaml
kubectl apply -f https://docs.projectcalico.org/v3.3/getting-started/kubernetes/
tutorials/stars-policy/manifests/01-management-ui.yaml
kubectl apply -f https://docs.projectcalico.org/v3.3/getting-started/kubernetes/
tutorials/stars-policy/manifests/02-backend.yaml
kubectl apply -f https://docs.projectcalico.org/v3.3/getting-started/kubernetes/
tutorials/stars-policy/manifests/03-frontend.yaml
kubectl apply -f https://docs.projectcalico.org/v3.3/getting-started/kubernetes/
tutorials/stars-policy/manifests/04-client.yaml
```

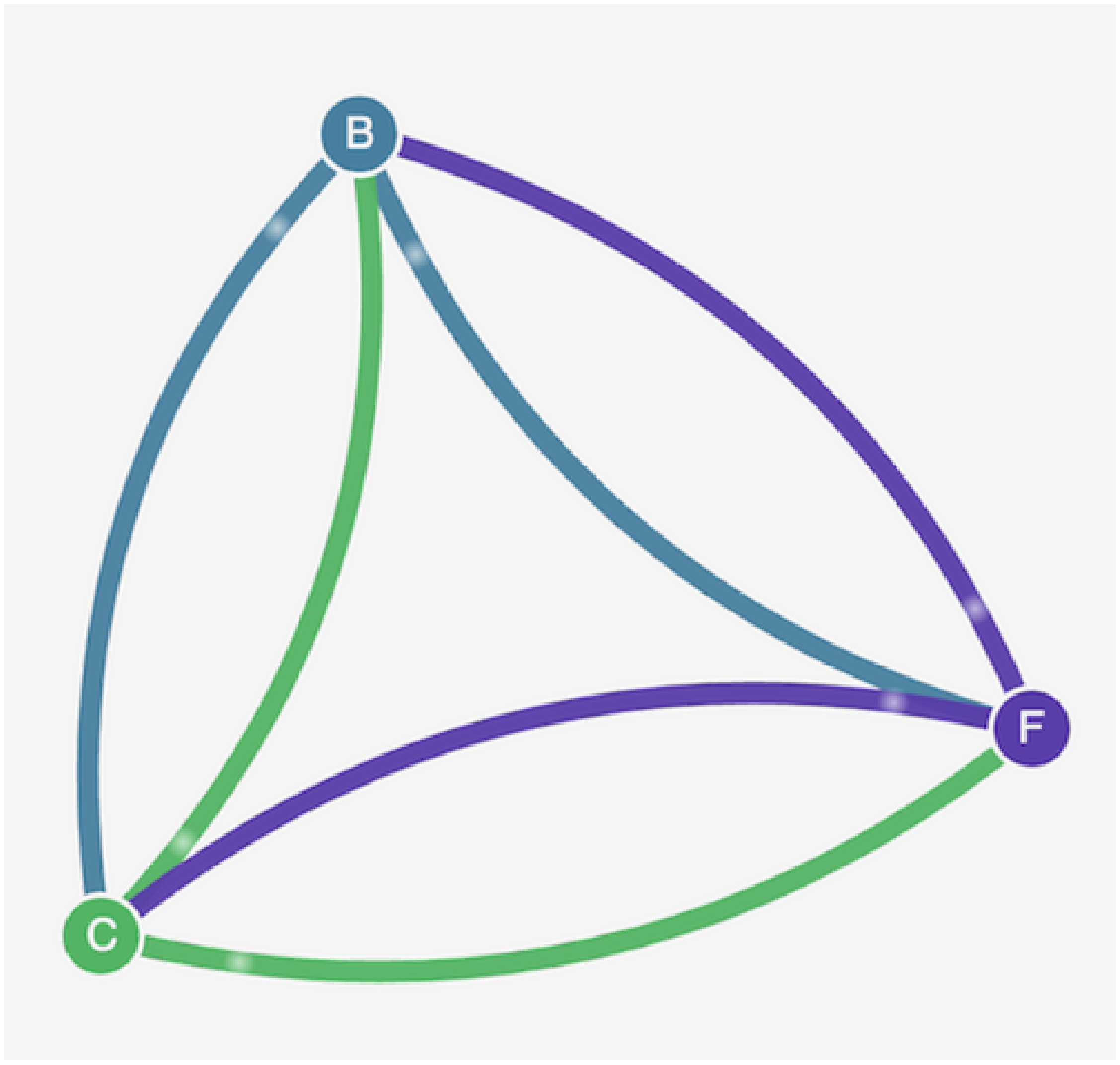
2. Wait for all of the pods to reach the `Running` status:

```
kubectl get pods --all-namespaces --watch
```

3. To connect to the management UI, forward your local port 9001 to the `management-ui` service running on your cluster:

```
kubectl port-forward service/management-ui -n management-ui 9001
```

4. Open a browser on your local system and point it to http://localhost:9001/. You should see the management UI. The **C** node is the client service, the **F** node is the frontend service, and the **B** node is the backend service. Each node has full communication access to all other nodes (as indicated by the bold, colored lines).



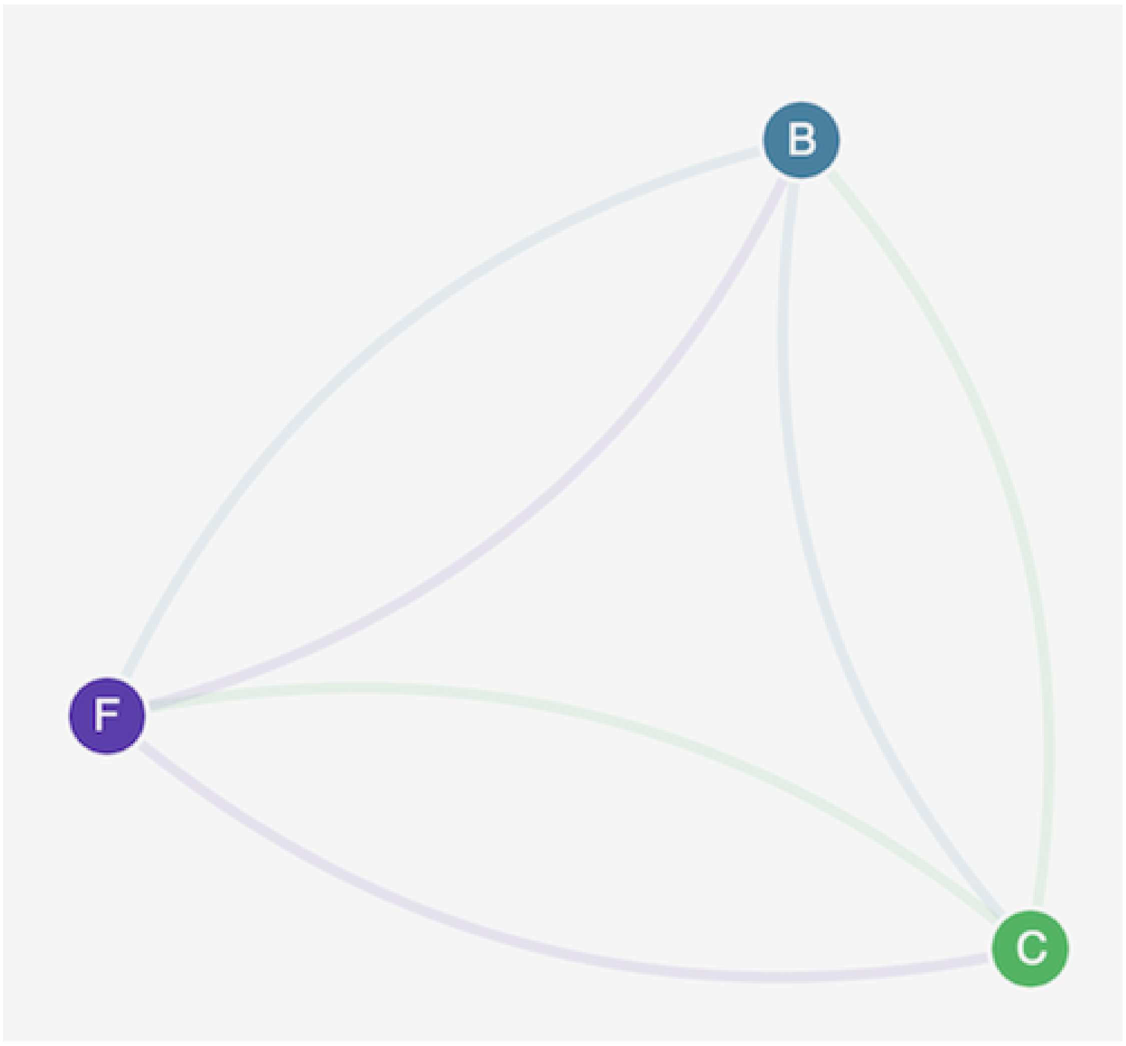
5. Apply the following network policies to isolate the services from each other:

```
kubectl apply -n stars -f https://docs.projectcalico.org/v3.3/getting-started/
kubernetes/tutorials/stars-policy/policies/default-deny.yaml
kubectl apply -n client -f https://docs.projectcalico.org/v3.3/getting-started/
kubernetes/tutorials/stars-policy/policies/default-deny.yaml
```

6. Refresh your browser. You see that the management UI can no longer reach any of the nodes, so they don't show up in the UI.
7. Apply the following network policies to allow the management UI to access the services:

```
kubectl apply -f https://docs.projectcalico.org/v3.3/getting-started/kubernetes/
tutorials/stars-policy/policies/allow-ui.yaml
kubectl apply -f https://docs.projectcalico.org/v3.3/getting-started/kubernetes/
tutorials/stars-policy/policies/allow-ui-client.yaml
```

8. Refresh your browser. You see that the management UI can reach the nodes again, but the nodes cannot communicate with each other.

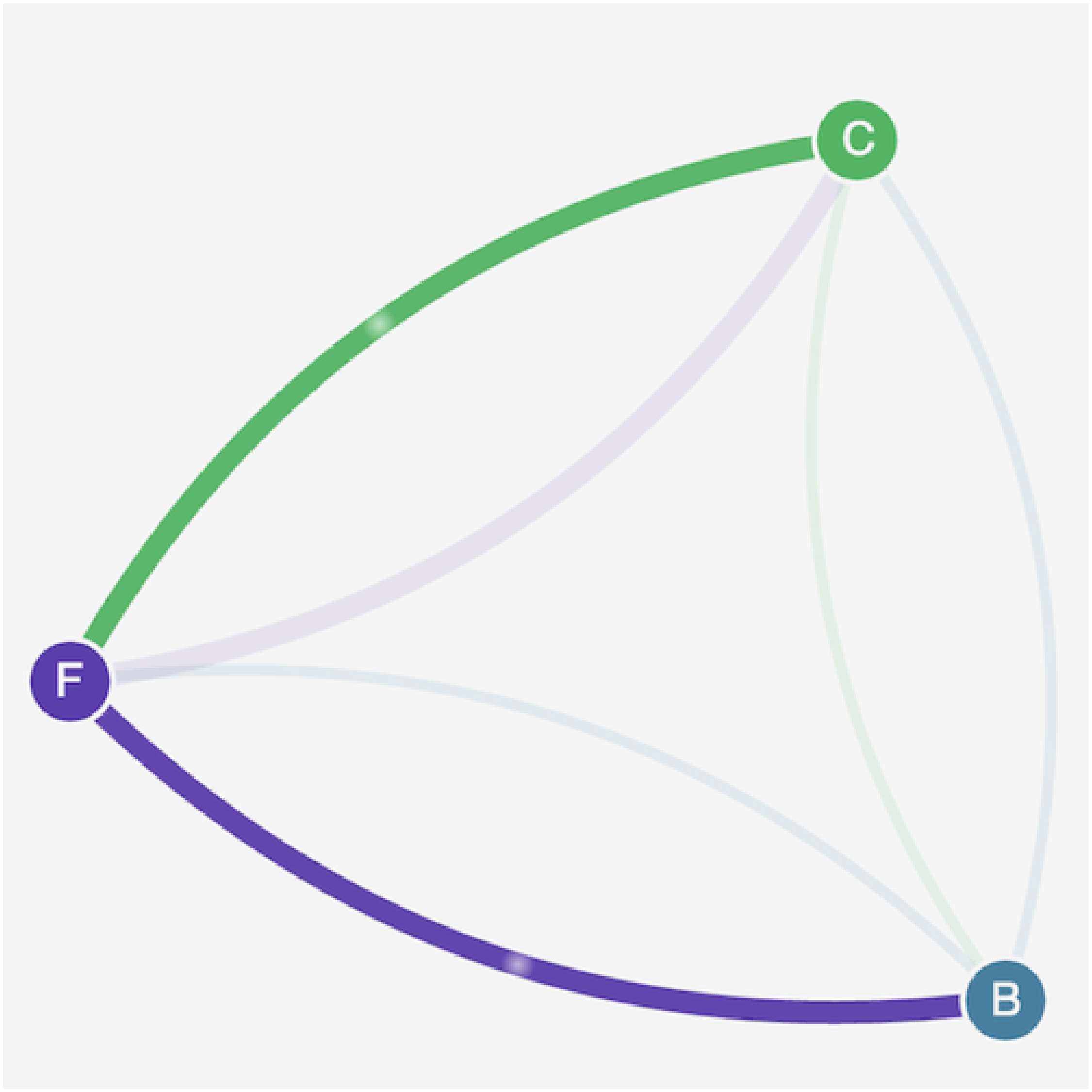


9. Apply the following network policy to allow traffic from the frontend service to the backend service:

```
kubectl apply -f https://docs.projectcalico.org/v3.3/getting-started/kubernetes/
tutorials/stars-policy/policies/backend-policy.yaml
```

10. Apply the following network policy to allow traffic from the `client` namespace to the frontend service:

```
kubectl apply -f https://docs.projectcalico.org/v3.3/getting-started/kubernetes/
tutorials/stars-policy/policies/frontend-policy.yaml
```



11. (Optional) When you are done with the demo, you can delete its resources with the following commands:

```
kubectl delete -f https://docs.projectcalico.org/v3.3/getting-started/kubernetes/
tutorials/stars-policy/manifests/04-client.yaml
kubectl delete -f https://docs.projectcalico.org/v3.3/getting-started/kubernetes/
tutorials/stars-policy/manifests/03-frontend.yaml
kubectl delete -f https://docs.projectcalico.org/v3.3/getting-started/kubernetes/
tutorials/stars-policy/manifests/02-backend.yaml
kubectl delete -f https://docs.projectcalico.org/v3.3/getting-started/kubernetes/
tutorials/stars-policy/manifests/01-management-ui.yaml
kubectl delete -f https://docs.projectcalico.org/v3.3/getting-started/kubernetes/
tutorials/stars-policy/manifests/00-namespace.yaml
```

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.

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. 253) running on each of your clusters using the AWS Management Console.
- Vertically scale pods up or down with the Kubernetes Vertical Pod Autoscaler (p. 259).
- Horizontally scale the number of pods needed to meet demand up or down with the Kubernetes Horizontal Pod Autoscaler (p. 263).
- Create an external (for internet-accessible pods) or an internal (for private pods) load balancer (p. 266) 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. 271) 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 Linux workload (p. 255).
- You can restrict IP addresses that can be assigned to a service (p. 274) with `externalIPs`.

View workloads

Workloads define applications running on a Kubernetes cluster. Every workload controls pods. Pods are the fundamental unit of computing within a Kubernetes cluster and represent one or more containers that run together.

You can use the Amazon EKS console to view information about the Kubernetes workloads and pods running on your cluster.

Prerequisites

The IAM user or IAM role that you sign into the AWS Management Console with must meet the following requirements.

- Has the `eks:AccessKubernetesApi` and other necessary IAM permissions to view workloads attached to it. For an example IAM policy, see the section called “View nodes and workloads for all clusters in the AWS Management Console” (p. 333) .
- Is mapped to Kubernetes user or group in the `aws-auth configmap`. For more information, see the section called “Managing users or IAM roles for your cluster” (p. 288).
- The Kubernetes user or group that the IAM user or role is mapped to in the configmap must be bound to a Kubernetes `role` or `clusterrole` that has permissions to view the resources in the namespaces that you want to view. For more information, see Using RBAC Authorization in the Kubernetes documentation. You can download the following example manifests that create a `clusterrole` and `clusterrolebinding` or a `role` and `rolebinding`:
  - **View Kubernetes resources in all namespaces** – The group name in the file is `eks-console-dashboard-full-access-group`, which is the group that your IAM user or role needs to be mapped to in the `aws-auth` configmap. You can change the name of the group before applying it to your cluster, if desired, and then map your IAM user or role to that group in the configmap. To download the file, select the appropriate link for the Region that your cluster is in.

- All Regions other than Beijing and Ningxia China
- Beijing and Ningxia China Regions

- **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. The group name in the file is `eks-console-dashboard-restricted-access-group`, which is the group that your IAM user or role needs to be mapped to in the `aws-auth` configmap. You can change the name of the group before applying it to your cluster, if desired, and then map your IAM user or role to that group in the configmap. To download the file, select the appropriate link for the Region that your cluster is in.
  - All Regions other than Beijing and Ningxia China
  - Beijing and Ningxia China Regions

To view workloads using 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 you want to view workloads for.
3. On the **Workloads** tab, you see a list of **Names** of all the Kubernetes workloads that are currently deployed to your cluster, the **Pod count**, and **Status** for each workload.

   Important
   If you can't see any workloads, or you see a **Your current user or role does not have access to Kubernetes objects on this EKS cluster**, you may need to select a different namespace from the **All Namespaces** drop-down list. If you're still having problems, see the prerequisites for this topic. If you don't resolve the issue, you can still view and manage your Amazon EKS cluster on the **Configuration** tab, but you won't be able to see information about your workloads.

   You can deploy the following types of workloads on a cluster.

   - **Deployment** – Ensures that a specific number of pods run and includes logic to deploy changes.
   - **ReplicaSet** – Ensures that a specific number of pods run. Can be controlled by deployments.
   - **StatefulSet** – Manages the deployment of stateful applications.
   - **DaemonSet** – Ensures that a copy of a pod runs on all (or some) nodes in the cluster.
   - **Job** – Creates one or more pods and ensures that a specified number of them run to completion.

   For more information, see Workloads in the Kubernetes documentation.

   By default, all Amazon EKS clusters have the following workloads:

   - **coredns** –A `Deployment` that deploys two replicas of the `coredns` pods. The pods provide name resolution for all pods in the cluster. Two pods are deployed by default for high availability, regardless of the number of nodes deployed in your cluster. For more information, see the section called “Installing or upgrading CoreDNS” (p. 243). The pods can be deployed to any node type. However, they can be deployed to Fargate nodes only if your cluster includes a Fargate profile with a namespace that matches the namespace for the workload.
   - **aws-node** – A `DaemonSet` that deploys one pod to each Amazon EC2 node in your cluster. The pod runs the Amazon Virtual Private Cloud (Amazon VPC) CNI controller, which provides VPC networking functionality to the pods and nodes in your cluster. For more information, see the section called “Pod networking (CNI)” (p. 214). This workload isn't deployed to Fargate nodes because Fargate already contains the Amazon VPC CNI controller.
   - **kube-proxy** – A `DaemonSet` that deploys one pod to each Amazon EC2 node in your cluster. The pods maintain network rules on nodes that enable networking communication to your pods. For more information, see `kube-proxy` in the Kubernetes documentation. This workload isn't deployed to Fargate nodes.

View workload details

Selecting the link in the **Name** column for one of the **Workloads** shows you the following information:

- The **Status**, **Namespace**, and **Selectors** (if any) assigned to the workload.
- The list of **Pods** managed by the workload, their **Status** and **Created** date and time.
- The Kubernetes **Labels** and **Annotations** assigned to the workload. These could have been assigned by you, by Kubernetes, or by the Amazon EKS API when the workload was created.

Some of the detailed information that you see when inspecting a workload might be different depending on the type of workload. To learn more about the unique properties of each workload type, see Workloads in the Kubernetes documentation.

View pod details

Pods are the fundamental unit of computing within a Kubernetes cluster and represent one or more containers that run together. Each workload controls one of more pods running on the cluster. Pods are designed as relatively ephemeral and immutable objects. Over time it's normal to have pods start and stop while your cluster is running. Pods starting or stopping reflect changes in your workloads as well as changes in the scale of your cluster. For more information, see Pods in the Kubernetes documentation.

When viewing a workload, select a link in the **Name** column for one of the **Pods** to see the following information about the pod:

- The Kubernetes **Namespace** that the pod is in and the pod's **Status**.
- The node on which the pod is running. The node might be an Amazon EC2 instance or a Fargate pod. For more information about viewing nodes, see the section called “View nodes” (p. 86).
- The **Containers** in the pod. Selecting the name of a container provides the **Image**, **Ports**, **Mounts**, and **Arguments** provided when the pod was started. Pods can define two types of containers. The first is *application* containers, which are containers that run as long as the pod is running. The second is *init* containers, which are containers that serve as processes that run during pod startup. You can see both types of containers on the pod detail page.
- The Kubernetes **Labels** and **Annotations** assigned to the pod.

Deploy a sample Linux workload

In this topic, you create a Kubernetes manifest and deploy it to your cluster.

Prerequisites

- You must have an existing Kubernetes cluster to deploy a sample application. If you don't have an existing cluster, you can deploy an Amazon EKS cluster using one of the Getting started with Amazon EKS (p. 4) guides.
- You must have `kubectl` installed on your computer. For more information, see Installing `kubectl` (p. 303).
- `kubectl` must be configured to communicate with your cluster. For more information, see Create a `kubeconfig` for Amazon EKS (p. 295).

To deploy a sample application

1. Create a Kubernetes namespace for the sample app.

   ```
   kubectl create namespace <my-namespace>
   ```

2. Create a Kubernetes service and deployment.

   a. Save the following contents to a file named `sample-service.yaml` on your computer. If you're deploying to AWS Fargate (p. 124) pods, then make sure that the value for `namespace` matches the namespace that you defined in your AWS Fargate profile (p. 129). This sample deployment will pull a container image from a public repository, deploy three replicas of it to your cluster, and create a Kubernetes service with its own IP address that can be accessed from within the cluster only. To access the service from outside the cluster, you need to deploy a load balancer (p. 266) or ALB Ingress Controller (p. 271).

      The image is a multi-architecture image, so if your cluster includes both x86 and Arm nodes, then the pod can be scheduled on either type of hardware architecture. Kubernetes will deploy the appropriate hardware image based on the hardware type of the node it schedules the pod on. Alternatively, if you only want the deployment to run on nodes with a specific hardware architecture, or your cluster only contains one hardware architecture, then remove either `amd64` or `arm64` from the example that follows.

      ```
      apiVersion: v1
      kind: Service
      metadata:
        name: my-service
        namespace: my-namespace
        labels:
          app: my-app
      spec:
        selector:
          app: my-app
        ports:
          - protocol: TCP
            port: 80
            targetPort: 80
      ---
      apiVersion: apps/v1
      kind: Deployment
      metadata:
        name: my-deployment
        namespace: my-namespace
        labels:
          app: my-app
      spec:
        replicas: 3
        selector:
          matchLabels:
            app: my-app
        template:
          metadata:
            labels:
              app: my-app
          spec:
            affinity:
              nodeAffinity:
                requiredDuringSchedulingIgnoredDuringExecution:
                  nodeSelectorTerms:
                  - matchExpressions:
                    - key: beta.kubernetes.io/arch
                      operator: In
      ```

```
                values:
                - amd64
                - arm64
      containers:
      - name: nginx
        image: public.ecr.aws/z9d2n7e1/nginx:1.19.5
        ports:
        - containerPort: 80
```

To learn more about Kubernetes services and deployments, see the Kubernetes documentation. The containers in the sample manifest do not use network storage, but they may be able to. For more information, see Storage (p. 180). Though not implemented in this example, we recommend that you create Kubernetes service accounts for your pods, and associate them to AWS IAM accounts. Specifying service accounts enables your pods to have the minimum permissions that they require to interact with other services. For more information, see IAM roles for service accounts (p. 345)

b. Deploy the application.

```
kubectl apply -f <sample-service.yaml>
```

3. View all resources that exist in the `my-namespace` namespace.

```
kubectl get all -n my-namespace
```

Output

```
NAME                                    READY   STATUS    RESTARTS   AGE
pod/my-deployment-776d8f8fd8-78w66      1/1     Running   0          27m
pod/my-deployment-776d8f8fd8-dkjfr      1/1     Running   0          27m
pod/my-deployment-776d8f8fd8-wmqj6      1/1     Running   0          27m

NAME                   TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)   AGE
service/my-service     ClusterIP   10.100.190.12   <none>        80/TCP    32m

NAME                            READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/my-deployment   3/3     3            3           27m

NAME                                       DESIRED   CURRENT   READY   AGE
replicaset.apps/my-deployment-776d8f8fd8   3         3         3       27m
```

In the output, you see the service and deployment that are specified in the sample manifest deployed in the previous step. You also see three pods, which are due to specifying `3` for `replicas` in the sample manifest. For more information about pods, see Pods in the Kubernetes documentation. Kubernetes automatically created the `replicaset` resource, even though it wasn't specified in the sample manifest. For more information about ReplicaSets, see ReplicaSet in the Kubernetes documentation.

Note
Kubernetes will maintain the number of replicas 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, you'd need to use the Horizontal Pod Autoscaler (p. 263) and the Vertical Pod Autoscaler (p. 259).

4. View the details of the deployed service.

```
kubectl -n <my-namespace> describe service <my-service>
```

Abbreviated output

```
Name:                 my-service
Namespace:            my-namespace
Labels:               app=my-app
Annotations:          kubectl.kubernetes.io/last-applied-configuration:
                        {"apiVersion":"v1","kind":"Service","metadata":{"annotations":
{},"labels":{"app":"my-app"},"name":"my-service","namespace":"my-namespace"}...
Selector:             app=my-app
Type:                 ClusterIP
IP:                   10.100.190.12
Port:                 <unset>  80/TCP
TargetPort:           80/TCP
...
```

In the output, the value for `IP:` is a unique IP address that can be reached from any pod within the cluster.

5. View the details of one of the pods that was deployed.

```
kubectl -n <my-namespace> describe pod <my-deployment-776d8f8fd8-78w66>
```

Abbreviated output

```
Name:         my-deployment-776d8f8fd8-78w66
Namespace:    my-namespace
Priority:     0
Node:         ip-192-168-9-36.us-west-2.compute.internal/192.168.9.36
...
IP:           192.168.16.57
IPs:
  IP:           192.168.16.57
Controlled By:  ReplicaSet/my-deployment-776d8f8fd8
...
Conditions:
  Type              Status
  Initialized       True
  Ready             True
  ContainersReady   True
  PodScheduled      True
...
Events:
  Type    Reason     Age    From
 Message
  ----    ------     ----   ----
 -------
  Normal  Scheduled  3m20s  default-scheduler
       Successfully assigned my-namespace/my-deployment-776d8f8fd8-78w66 to
 ip-192-168-9-36.us-west-2.compute.internal
...
```

In the output, the value for `IP:` is a unique IP that is assigned to the pod from the CIDR block assigned to the subnet that the node is in, by default. If you'd prefer that pods be assigned IP addresses from different CIDR blocks than the subnet that the node is in, you can change the default behavior. For more information, see CNI custom networking (p. 230). You can also see that the Kubernetes scheduler scheduled the pod on the node with the IP address `192.168.9.36`.

6. Execute a shell on one of the pods by replacing the <value> below with a value returned for one of your pods in step 3.

```
kubectl exec -it <my-deployment-776d8f8fd8-78w66> -n <my-namespace> -- /bin/bash
```

7. View the DNS resolver configuration file.

```
cat /etc/resolv.conf
```

   Output

```
nameserver 10.100.0.10
search my-namespace.svc.cluster.local svc.cluster.local cluster.local us-
west-2.compute.internal
options ndots:5
```

   In the previous output, the value for `nameserver` is the cluster's nameserver and is automatically assigned as the name server for any pod deployed to the cluster.
8. Disconnect from the pod by typing `exit`.
9. Remove the sample service, deployment, pods, and namespace.

```
kubectl delete namespace <my-namespace>
```

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 the section called “Metrics server” (p. 314).
- You are using a `kubectl` client that is configured to communicate with your Amazon EKS cluster (p. 11).

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 cluster isn't in the China (Beijing) or China (Ningxia) Region, skip to step 6. Edit the manifest files in `./vertical-pod-autoscaler/deploy` using the following steps.

   1. 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>
      ```

   2. Tag the image to be pushed to an Amazon Elastic Container Registry repository in China with the following command.

      ```
      docker tag image:<tag> <aws_account_id>.dkr.ecr.<cn-north-1>.amazonaws.com/
      image:<tag>
      ```

   3. Push the image to a China Amazon ECR repository with the following command.

      ```
      docker push image:<tag> <aws_account_id>.dkr.ecr.<cn-north-1>.amazonaws.com/
      image:<tag>
      ```

   4. Update the Kubernetes manifest file or files to reference the Amazon ECR image URL in your Region.

6. Deploy the Vertical Pod Autoscaler to your cluster with the following command.

   ```
   ./hack/vpa-up.sh
   ```

7. Verify that the Vertical Pod Autoscaler pods have been created successfully.

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

   Output:

   ```
   NAME                                        READY   STATUS    RESTARTS   AGE
   aws-node-949vx                              1/1     Running   0          122m
   aws-node-b4nj8                              1/1     Running   0          122m
   coredns-6c75b69b98-r9x68                    1/1     Running   0          133m
   coredns-6c75b69b98-rt9bp                    1/1     Running   0          133m
   kube-proxy-bkm6b                            1/1     Running   0          122m
   kube-proxy-hpqm2                            1/1     Running   0          122m
   metrics-server-8459fc497-kfj8w              1/1     Running   0          83m
   vpa-admission-controller-68c748777d-ppspd   1/1     Running   0          7s
   vpa-recommender-6fc8c67d85-gljpl            1/1     Running   0          8s
   vpa-updater-786b96955c-bgp9d                1/1     Running   0          8s
   ```

Test your Vertical Pod Autoscaler installation

In this section, you deploy a sample application to verify that the Vertical Pod Autoscaler is working.

To test your Vertical Pod Autoscaler installation

1. Deploy the `hamster.yaml` Vertical Pod Autoscaler example with the following command.

   ```
   kubectl apply -f examples/hamster.yaml
   ```

2. Get the pods from the `hamster` example application.

   ```
   kubectl get pods -l app=hamster
   ```

   Output:

   ```
   hamster-c7d89d6db-rglf5   1/1     Running   0          48s
   hamster-c7d89d6db-znvz5   1/1     Running   0          48s
   ```

3. Describe one of the pods to view its CPU and memory reservation.

   ```
   kubectl describe pod hamster-<c7d89d6db-rglf5>
   ```

   Output:

   ```
   Name:           hamster-c7d89d6db-rglf5
   Namespace:      default
   Priority:       0
   Node:           ip-192-168-9-44.<region-code>.compute.internal/192.168.9.44
   Start Time:     Fri, 27 Sep 2019 10:35:15 -0700
   Labels:         app=hamster
                   pod-template-hash=c7d89d6db
   Annotations:    kubernetes.io/psp: eks.privileged
                   vpaUpdates: Pod resources updated by hamster-vpa: container 0:
   Status:         Running
   IP:             192.168.23.42
   IPs:            <none>
   Controlled By:  ReplicaSet/hamster-c7d89d6db
   Containers:
     hamster:
       Container ID:  docker://
   e76c2413fc720ac395c33b64588c82094fc8e5d590e373d5f818f3978f577e24
       Image:         k8s.gcr.io/ubuntu-slim:0.1
       Image ID:      docker-pullable://k8s.gcr.io/ubuntu-
   slim@sha256:b6f8c3885f5880a4f1a7cf717c07242eb4858fdd5a84b5ffe35b1cf680ea17b1
       Port:          <none>
       Host Port:     <none>
       Command:
         /bin/sh
       Args:
         -c
         while true; do timeout 0.5s yes >/dev/null; sleep 0.5s; done
       State:          Running
         Started:      Fri, 27 Sep 2019 10:35:16 -0700
       Ready:          True
       Restart Count:  0
       Requests:
         cpu:        100m
         memory:     50Mi
   ...
   ```

   You can see that the original pod reserves 100 millicpu of CPU and 50 mebibytes of memory. For this example application, 100 millicpu is less than the pod needs to run, so it is CPU-constrained. It also reserves much less memory than it needs. The Vertical Pod Autoscaler `vpa-recommender` deployment analyzes the `hamster` pods to see if the CPU and memory requirements are appropriate. If adjustments are needed, the `vpa-updater` relaunches the pods with updated values.

4. Wait for the `vpa-updater` to launch a new `hamster` pod. This should take a minute or two. You can monitor the pods with the following command.

Note
If you are not sure that a new pod has launched, compare the pod names with your previous list. When the new pod launches, you will see a new pod name.

```
kubectl get --watch pods -l app=hamster
```

5. When a new `hamster` pod is started, describe it and view the updated CPU and memory reservations.

```
kubectl describe pod hamster-<c7d89d6db-jxgfv>
```

Output:

```
Name:           hamster-c7d89d6db-jxgfv
Namespace:      default
Priority:       0
Node:           ip-192-168-9-44.<region-code>.compute.internal/192.168.9.44
Start Time:     Fri, 27 Sep 2019 10:37:08 -0700
Labels:         app=hamster
                pod-template-hash=c7d89d6db
Annotations:    kubernetes.io/psp: eks.privileged
                vpaUpdates: Pod resources updated by hamster-vpa: container 0: cpu
 request, memory request
Status:         Running
IP:             192.168.3.140
IPs:            <none>
Controlled By:  ReplicaSet/hamster-c7d89d6db
Containers:
  hamster:
    Container ID:
 docker://2c3e7b6fb7ce0d8c86444334df654af6fb3fc88aad4c5d710eac3b1e7c58f7db
    Image:         k8s.gcr.io/ubuntu-slim:0.1
    Image ID:      docker-pullable://k8s.gcr.io/ubuntu-
slim@sha256:b6f8c3885f5880a4f1a7cf717c07242eb4858fdd5a84b5ffe35b1cf680ea17b1
    Port:          <none>
    Host Port:     <none>
    Command:
      /bin/sh
    Args:
      -c
      while true; do timeout 0.5s yes >/dev/null; sleep 0.5s; done
    State:          Running
      Started:      Fri, 27 Sep 2019 10:37:08 -0700
    Ready:          True
    Restart Count:  0
    Requests:
      cpu:        587m
      memory:     262144k
...
```

Here you can see that the CPU reservation has increased to 587 millicpu, which is over five times the original value. The memory has increased to 262,144 Kilobytes, which is around 250 mebibytes, or five times the original value. This pod was under-resourced, and the Vertical Pod Autoscaler corrected our estimate with a much more appropriate value.

6. Describe the `hamster-vpa` resource to view the new recommendation.

```
kubectl describe vpa/hamster-vpa
```

Output:

```
Name:         hamster-vpa
Namespace:    default
Labels:       <none>
Annotations:  kubectl.kubernetes.io/last-applied-configuration:
                {"apiVersion":"autoscaling.k8s.io/
v1beta2","kind":"VerticalPodAutoscaler","metadata":{"annotations":{},"name":"hamster-
vpa","namespace":"d...
API Version:  autoscaling.k8s.io/v1beta2
Kind:         VerticalPodAutoscaler
Metadata:
  Creation Timestamp:  2019-09-27T18:22:51Z
  Generation:          23
  Resource Version:    14411
  Self Link:           /apis/autoscaling.k8s.io/v1beta2/namespaces/default/
verticalpodautoscalers/hamster-vpa
  UID:                 d0d85fb9-e153-11e9-ae53-0205785d75b0
Spec:
  Target Ref:
    API Version:  apps/v1
    Kind:         Deployment
    Name:         hamster
Status:
  Conditions:
    Last Transition Time:  2019-09-27T18:23:28Z
    Status:                True
    Type:                  RecommendationProvided
  Recommendation:
    Container Recommendations:
      Container Name:  hamster
      Lower Bound:
        Cpu:     550m
        Memory:  262144k
      Target:
        Cpu:     587m
        Memory:  262144k
      Uncapped Target:
        Cpu:     587m
        Memory:  262144k
      Upper Bound:
        Cpu:     21147m
        Memory:  387863636
Events:          <none>
```

7. When you finish experimenting with the example application, you can delete it with the following command.

```
kubectl delete -f examples/hamster.yaml
```

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 the section called "Metrics server" (p. 314).
- You are using a `kubectl` client that is configured to communicate with your Amazon EKS cluster (p. 11).

Run a Horizontal Pod Autoscaler test application

In this section, you deploy a sample application to verify that the Horizontal Pod Autoscaler is working.

Note
This example is based on the Horizontal pod autoscaler walkthrough in the Kubernetes documentation.

To test your Horizontal Pod Autoscaler installation

1. 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 describe hpa
```

   Output:

```
Name:                                                  php-apache
Namespace:                                             default
Labels:                                                <none>
Annotations:                                           <none>
CreationTimestamp:                                     Thu, 11 Jun 2020 16:05:41 -0500
```

```
Reference:                                             Deployment/php-apache
Metrics:                                               ( current / target )
  resource cpu on pods  (as a percentage of request):  <unknown> / 50%
Min replicas:                                          1
Max replicas:                                          10
Deployment pods:                                       1 current / 0 desired
Conditions:
  Type           Status  Reason                   Message
  ----           ------  ------                   -------
  AbleToScale    True    SucceededGetScale        the HPA controller was able to get
 the target's current scale
  ScalingActive  False   FailedGetResourceMetric  the HPA was unable to compute the
 replica count: did not receive metrics for any ready pods
Events:
  Type     Reason                        Age                From
 Message
  ----     ------                        ----               ----
 -------
  Warning  FailedGetResourceMetric       42s (x2 over 57s)  horizontal-pod-autoscaler
 unable to get metrics for resource cpu: no metrics returned from resource metrics API
  Warning  FailedComputeMetricsReplicas  42s (x2 over 57s)  horizontal-pod-autoscaler
  invalid metrics (1 invalid out of 1), first error is: failed to get cpu utilization:
 unable to get metrics for resource cpu: no metrics returned from resource metrics API
  Warning  FailedGetResourceMetric       12s (x2 over 27s)  horizontal-pod-autoscaler
 did not receive metrics for any ready pods
  Warning  FailedComputeMetricsReplicas  12s (x2 over 27s)  horizontal-pod-autoscaler
 invalid metrics (1 invalid out of 1), first error is: failed to get cpu utilization:
 did not receive metrics for any ready pods
```

As you can see, the current CPU load is `<unknown>`, 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 -it --rm load-generator --image=busybox /bin/sh --generator=run-pod/v1
```

If you don't receive a command prompt after several seconds, you may need to press `Enter`. From the command prompt, enter the following command to generate load and cause the autoscaler to scale out the deployment.

```
while true; 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
```

Output:

```
NAME         REFERENCE                TARGETS    MINPODS   MAXPODS   REPLICAS   AGE
php-apache   Deployment/php-apache    250%/50%   1         10        5          4m44s
```

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 (from step 4), 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.

   ```
   kubectl get hpa
   ```

   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.
7. 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

When you create a Kubernetes `Service` of type `LoadBalancer`, an AWS Network Load Balancer (NLB) or Classic Load Balancer (CLB) is provisioned that load balances network traffic. To learn more about the differences between the two types of load balancers, see Elastic Load Balancing features on the AWS website. For more information about a Kubernetes Service, see Service in the Kubernetes documentation. NLBs can be used with pods deployed to nodes or to AWS Fargate IP targets. You can deploy an AWS load balancer to a public or private subnet.

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 the section called "Application load balancing" (p. 271). To learn more about the differences between the two types of load balancing, see Elastic Load Balancing features on the AWS website.

In Amazon EKS, you can load balance network traffic to an NLB (*instance* or *IP* target) or a CLB (*instance* target only). For more information about NLB target types, see Target type in the User Guide for Network Load Balancers. When you load balance network traffic to instance targets, the Kubernetes in-tree controller creates the NLB or CLB. To load balance network traffic to IP target types, the AWS Load Balancer Controller creates an NLB. For more information, see AWS load balancer controller on GitHub.

Prerequisites

Before you can load balance network 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're load balancing to IP targets, the cluster must be 1.18 or later. To update an existing cluster, see the section called "Updating a cluster" (p. 24).
- If you're load balancing to IP targets, you must have the AWS Load Balancer Controller provisioned on your cluster. For more information, see the section called "AWS Load Balancer Controller" (p. 238).
- At least one subnet. In the case of multiple tagged subnets found in an Availability Zone, the controller chooses the first subnet in lexicographical order by the subnet IDs.

- 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 per 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 is not 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>`* (including *<>*) with your cluster name.
  - **Key** – `kubernetes.io/cluster/`*`<cluster-name>`*
  - **Value** – `shared` or *`owned`*
- **Subnet tagging** – 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 are not required.
  - **Private subnets** – Must be tagged as follows 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 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 AWS CloudFormation VPC templates, see the section called "Creating a VPC for Amazon EKS" (p. 204).
    - **Key** – `kubernetes.io/role/internal-elb`
    - **Value** – `1`
  - **Public subnets** – Must be tagged as follows so that Kubernetes knows to use only those subnets for external load balancers instead of choosing a public subnet in each Availability Zone (in lexicographical order by subnet ID). If you use `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. 204).
    - **Key** – `kubernetes.io/role/elb`
    - **Value** – `1`

  If the subnet role tags are not 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 do not rely on this behavior, and instead explicitly add the private or public role tags. The AWS load balancer controller does not examine route tables, and requires the private and public tags to be present for successful auto discovery.

Considerations

- Use of the UDP protocol is supported with the load balancer on Amazon EKS clusters with the following platform versions. For more information, see the section called "Platform versions" (p. 65).

Amazon EKS version	Platform version
1.19	.1
1.18	.1
1.17	.2
1.16	.3
1.15	.4

- You can only use NLB *IP* targets with the Amazon EKS VPC CNI plugin (p. 214). You can use NLB *instance* targets with the Amazon EKS VPC CNI plugin or alternate compatible CNI plugins (p. 238).
- You can only use *IP* targets with NLB. You can't use IP targets with CLBs.
- The configuration of your load balancer is controlled by annotations that are added to the manifest for your service. 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 Other ELB annotations in the Kubernetes documentation.

```
service.beta.kubernetes.io/aws-load-balancer-additional-resource-tags
```

- If you're using Amazon EKS 1.16 or later, you can assign Elastic IP addresses to the Network Load Balancer by adding the following annotation. Replace the *<example-values>* (including *<>*) with the Allocation IDs of your Elastic IP addresses. The number of Allocation IDs must match the number of subnets used for the load balancer.

```
service.beta.kubernetes.io/aws-load-balancer-eip-allocations:
 eipalloc-<xxxxxxxxxxxxxxxxx>,eipalloc-<yyyyyyyyyyyyyyyyy>
```

- For each NLB that you create on 1.16 or later clusters, 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. On 1.15 clusters, one rule is added for each CIDR block assigned to the VPC. 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 the section called "Load balancer – IP targets" (p. 269), rather than instance targets. With IP targets, rules can potentially be shared for the same target ports. Load balancer subnets can be manually specified 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 (ALB) requires fewer rules than NLBs. An ALB can also be shared across multiple Ingresses. For more information, see the section called "Application load balancing" (p. 271).
  - 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.

Load balancer – Instance targets

NLB or CLBs with instance targets are created by the Kubernetes in-tree load balancing controller. The in-tree controller is included with Kubernetes, so you don't need to deploy it to your cluster. You can use NLB instance targets with pods deployed to nodes, but not to Fargate. To load balance network traffic across pods deployed to Fargate, you must use IP targets (p. 269). By default, external (public) Classic Load Balancers are created when you deploy a Kubernetes service of type `LoadBalancer`. To deploy a Network Load Balancer instead, apply the following annotation to your service:

```
service.beta.kubernetes.io/aws-load-balancer-type: nlb
```

Important
Do not edit this annotation 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.

To deploy a load balancer to a private subnet, your service specification must have the following annotation:

```
service.beta.kubernetes.io/aws-load-balancer-internal: "true"
```

For internal load balancers, your Amazon EKS cluster must be configured to use at least one private subnet in your VPC. Kubernetes examines the route table for your subnets to identify whether they are public or private. Public subnets have a route directly to the internet using an internet gateway, but private subnets do not.

For an example service manifest that specifies a load balancer, see Type LoadBalancer in the Kubernetes documentation. For more information about using Network Load Balancer with Kubernetes, see Network Load Balancer support on AWS in the Kubernetes documentation.

Load balancer – IP targets

NLBs with IP targets are created by the AWS Load Balancer Controller (you cannot use CLBs with IP targets). You can use NLB 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 annotation to a service manifest and deploy your service.

```
service.beta.kubernetes.io/aws-load-balancer-type: nlb-ip
```

> **Important**
> Do not edit this annotation 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. You can only use NLB *IP* targets with clusters running at least Amazon EKS version 1.18. To upgrade your current version, see the section called “Updating a cluster” (p. 24).

To deploy a sample application

1. Deploy a sample application.

   a. Save the following contents to a file named *`sample-deployment.yaml`* file on your computer.

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: sample-app
spec:
  replicas: 3
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
        - name: nginx
          image: public.ecr.aws/nginx/nginx:1.19.6
          ports:
            - name: http
```

```
              containerPort: 80
```

b. Apply the manifest to the cluster.

```
kubectl apply -f sample-deployment.yaml
```

2. Create a service of type `LoadBalancer` with an annotation to create an NLB with IP targets.

   a. Save the following contents to a file named *`sample-service.yaml`* file on your computer.

   ```
   apiVersion: v1
   kind: Service
   metadata:
     name: sample-service
     annotations:
       service.beta.kubernetes.io/aws-load-balancer-type: nlb-ip
   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
   ```

3. Verify that the service was deployed.

   ```
   kubectl get svc sample-service
   ```

   Output

   ```
   NAME             TYPE           CLUSTER-IP       EXTERNAL-IP
                                            PORT(S)        AGE
   sample-service   LoadBalancer   10.100.240.137   k8s-default-samplese-xxxxxxxxxx-
   xxxxxxxxxxxxxxxx.elb.us-west-2.amazonaws.com   80:32400/TCP   16h
   ```

4. Open the Amazon EC2 AWS Management Console. Select **Target Groups** (under **Load Balancing**) in the left panel. In the **Name** column, select the target group's name that matches the name in the `EXTERNAL-IP` column of the output in the previous step. For example, you'd select the target group named r `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 deployment manifest.
5. 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 may take several minutes before all targets are `healthy`. The targets may have an `unhealthy` state before changing to `healthy`.
6. Send traffic to the service replacing the example value with the value returned in a previous step for the `EXTERNAL-IP`.

   ```
   curl <k8s-default-samplese-xxxxxxxxxx-xxxxxxxxxxxxxxxx.elb.us-west-2.amazonaws.com>
   ```

   Output

   ```
   <!DOCTYPE html>
   <html>
   ```

```
<head>
<title>Welcome to nginx!</title>
...
```

7. When you're finished with the sample deployment and service, remove them.

```
kubectl delete service sample-service
kubectl delete deployment sample-app
```

Application load balancing on Amazon EKS

When you create a Kubernetes `Ingress`, an AWS Application Load Balancer 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 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 type `LoadBalancer`, which provisions an AWS Network Load Balancer. For more information, see the section called "Network load balancing" (p. 266). 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 the section called "Updating a cluster" (p. 24).
- The AWS Load Balancer Controller provisioned on your cluster. For more information, see the section called "AWS Load Balancer Controller" (p. 238).
- At least two subnets in different Availability Zones. The AWS load balancer controller chooses one subnet from each Availability Zone. In the case of multiple tagged subnets found in an Availability Zone, the controller chooses the first subnet in lexicographical order by the subnet IDs.
- 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 per cluster. Replace *`<cluster-name>`* (including <>) with your cluster name.
  - **Key** – `kubernetes.io/cluster/`*`<cluster-name>`*
  - **Value** – `shared` or `owned`
- **Subnet tagging** – 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 are not required.
  - **Private subnets** – Must be tagged as follows 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 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 AWS CloudFormation VPC templates, see the section called "Creating a VPC for Amazon EKS" (p. 204).

  - **Key** – `kubernetes.io/role/internal-elb`
  - **Value** – `1`
- **Public subnets** – Must be tagged as follows so that Kubernetes knows to use only those subnets for external load balancers instead of choosing a public subnet in each Availability Zone (in lexicographical order by subnet ID). If you use `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. 204).
  - **Key** – `kubernetes.io/role/elb`
  - **Value** – `1`

If the subnet role tags are not 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 do not rely on this behavior, and instead explicitly add the private or public role tags. The AWS load balancer controller does not examine route tables, and 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
```

- 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.

To share an application load balancer across multiple ingress resources using `IngressGroups`

To join an Ingress to an Ingress 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 characters or less in length.

- Consist of lower case alphanumeric characters, `-`, and `.`, and must start and end with an alphanumeric character.

The controller will automatically merge ingress rules for all Ingresses in the same Ingress group and support them with a single ALB. Most annotations 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**: You should only specify an Ingress group for an Ingress when all Kubernetes users with 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 may 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 between 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 are determined by the lexical order of an Ingress' 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.

To deploy a sample application

You can run the sample application on a cluster that has Amazon EC2 nodes only, Fargate pods, or both.

1. If you're deploying to Fargate, 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 you can create the profile with the AWS Management Console (p. 130) using the same values for `name` and `namespace` that are in the command.

   ```
   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.

   ```
   kubectl apply -f https://raw.githubusercontent.com/kubernetes-sigs/aws-load-balancer-
   controller/v2.1.3/docs/examples/2048/2048_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.us-
   west-2.elb.amazonaws.com   80      2m32s
   ```

Note

If your Ingress has not been created after several minutes, run the following command to view the Load Balancer Controller logs. These logs may contain error messages that can help you diagnose any issues with your deployment.

```
kubectl logs -n kube-system   deployment.apps/aws-load-balancer-controller
```

4. 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, wait a few minutes and refresh your browser.
5. When you finish experimenting with your sample application, delete it with the following commands.

```
kubectl delete -f https://raw.githubusercontent.com/kubernetes-sigs/aws-load-balancer-
controller/v2.1.3/docs/examples/2048/2048_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

1. 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/v1.1.1/
cert-manager.yaml
```

2. Verify that the cert-manager pods are running.

```
kubectl get pods -n cert-manager
```

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
```

3. 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 --all-namespaces
```

Output

```
NAMESPACE                      NAME                                     TYPE
 CLUSTER-IP       EXTERNAL-IP     PORT(S)         AGE
cert-manager                   cert-manager                             ClusterIP
 10.100.102.137   <none>          9402/TCP        20m
cert-manager                   cert-manager-webhook                     ClusterIP
 10.100.6.136     <none>          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   <none>          443/TCP         16s
kube-system                    kube-dns                                 ClusterIP
 10.100.0.10      <none>          53/UDP,53/TCP   2d1h
my-namespace                   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.

4. 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. 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 mutiple blocks, add a comma between blocks.

6. If your cluster is not in the `us-west-2` Region, replace *`us-west-2`*, *`602401143452`*, and *`.amazonaws.com/`* with the appropriate values for your Region from the list in the section called "Add-on container image addresses" (p. 275).

```
image:602401143452.dkr.ecr.us-west-2.amazonaws.com/externalip-webhook:v1.0.0
```

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 step 5 will fail.

Amazon EKS add-on container image addresses

When you deploy Amazon EKS add-ons such as the the section called "AWS Load Balancer Controller" (p. 238), the VPC CNI plug-in (p. 214), kube-proxy (p. 25), CoreDNS (p. 243), or storage drivers (p. 180), you pull an image from an Amazon ECR repository. The image name and tag are listed in the topics for each add-on.

The following table contains a list of Regions and the addresses you can use to pull images from.

Region	Address
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/
ca-central-1	602401143452.dkr.ecr.ca-central-1.amazonaws.com/
cn-north-1	918309763551.dkr.ecr.cn-north-1.amazonaws.com.cn/
cn-northwest-1	961992271922.dkr.ecr.cn-northwest-1.amazonaws.com.cn/
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/

Region	Address
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/
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/

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.
- The AWS CLI version 2.1.26 or later or 1.19.7 or later 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 configurein the AWS Command Line Interface User Guide.

- `Kubectl` version or later installed on your computer or AWS CloudShell. To install or upgrade `kubectl`, see the section called "Installing `kubectl`" (p. 303).
- 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 the section called "CNI upgrades" (p. 236).

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 Region that your cluster is in.

   ```
   aws ec2 describe-instance-types \
       --region us-west-2 \
       --filters Name=network-info.efa-supported,Values=true \
       --query "InstanceTypes[*].[InstanceType]" \
       --output text
   ```

   b. Determine which Availability Zones the instance you select from the previous output is available in.

   ```
   aws ec2 describe-instance-type-offerings \
       --location-type availability-zone \
       --filters Name=instance-type,Values=p4d.24xlarge \
       --region us-west-2 \
       --output table
   ```

   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

   `Eksctl` version 0.43.0 or later installed on your computer or AWS CloudShell. To install or upgrade `eksctl`, see the section called "Installing `eksctl`" (p. 308).

   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: us-west-2
     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="us-west-2"
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
```

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.yml
```

h. If you deployed an instance type with a GPU, deploy the NVIDIA Kubernetes device plugin.

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

(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/
   nccl-efa-tests.yaml
   ```

   Output

   mpijob.kubeflow.org/nccl-tests-efa created
2. View your running pods.

   ```
   kubectl get pods
   ```

   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).

3. 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.

Considerations

- 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.
- Amazon EC2 Inf1 instances are not currently supported with managed node groups.

Prerequisites

- Have `eksctl` installed on your computer. If you don't have it installed, see The `eksctl` command line utility (p. 308) for installation instructions.
- Have `kubectl` installed on your computer. For more information, see Installing `kubectl` (p. 303).
- (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

1. Create a cluster with Inf1 Amazon EC2 instance nodes. You can replace *<inf1.2xlarge>* with any Inf1 instance type. `Eksctl` detects that you are launching a node group with an `Inf1` instance type and will start your nodes using one of the the section called “Amazon EKS optimized accelerated Amazon Linux AMI” (p. 153).

   Note
   You can't use IAM roles for service accounts (p. 345) with TensorFlow Serving.

   ```
   eksctl create cluster \
       --name <inferentia> \
       --region <region-code> \
       --nodegroup-name <ng-inf1> \
       --node-type <inf1.2xlarge> \
       --nodes <2> \
   ```

```
    --nodes-min <1> \
    --nodes-max <4> \
    --ssh-access \
    --ssh-public-key <your-key> \
    --with-oidc \
    --managed
```

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-6djhp    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. 283). 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.

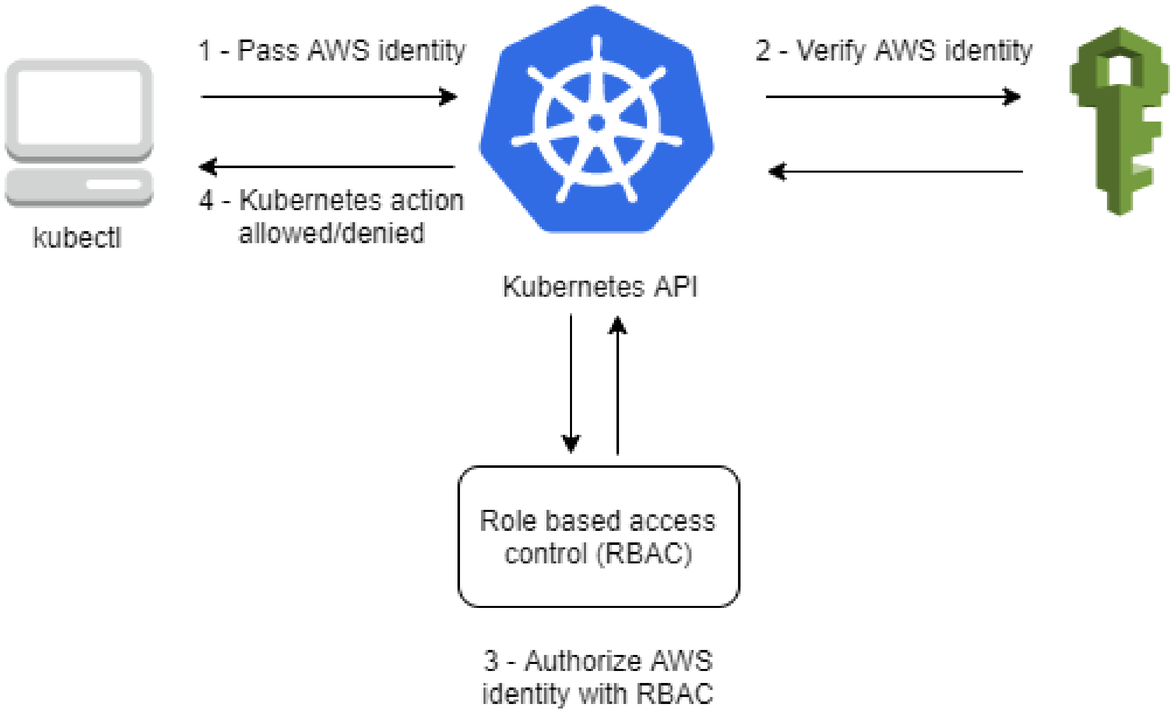
```
python3 tensorflow-model-server-infer.py
```

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.


1 - Pass AWS identity
2 - Verify AWS identity
kubectl
4 - Kubernetes action allowed/denied
Kubernetes API
Role based access control (RBAC)
3 - Authorize AWS identity with RBAC

Topics

Managing users or IAM roles for your cluster

When you create an Amazon EKS cluster, the IAM entity user or role, such as a federated user that creates the cluster, is automatically granted `system:masters` permissions in the cluster's RBAC configuration in the control plane. This IAM entity does not appear in the ConfigMap, or any other 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.

Note

For more information about different IAM identities, see Identities (Users, Groups, and Roles) in the *IAM User Guide*. For more information on Kubernetes RBAC configuration, see Using RBAC Authorization. For all ConfigMap settings, see Full Configuration Format on GitHub.

The aws-auth ConfigMap is applied as part of the Getting started with Amazon EKS (p. 4) guide which provides a complete end-to-end walkthrough from creating an Amazon EKS cluster to deploying a sample Kubernetes application. It is initially created to allow your nodes to join your cluster, but you also use this ConfigMap to add RBAC access to IAM users and roles. If you have not launched nodes and applied the aws-auth ConfigMap, you can do so with the following procedure.

To apply the aws-auth ConfigMap to your cluster

1. Check to see if you have already applied the aws-auth ConfigMap.

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

   If you receive an error stating "Error from server (NotFound): configmaps "aws-auth" not found", then proceed with the following steps to apply the stock ConfigMap.
2. Download, edit, and apply the AWS authenticator configuration map.

   a. Download the configuration map.

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

   b. Open the file with your favorite 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 cannot include a path. The format of the role ARN must be arn:aws:iam::<123456789012>:role/<role-name>. For more information, see aws-auth ConfigMap does not grant access to the cluster (p. 382).

```
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 worker 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. 375) in the troubleshooting section.
3. Watch the status of your nodes and wait for them to reach the Ready status.

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

To add an IAM user or role to an Amazon EKS cluster

1. Ensure that the AWS credentials that `kubectl` is using are already authorized for your cluster. The IAM user that created the cluster has these permissions by default.
2. Open the `aws-auth` ConfigMap.

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

   Note
   If you receive an error stating "`Error from server (NotFound): configmaps "aws-auth" not found`", then use the previous procedure to apply the stock ConfigMap.

   Example ConfigMap:

   ```
   apiVersion: v1
   data:
     mapRoles: |
       - groups:
         - system:bootstrappers
         - system:nodes
         rolearn: arn:aws:iam::111122223333:role/eksctl-my-cluster-nodegroup-standard-wo-
   NodeInstanceRole-1WP3NUE3O6UCF
         username: system:node:{{EC2PrivateDNSName}}
   kind: ConfigMap
   metadata:
     creationTimestamp: "2020-09-30T21:09:18Z"
     name: aws-auth
     namespace: kube-system
     resourceVersion: "1021"
     selfLink: /api/v1/namespaces/kube-system/configmaps/aws-auth
     uid: dcc31de5-3838-11e8-af26-02e00430057c
   ```
3. Add your IAM users, roles, or AWS accounts to the configMap. You cannot 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.
     - **username**: The user name within Kubernetes to map to the IAM role.
     - **groups**: A list of groups within Kubernetes to which the role is mapped. For more information, see Default Roles and Role Bindings in the Kubernetes documentation.
   - **To add an IAM user:** add the user details to the `mapUsers` section of the ConfigMap, under `data`. Add this section if it does not already exist in the file. Each entry supports the following parameters:
     - **userarn**: The ARN of the IAM user to add.
     - **username**: The user name within Kubernetes to map to the IAM user.
     - **groups**: A list of groups within Kubernetes to which the user is mapped to. For more information, see Default Roles and Role Bindings in the Kubernetes documentation.

   For example, the block below contains:

- A `mapRoles` section that adds the node instance role so that nodes can register themselves with the cluster.
- A `mapUsers` section with the AWS users `admin` from the default AWS account, and `ops-user` from another AWS account. Both users are added to the `system:masters` group.

Replace all `<example-values>` (including `<>`) 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: |
    - rolearn: <arn:aws:iam::111122223333:role/eksctl-my-cluster-nodegroup-standard-wo-
NodeInstanceRole-1WP3NUE3O6UCF>
      username: <system:node:{{EC2PrivateDNSName}}>
      groups:
        - <system:bootstrappers>
        - <system:nodes>
  mapUsers: |
    - userarn: <arn:aws:iam::111122223333:user/admin>
      username: <admin>
      groups:
        - <system:masters>
    - userarn: <arn:aws:iam::111122223333:user/ops-user>
      username: <ops-user>
      groups:
        - <system:masters>
```

4. Save the file and exit your text editor.
5. Ensure that the Kubernetes user or group that you mapped the IAM user or role to is bound to a Kubernetes role with a `RoleBinding` or `ClusterRoleBinding`. For more information, see Using RBAC Authorization in the Kubernetes documentation. You can download the following example manifests that create a `clusterrole` and `clusterrolebinding` or a `role` and `rolebinding`:

   - **View Kubernetes resources in all namespaces** – The group name in the file is `eks-console-dashboard-full-access-group`, which is the group that your IAM user or role needs to be mapped to in the `aws-auth` configmap. You can change the name of the group before applying it to your cluster, if desired, and then map your IAM user or role to that group in the configmap. Download the file from:

     ```
     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`, so if you want to specify a different namespace, edit the file before applying it to your cluster. The group name in the file is `eks-console-dashboard-restricted-access-group`, which is the group that your IAM user or role needs to be mapped to in the `aws-auth` configmap. You can change the name of the group before applying it to your cluster, if desired, and then map your IAM user or role to that group in the configmap. Download the file from:

     ```
     https://s3.us-west-2.amazonaws.com/amazon-eks/docs/eks-console-restricted-access.yaml
     ```

6. (Optional) If you want the users you've added to the configmap to be able to the section called "View nodes" (p. 86) or the section called "View workloads" (p. 253) in the AWS Management Console, then the user or role must have the appropriate permissions to view the resources in Kubernetes, but also need to have the appropriate IAM permissions to view those resources in the

AWS Management Console. For more information, see the section called “View nodes and workloads for all clusters in the AWS Management Console” (p. 333).

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 the section called “Managing users or IAM roles for your cluster” (p. 288). 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

- Your cluster must be running Kubernetes 1.16 or later.
- 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 the section called “Enabling and disabling control plane logs” (p. 56).
- You can't sign in to the AWS Management Console with an account from an OIDC provider. You can only View nodes (p. 86) and Workloads (p. 253) 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>* (including *<>*) 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: <us-west-2>

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 paris 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-2:111122223333: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-2.amazonaws.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": {
                    "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. You can quickly create or update a kubeconfig with the AWS CLI `update-kubeconfig` command automatically by using the AWS CLI, or you can create a kubeconfig 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 will use the same credentials that are returned with the following command:

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

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

Create `kubeconfig` automatically

To create your `kubeconfig` file with the AWS CLI

1. Ensure that you have version 1.16.156 or later of the AWS CLI installed. To install or upgrade the AWS CLI, see Installing the AWS Command Line Interface in the *AWS Command Line Interface User Guide*.

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

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

   ```
   aws --version
   ```

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

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

   - By default, the resulting configuration file is created at the default kubeconfig path (`.kube/config`) in your home directory or merged with an existing kubeconfig at that location. You can specify another path with the `--kubeconfig` option.
   - You can specify an IAM role ARN with the `--role-arn` option to use for authentication when you issue `kubectl` commands. Otherwise, the IAM entity in your default AWS CLI or SDK credential chain is used. You can view your default AWS CLI or SDK identity by running the `aws sts get-caller-identity` command.
   - For more information, see the help page with the `aws eks update-kubeconfig help` command or see update-kubeconfig in the *AWS CLI Command Reference*.

   Note
   To run the following command, you must have permission to the use the `eks:DescribeCluster` API action with the cluster that you specify. For more information, see Amazon EKS identity-based policy examples (p. 331).

   ```
   aws eks --region <region-code> update-kubeconfig --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. 375) 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. Create the default `~/.kube` directory if it does not already exist.

```
mkdir -p ~/.kube
```

2. Open your favorite text editor and copy one of the `kubeconfig` code blocks below into it, depending on your preferred client token method.

   - To use the AWS CLI `aws eks get-token` command (requires version 1.16.156 or later of the AWS CLI):

```
apiVersion: v1
clusters:
- cluster:
    server: <endpoint-url>
    certificate-authority-data: <base64-encoded-ca-cert>
  name: kubernetes
contexts:
- context:
    cluster: kubernetes
    user: aws
  name: aws
current-context: aws
kind: Config
preferences: {}
users:
- name: aws
  user:
    exec:
      apiVersion: client.authentication.k8s.io/v1alpha1
      command: aws
      args:
        - "eks"
        - "get-token"
        - "--cluster-name"
        - "<cluster-name>"
        # - "--role-arn"
        # - "<role-arn>"
      # env:
        # - name: AWS_PROFILE
        #   value: "<aws-profile>"
```

   - To use the AWS IAM authenticator for Kubernetes:

```
apiVersion: v1
clusters:
- cluster:
    server: <endpoint-url>
    certificate-authority-data: <base64-encoded-ca-cert>
  name: kubernetes
contexts:
- context:
    cluster: kubernetes
    user: aws
  name: aws
current-context: aws
kind: Config
preferences: {}
users:
```

```
- name: aws
  user:
    exec:
      apiVersion: client.authentication.k8s.io/v1alpha1
      command: aws-iam-authenticator
      args:
        - "token"
        - "-i"
        - "<cluster-name>"
        # - "-r"
        # - "<role-arn>"
      # env:
        # - name: AWS_PROFILE
        #   value: "<aws-profile>"
```

3. Replace the `<endpoint-url>` with the endpoint URL that was created for your cluster.
4. Replace the `<base64-encoded-ca-cert>` with the `certificateAuthority.data` that was created for your cluster.
5. Replace the `<cluster-name>` with your cluster name.
6. (Optional) To assume an IAM role to perform cluster operations instead of the default AWS credential provider chain, uncomment the `-r` or `--role` and `<role-arn>` lines and substitute an IAM role ARN to use with your user.
7. (Optional) To always use a specific named AWS credential profile (instead of the default AWS credential provider chain), uncomment the `env` lines and substitute `<aws-profile>` with the profile name to use.
8. Save the file to the default `kubectl` folder, with your cluster name in the file name. For example, if your cluster name is `<devel>`, save the file to `~/.kube/config-<devel>`.
9. Add that file path to your `KUBECONFIG` environment variable so that `kubectl` knows where to look for your cluster configuration.

   - For Bash shells on macOS or Linux:

   ```
   export KUBECONFIG=$KUBECONFIG:~/.kube/config-<devel>
   ```

   - For PowerShell on Windows:

   ```
   $ENV:KUBECONFIG="{0};{1}" -f  $ENV:KUBECONFIG, "$ENV:userprofile\.kube\config-
   <devel>"
   ```

10. (Optional) Add the configuration to your shell initialization file so that it is configured when you open a shell.

    - For Bash shells on macOS:

    ```
    echo 'export KUBECONFIG=$KUBECONFIG:~/.kube/config-<devel>' >> ~/.bash_profile
    ```

    - For Bash shells on Linux:

    ```
    echo 'export KUBECONFIG=$KUBECONFIG:~/.kube/config-<devel>' >> ~/.bashrc
    ```

    - For PowerShell on Windows:

    ```
    [System.Environment]::SetEnvironmentVariable('KUBECONFIG', $ENV:KUBECONFIG,
     'Machine')
    ```

11. Test your configuration.

```
kubectl get svc
```

> **Note**
> If you receive any authorization or resource type errors, see Unauthorized or access denied (`kubectl`) (p. 375) 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 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. 297).

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://amazon-eks.s3.us-
west-2.amazonaws.com/1.19.6/2021-01-05/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://amazon-eks.s3.us-
   west-2.amazonaws.com/1.19.6/2021-01-05/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.

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

2. (Optional) Verify the downloaded binary with the SHA-256 sum provided in the same bucket prefix.

   a. Download the SHA-256 sum for your system. To download the Arm version, change *`<amd64>`* to `arm64` before running the command.

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

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

```
openssl sha1 -sha256 aws-iam-authenticator
```

   c. Compare the generated SHA-256 sum in the command output against your downloaded `aws-iam-authenticator.sha256` file. The two should match.

3. Apply execute permissions to the binary.

```
chmod +x ./aws-iam-authenticator
```

4. Copy the binary to a folder in your `$PATH`. We recommend creating a `$HOME/bin/aws-iam-authenticator` and ensuring that `$HOME/bin` comes first in your `$PATH`.

```
mkdir -p $HOME/bin && cp ./aws-iam-authenticator $HOME/bin/aws-iam-authenticator &&
 export PATH=$PATH:$HOME/bin
```

5. Add `$HOME/bin` to your `PATH` environment variable.

```
echo 'export PATH=$PATH:$HOME/bin' >> ~/.bashrc
```

6. Test that the `aws-iam-authenticator` binary works.

```
aws-iam-authenticator help
```

Windows

To install `aws-iam-authenticator` on Windows with Chocolatey

1. If you do not already have Chocolatey installed on your Windows system, see Installing chocolatey.
2. Open a PowerShell terminal window and install the `aws-iam-authenticator` package with the following command:

```
choco install -y aws-iam-authenticator
```

3. Test that the `aws-iam-authenticator` binary works.

```
aws-iam-authenticator help
```

To install `aws-iam-authenticator` on Windows

1. Open a PowerShell terminal window and download the Amazon EKS vended `aws-iam-authenticator` binary from Amazon S3.

```
curl -o aws-iam-authenticator.exe https://amazon-eks.s3.us-
west-2.amazonaws.com/1.19.6/2021-01-05/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://amazon-eks.s3.us-
west-2.amazonaws.com/1.19.6/2021-01-05/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. 295). Otherwise, see Creating an Amazon EKS cluster (p. 17) to create a new Amazon EKS cluster.

Cluster management

This chapter includes the following topics to help you manage your cluster. You can also view information about your nodes (p. 86) and workloads (p. 253) using the AWS Management Console.

- Installing `kubectl` (p. 303) – Learn how to install `kubectl`; a command line tool for managing Kubernetes.
- The `eksctl` command line utility (p. 308) – Learn how to install a simple command line utility for creating and managing Kubernetes clusters on Amazon EKS.
- Tutorial: Deploy the Kubernetes Dashboard (web UI) (p. 310) – Learn how to install the dashboard, a web-based user interface for your Kubernetes cluster and applications.
- Installing the Kubernetes Metrics Server (p. 314) – 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. 263). In this topic you learn how to install the Metrics Server.
- Control plane metrics with Prometheus (p. 314) – 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. 317) – 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. 318) – 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. 322) – 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.

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 1.18 `kubectl` client works with Kubernetes 1.17, 1.18 and 1.19 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 1.19:**

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

- **Kubernetes 1.18:**

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

- **Kubernetes 1.17:**

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

- **Kubernetes 1.16:**

```
curl -o kubectl https://amazon-eks.s3.us-west-2.amazonaws.com/1.16.15/2020-11-02/
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 1.19:**

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

   - **Kubernetes 1.18:**

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

   - **Kubernetes 1.17:**

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

   - **Kubernetes 1.16:**

   ```
   curl -o kubectl.sha256 https://amazon-eks.s3.us-
   west-2.amazonaws.com/1.16.15/2020-11-02/bin/darwin/amd64/kubectl.sha256
   ```

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

   ```
   openssl sha1 -sha256 kubectl
   ```

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

3. Apply execute permissions to the binary.

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

4. Copy the binary to a folder in your `PATH`. If you have already installed a version of `kubectl`, then we recommend creating a `$HOME/bin/kubectl` and ensuring that `$HOME/bin` comes first in your `$PATH`.

```
mkdir -p $HOME/bin && cp ./kubectl $HOME/bin/kubectl && export PATH=$HOME/bin:$PATH
```

5. (Optional) Add the `$HOME/bin` path to your shell initialization file so that it is configured when you open a shell.

```
echo 'export PATH=$PATH:$HOME/bin' >> ~/.bash_profile
```

6. After you install `kubectl`, you can verify its version with the following command:

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

Linux

To install `kubectl` on Linux

1. Download the Amazon EKS vended `kubectl` binary for your cluster's Kubernetes version from Amazon S3. To download the Arm version, change *`amd64`* to `arm64` before running the command.

   - **Kubernetes 1.19:**

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

   - **Kubernetes 1.18:**

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

   - **Kubernetes 1.17:**

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

   - **Kubernetes 1.16:**

   ```
   curl -o kubectl https://amazon-eks.s3.us-west-2.amazonaws.com/1.16.15/2020-11-02/
   bin/linux/amd64/kubectl
   ```

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

   a. Download the SHA-256 sum for your cluster's Kubernetes version for Linux. To download the Arm version, change *`<amd64>`* to `arm64` before running the command.

      - **Kubernetes 1.19:**

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

      - **Kubernetes 1.18:**

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

      - **Kubernetes 1.17:**

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

- **Kubernetes 1.16:**

```
curl -o kubectl.sha256 https://amazon-eks.s3.us-
west-2.amazonaws.com/1.16.15/2020-11-02/bin/linux/amd64/kubectl.sha256
```

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

```
openssl sha1 -sha256 kubectl
```

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

3. Apply execute permissions to the binary.

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

4. Copy the binary to a folder in your `PATH`. If you have already installed a version of `kubectl`, then we recommend creating a `$HOME/bin/kubectl` and ensuring that `$HOME/bin` comes first in your `$PATH`.

```
mkdir -p $HOME/bin && cp ./kubectl $HOME/bin/kubectl && export PATH=$PATH:$HOME/bin
```

5. (Optional) Add the `$HOME/bin` path to your shell initialization file so that it is configured when you open a shell.

Note
This step assumes you are using the Bash shell; if you are using another shell, change the command to use your specific shell initialization file.

```
echo 'export PATH=$PATH:$HOME/bin' >> ~/.bashrc
```

6. After you install `kubectl` , you can verify its version with the following command:

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

Windows

To install `kubectl` on Windows

1. Open a PowerShell terminal.
2. Download the Amazon EKS vended `kubectl` binary for your cluster's Kubernetes version from Amazon S3:

- **Kubernetes 1.19:**

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

- **Kubernetes 1.18:**

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

- **Kubernetes 1.17:**

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

- **Kubernetes 1.16:**

```
curl -o kubectl.exe https://amazon-eks.s3.us-
west-2.amazonaws.com/1.16.15/2020-11-02/bin/windows/amd64/kubectl.exe
```

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

   a. Download the SHA-256 sum for your cluster's Kubernetes version for Windows:

      - **Kubernetes 1.19:**

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

      - **Kubernetes 1.18:**

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

      - **Kubernetes 1.17:**

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

      - **Kubernetes 1.16:**

      ```
      curl -o kubectl.exe.sha256 https://amazon-eks.s3.us-
      west-2.amazonaws.com/1.16.15/2020-11-02/bin/windows/amd64/kubectl.exe.sha256
      ```

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

   ```
   Get-FileHash kubectl.exe
   ```

   c. Compare the generated SHA-256 sum in the command output against your downloaded SHA-256 file. The two should match, although the PowerShell output will be uppercase.

4. Copy the binary to a folder in your `PATH`. If you have an existing directory in your PATH that you use for command line utilities, copy the binary to that directory. Otherwise, complete the following steps.

   a. Create a new directory for your command line binaries, such as `C:\bin`.

   b. Copy the `kubectl.exe` binary to your new directory.

   c. Edit your user or system PATH environment variable to add the new directory to your PATH.

   d. Close your PowerShell terminal and open a new one to pick up the new PATH variable.

5. After you install `kubectl`, you can verify its version with the following command:

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

The eksctl command line utility

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/.

Installing or upgrading eksctl

This section helps you to install or upgrade the latest version of the eksctl command line utility. Select the tab with the name of the operating system that you want to install eksctl on.

macOS

To install or upgrade eksctl on macOS using Homebrew

The easiest way to get started with Amazon EKS and macOS is by installing eksctl with Homebrew. The eksctl Homebrew recipe installs eksctl and any other dependencies that are required for Amazon EKS, such as kubectl. The recipe also installs the aws-iam-authenticator (p. 299), 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 0.43.0. If not, check your terminal output for any installation or upgrade errors, or manually download an archive of the release from https://github.com/weaveworks/eksctl/releases/download/0.43.0/eksctl_Darwin_amd64.tar.gz, extract eksctl, and then run it.

Linux

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

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

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

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

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

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

   ```
   eksctl version
   ```

   Note
   The `GitTag` version should be at least `0.43.0`. If not, check your terminal output for any installation or upgrade errors, or replace the address in step 1 with `https://github.com/weaveworks/eksctl/releases/download/0.43.0/eksctl_Linux_amd64.tar.gz` and complete steps 1-3 again.

Windows

To install or upgrade `eksctl` on Windows using Chocolatey

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

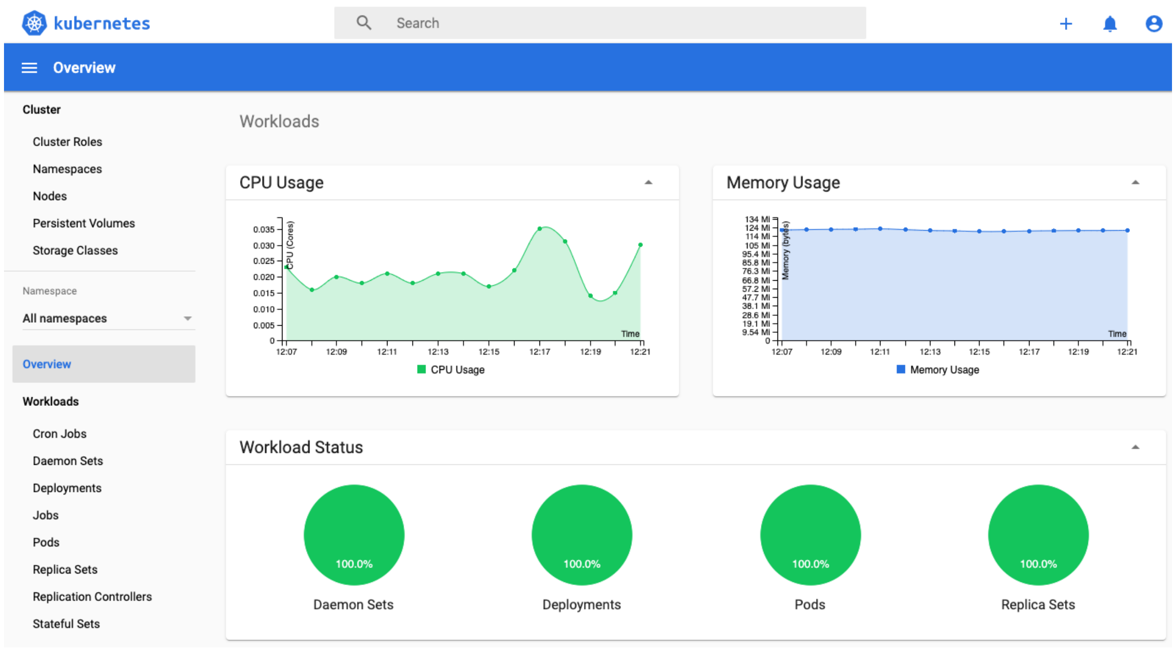
3. Test that your installation was successful with the following command.

   ```
   eksctl version
   ```

   Note
   The `GitTag` version should be at least `0.43.0`. If not, check your terminal output for any installation or upgrade errors, or manually download an archive of the release from https://github.com/weaveworks/eksctl/releases/download/0.43.0/eksctl_Windows_amd64.zip, extract `eksctl`, and then run it.

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 the section called "Metrics server" (p. 314).
- The security groups for your control plane elastic network interfaces and nodes follow the recommended settings in Amazon EKS security group considerations (p. 210).
- You are using a `kubectl` client that is configured to communicate with your Amazon EKS cluster (p. 11).

Step 2: Deploy the Kubernetes dashboard

Complete the instructions for the option that corresponds to the Region that your cluster is in.

- All Regions other than Beijing and Ningxia China

```
kubectl apply -f https://raw.githubusercontent.com/kubernetes/dashboard/v2.0.5/aio/
deploy/recommended.yaml
```

- Beijing and Ningxia China

1. Download the Kubernetes Dashboard manifest with the following command.

   ```
   curl -o recommended.yaml https://raw.githubusercontent.com/kubernetes/dashboard/v2.0.5/
   aio/deploy/recommended.yaml
   ```

2. Edit the manifest files using the following steps.

   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> <aws_account_id>.dkr.ecr.<cn-north-1>.amazonaws.com/
      image:<tag>
      ```

   c. Push the image to a China Amazon ECR repository with the following command.

      ```
      docker push image:<tag> <aws_account_id>.dkr.ecr.<cn-north-1>.amazonaws.com/
      image:<tag>
      ```

   d. Update the Kubernetes manifest file or files to reference the Amazon ECR image URL in your Region.

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 3: Create an `eks-admin` service account and cluster role binding

By default, the Kubernetes Dashboard user has limited permissions. In this section, you create an `eks-admin` service account and cluster role binding that you can use to securely connect to the dashboard with admin-level permissions. For more information, see Managing Service Accounts in the Kubernetes documentation.

To create the `eks-admin` service account and cluster role binding

Important
The example service account created with this procedure has full `cluster-admin` (superuser) privileges on the cluster. For more information, see Using RBAC authorization in the Kubernetes documentation.

1. Create a file called `eks-admin-service-account.yaml` with the text below. This manifest defines a service account and cluster role binding called `eks-admin`.

```
apiVersion: v1
kind: ServiceAccount
metadata:
  name: eks-admin
  namespace: kube-system
---
apiVersion: rbac.authorization.k8s.io/v1beta1
kind: ClusterRoleBinding
metadata:
  name: eks-admin
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: cluster-admin
subjects:
- kind: ServiceAccount
  name: eks-admin
  namespace: kube-system
```

2. Apply the service account and cluster role binding to your cluster.

```
kubectl apply -f eks-admin-service-account.yaml
```

Output:

```
serviceaccount "eks-admin" created
clusterrolebinding.rbac.authorization.k8s.io "eks-admin" created
```

Step 4: Connect to the dashboard

Now that the Kubernetes Dashboard is deployed to your cluster, and you have an administrator service account that you can use to view and control your cluster, you can connect to the dashboard with that service account.

To connect to the Kubernetes dashboard

1. Retrieve an authentication token for the `eks-admin` service account. Copy the `<authentication_token>` value from the output. You use this token to connect to the dashboard.

```
kubectl -n kube-system describe secret $(kubectl -n kube-system get secret | grep eks-
admin | awk '{print $1}')
```

Output:

```
Name:         eks-admin-token-b5zv4
Namespace:    kube-system
Labels:       <none>
```

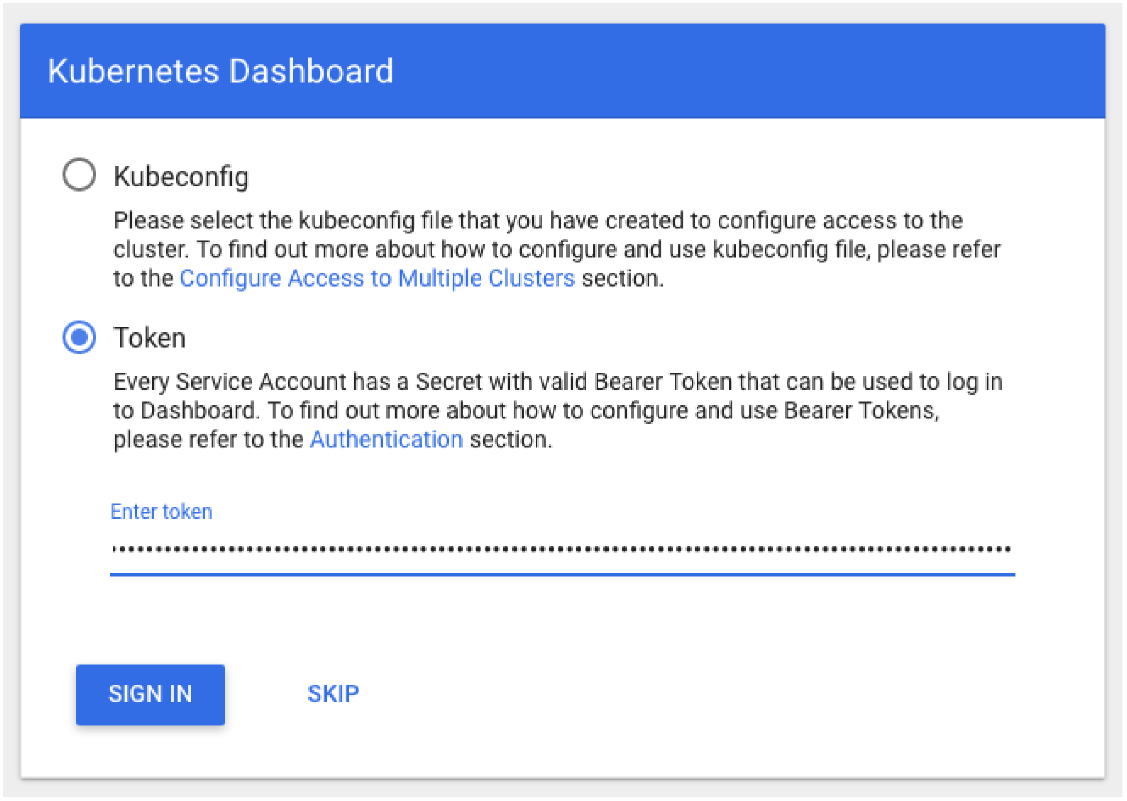
```
Annotations:  kubernetes.io/service-account.name=eks-admin
              kubernetes.io/service-account.uid=bcfe66ac-39be-11e8-97e8-026dce96b6e8

Type:  kubernetes.io/service-account-token

Data
====
ca.crt:      1025 bytes
namespace:   11 bytes
token:       <authentication_token>
```

2. Start the `kubectl proxy`.

```
kubectl proxy
```

3. To access the dashboard endpoint, open the following link with a web browser: http://localhost:8001/api/v1/namespaces/kubernetes-dashboard/services/https:kubernetes-dashboard:/proxy/#!/login.

4. Choose **Token**, paste the `<authentication_token>` output from the previous command into the **Token** field, and choose **SIGN IN**.



Kubernetes Dashboard

○ Kubeconfig

Please select the kubeconfig file that you have created to configure access to the cluster. To find out more about how to configure and use kubeconfig file, please refer to the Configure Access to Multiple Clusters section.

● Token

Every Service Account has a Secret with valid Bearer Token that can be used to log in to Dashboard. To find out more about how to configure and use Bearer Tokens, please refer to the Authentication section.

Enter token
••

Note
It may take a few minutes before CPU and memory metrics appear in the dashboard.

Step 5: Next steps

After you have connected to your Kubernetes Dashboard, you can view and control your cluster using your `eks-admin` service account. For more information about using the dashboard, see the project documentation on GitHub.

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. 263) or the Kubernetes Dashboard (p. 310). 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.

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
```

2. Verify that the `metrics-server` deployment is running the desired number of pods with the following command.

```
kubectl get deployment metrics-server -n kube-system
```

Output

```
NAME             READY   UP-TO-DATE   AVAILABLE   AGE
metrics-server   1/1     1            1           6m
```

Control plane metrics with Prometheus

The Kubernetes API server exposes a number of metrics that are useful for monitoring and analysis. These metrics are exposed internally through a metrics endpoint that refers to the `/metrics` HTTP API. Like other endpoints, this endpoint is exposed on the Amazon EKS control plane. This topic explains some of the ways you can use this endpoint to view and analyze what your cluster is doing.

Viewing the raw metrics

To view the raw metrics output, use `kubectl` with the `--raw` flag. This command allows you to pass any HTTP path and returns the raw response.

```
kubectl get --raw /metrics
```

Example output:

```
...
# HELP rest_client_requests_total Number of HTTP requests, partitioned by status code,
 method, and host.
# TYPE rest_client_requests_total counter
rest_client_requests_total{code="200",host="127.0.0.1:21362",method="POST"} 4994
rest_client_requests_total{code="200",host="127.0.0.1:443",method="DELETE"} 1
rest_client_requests_total{code="200",host="127.0.0.1:443",method="GET"} 1.326086e+06
rest_client_requests_total{code="200",host="127.0.0.1:443",method="PUT"} 862173
rest_client_requests_total{code="404",host="127.0.0.1:443",method="GET"} 2
rest_client_requests_total{code="409",host="127.0.0.1:443",method="POST"} 3
rest_client_requests_total{code="409",host="127.0.0.1:443",method="PUT"} 8
```

```
# HELP ssh_tunnel_open_count Counter of ssh tunnel total open attempts
# TYPE ssh_tunnel_open_count counter
ssh_tunnel_open_count 0
# HELP ssh_tunnel_open_fail_count Counter of ssh tunnel failed open attempts
# TYPE ssh_tunnel_open_fail_count counter
ssh_tunnel_open_fail_count 0
```

This raw output returns verbatim what the API server exposes. These metrics are represented in a Prometheus format. This format allows the API server to expose different metrics broken down by line. Each line includes a metric name, tags, and a value.

```
<metric_name>{"<tag>"="<value>"[<,...>]} <value>
```

While this endpoint is useful if you are looking for a specific metric, you typically want to analyze these metrics over time. To do this, you can deploy Prometheus into your cluster. Prometheus is a monitoring and time series database that scrapes exposed endpoints and aggregates data, allowing you to filter, graph, and query the results.

Deploying Prometheus

This topic helps you deploy Prometheus into your cluster with Helm V3. If you already have Helm installed, you can check your version with the `helm version` command. Helm is a package manager for Kubernetes clusters. For more information about Helm and how to install it, see Using Helm with Amazon EKS (p. 317).

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:

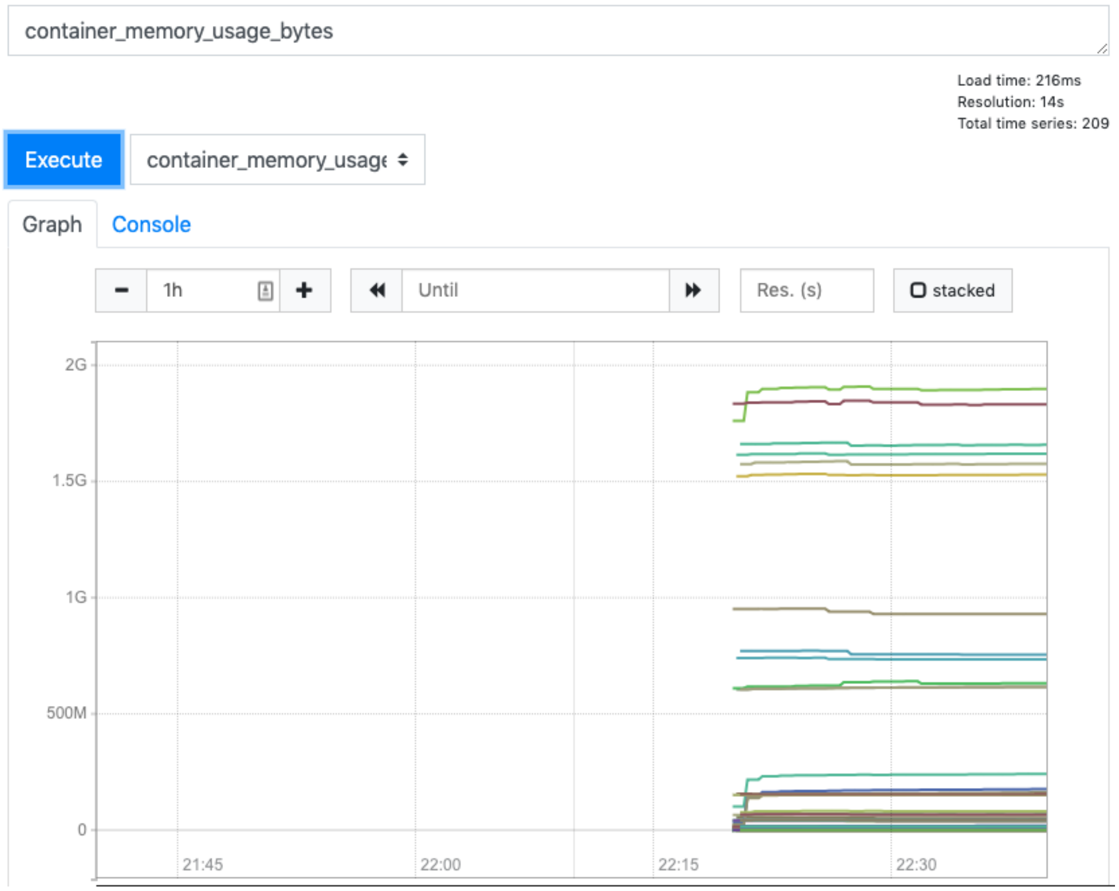
```
NAME                                             READY   STATUS    RESTARTS   AGE
prometheus-alertmanager-59b4c8c744-r7bgp         1/2     Running   0          48s
prometheus-kube-state-metrics-7cfd87cf99-jkz2f   1/1     Running   0          48s
prometheus-node-exporter-jcjqz                   1/1     Running   0          48s
prometheus-node-exporter-jxv2h                   1/1     Running   0          48s
prometheus-node-exporter-vbdks                   1/1     Running   0          48s
prometheus-pushgateway-76c444b68c-82tnw          1/1     Running   0          48s
prometheus-server-775957f748-mmht9               1/2     Running   0          48s
```

5. Use `kubectl` to port forward the Prometheus console to your local machine.

```
kubectl --namespace=prometheus port-forward deploy/prometheus-server 9090
```

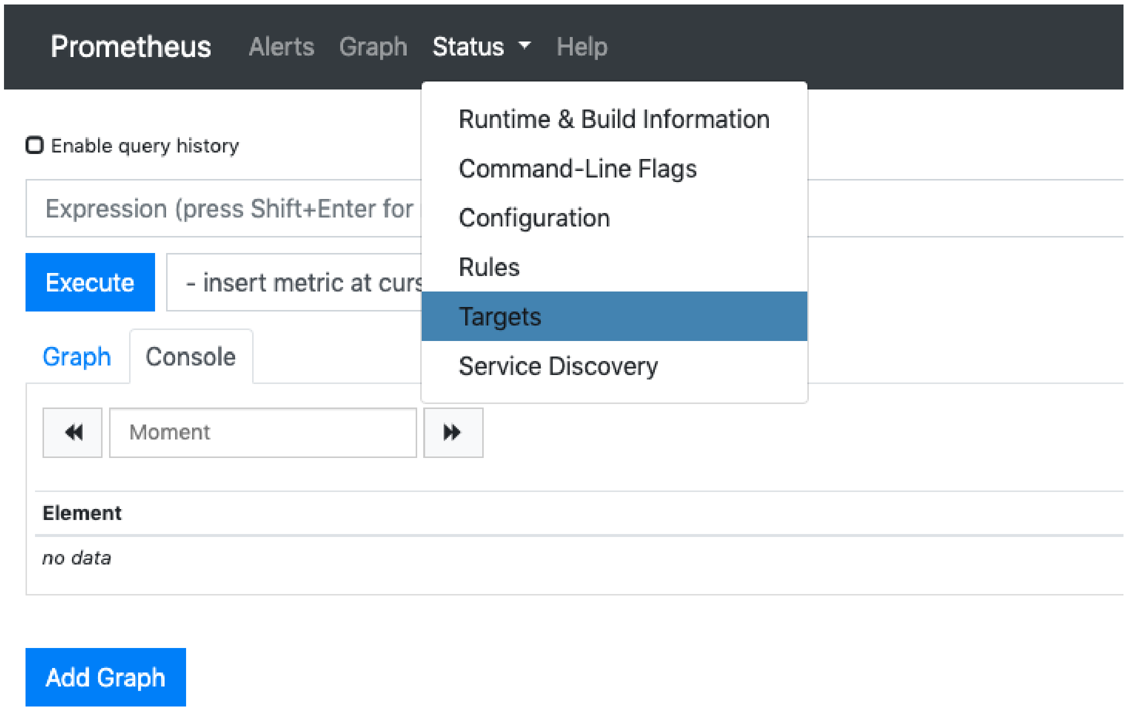
6. Point a web browser to localhost:9090 to view the Prometheus console.

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.


container_memory_usage_bytes

Load time: 216ms
Resolution: 14s
Total time series: 209

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.

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. 295) before proceeding. If the following command succeeds for your cluster, you're properly configured.

```
kubectl get svc
```

To install the Helm binaries on your local system

1. Run the appropriate command for your client operating system.

   - If you're using macOS with Homebrew, install the binaries with the following command.

   ```
   brew install helm
   ```

   - If you're using Windows with Chocolatey, install the binaries with the following command.

   ```
   choco install kubernetes-helm
   ```

   - If you're using Linux, install the binaries with the following commands.

```
curl https://raw.githubusercontent.com/helm/helm/master/scripts/get-helm-3 >
 get_helm.sh
chmod 700 get_helm.sh
./get_helm.sh
```

2. To pick up the new binary in your `PATH`, Close your current terminal window and open a new one.
3. Confirm that Helm is running with the following command.

```
helm help
```

4. At this point, you can run any Helm commands (such as `helm install <chart_name>`) to install, modify, delete, or query Helm charts in your cluster. If you're new to Helm and don't have a specific chart to install, you can:

   - Experiment by installing an example chart. See Install an example chart in the Helm Quickstart guide.
   - 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.

Contents

Tag basics

A tag is a label that you assign to an AWS resource. Each tag consists of a *key* and an optional *value*, both of which you define.

Tags enable you to categorize your AWS resources by, for example, purpose, owner, or environment. When you have many resources of the same type, you can quickly identify a specific resource based on the tags you've assigned to it. For example, you can define a set of tags for your Amazon EKS clusters to help you track each cluster's owner and stack level. We recommend that you devise a consistent set of tag keys for each resource type. You can then search and filter the resources based on the tags that you add.

Tags are not automatically assigned to your resources. After you add a tag, you can edit tag keys and values or remove tags from a resource at any time. If you delete a resource, any tags for the resource are also deleted.

Tags don't have any semantic meaning to Amazon EKS and are interpreted strictly as a string of characters. You can set the value of a tag to an empty string, but you can't set the value of a tag to null. If you add a tag that has the same key as an existing tag on that resource, the new value overwrites the 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 `eksctl`)
Amazon EKS clusters	Yes	No. Cluster tags do not propagate to any other resources associated with the cluster.	Yes
Amazon EKS managed node groups	Yes	No. Managed node group tags do not propagate to any other resources associated with the node group.	Yes
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 **Clusters** from the navigation panel, 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. 17).

Adding and deleting tags on an individual resource

Amazon EKS allows you to add or delete tags associated with your clusters directly from the resource's page.

To add or delete a tag on an individual resource

1. Open the Amazon EKS console at https://console.aws.amazon.com/eks/home#/clusters.
2. From the navigation bar, select the Region to use.
3. In the navigation panel, choose **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.	`tag-resource`	`Add-EKSResourceTag`	`TagResource`
Delete one or more tags.	`untag-resource`	`Remove-EKSResourceTag`	`UntagResource`

The following examples show how to tag or untag resources using the AWS CLI.

Example 1: Tag an existing cluster

The following command tags an existing cluster.

```
aws eks tag-resource --resource-arn <resource_ARN> --tags <team>=<devs>
```

Example 2: Untag an existing cluster

The following command deletes a tag from an existing cluster.

```
aws eks untag-resource --resource-arn <resource_ARN> --tag-keys <tag_key>
```

Example 3: List tags for a resource

The following command lists the tags associated with an existing resource.

```
aws eks list-tags-for-resource --resource-arn <resource_ARN>
```

Some resource-creating actions enable you to specify tags when you create the resource. The following actions support tagging when creating a resource.

Task	AWS CLI	AWS Tools for Windows PowerShell	API action	`eksctl`
Create a cluster	`create-cluster`	`New-EKSCluster`	`CreateCluster`	`create cluster`
Create a managed node group*	`create-nodegroup`	`New-EKSNodegroup`	`CreateNodegroup`	`create nodegroup`
Create a Fargate profile	`create-fargate-profile`	`New-EKSFargateProfile`	`CreateFargateProfile.html`	`create fargateprofile`

*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 the section called “Tagging Amazon EC2 instances” (p. 101). 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.

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 navigation panel, 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*.

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 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.

The following tables provide the default quotas (also referred to as limits) for Amazon EKS and AWS Fargate for an AWS account.

Amazon EKS service quotas

The following quotas are Amazon EKS service quotas. Most of these service quotas are listed under the Amazon Elastic Kubernetes Service (Amazon EKS) 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
Clusters	The maximum number of EKS clusters in this account in the current Region.	100	Yes
Control plane security groups per cluster	The maximum number of control plane security groups per cluster (these are specified when you create the cluster).	4	No
Managed node groups per cluster	The maximum number of managed node groups per cluster.	30	Yes
Nodes per managed node group	The maximum number of nodes per managed node group.	450	Yes
Public endpoint access CIDR ranges per cluster	The maximum number of public endpoint access CIDR ranges per cluster (these are specified when you create or update the cluster).	40	No
Fargate profiles per cluster	The maximum number Fargate profiles per cluster.	10	Yes
Selectors per Fargate profile	The maximum number selectors per Fargate profile	5	Yes
Label pairs per Fargate profile selector	The maximum number of label pairs per Fargate profile selector	5	Yes

AWS Fargate service quotas

The following quota is an Amazon EKS on AWS Fargate service quota. The service quota is 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 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

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. 356).

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. 328).

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. 331).

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. Service roles provide access only within your account and cannot be used to grant access to services in other accounts. 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

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 *IAM User Guide*.

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>:123456789012:cluster/dev"
```

To specify all clusters that belong to a specific account and Region, use the wildcard (*):

```
"Resource": "arn:aws:eks:<region-code>:123456789012:cluster/*"
```

Some Amazon EKS actions, such as those for creating resources, cannot be performed on a specific resource. In those cases, you must use the wildcard (*).

```
"Resource": "*"
```

To see a list of Amazon EKS resource types and their ARNs, see Resources Defined by Amazon Elastic Kubernetes Service in the *IAM User Guide*. To learn with which actions you can specify the ARN of each resource, see Actions Defined by Amazon Elastic Kubernetes Service.

Condition keys

Amazon EKS 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 the section called “Example IAM policy” (p. 294).

All Amazon EC2 actions support the `aws:RequestedRegion` and `ec2:Region` condition keys. For more information, see Example: Restricting Access to a Specific Region.

For a list of Amazon EKS condition keys, see Condition Keys for Amazon Elastic Kubernetes Service in the *IAM User Guide*. 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. 331).

When you create an Amazon EKS cluster, the IAM entity user or role, such as a federated user that creates the cluster, is automatically granted `system:masters` permissions in the cluster's RBAC configuration in the control plane. This IAM entity does not appear in the ConfigMap, or any other 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.

For additional information about working with the ConfigMap, see Managing users or IAM roles for your cluster (p. 288).

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. 318).

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. 335).

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. 339) and Amazon EKS node IAM role (p. 341).

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. 339) and Check for an existing node role (p. 342).

Amazon EKS identity-based policy examples

By default, IAM users and roles don't have permission to create or modify Amazon EKS resources. They also can't perform tasks using the AWS Management Console, AWS CLI, or AWS API. An IAM administrator must create IAM policies that grant users and roles permission to perform specific API operations on the specified resources they need. The administrator must then attach those policies to the IAM users or groups that require those permissions.

To learn how to create an IAM identity-based policy using these example JSON policy documents, see Creating policies on the JSON tab in the *IAM User Guide*.

When you create an Amazon EKS cluster, the IAM entity user or role, such as a federated user that creates the cluster, is automatically granted `system:masters` permissions in the cluster's RBAC configuration in the control plane. This IAM entity does not appear in the ConfigMap, or any other 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.

For additional information about working with the ConfigMap, see Managing users or IAM roles for your cluster (p. 288).

Topics

Policy best practices

Identity-based policies are very powerful. They determine whether someone can create, access, or delete Amazon EKS resources in your account. These actions can incur costs for your AWS account. When you create or edit identity-based policies, follow these guidelines and recommendations:

- **Get started using AWS managed policies** – To start using Amazon EKS quickly, use AWS managed policies to give your employees the permissions they need. These policies are already available in your account and are maintained and updated by AWS. For more information, see Get started using permissions with AWS managed policies in the *IAM User Guide*.
- **Grant least privilege** – When you create custom policies, grant only the permissions required to perform a task. Start with a minimum set of permissions and grant additional permissions as necessary. Doing so is more secure than starting with permissions that are too lenient and then trying to tighten them later. For more information, see Grant least privilege in the *IAM User Guide*.
- **Enable MFA for sensitive operations** – For extra security, require IAM users to use multi-factor authentication (MFA) to access sensitive resources or API operations. For more information, see Using multi-factor authentication (MFA) in AWS in the *IAM User Guide*.
- **Use policy conditions for extra security** – To the extent that it's practical, define the conditions under which your identity-based policies allow access to a resource. For example, you can write conditions to specify a range of allowable IP addresses that a request must come from. You can also write conditions to allow requests only within a specified date or time range, or to require the use of SSL or MFA. For more information, see IAM JSON policy elements: Condition in the *IAM User Guide*.

Using the Amazon EKS console

To access the Amazon EKS console, you must have a minimum set of permissions. These permissions must allow you to list and view details about the Amazon EKS resources in your AWS account. If you create an identity-based policy that is more restrictive than the minimum required permissions, the console won't function as intended for entities (IAM users or roles) with that policy.

Important
If you see an **Error loading Namespaces** error in the console, or don't see anything on the **Overview** or **Workloads** tabs, see Can't see workloads or nodes and receive an error in the AWS

Management Console (p. 381) to resolve the issue. If you don't resolve the issue, you can still view and manage aspects of your Amazon EKS cluster on the **Configuration** tab.

To ensure that those entities can still use the Amazon EKS console, create a policy with your own unique name, such as `AmazonEKSAdminPolicy`. Attach the policy to the entities. For more information, see Adding permissions to a user in the *IAM User Guide*.

Important
The following example policy will allow you to view information on the **Configuration** tab in the console. To view information on the **Nodes** and **Workloads** in the AWS Management Console, you need additional IAM permissions, as well as Kubernetes permissions. For more information, see the the section called "View nodes and workloads for all clusters in the AWS Management Console" (p. 333) example policy.

```
{
    "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.

View nodes and workloads for all clusters in the AWS Management Console

This example shows how you can create a policy that allows a user to the section called "View nodes" (p. 86) and the section called "View workloads" (p. 253) for all clusters.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "eks:DescribeNodegroup",
                "eks:ListNodegroups",
                "eks:DescribeCluster",
                "eks:ListClusters",
                "eks:AccessKubernetesApi",
                "ssm:GetParameter",
                "eks:ListUpdates",
                "eks:ListFargateProfiles"
            ],
            "Resource": "*"
        }
    ]
}
```

Important
The policy must be attached to a user or role that is mapped to a Kubernetes user or group in the `aws-auth` configmap. The user or group must be a `subject` in a `rolebinding` or `clusterrolebinding` that is bound to a Kubernetes `role` or `clusterrole` that has the necessary permissions to view the Kubernetes resources. For more information about adding IAM users or roles to the `aws-auth` configmap, see the section called “Managing users or IAM roles for your cluster” (p. 288). To create roles and role bindings, see Using RBAC Authorization in the Kubernetes documentation. You can download the following example manifests that create a `clusterrole` and `clusterrolebinding` or a `role` and `rolebinding`:

- **View Kubernetes resources in all namespaces** – The group name in the file is `eks-console-dashboard-full-access-group`, which is the group that your IAM user or role needs to be mapped to in the `aws-auth` configmap. You can change the name of the group before applying it to your cluster, if desired, and then map your IAM user or role to that group in the configmap. To download the file, select the appropriate link for the Region that your cluster is in.
  - All Regions other than Beijing and Ningxia China
  - Beijing and Ningxia China Regions
- **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. The group name in the file is `eks-console-dashboard-restricted-access-group`, which is the group that your IAM user or role needs to be mapped to in the `aws-auth` configmap. You can change the name of the group before applying it to your cluster, if desired, and then map your IAM user or role to that group in the configmap. To download the file, select the appropriate link for the Region that your cluster is in.
  - All Regions other than Beijing and Ningxia China
  - Beijing and Ningxia China Regions

Allow users to view their own permissions

This example shows how you might create a policy that allows IAM users to view the inline and managed policies that are attached to their user identity. This policy includes permissions to complete this action on the console or programmatically using the AWS CLI or AWS API.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "ViewOwnUserInfo",
            "Effect": "Allow",
            "Action": [
                "iam:GetUserPolicy",
                "iam:ListGroupsForUser",
                "iam:ListAttachedUserPolicies",
                "iam:ListUserPolicies",
                "iam:GetUser"
            ],
            "Resource": ["arn:aws:iam::*:user/${aws:username}"]
        },
        {
            "Sid": "NavigateInConsole",
            "Effect": "Allow",
            "Action": [
                "iam:GetGroupPolicy",
                "iam:GetPolicyVersion",
                "iam:GetPolicy",
                "iam:ListAttachedGroupPolicies",
                "iam:ListGroupPolicies",
                "iam:ListPolicyVersions",
                "iam:ListPolicies",
                "iam:ListUsers"
            ],
            "Resource": "*"
        }
    ]
}
```

Update a Kubernetes cluster

This example shows how you can create a policy that allows a user to update the Kubernetes version of any *dev* cluster for an account, in any region.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "eks:UpdateClusterVersion",
            "Resource": "arn:aws:eks:*:<111122223333>:cluster/<dev>"
        }
    ]
}
```

List or describe all clusters

This example shows how you can create a policy that allows a user read-only access to list or describe all clusters. An account must be able to list and describe clusters to use the `update-kubeconfig` AWS CLI command.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "eks:DescribeCluster",
                "eks:ListClusters"
            ],
            "Resource": "*"
        }
    ]
}
```

Using Service-Linked Roles for Amazon EKS

Amazon Elastic Kubernetes Service uses AWS Identity and Access Management (IAM) service-linked roles. A service-linked role is a unique type of IAM role that is linked directly to Amazon EKS. Service-linked roles are predefined by Amazon EKS and include all the permissions that the service requires to call other AWS services on your behalf.

Topics

Using Roles for Amazon EKS

Amazon Elastic Kubernetes Service uses AWS Identity and Access Management (IAM) service-linked roles. A service-linked role is a unique type of IAM role that is linked directly to Amazon EKS. Service-linked roles are predefined by Amazon EKS and include all the permissions that the service requires to call other AWS services on your behalf.

A service-linked role makes setting up Amazon EKS easier because you don't have to manually add the necessary permissions. Amazon EKS defines the permissions of its service-linked roles, and unless defined otherwise, only Amazon EKS can assume its roles. The defined permissions include the trust policy and the permissions policy, and that permissions policy cannot be attached to any other IAM entity.

You can delete a service-linked role only after first deleting their related resources. This protects your Amazon EKS resources because you can't inadvertently remove permission to access the resources.

For information about other services that support service-linked roles, see AWS Services That Work with IAM and look for the services that have **Yes** in the **Service-Linked Role** column. Choose a **Yes** with a link to view the service-linked role documentation for that service.

Service-Linked Role Permissions for Amazon EKS

Amazon EKS uses the service-linked role named **AWSServiceRoleForAmazonEKS** – These permissions are required for Amazon EKS to manage clusters in your account. These policies are related to management of the following resources: network interfaces, security groups, logs, and VPCs.

> **Note**
> The **AWSServiceRoleForAmazonEKS** service-linked role is distinct from the role required for cluster creation. For more information, see Amazon EKS cluster IAM role (p. 339).

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

1. Open the Amazon EKS console at https://console.aws.amazon.com/eks/home#/clusters.
2. Choose a cluster.
3. On the cluster page, if there are any managed node groups in the **Node Groups** section, select each one individually, and then choose **Delete**.
4. Type the name of the node group in the deletion confirmation window, and then choose **Confirm** to delete.
5. Repeat this procedure for any other node groups in the cluster. Wait for all of the delete operations to finish.
6. On the cluster page choose **Delete**.
7. Repeat this procedure for any other clusters in your account.

Manually Delete the Service-Linked Role

Use the IAM console, the AWS CLI, or the AWS API to delete the AWSServiceRoleForAmazonEKS service-linked role. For more information, see Deleting a Service-Linked Role in the *IAM User Guide*.

Supported Regions for Amazon EKS Service-Linked Roles

Amazon EKS supports using service-linked roles in all of the regions where the service is available. For more information, see Amazon EKS Service Endpoints and Quotas.

Using Roles for Amazon EKS 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** – These permissions are required for managing nodegroups in your account. These policies are related to management of the following resources: Auto Scaling groups, security groups, launch templates and IAM instance profiles..

The AWSServiceRoleForAmazonEKSNodegroup service-linked role trusts the following services to assume the role:

- `eks-nodegroup.amazonaws.com`

The role permissions policy allows Amazon EKS to complete the following actions on the specified resources:

- 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

1. Open the Amazon EKS console at https://console.aws.amazon.com/eks/home#/clusters.
2. Choose a cluster.
3. On the cluster page, if there are any managed node groups in the **Node Groups** section, select each one individually, and then choose **Delete**.
4. Type the name of the cluster in the deletion confirmation window, and then choose **Confirm** to delete.
5. Repeat this procedure for any other node groups in the cluster and for any other clusters in your account.

Manually Delete the Service-Linked Role

Use the IAM console, the AWS CLI, or the AWS API to delete the AWSServiceRoleForAmazonEKSNodegroup service-linked role. For more information, see Deleting a Service-Linked Role in the *IAM User Guide*.

Supported Regions for Amazon EKS Service-Linked Roles

Amazon EKS supports using service-linked roles in all of the regions where the service is available. For more information, see Amazon EKS Service Endpoints and Quotas.

Amazon EKS cluster IAM role

Kubernetes clusters managed by Amazon EKS make calls to other AWS services on your behalf to manage the resources that you use with the service. Before you can create Amazon EKS clusters, you must create an IAM role with the following IAM policies:

- `AmazonEKSClusterPolicy`

> **Note**
> Prior to April 16, 2020, AmazonEKSServicePolicy was also required and the suggested name was `eksServiceRole`. With the `AWSServiceRoleForAmazonEKS` service-linked role, that policy is no longer required 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 navigation panel, choose **Roles**.
3. Search the list of roles for `eksClusterRole`. If a role that includes `eksClusterRole` does not exist, then see Creating the Amazon EKS cluster role (p. 340) to create the role. If a role that includes `eksClusterRole` does exist, then select the role to view the attached policies.
4. Choose **Permissions**.
5. Ensure that the **AmazonEKSClusterPolicy** managed policy is attached to the role. If the policy is attached, your Amazon EKS cluster role is properly configured.
6. Choose **Trust Relationships**, **Edit Trust Relationship**.
7. Verify that the trust relationship contains the following policy. If the trust relationship matches the policy below, choose **Cancel**. If the trust relationship does not match, copy the policy into the **Policy Document** window and choose **Update Trust Policy**.

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

Creating the Amazon EKS cluster role

You can use the AWS Management Console or AWS CloudFormation to create the cluster role. Select the tab with the name of the tool that you want to use to create the role.

AWS Management Console

To create your Amazon EKS cluster role in the IAM console

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

AWS CloudFormation

[To create your Amazon EKS cluster role with AWS CloudFormation]

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

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


Resources:

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

Outputs:

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

Note

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

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

Amazon EKS 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`

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 navigation panel, choose **Roles**.
3. Search the list of roles for `eksNodeRole`. If a role that contains `eksNodeRole` or `NodeInstanceRole` does not exist, then see Creating the Amazon EKS node IAM role (p. 342) to create the role. If a role that contains `eksNodeRole` 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 the section called "Configure plugin for IAM account" (p. 217).
6. Choose **Trust Relationships**, **Edit Trust Relationship**.
7. Verify that the trust relationship contains the following policy. If the trust relationship matches the policy below, choose **Cancel**. If the trust relationship does not match, copy the policy into the **Policy Document** window and choose **Update Trust Policy**.

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

Creating the Amazon EKS node IAM role

You can create the node IAM role with the AWS Management Console or AWS CloudFormation. 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. Choose **Roles**, then **Create role**.
3. Choose **EC2** from the list of **Common use cases** under **Choose a use case,** then choose **Next: Permissions**.

4. In the **Filter policies** box, enter `AmazonEKSWorkerNodePolicy`. Check the box to the left of **AmazonEKSWorkerNodePolicy**.
5. In the **Filter policies** box, enter `AmazonEC2ContainerRegistryReadOnly`. Check the box to the left of **AmazonEC2ContainerRegistryReadOnly**.
6. The **AmazonEKS_CNI_Policy** policy must be attached to either this role or to a different role that is mapped to the `aws-node` Kubernetes service account. We recommend assigning the policy to the role associated to the Kubernetes service account instead of assigning it to this role. For more information, see Configuring the VPC CNI plugin to use IAM roles for service accounts (p. 217).
7. Choose **Next: Tags**.
8. (Optional) Add metadata to the role by attaching tags as key–value pairs. For more information about using tags in IAM, see Tagging IAM Entities in the *IAM User Guide*.
9. Choose **Next: Review**.
10. For **Role name**, enter a unique name for your role, such as NodeInstanceRole. For **Role description**, replace the current text with descriptive text such as Amazon EKS - Node Group Role, then choose **Create role**.

AWS CloudFormation

To create your Amazon EKS node role using AWS CloudFormation

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

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

5. On the **Specify stack details** page, for **Stack name** enter a name such as **eks-node-group-instance-role** and choose **Next**.
6. (Optional) On the **Configure stack options** page, you can choose to tag your stack resources. Choose **Next**.
7. On the **Review** page, check the box in the **Capabilities** section and choose **Create stack**.
8. When your stack is created, select it in the console and choose **Outputs**.
9. Record the **NodeInstanceRole** value for the IAM role that was created. You need this when you create your node group.
10. (Optional, but recommended) One of the IAM policies attached to the role by the AWS CloudFormation template in a previous step is the **AmazonEKS_CNI_Policy** managed policy. The policy must be attached to this role or to a role associated to the Kubernetes `aws-node` service account that is used for the Amazon EKS VPC CNI plugin. We recommend assigning the policy to the role associated to the Kubernetes service account. For more information, see Configuring the VPC CNI plugin to use IAM roles for service accounts (p. 217).

Pod execution role

The Amazon EKS pod execution role is required to run pods on AWS Fargate infrastructure.

When your cluster creates pods on AWS Fargate infrastructure, the components running on the Fargate infrastructure need to make calls to AWS APIs on your behalf to do things like 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, so that the `kubelet` that is running on the Fargate infrastructure can register with your Amazon EKS cluster. This is what allows Fargate infrastructure to appear in your cluster as nodes.

The containers running in the Fargate pod cannot assume the IAM permissions associated with the pod execution role. To give the containers in your Fargate pod permissions to access other AWS services, you must use the section called "IAM roles for service accounts" (p. 345).

Before you create a Fargate profile, you must create an IAM role with the following IAM policy:

- `AmazonEKSFargatePodExecutionRolePolicy`

Check for an existing pod execution role

You can use the following procedure to check and see if your account already has the Amazon EKS pod execution role.

To check for the `AmazonEKSFargatePodExecutionRole` in the IAM console

1. Open the IAM console at https://console.aws.amazon.com/iam/.
2. In the navigation panel, choose **Roles**.
3. Search the list of roles for `AmazonEKSFargatePodExecutionRole`. If the role does not exist, see Creating the Amazon EKS pod execution role (p. 344) to create the role. If the role does exist, select the role to view the attached policies.
4. Choose **Permissions**.
5. Ensure that the **AmazonEKSFargatePodExecutionRolePolicy** Amazon managed policy is attached to the role. If the policy is attached, then your Amazon EKS pod execution role is properly configured.
6. Choose **Trust Relationships**, **Edit Trust Relationship**.
7. Verify that the trust relationship contains the following policy. If the trust relationship matches the policy below, choose **Cancel**. If the trust relationship does not match, copy the policy into the **Policy Document** window and choose **Update Trust Policy**.

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

Creating the Amazon EKS pod execution role

You can use the following procedure to create the Amazon EKS pod execution role if you do not already have one for your account.

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

IAM roles for service accounts

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 the section called "Node IAM role" (p. 341) 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 the section called "Node IAM role" (p. 341), unless you block pod access to the IMDS. For more information, see the section called "Restricting access to the instance profile" (p. 359).

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. 349)** – 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. 350)** – We recommend creating separate roles for each unique collection of permissions that pods need.
3. **Associate an IAM role with a service account (p. 354)** – Complete this task for each Kubernetes service account that needs access to AWS resources.

Technical overview

In 2014, AWS Identity and Access Management added support for federated identities using OpenID Connect (OIDC). This feature allows you to authenticate AWS API calls with supported identity providers and receive a valid OIDC JSON web token (JWT). You can pass this token to the AWS STS `AssumeRoleWithWebIdentity` API operation and receive IAM temporary role credentials. You can use these credentials to interact with any AWS service, like Amazon S3 and DynamoDB.

Kubernetes has long used service accounts as its own internal identity system. Pods can authenticate with the Kubernetes API server using an auto-mounted token (which was a non-OIDC JWT) that only the Kubernetes API server could validate. These legacy service account tokens do not expire, and rotating the signing key is a difficult process. In Kubernetes version 1.12, support was added for a new `ProjectedServiceAccountToken` feature, which is an OIDC JSON web token that also contains the service account identity, and supports a configurable audience.

Amazon EKS now hosts a public OIDC discovery endpoint per cluster containing the signing keys for the `ProjectedServiceAccountToken` JSON web tokens so external systems, like IAM, can validate and accept the OIDC tokens issued by Kubernetes.

IAM role configuration

In IAM, you create an IAM role with a trust relationship that is scoped to your cluster's OIDC provider, the service account namespace, and (optionally) the service account name, and then attach the IAM policy that you want to associate with the service account. You can add multiple entries in the `StringEquals` and `StringLike` conditions below to use multiple service accounts or namespaces with the role.

- To scope a role to a specific service account:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Federated": "arn:aws:iam::<AWS_ACCOUNT_ID>:oidc-provider/<OIDC_PROVIDER>"
      },
      "Action": "sts:AssumeRoleWithWebIdentity",
      "Condition": {
        "StringEquals": {
          "<OIDC_PROVIDER>:sub":
 "system:serviceaccount:<SERVICE_ACCOUNT_NAMESPACE>:<SERVICE_ACCOUNT_NAME>"
        }
      }
    }
  ]
}
```

- To scope a role to an entire namespace (to use the namespace as a boundary):

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Federated": "arn:aws:iam::<AWS_ACCOUNT_ID>:oidc-provider/<OIDC_PROVIDER>"
      },
      "Action": "sts:AssumeRoleWithWebIdentity",
      "Condition": {
        "StringLike": {
          "<OIDC_PROVIDER>:sub": "system:serviceaccount:<SERVICE_ACCOUNT_NAMESPACE>:*"
          }
        }
      }
    ]
}
```

Service account configuration

In Kubernetes, you define the IAM role to associate with a service account in your cluster by adding the `eks.amazonaws.com/role-arn` annotation to the service account.

```
apiVersion: v1
kind: ServiceAccount
metadata:
  annotations:
    eks.amazonaws.com/role-arn: arn:aws:iam::<AWS_ACCOUNT_ID>:role/<IAM_ROLE_NAME>
```

Pod configuration

The Amazon EKS Pod Identity Webhook on the cluster watches for pods that are associated with service accounts with this annotation and applies the following environment variables to them.

```
AWS_ROLE_ARN=arn:aws:iam::<AWS_ACCOUNT_ID>:role/<IAM_ROLE_NAME>
AWS_WEB_IDENTITY_TOKEN_FILE=/var/run/secrets/eks.amazonaws.com/serviceaccount/token
```

Note
Your cluster does not need to use the mutating web hook to configure the environment variables and token file mounts; you can choose to configure pods to add these environment variables manually.

Supported versions of the AWS SDK (p. 355) 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 the section called "Node IAM role" (p. 341) too, unless you restrict access to those credentials. For more information, see the section called "Restricting access to the instance profile" (p. 359).

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. 349) and Creating an IAM role and policy for your service account (p. 350) using the OIDC issuer URL from Account A's cluster. Then a cluster administrator annotates the service account in Account A's cluster to use the role from Account B.

```
apiVersion: v1
kind: ServiceAccount
metadata:
  annotations:
    eks.amazonaws.com/role-arn:
 arn:aws:iam::<ACCOUNT_B_AWS_ACCOUNT_ID>:role/<IAM_ROLE_NAME>
```

Example : Use chained `AssumeRole` operations

Example

In this example, Account B creates an IAM policy with the permissions to give to pods in Account A's cluster. Account B attaches that policy to an IAM role with a trust relationship that allows `AssumeRole` permissions to Account A (*111111111111*), as shown below.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "AWS": "arn:aws:iam::111111111111:root"
      },
      "Action": "sts:AssumeRole",
      "Condition": {}
    }
  ]
}
```

Account A creates a role with a trust policy that gets credentials from the identity provider created with the cluster's OIDC issuer URL, as shown below.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Federated": "arn:aws:iam::111111111111:oidc-provider/oidc.eks.<region-
code>.amazonaws.com/id/EXAMPLEC061A78C479E31025A21AC4CDE191335D05820BE5CE"
      },
      "Action": "sts:AssumeRoleWithWebIdentity"
    }
  ]
}
```

Account A attaches a policy to that role with the following permissions to assume the role that Account B created.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "sts:AssumeRole",
            "Resource": "arn:aws:iam::222222222222:role/account-b-role"
        }
    ]
}
```

The application code for pods to assume Account B's role uses two profiles: `account_b_role` and `account_a_role`. The `account_b_role` profile uses the `account_a_role` profile as its source. For the AWS CLI, the `~/.aws/config` file would look like the following example.

```
[profile account_b_role]
source_profile = account_a_role
role_arn=arn:aws:iam::222222222222:role/account-b-role

[profile account_a_role]
web_identity_token_file = /var/run/secrets/eks.amazonaws.com/serviceaccount/token
role_arn=arn:aws:iam::111111111111:role/account-a-role
```

To specify chained profiles for other AWS SDKs, consult their documentation.

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.

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 <cluster_name> --query "cluster.identity.oidc.issuer"
 --output text
```

Example output:

```
https://oidc.eks.us-west-2.amazonaws.com/id/EXAMPLED539D4633E53DE1B716D3041E
```

List the IAM OIDC providers in your account. Replace `<EXAMPLED539D4633E53DE1B716D3041E>` (including `<>`) with the value returned from the previous command.

```
aws iam list-open-id-connect-providers | grep <EXAMPLED539D4633E53DE1B716D3041E>
```

Example output

```
"Arn": "arn:aws:iam::111122223333:oidc-provider/oidc.eks.us-west-2.amazonaws.com/
id/EXAMPLED539D4633E53DE1B716D3041E"
```

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 `<cluster_name>` (including `<>`) with your own value.

```
eksctl utils associate-iam-oidc-provider --cluster <cluster_name> --approve
```

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 navigation panel, choose **Identity Providers**. 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 navigation panel, 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 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. 349).
- 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. 351).

You can create the IAM role with `eksctl`, 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.

eksctl

Create the service account and IAM role with the following command. Replace the *`<example values>`* (including *`<>`*) with your own values.

```
eksctl create iamserviceaccount \
    --name <service_account_name> \
    --namespace <service_account_namespace> \
    --cluster <cluster_name> \
    --attach-policy-arn <IAM_policy_ARN> \
    --approve \
    --override-existing-serviceaccounts
```

An AWS CloudFormation template is deployed that creates an IAM role and attaches the IAM policy to it. The role is associated with a Kubernetes service account. If your cluster didn't have an existing IAM OIDC provider, one was created. If the service account doesn't exist, it is created in the namespace that you provided. If the service account does exist, then it is annotated with `eks.amazonaws.com/role-arn: arn:aws:iam::<your-account-id>:role/<iam-role-name-that-was-created>`.

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 navigation panel, choose **Roles**, **Create Role**.
6. In the **Select type of trusted entity** section, choose **Web identity**.
7. In the **Choose a web identity provider** section:
   1. For **Identity provider**, choose the URL for your cluster.
   2. For **Audience**, choose `sts.amazonaws.com`.
8. Choose **Next: Permissions**.
9. In the **Attach Policy** section, select the IAM policy that has the permissions that you want your service account to use.
10. Choose **Next: Tags**.

11. On the **Add tags (optional)** screen, you can add tags for the account. Choose **Next: Review**.
12. For **Role Name**, enter a name for your role and then choose **Create Role**.
13. After the role is created, choose the role in the console to open it for editing.
14. Choose the **Trust relationships** tab, and then choose **Edit trust relationship**.
15. Find the line that looks similar to the following:

    ```
    "oidc.eks.us-west-2.amazonaws.com/id/EXAMPLED539D4633E53DE1B716D3041E:aud":
     "sts.amazonaws.com"
    ```

    Change the line to look like the following line. Replace `<EXAMPLED539D4633E53DE1B716D3041E>` (including `<>`)with your cluster's OIDC provider ID and replace <region-code> with the Region code that your cluster is in.

    ```
    "oidc.eks.<region-code>.amazonaws.com/id/<EXAMPLED539D4633E53DE1B716D3041E>:sub":
     "system:serviceaccount:<SERVICE_ACCOUNT_NAMESPACE>:<SERVICE_ACCOUNT_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. 214) is deployed, for example, the deployment manifest creates a service account, cluster role, and cluster role binding. You can view the manifest on GitHub.
16. Choose **Update Trust Policy** to finish.

AWS CLI

1. Set your AWS account ID to an environment variable with the following command.

    ```
    AWS_ACCOUNT_ID=$(aws sts get-caller-identity --query "Account" --output text)
    ```
2. Set your OIDC identity provider to an environment variable with the following command. Replace the `<example values>` (including `<>`) with your own values.

    Important
    You must use at least version 1.19.7 or 2.1.26 of the AWS CLI to receive the proper output from this command. For more information, see Installing the AWS CLI in the *AWS Command Line Interface User Guide*.

    ```
    OIDC_PROVIDER=$(aws eks describe-cluster --name <cluster-name> --query
     "cluster.identity.oidc.issuer" --output text | sed -e "s/^https:\/\///")
    ```
3. Copy the following code block to your computer and replace the `<example values>` (including `<>`) with your own values.

    ```
    read -r -d '' TRUST_RELATIONSHIP <<EOF
    {
      "Version": "2012-10-17",
      "Statement": [
        {
          "Effect": "Allow",
          "Principal": {
            "Federated": "arn:aws:iam::${AWS_ACCOUNT_ID}:oidc-provider/
    ${OIDC_PROVIDER}"
    ```

```
      },
      "Action": "sts:AssumeRoleWithWebIdentity",
      "Condition": {
        "StringEquals": {
          "${OIDC_PROVIDER}:sub": "system:serviceaccount:<my-namespace>:<my-
service-account>"
        }
      }
    }
  ]
}
EOF
echo "${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.

```
aws iam create-role --role-name <IAM_ROLE_NAME> --assume-role-policy-document
 file://trust.json --description "<IAM_ROLE_DESCRIPTION>"
```

6. Run the following command to attach your IAM policy to your role.

```
aws iam attach-role-policy --role-name <IAM_ROLE_NAME> --policy-
arn=<IAM_POLICY_ARN>
```

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.

Note
If you created an IAM role to use with your service account using `eksctl`, this has already been done for you with the service account that you specified when creating the role.

```
apiVersion: v1
kind: ServiceAccount
metadata:
  annotations:
    eks.amazonaws.com/role-arn: arn:aws:iam::<AWS_ACCOUNT_ID>:role/<IAM_ROLE_NAME>
```

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. 349).
- An existing 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. 350).

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. Be sure to replace the `<example values>` (including <>) with your own.

```
kubectl annotate serviceaccount -n <SERVICE_ACCOUNT_NAMESPACE> <SERVICE_ACCOUNT_NAME> \
eks.amazonaws.com/role-arn=arn:aws:iam::<AWS_ACCOUNT_ID>:role/<IAM_ROLE_NAME>
```

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. 214) 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. The following command deletes the existing the `aws-node` DaemonSet pods and deploys them with the service account annotation. You can modify the namespace, deployment type, and label to update your specific pods.

```
kubectl delete pods -n <kube-system> -l <k8s-app=aws-node>
```

3. Confirm that the pods all restarted.

```
kubectl get pods -n <kube-system>  -l <k8s-app=aws-node>
```

4. Describe one of the pods and verify that the `AWS_WEB_IDENTITY_TOKEN_FILE` and `AWS_ROLE_ARN` environment variables exist.

```
kubectl exec -n kube-system <aws-node-9rgzw> env | grep AWS
```

Output:

```
AWS_VPC_K8S_CNI_LOGLEVEL=DEBUG
AWS_ROLE_ARN=arn:aws:iam::<AWS_ACCOUNT_ID>:role/<IAM_ROLE_NAME>
AWS_WEB_IDENTITY_TOKEN_FILE=/var/run/secrets/eks.amazonaws.com/serviceaccount/token
```

Using a supported AWS SDK

The containers in your pods must use an AWS SDK version that supports assuming an IAM role via an OIDC web identity token file. AWS SDKs that are included in Linux distribution package managers may not be new enough to support this feature. Be sure to use at least the minimum SDK versions listed below:

- Java (Version 2) — 2.10.11
- Java — 1.11.704
- Go — 1.23.13
- Python (Boto3) — 1.9.220
- Python (botocore) — 1.12.200
- AWS CLI — 1.16.232
- Node — 2.521.0
- Ruby — 2.11.345
- C++ — 1.7.174

- .NET — 3.3.659.1
- PHP — 3.110.7

Many popular Kubernetes add-ons, such as the Cluster Autoscaler and the the section called "AWS Load Balancer Controller" (p. 238), 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.

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. 381).

Logging and monitoring in Amazon EKS

Amazon EKS control plane logging provides audit and diagnostic logs directly from the Amazon EKS control plane to CloudWatch Logs in your account. These logs make it easy for you to secure and run your clusters. You can select the exact log types you need, and logs are sent as log streams to a group for each Amazon EKS cluster in CloudWatch. For more information, see Amazon EKS Control Plane Logging (p. 55).

Note
When you check the Amazon EKS authenticator logs in Amazon CloudWatch, you'll see entries that contain text similar to the following example text.

```
level=info msg="mapping IAM role" groups="[]"
 role="arn:aws:iam::<111122223333:>role/<XXXXXXXXXXXXXXXXXX>-NodeManagerRole-
<XXXXXXXX>" username="eks:node-manager"
```

Entries that contain this text are expected. The `username` is an Amazon EKS internal service role that performs specific operations for managed node groups and Fargate.

Amazon EKS is integrated with AWS CloudTrail, a service that provides a record of actions taken by a user, role, or an AWS service in Amazon EKS. CloudTrail captures all API calls for Amazon EKS as events. The calls captured include calls from the Amazon EKS console and code calls to the Amazon EKS API operations. For more information, see Logging Amazon EKS API calls with AWS CloudTrail (p. 369).

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. 314).

Compliance validation for Amazon EKS

Third-party auditors assess the security and compliance of Amazon EKS as part of multiple AWS compliance programs. These include SOC, PCI, ISO, HIPAA, and others.

For a list of AWS services in scope of specific compliance programs, see AWS services in scope by compliance program. For general information, see AWS compliance programs.

You can download third-party audit reports using AWS Artifact. For more information, see Downloading reports in AWS Artifact.

Your compliance responsibility when using Amazon EKS is determined by the sensitivity of your data, your company's compliance objectives, and applicable laws and regulations. AWS provides the following resources to help with compliance:

- Security and compliance quick start guides – These deployment guides discuss architectural considerations and provide steps for deploying security- and compliance-focused baseline environments on AWS.
- Architecting for HIPAA security and compliance paper – This paper describes how companies can use AWS to create HIPAA-compliant applications.
- AWS compliance resources – This collection of workbooks and guides might apply to your industry and location.
- AWS Config – This AWS service assesses how well your resource configurations comply with internal practices, industry guidelines, and regulations.
- AWS Security Hub – This AWS service provides a comprehensive view of your security state within AWS that helps you check your compliance with security industry standards and best practices.

Resilience in Amazon EKS

The AWS global infrastructure is built around AWS Regions and Availability Zones. AWS Regions provide multiple physically separated and isolated Availability Zones, which are connected with low-latency, high-throughput, and highly redundant networking. With Availability Zones, you can design and operate applications and databases that automatically fail over between Availability Zones without interruption. Availability Zones are more highly available, fault tolerant, and scalable than traditional single or multiple data center infrastructures.

Amazon EKS runs 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 it provides automated version updates and patching for them.

This control plane consists of at least two API server instances and three `etcd` instances that run across three Availability Zones within a 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 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) 1.0 or later. We recommend TLS 1.2 or later. Clients must also support cipher suites with perfect forward secrecy (PFS) such as Ephemeral Diffie-Hellman (DHE) or Elliptic Curve Ephemeral Diffie-Hellman (ECDHE). Most modern systems such as Java 7 and later support these modes.

Additionally, requests must be signed by using an access key ID and a secret access key that is associated with an IAM principal. Or you can use the AWS Security Token Service (AWS STS) to generate temporary security credentials to sign requests.

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

For more information about VPC considerations, see Cluster VPC considerations (p. 208).

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. 210).

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. 41).

You can implement network policies with tools such as Project Calico (p. 246). 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. 65).

You can update an Amazon EKS cluster (p. 24) 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. 58).

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.

Amazon EKS security best practices

This topic provides security best practices for your cluster.

Restricting access to the IMDS and Amazon EC2 instance profile credentials

By default, the Amazon EC2 instance metadata service (IMDS) provides the credentials assigned to the node IAM role (p. 341) to the instance, and any container running on the instance. When you use IAM roles for service accounts (p. 345), it updates the credential chain of the pod to use the IAM roles for service accounts token. The pod, however, can still inherit the rights of the instance profile assigned to the node. We recommended that you block pod access to IMDS to minimize the permissions available to your containers if:

- You've implemented IAM roles for service accounts and have assigned necessary permissions directly to all pods that require access to AWS services.
- No pods in your cluster require access to IMDS for other reasons, such as retrieving the current Region.

For more information, see Retrieving Security Credentials from Instance Metadata. You can prevent access to IMDS from your instance and containers using one of the following options.

Important
If you use the AWS Load Balancer Controller in your cluster, you may need to change your load balancer configuration. For more information, see To deploy the AWS Load Balancer Controller to an Amazon EKS cluster (p. 239).

- **Block access to IMDSv1 from the node and all containers and block access to IMDSv2 for all containers that don't use host networking** – Your instance and pods that have `hostNetwork: true` in their pod spec use host networking. To implement this option, complete the steps in the row and column that apply to your situation.

Deployment method	New node group	Existing node group
Managed nodes without a custom launch template	Not possible using any deployment method other than `eksctl`. If deploying with `eksctl`, use the `--disable-pod-imds` option with `eksctl create nodegroup`. `eksctl` can be used because it creates a launch template based on options you specify and deploys	We recommend creating a new node group with a custom launch template that includes the settings in the `New node group` column of the next row of this table.

Deployment method	New node group	Existing node group
	the node group using that launch template.	
Managed nodes with a custom launch template	Set the following settings in the launch template's (p. 100) **Advanced details**: • **Metadata accessible** – `Enabled` • **Metadata version** – `V2 only (token required)` • **Metadata response hop limit** – `1`	Update your launch template with the settings in the **New** column and then update your node group (p. 97) using the new launch template version.

Deployment method	New node group	Existing node group

Self-managed	• If creating the node group using the AWS Management Console
	1. Download the self-managed node group AWS CloudFormation template for your Region and operating system.
	• Linux – All Regions other than China (Beijing) and China (Ningxia)
	`https://s3.us-west-2.amazonaws.com/amazon-eks/cloudformation/2020-10-29/amazon-eks-nodegroup.yaml`
	• Linux – China (Beijing) and China (Ningxia)
	`https://s3.cn-north-1.amazonaws.com.cn/amazon-eks/cloudformation/2020-10-29/amazon-eks-nodegroup.yaml`
	• Windows – All Regions other than China (Beijing) and China (Ningxia)
	`https://s3.us-west-2.amazonaws.com/amazon-eks/cloudformation/2020-10-29/amazon-eks-windows-nodegroup.yaml`
	• Windows – China (Beijing) and China (Ningxia)
	`https://s3.cn-north-1.amazonaws.com.cn/amazon-eks/cloudformation/2020-10-29/amazon-eks-windows-nodegroup.yaml`
	2. Edit the file. Change the line that says `HttpPutResponseHopLimit : 2` to
		• If updating or migrating the node group using the AWS Management Console
		1. Download the self-managed node group AWS CloudFormation template for your Region and operating system.
		• Linux – All Regions other than China (Beijing) and China (Ningxia)
		`https://s3.us-west-2.amazonaws.com/amazon-eks/cloudformation/2020-10-29/amazon-eks-nodegroup.yaml`
		• Linux – China (Beijing) and China (Ningxia)
		`https://s3.cn-north-1.amazonaws.com.cn/amazon-eks/cloudformation/2020-10-29/amazon-eks-nodegroup.yaml`
		• Windows – All Regions other than China (Beijing) and China (Ningxia)
		`https://s3.us-west-2.amazonaws.com/amazon-eks/cloudformation/2020-10-29/amazon-eks-windows-nodegroup.yaml`
		• Windows – China (Beijing) and China (Ningxia)
		`https://s3.cn-north-1.amazonaws.com.cn/amazon-eks/cloudformation/2020-10-29/amazon-eks-windows-nodegroup.yaml`
		2. Edit the file. Change the line that says `HttpPutResponseHopLimit : 2` to

Deployment method	New node group	Existing node group
	`HttpPutResponseHopLimit : 1` and save the file. 3. Deploy the node group using the instructions in the section called “Amazon Linux” (p. 105) or the section called “Windows” (p. 112). For the instruction about specifying an AWS CloudFormation template, instead of selecting **Amazon S3 URL**, select **Upload a template file**, then select **Choose file**, choose your edited file, and then continue on with the instructions. Set the value of **DisableIMDSv1** to `true` too. • If creating the node group using `eksctl`, use the `--disable-pod-imds` option with `eksctl create nodegroup`.	`HttpPutResponseHopLimit : 1` and save the file. 3. Update your self-managed node group using the instructions in ??? (p. 122). For the instruction about specifying an AWS CloudFormation template, instead of selecting **Amazon S3 URL**, select **Upload a template file**, then select **Choose file**, choose your edited file, and then continue on with the instructions. Set the value of **DisableIMDSv1** to `true` too. **Important** Each time you update the node group, for any reason, such as a new AMI or Kubernetes version, you need to change the previous settings in the template again. • If migrating the node group using `eksctl`, then when you create the new node group, use the `--disable-pod-imds` option with `eksctl create nodegroup`. **Important** Each time you update the node group, for any reason, such as a new AMI or Kubernetes version, make sure to use this option when creating your new node group.

- **Block access to IMDSv1 and IMDSv2 for all containers that don't use host networking** – Your instance and pods that have `hostNetwork: true` in their pod spec use host networking, but for legacy reasons still require access to IMDSv1. Run the following `iptables` commands on each of your Amazon Linux nodes (as root) or include them in your instance bootstrap user data script.

```
yum install -y iptables-services
iptables --insert FORWARD 1 --in-interface eni+ --destination 169.254.169.254/32 --jump
 DROP
```

```
iptables-save | tee /etc/sysconfig/iptables
systemctl enable --now iptables
```

Important

- The previous rule applies only to network interfaces within the node that have a name that starts with `eni`, which is all network interfaces that the CNI plugin creates for pods that don't use host networking. Traffic to the IMDS is not dropped for the node, or for pods that use host networking, such as `kube-proxy` and the CNI plugin.
- If you implement network policy, using a tool such as Calico (p. 246), the previous rule may be overridden. When implementing network policy, ensure that it doesn't override this rule, or that your policy includes this rule.
- If you've applied security groups to pods and therefore, have branch network interfaces, in addition to the previous command, also run the following command.

```
iptables -t mangle -A POSTROUTING -o vlan+ --destination 169.254.169.254/32 --
jump DROP
```

For more information about branch network interfaces, see Security groups for pods (p. 219).

Pod security policy

The Kubernetes pod security policy admission controller validates pod creation and update requests against a set of rules. By default, Amazon EKS clusters ship with a fully permissive security policy with no restrictions. For more information, see Pod Security Policies in the Kubernetes documentation.

Note
The pod security policy admission controller is only enabled on Amazon EKS clusters running Kubernetes version 1.13 or later. You must update your cluster's Kubernetes version to at least 1.13 to use pod security policies. For more information, see Updating a cluster (p. 24).

Amazon EKS default pod security policy

Amazon EKS clusters with Kubernetes version 1.13 and higher have a default pod security policy named `eks.privileged`. This policy has no restriction on what kind of pod can be accepted into the system, which is equivalent to running Kubernetes with the `PodSecurityPolicy` controller disabled.

Note
This policy was created to maintain backwards compatibility with clusters that did not have the `PodSecurityPolicy` controller enabled. You can create more restrictive policies for your cluster and for individual namespaces and service accounts and then delete the default policy to enable the more restrictive policies.

You can view the default policy with the following command.

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

Output:

```
NAME             PRIV   CAPS   SELINUX    RUNASUSER   FSGROUP    SUPGROUP   READONLYROOTFS
  VOLUMES
eks.privileged   true   *      RunAsAny   RunAsAny    RunAsAny   RunAsAny   false
  *
```

For more details, you can describe the policy with the following command.

```
kubectl describe psp eks.privileged
```

Output:

```
Name:  eks.privileged

Settings:
  Allow Privileged:                       true
  Allow Privilege Escalation:             0xc0004ce5f8
  Default Add Capabilities:               <none>
  Required Drop Capabilities:             <none>
  Allowed Capabilities:                   *
  Allowed Volume Types:                   *
  Allow Host Network:                     true
  Allow Host Ports:                       0-65535
  Allow Host PID:                         true
  Allow Host IPC:                         true
  Read Only Root Filesystem:              false
  SELinux Context Strategy: RunAsAny
    User:                                 <none>
    Role:                                 <none>
    Type:                                 <none>
    Level:                                <none>
  Run As User Strategy: RunAsAny
    Ranges:                               <none>
  FSGroup Strategy: RunAsAny
    Ranges:                               <none>
  Supplemental Groups Strategy: RunAsAny
    Ranges:                               <none>
```

The following example shows the full YAML file for the `eks.privileged` pod security policy, its cluster role, and cluster role binding.

```
---
apiVersion: policy/v1beta1
kind: PodSecurityPolicy
metadata:
  name: eks.privileged
  annotations:
    kubernetes.io/description: 'privileged allows full unrestricted access to
      pod features, as if the PodSecurityPolicy controller was not enabled.'
    seccomp.security.alpha.kubernetes.io/allowedProfileNames: '*'
  labels:
    kubernetes.io/cluster-service: "true"
    eks.amazonaws.com/component: pod-security-policy
spec:
  privileged: true
  allowPrivilegeEscalation: true
  allowedCapabilities:
  - '*'
  volumes:
  - '*'
  hostNetwork: true
  hostPorts:
  - min: 0
    max: 65535
  hostIPC: true
  hostPID: true
  runAsUser:
    rule: 'RunAsAny'
  seLinux:
    rule: 'RunAsAny'
  supplementalGroups:
    rule: 'RunAsAny'
  fsGroup:
    rule: 'RunAsAny'
  readOnlyRootFilesystem: false

---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: eks:podsecuritypolicy:privileged
  labels:
    kubernetes.io/cluster-service: "true"
    eks.amazonaws.com/component: pod-security-policy
rules:
- apiGroups:
  - policy
  resourceNames:
  - eks.privileged
  resources:
  - podsecuritypolicies
  verbs:
  - use

---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: eks:podsecuritypolicy:authenticated
  annotations:
    kubernetes.io/description: 'Allow all authenticated users to create privileged pods.'
  labels:
    kubernetes.io/cluster-service: "true"
    eks.amazonaws.com/component: pod-security-policy
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: eks:podsecuritypolicy:privileged
subjects:
  - kind: Group
    apiGroup: rbac.authorization.k8s.io
    name: system:authenticated
```

To delete the default pod security policy

After you create custom pod security policies for your cluster, you can delete the default Amazon EKS `eks.privileged` pod security policy to enable your custom policies.

1. Create a file called `privileged-podsecuritypolicy.yaml` and paste the full `eks.privileged` YAML file contents from the preceding example into it (this allows you to delete the pod security policy, the `ClusterRole`, and the `ClusterRoleBinding` associated with it).
2. Delete the YAML with the following command.

   ```
   kubectl delete -f privileged-podsecuritypolicy.yaml
   ```

To install or restore the default pod security policy

If you are upgrading from an earlier version of Kubernetes, or have modified or deleted the default Amazon EKS `eks.privileged` pod security policy, you can restore it with the following steps.

1. Create a file called `privileged-podsecuritypolicy.yaml` and paste the YAML file contents below into it.

```
---
apiVersion: policy/v1beta1
kind: PodSecurityPolicy
metadata:
  name: eks.privileged
  annotations:
    kubernetes.io/description: 'privileged allows full unrestricted access to
      pod features, as if the PodSecurityPolicy controller was not enabled.'
    seccomp.security.alpha.kubernetes.io/allowedProfileNames: '*'
  labels:
    kubernetes.io/cluster-service: "true"
    eks.amazonaws.com/component: pod-security-policy
spec:
  privileged: true
  allowPrivilegeEscalation: true
  allowedCapabilities:
  - '*'
  volumes:
  - '*'
  hostNetwork: true
  hostPorts:
  - min: 0
    max: 65535
  hostIPC: true
  hostPID: true
  runAsUser:
    rule: 'RunAsAny'
  seLinux:
    rule: 'RunAsAny'
  supplementalGroups:
    rule: 'RunAsAny'
  fsGroup:
    rule: 'RunAsAny'
  readOnlyRootFilesystem: false

---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: eks:podsecuritypolicy:privileged
  labels:
    kubernetes.io/cluster-service: "true"
    eks.amazonaws.com/component: pod-security-policy
rules:
- apiGroups:
  - policy
  resourceNames:
  - eks.privileged
  resources:
  - podsecuritypolicies
  verbs:
  - use

---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: eks:podsecuritypolicy:authenticated
  annotations:
    kubernetes.io/description: 'Allow all authenticated users to create privileged
 pods.'
  labels:
    kubernetes.io/cluster-service: "true"
    eks.amazonaws.com/component: pod-security-policy
```

```
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: eks:podsecuritypolicy:privileged
subjects:
  - kind: Group
    apiGroup: rbac.authorization.k8s.io
    name: system:authenticated
```

2. Apply the YAML with the following command.

```
kubectl apply -f privileged-podsecuritypolicy.yaml
```

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.

Contents

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

Logging Amazon EKS API calls with AWS CloudTrail

Amazon EKS is integrated with AWS CloudTrail, a service that provides a record of actions taken by a user, role, or an AWS service in Amazon EKS. CloudTrail captures all API calls for Amazon EKS as events, including 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, including events for Amazon EKS. If you don't configure a trail, you can still view the most recent events in the CloudTrail console in **Event history**. Using the information collected by CloudTrail, you can determine the request that was made to Amazon EKS, the IP address from which the request was made, who made the request, when it was made, and additional details.

To learn more about CloudTrail, see the AWS CloudTrail User Guide.

Amazon EKS information in CloudTrail

CloudTrail is enabled on your AWS account when you create the account. When activity occurs in Amazon EKS, that activity is recorded in a CloudTrail event along with other AWS service events in **Event history**. You can view, search, and download recent events in your AWS account. For more information, see Viewing events with CloudTrail event history.

For an ongoing record of events in your AWS account, including events for Amazon EKS, create a trail. A *trail* enables CloudTrail to deliver log files to an Amazon S3 bucket. By default, when you create a trail in the console, the trail applies to all AWS Regions. The trail logs events from all Regions in the AWS partition and delivers the log files to the Amazon S3 bucket that you specify. Additionally, you can configure other AWS services to further analyze and act upon the event data collected in CloudTrail logs. For more information, see the following:

- Overview for creating a trail
- CloudTrail supported services and integrations
- Configuring Amazon SNS notifications for CloudTrail
- Receiving CloudTrail log files from multiple regions and Receiving CloudTrail log files from multiple accounts

All Amazon EKS actions are logged by CloudTrail and are documented in the Amazon EKS API Reference. For example, calls to the `CreateCluster`, `ListClusters` and `DeleteCluster` sections generate entries in the CloudTrail log files.

Every event or log entry contains information about who generated the request. The identity information helps you determine the following:

- Whether the request was made with root or AWS Identity and Access Management (IAM) user credentials.
- Whether the request was made with temporary security credentials for a role or federated user.
- Whether the request was made by another AWS service.

For more information, see the CloudTrail userIdentity element.

Understanding Amazon EKS log file entries

A trail is a configuration that enables delivery of events as log files to an Amazon S3 bucket that you specify. CloudTrail log files contain one or more log entries. An event represents a single request from any source and includes information about the requested action, the date and time of the action, request parameters, and so on. CloudTrail log files aren't an ordered stack trace of the public API calls, so they don't appear in any specific order.

The following example shows a CloudTrail log entry that demonstrates the `CreateCluster` action.

```
{
  "eventVersion": "1.05",
  "userIdentity": {
    "type": "IAMUser",
    "principalId": "AKIAIOSFODNN7EXAMPLE",
    "arn": "arn:aws:iam::<your-account-id>:user/username",
    "accountId": "<your-account-id>",
    "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::<your-account-id>:role/AWSServiceRoleForAmazonEKS-
CAC1G1VH3ZKZ",
    "clusterName": "test"
  },
  "responseElements": {
    "cluster": {
      "clusterName": "test",
      "status": "CREATING",
      "createdAt": 1527535003.208,
      "certificateAuthority": {},
      "arn": "arn:aws:eks:<region-code>:<your-account-id>:cluster/test",
      "roleArn": "arn:aws:iam::<your-account-id>: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": "<your-account-id>"
}
```

Log Entries for Amazon EKS Service Linked Roles

The Amazon EKS service linked roles make API calls to AWS resources. You will see CloudTrail log entries with `username: AWSServiceRoleForAmazonEKS` and `username: AWSServiceRoleForAmazonEKSNodegroup` for calls made by the Amazon EKS service linked roles. For more information about Amazon EKS and service linked roles, see Using Service-Linked Roles for Amazon EKS (p. 335).

The following example shows a CloudTrail log entry that demonstrates a `DeleteInstanceProfile` action made by the `AWSServiceRoleForAmazonEKSNodegroup` service linked role, noted in the `sessionContext`.

```
{
    "eventVersion": "1.05",
    "userIdentity": {
        "type": "AssumedRole",
        "principalId": "AROA3WHGPEZ7SJ2CW55C5:EKS",
        "arn": "arn:aws:sts::<your-account-id>:assumed-role/
AWSServiceRoleForAmazonEKSNodegroup/EKS",
        "accountId": "<your-account-id>",
        "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
        "sessionContext": {
            "sessionIssuer": {
                "type": "Role",
                "principalId": "AROA3WHGPEZ7SJ2CW55C5",
                "arn": "arn:aws:iam::<your-account-id>:role/aws-service-role/eks-
nodegroup.amazonaws.com/AWSServiceRoleForAmazonEKSNodegroup",
                "accountId": "<your-account-id>",
                "userName": "AWSServiceRoleForAmazonEKSNodegroup"
            },
            "webIdFederationData": {},
            "attributes": {
                "mfaAuthenticated": "false",
                "creationDate": "2020-02-26T00:56:33Z"
            }
        },
        "invokedBy": "eks-nodegroup.amazonaws.com"
    },
    "eventTime": "2020-02-26T00:56:34Z",
    "eventSource": "iam.amazonaws.com",
    "eventName": "DeleteInstanceProfile",
    "awsRegion": "us-east-1",
    "sourceIPAddress": "eks-nodegroup.amazonaws.com",
    "userAgent": "eks-nodegroup.amazonaws.com",
    "requestParameters": {
        "instanceProfileName": "eks-11111111-2222-3333-4444-abcdef123456"
    },
    "responseElements": null,
    "requestID": "11111111-2222-3333-4444-abcdef123456",
    "eventID": "11111111-2222-3333-4444-abcdef123456",
    "eventType": "AwsApiCall",
    "recipientAccountId": "<your-account-id>"
}
```

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. In AWS Outposts environments, you can use the same AWS APIs, tools, and infrastructure that you use in the AWS Cloud. Amazon EKS nodes on AWS Outposts is ideal for low-latency workloads that need to be run in close proximity to on-premises data and applications. For more information about AWS Outposts, see the AWS Outposts User Guide.

Prerequisites

The following are the prerequisites for using Amazon EKS 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 AWS Region. We recommend that you provide highly available and low-latency connectivity between your Outpost and its 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 Regions, see Amazon EKS service endpoints in the *AWS General Reference*.

Considerations

- AWS Identity and Access Management, Network Load Balancer, Classic Load Balancer, and Amazon Route 53 run in the AWS Region, not on Outposts. This increases latencies between the services and the containers.
- You can deploy self-managed nodes to AWS Outposts, but not managed or Fargate nodes. For more information, see the section called “Amazon Linux” (p. 105), the section called “Bottlerocket” (p. 110), or the section called “Windows” (p. 112).
- You can't pass Outposts subnets in when creating a cluster. For more information, see the section called “Creating a cluster” (p. 17).
- You can't use AWS Outposts in China Regions.

- If network connectivity between your Outpost and its AWS Region is lost, your nodes will continue to run. However, you cannot create new nodes or take new actions on existing deployments until connectivity is restored. In case of instance failures, the instance will not be automatically replaced. The Kubernetes control plane runs in the Region, and missing heartbeats caused by things like a loss of connectivity to the Availability Zone could lead to failures. The failed heartbeats will lead to pods on the Outposts being marked as unhealthy, and eventually the node status will time out and pods will be marked for eviction. For more information, see Node Controller in the Kubernetes documentation.

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 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, including Amazon EC2 instances, Amazon EBS volumes, and Application Load Balancers from a Local Zone as part of your cluster. There are several considerations when using Local Zone infrastructure as part of your Amazon EKS cluster.

Kubernetes versions

Only Amazon EKS clusters running Kubernetes versions 1.17 and later can use Local Zone compute resources.

Nodes

Amazon EKS does not support creating managed node groups in AWS Local Zones. You must create self-managed nodes using the Amazon EC2 API or AWS CloudFormation. **Do not use eksctl to create your cluster or nodes in Local Zones**. For details and options, follow the instructions in Self-managed nodes (p. 104).

Network architecture

The Amazon EKS managed Kubernetes control plane always runs in the AWS Region. The Amazon EKS managed Kubernetes control plane cannot run in the Local Zone. Since 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. 203).

Unlike regional subnets, Amazon EKS cannot 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. 210). You can then deploy self-managed nodes to the Local Zone subnets and the nodes will be able to 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` 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. 105).
- 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
`kubernetes.io/cluster/<cluster-name>`	`owned`

- 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 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 Managing users or IAM roles for your cluster (p. 288). 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. 295).

```
aws --region <region-code> eks update-kubeconfig --name <cluster_name> --role-arn
 arn:aws:iam::<aws_account_id>:role/<role_name>
```

To map an IAM user to a Kubernetes RBAC user, see Managing users or IAM roles for your cluster (p. 288).

`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. 299).

> **Note**
> The `aws-iam-authenticator` is not 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. 40).

If your managed node group encounters a health issue, Amazon EKS returns an error message to help you to diagnose the issue. The following error messages and their associated descriptions are shown below.

- **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. 377).
- **AmiIdNotFound**: 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 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 may be able to recreate a launch template with the same settings to recover.
- **Ec2LaunchTemplateVersionMismatch**: The Amazon EC2 launch template version for your managed node group does not match the version that Amazon EKS created. You may be able to revert to the version that Amazon EKS created to recover.
- **IamInstanceProfileNotFound**: We couldn't find the IAM instance profile for your managed node group. You may be able to recreate an instance profile with the same settings to recover.
- **IamNodeRoleNotFound**: We couldn't find the IAM role for your managed node group. You may be able to recreate an IAM role with the same settings to recover.
- **AsgInstanceLaunchFailures**: Your Auto Scaling group is experiencing failures while attempting to launch instances.
- **NodeCreationFailure**: Your launched instances are unable to register with your Amazon EKS cluster. Common causes of this failure are insufficient node IAM role (p. 341) 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. 209). Your nodes must also have ports open to the internet. For more information, see Amazon EKS security group considerations (p. 210).
- **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 `AcessDenied` 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 (which is available on nodes at `/opt/cni/bin/aws-cni-support.sh`) that you can use to collect diagnostic logs for support cases and general troubleshooting.

Use the following command to run the script on your 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-0717c9d54b6cfaa19_2020-03-24_0103-
UTC_0.6.1.tar.gz
```

The diagnostic information is collected and stored at

```
/var/log/eks_i-0717c9d54b6cfaa19_2020-03-24_0103-UTC_0.6.1.tar.gz
```

Container runtime network not ready

You may receive a `Container runtime network not ready` error and authorization errors similar to the following:

```
4191 kubelet.go:2130] Container runtime network not ready: NetworkReady=false
 reason:NetworkPluginNotReady message:docker: network plugin is not ready: cni config
 uninitialized
```

```
4191 reflector.go:205] k8s.io/kubernetes/pkg/kubelet/kubelet.go:452: Failed to list
 *v1.Service: Unauthorized
4191 kubelet_node_status.go:106] Unable to register node "ip-10-40-175-122.ec2.internal"
 with API server: Unauthorized
4191 reflector.go:205] k8s.io/kubernetes/pkg/kubelet/kubelet.go:452: Failed to list
 *v1.Service: Unauthorized
```

The errors are most likely related to the AWS IAM Authenticator configuration map not being applied to the nodes. The configuration map provides the `system:bootstrappers` and `system:nodes` Kubernetes RBAC permissions for nodes to register to the cluster. For more information, see **To enable nodes to join your cluster** on the **Self-managed nodes** tab of Launching self-managed Amazon Linux nodes (p. 105). Ensure that you specify the **Role ARN** of the instance role in the configuration map, not the **Instance Profile ARN**.

The authenticator does not recognize a **Role ARN** if it includes a path other than `/`, such as the following example:

```
arn:aws:iam::<111122223333>:role/<development/apps/prod-iam-role-
NodeInstanceRole-621LVEXAMPLE>
```

When specifying a **Role ARN** in the configuration map that includes a path other than `/`, you must drop the path. The ARN above would be specified as the following:

```
arn:aws:iam::<111122223333>:role/<prod-iam-role-NodeInstanceRole-621LVEXAMPLE>
```

TLS handshake timeout

When a node is unable to establish a connection to the public API server endpoint, you may an error similar to the following error.

```
server.go:233] failed to run Kubelet: could not init cloud provider "aws": error finding
 instance i-1111f2222f333e44c: "error listing AWS instances: \"RequestError: send request
 failed\\ncaused by: Post  net/http: TLS handshake timeout\""
```

The `kubelet` process will continually respawn and test the API server endpoint. The error can also occur temporarily during any procedure that performs a rolling update of the cluster in the control plane, such as a configuration change or version update.

To resolve the issue, check the route table and security groups to ensure that traffic from the 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 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>"
```

Troubleshooting IAM

This topic covers some common errors that you may see while using Amazon EKS with IAM and how to work around them.

AccessDeniedException

If you receive an `AccessDeniedException` when calling an AWS API operation, then the AWS Identity and Access Management (IAM) user or role credentials that you are using do not have the required permissions to make that call.

```
An error occurred (AccessDeniedException) when calling the DescribeCluster operation:
User: arn:aws:iam::<111122223333>:user/<user_name> is not authorized to perform:
eks:DescribeCluster on resource: arn:aws:eks<:region>:<111122223333>:cluster/<cluster_name>
```

In the above example message, the user does not have permissions to call the Amazon EKS `DescribeCluster` API operation. To provide Amazon EKS admin permissions to a user, see Amazon EKS identity-based policy examples (p. 331).

For more general information about IAM, see Controlling access using policies in the *IAM User Guide*.

Can't see workloads or nodes and 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 entity (user or role) that you're signed into the AWS Management Console with meets all of the following requirements:

- Has an IAM policy attached to it that includes the `eks:AccessKubernetesApi` action. For an example IAM policy, see View nodes and workloads for all clusters in the AWS Management Console (p. 333). If you're using the IAM policy visual editor in the AWS Management Console and you don't see the `eks:AccessKubernetesApi` **Permission** listed, edit the policy's JSON and add `eks:AccessKubernetesApi` to the list of `Actions` in the JSON.
- Has a mapping to a Kubernetes user or group in the `aws-auth` configmap. For more information about adding IAM users or roles to the `aws-auth` configmap, see Managing users or IAM roles for your cluster (p. 288). If the user or role isn't mapped, the console error may include **Unauthorized: Verify you have access to the Kubernetes cluster**.
- The Kubernetes user or group that the IAM account or role is mapped to in the configmap must be a `subject` in a `rolebinding` or `clusterrolebinding` that is bound to a Kubernetes `role`

or `clusterrole` that has the necessary permissions to view the Kubernetes resources. If the user or group doesn't have the necessary permissions, the console error may include **Unauthorized: Verify you have access to the Kubernetes cluster**. To create roles and bindings, see Using RBAC Authorization in the Kubernetes documentation. You can download the example manifests that create a `clusterrole` and `clusterrolebinding` or a `role` and `rolebinding` by following the instructions in the **Important** section of View nodes and workloads for all clusters in the AWS Management Console.

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 configuration map. Therefore, before you specify `rolearn`, remove the path. For example, change `arn:aws:iam::<123456789012>:role/<team>/<developers>/<eks-admin>` to `arn:aws:iam::<123456789012>: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. 328).
- To learn how to provide access to your resources across AWS accounts that you own, see Providing access to an IAM user in another AWS account that you own in the *IAM User Guide*.
- To learn how to provide access to your resources to third-party AWS accounts, see Providing access to AWS accounts owned by third parties in the *IAM User Guide*.
- To learn how to provide access through identity federation, see Providing access to externally authenticated users (identity federation) in the *IAM User Guide*.
- To learn the difference between using roles and resource-based policies for cross-account access, see How IAM roles differ from resource-based policies in the *IAM User Guide*.

Amazon EKS Distro

Amazon EKS Distro is a distribution of the same open source Kubernetes software and dependencies deployed by Amazon EKS in the cloud. With Amazon EKS Distro, you can create reliable and secure clusters wherever your applications are deployed. You can rely on the same versions of Kubernetes deployed by Amazon EKS, etcd, CoreDNS, upstream CNI, and CSI sidecars with the latest updates and extended security patching support. Amazon EKS Distro follows the same Kubernetes version release cycle as Amazon EKS and is provided as an open-source project.

Note
The source code for the Amazon EKS Distro is available on GitHub. The latest documentation is available on the Amazon EKS Distro website. If you find any issues, you can report them with Amazon EKS Distro by connecting with us on GitHub. There you can open issues, provide feedback, and report bugs.

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: https://eksctl.io/
- Project documentation: https://eksctl.io/
- 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: https://aws.github.io/aws-controllers-k8s/
- 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: https://fluxcd.io/
- Project documentation: https://docs.fluxcd.io/

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: https://cdk8s.io/
- Project documentation: https://github.com/cdk8s-team/cdk8s/blob/master/docs/getting-started.md

- 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. Using this CNI plugin allows Kubernetes pods to have the same IP address inside the pod as they do on the VPC network.

- Project URL: https://github.com/aws/amazon-vpc-cni-k8s
- Project documentation: https://github.com/aws/amazon-vpc-cni-k8s/blob/master/README.md

AWS 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: https://github.com/kubernetes-sigs/aws-load-balancer-controller
- Project documentation: https://kubernetes-sigs.github.io/aws-load-balancer-controller/latest/

ExternalDNS

ExternalDNS synchronizes exposed Kubernetes services and ingresses with DNS providers including Amazon Route 53 and AWS Service Discovery.

- Project URL: https://github.com/kubernetes-incubator/external-dns
- Project documentation: https://github.com/kubernetes-incubator/external-dns/blob/master/docs/tutorials/aws.md

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: https://github.com/aws/aws-app-mesh-controller-for-k8s
- Project documentation: https://docs.aws.amazon.com/app-mesh/latest/userguide/mesh-k8s-integration.html
- 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. 299).

- Project URL: https://github.com/kubernetes-sigs/aws-iam-authenticator
- Project documentation: https://github.com/kubernetes-sigs/aws-iam-authenticator/blob/master/README.md
- AWS open source blog: Deploying the AWS IAM authenticator to kops

Machine learning

Related machine learning projects for Amazon EKS and Kubernetes clusters.

Kubeflow

A machine learning toolkit for Kubernetes.

- Project URL: https://www.kubeflow.org/
- Project documentation: https://www.kubeflow.org/docs/
- AWS open source blog: Kubeflow on Amazon EKS

Auto Scaling

Related auto scaling projects for Amazon EKS and Kubernetes clusters.

Cluster autoscaler

Cluster Autoscaler is a tool that automatically adjusts the size of the Kubernetes cluster based on CPU and memory pressure.

- Project URL: https://github.com/kubernetes/autoscaler/tree/master/cluster-autoscaler
- Project documentation: https://github.com/kubernetes/autoscaler/blob/master/cluster-autoscaler/cloudprovider/aws/README.md
- Amazon EKS workshop: https://eksworkshop.com/scaling/deploy_ca/

Escalator

Escalator is a batch or job optimized horizontal autoscaler for Kubernetes.

- Project URL: https://github.com/atlassian/escalator
- Project documentation: https://github.com/atlassian/escalator/blob/master/docs/README.md

Monitoring

Related monitoring projects for Amazon EKS and Kubernetes clusters.

Prometheus

Prometheus is an open-source systems monitoring and alerting toolkit.

- Project URL: https://prometheus.io/
- Project documentation: https://prometheus.io/docs/introduction/overview/
- Amazon EKS workshop: https://eksworkshop.com/intermediate/240_monitoring/

Continuous integration / continuous deployment

Related CI/CD projects for Amazon EKS and Kubernetes clusters.

Jenkins X

CI/CD solution for modern cloud applications on Amazon EKS and Kubernetes clusters.

- Project URL: https://jenkins-x.io/
- Project documentation: https://jenkins-x.io/docs/

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
Envelope encryption for existing clusters (p. 390)	Amazon EKS now supports adding envelope 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 1.16 or later cluster	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 let Amazon EKS control the launch and version of an add-on through the Amazon EKS API for clusters 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 across multiple Kubernetes Ingresses. In the past, you had to deploy a separate Application Load Balancer for each Ingress.	October 23, 2020
NLB IP target support	You can now deploy a network load balancer (NLB) with IP targets. This enables you to use an NLB to load balance network traffic to Fargate pods and directly to pods 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 will assign 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 running on many Amazon EC2 instance types.	September 9, 2020
Deploy Bottlerocket on your nodes	You can now deploy nodes 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 using 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	New platform version with security fixes and enhancements, including UDP support for services of type `LoadBalancer` 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 Region expansion (p. 390)	Amazon EKS is now available in the Africa (Cape Town) (`af-south-1`) and Europe (Milan) (`eu-south-1`) Regions.	August 6, 2020
Fargate usage metrics	AWS Fargate provides CloudWatch usage metrics which provide visibility into your accounts 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 Region expansion (p. 390)	Amazon EKS is now available in the AWS GovCloud (US-East) (`us-gov-east-1`) and AWS GovCloud (US-West) (`us-gov-west-1`) Regions.	May 13, 2020
Kubernetes 1.12 is no longer supported on Amazon EKS	Kubernetes version 1.12 is no longer supported on Amazon EKS. Please update any 1.12 clusters to version 1.13 or higher in order to avoid service interruption.	May 12, 2020
Kubernetes version 1.16	Added Kubernetes version 1.16 support for new clusters and version upgrades.	April 30, 2020
Added the AWSServiceRoleForAmazonEKS service-linked role	Added the **AWSServiceRoleForAmazonEKS** service-linked role.	April 16, 2020
Kubernetes version 1.15	Added Kubernetes version 1.15 support for new clusters and version upgrades.	March 10, 2020
Amazon EKS Region expansion (p. 390)	Amazon EKS is now available in the Beijing (`cn-north-1`) and Ningxia (`cn-northwest-1`) Regions.	February 26, 2020
Amazon FSx for Lustre CSI driver	Added topic for installing the Amazon FSx for Lustre CSI Driver on Kubernetes 1.14 Amazon EKS clusters.	December 23, 2019
Restrict network access to the public access endpoint of a cluster	Amazon EKS now enables you to restrict the CIDR ranges that can communicate to the public access endpoint of the Kubernetes API server.	December 20, 2019
Resolve the private access endpoint address for a cluster from outside of a VPC	Amazon EKS now enables you to resolve the private access endpoint of the Kubernetes API server from outside of a VPC.	December 13, 2019
(Beta) Amazon EC2 A1 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 an AWS Outpost.	December 3, 2019

AWS Fargate on Amazon EKS	Amazon EKS Kubernetes clusters now support running pods on Fargate.	December 3, 2019
Amazon EKS Region expansion (p. 390)	Amazon EKS is now available in the Canada (Central) (`ca-central-1`) Region.	November 21, 2019
Managed node groups	Amazon EKS managed node groups automate the provisioning and lifecycle management of nodes (Amazon EC2 instances) for Amazon EKS Kubernetes clusters.	November 18, 2019
Amazon EKS platform version update	New platform versions to address CVE-2019-11253.	November 6, 2019
Kubernetes 1.11 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 in order to avoid service interruption.	November 4, 2019
Amazon EKS Region expansion (p. 390)	Amazon EKS is now available in the South America (São Paulo) (`sa-east-1`) Region.	October 16, 2019
Windows support	Amazon EKS clusters running Kubernetes version 1.14 now support Windows workloads.	October 7, 2019
Autoscaling	Added a chapter to cover some of the different types of Kubernetes autoscaling that are supported on Amazon EKS clusters.	September 30, 2019
Kubernetes Dashboard update	Updated topic for installing the Kubernetes Dashboard on Amazon EKS clusters to use the beta 2.0 version.	September 28, 2019
Amazon EFS CSI driver	Added topic for installing the Amazon EFS CSI Driver on Kubernetes 1.14 Amazon EKS clusters.	September 19, 2019
Amazon EC2 Systems Manager parameter for Amazon EKS optimized AMI ID	Added topic for retrieving the Amazon EKS optimized AMI ID using an Amazon EC2 Systems Manager parameter. The parameter eliminates the need for you to look up AMI IDs.	September 18, 2019
Amazon EKS resource tagging	Manage tagging of your Amazon EKS clusters.	September 16, 2019

Amazon EBS CSI driver	Added topic for installing the Amazon EBS CSI driver on Kubernetes 1.14 Amazon EKS clusters.	September 9, 2019
New Amazon EKS optimized AMI patched for CVE-2019-9512 and CVE-2019-9514	Amazon EKS has updated the Amazon EKS optimized AMI to address CVE-2019-9512 and CVE-2019-9514.	September 6, 2019
Announcing deprecation of Kubernetes 1.11 in Amazon EKS	Amazon EKS discontinued support for Kubernetes version 1.11 on November 4, 2019.	September 4, 2019
Kubernetes version 1.14	Added Kubernetes version 1.14 support for new clusters and version upgrades.	September 3, 2019
IAM roles for service accounts	With IAM roles for service accounts on Amazon EKS clusters, you can associate an IAM role with a Kubernetes service account. With this feature, you no longer need to provide extended permissions to the node IAM role so that pods on that node can call AWS APIs.	September 3, 2019
Amazon EKS Region expansion (p. 390)	Amazon EKS is now available in the Middle East (Bahrain) (`me-south-1`) Region.	August 29, 2019
Amazon EKS platform version update	New platform versions to address CVE-2019-9512 and CVE-2019-9514.	August 28, 2019
Amazon EKS platform version update	New platform versions to address CVE-2019-11247 and CVE-2019-11249.	August 5, 2019
Amazon EKS Region expansion (p. 390)	Amazon EKS is now available in the Asia Pacific (Hong Kong) (`ap-east-1`) Region.	July 31, 2019
Kubernetes 1.10 no longer supported on Amazon EKS	Kubernetes version 1.10 is no longer supported on Amazon EKS. Please update any 1.10 clusters to version 1.11 or higher in order to avoid service interruption.	July 30, 2019
Added topic on ALB ingress controller	The AWS ALB Ingress Controller for Kubernetes is a controller that triggers the creation of an Application Load Balancer when ingress resources are created.	July 11, 2019
New Amazon EKS optimized AMI	Removing unnecessary `kubectl` binary from AMIs.	July 3, 2019

Kubernetes version 1.13	Added Kubernetes version 1.13 support for new clusters and version upgrades.	June 18, 2019
New Amazon EKS optimized AMI patched for AWS-2019-005	Amazon EKS has updated the Amazon EKS optimized AMI to address the vulnerabilities described in AWS-2019-005.	June 17, 2019
Announcing 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
Getting started with eksctl	This getting started guide helps you to install all of the required resources to get started with Amazon EKS using `eksctl`, a simple command line utility for creating and managing Kubernetes clusters on Amazon EKS.	May 10, 2019
AWS CLI get-token command (p. 390)	The `aws eks get-token` command was added to the AWS CLI so that you no longer need to install the AWS IAM Authenticator for Kubernetes to create client security tokens for cluster API server communication. Upgrade your AWS CLI installation to the latest version to take advantage of this new functionality. For more information, see Installing the AWS Command Line Interface in the *AWS Command Line Interface User Guide*.	May 10, 2019
Amazon EKS platform version update	New platform version for Kubernetes 1.12 clusters to support custom DNS names in the Kubelet certificate and improve `etcd` performance. This fixes a bug that caused node Kubelet daemons to request a new certificate every few seconds.	May 8, 2019
Prometheus tutorial	Added topic for deploying Prometheus to your Amazon EKS cluster.	April 5, 2019

Amazon EKS control plane logging	Amazon EKS control plane logging makes it easy for you to secure and run your clusters by providing audit and diagnostic logs directly from the Amazon EKS control plane to CloudWatch Logs in your account.	April 4, 2019
Kubernetes version 1.12 (p. 390)	Added Kubernetes version 1.12 support for new clusters and version upgrades.	March 28, 2019
Added App Mesh getting started guide	Added documentation for getting started with App Mesh and Kubernetes.	March 27, 2019
Amazon EKS API server endpoint private access	Added documentation for disabling public access for your Amazon EKS cluster's Kubernetes API server endpoint.	March 19, 2019
Added topic for installing the Kubernetes Metrics Server	The Kubernetes Metrics Server is an aggregator of resource usage data in your cluster.	March 18, 2019
Added list of related open source projects	These open source projects extend the functionality of Kubernetes clusters running on AWS, including clusters managed by Amazon EKS.	March 15, 2019
Added topic for installing Helm locally	The `helm` package manager for Kubernetes helps you install and manage applications on your Kubernetes cluster. This topic helps you install and run the `helm` and `tiller` binaries locally so that you can install and manage charts using the `helm` CLI on your local system.	March 11, 2019
Amazon EKS platform version update	New platform version updating Amazon EKS Kubernetes 1.11 clusters to patch level 1.11.8 to address CVE-2019-1002100.	March 8, 2019
Increased cluster limit	Amazon EKS has increased the number of clusters that you can create in a Region from 3 to 50.	February 13, 2019
Amazon EKS Region expansion (p. 390)	Amazon EKS is now available in the Europe (London) (`eu-west-2`), Europe (Paris) (`eu-west-3`), and Asia Pacific (Mumbai) (`ap-south-1`) Regions.	February 13, 2019

New Amazon EKS optimized AMI patched for ALAS-2019-1156	Amazon EKS has updated the Amazon EKS optimized AMI to address the vulnerability described in ALAS-2019-1156.	February 11, 2019
New Amazon EKS optimized AMI patched for ALAS2-2019-1141	Amazon EKS has updated the Amazon EKS optimized AMI to address the CVEs referenced in ALAS2-2019-1141.	January 9, 2019
Amazon EKS Region expansion (p. 390)	Amazon EKS is now available in the Asia Pacific (Seoul) (`ap-northeast-2`) Region.	January 9, 2019
Amazon EKS region expansion (p. 390)	Amazon EKS is now available in the following additional regions: Europe (Frankfurt) (`eu-central-1`), Asia Pacific (Tokyo) (`ap-northeast-1`), Asia Pacific (Singapore) (`ap-southeast-1`), and Asia Pacific (Sydney) (`ap-southeast-2`).	December 19, 2018
Amazon EKS cluster updates	Added documentation for Amazon EKS cluster Kubernetes version updates and node replacement.	December 12, 2018
Amazon EKS Region expansion (p. 390)	Amazon EKS is now available in the Europe (Stockholm) (`eu-north-1`) Region.	December 11, 2018
Amazon EKS platform version update	New platform version updating Kubernetes to patch level 1.10.11 to address CVE-2018-1002105.	December 4, 2018
Added version 1.0.0 support for the Application Load Balancer ingress controller	The Application Load Balancer ingress controller releases version 1.0.0 with formal support from AWS.	November 20, 2018
Added support for CNI network configuration	The Amazon VPC CNI plugin for Kubernetes version 1.2.1 now supports custom network configuration for secondary pod network interfaces.	October 16, 2018
Added support for MutatingAdmissionWebhook and ValidatingAdmissionWebhook	Amazon EKS platform version `1.10-eks.2` now supports `MutatingAdmissionWebhook` and `ValidatingAdmissionWebhook` admission controllers.	October 10, 2018
Added partner AMI information	Canonical has partnered with Amazon EKS to create node AMIs that you can use in your clusters.	October 3, 2018

Added instructions for AWS CLI update-kubeconfig command	Amazon EKS has added the `update-kubeconfig` to the AWS CLI to simplify the process of creating a `kubeconfig` file for accessing your cluster.	September 21, 2018
New Amazon EKS optimized AMIs	Amazon EKS has updated the Amazon EKS optimized AMIs (with and without GPU support) to provide various security fixes and AMI optimizations.	September 13, 2018
Amazon EKS Region expansion (p. 390)	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 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. 390)	Initial documentation for service launch	June 5, 2018